Altova UModel 2023 Enterprise Edition



Benutzer- und Referenzhandbuch

Altova UModel 2023 Enterprise Edition
Benutzer- und Referenzhandbuch

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: 2022

© 2016-2022 Altova GmbH

Inhaltsverzeichnis

1 Einführung 12

1.1 Anmerkungen zur Unterstützung ... 13
1.2 Datenbankunterstützung ... 17

2 UModel Tutorial 18

2.1 Erste Schritte ... 19
2.2 Use Cases ... 22
2.3 Klassendiagramme ... 31
2.3.1 Erstellen von abgeleiteten Klassen ... 40
2.4 Objektdiagramme ... 46
2.5 Komponentendiagramme ... 54
2.6 Deployment-Diagramme ... 60
2.7 Forward Engineering (Modell zu Code) ... 65
2.8 Reverse Engineering (Code zu Modell) ... 75

3 Grafische Benutzeroberfläche von UModel 84

3.1 Fenster "Modell-Struktur" ... 86
3.2 Fenster "Diagramm-Struktur" ... 90
3.3 Fenster "Favoriten" ... 91
3.4 Fenster "Eigenschaften" ... 92
3.5 Fenster "Stile" ... 93
3.6 Fenster "Hierarchie" ... 94
3.7 Fenster "Übersicht" ... 96
3.8 Fenster "Dokumentation" ... 97
3.9 Fenster "Ebene" ... 98
3.10 Fenster "Meldungen" ... 99
3.11 Diagrammfenster ... 101
3.12 Diagrammbereich ... 102

4 UModel Befehlszeilenschnittstelle 104

4.1 Datei: New / Load / Save-Optionen ... 109

5 Anleitung zur Modellierung von... 111

5.1 Elemente ... 113

5.1.1 Erstellen von Elementen ... 113

5.1.2 Einfügen von Elementen aus dem Modell in ein Diagramm ... 114

5.1.3 Umbenennen, Verschieben und Kopieren von Elementen ... 116

5.1.4 Löschen von Elementen ... 117

5.1.5 Konvertieren von Elementen ... 118

5.1.6 Suchen und Ersetzen von Text ... 118

5.1.7 Überprüfen, wo und ob Elemente verwendet werden ... 120

5.1.8 Einschränken von Elementen ... 121

5.1.9 Erstellen von Hyperlinks zu Elementen ... 122

5.1.10 Dokumentieren von Elementen ... 125

5.1.11 Ändern des Stils von Elementen ... 126

5.2 Diagramme ... 129

5.2.1 Erstellen von Diagrammen ... 129

5.2.2 Generieren von Diagrammen ... 130

5.2.3 Öffnen von Diagrammen ... 133

5.2.4 Löschen von Diagrammen ... 134

5.2.5 Ändern des Stils von Diagrammen ... 134

5.2.6 Ausrichten von Modellierungselementen und Anpassen der Größe ... 136

5.2.7 Hinzufügen von Ebenen zu Diagrammen ... 138

5.2.8 Typ-Autokomplettierung in Klassen ... 140

5.2.9 Vergrößern und Verkleinern von Diagrammen ... 142

5.3 Beziehungen ... 143

5.3.1 Erstellen von Beziehungen ... 143

5.3.2 Ändern des Stils von Linien und Beziehungen ... 144

5.3.3 Anzeigen von Elementbeziehungen ... 146

5.3.4 Assoziationen ... 147

5.3.5 Collection-Assoziationen ... 150

5.3.6 Enthältbeziehung ... 153
5.4 Stereotype und Eigenschaftswerte ... 155
5.4.1 Eigenschaftswerte ... 156
5.4.2 Anwenden von Stereotypen ... 157
5.4.3 Anzeigen oder Ausblenden von Eigenschaftswerten ... 159

6 Projekte und Code Engineering 162

6.1 Verwalten von UModel-Projekten ... 163
6.1.1 Erstellen, Öffnen und Speichern von Projekten ... 163
6.1.2 Öffnen von Projekten über eine URL ... 164
6.1.3 Verschieben von Projekten in ein neues Verzeichnis ... 168
6.1.4 Anwenden von UModel-Profilen ... 170
6.1.5 Aufteilen von UModel-Projekten ... 170
6.1.6 Inkludieren von Unterprojekten ... 174
6.1.7 Freigeben von Paketen und Diagrammen ... 176
6.1.8 Verbesserung der Performance ... 179
6.2 Generieren von Programmcode ... 180
6.2.1 Definieren eines Pakets als Namespace Root ... 180
6.2.2 Hinzufügen einer Code Engineering-Komponente ... 181
6.2.3 Überprüfen der Projektsyntax ... 183
6.2.4 Codegenerierungsoptionen ... 186
6.2.5 Beispiel: Generieren von C#-Code ... 187
6.2.6 Beispiel: Generieren von Java-Code ... 193
6.2.7 Beispiel: Generieren von C++-Code ... 202
6.2.8 SPL-Vorlagen ... 207
6.3 Importieren von Quellcode ... 209
6.3.1 Reverse Engineering von C++ Code ... 211
6.3.2 Optionen für den Code-Import ... 212
6.3.3 Beispiel: Importieren eines C#-Projekts ... 218
6.4 Importieren von Java-, C#- und VB.NET-Binärdateien ... 225
6.4.1 Hinzufügen von benutzerdefinierten Java Runtimes ... 226
6.4.2 Optionen für den Import von Binärdateien ... 227
6.4.3 Beispiel: Import von .NET Assemblys ... 230
6.4.4 Beispiel: Import von Java .class-Dateien ... 233

6.5 Synchronisieren von Modell und Quellcode ... 240
6.5.1 Tipps zur Synchronisierung ... 241
6.5.2 Refactoring und Synchronisierung von Code ... 243
6.5.3 Codesynchronisierungseinstellungen ... 244
6.6 UModel-Elemententsprechungen ... 247
6.6.1 C++-Entsprechungen ... 247
6.6.2 C#-Entsprechungen ... 253
6.6.3 VB.NET-Entsprechungen ... 273
6.6.4 Java-Entsprechungen ... 288
6.6.5 XML Schema-Entsprechungen ... 294
6.6.6 Datenbank-Entsprechungen ... 304
6.7 Zusammenführen von UModel-Projekten ... 307
6.7.1 3-Weg-Projektzusammenführung ... 308
6.7.2 Beispiel: Manuelle 3-Weg-Projektzusammenführung ... 310
6.8 UML-Vorlagen ... 313
6.8.1 Vorlagensignaturen ... 314
6.8.2 Vorlagenverwendung ... 315
6.8.3 Vorlagenverwendung in Operationen und Eigenschaften ... 315

7 Transformieren von UML-Modellen 317

7.1 Transformationseinstellungen ... 320
7.2 Beispiel: Transformieren von Java in C++ ... 322
7.3 Beispiel: Transformieren von C# in Java ... 329
7.4 Beispiel: Transformieren einer Access-Datenbank in SQLite ... 335

8 Generieren von UML-Dokumentation 345

8.1 Optionen zur Generierung von Dokumentation ... 349
8.2 Anpassen der Ausgabe mit StyleVision ... 354

9 UML-Diagramme 356

9.1 Verhaltensdiagramme ... 357
9.1.1 Aktivitätsdiagramm ... 357
9.1.2 Zustandsdiagramm ... 374

9.1.3 Protokoll-Zustandsautomat ... 397
9.1.4 Use Case-Diagramm ... 402
9.1.5 Kommunikationsdiagramm ... 402
9.1.6 Interaktionsübersichtsdiagramm ... 406
9.1.7 Sequenzdiagramm ... 411
9.1.8 Zeitverlaufsdiagramm ... 440
9.2 Strukturdiagramme ... 449
9.2.1 Klassendiagramm ... 449
9.2.2 Kompositionsstrukturdiagramm ... 465
9.2.3 Komponentendiagramm ... 468
9.2.4 Deployment-Diagramm ... 468
9.2.5 Objektdiagramm ... 469
9.2.6 Paketdiagramm ... 469
9.2.7 Profildiagramm ... 476
9.3 Zusätzliche Diagramme ... 490
9.3.1 XML-Schema-Diagramme ... 490
9.3.2 Business Process Modeling Notation 1.0 / 2.0 ... 509
9.3.3 SysML-Diagramme ... 536

10 UModel und Datenbanken 555

10.1 Modellieren von Datenbanken in UModel ... 557
10.1.1 Importieren von SQL-Datenbanken in UModel ... 558
10.1.2 Erstellen von Datenbanobjekten ... 565
10.1.3 Konfigurieren des Round-Trip Engineering für Datenbanken ... 570
10.1.4 Beispiel: Aktualisieren einer Datenbank anhand des Modells ... 572
10.2 Herstellen einer Verbindung zu einer Datenquelle ... 577
10.2.1 Starten des Verbindungsassistenten ... 578
10.2.2 Übersicht über Datenbanktreiber ... 580
10.2.3 ADO-Verbindung ... 583
10.2.4 ADO.NET-Verbindung ... 589
10.2.5 ODBC-Verbindung ... 596
10.2.6 JDBC-Verbindung ... 599
10.2.7 PostgreSQL-Verbindung ... 604
10.2.8 SQLite-Verbindung ... 605

10.2.9 Beispiele für Datenbankverbindungen ... 607

11 Austausch von Metadaten zwischen XMI und XML 663

12 UModel Plug-in für Visual Studio 665

12.1 Installieren des UModel Plug-in für Visual Studio ... 667
12.2 Hinzufügen von UModel-Unterstützung zu Visual Studio-Projekten ... 668
12.3 Laden/Deaktivieren von UModel-Projekten ... 672
12.4 Synchronisieren von Modell und Code ... 673

13 UModel Plug-in für Eclipse 676

13.1 Installieren des UModel Plug-in für Eclipse ... 679
13.2 Die UModel-Perspektive ... 681
13.3 Hinzufügen von UModel-Unterstützung zu Eclipse-Projekten ... 684
13.4 Importieren vorhandener UModel-Projekte ... 686
13.5 Laden/Deaktivieren von UModel-Projekten ... 688
13.6 Funktionsweise der automatischen Synchronisierung ... 689
13.7 Beispiel: Einrichten der automatischen Synchronisierung ... 690

14 Versionskontrolle 702

14.1 Einrichten der Versionskontrolle ... 704
14.2 Unterstützte Versionskontrollsysteme ... 705
14.3 Versionskontrollbefehle ... 707
14.3.1 Aus Versionskontrolle öffnen ... 707
14.3.2 Versionskontrolle aktivieren ... 710
14.3.3 Aktuellste Version holen ... 711
14.3.4 Abrufen ... 711
14.3.5 Ordner abrufen ... 712
14.3.6 Auschecken ... 714
14.3.7 Einchecken ... 715
14.3.8 Auschecken rückgängig ... 716
14.3.9 Zu Versionskontrolle hinzufügen ... 717

14.3.10 Von Versionskontrolle ausgliedern ... 720
14.3.11 Aus Versionskontrolle freigeben ... 721
14.3.12 Verlauf anzeigen ... 722
14.3.13 Unterschiede anzeigen ... 724
14.3.14 Eigenschaften anzeigen ... 725
14.3.15 Status aktualisieren ... 726
14.3.16 Versionskontrollmanager ... 726
14.3.17 Versionskontrolle wechseln ... 726
14.4 Versionskontrolle mit Git ... 728
14.4.1 Aktivieren der Git-Versionskontrolle mit dem Git SCC Plug-in ... 729
14.4.2 Hinzufügen eines Projekts zur Git-Versionskontrolle ... 729
14.4.3 Klonen eines Projekts anhand der Git-Versionskontrolle ... 731

15 UModel Diagrammsymbole 733

15.1 Aktivitätsdiagramm ... 734
15.2 Klassendiagramm ... 735
15.3 Kommunikationsdiagramm ... 736
15.4 Kompositionsstrukturdiagramm ... 737
15.5 Komponentendiagramm ... 738
15.6 Deployment-Diagramm ... 739
15.7 Interaktionsübersichtsdiagramm ... 740
15.8 Objektdiagramm ... 741
15.9 Paketdiagramm ... 742
15.10 Profildiagramm ... 743
15.11 Protokoll-Zustandsdiagramm ... 744
15.12 Sequenzdiagramm ... 745
15.13 Zustandsdiagramm ... 746
15.14 Zeitverlaufsdiagramm ... 747
15.15 Use Case-Diagramm ... 748
15.16 XML-Schema-Diagramm ... 749
15.17 Business Process Modeling Notation ... 750
15.18 Business Process Modeling Notation 2.0 ... 752
15.19 Datenbankmodellierung ... 753

16 Menüreferenz 754

16.1 Datei 755
16.2 Bearbeiten 757
16.3 Projekt 759
16.4 Layout 762
16.5 Ansicht 764
16.6 Extras 765
16.6.1 Rechtschreibung 765
16.6.2 Rechtschreiboptionen 769
16.6.3 Skript-Editor 771
16.6.4 Makros 771
16.6.5 Benutzerdefinierte Tools 771
16.6.6 Anpassen 771
16.6.7 Symbolleisten und Fenster wiederherstellen 781
16.6.8 Optionen 781
16.7 Fenster 793
16.8 Hilfe 795

17 UModel Referenz für Programmierer 800

17.1 Anmerkungen zur Release 802
17.2 Skript-Editor 809
17.2.1 Erstellen eines Skripting-Projekts 810
17.2.2 Vordefinierte Befehle 824
17.2.3 Aktivieren von Skripts und Makros 834
17.3 UModel IDE PlugIns 837
17.3.1 Erstellen eines UModel IDE Plug-in in 837
17.3.2 Bereitstellung von UModel-Plug-ins 847
17.3.3 XML-Konfigurationsdatei 849
17.3.4 Plug-ins als ActiveX Controls 852
17.3.5 IUModelPlugIn-Schnittstelle 853
17.4 Die UModel API 857
17.4.1 Aufruf der API 857

17.4.2 Objektmodell ... 858
17.4.3 Anleitungen ... 864
17.4.4 C# API-Beispiele ... 876
17.4.5 Java-Beispielprojekt ... 903
17.4.6 JScript-Beispiele ... 904
17.5 UModel API-Referenz ... 919
17.5.1 UModel Plug-Ins ... 919
17.5.2 UModel API-Schnittstellen ... 921
17.5.3 UMLData-Schnittstellen ... 1010

18 SPL-Referenz 1379

18.1 Grundlegende SPL-Struktur ... 1380
18.2 Variablen ... 1381
18.3 Operatoren ... 1390
18.4 Bedingungen ... 1391
18.5 Collections und foreach ... 1392
18.6 Subroutinen ... 1394
18.6.1 Deklaration einer Subroutine ... 1394
18.6.2 Subroutinenaufruf ... 1395

19 Lizenzinformationen 1396

19.1 Electronic Software Distribution ... 1397
19.2 Software-Aktivierung und Lizenzüberwachung ... 1398
19.3 Altova Endbenutzer-Lizenzvereinbarung ... 1400

Index 1401

1 Einführung

Altova Web site: UML-Tool

Altova UModel 2023 Enterprise Edition ist eine UML-Modellierapplikation mit visueller Benutzeroberfläche und zahlreichen, benutzerfreundlichen Funktionen, mit denen das Erlernen von UML kein Problem mehr ist. UModel bietet viele hochspezifische Funktionen für die Implementierung der nützlichsten Aspekte der UML 2.5-Spezifikation. UModel ist eine 32/64-Bit Windows-Applikation, die auf Windows 7 SP1 mit Plattformupdate, Windows 8, Windows 10, Windows 11 und Windows Server 2008 R2 SP1 mit Plattformupdate oder höher läuft. Die 64-Bit-Version steht für die Enterprise und Professional Version zur Verfügung. Eine Übersicht über die UModel-Funktionen findne Sie unter Anmerkungen zur Unterstützung 13.



UML®, OMG™, Object Management Group™ und Unified Modeling Language™ sind entweder eingetragene Markenzeichen oder Markenzeichen der Object Management Group, Inc. in den USA und/oder anderen Ländern.

Letzte Aktualisierung: 06.10.2022

© 2016-2022 Altova GmbH

1.1 Anmerkungen zur Unterstützung

UModel ist eine 32/64-Bit-Windows-Applikation, die auf den folgenden Betriebssystemen läuft:

- Windows Server 2008 R2 SP1 mit Plattformupdate oder höher
- Windows 7 SP1 mit Plattformupdate, Windows 8, Windows 10, Windows 11

64-Bit-Unterstützung steht für die Enterprise und die Professional Edition zur Verfügung.

UML-Diagramme

UModel unterstützt alle vierzehn Diagramme der UML 2.5-Spezifikation sowie zusätzliche spezielle Diagrammarten.

Strukturdiagramme	Verhaltensdiagramme	Zusätzliche Diagrammarten
Klassendiagramme	Aktivitätsdiagramm	XML-Schema-Diagramme
Komponentendiagramm	Kommunikationsdiagramm	BPMN (Business Process Modeling Notation) 1.0 / 2.0-Diagramme (*UModel Enterprise und Professional Edition*)
Kompositionsstrukturdiagramm	Interaktionsübersichtsdiagramm	SysML 1.2-, 1.3-, 1.4-, 1.5-, 1.6-Diagramme (*UModel Enterprise und Professional Edition*)
Deployment-Diagramm	Sequenzdiagramm	Datenbankdiagramme (*UModel Enterprise und Professional Edition*)
Objektdiagramm	Zustandsdiagramme (Zustandsdiagramm und Protokoll-Zustandsdiagramm)	
Paketdiagramm	Zeitverlaufsdiagramm	
Profildiagramm	Use Case-Diagramm	

UModel wurde so konzipiert, dass der Benutzer bei der Modellierung vollkommen freie Hand hat:

- UModel-Diagramme können jederzeit und in jeder Reihenfolge erstellt werden; es muss bei der Modellierung keine vorgeschriebene Reihenfolge eingehalten werden.
- Die Syntaxfarbe in Diagrammen kann angepasst werden. So können Sie etwa Modellierungselemente und deren Eigenschaften (Schriftart, Farbe, Umrandung, usw.) auf hierarchische Weise auf Projekt-, Node/Zeilen-, Elementfamilien- und Elementebene anpassen, siehe Ändern des Stils von Elementen 126.
- Die Funktion zum unbegrenzten Rückgängigmachen und Wiederherstellen hält nicht nur Änderungen am Inhalt fest, sondern auch alle Änderungen am Stil von Modellelementen.
- Modellierungselemente unterstützen Hyperlinks, siehe Erstellen von Hyperlinks zu Elementen 122.
- Sie können im selben UML-Diagramm mehrere Ebenen erstellen, siehe Hinzufügen von Ebenen zu Diagrammen 138.

Code Engineering und Import von Binärdateien

UModel unterstützt die Codegenerierung und das Reverse Engineering von in den folgenden Sprachen geschriebenem Programmcode:

Sprache	Code Engineering	Import von Binärdateien
C#	1.2, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 7.1, 7.2, 7.3, 8.0, 9.0[1], 10	Dieselben Sprachversionen wie beim Code Engineering[2]
C++ (*UModel Enterprise Edition*)	C++98, C++11 und C++14, C++17, C++20 Nur teilweise Unterstützung für C++20: Module werden nicht unterstützt.	Nicht anwendbar
Java	1.4, 5.0 (1.5), 6 (1.6), 7 (1.7), 8 (1.8), 9 (1.9), 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	Dieselben Sprachversionen wie beim Code Engineering[3]
Visual Basic .NET	7.1 oder neuer	Dieselben Sprachversionen wie beim Code Engineering
XML-Schemas[4]	1.0	Nicht anwendbar
Datenbanken[5] (*UModel Enterprise und Professional Edition*)	Nähere Informationen zu unterstützten Datenbanken finden Sie unter Datenbankunterstützung 17.	Nicht anwendbar

Fußnoten zur Tabelle:

1. Wenn Sie anhand von C# 9.0-Code kompilierte Binärdateien importieren, werden alle *Datensätze* als *Klassen* importiert. Dies liegt daran, dass Datensätze in der Assembly als Klassen gekennzeichnet sind, wodurch sie von Klassen nicht unterschieden werden können.
2. Beim C# Code Engineering und Import von Binärdateien werden NET Framework, .NET Core, .NET 5 und .NET 6 unterstützt. Beachten Sie, dass, je nach Bedarf, .NET Framework, .NET Core, .NET 5 oder .NET 6 installiert sein muss. Auch Binärdateien anderer hier nicht erwähnter .NET-Implementierungen werden wahrscheinlich ebenfalls importiert. Siehe auch Importieren von Java-, C#- und VB.NET-Binärdateien 225.
3. Auch Binärdateien für andere Java Virtual Machines als Oracle JDK wie OpenJDK, SapMachine, Liberica JDK und andere können importiert werden, siehe Hinzufügen von benutzerdefinierten Java Runtimes 226.
4. Im Fall von XML-Schemas bedeutet Code Engineering, dass Sie ein Schema (oder mehrere Schemas aus einem Verzeichnis) in UModel importieren, das Modell anzeigen oder ändern und die Änderungen wieder in die Schema-Datei schreiben können. Wenn Sie Daten aus dem Modell in einer Schema-Datei synchronisieren, wird die Schema-Datei immer durch das Modell überschrieben. Siehe auch XML-Schema-Diagramme 490.
5. Im Fall von Datenbanken bedeutet Code Engineering, dass Sie (i) eine Datenbank in UModel modellieren können mit der Option, die Datenbank mit Hilfe eines anhand des Modells generierten Skripts zu aktualisieren oder (ii) eine bestehende Datenbankstruktur in ein Modell importieren, Änderungen daran vornehmen und anschließend ein anhand des Modells generiertes Skript an der

© 2016-2022 Altova GmbH

Datenbank anwenden können. Einige Datenbankobjekttypen werden beim Modellieren nicht unterstützt, siehe UModel und Datenbanken[555].

Allgemeine Anmerkungen:

- Die Synchronisierung zwischen Code und Modell kann auf Ebene eines Projekts, Pakets oder sogar Klassen erfolgen. Bei UModel müssen kein Pseudocode oder Kommentare im generierten Code vorhanden sein, damit ein Round-Trip Engineering durchgeführt werden kann.
- Ein einziges Projekt kann gleichzeitig sowohl Java-, C#-, C++ oder VB.NET-Code unterstützen.
- UModel unterstützt die Verwendung von UML-Vorlagen und deren Mapping von oder auf generische Java-, C#- und Visual Basic-Typen.
- UModel bietet Unterstützung für Model Driven Architecture (MDA), sodass Sie die Programmiersprache Ihrer Modelle wechseln können (z.B. von Java zu C# oder umgekehrt), siehe Transformieren von UML-Modellen[317].
- Beim Import von Quellcode können optional Klassen-[462] und Paketdiagramme[473] generiert werden. Nachdem der Quellcode in das Modell importiert wurde, können auch Sequenzdiagramme[426] generiert werden.
- Sie können Programmcode anhand von Sequenzdiagrammen[434] und Zustandsdiagrammen[386] generieren.
- UModel-Projekte können in mehrere Unterprojekte aufgeteilt werden, sodass mehrere Entwickler gleichzeitig an verschiedenen Teilen eines einzigen Projekts arbeiten können. Sie können die Änderungen anschließend wieder in einem gemeinsamen Modell miteinander integrieren. Außerdem können UModel-Projekte in Form einer 2- oder 3-Weg-Zusammenführung miteinander zusammengeführt werden, siehe Zusammenführen von UModel-Projekten[307].
- Die Codegenerierung in UModel basiert auf Spy Programming Language (SPL)-Vorlagen und kann angepasst werden.

Generierung von UML-Dokumentation

Sie können anhand von UModel-Projekten Dokumentation in HTML, RTF, Microsoft Word 2000 oder höher generieren. Zur Konfiguration des Detailliertheitsgrads der generierten Dokumentation, des Aussehens und anderer Einstellungen stehen verschiedene Optionen zur Verfügung. Die Generierung von Dokumentation im PDF-Format und eine genaue Anpassung der Vorlagen für die Dokumentgenerierung kann mit Hilfe von Altova StyleVision (https://www.altova.com/de/stylevision) durchgeführt werden. Nähere Informationen dazu finden Sie unter Generieren von UML-Dokumentation[345].

IDE-Integration

UModel ist optional als Plug-in für die folgenden integrierten Entwicklungsumgebungen erhältlich:

- Visual Studio 2012/2013/2015/2017/2019/2022, siehe UModel Plug-in für Visual Studio[665]
- Eclipse 2022-09, 2022-06, 2022-03, 2021-12, siehe UModel Plug-in für Eclipse[676]

UModel bietet eine COM-basierte API[857] und ermöglicht auch die Integration von benutzerdefinierten IDE Plug-Ins[837] (DLL-Bibliotheken) in seine grafische Benutzeroberfläche. Im Skript-Editor[809] können zur Automatisierung verschiedener Aufgaben benutzerdefinierte VBScript- oder JScript-Skripts und Makros entwickelt werden.

Microsoft Office-Integration

Dank seiner Unterstützung für die Datenbankmodellierung können in UModel Access-Datenbanken in ein Modell importiert und SQL-Skripts für Access-Datenbanken generiert werden, siehe UModel und Datenbanken[555].

Interoperabilität

Außerdem bietet UModel Unterstützung für den Import oder Export von Projekten von oder in das XML Metadata Interchange (XMI)-Format, siehe Austausch von Metadaten zwischen XMI und XML 663.

© 2016-2022 Altova GmbH

1.2 Datenbankunterstützung

Die nachstehende Tabelle enthält eine Liste aller unterstützten Datenbanken. Altova ist bestrebt, auch Unterstützung für andere Datenbanken zu gewährleisten, doch wurde die erfolgreiche Herstellung der Verbindung und Verarbeitung der Daten nur für die aufgelisteten Datenbanken getestet. Wenn es sich bei Ihrer Altova-Applikation um eine 64-Bit-Version handelt, stellen Sie sicher, dass Sie Zugriff auf die 64-Bit-Datenbanktreiber haben, die für die entsprechenden Datenbank benötigt werden.

Datenbank	Anmerkungen
Firebird 2.x, 3.x	
IBM DB2 8.x, 9.x, 10.x, 11.x	
IBM Db2 for i 6.x, 7.4	Logische Dateien werden unterstützt und als Ansichten angezeigt.
IBM Informix 11.70 und höher	
MariaDB 10 und höher	
Microsoft Access 2003 und höher	Zum Zeitpunkt der Verfassung dieser Dokumentation (Anfang September 2019) gibt es kein Microsoft Access Runtime für Access 2019. Sie können nur dann eine Verbindung von Altova-Produkten zu einer Access 2019-Datenbank herstellen, wenn Microsoft Access 2016 Runtime installiert ist und der Datentyp "Large Number" in der Datenbank nicht verwendet wird.
Microsoft Azure SQL-Datenbank	SQL Server 2016 Codebase
Microsoft SQL Server 2005 und höher Microsoft SQL Server unter Linux	
MySQL 5 und höher	
Oracle 9i und höher	
PostgreSQL 8 und höher	PostgreSQL-Verbindungen werden sowohl als native Verbindungen als auch als treiberbasierte Verbindungen über Schnittstellen (Treiber) wie ODBC oder JDBC unterstützt. Für native Verbindungen werden keine Treiber benötigt.
Progress OpenEdge 11.6	
SQLite 3.x	SQLite-Verbindungen werden als native, direkte Verbindungen zur SQLite-Datenbankdatei unterstützt. Es sind keine separaten Treiber erforderlich.
Sybase ASE15, 16	
Teradata 16	

2 UModel Tutorial

In diesem Tutorial wird beschrieben, wie Sie verschiedene UModel-Diagramme mit UModel erstellen, während Sie gleichzeitig mit der grafischen Benutzeroberfläche von UModel vertraut gemacht werden. Außerdem lernen Sie, wie Sie anhand eines UML-Modells Code generieren (Forward Engineering) und wie Sie vorhandenen Code in ein UML-Modell importieren (Reverse Engineering). Hinsichtlich Code Engineering erfahren Sie außerdem, wie Sie ein vollständiges Round-Trip Engineering durchführen (entweder Modell->Code->Modell oder Code->Modell->Code). Grundlegende UML-Kenntnisse werden für dieses Tutorial vorausgesetzt.

Das Tutorial ist in die unten aufgeführten Abschnitte gegliedert. In den Anfangsabschnitten des Tutorials arbeiten Sie mit einem mit UModel vorinstallierten Beispielprojekt. Wenn Sie mit UModel schnell ein neues Modellierungsprojekt von Grund auf neu erstellen möchten, überspringen Sie diese Abschnitte und gehen Sie direkt zu Forward Engineering (Modell zu Code) 65.

- Erste Schritte 19
- Use Cases 22
- Klassendiagramme 31
- Erstellen von abgeleiteten Klassen 40
- Objektdiagramme 46
- Komponentendiagramme 54
- Deployment-Diagramme 60
- Forward Engineering (Modell zu Code) 65
- Reverse Engineering (Code zu Modell) 75

In diesem Tutorial werden die folgenden Beispielprojektdateien aus dem Verzeichnis **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial** verwendet:

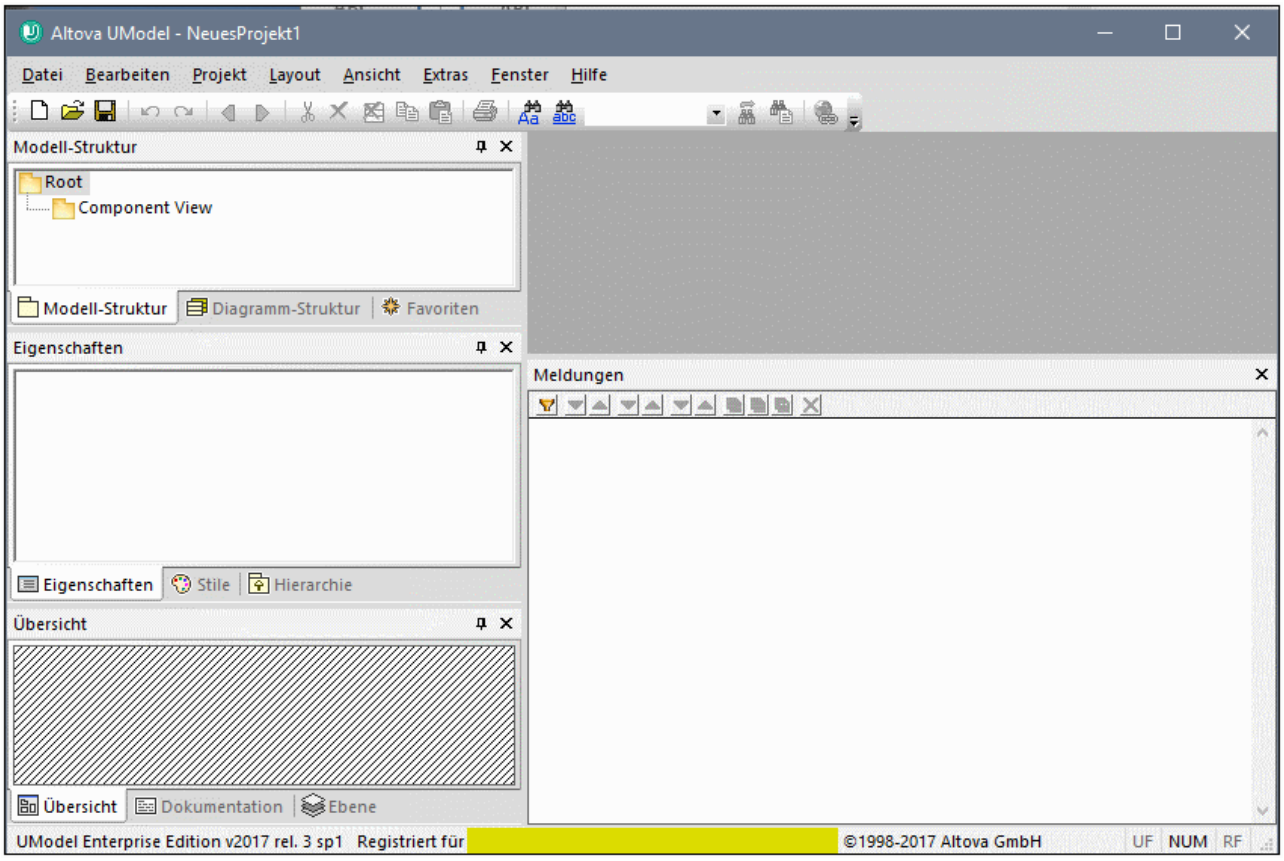
BankView-start.ump	Dies ist die UModel-Projektdatei, die den Anfangszustand des Tutorial-Beispiels darstellt. Zu diesem Zeitpunkt sind mehrere Modelldiagramme, sowie Klassen, Objekte und Modellelemente vorhanden. Während Sie das Tutorial durcharbeiten, werden Sie mit UModel neue Elemente oder Diagramme hinzufügen oder vorhandene bearbeiten. Bitte beachten Sie, dass dieses Projekt absichtlich nicht fertig gestellt wurde, daher werden Validierungsfehler und Warnungen angezeigt, wenn Sie die Projektsyntax über den Menübefehl **Projekt | Projektsyntax überprüfen** überprüfen. Im Tutorial wird erläutert, wie Sie diese Fehler beheben.
BankView-finish.ump	Dies ist die UModel-Projektdatei, die den Endzustand der Tutorial-Beispieldatei darstellt.

Anmerkung: Alle UModel-Beispieldateien stehen anfangs im Verzeichnis **C:\ProgramData\Altova\UModel2023** zur Verfügung. Wenn ein Benutzer die Applikation zum ersten Mal startet, werden die Beispieldateien in **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples** kopiert. Sie sollten die Beispieldateien im Anfangsverzeichnis daher nicht verschieben, bearbeiten oder löschen.

© 2016-2022 Altova GmbH

2.1 Erste Schritte

Wenn Sie UModel nach der Installation zum ersten Mal öffnen, wird ein neues Standardprojekt "NeuesProjekt1" geöffnet. Wenn Sie UModel später wieder starten, wird das zuletzt geladene Projekt geöffnet. Um UModel-Projekte (.ump-Dateien) zu erstellen, zu öffnen und zu speichern, verwenden Sie die Windows-Standardbefehle auf dem Menü **Datei** oder der Symbolleiste.



Grafische Benutzeroberfläche von UModel

Beachten Sie die Hauptbereiche der Benutzeroberfläche: mehrere Hilfsfenster auf der linken Seite und das leere Diagrammfenster auf der rechten Seite. Im Fenster "Modell-Struktur" werden zwei Standardpakete angezeigt: "Root" und "Component View". Diese beiden Pakete können in einem Projekt nicht gelöscht oder umbenannt werden.

Im linken oberen Bereich sehen Sie die folgenden Hilfsfenster:

- Im Fenster **Modell-Struktur** werden alle Modellierungselemente Ihres UModel-Projekts angezeigt. Elemente können in diesem Fenster mit Hilfe der Standard-Bearbeitungstasten sowie Drag and Drop direkt manipuliert werden.
- Im Fenster **Diagramm-Struktur** haben Sie schnellen Zugriff auf die Modellierungsdiagramme Ihres Projekts, egal an welcher Stelle in der Projektstruktur sich diese befinden. Diagramme werden nach Diagrammtyp gruppiert.

- Das Fenster **Favoriten** ist eine konfigurierbare Ablage für Modellierungselemente. Mit dem Kontextmenü-Befehl "Zu Favoriten hinzufügen" können Sie jede Art von Modellierungselement in dieses Fenster platzieren.

In der Mittel links sehen Sie die folgenden Hilfsfenster:

- Im Fenster **Eigenschaften** sehen Sie die Eigenschaften des aktuell ausgewählten Elements aus dem Fenster "Modell-Struktur" oder "Diagramm". In diesem Fenster können Elementeigenschaften definiert oder aktualisiert werden.
- Im Fenster **Stile** sehen Sie Attribute von Diagrammen oder Elementen, die in der Diagrammansicht angezeigt werden. Diese Stilattribute lassen sich in zwei allgemeine Gruppen unterteilen: Formatierungen und Anzeigeeinstellungen.
- Im Fenster **Hierarchie** sehen Sie alle Beziehungen des aktuell ausgewählten Modellierungselements in zwei unterschiedlichen Ansichten. Das Modellierungselement kann in einem Modellierungsdiagramm, der Modell-Struktur oder im Fenster "Favoriten" ausgewählt werden.

Im linken unteren Bereich sehen Sie die folgenden Hilfsfenster:

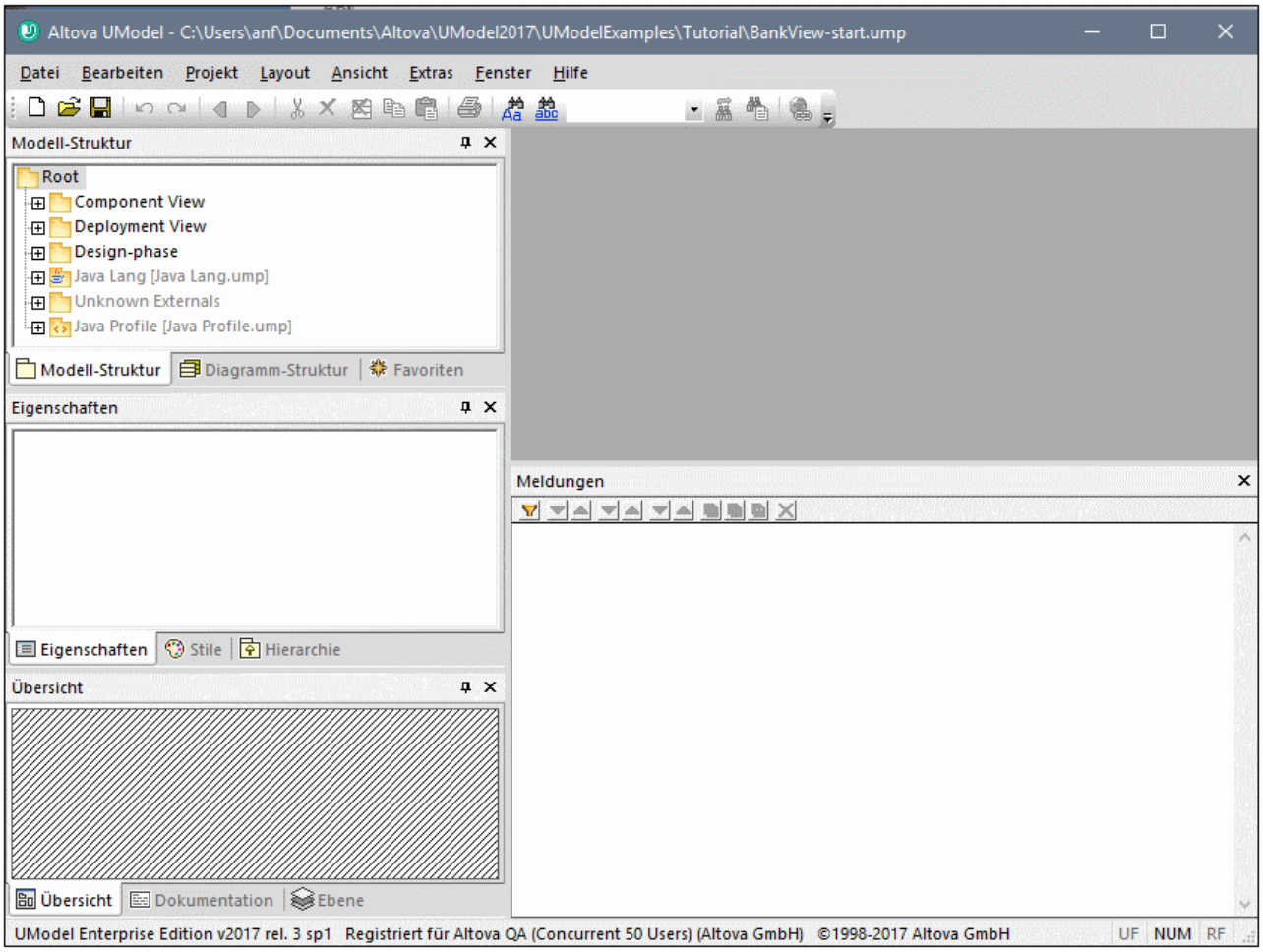
- Das Fenster **Übersicht,** in dem das aktive Diagramm im Umriss dargestellt ist.
- Das Fenster **Dokumentation**, in dem Sie Ihre Klassen nach Klassen dokumentieren können.
- Im Fenster **Ebene** können Sie mehrere Ebenen für die einzelnen UModel Diagramme definieren. Es können sowohl einzelne als auch mehrere Ebenen eingeblendet, gesperrt und ausgeblendet werden. Mit Hilfe von Ebenen können Sie die Modellierungselemente in einem Diagramm in logische Gruppen einteilen.

In diesem Tutorial arbeiten Sie hauptsächlich in den Fenstern **Modell-Struktur** und **Diagramm-Struktur** sowie im Hauptdiagrammfenster. Nähere Informationen zu den Elementen der grafischen Benutzeroberfläche finden Sie unter UModel Benutzeroberfläche 84.

So öffnen Sie das Tutorial-Projekt:

1. Wählen Sie die Menüoption **Datei | Öffnen** und navigieren Sie zum UModel-Ordner ... **UModelExamples\Tutorial**. Beachten Sie, dass Sie eine UMP-Datei auch über eine URL öffnen können. Nähere Informationen finden Sie unter Zu URL wechseln 755.
2. Öffnen Sie die Projektdatei **BankView-start.ump**. Die Projektdatei wird nun in UModel geladen. Unter dem Root-Paket sehen Sie nun einige vordefinierte Pakete. Beachten Sie, dass das Hauptfenster zu diesem Zeitpunkt leer ist.

© 2016-2022 Altova GmbH



Projekt BankView-start.ump

2.2 Use Cases

In diesem Tutorialabschnitt wird gezeigt, wie Sie ein Use Case-Diagramm erstellen. Dabei werden Sie mit den Basisfunktionalitäten der Benutzeroberfläche von UModel vertraut gemacht. Es werden die folgenden Aufgaben erläutert:

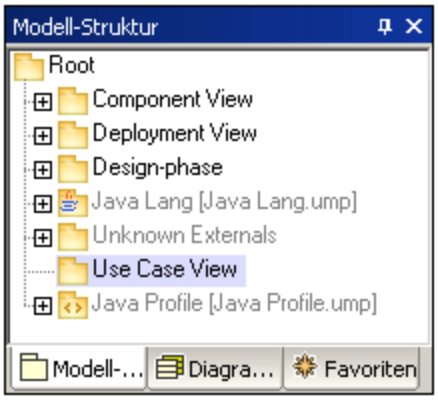
- Hinzufügen eines neuen Pakets zum Projekt
- Hinzufügen eines neuen Use Case-Diagramms zum Projekt
- Hinzufügen von Case-Elemente zum Diagramm und Definieren der Abhängigkeiten zwischen den Elementen
- Ausrichten und Anpassen der Elemente im Diagramm
- Ändern des Stil von Diagrammen in einem UModel-Projekt

Um fortzufahren, starten Sie UModel und öffnen Sie das Projekt **BankView-start.ump** (siehe auch Öffnen des Tutorial-Projekts 19).

Hinzufügen eines neues Pakets zu einem Projekt

Wie Sie bereits aus UML wissen, ist ein Paket ein Behälter für Klassen und andere UML-Elemente, sie z.B. Use Cases. Beginnen wir also, indem wir ein Paket erstellen, in dem ein neues Use Case-Diagramm gespeichert wird. Beachten Sie, dass es in UModel nicht unbedingt notwendig ist, dass sich ein bestimmtes Diagramm in einem bestimmten Paket befindet; Aus Gründen der Übersichtlichkeit und Einheitlichkeit empfiehlt es sich jedoch, Diagramme in Paketen zu organisieren.

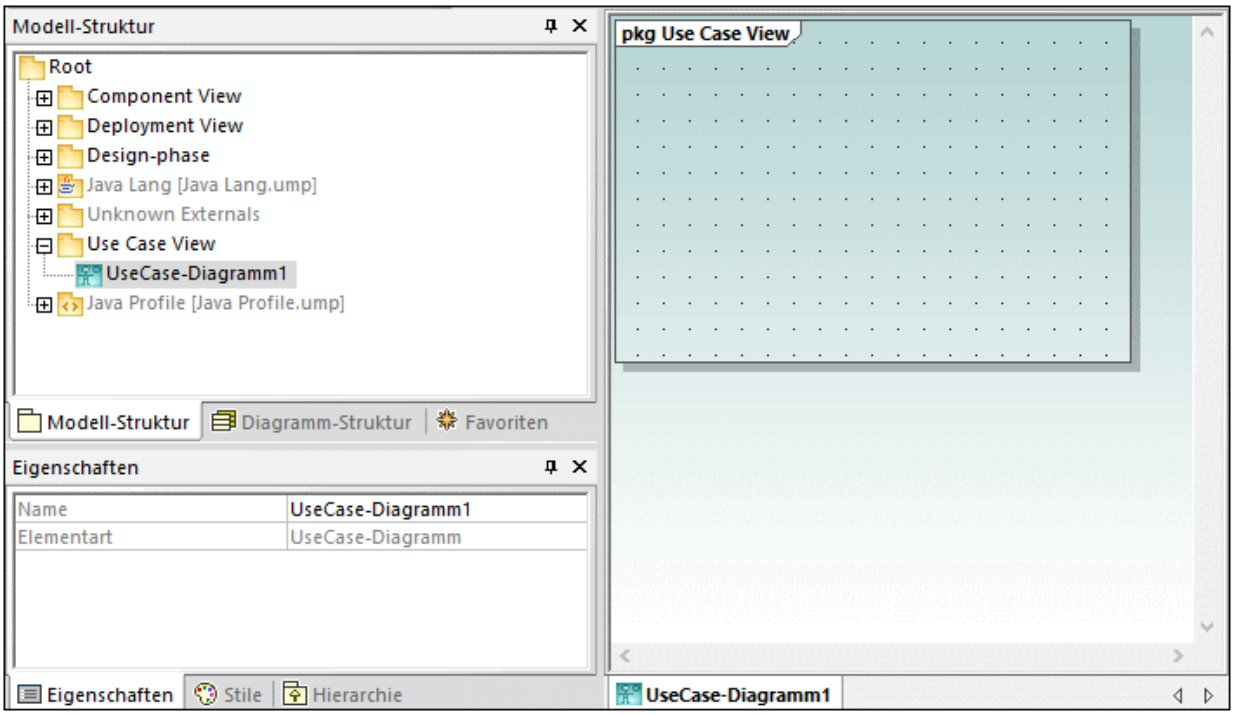
1. Rechtsklicken Sie im Fenster "Modell-Struktur" auf das **Root**-Paket und wählen Sie **Neues Element | Paket**.
2. Geben Sie den Namen des neuen Pakets ein, (in diesem Beispiel. "Use Case View") und drücken Sie die Eingabetaste.



Hinzufügen eines Use Case-Diagramms zu einem Paket

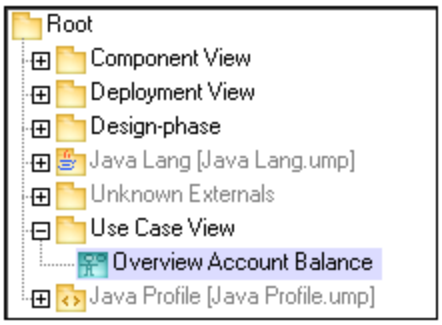
1. Rechtsklicken Sie auf das zuvor erstellte Paket "Use Case View".
2. Wählen Sie den Befehl **Neues Diagramm | UseCase-Diagramm**.

© 2016-2022 Altova GmbH



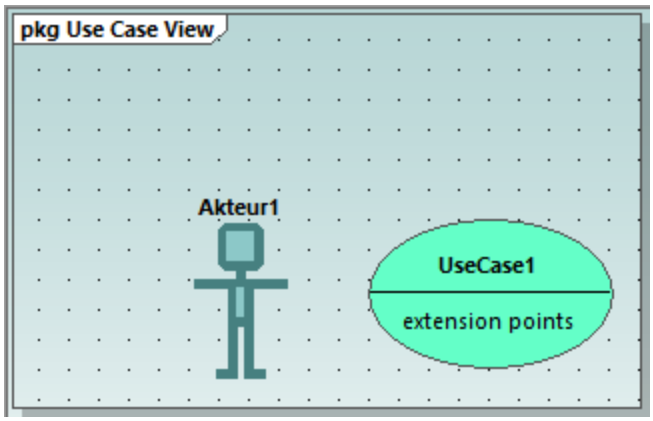
Dem Paket wurde nun im Fenster **Modell-Struktur** ein Use Case-Diagramm hinzugefügt und es wurde ein neues **Diagramm**-Fenster angelegt, das automatisch einen Standardnamen erhielt.

3. Doppelklicken Sie im Fenster "Modell-Struktur" auf den Diagrammnamen, ändern Sie ihn in "Overview Account Balance" und drücken Sie die **Eingabetaste**.

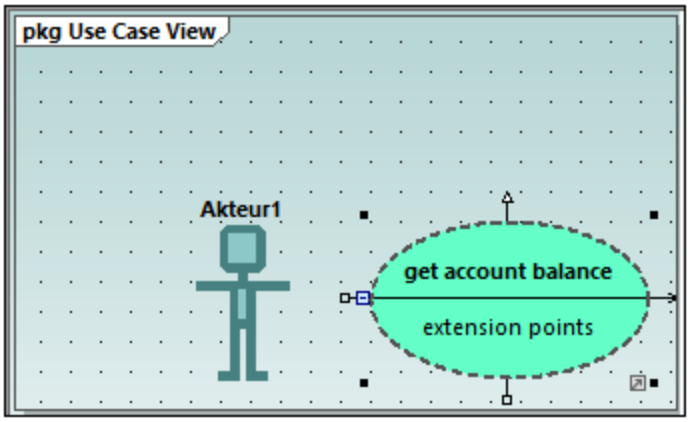


Hinzufügen von Use Case-Elementen zum Use Case-Diagramm

1. Rechtsklicken Sie in das neu erstellte Diagramm und wählen Sie **Neu | Akteur**. Das Akteurelement wird an der Mausposition eingefügt.
2. Klicken Sie auf das Use Case-Symbol  in der Symbolleiste und dann in das Diagrammfenster, um das Element einzufügen. Es wird ein Element "UseCase1" eingefügt. Beachten Sie, dass das Element und sein Name aktuell ausgewählt sind und dass seine Eigenschaften im Fenster **Eigenschaften** angezeigt werden.



3. Ändern Sie den Titel in "get account balance" und drücken Sie zur Bestätigung die Eingabetaste. Doppelklicken Sie auf den Titel, wenn er nicht aktiv ist.



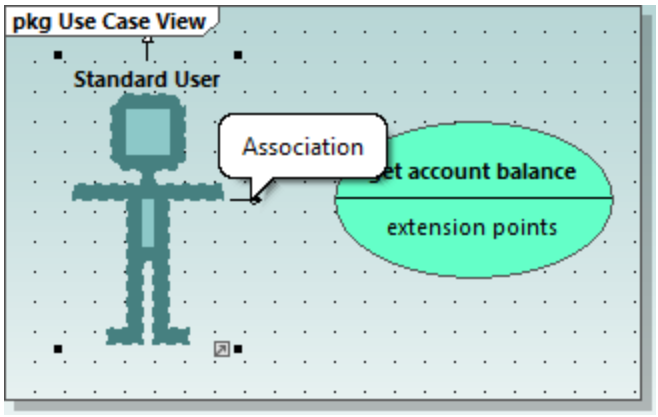
Anmerkung: Durch Drücken von Strg + Eingabetaste können Sie einen mehrzeiligen Use Case-Namen erstellen.

Bearbeiten von UModel Elementen: Ziehpunkte und Bereiche

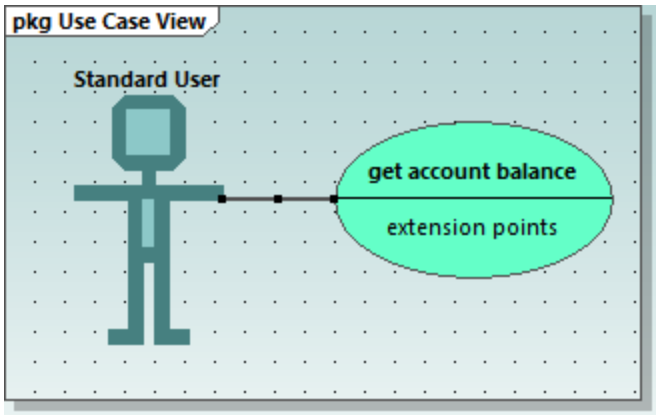
Wenn ein Element in der Diagrammanzeige ausgewählt ist, werden verschiedene Ziehpunkte und andere Elemente zum Bearbeiten angezeigt. Ziehpunkte dienen zum Erstellen von Beziehungen zwischen Elementen oder zum Anzeigen bzw. Ausblenden bestimmter Bereiche des Elements, wie unten gezeigt.

1. Doppelklicken Sie auf den Text "Akteur1" des Akteur-Elements, ändern Sie den Namen in "Standard User" und drücken Sie zur Bestätigung die **Eingabetaste**.
2. Platzieren Sie den Mauszeiger über den "**Ziehpunkt**" rechts vom Akteur. Es erscheint ein Tooltip mit dem Inhalt "Assoziation".

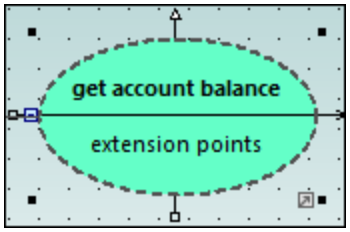
© 2016-2022 Altova GmbH



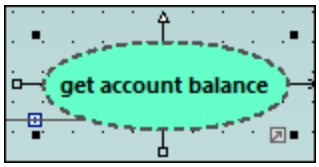
3. Klicken Sie auf den Ziehpunkt, ziehen Sie die Assoziationslinie nach rechts auf den Anwendungsfall (Use case) "get account balance". Zwischen Dem Akteur und dem Anwendungsfall wurde nun eine Assoziation erstellt. Die Eigenschaften der Assoziation sind auch im Fenster "Eigenschaften" zu sehen. Die neue Assoziation wurde unter dem Element "Beziehungen" des Use Case View-Pakets hinzugefügt.



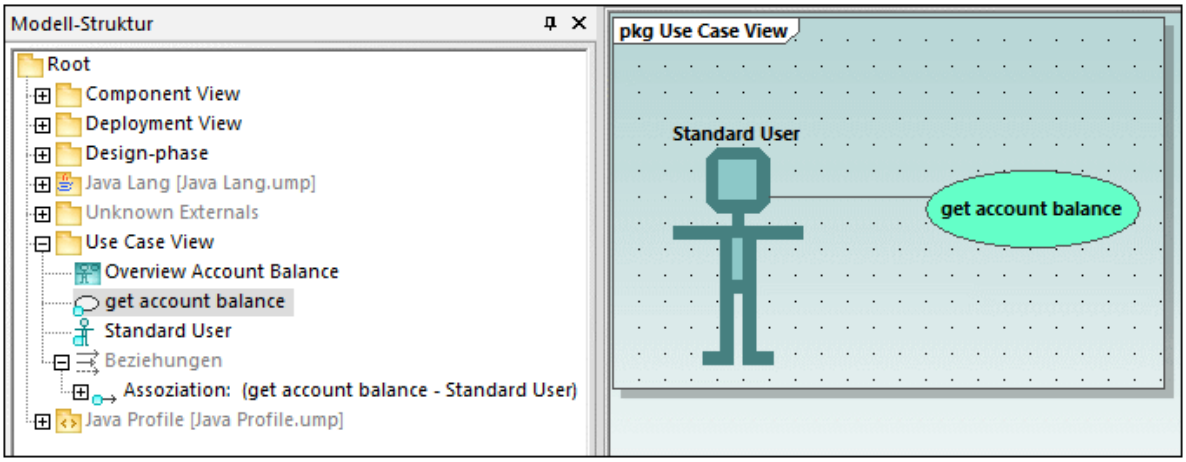
4. Klicken Sie auf den Anwendungsfall (Use Case) und ziehen Sie ihn nach rechts, um ihn neu zu positionieren. Die Eigenschaften der Assoziation sind auf dem Assoziationsobjekt zu sehen.
5. Klicken Sie auf den Anwendungsfall, um ihn auszuwählen und anschließend auf das Symbol zum Reduzieren am linken Rand der Use Case-Ellipse.



Der Bereich "extension points" wird nun ausgeblendet.



Ein blauer Punkt neben einem Elementsymbol im Fenster "Modell-Struktur" bedeutet, dass das Element im dem aktuellen Diagrammfenster sichtbar ist. So sehen Sie etwa in der Abbildung unten drei Elemente im Diagramm. Diese drei Elemente sind in der Modell-Struktur mit einem blauen Punkt gekennzeichnet:

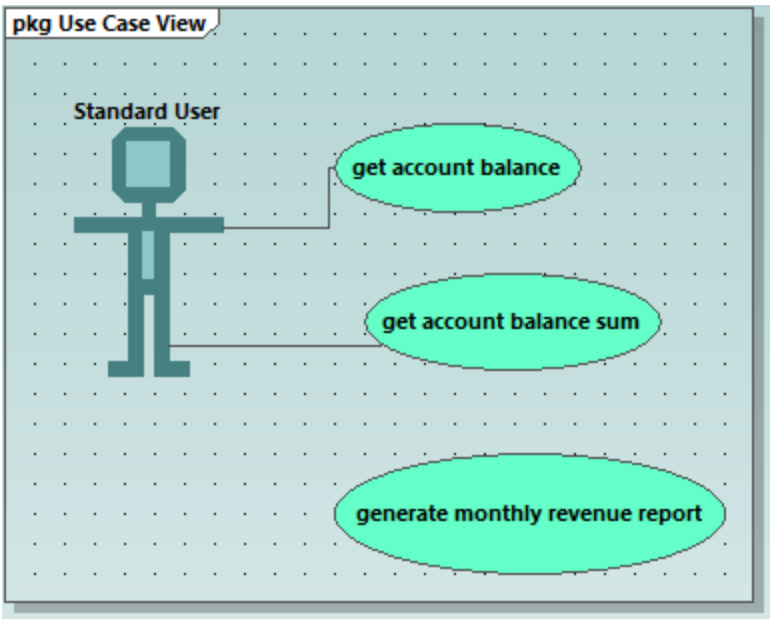


Wenn Sie die Größe des Akteurs anpassen, wird auch das Textfeld angepasst, das auch mehrere Zeilen enthalten kann. Mit **Strg + Eingabetaste** wird eine Zeilenschaltung in den Text eingefügt.

So stellen Sie das Use Case-Diagramm fertig:

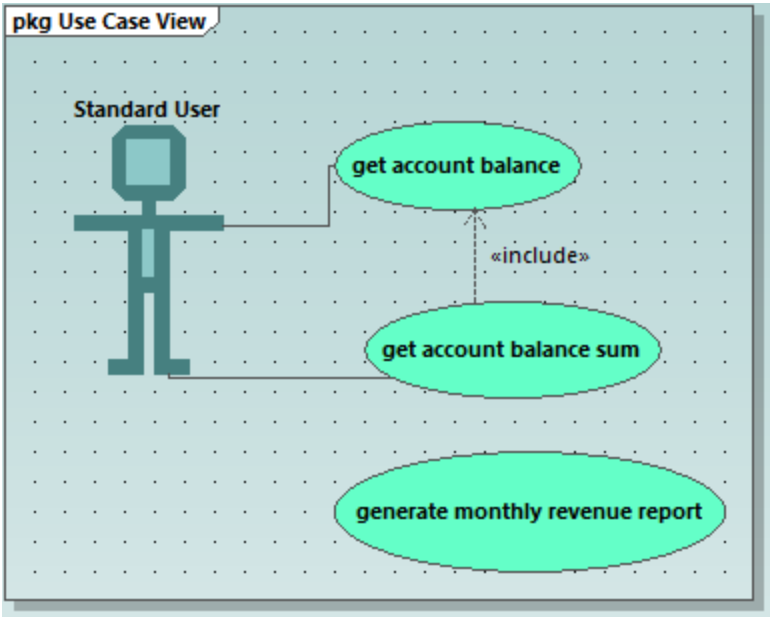
1. Klicken Sie in der Symbolleiste auf das Use Case-Symbol  und halten Sie **gleichzeitig** die Strg-Taste gedrückt.
2. Klicken Sie im Diagramm an zwei verschiedenen Stellen vertikal übereinander in das Fenster, um zwei weitere Use Cases hinzuzufügen. Lassen Sie anschließend die Strg-Taste los.
3. Geben Sie dem ersten Use Case den Namen "get account balance sum" und dem zweiten den Namen, "generate monthly revenue report".
4. Klicken Sie auf das Einklappsymbol der einzelnen Use Case-Ellipsen um den extension-Bereich auszublenden.
5. Klicken Sie auf den Akteur und erstellen Sie mit Hilfe des Assoziationsziehpunkts eine Assoziation zwischen "Standard User" und "get account balance sum".

© 2016-2022 Altova GmbH



So erstellen Sie eine "include"-Abhängigkeit zwischen Use Cases (und somit einen untergeordneten Use Case)

- Klicken Sie auf den **Include**-Ziehpunkt der "get account balance sum" Use Case-Ellipse (unter der Ellipse) und ziehen Sie die Abhängigkeit auf "get account balance". Es wird eine "include"-Abhängigkeit erstellt und auf dem strichlierten Pfeil wird das include-Stereotyp angezeigt.

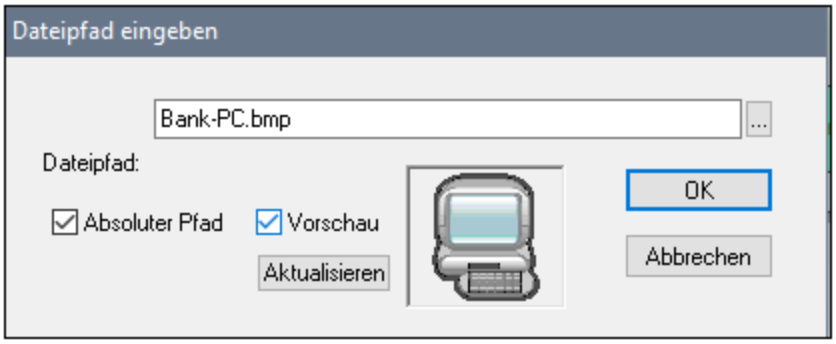


Einfügen von benutzerdefinierten Akteuren

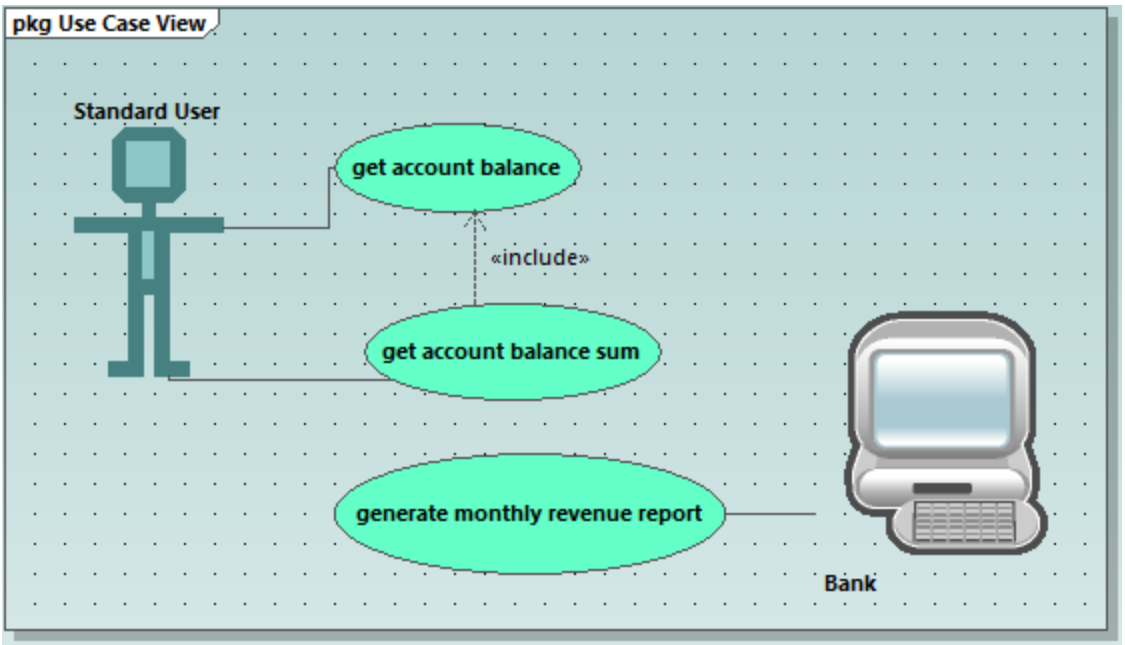
Beim Akteur im Use Case "generate monthly revenue report" handelt es sich nicht um eine Person, sondern um einen automatisierten Batch-Auftrag, der von einem Bankcomputer ausgeführt wird. In der Anleitung unten

wird erläutert, wie Sie einen neuen Akteur zum Diagramm hinzufügen und ein benutzerdefiniertes Bild dafür verwenden.

1. Fügen Sie mit Hilfe des Symbolleistensymbols **Akteur** einen Akteur in das Diagramm ein.
2. Benennen Sie den Akteur in "Bank" um.
3. Klicken Sie im Fenster "Eigenschaften" auf das Durchsuchen-Symbol neben dem Eintrag "Symboldateiname" und navigieren Sie zur **Datei Bank-PC.bmp**, die sich im selben Ordner wie das Projekt befindet.
4. Deaktivieren Sie das Kontrollkästchen **Absoluter Pfad**, um den Pfad relativ zu machen. Bei Auswahl von **Vorschau** wird im Dialogfeld eine Vorschau der ausgewählten Datei angezeigt.



5. Klicken Sie auf OK, um die Einstellungen zu bestätigen und fügen Sie den neuen Akteur ein. Verschieben Sie den neuen Akteur "Bank" und platzieren Sie ihn rechts vom untersten Use Case.
6. Klicken Sie in der Symbolleiste auf das Symbol **Assoziation** und ziehen Sie es vom Akteur "Bank" zum Use Case "generate monthly revenue report".
   Dies ist eine alternative Methode zum Erstellen einer Assoziation.



Anmerkung: Die Hintergrundfarbe, mit der die Bitmap-Grafik transparent gemacht werden kann, hat die RGB-Werte 82.82.82.

© 2016-2022 Altova GmbH

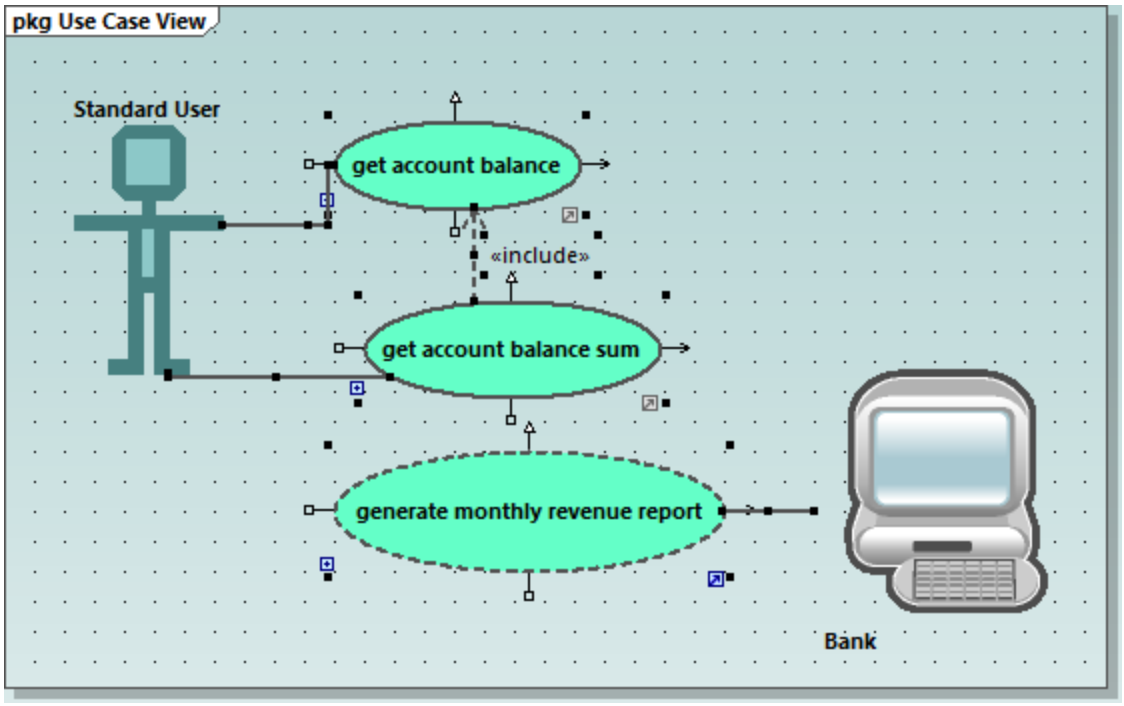
Ausrichten und Anpassen der Größe von Diagrammelementen

Wenn Sie Komponenten in einem Diagramm mit der Maus ziehen, erscheinen Hilfslinien, an denen Sie ein Element an jedem anderen Element im Diagramm ausrichten können. Diese Option kann folgendermaßen aktiviert bzw. deaktiviert werden:

1. Klicken Sie im Menü **Extras** auf **Optionen.**
2. Klicken Sie auf das Register **Ansicht.**
3. Aktivieren Sie in der Gruppe **Ausrichtung** das Kontrollkästchen **Hilfslinien aktivieren**.

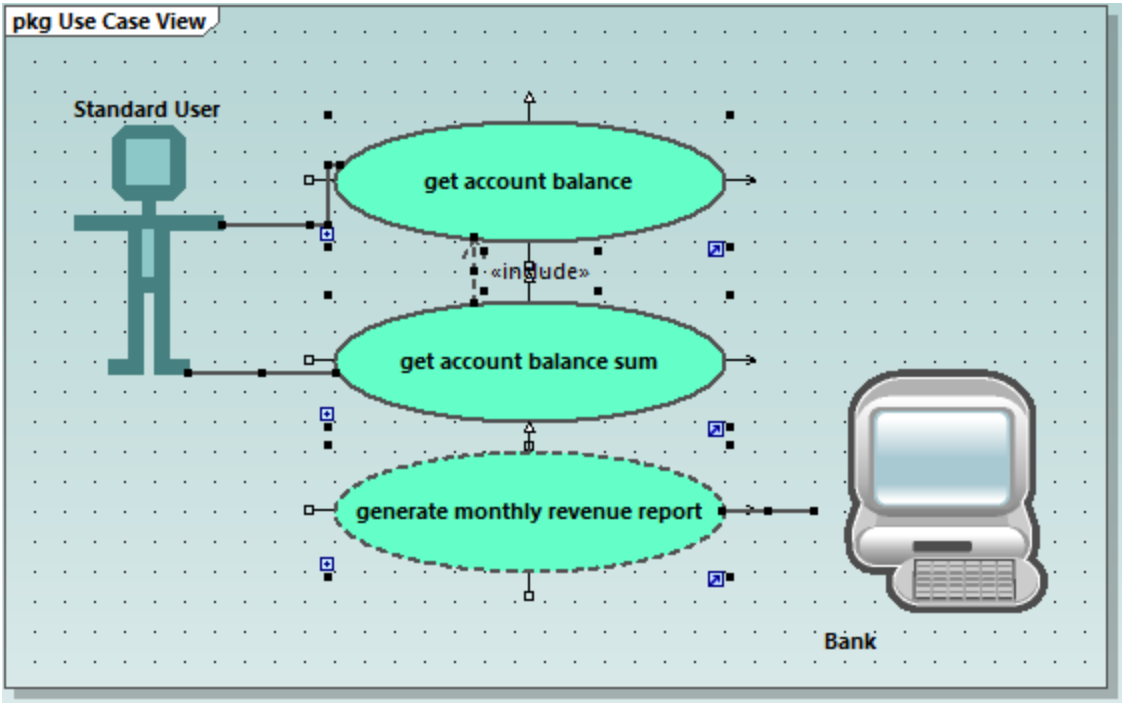
Sie können die Größe mehrerer Elemente auch folgendermaßen ausrichten und anpassen:

1. Ziehen Sie einen Auswahlrahmen auf, indem Sie auf dem Diagrammhintergrund ein Rechteck aufziehen, in dem von oben angefangen alle drei Use Cases enthalten sind. Alternativ dazu können Sie mehrere Elemente auch durch Drücken der **Strg**-Taste beim Anklicken der Elemente auswählen. Beachten Sie, das die Umrandung des zuletzt markierten Use Case im Diagramm und im Übersichtsfenster strichliert angezeigt wird.



   Alle Use Cases sind nun ausgewählt, wobei die unterste Ellipse die Basis für die folgenden Anpassungen bildet.

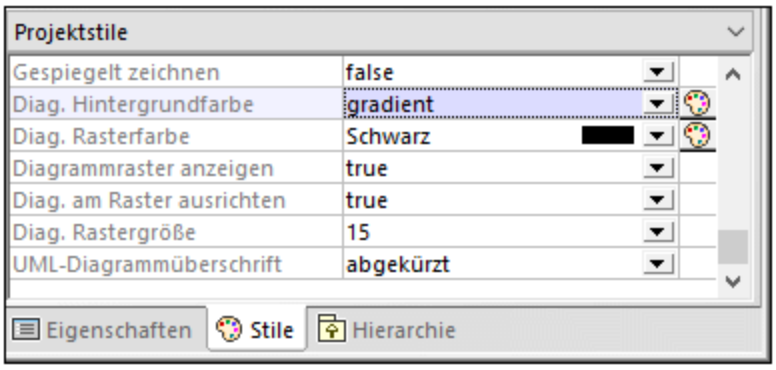
2. Klicken Sie in der Symbolleiste auf das Symbol **Größe angleichen** .
3. Klicken Sie auf das Symbol **Horizontal zentrieren** , um alle Ellipsen parallel auszurichten.



Ändern des Stils von Diagrammen in einem Projekt

Standardmäßig haben alle Diagramme des Tutorial-Projekts eine Hintergrunde mit Farbverlauf und es wird ein Hintergrundraster angezeigt. Das Aussehen von Diagrammen in einem Projekt kann konfiguriert werden. Um z.B. die Hintergrundfarbe aller Diagramme zu ändern, gehen Sie folgendermaßen vor:

1. Klicken Sie im Fenster **Eigenschaften** auf **Stile**.
2. Finden Sie unter **Projekstile** die Einstellung **Diag. Hintergrundfarbe**.



3. Ändern Sie den Wert von "gradient" in eine Farbe Ihrer Wahl.

So aktivieren oder deaktivieren Sie das Diagrammhintergrundraster:

- Ändern Sie die Einstellung **Diagrammraster anzeigen** von "true" in "false". (Wenn ein Diagramm gerade offen ist, können Sie alternativ dazu auch auf die Symbolleisten-Schaltfläche **Raster anzeigen** klicken.)

© 2016-2022 Altova GmbH

2.3 Klassendiagramme

In diesem Tutorialabschnitt werden die folgenden Aufgaben erläutert:

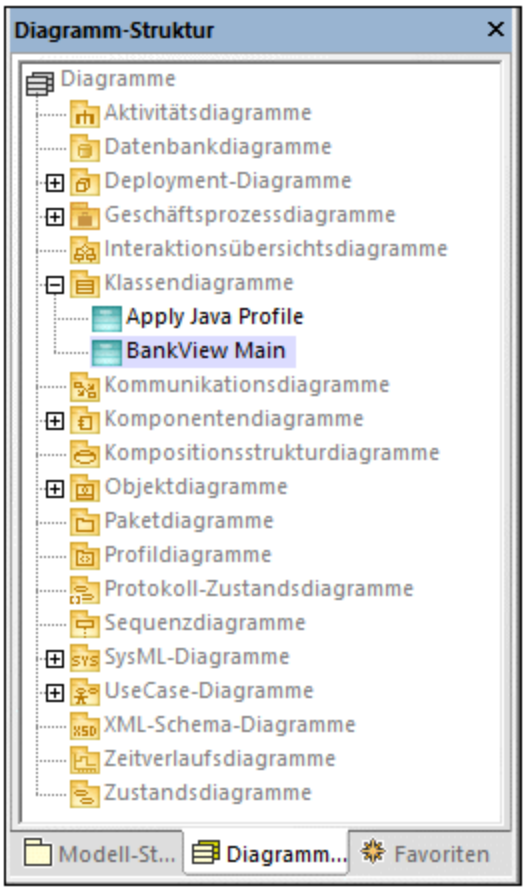
- Hinzufügen einer abstrakten Klasse zu einem vorhandenen Klassendiagramm
- Hinzufügen von Klasseneigenschaften und -operationen und Definieren von Parametern, ihrer Richtung und ihres Typs
- Hinzufügen eines Rückgabetyps zu einer Operation
- Ändern von Symbolen in UML-konforme Symbole
- Löschen und Ausblenden von Klasseneigenschaften und -operationen
- Erstellen einer Kompositions-Assoziation zwischen zwei Klassen

Um fortzufahren, starten Sie UModel und öffnen Sie das Projekt **BankView-start.ump** (siehe auch Öffnen des Tutorial-Projekts 19).

Hinzufügen einer abstrakten Klasse

Das Diagramm, zu dem die abstrakte Klasse hinzugefügt wird, hat den Namen "BankView Main" und kann folgendermaßen geöffnet werden:

1. Erweitern Sie im Fenster **Diagramm-Struktur** das Paket "Klassendiagramme", um alle im Projekt enthaltenen Klassendiagramme zu sehen.



2. Wählen Sie eine der folgenden Methoden:

- Doppelklicken Sie auf das "BankView Main"-Diagrammsymbol.
- Kicken Sie mit der rechten Maustaste auf das Diagramm und wählen Sie im Kontextmenü den Befehl **Diagramm öffnen**.

Anmerkung: Sie können das Diagramm auch über das Fenster **Modell-Struktur** öffnen. Suchen Sie das Diagramm zuerst unter dem Paket "Root | Design-phase | BankView | com | altova | bankview" und verwenden Sie anschließend eine der oben beschriebenen Methoden, um es zu öffnen.

Im Klassendiagramm sehen Sie zwei konkrete Klassen mit einer Kompositions-Assoziation zwischen diesen Klassen.

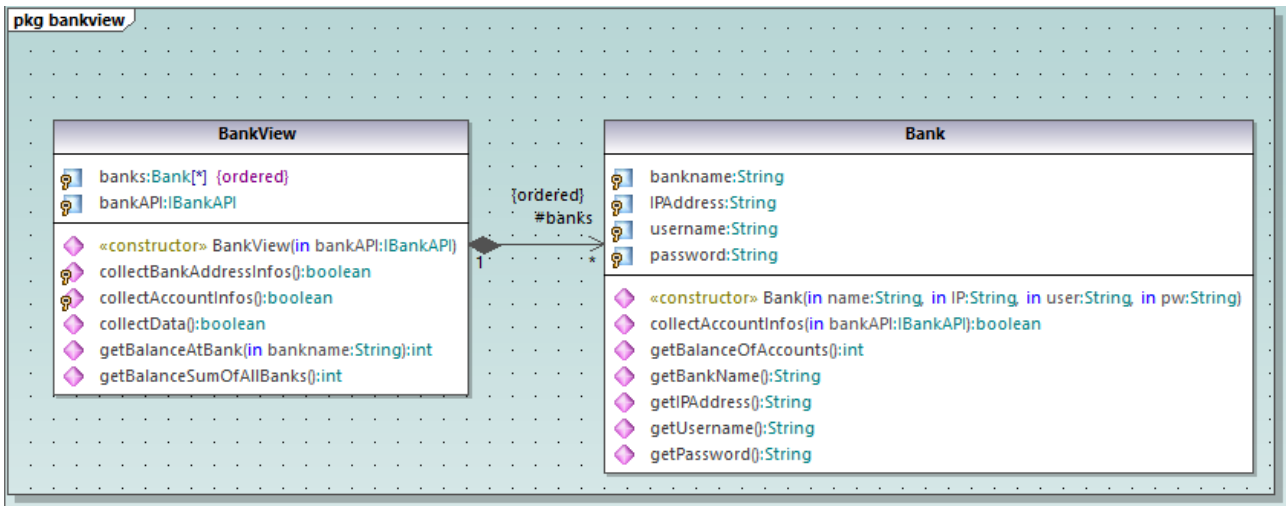
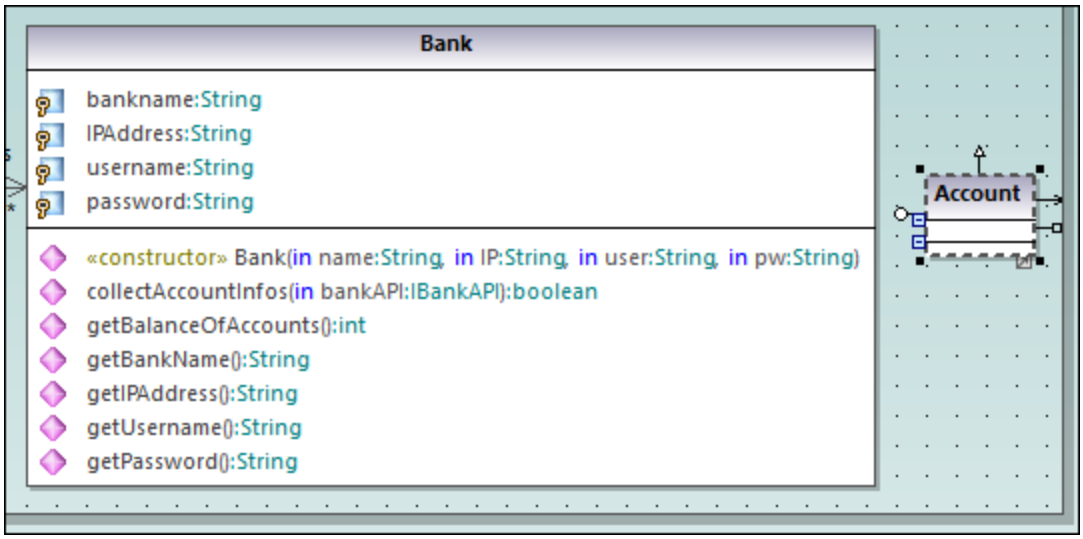


Diagramm "Bank View Main"

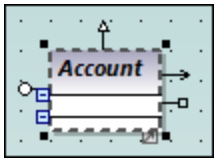
Die neue abstrakte Klasse kann folgendermaßen hinzugefügt werden:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **Klasse**  und klicken Sie anschließend rechts von der Klasse `Bank`, um die neue Klasse einzufügen.
2. Doppelklicken Sie auf den Namen der neuen Klasse und ändern Sie ihn in `Account`.

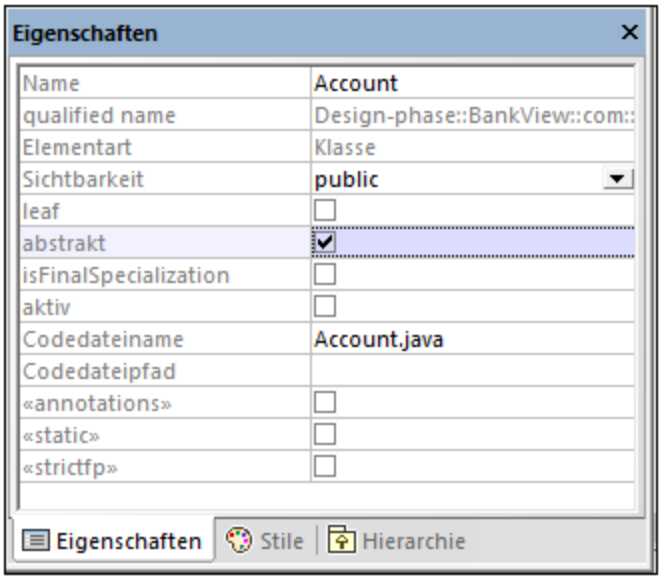
© 2016-2022 Altova GmbH



3. Aktivieren Sie im Fenster **Eigenschaften** das Kontrollkästchen **abstrakt,** um die Klasse zu einer abstrakten zu machen. Der Klassentitel wird nun kursiv gedruckt angezeigt. Dadurch werden abstrakte Klassen gekennzeichnet.

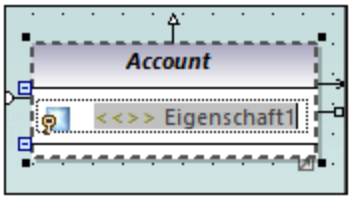


4. Geben Sie in das Textfeld **Codedateiname** "Account.java" ein, um die Java-Klasse zu definieren.

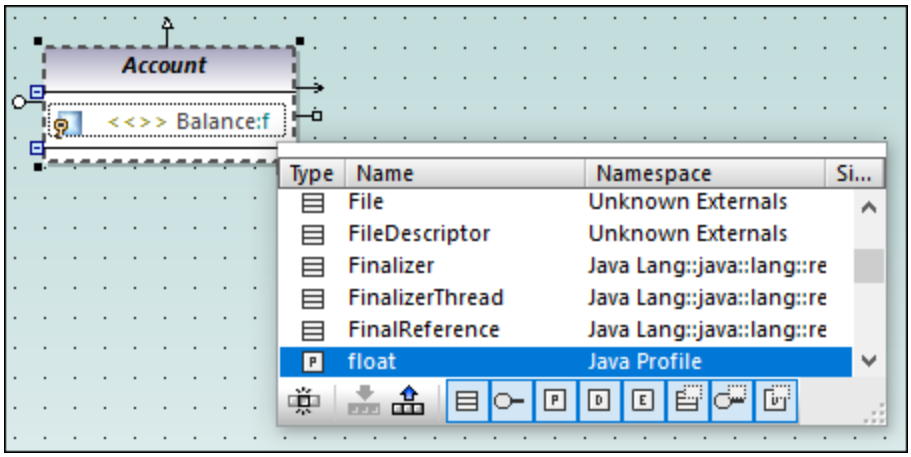


Hinzufügen von Eigenschaften zu einer Klasse

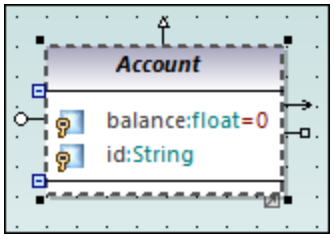
1. Klicken Sie mit der rechten Maustaste auf die Klasse "Account" und wählen Sie **Neu | Eigenschaft** oder drücken Sie **F7**. Daraufhin wird eine Standardeigenschaft `Eigenschaft 1` mit Stereotypmarkierungen << >> eingefügt.



2. Ändern Sie den Eigenschaftsnamen in `balance` gefolgt von einem Doppelpunkt (:) . Daraufhin wird eine Dropdown-Liste mit allen gültigen Typen angezeigt.
3. Geben Sie "f" ein und drücken Sie die Eingabetaste, um den Rückgabetyp "float" einzugeben. Beachten Sie dass, die Groß- und Kleinschreibung in Dropdown-Listen eine Rolle spielt.



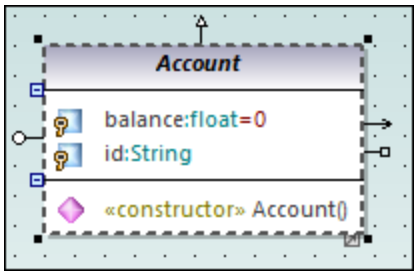
4. Hängen Sie in derselben Zeile "=0" an, um den Standardwert zu definieren.
5. Erstellen Sie auf dieselbe Art, wie beschrieben, eine neue Eigenschaft `id` vom Typ `String`.



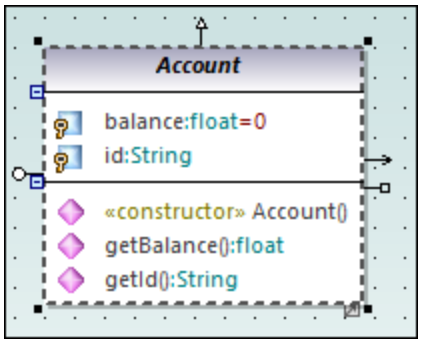
Hinzufügen von Operationen zu einer Klasse

1. Klicken Sie mit der rechten Maustaste auf die Klasse `Account` und wählen Sie **Neu | Operation** oder drücken Sie **F8**.
2. Geben Sie als Operationsnamen "Account()" ein. Beachten Sie, dass sich das Stereotyp in <<`constructor`>>, geändert hat, da der Operationsname mit dem Klassennamen identisch ist.

© 2016-2022 Altova GmbH

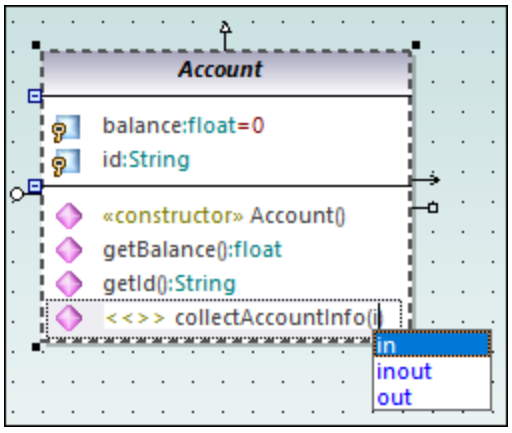


3. Fügen Sie auf dieselbe Art zwei weitere Operationen hinzu und geben Sie ihnen den Namen `getBalance():float` bzw. `getId():String`.

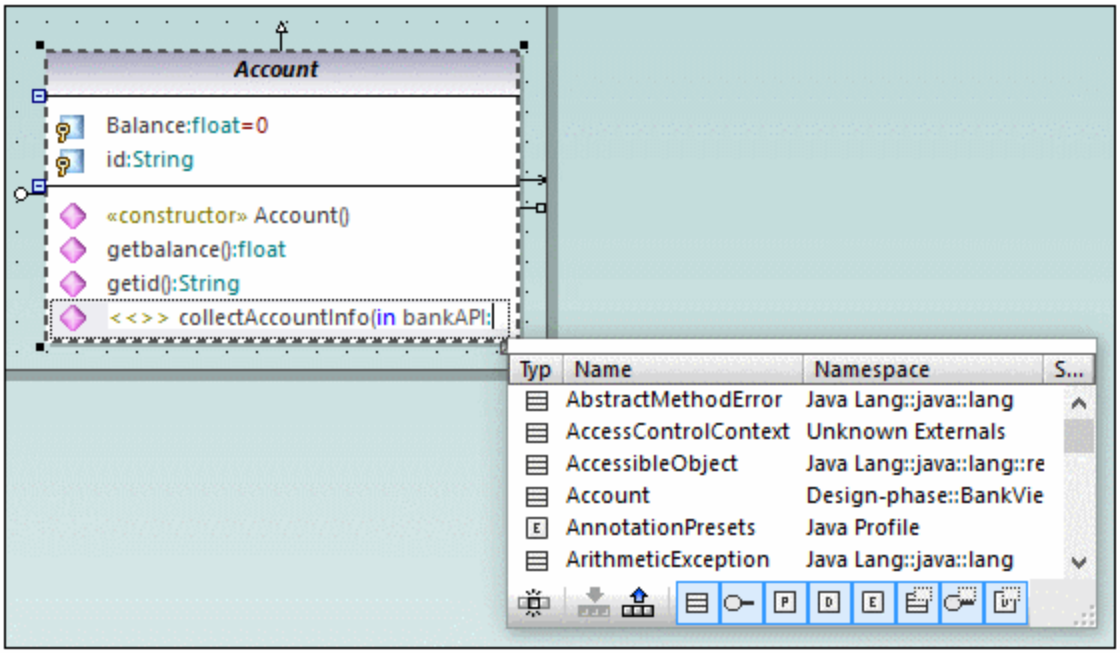


Fügen wir nun einen neue Operation hinzu, die einen Parameter hat. Außerdem werden wir die Richtung und den Typ des Parameters definieren.

1. Drücken Sie **F8,** um eine weitere Operation, `collectAccountInfo()`, zu erstellen.
2. Platzieren sie den Mauscursor innerhalb der Klammern und geben Sie "i" ein. Daraufhin wird eine Dropdown-Liste angezeigt, aus der Sie die Parameterrichtung: `in`, `inout` oder `out` auswählen können.



3. Wählen Sie den Eintrag "in" aus der Dropdown-Liste aus, geben Sie ein Leerzeichen ein und bearbeiten Sie den Eintrag weiter in derselben Zeile.
4. Geben Sie als Parameter "bankAPI", gefolgt von einem Doppelpunkt (:) ein. Daraufhin wird eine Dropdown-Liste angezeigt, aus der Sie den Parametertyp auswählen können.



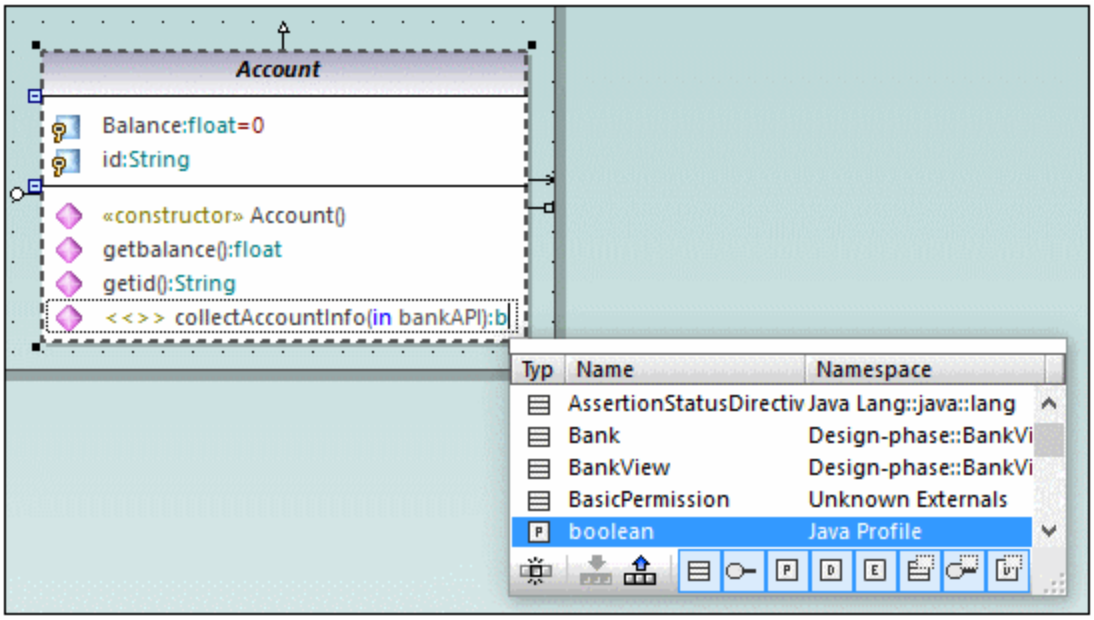
5. Wählen Sie **IBankAPI** aus der Dropdown-Liste aus.

Hinzufügen eines Rückgabetyps zu einer Operation

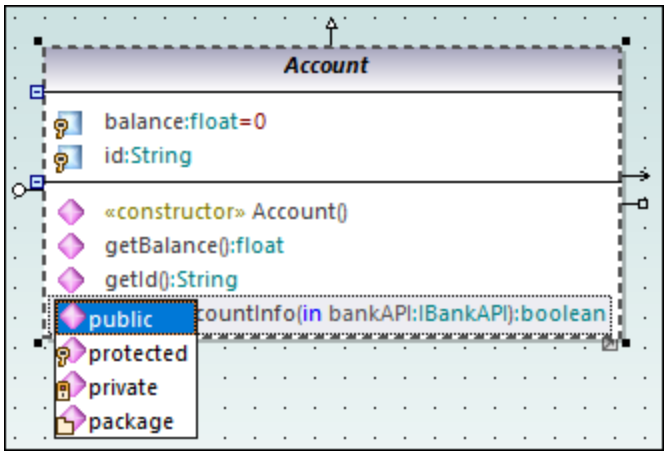
Bisher wurde der Operationsparameter hinzugefügt, hat aber noch keinen Rückgabetyp. So fügen Sie einen Rückgabetyp hinzu:

1. Platzieren Sie den Mauscursor hinter das schließende Klammerzeichen ")" und geben Sie einen Doppelpunkt ein (:). Daraufhin wird eine Dropdown-Liste angezeigt, aus der Sie den Rückgabetyp auswählen können.
2. Drücken Sie die Taste "b" und wählen Sie als Datentyp `boolean` aus.

© 2016-2022 Altova GmbH



Um die Sichtbarkeit einer Operation zu definieren (z.B. "private", "protected", "public"), klicken Sie auf das Symbol vor dem Operationsnamen und wählen Sie den gewünschten Wert aus, z.B.:



Die Sichtbarkeit "package" gilt für Java. In C# können Sie die Sichtbarkeit mit Hilfe von "package" als "internal" definieren. Nähere Informationen zu den UModel-Elemententsprechungen in den einzelnen Sprachen finden Sie unter UModel-Elemententsprechungen 247.

Ändern von Symbolen in UML-konforme Symbole

Die Sichtbarkeitssymbole können gegebenenfalls in UML-konforme Symbole geändert werden:

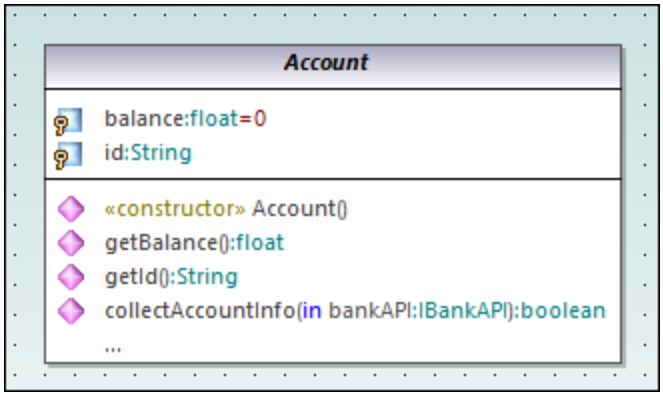
1. Wählen Sie im Fenster **Stile**, aus der Dropdown-Liste ganz oben, den Eintrag **Projektstile**.
2. Scrollen Sie hinunter zur Einstellung **Sichtbarkeit anzeigen** und wählen Sie **UML-Stil** aus.

Löschen und Ausblenden von Klasseneigenschaften und -Operationen aus einem Klassendiagramm

Drücken Sie **F8,** um z.B. eine Operation `Operation1` zur Klasse `Account` hinzuzufügen.

Um diese Operation zu löschen, wählen Sie sie aus und drücken Sie die Taste **Entfernen**. (Alternativ dazu klicken Sie mit der rechten Maustaste darauf und wählen Sie im Kontextmenü die Option **Löschen** aus). Daraufhin erscheint ein Meldungsfeld, in dem Sie gefragt werden, ob Sie das Element aus dem Projekt löschen möchten. Klicken Sie auf **Ja**, um die `Operation1` aus dem Klassendiagramm und aus dem Projekt zu löschen.

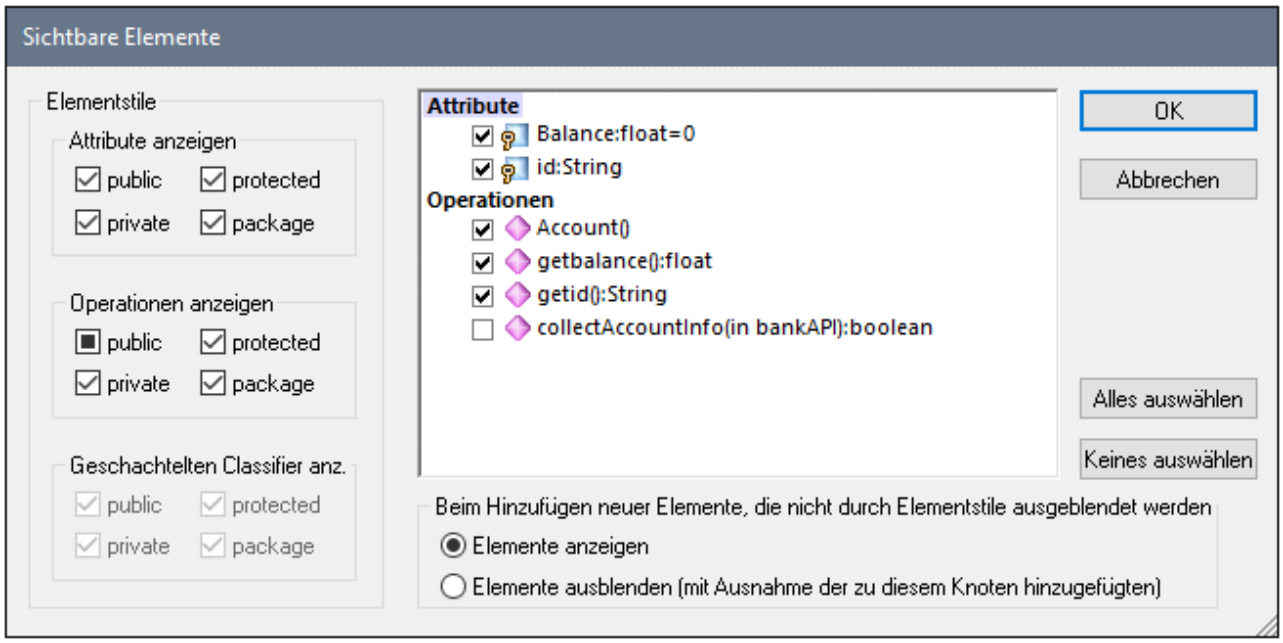
Um die Operation aus der Klasse im Diagramm, nicht aber aus dem Projekt zu löschen, drücken Sie **Strg+Entf**. Dadurch wird die Operation aus dem Diagramm ausgeblendet, obwohl sie im Projekt weiterhin vorhanden ist. Klassen mit ausgeblendeten Mitgliedern werden mit einem Auslassungszeichen (...) angezeigt (siehe Abbildung unten):



Eine Klasse mit ausgeblendeten Operationen

Um die Operation wieder einzublenden, doppelklicken Sie auf die Auslassungspunkte am unteren Rand der Klasse. Daraufhin wird ein Dialogfeld angezeigt, in dem Sie die Elemente, die im Diagramm angezeigt werden sollen, auswählen können, z.B.:

© 2016-2022 Altova GmbH



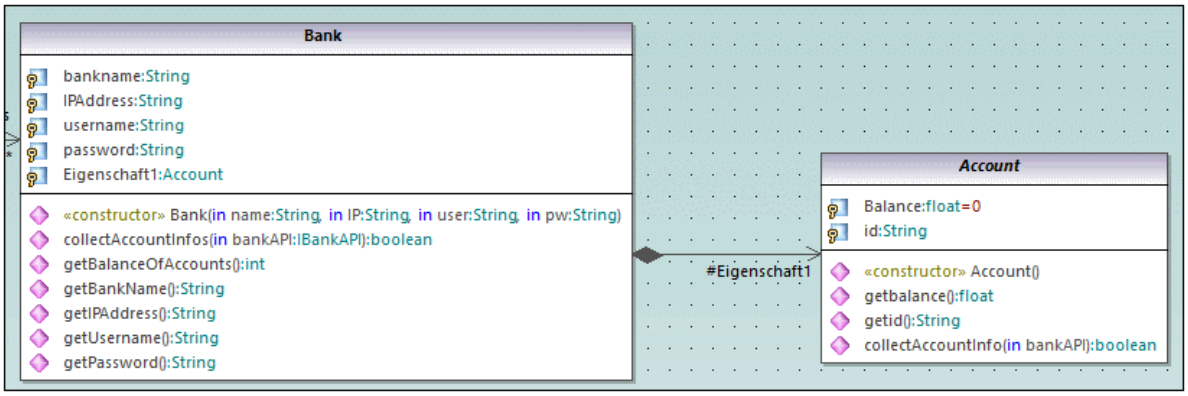
Dialogfeld "Sichtbare Elemente"

Sie können UModel auch so konfigurieren, dass ein Meldungsfeld angezeigt wird, wenn Sie versuchen, ein Objekt aus dem Diagramm zu löschen. Gehen Sie dazu folgendermaßen vor:

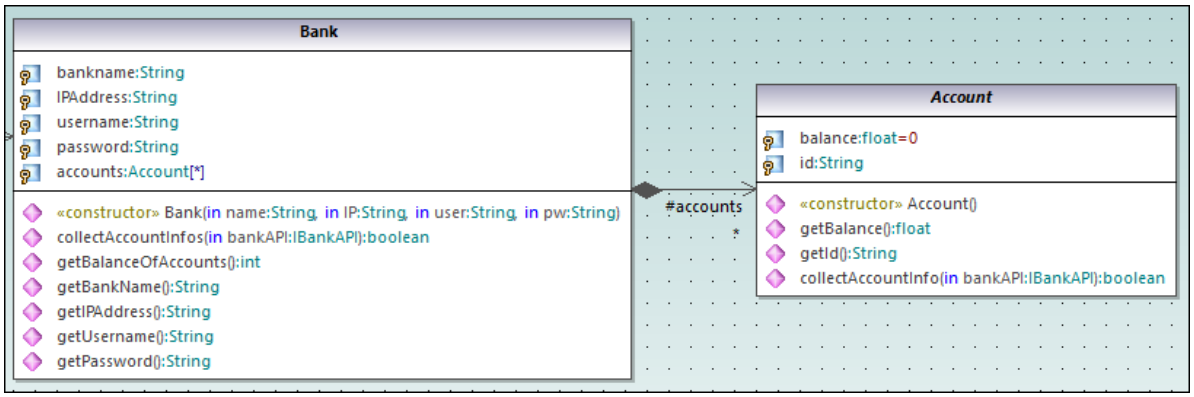
1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Bearbeiten**.
3. Deaktivieren Sie unter **Rückfrage vor Löschen aus dem Projekt** das Kontrollkästchen **in Diagrammen**.

Erstellen einer Kompositions-Assoziation zwischen den Klassen "Bank" und "Account"

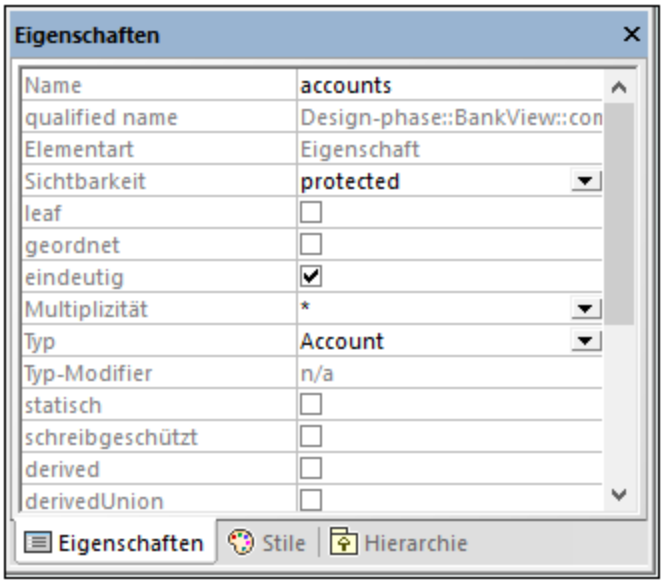
1. Klicken Sie auf die Symbolleisten-Schaltfläche **Komposition**  und ziehen Sie diese anschließend von der Klasse `Bank` zur Klasse `Account`. Die Klasse wird markiert, wenn die Assoziation erstellt werden kann. In der Klasse Bank wird eine neue Eigenschaft (`Eigenschaft1:Account`) erstellt und die beiden Klassen werden durch einen Kompositions-Assoziationspfeil verbunden.



2. Doppelklicken Sie auf die neue Eigenschaft `Eigenschaft1` in der Klasse `Bank` und ändern Sie sie in "accounts", ohne dabei die Typdefinition `Account` (in Blaugrün) zu löschen.
3. Drücken Sie die Taste **Ende**, um den Textcursor ans Ende der Zeile zu setzen.
4. Geben Sie die das eckige Klammer auf-Zeichen ([ ) ein und wählen Sie aus der Dropdown-Liste ein Sternchen (*) aus. Damit definieren Sie die *Multiplizität*, d.h. dass eine Bank mehrere Konten (accounts) haben kann.



Beachten Sie, dass der zuvor zum Diagramm hinzugefügte Multipilizitätsbereich auch im Fenster **Eigenschaften** angezeigt wird:



2.3.1 Erstellen von abgeleiteten Klassen

In diesem Tutorialabschnitts werden die folgenden Aufgaben erläutert:

- Hinzufügen eines neuen Klassendiagramms zum Projekt
- Hinzufügen bestehender Klassen zu einem Diagramm
- Hinzufügen einer neuen Klasse zu einem Diagramm
- Erstellen von abgeleiteten Klassen anhand einer abstrakten Klassen mit Hilfe von Generalisierungen

© 2016-2022 Altova GmbH

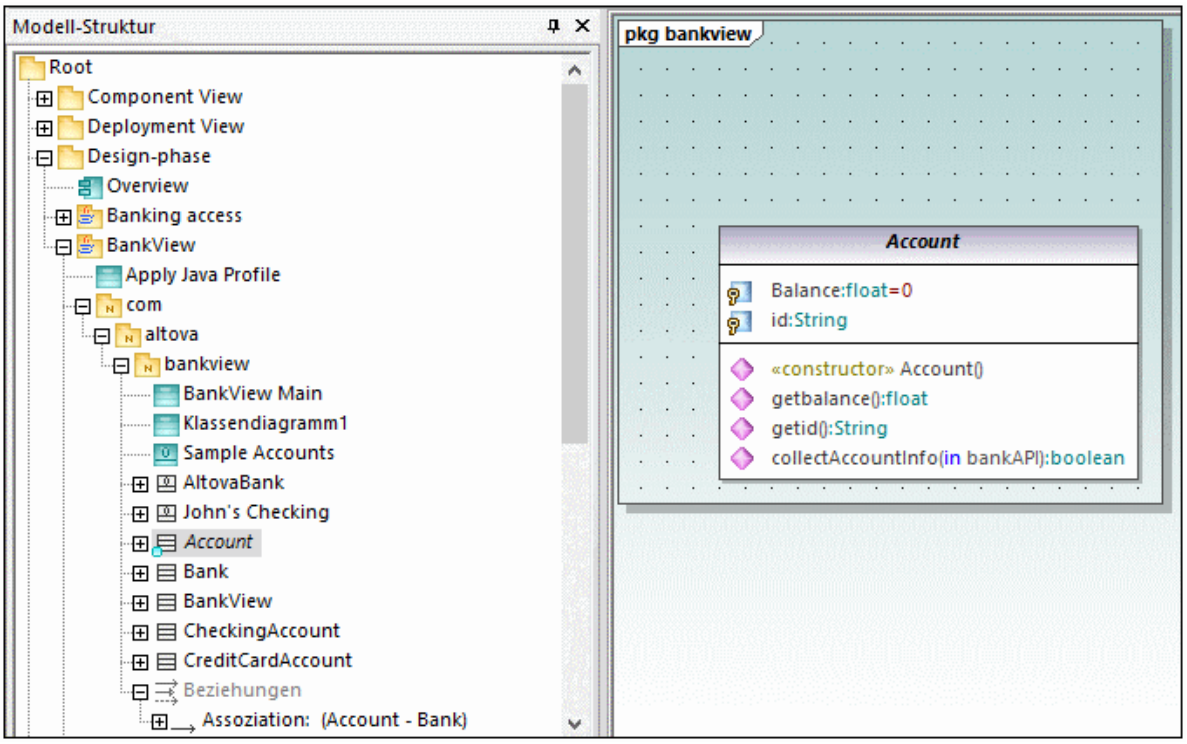
Anmerkung: Es wird davon ausgegangen, dass Sie den vorherigen Tutorial-Abschnitt, Klassendiagramme 31 bereits durchgearbeitet haben, um die abstrakte Klasse `Account` zu erstellen.

Erstellen eines neuen Klassendiagramms

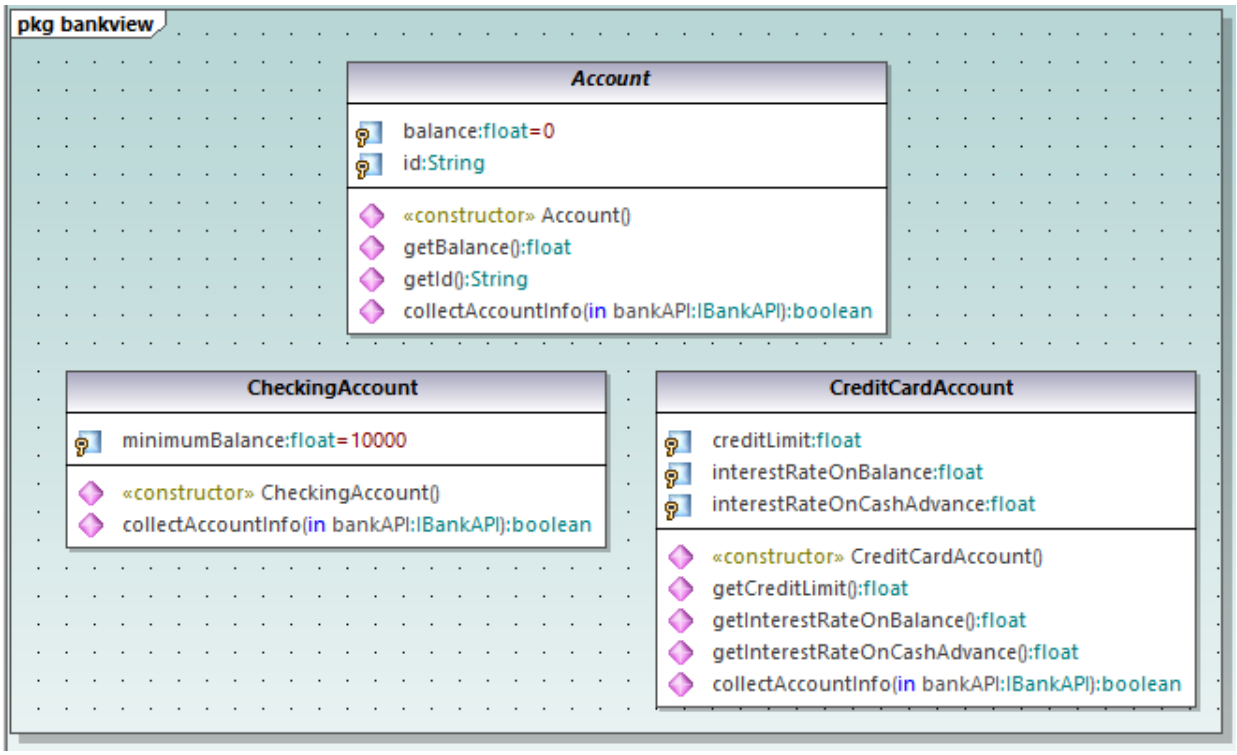
1. Rechtsklicken Sie im Fenster **Modell-Struktur** auf das Paket `bankview` (unter **Root | Design-phase | BankView | com | altova**) und wählen Sie **Neues Diagramm | Klassendiagramm**.
2. Doppelklicken Sie auf den neuen Eintrag "Klassendiagramm1", benennen Sie ihn in "Account Hierarchy" um und drücken Sie die **Eingabetaste**. Im Arbeitsbereich wird nun das neue Diagramm "Account Hierarchy" angezeigt.

Hinzufügen von bestehenden Klassen zu einem Diagramm

1. Klicken Sie im Paket `bankview` auf die Klasse `Account` (unter **com | altova | bankview**) und ziehen Sie sie in das Diagramm.



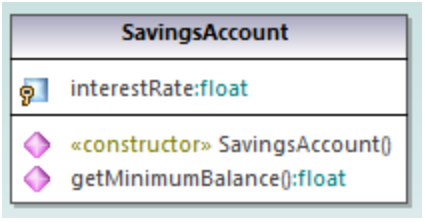
2. Klicken Sie auf die Klasse `CheckingAccount` (im selben Paket) und ziehen Sie diese ebenfalls in das Diagramm. Platzieren Sie die Klasse links unterhalb der Klasse "Account".
3. Fügen Sie auf dieselbe Art die Klasse `CreditCardAccount` ein. Platzieren Sie sie rechts von der Klasse `CheckingAccount`.



Hinzufügen einer neuen Klasse

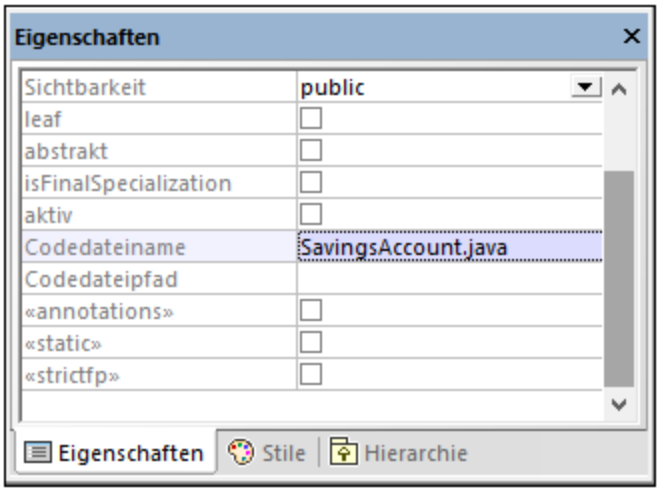
Die dritte abgeleitete Klasse, `SavingsAccount`, wird manuell zum Diagramm hinzugefügt.

1. Rechtsklicken Sie auf das Diagramm und wählen Sie **Neu | Klasse**. Daraufhin wird automatisch eine neue Klasse zum richtigen Paket (`bankview`) hinzugefügt, welches das aktuelle Klassendiagramm "Account Hierarchy" enthält.
2. Doppelklicken Sie auf den Klassennamen und ändern Sie ihn in `SavingsAccount`.
3. Erstellen Sie die Klassenstruktur wie unten gezeigt. Um Eigenschaften und Operationen hinzuzufügen, verwenden Sie die im vorherigen Tutorialabschnitt beschriebenen Methoden (siehe Klassendiagramme) 31.



3. Geben Sie auf dem Register **Eigenschaften** im Textfeld "Codedateiname" "SavingsAccount.java" ein, um die Java Codeklasse zu definieren.

 © 2016-2022 Altova GmbH



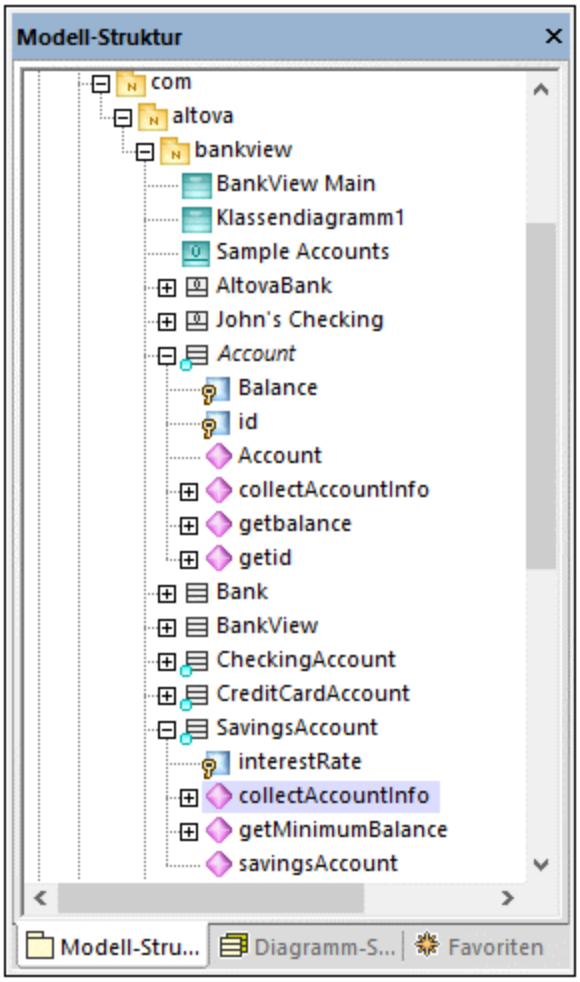
Eigenschaften und Operationen können direkt mit Hilfe von Drag & Drop oder den Standard-Tastenkombinationen von einer Klasse in die andere kopiert oder verschoben werden. Dies funktioniert in folgenden Fällen:

- innerhalb einer Klasse im aktuellen Diagramm
- zwischen verschiedenen Klassen im selben Diagramm
- im Fenster **Modell-Struktur**
- zwischen verschiedenen UML-Diagrammen durch Ziehen der kopierten Daten in ein anderes Diagramm

Dies kann mit Drag-and-Drop sowie den Standardtastaturbefehlen **Kopieren** und **Einfügen** (**Strg+C, Strg+V**) bewerkstelligt werden, siehe auch Umbenennen, Verschieben und Kopieren von Elementen 116. In diesem Beispiel können Sie die Operation `collectAccountInfo()` schnell aus der Klasse `Account` in die neue Klasse `SavingsAccount` kopieren. Gehen Sie dabei folgendermaßen vor:

1. Erweitern Sie die Klasse `Account` in der **Modell-Struktur**.
2. Rechtsklicken Sie auf die Operation `collectAccountInfo` und wählen Sie **Kopieren**.
3. Rechtsklicken Sie auf die Klasse `SavingsAccount` und wählen Sie **Einfügen.**

Die Operation wird in die Klasse `SavingsAccount` kopiert, wobei diese Klasse automatisch erweitert wird, um die neue Operation anzuzeigen.

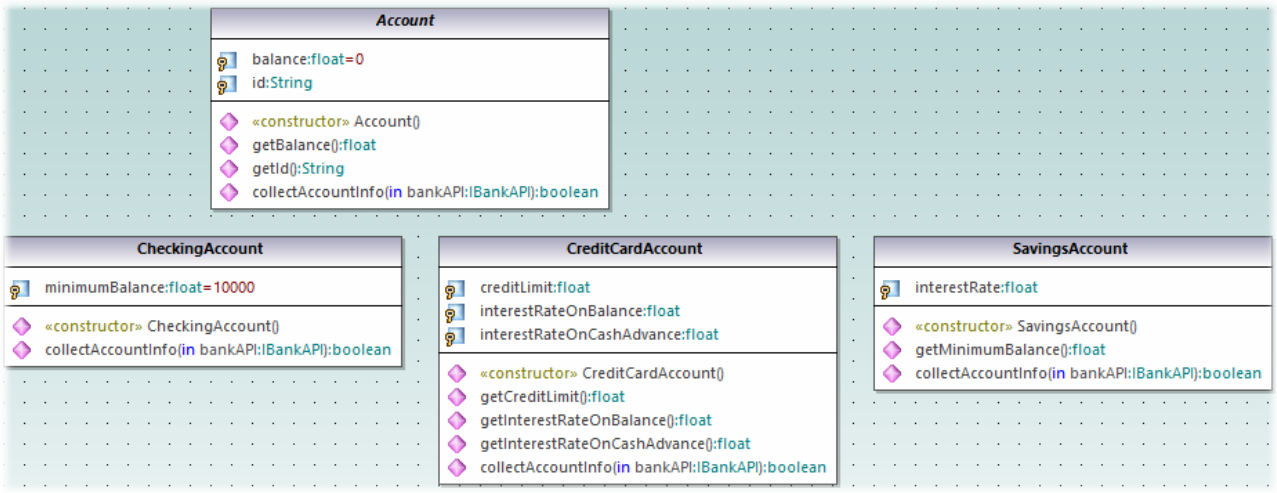


Die neue Operation ist nun auch im Klassendiagramm in der Klasse `SavingsAccount` zu sehen.

Erstellen von abgeleiteten Klassen mittels Generalisierung/Spezialisierung

Derzeit enthält das Klassendiagramm die abstrakte Klasse `Account` sowie drei spezifische Klassen.

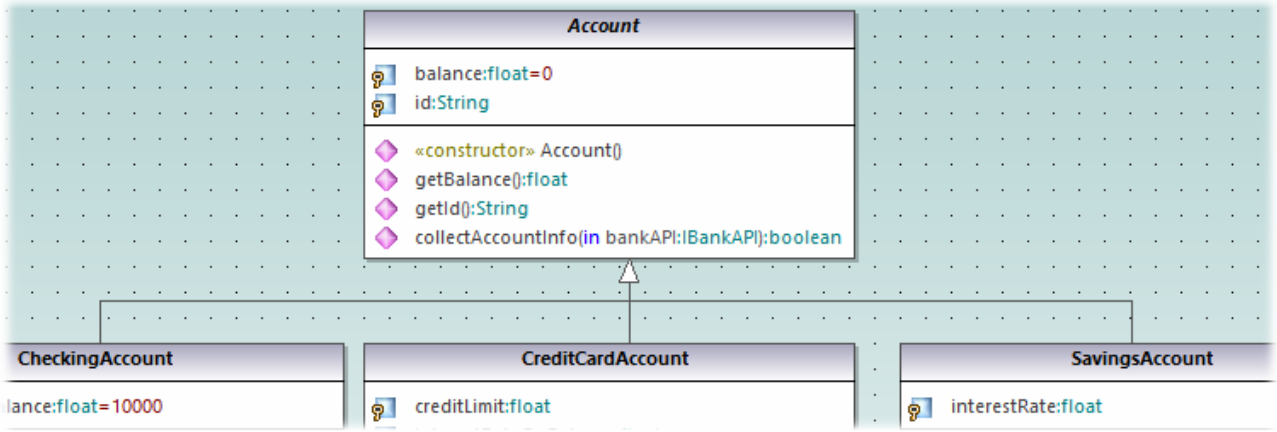
© 2016-2022 Altova GmbH



Wir wollen nun eine Generalisierungs-/Spezialisierungsbeziehung zwischen "`Account`" und den spezifischen Klassen erstellen, d.h. wir wollen drei abgeleitete konkrete Klassen erstellen.

1. Klicken Sie in der Symbolleiste auf das Symbol **Generalisierung**  und halten Sie die **Strg**-Taste gedrückt.
2. Ziehen Sie den Cursor von `CreditCardAccount` in die Klasse `Account`.
3. Ziehen Sie von der Klasse `CheckingAccount` auf die *Pfeilspitze* der zuvor erstellten Generalisierung.
4. Ziehen Sie von der Klasse `SavingsAccount` auf die *Pfeilspitze* der zuvor erstellten Generalisierung: lassen Sie die **Strg**-Taste jetzt los.

Zwischen den drei Unterklassen und der Überklasse `Account` werden Generalisierungspfeile erstellt.



2.4 Objektdiagramme

In diesem Tutorialabschnitt werden die folgenden Aufgaben erläutert:

- Kombinieren von Klassen- und Objektdiagrammen zu einem Diagramm
- Erstellen von Objekten/Instanzen und Definieren von Beziehungen zwischen diesen
- Formatieren von Assoziationen/Objektbeziehungen
- Eingabe von realen Daten in Objekte/Instanzen

Um fortzufahren, starten Sie UModel und öffnen Sie das Projekt **BankView-start.ump** (siehe auch Öffnen des Tutorial-Projekts 19). Das Projekt enthält ein vordefiniertes Objektdiagramm "Sample Accounts", anhand dessen die oben beschriebenen Aufgaben demonstriert werden.

Kombinieren von Objekten und Klassen zu einem Diagramm

Navigieren Sie im Fenster **Modell-Struktur** zu folgendem Pfad: **Root | Design-phase | BankView | com | altova | bankview** und doppelklicken Sie auf das Symbol neben dem Diagramm "Sample Accounts".

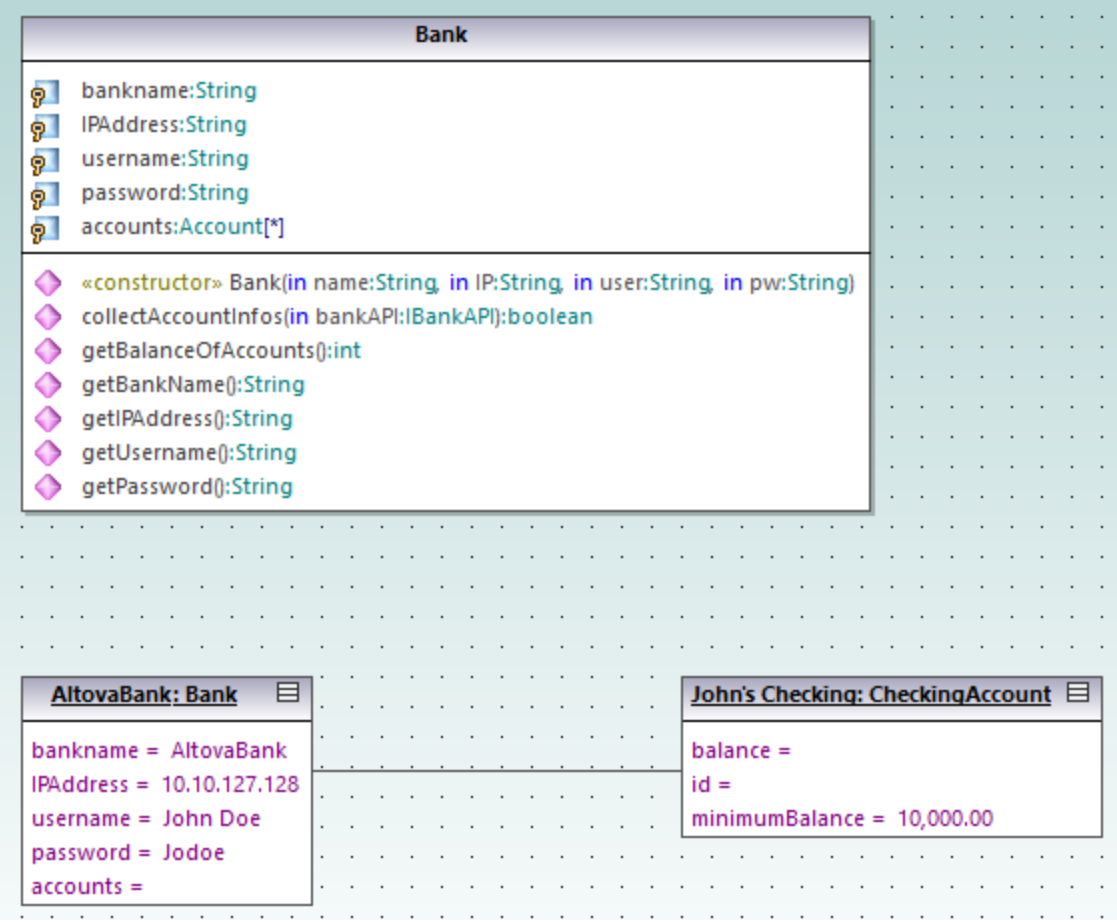
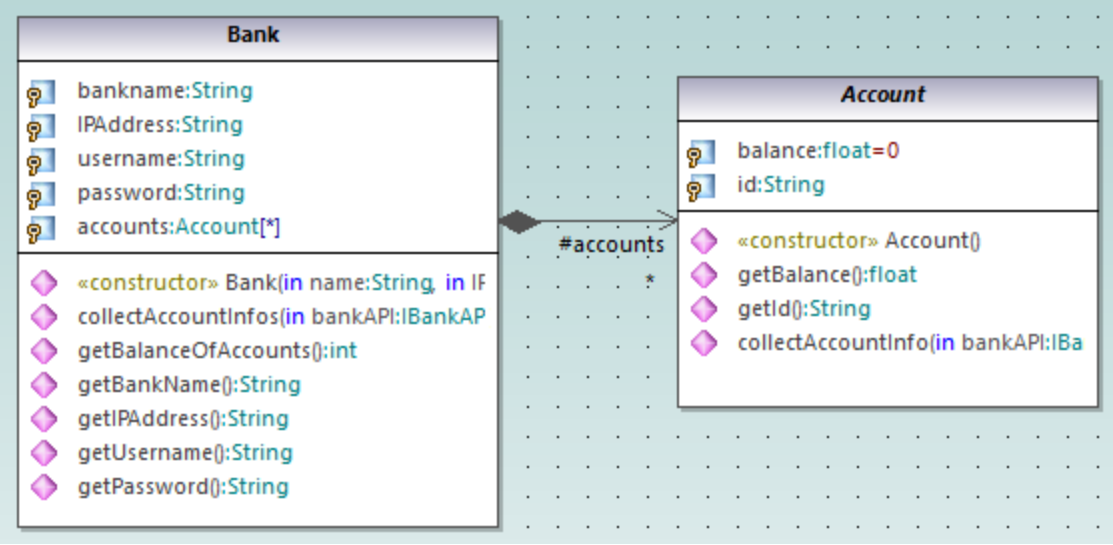


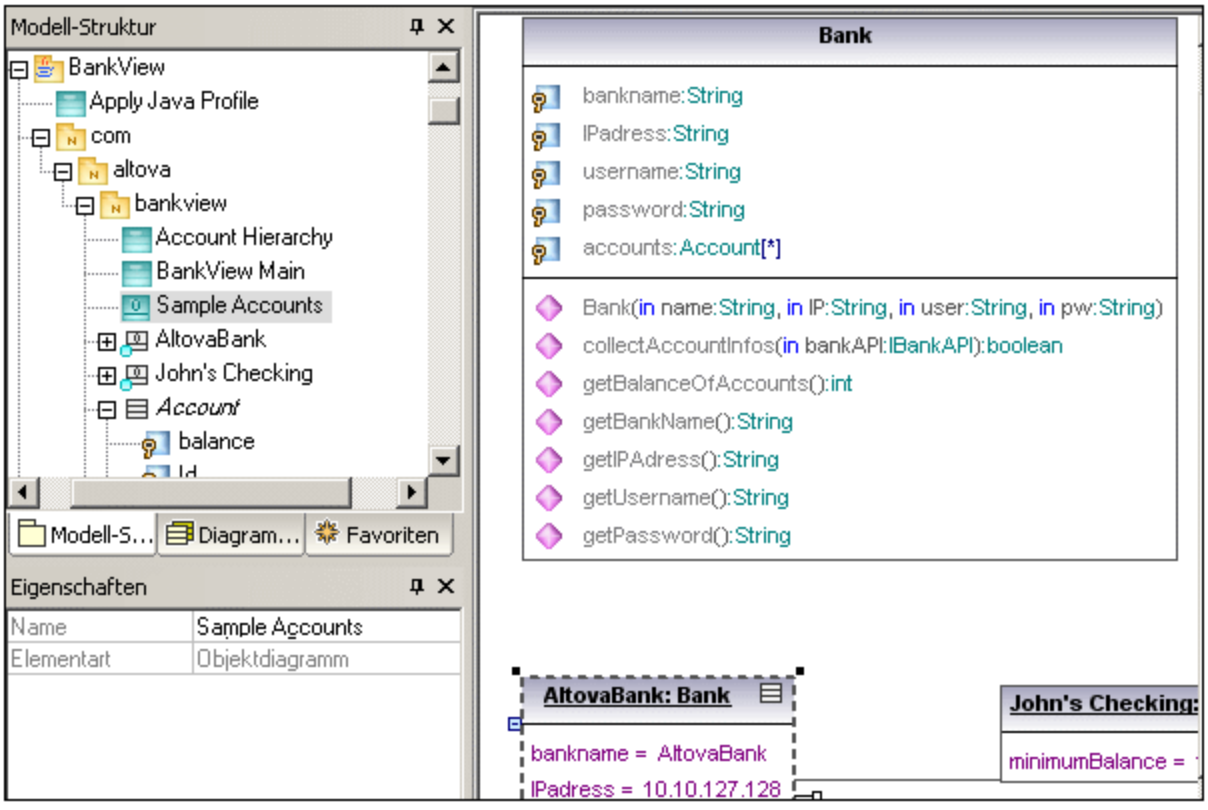
Diagramm "Sample Accounts"

 © 2016-2022 Altova GmbH

In diesem Objektdiagramm sind sowohl Klassen als auch Instanzen davon (Objekte) kombiniert. `AltovaBank:Bank` ist das Objekt/die Instanz der Klasse `Bank`, während `John's checking: CheckingAccount` eine Instanz der (noch nicht zum Diagramm hinzugefügten) Klasse `CheckingAccount` ist.

Fügen wir nun die fehlende Klasse `Account` zum Diagramm hinzu, indem wir sie aus der **Modell-Struktur** in das Diagramm ziehen. Beachten Sie, dass die Kompositions-Assoziation zwischen `Bank` und `Account` automatisch angezeigt wird (diese Assoziation wurde in einem der vorhergehenden Tutorial-Abschnitte definiert, siehe Klassendiagramme 31).





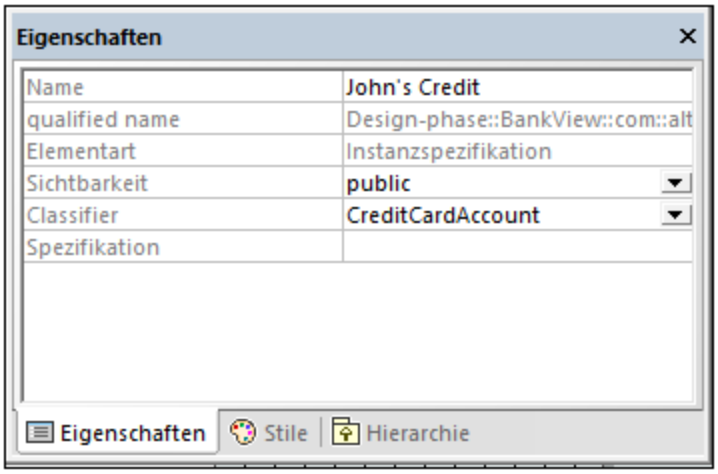
Hinzufügen eines neuen Objekts/einer neuen Instanz (Methode 1)

Wir werden nun ein neues Objekt zum Diagramm namens `John's Credit` hinzufügen. Dieses Objekt instantiiert die Klasse `CreditCardAccount`.

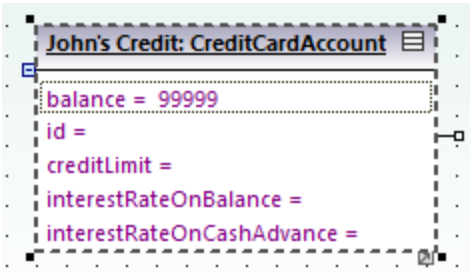
1. Klicken Sie in der Symbolleiste auf das **Instanzspezifikation**-Symbol  und klicken Sie anschließend in das Diagramm unterhalb des Objekts `John's Checking: Checking Account`.
2. Ändern Sie den Namen der Instanz in `John's Credit` und drücken Sie die **Eingabetaste**.



3. Wählen Sie die Instanz aus, damit ihre Eigenschaften im Fenster **Eigenschaften** angezeigt werden.
4. Wählen Sie im Fenster Eigenschaften neben "Classifier" den Eintrag **CreditCardAccount** aus der Dropdown-Liste aus.



Die Instanz hat sich nun geändert und zeigt alle Eigenschaften der Klasse an. Doppelklicken Sie auf eine beliebige Eigenschaft, um einen Wert einzugeben, z.B.:



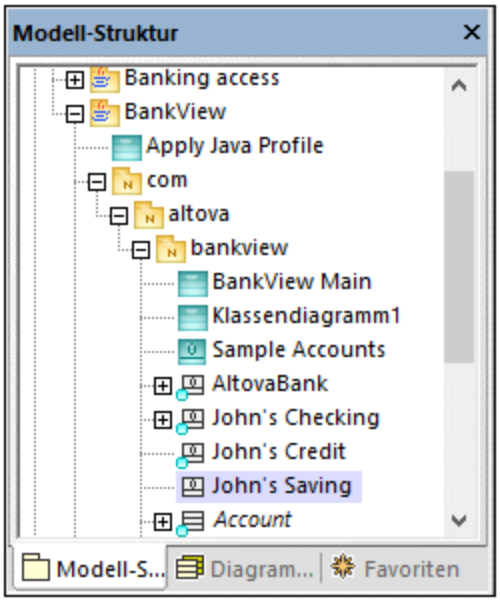
© 2016-2022 Altova GmbH

Um bestimmte Knoten ein- oder auszublenden, klicken Sie mit der rechten Maustaste auf die Instanz und wählen Sie im Kontextmenü den Befehl **Knoteninhalt ein-/ausblenden (Strg+Umschalt+H)**.

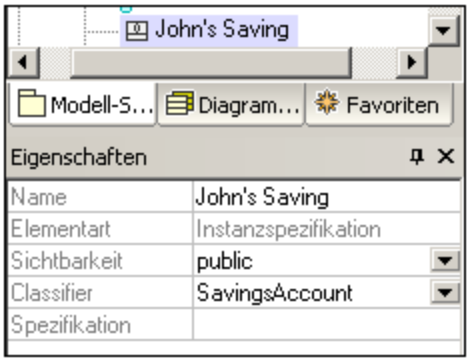
Hinzufügen eines neuen Objekts/einer neuen Instanz (Methode 2)

Wir werden nun eine neue Instanz der Klasse `SavingsAccount` hinzufügen, diesmal mit einer anderen Methode:

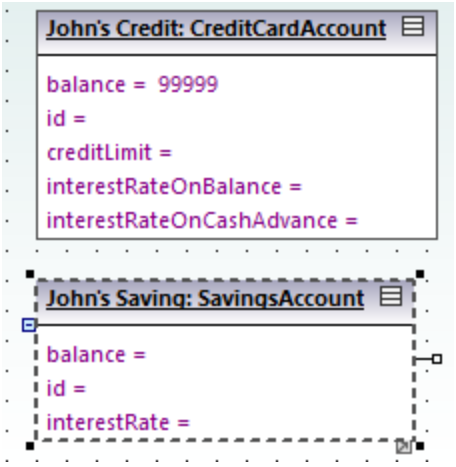
1. Rechtsklicken Sie im Fenster **Modell-Struktur** auf das `bankview`-Paket und wählen Sie **Neues Element | Instanzspezifikation**.
2. Benennen Sie die neue Instanz in `John's Saving` um und drücken Sie zum Bestätigen die Eingabetaste. Das neue Objekt wird zum Paket hinzugefügt und entsprechend gereiht.



3. Wählen, Sie während das Objekt weiterhin im Fenster **Modell-Struktur** ausgewählt ist, im Fenster **Eigenschaften** neben "Classifier" **SavingsAccount** aus.



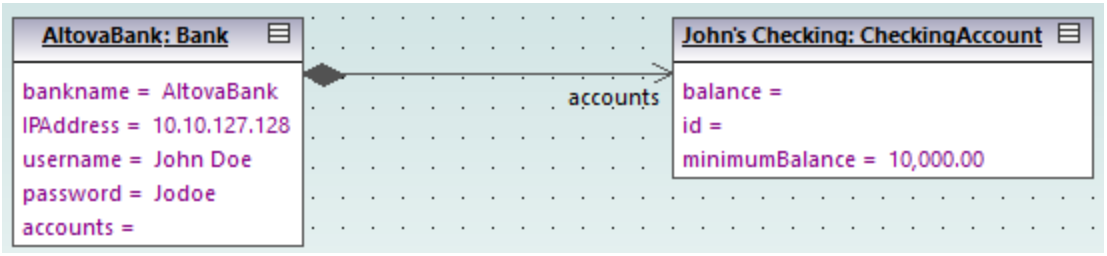
4. Ziehen Sie das Objekt `John's Saving` aus dem Fenster **Modell-Struktur** in das Diagramm und platzieren Sie es unterhalb von `John's credit`.



Erstellen von "Objektbeziehungen" zwischen Objekten

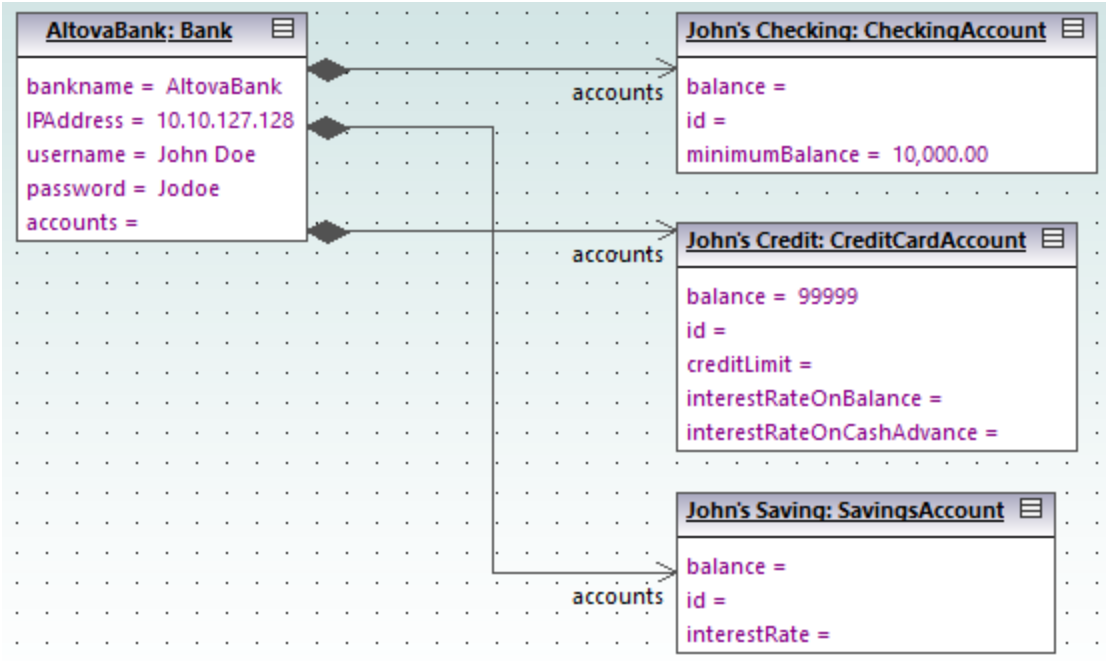
Objektbeziehungen sind die Instanzen von Klassenassoziationen und beschreiben die Assoziationen zwischen Objekten/Instanzen zu einem bestimmten Zeitpunkt.

1. Klicken Sie auf die bestehende Objektbeziehung (Assoziation) zwischen dem Objekt `AltovaBank` und dem Objekt `John's Checking: Checking Account`.
2. Wählen Sie im Fenster **Eigenschaften** neben "Classifier" den Eintrag **Account - Bank** aus. Die Objektbeziehung ändert sich nun gemäß den Klassendefinitionen in eine Kompositions-Assoziation.



3. Klicken Sie in der Symbolleiste auf das **Instanzspezifikation**-Symbol  und positionieren Sie den Cursor über das Objekt `John's Credit: CreditAccount`. Der Cursor wird nun als +-Symbol angezeigt.
4. Ziehen Sie den Cursor vom Objekt `John's Credit: CreditAccount` zu `AltovaBank: Bank`, um die beiden Objekte zu verknüpfen.
5. Wählen Sie im Fenster **Eigenschaften** neben "Classifier" den Eintrag **Account - Bank** aus.
5. Erstellen Sie, wie oben beschrieben, eine Objektbeziehung zwischen dem Objekt `AltovaBank: Bank` und dem Objekt `John's Saving: SavingsAccount`.

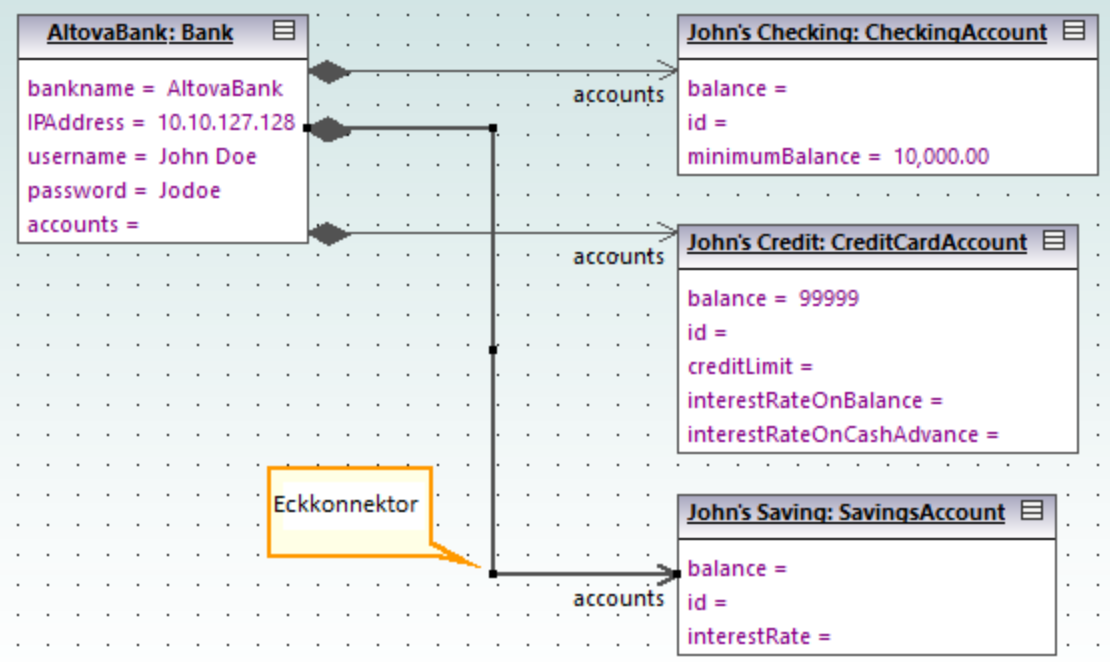
© 2016-2022 Altova GmbH



Beachten Sie, dass Änderungen, die in einem Klassendiagramm am Assoziationstyp vorgenommen werden, automatisch im Objektdiagramm aktualisiert werden.

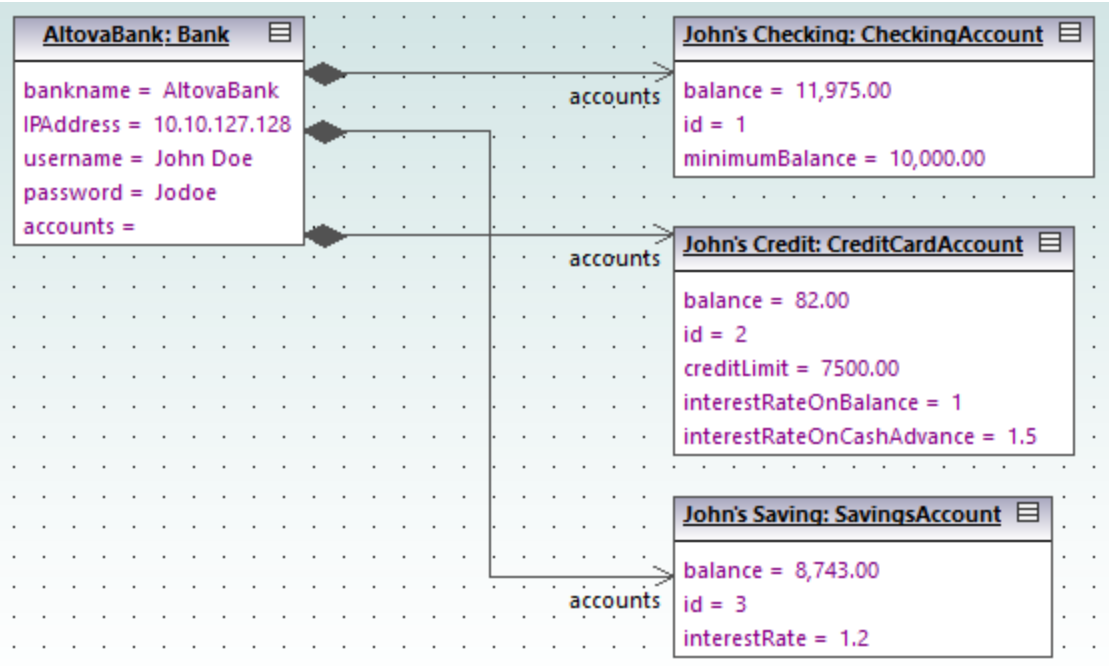
Formatieren von Assoziations-/Objektbeziehungslinien in einem Diagramm

Um Objektbeziehungen zwischen Objekten zu formatieren, platzieren Sie den Cursor auf die Linie und ziehen Sie sie an die gewünschte Position. Um die Linie horizontal und vertikal zu verschieben, ziehen Sie den Eckkonnektor, wie unten gezeigt.



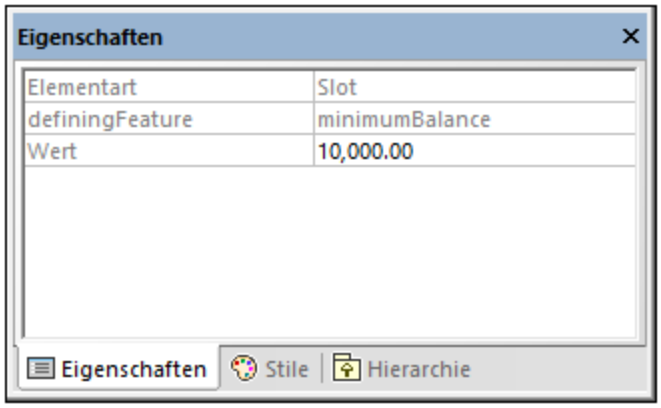
Eingabe von Beispieldaten in Objekte

Der Instanzwert eines Attributs/einer Eigenschaft in einem Objekt wird *Slot* genannt. Um den Zustand eines Objekts zu beschreiben, doppelklicken Sie auf die Slots und geben Sie nach dem Zeichen "=" Beispielinstanzdaten ein, z.B.:



© 2016-2022 Altova GmbH

Objekt-Slots können auch über das Fenster Eigenschaften befüllt werden, indem Sie das Objekt auswählen und den entsprechenden Text neben "Wert" eingeben, z.B.:



2.5 Komponentendiagramme

In diesem Tutorialabschnitt werden die folgenden Aufgaben erläutert:

- Erstellen von Realisierungsbeziehungen zwischen Klassen und Komponenten
- Ändern der Darstellung von im Diagramm verwendeten Linien
- Hinzufügen von Verwendungsbeziehungen zu einer Schnittstelle
- Verwendung der "Ball-and-Socket"-Schnittstellen-Notation

Um fortzufahren, starten Sie UModel und öffnen Sie das Projekt **BankView-start.ump** (siehe auch Öffnen des Tutorial-Projekts 19). Das Projekt enthält eine Reihe von vordefinierten Objektdiagrammen, anhand derer die oben beschriebenen Aufgaben demonstriert werden. Es wird davon ausgegangen, dass Sie den Tutorial-Abschnitt Erstellen von abgeleiteten Klassen 40 bereits durchgearbeitet haben und die Klasse `SavingsAccount` bereits erstellt haben.

Erstellen von Realisierungsbeziehungen zwischen Klassen und Komponenten

Erweitern Sie im Fenster **Diagramm-Struktur** den Eintrag "Komponentendiagramme" und doppelklicken Sie auf das "BankView realization"-Diagrammsymbol. Dieses Diagramm enthält bereits die Komponente `BankView` sowie eine Reihe von Klassen, die über Abhängigkeiten vom Typ "Komponentenrealisierung" damit verbunden sind. Der Text "von bankview" innerhalb der einzelnen Klassen gibt den Namen des Pakets an, zu dem die Klasse gehört.

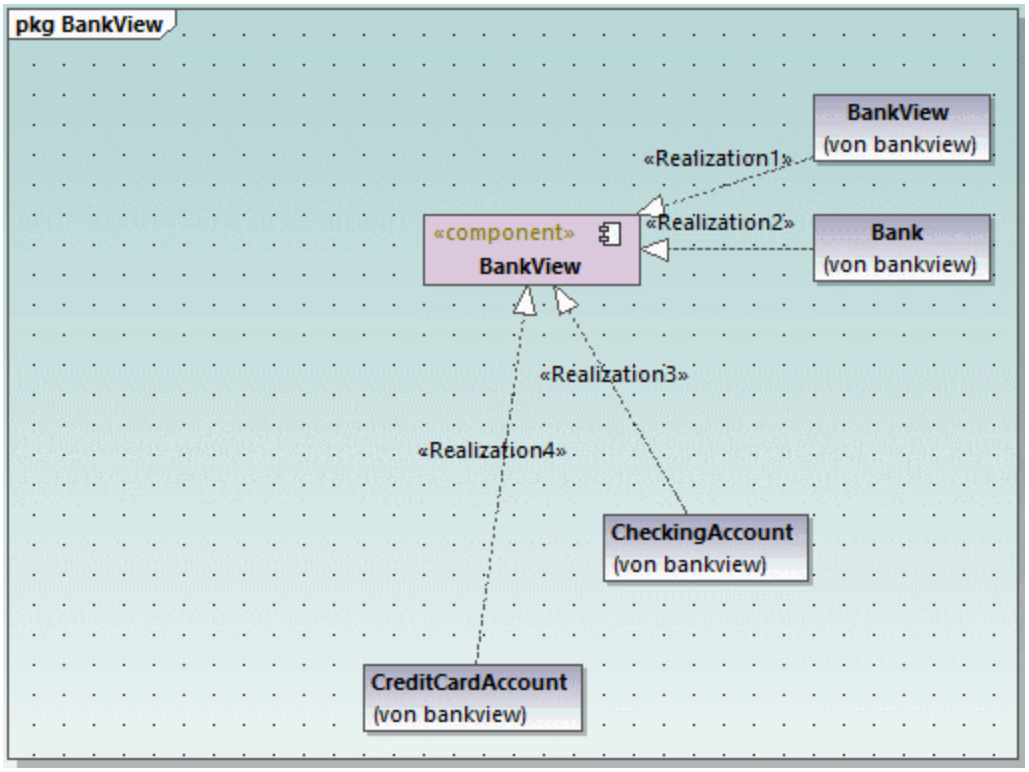


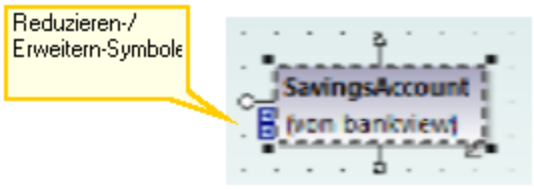
Diagramm "Bank View realization"

Fügen wir nun eine neue Klasse zum Diagramm sowie eine Realisierungsbeziehung zwischen der neuen Klasse und der Komponente `BankView` hinzu.

 © 2016-2022 Altova GmbH

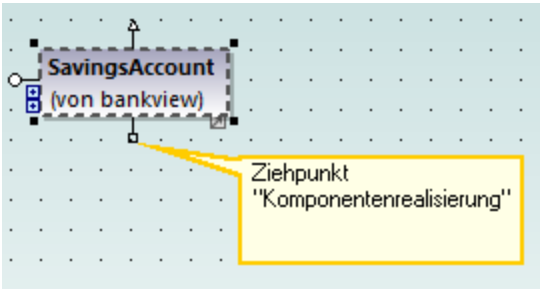
1. Gehen Sie im Fenster **Modell-Struktur** zur Klasse `SavingsAccount` im Paket `bankview`. Falls diese Klasse fehlt, gehen Sie vor, wie im Tutorial-Abschnitt Erstellen von abgeleiteten Klassen 40 beschrieben, um diese zuerst zu erstellen.
2. Ziehen Sie die Klasse `SavingsAccount` aus der **Modell-Struktur** in das Diagramm.

Standardmäßig wird die Klasse so angezeigt, dass alle Bereiche erweitert sind. Klicken Sie auf die Erweitern/Reduzieren-Symbole links von der Klasse, um Eigenschaften und Operationen ein- oder auszublenden.

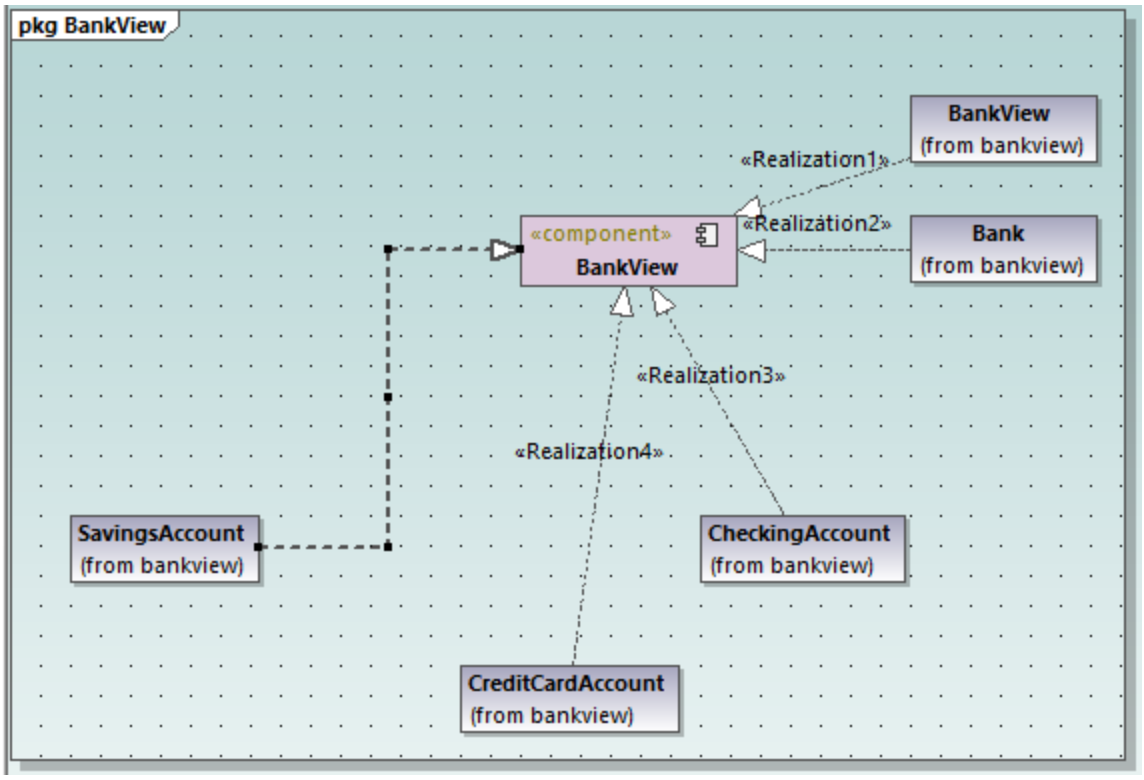


Wählen Sie eine der folgenden Methoden, um eine Realisierungsbeziehung zwischen der Klasse und der Komponente zu erstellen:

- Klicken Sie auf die Symbolleisten-Schaltfläche **Komponentenrealisierung**  und ziehen Sie die Maus von der Klasse `SavingsAccount` zur Komponente `BankView`.
- Bewegen Sie den Cursor über den Ziehpunkt "Komponentenrealisierung" der Klasse und ziehen Sie ihn auf die Komponente **BankView**.



Die Realisierungsbeziehung zwischen `SavingsAccount` und `BankView` wurde nun erstellt.



Um der neuen Abhängigkeitslinie einen Namen zu geben (z.B., "Realization5"), wählen Sie zuerst die Linie aus und geben Sie dann den Namen direkt ein. Alternativ dazu können Sie die Linie auswählen und dann die Eigenschaft **Name** im Fenster **Eigenschaften** bearbeiten.

Ändern der Darstellung von Diagrammlinien

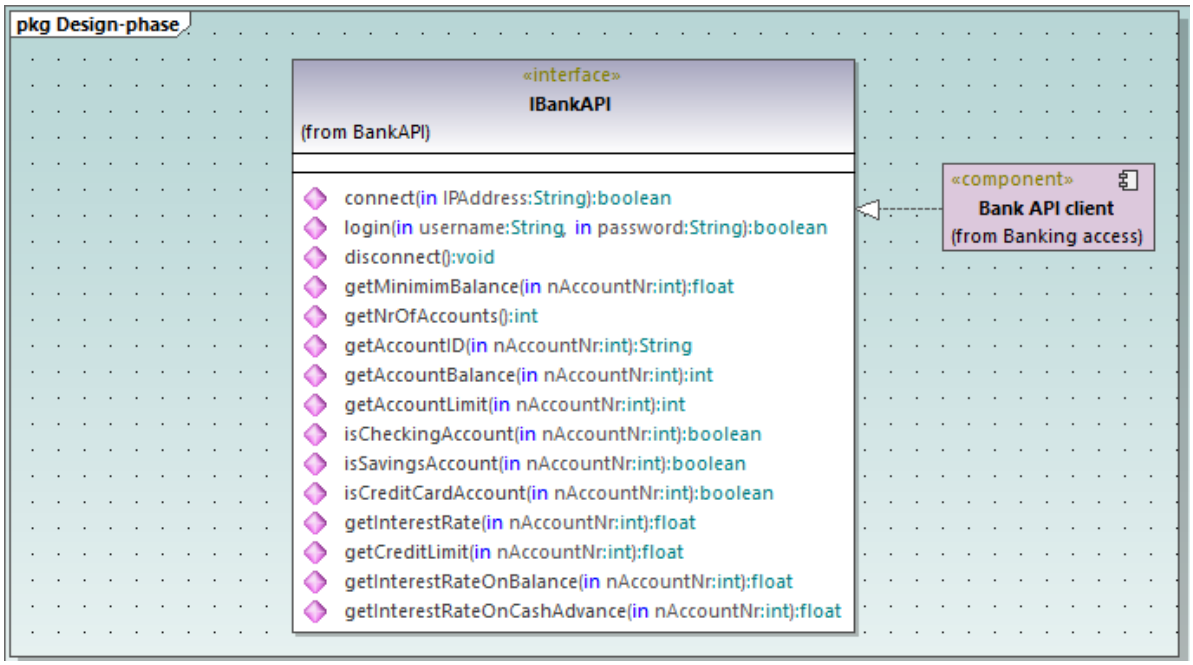
Wir werden nun die Liniendarstellung folgendermaßen von "gebogen" in "direkte Linie" ändern:

1. Wählen Sie die zuvor erstellte Linie aus (d.h. die Linie zwischen `SavingsAccount` und `BankView`).
2. Klicken Sie auf die Symbolleisten-Schaltfläche **Gerade Linie** .

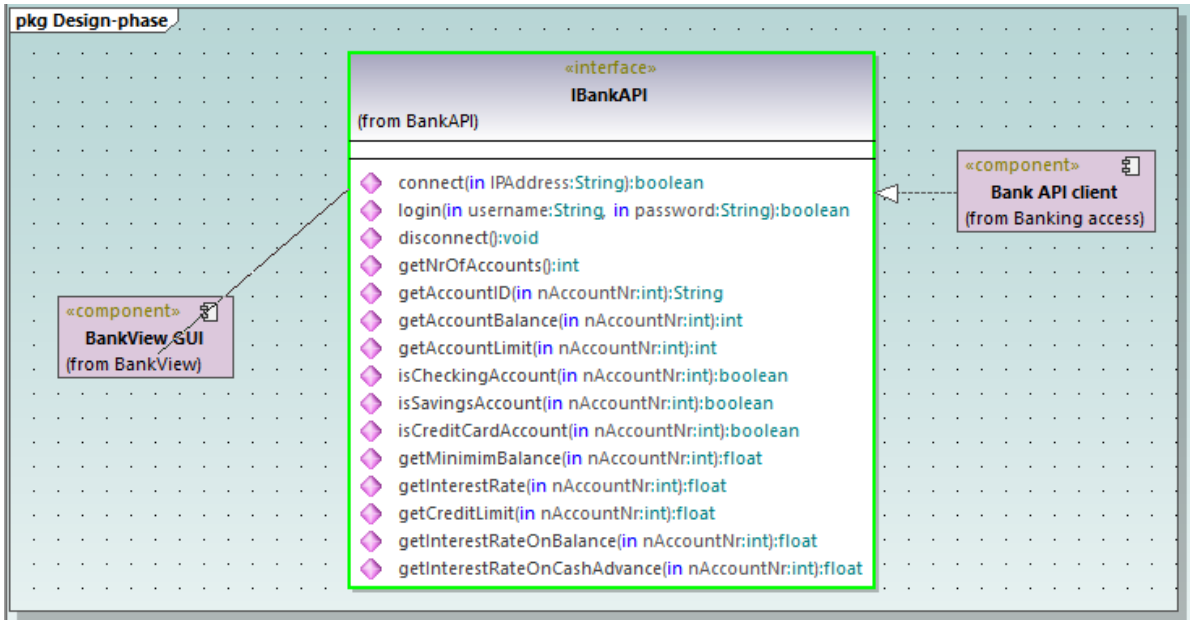
Hinzufügen von Verwendungsbeziehungen zu einer Schnittstelle

1. Navigieren Sie im Fenster **Modell-Struktur** zu **Root | Design-phase** und doppelklicken Sie auf das Symbol neben dem Diagramm "Overview". Daraufhin wird das Komponentendiagramm "Overview" geöffnet. Die gerade definierten Systemabhängigkeiten zwischen Komponenten und Schnittstellen werden darin angezeigt.

© 2016-2022 Altova GmbH

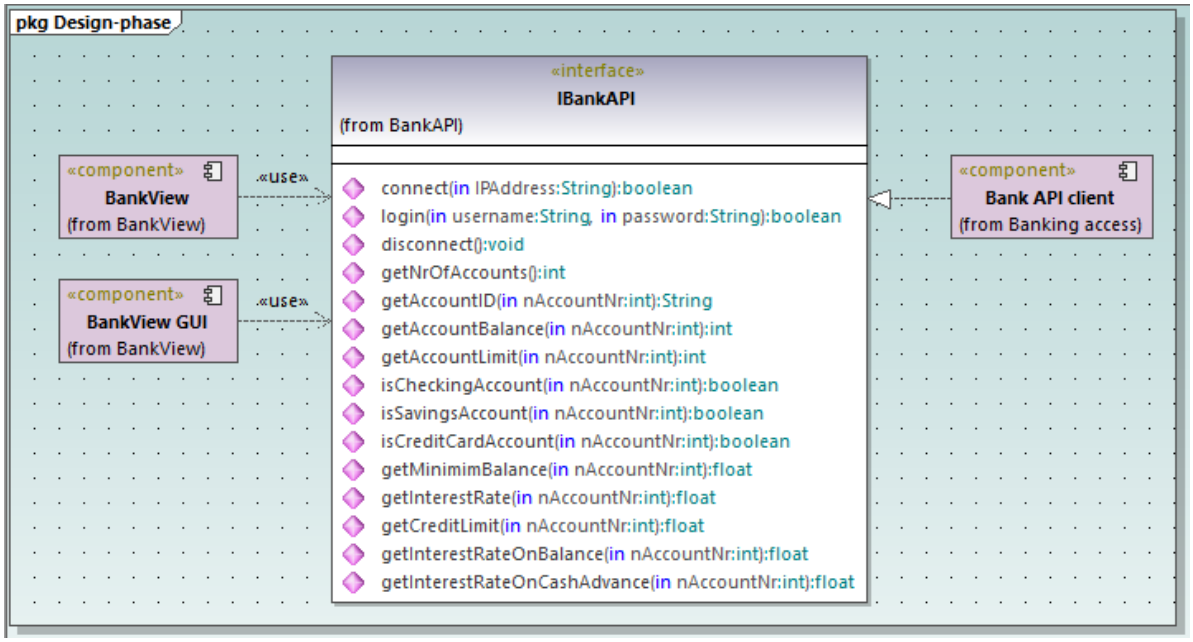


2. Navigieren Sie im Fenster **Modell-Struktur** zu **Root | Component View | BankView** und ziehen Sie das Paket `BankView GUI` in das Diagramm.
3. Ziehen Sie auch das Paket `BankView` in das Diagramm.
4. Klicken Sie auf die Symbolleisten-Schaltfläche **Verwendung**  und ziehen Sie die Maus vom Paket `BankView GUI` zur Schnittstelle `IBankAPI`.



5. Wiederholen Sie den vorhergehenden Schritt für das Paket `BankView`.

Wie unten gezeigt, haben beide Pakete nun eine Verwendungsbeziehung zur Schnittstelle. Für die Pakete `BankView` und `BankView GUI` wird die Schnittstelle `IBankAPI` benötigt. Sie stellt wie für das Paket `Bank API Client` die Schnittstelle bereit.

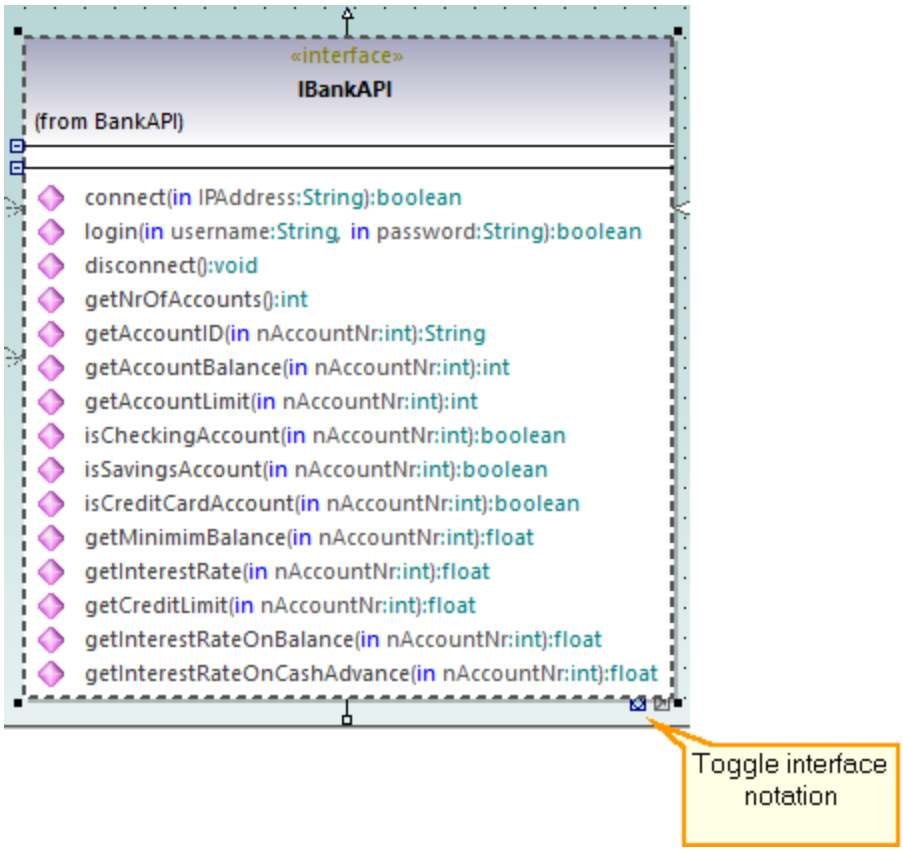


Verwendung der "Ball-and-Socket"-Notation

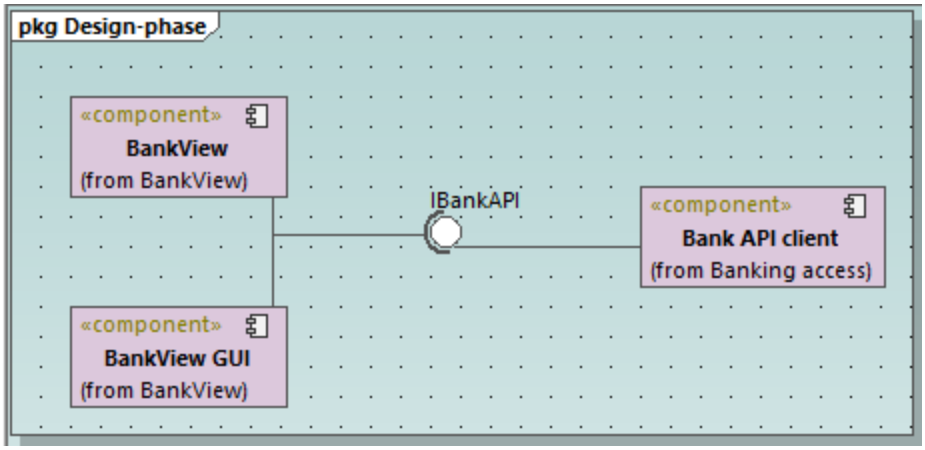
Sie haben optional die Möglichkeit, die aktuelle Diagrammnotation in "Ball-and-Socket"-Notation zu konvertieren. Gehen Sie dazu folgendermaßen vor:

- Wählen Sie die Schnittstelle aus und klicken Sie anschließend in ihrer rechten unteren Ecke auf die Schaltfläche **Darstellung wechseln**.

© 2016-2022 Altova GmbH



Das Diagramm wird nun in der "Ball-and-Socket"-Notation dargestellt.



Um wieder zurück zum vorherigen Notationsstil zu wechseln, wählen Sie die Schnittstelle aus und klicken Sie anschließend wieder auf die Schaltfläche **Darstellung wechseln**.

2.6 Deployment-Diagramme

In diesem Tutorialabschnitt werden die folgenden Aufgaben erläutert:

- Hinzufügen einer Abhängigkeit zwischen zwei Artefakten in einem Deployment-Diagramm
- Hinzufügen von Elementen zu einem Deployment-Diagramm
- Einbetten von Artefakten in einen Knoten in einem Deployment-Diagramm
- Erstellen von Artefakt-Elementen (z.B. Eigenschaften, Operationen, geschachtelten Artefakten)

Um fortzufahren, starten Sie UModel und öffnen Sie das Projekt **BankView-start.ump** (siehe auch Öffnen des Tutorial-Projekts 19).

Hinzufügen einer Abhängigkeit zwischen zwei Artefakten in einem Deployment-Diagramm

Doppelklicken Sie im Fenster Diagramm-Struktur unter "Deployment-Diagramme" auf das Symbol neben dem Diagramm "Artifacts", um es zu öffnen. Wie unten gezeigt, sehen Sie in diesem Diagramm die Manifestation der Komponenten `Bank API client` und `BankView` zu ihren entsprechenden kompilierten Java .jar-Dateien.

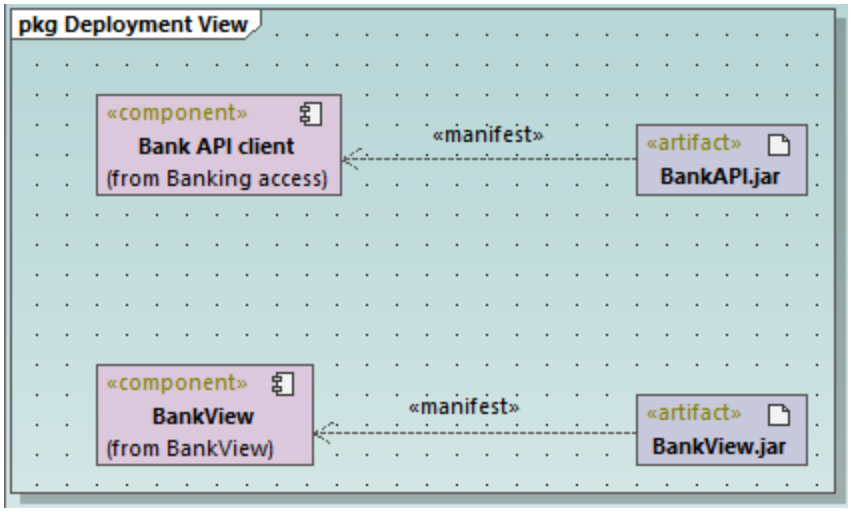


Diagramm "Artifacts"

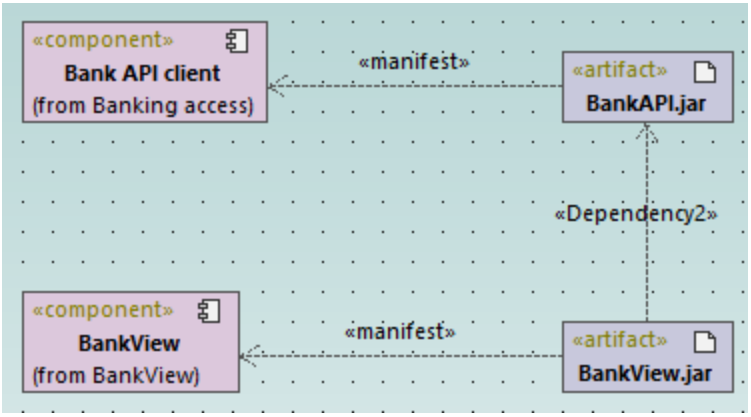
Diese Manifestationen wurden auf ähnliche Weise wie die anderen, zuvor in diesem Tutorial beschriebenen Beziehungen erstellt:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **Manifestation** .
2. Bewegen Sie den Mauszeiger über das Artefakt und ziehen Sie ihn in die Komponente.

Fügen wir nun auf dieselbe Art eine Abhängigkeit zwischen den beiden .jar-Dateien hinzu:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **Abhängigkeit** .
2. Bewegen Sie den Mauszeiger über das Artefakt `BankView.jar` und ziehen Sie ihn in das Artefakt `BankAPI.jar`.
3. Wählen Sie die Abhängigkeitslinie aus und geben Sie "Dependency2" ein.

© 2016-2022 Altova GmbH



Hinzufügen von Elementen zu einem Deployment-Diagramm

Doppelklicken Sie im Fenster **Diagramm-Struktur** unter "Deployment-Diagramme" auf das Symbol neben dem Diagramm "Deployment", um es zu öffnen. Dieses Diagramm wurde absichtlich unfertig gelassen und besteht aus einem einzigen Knoten, der einen Heim-PC darstellt. In den folgenden Schritten werden wir weitere Elemente zu diesem Diagramm hinzufügen.

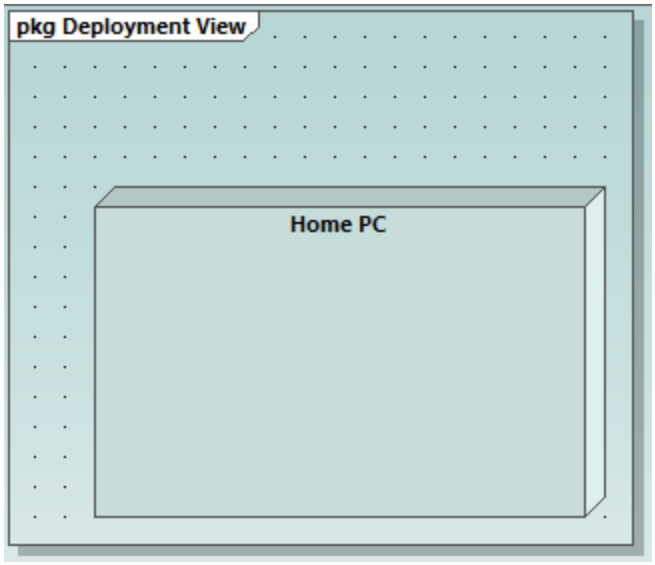
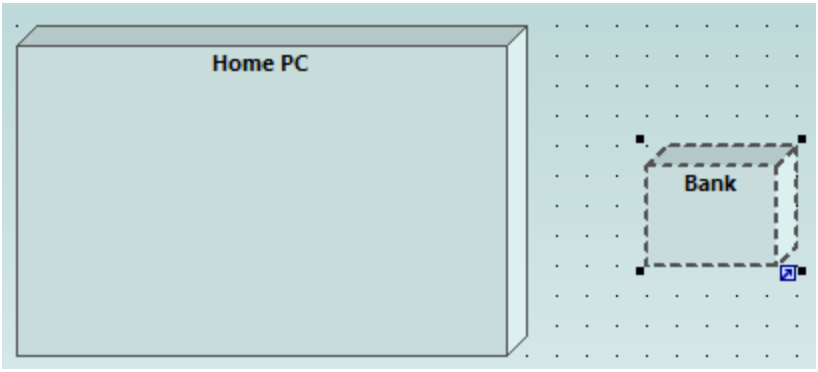


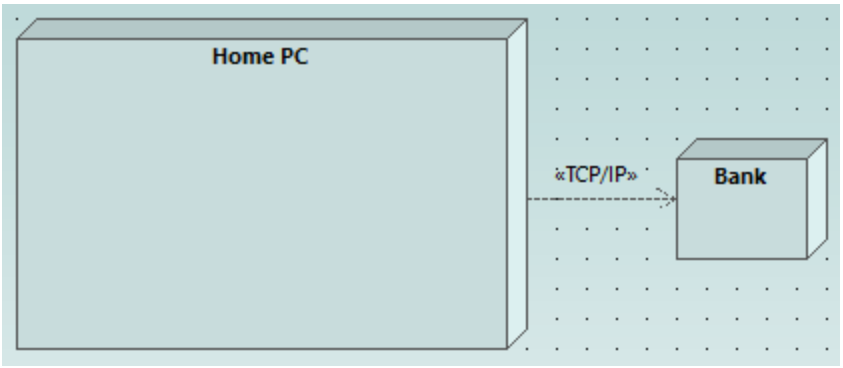
Diagramm "Deployment"

Angenommen, Sie möchten eine TCP/IP-Verbindung zwischen dem Heim-PC und einer Bank darstellen. Fügen wir also nun die erforderlichen Elemente hinzu:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **Knoten**  und klicken Sie an eine Stelle rechts vom Knoten "Home PC", um den Knoten einzufügen.
2. Benennen Sie den Knoten um in "Bank" und vergrößern Sie ihn durch Ziehen einer seiner Kanten.



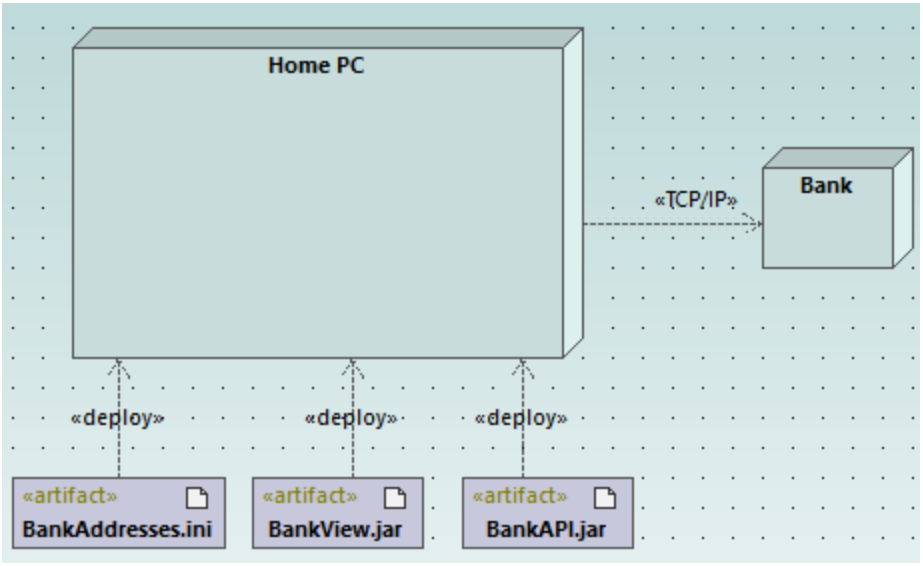
3. Klicken Sie auf die Symbolleisten-Schaltfläche **Abhängigkeit** und ziehen Sie die Abhängigkeit vom Knoten "Home PC" zum Knoten "Bank". Dadurch wird eine Abhängigkeit zwischen den beiden Knoten erstellt.
4. Wählen Sie die Abhängigkeitslinie aus und geben Sie als Namen der neuen Abhängigkeit "TCP/IP" ein. (Bearbeiten Sie alternativ dazu die Eigenschaft **Name** im Fenster **Eigenschaften**).



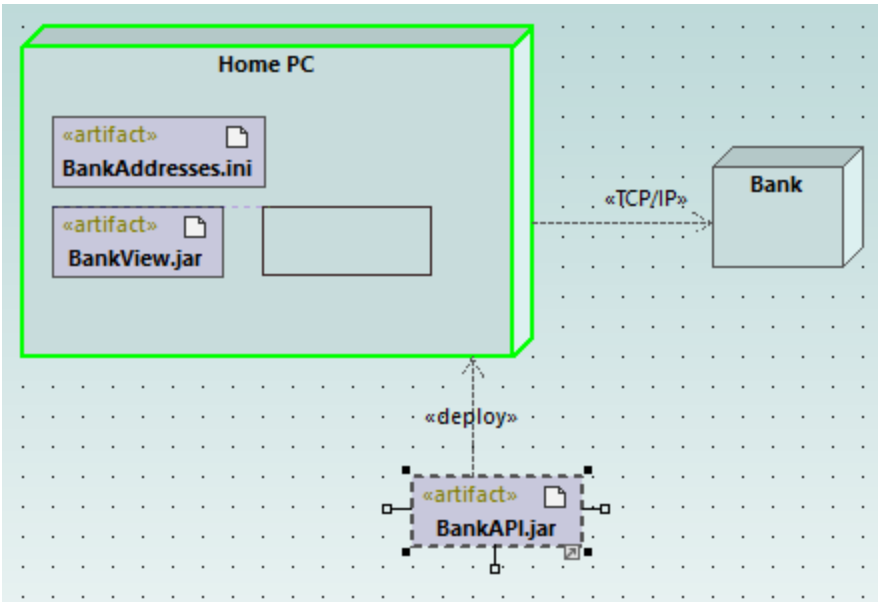
Einbetten von Artefakten

Erweitern Sie im Fenster **Modell-Struktur** das Paket "Deployment View" und ziehen Sie anschließend alle der folgenden Artefakte in das Diagramm: **BankAddresses.ini**, **BankAPI.jar** und **BankView.jar**. Das Projekt wurde so vorkonfiguriert, dass es Deploy-Abhängigkeiten zwischen diesen Artefakten und dem Knoten "HomePC" enthält, daher werden alle diese Abhängigkeiten im Diagramm nun angezeigt:

© 2016-2022 Altova GmbH

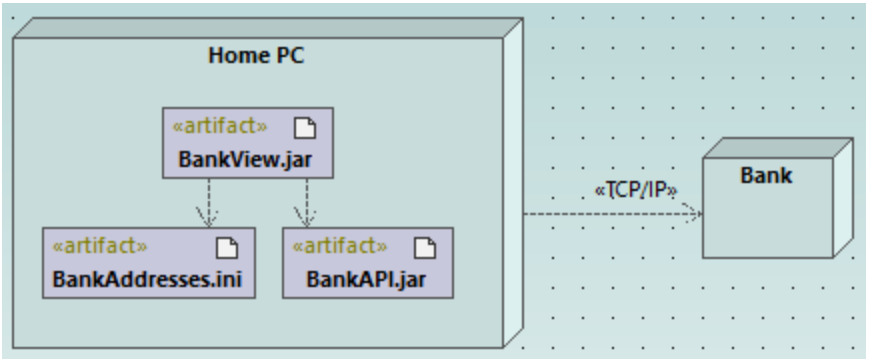


Sie können die Artefakte auch durch Ziehen der einzelnen Artefakte in den Knoten "Home PC" einbetten. Beachten Sie, dass die Deploy-Abhängigkeiten im Diagramm nicht mehr sichtbar sind, obwohl sie logisch weiterhin bestehen.



In den Knote eingebettete Artefakte können auch untereinander Abhängigkeiten aufweisen. Tun Sie zur Veranschaulichung Folgendes:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **Abhängigkeit** und ziehen Sie die Maus bei gedrückter Strg-Taste vom Artefakt "BankView.jar" in das Artefakt "BankAddresses.ini".
2. Ziehen Sie die Maus vom Artefakt "BankView.jar" in das Artefakt "BankAPI.jar", während Sie die Strg-Taste gedrückt halten.



Anmerkung: Wenn Sie ein Artefakt aus einem Knoten in ein Diagramm ziehen, wird immer automatisch eine Deployment-Abhängigkeit erstellt.

Erstellen von Artefaktelementen (Eigenschaften, Operationen, geschachtelten Artefakten)

Artefakte können in UML aus Eigenschaften, Operationen und anderen Elementen, wie z.B. geschachtelten Artefakten, bestehen. Um solche geschachtelten Elemente zu erstellen, klicken Sie mit der rechten Maustaste im Fenster **Modell-Struktur** auf das Artefakt und wählen Sie im Kontextmenü die entsprechende Aktion aus (z.B. **Neues Element | Operation** oder **Neues Element | Eigenschaft**). Das neue Element erscheint im Fenster **Modell-Struktur** geschachtelt unterhalb des ausgewählten Artefakts.

© 2016-2022 Altova GmbH

2.7 Forward Engineering (Modell zu Code)

In diesem Beispiel wird gezeigt, wie Sie ein neues UModel-Projekt erstellen und Programmcode anhand dieses Projekts generieren (ein Prozess, der als "Forward Engineering" bezeichnet wird). Der Einfachheit halber besteht das Projekt aus nur einer Klasse. Außerdem lernen Sie, wie Sie das Projekt für die Codegenerierung vorbereiten und sicher stellen, dass im Projekt die richtige Syntax verwendet wird. Nachdem Sie Programmcode generiert haben, werden Sie diesen außerhalb von UModel ändern, indem Sie eine neue Methode zur Klasse hinzufügen. Zum Abschluss werden Sie die Codeänderungen wieder im ursprünglichen UModel-Projekt zusammenführen (ein als "Reverse Engineering" bekannter Prozess).

In diesem Tutorial wird als Codegenerierungssprache Java verwendet. Die Vorgangsweise ist aber auch bei anderen Codegenerierungssprachen ähnlich.

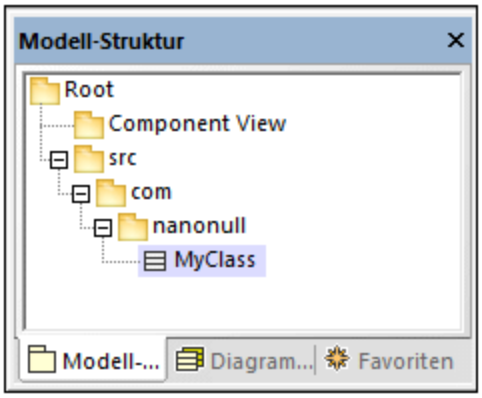
Erstellen eines neue UModel-Projekts

Sie können folgendermaßen ein neues UModel-Projekt erstellen:

- Klicken Sie im Menü **Datei** auf **Neu**. (Oder drücken Sie alternativ dazu **Strg+N** oder klicken Sie auf die Symbolleisten-Schaltfläche "Neu" .)

Das Projekt enthält zu diesem Zeitpunkt nur die Standardpakete "Root" und "Component View". Diese beiden Pakete können nicht gelöscht oder umbenannt werden. "Root" bildet die oberste Hierarchieebene für alle anderen Pakete und Elemente im Projekt. "Component View" wird für das Code Engineering benötigt; normalerweise enthält es eine oder mehrere UML-Komponenten, die von den Klassen oder Schnittstellen Ihres Projekts realisiert werden; wir haben zu diesem Zeitpunkt jedoch noch keine Klassen erstellt. Erstellen wir also zuerst die Struktur unseres Programms. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf das Paket "Root" und wählen Sie im Kontextmenü den Befehl **Neues Element | Paket**. Benennen Sie das neue Paket in "src" um.
2. Klicken Sie mit der rechten Maustaste auf "src" und wählen Sie im Kontextmenü den Befehl **Neues Element | Paket.** Benennen Sie das neue Paket in "com" um.
3. Klicken Sie mit der rechten Maustaste auf "com" wählen Sie im Kontextmenü den Befehl **Neues Element | Paket**. Benennen Sie das neue Paket in "nanonull" um.
4. Klicken Sie mit der rechten Maustaste auf "nanonull" wählen Sie im Kontextmenü den Befehl **Neues Element | Klasse**. Benennen Sie die neue Klasse in "MyClass" um.



Vorbereiten des Projekts für die Codegenerierung

Um Code anhand eines UModel-Modells zu generieren, müssen die folgenden Voraussetzungen gegeben sein:

- Es muss ein C#- oder VB.NET-Namespace-Root-Pakte definiert sein.
- Es muss eine Komponente geben, die von allen Klassen oder Schnittstellen, für die Code generiert werden muss, realisiert wird.
- Der Komponente muss ein physischer Pfad (Verzeichnis) zugewiesen worden sein. Der Code wird in diesem Verzeichnis generiert.
- Für die Komponente muss die Eigenschaft **für Code Engineering verwenden** aktiviert sein.

Alle diese Anforderungen werden weiter unten ausführlicher beschrieben. Beachten Sie, dass Sie jederzeit überprüfen können, ob das Projekt alle Voraussetzungen für die Codegenerierung erfüllt, indem Sie es validieren:

- Klicken Sie im Menü **Projekt** auf **Projektsyntax überprüfen**. (Drücken Sie alternativ dazu **F11**.)

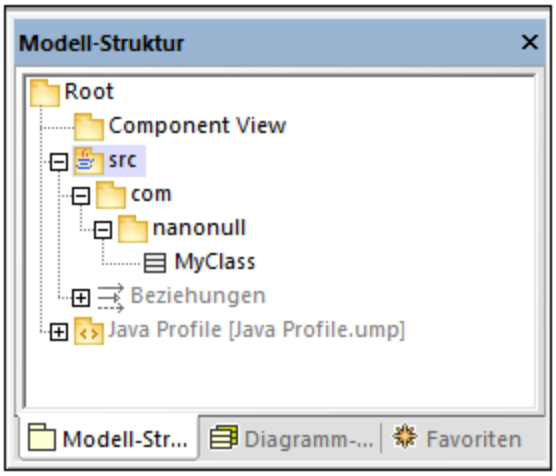
Wenn Sie das Projekt in dieser Phase validieren, wird im Fenster "Meldungen" ein Validierungsfehler angezeigt ("*Es wurde keine Namespace Root gefunden! Verwenden Sie bitte das Kontextmenü in der Modell-Struktur, um ein Paket als Namespace Root zu definieren*"). Um diesen Fehler zu beheben, weisen Sie das Paket "src" als Namespace Root zu:

- Klicken Sie mit der rechten Maustaste auf das Paket "src" und wählen Sie im Kontextmenü den Befehl **Code Engineering | Als Java Namespace Root definieren**.
- Wenn Sie informiert werden, dass das UModel Java-Profil inkludiert wird, klicken Sie auf **OK**.



Beachten Sie, dass sich das Paketsymbol nun in das Symbol geändert hat, was bedeutet, dass es sich beim Paket um eine Java Namespace Root handelt. Zusätzlich dazu wurde ein Java-Profil zum Projekt hinzugefügt.

© 2016-2022 Altova GmbH



Der eigentliche Namespace kann folgendermaßen definiert werden:

1. Wählen Sie das Paket "com" im Fenster **Modell-Struktur** aus.
2. Aktivieren Sie im Fenster **Eigenschaften** die Eigenschaft **<<namespace>>**.

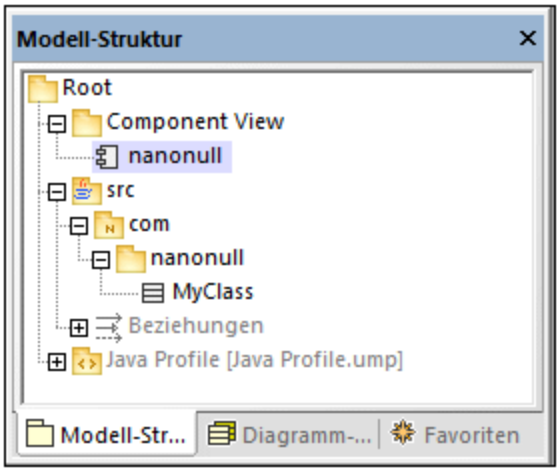


3. Wiederholen Sie die obigen Schritte für das Paket "nanonull".

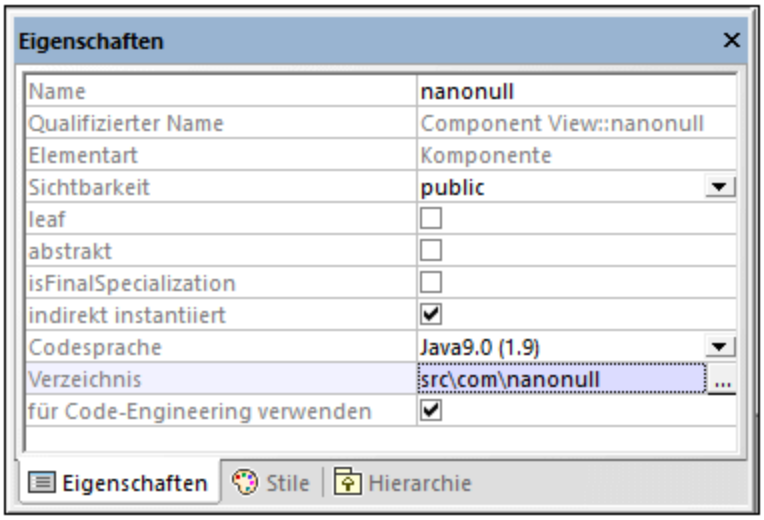
Beachten Sie, dass sich das Symbol der beiden Pakete "com" und "nanonull" nun in geändert hat, was bedeutet, dass es sich nun um einen Namespace handelt.

Eine weitere Voraussetzung für die Codegenerierung ist, dass eine Komponente von mindestens einer Klasse oder Schnittstelle realisiert werden muss. Eine Komponente ist in UML ein Teilstück des Systems. In UModel können Sie über die Komponente das Verzeichnis für die Codegenerierung und andere Einstellungen definieren. Andernfalls wäre die Codegenerierung nicht möglich. Wenn Sie das Projekt zu diesem Zeitpunkt validieren, wird im Fenster **Meldungen** eine Warnung angezeigt: "*MyClass hat keine Komponentenrealisierung zu einer Komponente - es wird kein Code generiert*". Um diesen Fehler zu beheben, muss eine Komponente zum Projekt hinzugefügt werden. Gehen Sie folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf "Component View" und wählen Sie im Kontextmenü den Befehl **Neues Element | Komponente**.
2. Benennen Sie die neue Komponente in "nanonull" um.



3. Ändern Sie im Fenster **Eigenschaften** die Eigenschaft **Verzeichnis** in ein Verzeichnis, in dem der Code generiert werden soll (in diesem Beispiel, "src\com\nanonull"). Beachten Sie dass die Eigenschaft **für Code Engineering verwenden** aktiviert ist, was eine weitere Vorbedingung für die Codegenerierung ist.



4. Speichern Sie das UModel-Projekt in einem Verzeichnis und geben Sie ihm einen beschreibenden Namen (in diesem Beispiel **C:\UModelDemo\Tutorial.ump**).

Anmerkung: Der Pfad für die Codegenerierung kann absolut oder relativ zum .ump-Projekt sein. Wenn er, wie in diesem Beispiel, relativ ist, würden mit dem Pfad **src\com\nanonull** alle Verzeichnisse im selben Verzeichnis, in dem auch das UModel-Projekt gespeichert ist, erstellt werden.

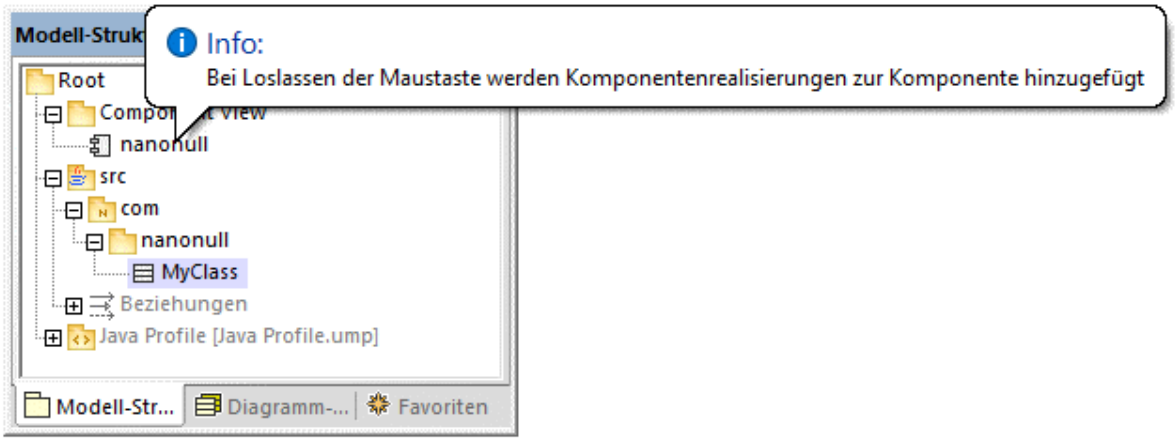
Wir haben uns absichtlich entschieden, Code in einem Verzeichnis zu generieren, das den Namespace-Namen enthält; andernfalls würde es zu Warnungen kommen. UModel zeigt standardmäßig Projektvalidierungswarnungen an, wenn die Komponente so konfiguriert ist, das Java-Code in einem Verzeichnis generiert wird, das nicht denselben Namen wie der Namespace hat. Die Komponente "nanonull" in diesem Beispiel hat den Pfad "C:\UModelDemo\src\com\nanonull", sodass es zu keinen Validierungswarnungen kommt. Wenn Sie eine ähnliche Überprüfung für C# oder VB.NET implementieren möchten oder wenn Sie die Namespace-Validierung für Java deaktivieren möchten, gehen Sie folgendermaßen vor:

 © 2016-2022 Altova GmbH

1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Code Engineering**.
3. Aktivieren Sie unter **Namespace für Codedateipfad verwenden** das entsprechende Kontrollkästchen.

Die Komponentenrealisierungsbeziehung kann folgendermaßen erstellt werden:

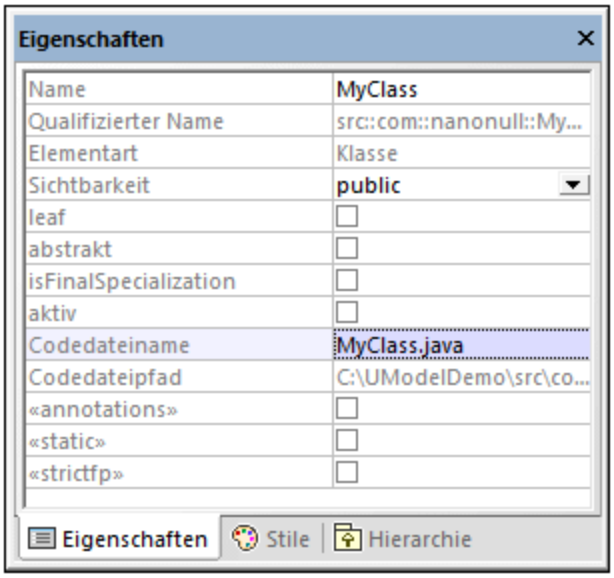
- Ziehen Sie die Maus im Fenster Modell-Struktur bei gedrückter Maustaste von der zuvor erstellten Klasse `MyClass` auf die Komponente `nanonull`.



Die Komponente wird nun von der einzigen Klasse des Projekts, nämlich `MyClass` realisiert. Beachten Sie, dass die oben beschriebene Methode nun eine von mehreren Möglichkeiten ist, die Komponentenrealisierung zu erstellen. Eine weitere Methode ist, sie, wie im Tutorialabschnitt Komponentendiagramme 54 beschrieben, von einem Komponentendiagramm aus zu erstellen.

Als nächstes wird empfohlen, den an der Codegenerierung beteiligten Klassen oder Schnittstellen einen Dateinamen zu geben. Andernfalls generiert UModel die entsprechende Datei mit einem Standarddateinamen und im Fenster **Meldungen** wird eine Warnung angezeigt ("*'Codedateiname wurde nicht definiert - es wird ein Standardname generiert.*"). Um diese Warnung zu entfernen, gehen Sie folgendermaßen vor:

1. Wählen Sie die Klasse `MyClass` im Fenster **Modell-Struktur** aus.
2. Ändern Sie die Eigenschaft **Codedateiname** im Fenster **Eigenschaften** in den gewünschten Datennamen (in diesem Beispiel, `MyClass.java`).

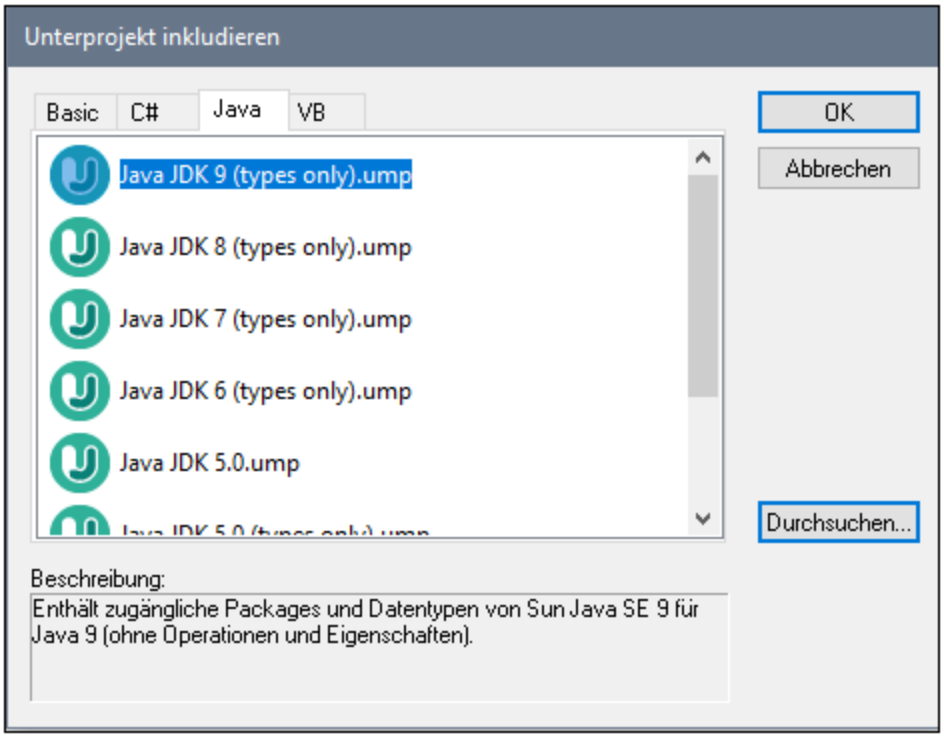


Inkludieren der JDK-Typen

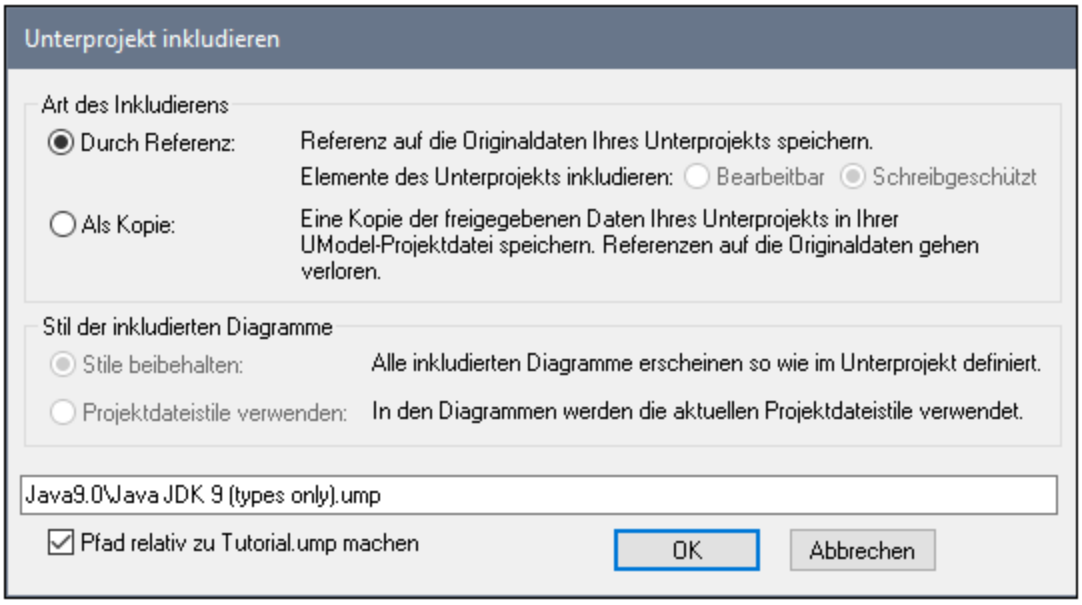
Dieser Schritt ist zwar optional, dennoch wird empfohlen, die Java Development Kit (JDK)-Sprachentypen als Unterprojekt Ihres aktuellen UModel-Projekts zu inkludieren. Andernfalls stehen die JDK-Typen beim Erstellen von Klassen oder Schnittstellen nicht zur Verfügung. Gehen Sie dazu folgendermaßen vor (die Vorgangsweise ist für C#, C++ und VB.NET ähnlich):

1. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.
2. Klicken Sie auf das Register **Java** und wählen Sie das Projekt **Java JDK 9 (types only)** aus.

© 2016-2022 Altova GmbH



3. Wenn Sie gefragt werden, ob das Projekt über eine Referenz oder als Kopie inkludiert werden soll, wählen Sie **Durch Referenz**.

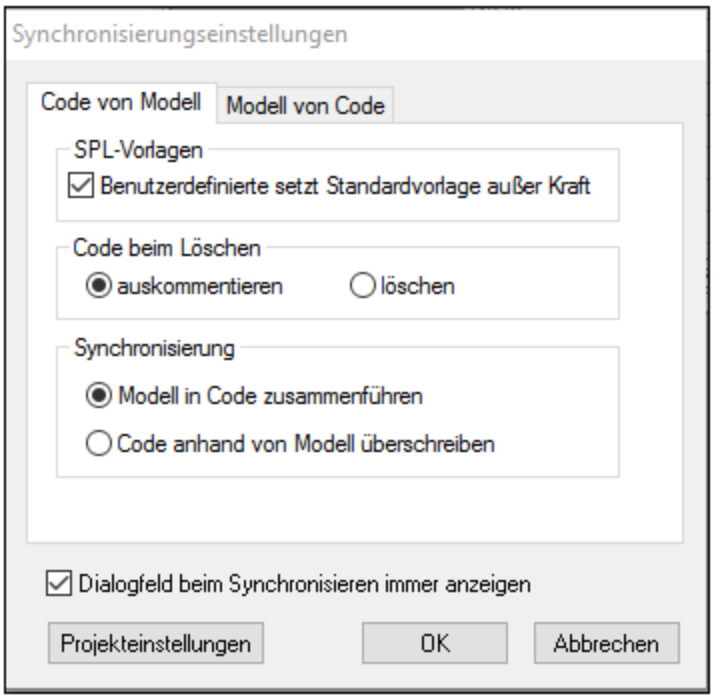


Generieren von Code

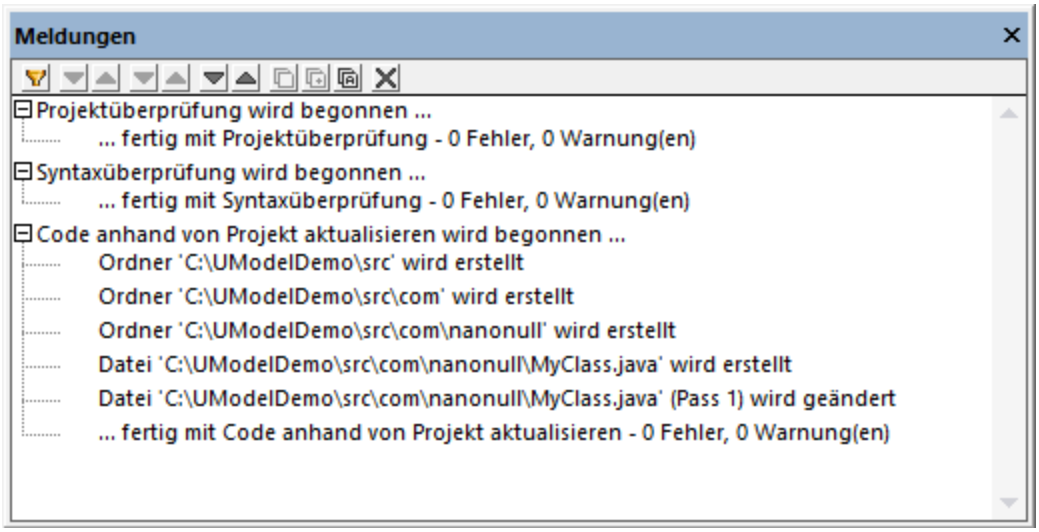
Nachdem nun alle Vorbedingungen erfüllt werden, kann Code folgendermaßen generiert werden:

1. Klicken Sie im Menü **Projekt** auf **Merge Programmcode aus UModel-Projekt**. (Drücken Sie

alternativ dazu **F12**). Beachten Sie, dass dieser Befehl den Namen **Überschreibe Programmcode aus UModel-Projekt** hat, wenn zuvor im unten gezeigten Dialogfeld "Synchronisierungseinstellungen" die Option **Code anhand von Modell überschreiben** aktiviert wurde.



2. Lassen Sie die Standardsynchronisierungseinstellungen unverändert und klicken Sie auf **OK**. Es wird automatisch eine Projektsyntaxüberprüfung durchgeführt, deren Ergebnis im Fenster **Meldungen** angezeigt wird:



Ändern von Code außerhalb von UModel

Die Generierung von Programmcode ist erst der erste Schritt bei der Entwicklung Ihrer Software-Applikation oder Ihres Systems. In einem realen Szenario würde der Code mehrmals verändert, bevor das ein Programm

 © 2016-2022 Altova GmbH

mit allen seinen Features fertig entwickelt ist. Öffnen Sie die generierte Datei **MyClass.java** zu diesem Zweck in einem Text-Editor und fügen Sie eine neue Methode zur Klasse hinzu, wie unten gezeigt. Die Datei **MyClass.java** sollte nun folgendermaßen aussehen:

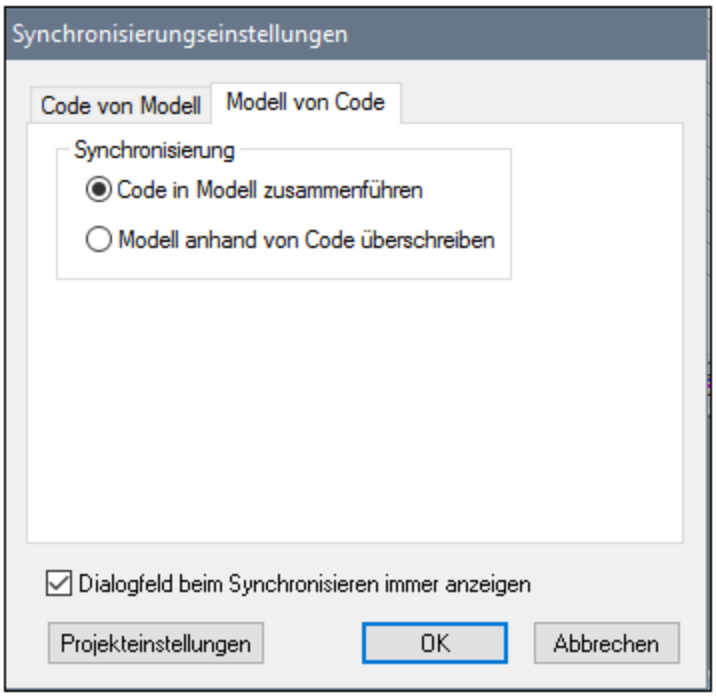
```
package com.nanonull;
public class MyClass{
   public float sum(float num1, float num2){
    return num1 + num2;
   }
}
```

MyClass.java

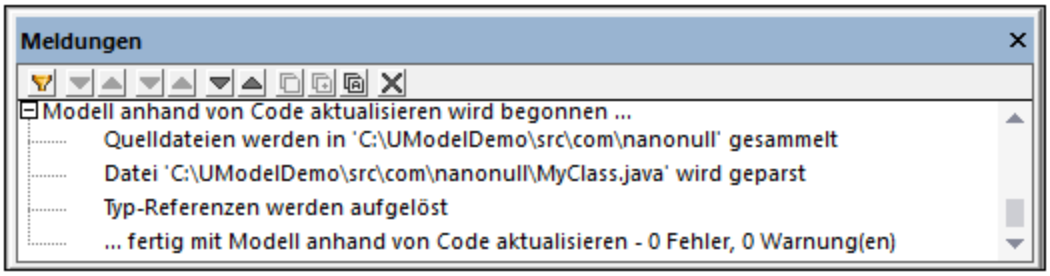
Zusammenführen der Codeänderungen im Modell

Sie können den geänderten Code nun wieder im Modell zusammenführen. Gehen Sie folgendermaßen vor:

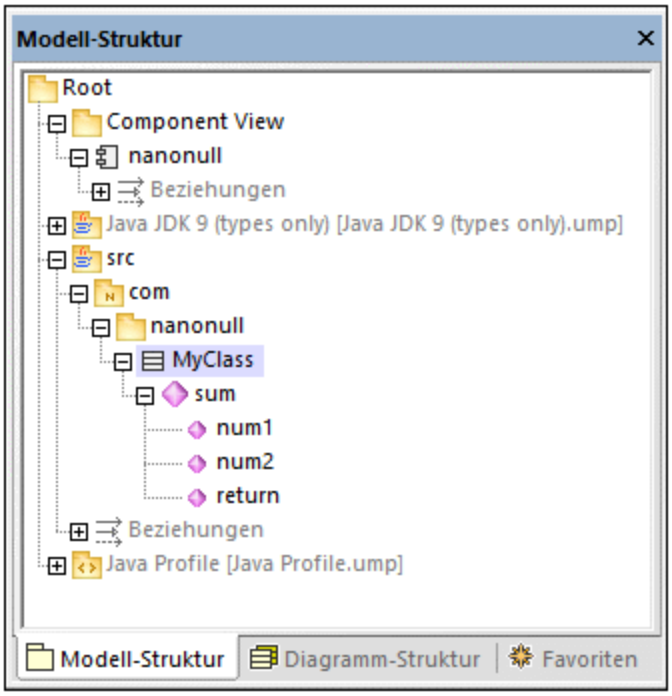
1. Klicken Sie im Menü **Projekt** auf **Merge UModel-Projekt aus Programmcode** (Drücken Sie alternativ dazu **Ctrl + F12**).



2. Lassen Sie die Standardsynchronisierungseinstellungen unverändert und klicken Sie auf **OK**. Es wird automatisch eine Codesyntaxüberprüfung durchgeführt, deren Ergebnis im Fenster **Meldungen** angezeigt wird:



Die Operation `sum` (die mit Reverse Engineering anhand des Codes generiert wurde) wird nun im Fenster "Modell-Struktur" angezeigt.



© 2016-2022 Altova GmbH

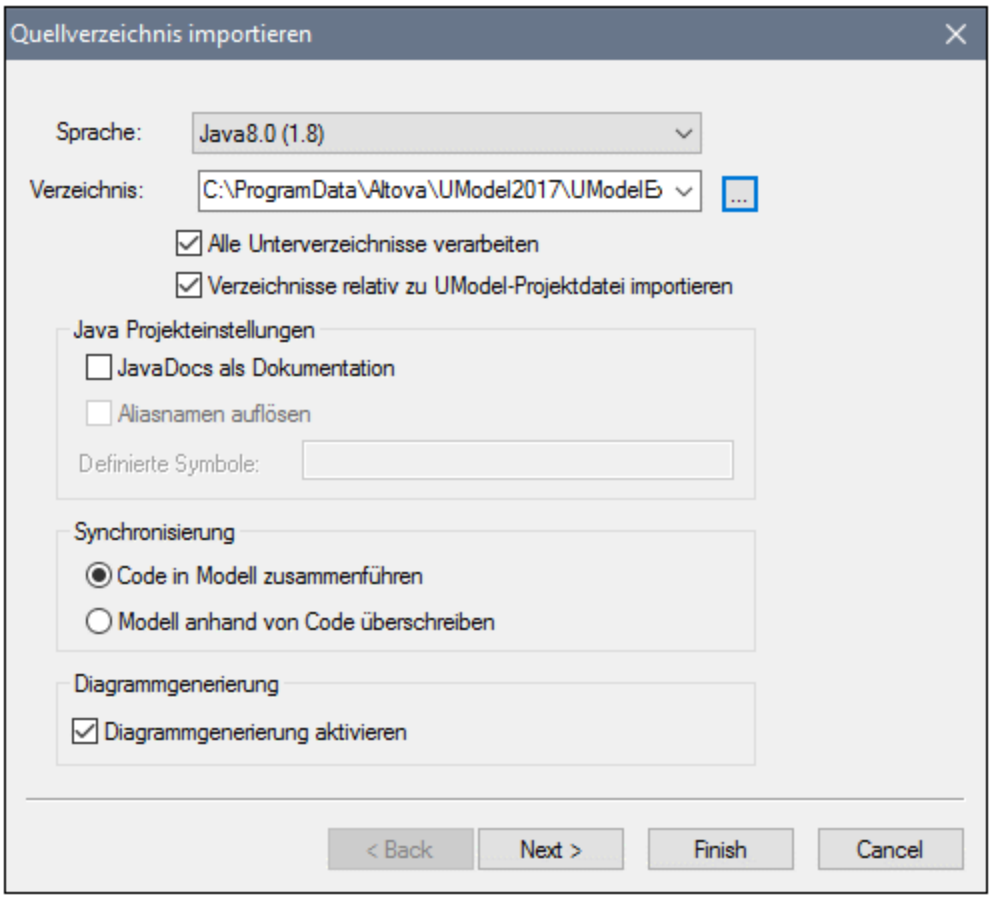
2.8 Reverse Engineering (Code zu Modell)

In diesem Tutorialabschnitt wird gezeigt, wie Sie vorhandenen Programmcode aus einem Verzeichnis in ein neues UModel-Projekt importieren (Reverse Engineering). Außerdem werden Sie im Modell eine neue Klasse hinzufügen, es für die Codegenerierung vorbereiten und die Änderungen anschließend mittels Forward Engineering im Java-Code wieder zusammenführen. Zwar wird in diesem Tutorial der Import von Java-Code beschrieben, doch ist das Verfahren für den Import von C# oder VB.NET-Code ähnlich.

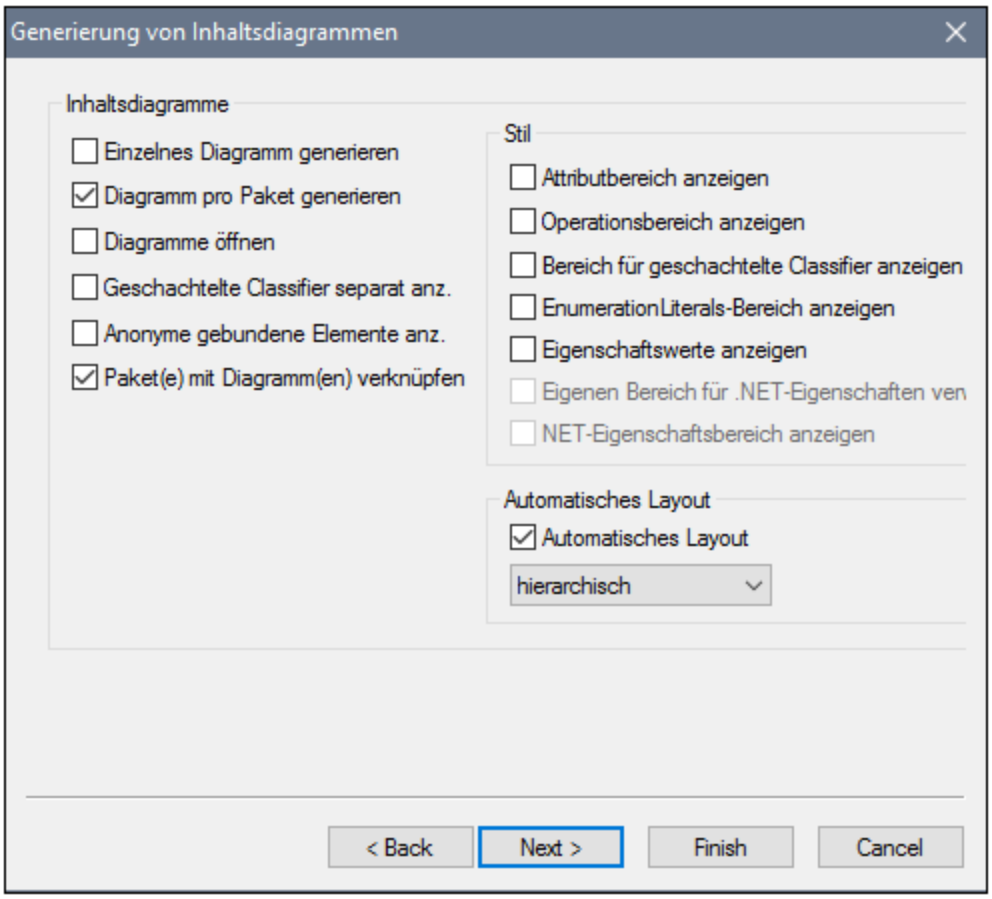
Anmerkung: Der in diesem Tutorial verwendete Java-Beispielcode steht in einem ZIP-Archiv unter dem folgenden Pfad zur Verfügung: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\OrgChart.zip**. Entpacken Sie das Archiv bitte in dasselbe Verzeichnis, bevor Sie mit dem Tutorial beginnen.

Importieren von bestehendem Code aus einem Verzeichnis

1. Klicken Sie im Menü **Datei** auf **Neu**.
2. Klicken Sie im Menü **Projekt** auf **Quellverzeichnis importieren**.
3. Wählen Sie die Quellcodesprache aus (in diesem Beispiel Java).
4. Klicken Sie auf die Durchsuchen-Schaltfläche ... und wählen Sie das zuvor entpackte Verzeichnis **OrgChart** und klicken Sie auf **Weiter**. Beachten Sie, dass das Kontrollkästchen Diagrammgenerierung aktivieren aktiviert ist, damit UModel Klassen[449]- und Paketdiagramme[469] anhand des Quellcodes generiert.

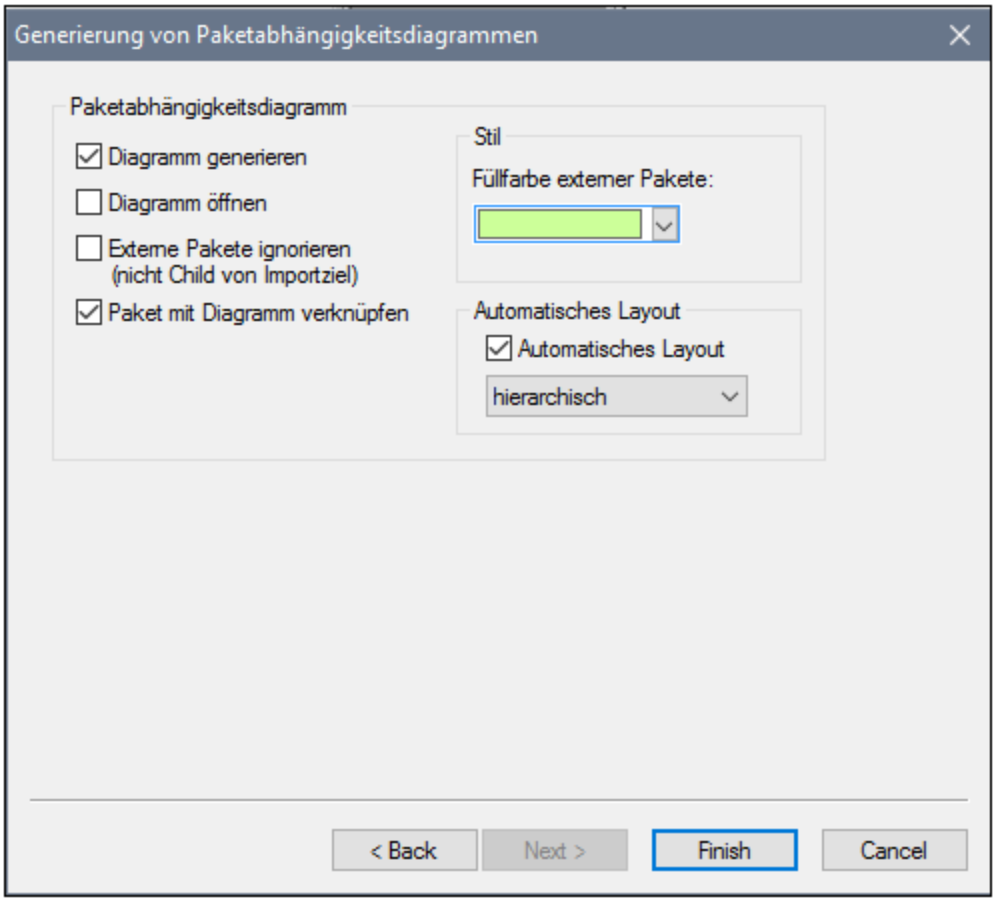


5. Wählen Sie die Option **Diagramm pro Paket generieren,** damit UModel für jedes Paket ein neues Diagramm generiert. Sie können die Diagrammstile später gegebenenfalls ändern.

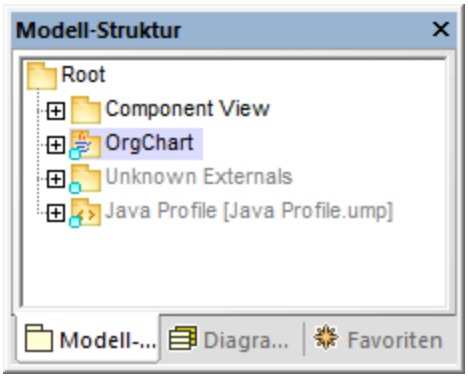


6. Klicken Sie zum Fortfahren auf **Weiter**. In diesem Dialogfeld können Sie die Einstellungen für die Generierung von Paketabhängigkeiten definieren.

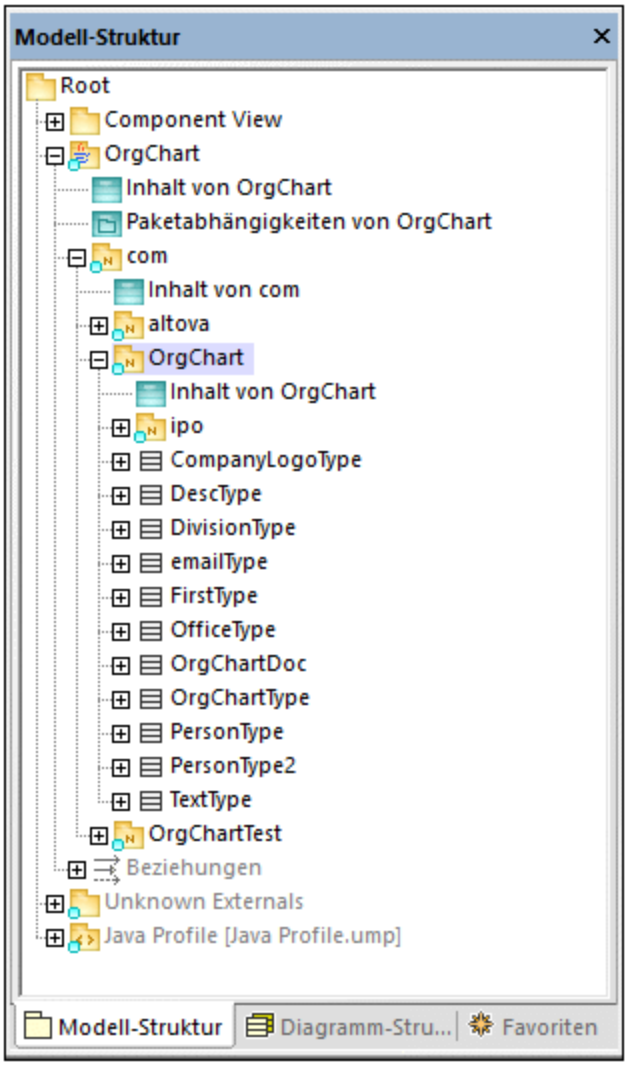
© 2016-2022 Altova GmbH



7. Klicken Sie auf "**Fertig stellen**". Wenn Sie dazu aufgefordert werden, speichern Sie das neue Modell in einem Verzeichnis auf Ihrem Rechner. Die Daten werden während der Eingabe geparst und es wird ein neues Paket namens "**OrgChart**" erstellt.

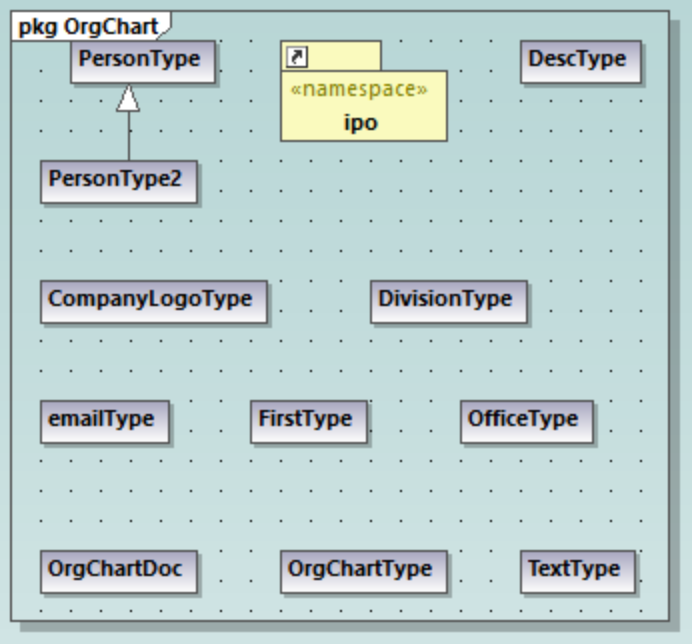


8. Erweitern Sie das neue Paket und dessen Unterpakete bis Sie zum Paket **OrgChart** (**com | OrgChart**) gelangen. Doppelklicken Sie auf das Diagrammsymbol "**Inhalt von OrgChart**":



Das Diagramm "Inhalt von OrgChart" wird nun im Hauptfenster angezeigt.

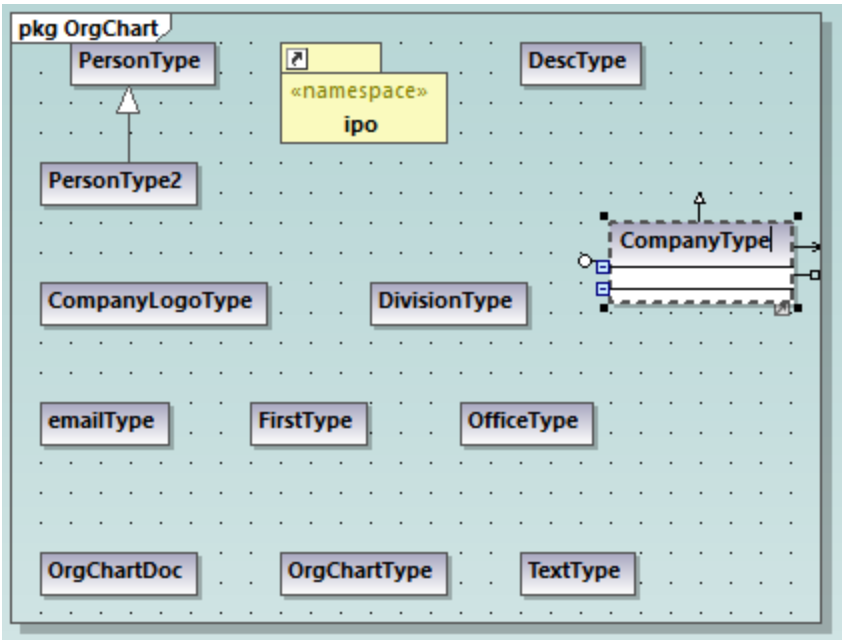
© 2016-2022 Altova GmbH



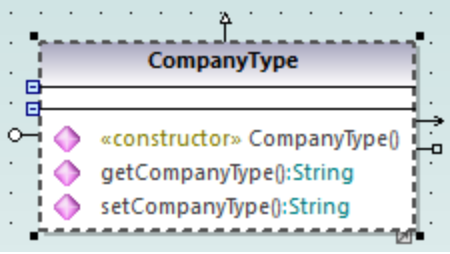
Hinzufügen einer neuen Klasse zum OrgChart-Diagramm:

Sie haben in dieser Phase des Tutorials aus bereits vorhandenem Java-Code mit Reverse Engineering ein Modell erstellt, das auch eine Reihe von automatisch generierten Diagrammen enthält. Wir werden nun einen Schritt weitergehen und das Modell um eine neue Klasse erweitern.

1. Klicken Sie mit der rechten Maustaste in das Diagramm "Inhalt von OrgChart" und wählen Sie im Kontextmenü den Befehl **Neu | Klasse**.
2. Klicken Sie auf die Überschrift der neuen Klasse und geben Sie als Name der neuen Klasse **CompanyType** ein.



3. Fügen Sie mittels der Taste **F8** in diesem Beispiel die folgenden Operationen zur Klasse hinzu: `CompanyType()`, `getCompanyType():String`, `setCompanyType():String`.

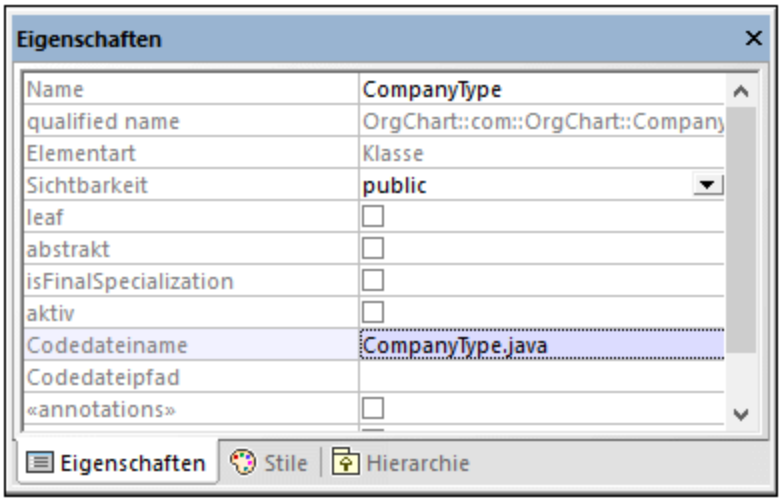


Anmerkung: Da der Klassenname `CompanyType` ist, wird der Operation `CompanyType()` automatisch das Stereotyp `<<constructor>>` zugewiesen.

Verfügbarmachen der neuen Klasse für die Codegenerierung:

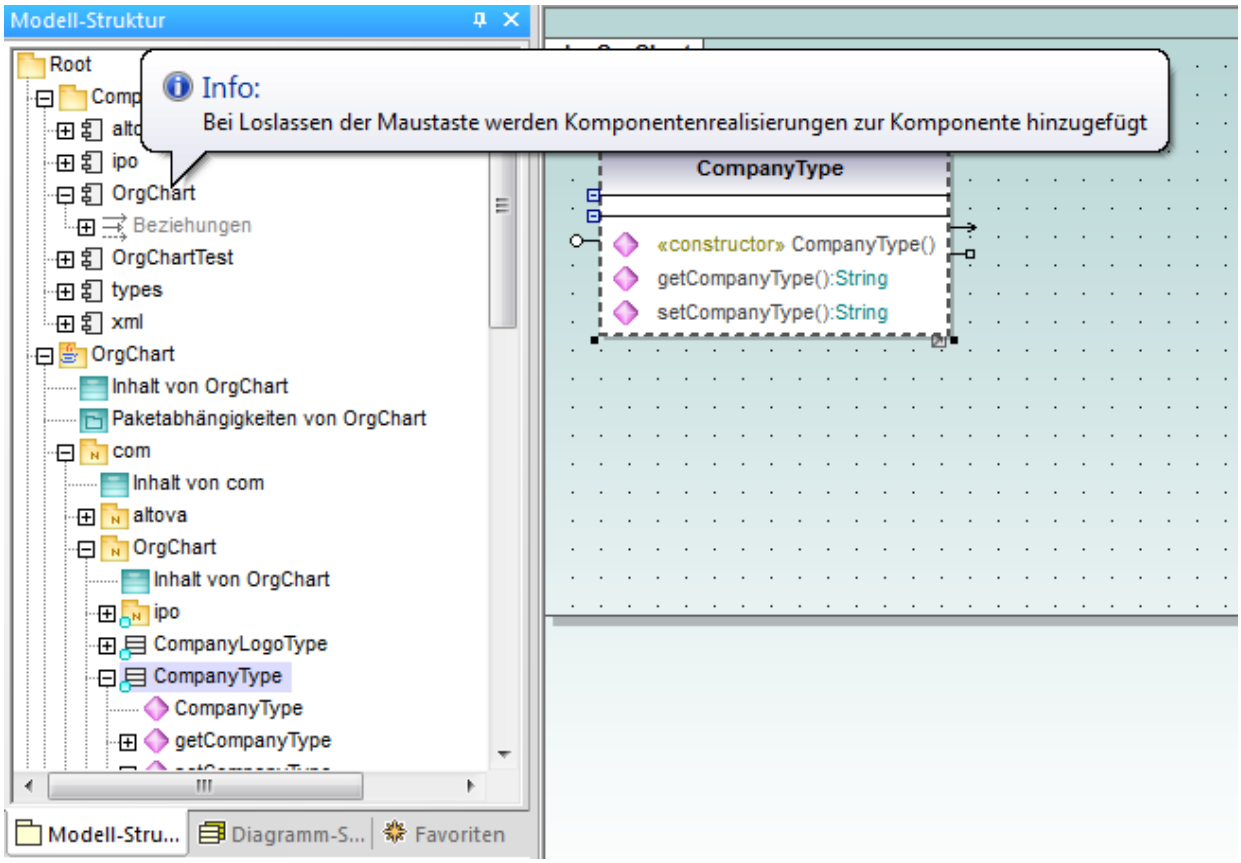
Da das Modell nun durch eine neue Klasse erweitert wurde, möchten Sie den zugrunde liegenden Code wahrscheinlich entsprechend aktualisieren, um Modell und Code zu synchronisieren. Wenn Sie jedoch jetzt **F11** drücken, um die Projektsyntax in dieser Phase des Projekts zu überprüfen, wird im Fenster "Meldungen" eine Warnung angezeigt: *'CompanyType' hat keine Komponentenrealisierung zu einer Komponente - die Komponentenrealisierung zur Komponente 'OrgChart' wird generiert*. Der Grund hierfür ist, dass für die neue Klasse eine Realisierung zu einer Komponente benötigt wird, bevor anhand dieser Klasse Code generiert werden kann (siehe Erklärung in Forward Engineering (Modell zu Code) 65. In einigen Fälle (wie in diesem Beispiel) kann UModel die erforderliche Realisierung automatisch generieren. Sie können die Realisierungsabhängigkeit allerdings auch folgendermaßen manuell definieren:

1. Während die Klasse "CompanyType" aktiv ist, klicken Sie im Fenster "Eigenschaften" in das Feld "Codedateiname" und geben Sie als Dateinamen "CompanyType.java" ein.

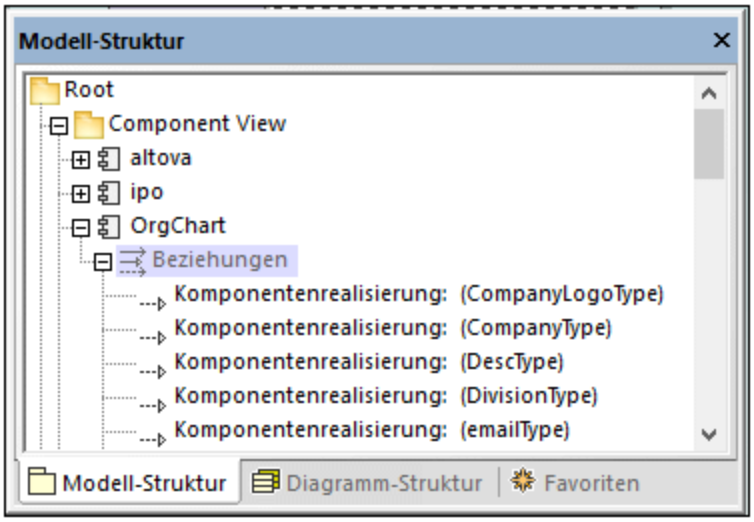


2. Klicken Sie in der Modellstruktur auf die neue Klasse `CompanyType` und ziehen Sie sie nach oben auf die **OrgChart**-Komponente unterhalb des Component View-Pakets. Es erscheint eine Info, wenn der Mauszeiger sich über einer Komponente befindet.

© 2016-2022 Altova GmbH



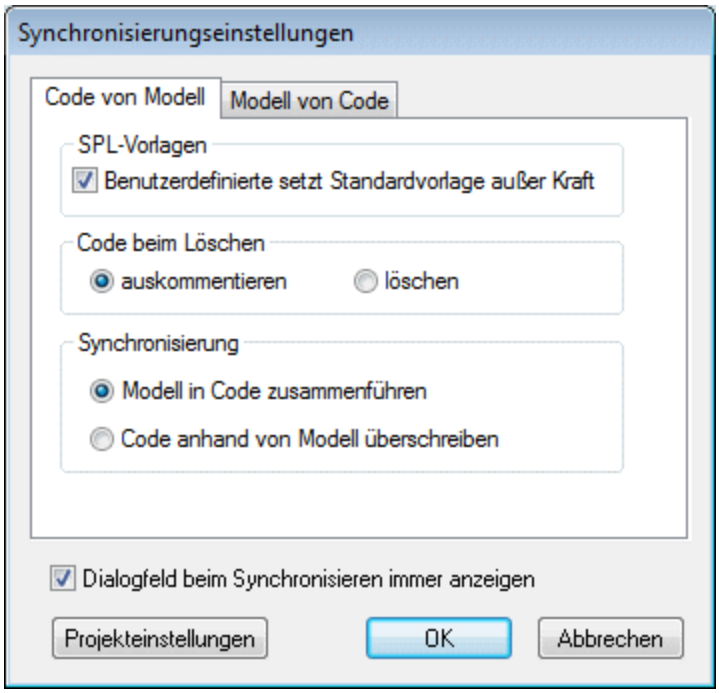
Mit dieser Methode wird eine Beziehung vom Typ "Komponentenrealisierung" zwischen einer Klasse und einer Komponente erstellt. Eine alternative Methode wäre, die Beziehung in einem Komponentendiagramm zu ziehen, siehe Komponentendiagramme[54]. Erweitern Sie das Element **Beziehungen** unterhalb der Komponente **Orgchart** um die neu erstellte Beziehung zu sehen.



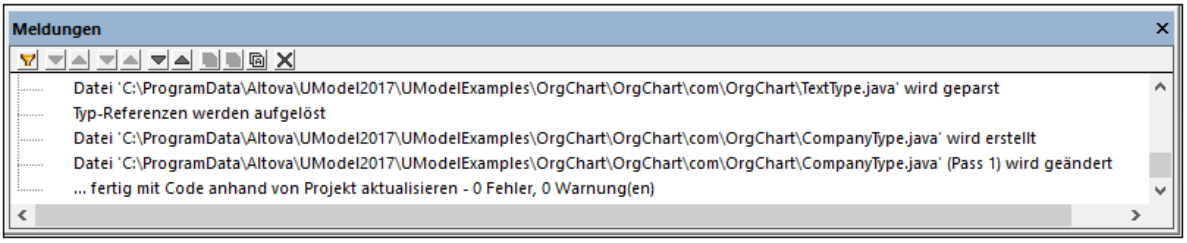
Merge von Programmcode aus einem Paket

Sie können in UModel Code auf Paket-, Komponentenebene oder für das gesamte Projekt generieren, siehe auch Synchronisieren von Modell und Quellcode 240. In diesem Beispiel generieren wir Code auf Komponentenebene. Gehen Sie dazu folgendermaßen vor:

1. Gehen Sie im Fenster "Modell-Struktur" unter "Component View" zur Komponente OrgChart.
2. Rechtsklicken Sie auf die Komponente "OrgChart**",** wählen Sie im Kontextmenü **Code Engineering | Merge Programmcode von UModel-Komponente**.



Im Meldungsfenster wird die Syntaxüberprüfung angezeigt, die durchgeführt wird, und der Status des Synchronisierungsvorgangs.



Nach Abschluss des Vorgangs ist die neue Klasse **CompanyType.java** zum Ordner **...\OrgChart\com\OrgChart** hinzugefügt.

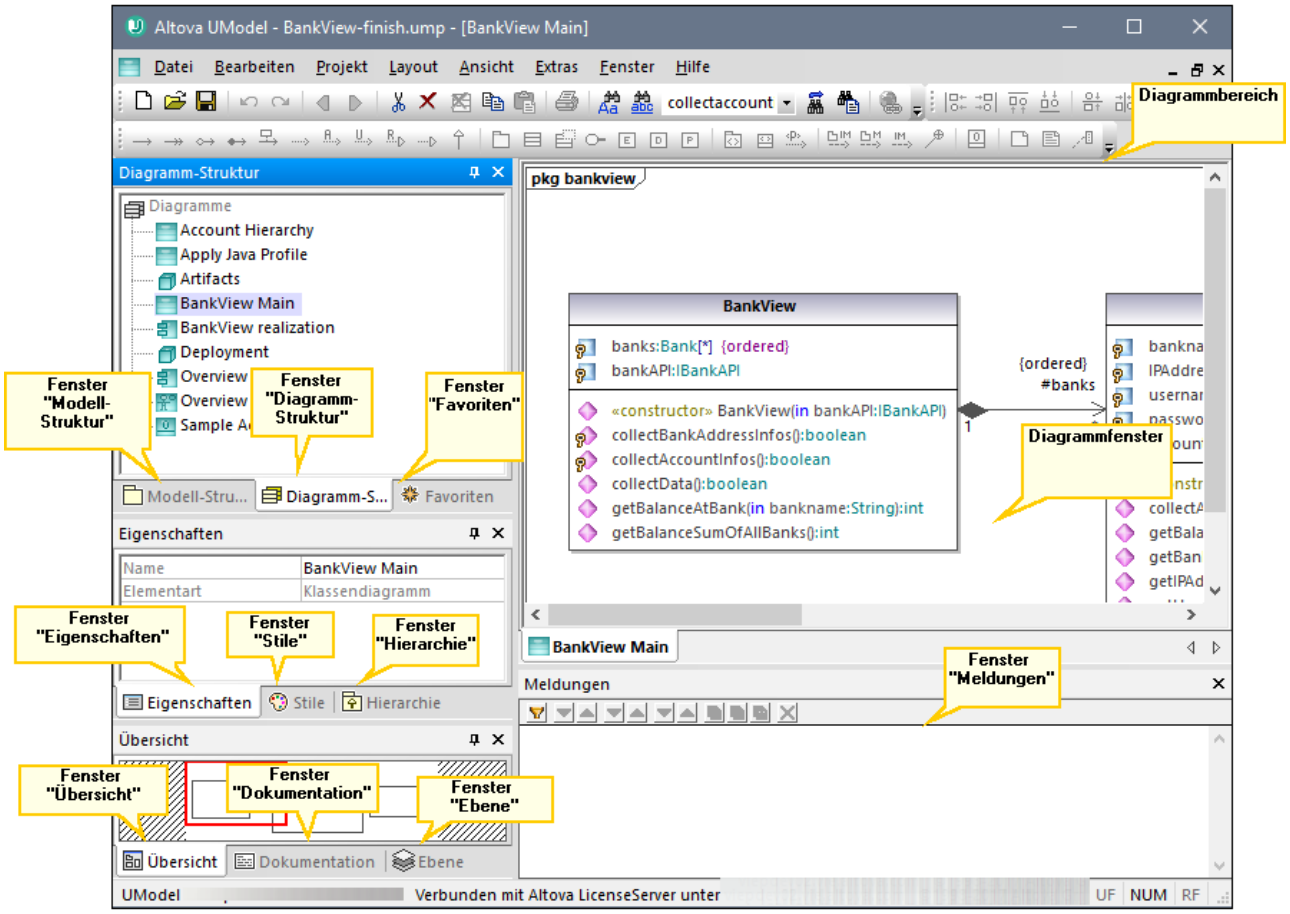
Alle Methoden-Body-Bereiche und Änderungen am Code werden entweder auskommentiert oder gelöscht, je nachdem, welche Option im Dialogfeld "Synchronisierungseinstellungen" in der Gruppe "Code beim Löschen" ausgewählt wurde.

© 2016-2022 Altova GmbH

Sie haben nun ein komplettes Round-Trip Engineering mit UModel durchgeführt.

3 Grafische Benutzeroberfläche von UModel

Die grafische Benutzeroberfläche von UModel besteht aus dem Diagrammhauptbereich und einer Reihe von kleineren Hilfsfenstern, über die Sie Daten eingeben oder anzeigen können. Der Diagrammbereich dient als Container für alle offenen Diagrammfenster. Um der Reihe nach alle offenen Diagrammfenster anzuzeigen, drücken Sie **Strg+Tab**.



Grafische Benutzeroberfläche von UModel

Standardmäßig sind die Hilfsfenster auf der linken Seite in Dreiergruppen angedockt und das Fenster "Meldungen" wird unterhalb des Diagrammbereichs angezeigt. Sie können alle Fenster jedoch nach Bedarf verschieben und andocken oder freischwebend anzeigen. Alle Fenster können über die Auswahlliste **Suchen** in der Hauptsymbolleiste oder durch Drücken von **Strg+F** durchsucht werden. Siehe auch Suchen und Ersetzen von Text 118.

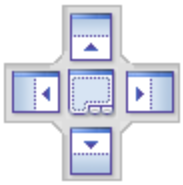
So docken Sie ein Fenster an oder ab:

- Klicken Sie mit der rechten Maustaste auf seine Titelleiste und wählen Sie im Kontextmenü den Befehl **Angedockt** (bzw. **Abgedockt**).

© 2016-2022 Altova GmbH

So verschieben Sie ein Fenster:

1. Klicken Sie auf die Titelleiste des Fensters und ziehen Sie es an eine neue Position. Daraufhin wird eine Reihe von Andockhilfen angezeigt.



2. Ziehen Sie das Fenster über den oberen, rechten, linken oder unteren Ziehpunkt, um das Fenster an der neuen Position anzudocken.

So setzen Sie alle Symbolleisten und Fenster wieder in den Originalzustand zurück:

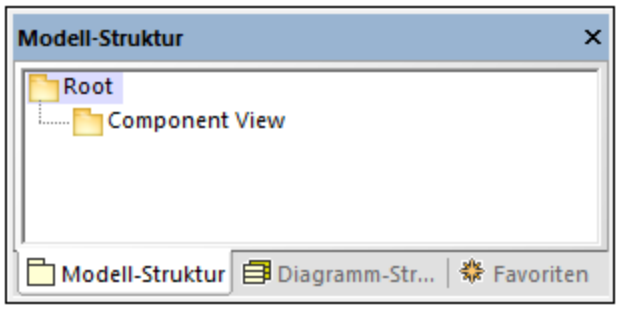
- Klicken Sie im Menü **Extras** auf **Symbolleisten und Fenster wiederherstellen**.

Dieses Kapitel bietet Referenzinformationen zu den Teilen, aus denen die grafische Benutzeroberfläche von UModel, besteht:

- Fenster "Modell-Struktur" 86
- Fenster "Diagramm-Struktur" 90
- Fenster "Favoriten" 91
- Fenster "Eigenschaften" 92
- Fenster "Stile" 93
- Fenster "Hierarchie" 94
- Fenster "Übersicht" 96
- Fenster "Dokumentation" 97
- Fenster "Ebene" 98
- Fenster "Meldungen" 99
- Diagrammfenster 101
- Diagrammbereich 102

3.1 Fenster "Modell-Struktur"

Im Fenster "Modell-Struktur" können Sie alle Einträge im UModel-Projekt (Pakete, Klassen, Diagramme, Beziehungen, usw.) anzeigen und bearbeiten.



Fenster "Modell-Struktur"

Wenn Sie ein neues UModel-Projekt erstellen, stehen standardmäßig zwei Pakete zur Verfügung: "Root" und "Component View". Diese beiden Pakete sind die einzigen, die nicht umbenannt oder gelöscht werden können. Das "Root"-Paket dient als Ausgangspunkt für die Modellierung aller anderen Elemente, während das "Component View"-Paket für das Code Engineering benötigt wird.

Sie können entweder über dieses Fenster oder direkt über ein Diagramm zusätzliche Pakete, Klassen, Diagramme und deren Hierarchie erstellen, siehe Erstellen von Elementen 113. Informationen zu weiteren Operationen, die Sie an Modellelementen in der Modell-Struktur ausführen können, finden Sie im Kapitel Anleitung zur Modellierung von... 111.

Anmerkung: UModel enthält eine Reihe von Beispielprojekten, anhand welcher Sie die Grundlagen der Modellierung erlernen und die grafische Benutzeroberfläche von UModel kennenlernen können. Diese Projekte befinden sich im folgenden Ordner: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**.

Anzeigen, Ausblenden und Sortieren von Modellelementen in der Modell-Struktur

Um zu konfigurieren, was im Fenster "Modell-Struktur" angezeigt werden soll und um die Sortieroptionen zu definieren, klicken Sie mit der rechten Maustaste in das Fenster und wählen Sie die gewünschte Menüoption aus. Um alle Aktionen, die an im Fenster "Modell-Struktur" angezeigten Modellelementen, ausgeführt werden können, anzuzeigen, klicken Sie mit der rechten Maustaste auf das Modellelement und sehen Sie sich die Kontextmenüoptionen an.

Erweitern und Reduzieren von Modellelementen in der Modell-Struktur

So erweitern Sie Modellelemente (z.B. Pakete) im Fenster "Modell-Struktur":

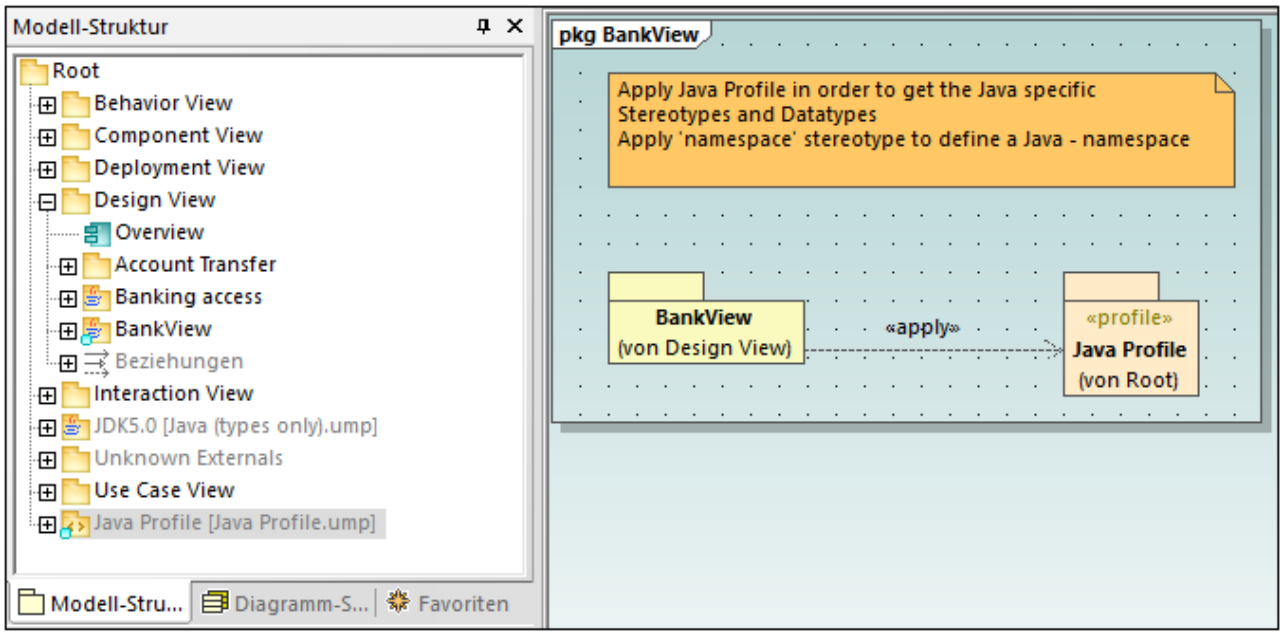
- Drücken Sie die *-Taste (Sternchen), um das aktuelle Modellelement sowie alle untergeordneten Elemente zu erweitern.
- Drücken Sie die +-Taste (Plus), um nur das aktuelle Modellelement zu erweitern.

© 2016-2022 Altova GmbH

Um die Pakete reduziert anzuzeigen, drücken Sie die -Taste (Bindestrich). Um alle Modellelemente zu reduzieren, klicken Sie auf das "Root"-Paket und drücken Sie - (Bindestrich). Beachten Sie, dass Sie dazu sowohl die Tasten der Standardtastatur als auch die Tasten des Zahlenblocks verwenden können.

Identifizieren von aktiven Diagrammeinträgen

Wenn ein Diagramm im Diagrammbereich geöffnet ist, werden einige Einträge im Fenster "Modell-Struktur" mit einem hellblauen Punkt am unteren Rand des Eintrags angezeigt. Dies sind die Modellelemente, die im gerade aktiven Diagramm angezeigt werden (wie z.B. "BankView" und "Java Profile" im Beispiel unten):



Symbolreferenz

Im Fenster "Modell-Struktur" können zahlreiche Symbole für Elemente und Diagramme in Ihrem Projekt, in den Code Engineering-Paketen sowie den importierten Profilen oder Unterprojekten angezeigt werden. Insbesondere können die folgenden Pakettypen angezeigt werden:

Symbol	Beschreibung
	UML-Standardpaket
	Java Namespace Root-Paket. Wird für die Generierung oder das Reverse Engineering von Java-Code verwendet.
	C# Namespace Root-Paket. Wird für die Generierung oder das Reverse Engineering von C#-Code verwendet.
	C++ Namespace Root-Paket. Wird für das Reverse Engineering von C++-Code verwendet.

Symbol	Beschreibung
	Visual Basic Namespace Root-Paket. Wird für die Generierung oder das Reverse Engineering von VB.NET-Code verwendet.
	XML Schema Namespace Root-Paket. Wird für die Generierung von XML-Schemas anhand des Modells oder den Import dieser Schemas in das Modell verwendet, siehe XML-Schema-Diagramme 490.
	Datenbank-Namespace Root-Paket. Wird für den Import von Datenbanken in das Modell und die Änderung ihrer Struktur anhand des Modells verwendet, siehe UModel und Datenbanken 555.
	Ein Namespace-Paket (ein Paket auf den das `<<namespace>>`-Stereotyp angewendet wurde).
	Ein-UML-Profil

In der folgenden Tabelle sind die Diagramme, die im Fenster "Modell-Struktur" angezeigt werden können, aufgelistet.

Symbol	Beschreibung
	Aktivitätsdiagramm
	BPMN 1 (Business Process Modeling Notation)-Geschäftsprozessdiagramm
	BPMN 2-Geschäftsprozessdiagramm
	BPMN 2-Choreographiediagramm
	BPMN 2-Kollaborationsdiagramm
	Klassendiagramm
	Kommunikationsdiagramm
	Komponentendiagramm
	Kompositionsstrukturdiagramm
	Datenbankdiagramm
	Deployment-Diagramm
	Interaktionsübersichtsdiagramm
	Objektdiagramm
	Paketdiagramm
	Profildiagramm
	Protokoll-Zustandsdiagramm

© 2016-2022 Altova GmbH

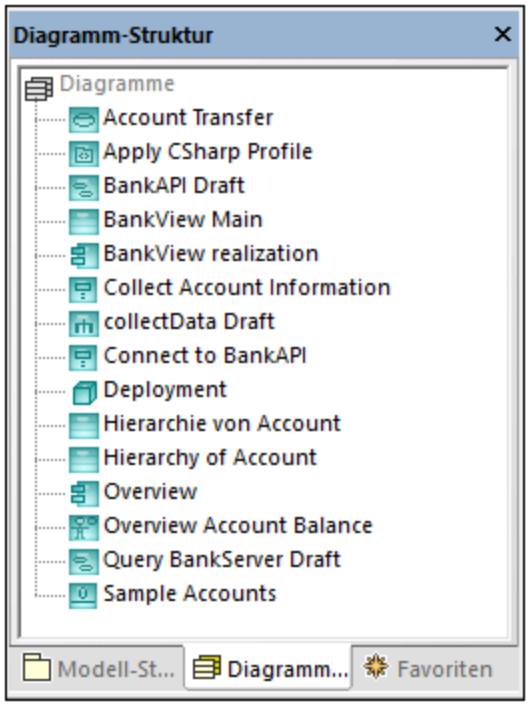
Symbol	Beschreibung
	Sequenzdiagramm
	Zustandsdiagramm
	SysML-Diagramme (9 Diagrammtypen)
	Zeitverlaufsdiagramm
	Use Case-Diagramm
	XML-Schema-Diagramm

Unten finden Sie einige Beispiele für UML-Modellierungselemente, die im Fenster "Modell-Struktur" angezeigt werden können. Nähere Informationen zu UML-Elementen und den Diagrammtypen, in denen diese vorkommen, finden Sie im Kapitel UML-Diagramme 356.

Symbol	Beschreibung
	Klasse
	Eigenschaft
	Operation
	Parameter
	Akteur
	Use Case
	Komponente
	Knoten
	Artefakt
	Schnittstelle
	Klasseninstanz (Objekt)
	Klassen-Instanz-Slot
	Beziehungen
	Einschränkungen

3.2 Fenster "Diagramm-Struktur"

Im Fenster "Diagramm-Struktur" werden alle im UModel-Projekt enthaltenen Diagramme angezeigt.



Fenster "Diagramm-Struktur"

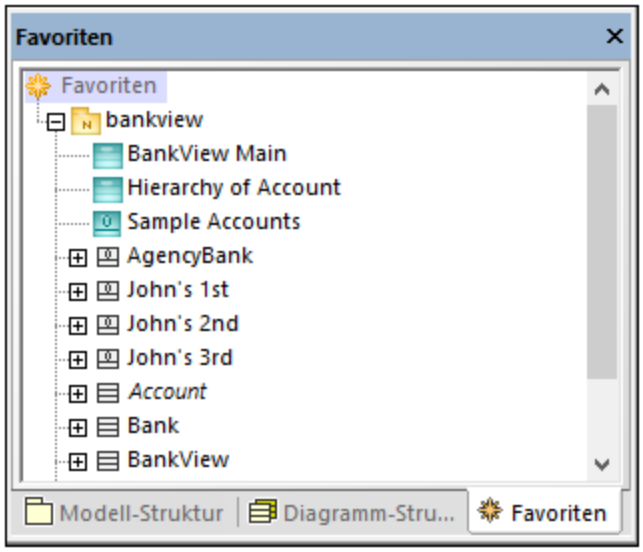
Diagramme in diesem Fenster können entweder als alphabetische Liste oder nach Typ gruppiert angezeigt werden. Um die Anzeigeoption zu ändern, klicken Sie mit der rechten Maustaste in das Fenster und aktivieren bzw. deaktivieren Sie die Option **Nach Diagrammtyp gruppieren**.

Anleitungen zum Erstellen, Öffnen und Generieren von Diagrammen sowie zum Modellieren des Inhalts finden Sie im Kapitel Anleitung zur Modellierung von... 111. Nähere Informationen zu den einzelnen Diagrammtypen finden Sie im Kapitel UML-Diagramme 356.

© 2016-2022 Altova GmbH

3.3 Fenster "Favoriten"

Im Fenster "Favoriten" werden alle Modellierungselemente oder Diagramme, die Sie als Favoriten hinzugefügt haben, angezeigt. Die "Favoriten" stellen eine persönliche, benutzerdefinierte Liste von Modellierungselementen oder Diagrammen dar, die Sie auf diese Art schnell aufrufen können.



Fenster "Favoriten"

Standardmäßig wird der Inhalt des Fensters "Favoriten" automatisch beim Speichern des Projekts gespeichert. Sie können diese Option über das Menü **Extras | Optionen** Register **Datei** ändern. Der Name der entsprechenden Option ist **Mit Projektdatei laden und speichern | Favoriten**.

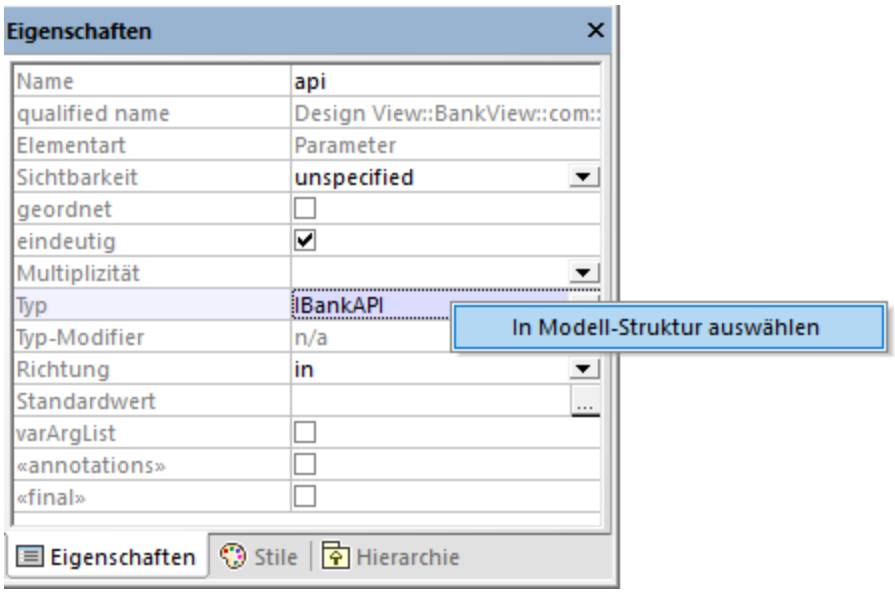
Einträge im Fenster "Favoriten" sind keine Kopien oder Klone, sondern stehen für die tatsächlichen Elemente bzw. Diagramme. Die meisten Aktionen, die Sie im Fenster "Modell-Struktur" anwenden können, lassen sich auch im Fenster "Favoriten" ausführen, darunter auch Hinzufügen und Löschen von Elementen. Nähere Informationen dazu finden Sie im Kapitel Anleitung zur Modellierung von... 111.

3.4 Fenster "Eigenschaften"

Im Fenster "Eigenschaften" werden Informationen über das gerade ausgewählte Modellelement (das sich im Fokus befindet) angezeigt. Bei dem "im Fokus" befindlichen Element kann es sich um ein im Fenster "Modell-Struktur" (oder in anderen Fenstern) ausgewähltes Element, ein im Diagramm ausgewähltes Element oder ein Diagramm selbst handeln.

Außerdem können Sie über das Fenster "Eigenschaften" die Eigenschaften des aktuell ausgewählten Elements bzw. der Beziehung ändern. Welche Eigenschaften zur Verfügung stehen, hängt von der Art des ausgewählten Elements ab. Es gibt Eigenschaften, die schreibgeschützt und somit ausgegraut (z.B. "Elementart") sind und solche, die Sie ändern können (z.B. "Name").

Wenn eine Operation oder Eigenschaft einen Parameter erhält, können Sie direkt vom Fenster "Eigenschaften" aus schnell zum entsprechenden Parametertyp im Fenster "Modell-Struktur" springen. Klicken Sie dazu im Fenster "Eigenschaften" mit der rechten Maustaste auf die Eigenschaft "Typ" des Parameters und wählen Sie im Kontextmenü den Befehl **In Modell-Struktur auswählen**. Dasselbe gilt auch für Rückgabeparameter.



Fenster "Eigenschaften"

Wenn Sie die Eigenschaft eines Elements im Fenster "Eigenschaften" ändern, wird diese sofort im Diagramm angezeigt. Wenn Sie umgekehrt eine Änderung im Diagramm vornehmen (z.B. die Sichtbarkeit einer Operation von `public` in `private` ändern), wirkt sich dies auf die entsprechende Eigenschaft im Fenster "Eigenschaften" aus.

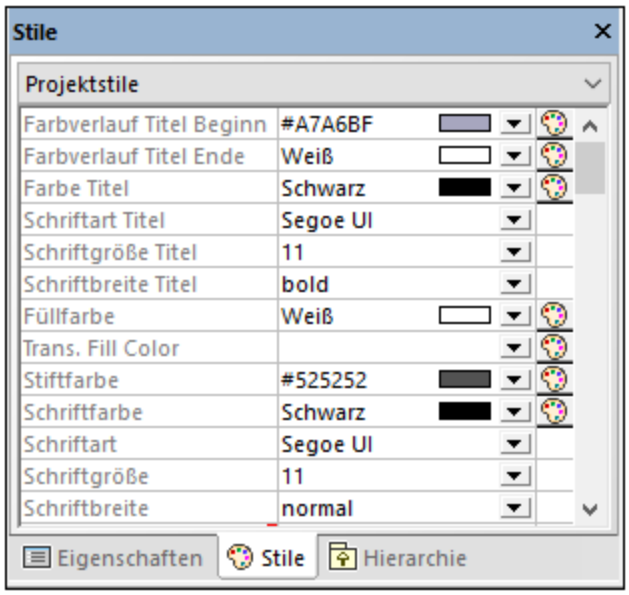
Innerhalb von doppelten spitzen Klammern stehende Eigenschaften, sind Stereotype (z.B. `«final»`). Sie können benutzerdefinierte Stereotype zum Projekt hinzufügen. Diese würden dann im Fenster "Eigenschaften" zusätzlich zu den Standardstereotypen im Fenster "Eigenschaften" angezeigt werden. Nähere Informationen dazu finden Sie unter Beispiel: Erstellen und Anwenden von Stereotypen 482.

 © 2016-2022 Altova GmbH

3.5 Fenster "Stile"

Im Fenster "Stile" können Sie die visuelle Darstellung von gerade ausgewählten Diagrammen oder Elementen (im Fokus) anzeigen oder ändern. Die Stilattribute werden in zwei allgemeine Gruppen eingeteilt:

- Formatierungseinstellungen (z.B. Schriftgröße, Schriftbreite, Farbe, usw.)
- Anzeigeeinstellungen (z.B. Hintergrundfarbe anzeigen, Raster, Sichbarkeitseinstellungen, usw.).



Fenster "Stile"

Wenn Sie die Eigenschaft im Fenster "Stile" ändern, wird dies sofort auf der Benutzeroberfläche angezeigt. Wenn Sie umgekehrt an einer anderen Stelle eine Stiländerung vornehmen (z.B. die Sichtbarkeit des Diagrammrasters über die Symbolleisten-Schaltfläche **Raster anzeigen** ändern), wirkt sich dies auf die entsprechende Eigenschaft im Fenster "Stile" aus.

Das Fenster "Stile" hat im oberen Bereich eine Dropdown-Liste, über die Sie die Ebene, auf der die Stiländerung angewendet werden soll, auswählen können (z.B. auf ein einzelnes Element oder auf Projektebene). Nähere Informationen dazu finden Sie unter:

- Ändern des Stils von Elementen 126
- Ändern des Stils von Diagrammen 134
- Ändern des Stils von Linien und Beziehungen 144

3.6 Fenster "Hierarchie"

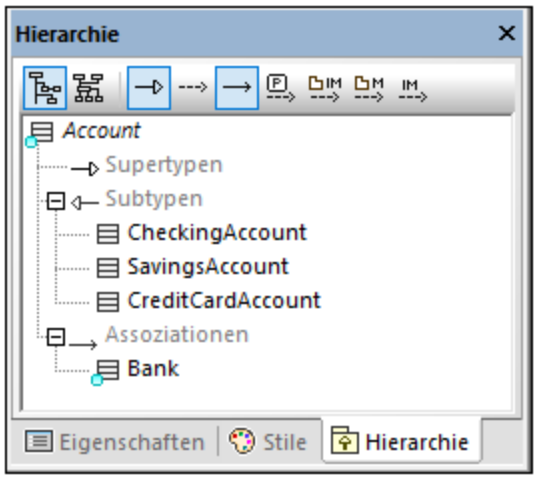
Im Fenster "Hierarchie" werden alle Beziehungen des aktuell ausgewählten Modellierungselements in zwei verschiedenen Ansichten angezeigt. Das Modellierungselement kann in einem Diagramm, im Fenster "Modell-Struktur" oder im Fenster "Favoriten" ausgewählt werden.

Einträge im Fenster "Hierarchie" können in zwei Ansichten angezeigt werden:

- Strukturansicht
- Graph-Ansicht

Um zwischen den Ansichten zu wechseln, klicken Sie in der linken oberen Ecke des Fensters auf die Schaltfläche **Strukturansicht anzeigen** bzw. **Graph. Ansicht anzeigen** .

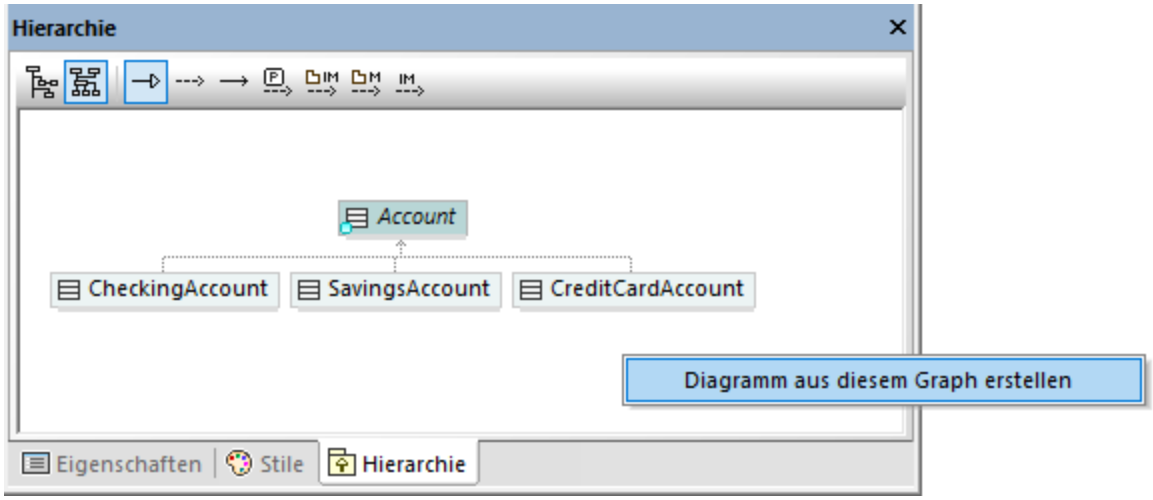
In der *Strukturansicht* werden mehrere Beziehungen des aktuell ausgewählten Elements in Form einer Struktur angezeigt. Klicken Sie auf die Schaltflächen am oberen Rand des Fensters, um die Arten der anzeigenden Beziehungen auszuwählen. In der Abbildung unten wurden nur Generalisierungen und Assoziationen für die Anzeige ausgewählt.



Fenster "Hierarchie" (Strukturansicht)

In der *Graph-Ansicht* sehen Sie eine einzige Gruppe von Beziehungen in einer hierarchischen Übersicht in Form eines Diagramms. In dieser Ansicht kann immer nur eine der Beziehungsschaltflächen aktiv sein. In der Abbildung unten ist gerade die Schaltfläche **Generalisierungen anzeigen** aktiv.

© 2016-2022 Altova GmbH



Fenster "Hierarchie" (graphische Ansicht)

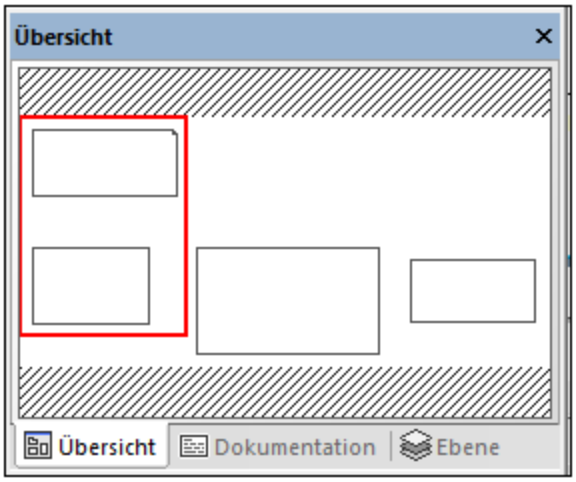
In der Graph-Ansicht können Sie Diagramme generieren, die die im Fenster sichtbaren Elemente enthalten. Klicken Sie dazu mit der rechten Maustaste in das Fenster und wählen Sie im Kontextmenü den Befehl **Diagramm aus diesem Graph erstellen**.

Die Einstellungen zum Fenster "Hierarchie" können über die Menüoption **Extras | Optionen | Ansicht** Gruppe **Hierarchie** im unteren Bereich des Dialogfelds geändert werden.

Um durch das Fenster "Hierarchie" zu navigieren, doppelklicken Sie auf eines der Elementsymbole im Fenster, um die Beziehungen dieses Elements anzuzeigen. Dies gilt sowohl für die Strukturansicht als auch für die Graph-Ansicht.

3.7 Fenster "Übersicht"

Im Fenster "Übersicht" werden die Umrisse des aktuell aktiven Diagramms in Form einer Übersicht angezeigt. Dieses Fenster ist besonders hilfreich, wenn Sie den Ansichtsbereich in großen Diagrammen verschieben möchten. Wenn Sie auf das rote Rechteck klicken und es ziehen, verschiebt sich der angezeigte Bereich im Diagramm.



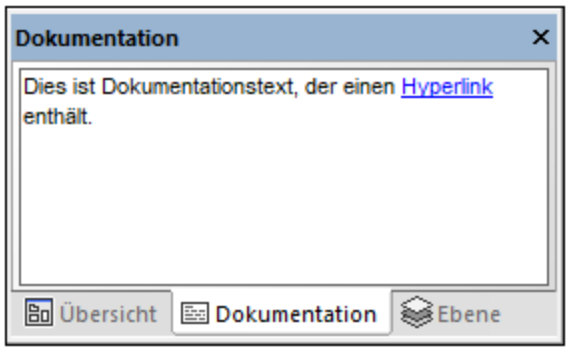
Fenster "Übersicht"

Siehe auch Vergrößern und Verkleinern von Diagrammen 142.

© 2016-2022 Altova GmbH

3.8 Fenster "Dokumentation"

Über das Fenster "Dokumentation" können alle der im Fenster "Modell-Struktur" verfügbaren UML-Elemente dokumentiert werden. Um Dokumentation hinzuzufügen, klicken Sie zuerst auf das gewünschte Element und geben Sie anschließend im Fenster "Dokumentation" Text ein. Dabei werden auch die Tastenkombinationen für die Standardbearbeitungsbefehle **Alle auswählen (Strg+A), Ausschneiden (Strg+X), Kopieren (Strg+C)** und **Einfügen (Strg+V)** unterstützt.



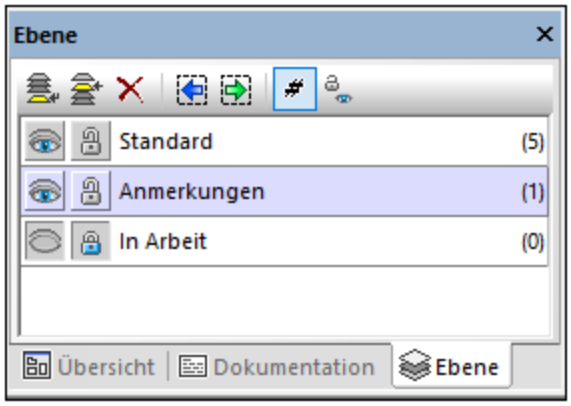
Fenster "Dokumentation"

Sie können die Rechtschreibung von Text im Dokumentationsfenster prüfen. Klicken Sie dazu mit der rechten Maustaste in das Fenster und wählen Sie im Kontextmenü den Befehl **Rechtschreibung der Dokumentation**.

Dokumentationstext kann auch in Form von Kommentaren in den generierten Quellcode exportiert oder beim Reverse Engineering aus dem Quellcode importiert werden. Nähere Informationen dazu finden Sie unter Dokumentieren von Elementen[125].

3.9 Fenster "Ebene"

Über das Fenster "Ebene" können Sie mehrere Ebenen für ein UModel-Diagramm definieren. Mit Hilfe von Ebenen können Sie Modellierungselemente eines Diagramms logisch gruppieren. So könnten Sie z.B. zusätzlich zur Standardebene einige zusätzliche Ebenen mit internen Anmerkungen oder nicht fertig gestellten Klassen erstellen.



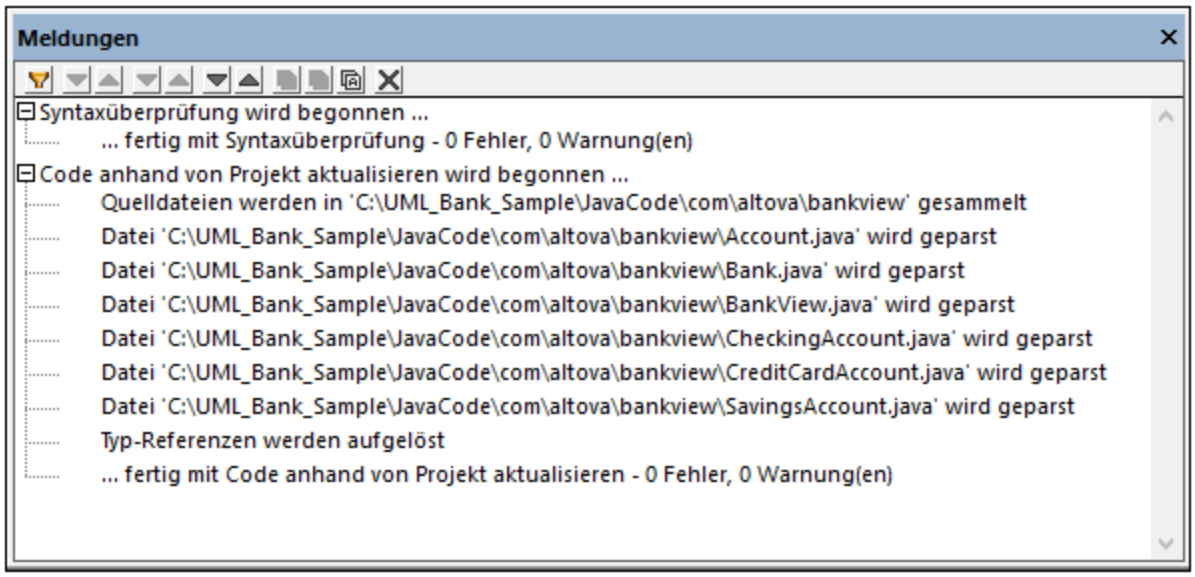
Fenster "Ebene"

Nähere Informationen dazu finden Sie unter Hinzufügen von Ebenen zu Diagrammen 138.

 © 2016-2022 Altova GmbH

3.10 Fenster "Meldungen"

Im Fenster "Meldungen" werden die folgenden Arten von Meldungen angezeigt: Informationsmeldungen, Warnungen und Fehler. Solche Meldungen können vorkommen, wenn Sie die Projektsyntax überprüfen (siehe Überprüfen der Projektsyntax[183]) oder wenn Sie Code Engineering-Aufgaben durchführen. Nähere Informationen zum Code Engineering finden Sie unter Generieren von Programmcode[180] und Importieren von Quellcode[209].



Fenster "Meldungen"

In der Tabelle unten sind mögliche Fehlerarten und deren Symbole aufgelistet.

Symbol	Beschreibung
Keines	Kennzeichnet eine Informationsmeldung.
	Kennzeichnet eine Warnmeldung. Warnungen sind weniger gravierend als Fehler, es kann aber auch bei einer Warnung vorkommen, dass Code nicht importiert oder generiert werden kann.
	Kennzeichnet eine Fehlermeldung. Bei einem Fehler schlägt die Codegenerierung oder der Codeimport fehl.

Über die Schaltflächen am oberen Rand des Fensters "Meldungen" können Sie die folgenden Aktionen ausführen:

Symbol	Beschreibung
	Filtert Meldungen nach ihrem Schweregrad (Informationsmeldungen und Warnungen). Wählen Sie **Alle aktivieren,** um alle Schweregrade zu inkludieren (dies ist die Standardeinstellung). Wählen Sie **Alle deaktivieren,** um alle Schweregrade aus dem Filter

Symbol	Beschreibung
	zu entfernen.
	Geht zum nächsten Fehler.
	Geht zum vorherigen Fehler.
	Geht zur nächsten Warnung.
	Geht zur vorherigen Warnung.
	Geht zur nächsten Zeile.
	Geht zur vorherigen Zeile.
	Kopiert die ausgewählte Zeile in die Zwischenablage.
	Kopiert die ausgewählte Zeile einschließlich aller untergeordneten Zeilen in die Zwischenablage.
	Kopiert den gesamten Inhalt des Fensters **Meldungen** in die Zwischenablage.
	Löscht die Meldungen im Fenster "Meldungen".

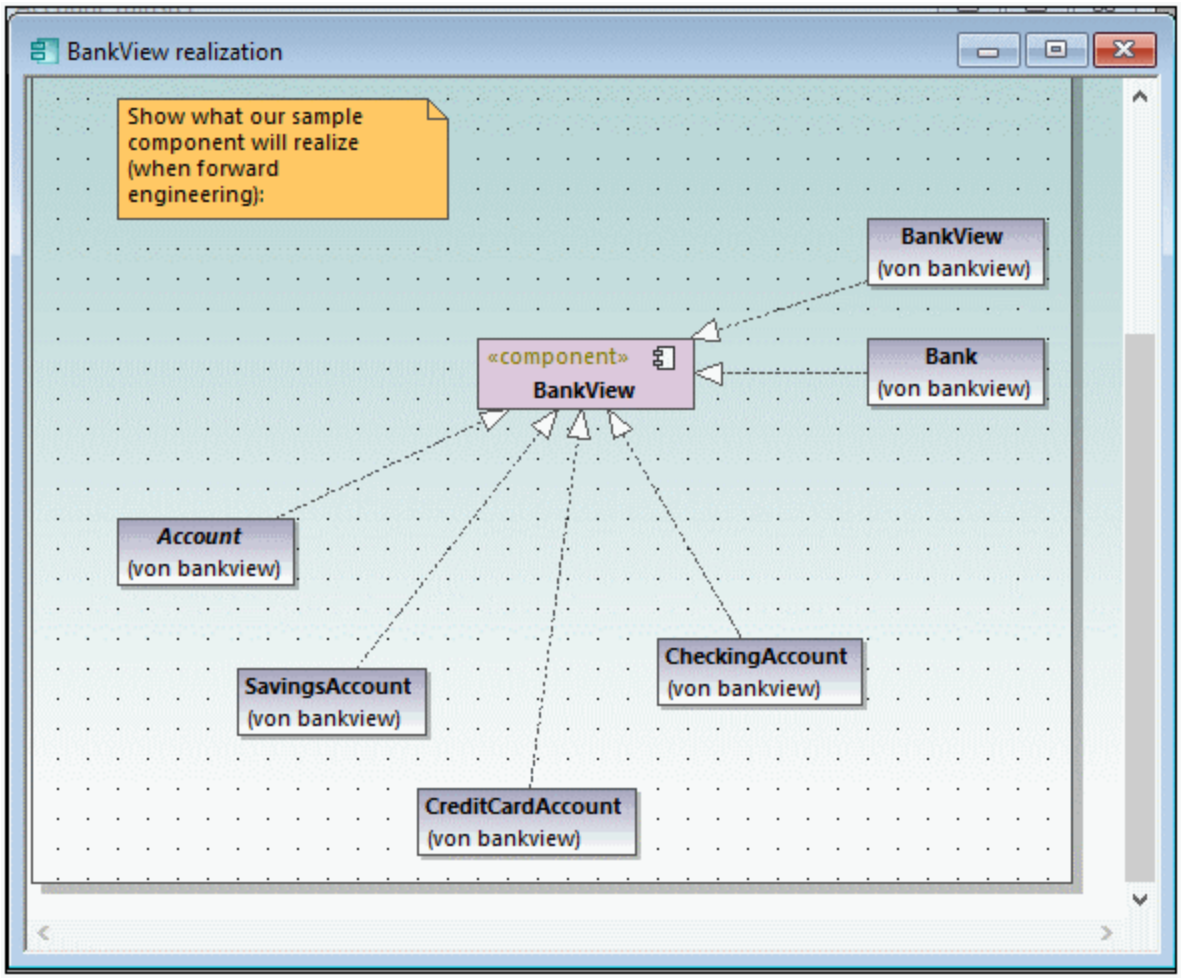
Wenn UModel als Visual Studio oder der Eclipse Plug-in ausgeführt wird und ein Parser-Fehler auftritt, können Sie direkt über das Fenster "Meldungen" zur Quellcodedatei gelangen, aus der der Fehler stammt. Klicken Sie dazu im Fenster "Meldungen" auf den Parser-Fehler. Nähere Informationen dazu finden Sie unter UModel Plug-in für Visual Studio 665 und UModel Plug-in für Eclipse 676.

 © 2016-2022 Altova GmbH

3.11 Diagrammfenster

Wenn Sie ein neues Diagramm erstellen oder ein vorhandenes öffnen, wird im Diagrammbereich[102] ein neues Diagrammfenster geladen. Das Diagrammfenster stellt den Zeichenbereich, in dem UML-Diagramme erstellt werden, dar. Wenn Sie mit der rechten Maustaste entweder in den Zeichenbereich selbst oder auf ein beliebiges Element darin klicken, stehen verschiedene Modellierungsbefehle zur Verfügung.

Je nachdem, welcher Diagrammtyp gerade aktiv (im Fokus) ist, stehen jeweils andere Symbolleisten-Schaltflächen und Kontextmenübefehle zur Verfügung. Wenn Sie z.B. in ein Klassendiagramm klicken, stehen nur Symbolleisten-Schaltflächen für Klassendiagrammelemente zur Verfügung. Um den Diagrammtyp anzuzeigen, klicken Sie in einen leeren Bereich im Diagramm und sehen Sie nach, was im Fenster "Eigenschaften"[92] als Elementart angezeigt wird. Man erkennt den Diagrammtyp auch am Symbol für das Diagramm, siehe Erstellen von Diagrammen[129].



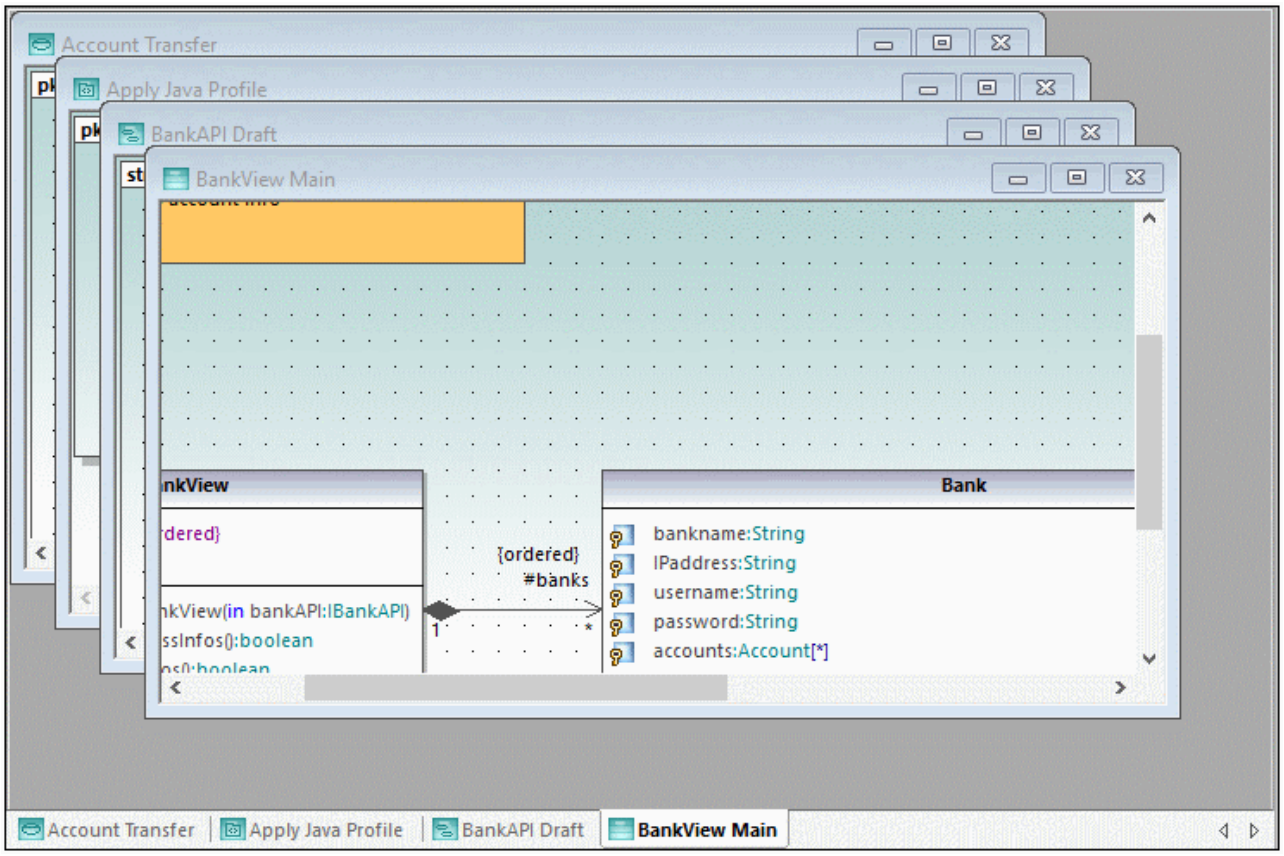
Diagrammfenster

Informationen zum Erstellen von neuen Diagrammen, zum Öffnen von bestehenden und zum Bearbeiten von Elementen im Diagramm finden Sie im Kapitel Anleitung zur Modellierung von...[111].

3.12 Diagrammbereich

Der Diagrammbereich enthält alle derzeit geöffneten Diagrammfenster. Informationen zum Erstellen neuer Diagramme, zum Öffnen bestehender und zum Bearbeiten von Elementen im Diagramm finden Sie im Kapitel Anleitung zur Modellierung von... 111.

In der Abbildung unten sehen Sie den Diagrammbereich, in dem vier Fenster geöffnet sind und mit Hilfe des Menübefehls **Fenster | Überlappend** positioniert werden.

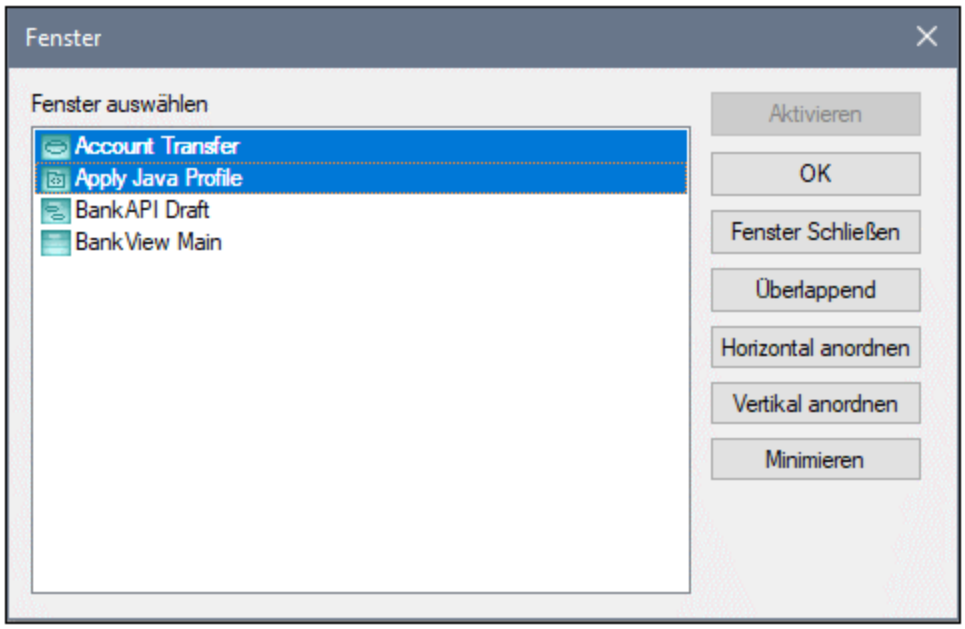


Diagrammbereich

Wenn Sie mit der rechten Maustaste im unteren Bereich des Diagrammbereichs auf das entsprechende Fensterregister klicken, stehen verschiedene Befehle für das aktuelle Diagrammfenster zur Verfügung.

Um diverse Befehle auf Fenster innerhalb des Diagrammbereichs anzuwenden, verwenden Sie die Befehle im Menü **Fenster**. Auch im Dialogfeld "Fenster" (Aufruf über den Menübefehl **Fenster | Fenster**) stehen einige Befehle zu Fenstern zur Verfügung.

© 2016-2022 Altova GmbH



Dialogfeld "Fenster"

Um im obigen Dialogfeld mehrere Fenster auszuwählen, halten Sie die **Strg**-Taste gedrückt, während Sie auf die entsprechenden Einträge klicken.

Um der Reihe nach alle offenen Diagrammfenster aufzurufen, drücken Sie **Strg+Tab**.

4 UModel Befehlszeilenschnittstelle

Zusätzlich zur grafischen Benutzeroberfläche verfügt UModel auch über eine Befehlszeilenschnittstelle. Um die Befehlszeilenschnittstelle aufzurufen, führen Sie die Datei **UModelBatch.exe** aus dem Ordner **C:\Programme\Altova\UModel2023** aus. Wenn Sie die UModel 32-Bit-Version auf einem 64-Bit-Betriebssystem ausführen, so lautet der Pfad **C:\Programme (x86)\Altova\UModel2023**.

Die Befehlszeilenparametersyntax wird unten gezeigt und kann im Fenster zur Befehlseingabe angezeigt werden, wenn Sie eingeben: **umodelbatch /?**

Hinweis: Wenn der Pfad oder Dateiname Leerzeichen enthält, setzen Sie den Pfad/Dateinamen bitte in Anführungszeichen, d.h. "C:\Programme\...\Dateiname"

```
usage: UModelBatch.exe [project] [options]

/? oder /help ... diese Hilfeinformationen anzeigen

Projekt     ... Projektdatei (*.ump)
/new[=Datei]... neues Projekt erstellen/speichern/unter einem neuen Namen speichern,
siehe Erstellen, Laden und Speichern von Projekten im Batch-Modus 109
/set        ... Optionen permanent festlegen
/gui        ... UModel-Benutzeroberfläche anzeigen

Befehle (in der angeführten Reihenfolge ausgeführt):
/chk        ... Projektsyntax überprüfen
/isd=Pfad   ... Quellverzeichnis importieren
/isp=Datei  ... Quellprojektdatei importieren
(*.project,*.xml,*.jpx,*.csproj,*.csdproj,*.vcxproj,*.vbproj,*.vbdproj,*.sln,*)
/ibt=Liste  ... Binärtypen importieren (Liste der Binär[Typnamen] definieren)
                (';'=Trennzeichen, '*'=alle Typen, '#' vor Assembly-Namen)
/ixd=Pfad   ... XML-Schemaverzeichnis importieren
/ixs=Datei  ... XML-Schemadatei importieren (*.xsd)
/m2c        ... Programmcode anhand von Modell aktualisieren (exportieren/Forward
Engineering)
/c2m        ... Modell anhand von Programmcode aktualisieren (importieren/Reverse
Engineering)
/ixf=Datei  ... XMI-Datei importieren
/exf=Datei  ... XMI-Datei exportieren
/inc=Datei  ... Datei inkludieren
/mrg=Datei  ... Datei zusammenführen
/doc=Datei  ... Dokumentation in angegebene Datei schreiben
/lue[=cpri] ... alle Elemente auflisten, die in keinem Diagramm verwendet werden (d.h.
nicht verwendete)
/ldg        ... alle Diagramme auflisten
/lcl        ... alle Klassen auflisten
/lsp        ... alle freigegebenen Pakete auflisten
/lip        ... alle inkludierten Pakete auflisten

Optionen zum Speichern als neues Projekt:
/npad=opt   ... relative Dateipfade anpassen (Yes | No | MakeAbsolute)

Optionen für Import-Befehle:
```


© 2016-2022 Altova GmbH

```
/iclg[=lang ... Codesprache (Java1.4 | Java5.0 | Java6.0 | Java7.0 | Java8.0 | Java9.0
|
                              Java10.0 | Java11.0 | Java12.0 | Java13.0 | Java14.0 |
Java15.0 |
                              C#1.2 | C#2.0 | C#3.0 | C#4.0 | C#5.0 | C#6.0 | C#7.0 |
C#7.1 | C#7.2 | C#7.3 | C#8.0 | C#9.0 |
                              VB7.1 | VB8.0 | VB9.0 |
                              C++98 | C++11 | C++14 | C++17)
/ipsd[=0|1] ... Unterverzeichnisse verarbeiten (rekursiv)
/irpf[=0|1] ... relativ zur UModel-Projektdatei importieren
/ijdc[=0|1] ... JavaDocs als Java-Kommentare
/icdc[=0|1] ... DocComments als C#-Kommentare
/icds[=lst] ... C# definierte Symbole
/ivdc[=0|1] ... DocComments als VB-Kommentare
/ivds[=lst] ... VB-definierte Symbole (benutzerdefinierte Konstanten)
/icppdm[=lst] ... C++-definierte Makros
/icpphi[=0|1] ... schreibgeschützte C++-Header-Dateien
/icpphc[=0|1] ... .h-Dateien als .cpp-Dateien behandeln
/icppms[=0|1] ... C++ Microsoft Compiler-Kompatibilität aktivieren compatibility
/icppmv[=ver] ... zu verwendende MSVC-Version (1900 | 1800 | 1700 | 1600 | 1500 | 1400
| 1310 | 1300 | 1200)
/icppsy[=0|1] ... C++ System Include-Dateien automatisch ermitteln
/icppid[=lst] ... Liste der zu verwendenden C++ Include-Verzeichnisse
/icppsd[=lst] ... Liste der zu verwendenden C++ System-Include-Verzeichnisse
/icppag[=arg] ... zusätzliche C++-Argumente für den Compiler
/imrg[=0|1] ... zusammengeführte synchronisieren
/iudf[=0|1] ... Verzeichnisfilter verwenden
/iflt[=lst] ... Verzeichnisfilter (Voreinstellungen /iudf)


Optionen für den Import von Binärtypen (nach /iclg):
/ibrt=vers  ... Runtime Version
/ibpv=Pfad  ... Variable PATH zum Durchsuchen von nativen Code-Bibliotheken außer
Kraft setzen
/ibro[=0|1] ... nur Reflection-Kontext verwenden
/ibua[=0|1] ... hinzugefügte referenzierte Typen mit Paketfilter verwenden
/ibar[=flt] ... referenzierte Typpaketfilter hinzufügen (presets /ibua)
/ibot[=0|1] ... nur Typen importieren
/ibuv[=0|1] ... Mindestsichtbarkeitsfilter verwenden
/ibmv[=key] ... Schlüsselwort der erforderlichen Mindestsichtbarkeit (presets /ibuv)
/ibsa[=0|1] ... Attributabschnitte / Annotations-Modifier unterdrücken
/iboa[=0|1] ... nur ein Attribut pro Attributabschnitt erstellen
/ibss[=0|1] ... 'Attribute'-Suffix bei Attributtypnamen unterdrücken


Optionen für die Diagrammgenerierung:
/dgen[=0|1] ... Diagramme generieren
/dopn[=0|1] ... generierte Diagramme öffnen
/dsac[=0|1] ... Attributbereich anzeigen
/dsoc[=0|1] ... Operation-Bereich anzeigen
/dscc[=0|1] ... Bereiche für geschachtelte Classifier anzeigen
/dstv[=0|1] ... Eigenschaftswerte anzeigen
/dudp[=0|1] ... .NET-Eigenschaftsbereich verwenden
/dspd[=0|1] ... .NET-Eigenschaftsbereich anzeigen
```

```
Optionen für Export-Befehle:
/ejdc[=0|1] ... Java-Kommentare als JavaDocs
/ecdc[=0|1] ... C#-Kommentare als DocComments
/evdc[=0|1] ... VB-Kommentare als DocComments
/espl[=0|1] ... benutzerdefinierte SPL-Vorlagen verwenden
/ecod[=0|1] ... gelöschte auskommentieren
/emrg[=0|1] ... zusammengeführte synchronisieren
/egfn[=0|1] ... fehlende Dateinamen generieren
/eusc[=0|1] ... Syntaxüberprüfung verwenden

Optionen für den XMI-Export:
/exid[=0|1] ... UUIDs exportieren
/exex[=0|1] ... UModel-spezifische Erweiterungen exportieren
/exdg[=0|1] ... Diagramme exportieren (presets /exex)
/exuv[=ver] ... UML-Version (UML2.0 | UML2.1.2 | UML2.2 | UML2.3 | UML2.4 | UML2.5 |
UML2.5.1)
)

Optionen für die Dateizusammenführung:
/mcan=Datei  ... gemeinsame Vorgängerdatei

Optionen für die Dokumentationsgenerierung:
/doof=fmt ... Ausgabeformat (HTML | RTF | MSWORD | PDF )
/dsps=Datei      ...SPS-Designdatei
```


© 2016-2022 Altova GmbH

Beispiel 1: Importieren von Java-Quellcode unter Beibehaltung der Einstellungen

Der folgende Befehl importiert Quellcode und erstellt eine neue Projektdatei. Beachten Sie, dass der Projektpfad Leerzeichen enthält und innerhalb von Anführungszeichen steht.

```
"C:\Programme\Altova\UModel2023\UModelBatch.exe" /new="C:\My
Projects\Fred.ump" /isd="X:TestCases\UModel\Fred" /set /gui /iclg=Java8.0 /ipsd=1 /ijdc=1
/dgen=1 /dopn=1 /dmax=5 /chk
```

Im folgenden sehen Sie eine Liste aller Optionen mit ihrer Beschreibung:

/new	Gibt an, dass die neu erstellte Projektdatei in **C:\My Projects** den Namen "Fred.ump" erhalten soll.
/isd	Gibt an, dass das Quellverzeichnis **X:\TestCases\UModel\Fred** sein soll.
/set	Gibt an, dass alle in der Befehlszeile verwendeten Optionen in der Registrierungsdatei gespeichert werden sollen (Wenn UModel anschließend gestartet wird, werden diese Einstellungen die Standardeinstellungen).
/gui	die grafische Benutzeroberfläche von UModel bei der Batch-Verarbeitung anzeigen
/iclg	UModel importiert den Code als Java 8.0
/ipsd=1	alle im Parameter /**isd** definierten Unterverzeichnisse des Root-Verzeichnisses rekursiv verarbeiten
/ijdc=1	gegebenenfalls JavaDoc anhand von Kommentaren erstellen
/dgen=1	Diagramme generieren
/dopn=1	generierte Diagramme öffnen
/chk	Syntaxüberprüfung durchführen

Beispiel 2: Synchronisieren von Code anhand des Modells

Der folgende Befehl aktualisiert Code anhand einer vorhandenen Projektdatei ("**C:\UModel\Fred.ump**").

```
"C:\Programme\Altova\UModel2023\UModelBatch.exe" "C:
\UModel\Fred.ump" /m2c /ejdc=1 /ecod=1 /emrg=1 /egfn=1 /eusc=1
```

Die Optionen haben dieselbe Bedeutung, wie in den vorhergehenden Beispielen:

/m2c	Code anhand von Modell aktualisieren
/ejdc	Kommentare im Projektmodell sollen als JavaDoc generiert werden
/ecod=1	gelöschten Code auskommentieren
/emrg=1	zusammengeführten Code synchronisieren

/egfn=1	fehlende Dateinamen im Projekt generieren
/eusc=1	Syntaxüberprüfung verwenden

Beispiel 3: Importieren von Java-Binärdateien in das Modell

Angenommen, das Verzeichnis **C:\JavaProject\bin** enthält einige Java-Klassen-Binärdateien, die Sie in UModel importieren möchten. Führen Sie dazu den folgenden Befehl aus:

```
"<C:\Programme\Altova\UModel2023\UModelBatch.exe>" /new="C:
\JavaProject\Result.ump" /ibt=*C:
\JavaProject\bin /iclg=Java8.0 /ibrt=JDK1.8.0_144 /dgen=1 /chk
```

Es werden die folgenden Optionen verwendet:

/new	Erstellt unter dem angegebenen Pfad ein neues UModel-Projekt.
/ibt	Weist UModel an, Binärtypen zu importieren. Das Sternchen vor dem Pfad gibt an, dass alle Binärtypen unter diesem Pfad importiert werden sollen.
/iclg	Definiert die Codegenerierungssprache (in diesem Beispiel "Java8.0").
/ibrt	Definiert das Runtime Environment (in diesem Beispiel "JDK1.8.0_144"). Dies ist derselbe Wert, der im Dialogfeld "Binärtypen importieren" angezeigt wird, siehe Importieren von Java-, C#- und VB.NET-Binärdateien 225. Sie können auch einen Wert wie z.B. "jdk-10.0.1", wie in der JAVA_HOME Umgebungsvariablen definiert, verwenden. Für C# können Sie den Wert /ibrt:any oder Werte, wie Sie in der "Runtime" Dropdown-Liste der Benutzeroberfläche angezeigt werden, verwenden. Stellen Sie dabei sicher, dass Sie alle Leerzeichen eliminieren. Beispiele: /ibrt:any /ibrt:.NET5 /ibrt:.NETFramework4.8(v4.8.3752) Die Option "any" ist dieselbe wie wenn Sie in der Runtime-Dropdown-Liste "beliebige (Disassembler verwenden)" auswählen und ist die empfohlene Option.
/dgen=1	Generiert Diagramme.
/chk	Führt nach dem Import eine Syntaxüberprüfung durch.

© 2016-2022 Altova GmbH

4.1 Datei: New / Load / Save-Optionen

Wenn Sie **UModelBatch.exe** mit einem Befehl wie `UModelBatch MeinProjekt.ump` ausführen, können Sie die folgenden Parameter verwenden:

`/new`	Dieser Parameter definiert den Pfad und Dateinamen der neu zu erstellenden UModel-Projektdatei (*.ump). Sie können mit diesem Befehl auch ein vorhandenes Projekt laden und unter einem anderen Namen speichern, z.B.: `UmodelBatch.exe MyFile.ump /new=MyBackupFile.ump`
`/set`	Dieser Parameter überschreibt die aktuellen Standardeinstellungen in der Registry durch die von Ihnen definierten Optionen.
`/gui`	Dieser Parameter zeigt die grafische Benutzeroberfläche von UModel während des Batch-Vorgangs an.

In den folgenden Beispielen wird gezeigt, wie Sie Projekte im vollständigen Batch-Modus (d.h. ohne, dass der Parameter `/gui` gesetzt ist) erstellen, laden oder speichern.

new

UModelBatch /new=xxx.ump (options)
Erstellt ein neues Projekt, führt die Optionen aus, xxx.ump wird immer (unabhängig von den Optionen) gespeichert

auto save

UModelBatch xxx.ump (options)
Lädt das Projekt xxx.ump, führt die Optionen aus, xxx.ump wird **nur dann** gespeichert, wenn sich das Dokument geändert hat (wie **/ibt**)

save

UModelBatch xxx.ump (options) /new
Lädt das Projekt xxx.ump, führt die Optionen aus, xxx.ump wird **immer** gespeichert (unabhängig von den Optionen)

save as

UModelBatch xxx.ump (options) /new=yyy.ump
Lädt das Projekt xxx.ump, führt die Optionen aus, speichert xxx.ump immer als yyy.ump (unabhängig von den Optionen)

In den folgenden Beispielen wird gezeigt, wie Sie Projekte im Batch-Modus, wobei die Benutzeroberfläche von UModel angezeigt wird (Parameter `/gui` ist gesetzt) erstellen, laden oder speichern.

new

UModelBatch /gui /new (options)

Erstellt ein neues Projekt, führt die Optionen aus, nichts wird gespeichert, die GUI bleibt offen

save new

UModelBatch /gui /new=xxx.ump (options)
Erstellt ein neues Projekt, führt die Optionen aus, xxx.ump wird gespeichert, die GUI bleibt offen

user mode

UModelBatch /gui xxx.ump (options)
Lädt das Projekt xxx.ump, führt die Optionen aus, nichts wird gespeichert, die GUI bleibt offen

save

UModelBatch /gui xxx.ump (options) /new
Lädt das Projekt xxx.ump, führt die Optionen aus, xxx.ump wird gespeichert, die GUI bleibt offen

save as

UModelBatch /gui xxx.ump (options) /new=yyy.ump
Lädt das Projekt xxx.ump, führt die Optionen aus, speichert xxx.ump als yyy.ump, die GUI bleibt offen

Das Projekt wird erfolgreich gespeichert, wenn bei Ausführung der Optionen kein schwerwiegender Fehler auftritt.

© 2016-2022 Altova GmbH

5 Anleitung zur Modellierung von...

Altova Website: UML-Modellierung

In diesem Kapitel wird erläutert, wie Sie UML-Elemente, Diagramme und Beziehungen über die grafische Benutzeroberfläche von UModel erstellen und bearbeiten. Es dient als Anleitung zur Modellierung mit UModel. Die Anleitungen in diesem Kapitel gelten für UModel als ganzes und sind nicht für ein bestimmtes Element oder Diagramm spezifisch, es sei denn, dies ist explizit erwähnt. Informationen zu den einzelnen Diagrammtypen (gruppiert nach Diagrammtyp) finden Sie im Kapitel UML-Diagramme 356.

Die Informationen in diesem Kapitel sind in die folgenden Kategorien gegliedert: Elemente, Diagramme, Beziehungen und Stereotype.

Elemente	Diagramme	Beziehungen	Stereotype
Erstellen von Elementen 113	Erstellen von Diagrammen 129	Erstellen von Beziehungen 143	Stereotype und Eigenschaftswerte 155
Einfügen von Elementen aus dem Modell in ein Diagramm 114	Generieren von Diagrammen 130	Ändern des Stils von Linien und Beziehungen 144	Eigenschaftswerte 156
Umbenennen, Verschieben und Kopieren von Elementen 116	Öffnen von Diagrammen 133	Anzeigen von Elementbeziehungen 146	Anwenden von Stereotypen 157
Löschen von Elementen 117	Löschen von Diagrammen 134	Assoziationen 147	Anzeigen oder Ausblenden von Eigenschaftswerten 159
Konvertieren von Elementen 118	Ändern des Stils von Diagrammen 134	Collection-Assoziationen 150	
Suchen und Ersetzen von Text 118	Ausrichten von Modellierungselementen und Anpassen der Größe 136	Enthältbeziehung 153	
Überprüfen, wo und ob Elemente verwendet werden 120	Typ-Autokomplettierung in Klassen 140		
Einschränken von Elementen 121	Vergrößern und Verkleinern von Diagrammen 142		
Erstellen von Hyperlinks zu Elementen 122	Hinzufügen von Ebenen zu Diagrammen 138		

Elemente	Diagramme	Beziehungen	Stereotype
Dokumentieren von Elementen 125			
Ändern des Stils von Elementen 126			

Anmerkung: UModel enthält eine Reihe von Beispielprojekten, anhand welcher Sie die Grundlagen der Modellierung erlernen und die grafische Benutzeroberfläche von UModel kennenlernen können. Diese Projekte befinden sich im folgenden Ordner: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**.

© 2016-2022 Altova GmbH

5.1 Elemente

5.1.1 Erstellen von Elementen

Sie können Elemente mit UModel auf folgende Arten erstellen:

- Über das Fenster Modell-Struktur[86]. Bei dieser Methode werden die Elemente nur zum Modell hinzugefügt und Sie können diese später gegebenenfalls zu Diagrammen hinzufügen.
- Über das Diagrammfenster. Elemente, die zu einem Diagramm hinzugefügt werden, werden automatisch auch zum Modell hinzugefügt. Falls Sie ein Element später löschen müssen, können Sie wählen, ob Sie es nur aus dem Diagramm oder auch aus dem Modell löschen möchten.

So fügen Sie Elemente über das Fenster "Modell-Struktur" hinzu:

- Klicken Sie im Fenster Modell-Struktur[86] (oder im Fenster Favoriten[91]) mit der rechten Maustaste auf das Element (z.B. Paket), unter dem das neue Element angezeigt werden soll und wählen Sie im Kontextmenü den Befehl **Neues Element | <Elementname>**. Um z.B. unterhalb des Pakets "Root" ein neues Paket hinzuzufügen, klicken Sie mit der rechten Maustaste auf das Paket "Root" und wählen Sie **Neues Element | Paket**.

So fügen Sie Elemente über das Diagrammfenster hinzu:

1. Erstellen Sie ein neues Diagramm (siehe Erstellen von Diagrammen[129]) oder öffnen Sie ein vorhandenes (siehe Öffnen von Diagrammen[133]).
2. Wählen Sie eine der folgenden Methoden:
   a. Klicken Sie mit der rechten Maustaste in das Diagramm und wählen Sie den Kontextmenübefehl **Neu | <Elementname>**.
   b. Klicken Sie auf die Symbolleistenschaltfläche des gewünschten Elements und klicken Sie in das Diagramm. Um mehrere Elemente desselben Typs einzufügen, halten Sie die **Strg**-Taste gedrückt, bevor Sie in das Diagramm klicken.

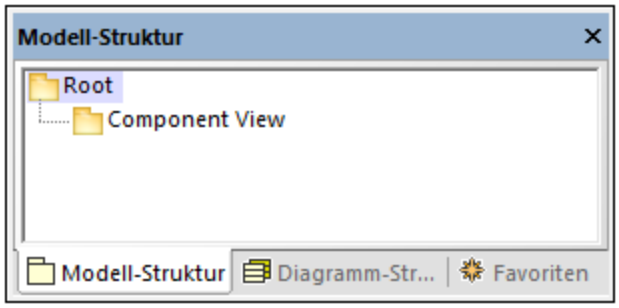
Pakete

Sie werden beim Modellieren von Elementen wahrscheinlich mit Paketen öfter als mit anderen Elementen arbeiten müssen. Jeder mit einem Ordnersymbol markierte Eintrag im Fenster "Modell-Struktur" steht für ein UML-Paket. Pakete in UModel dienen als Container für alle anderen UML-Modellierungselemente (einschließlich Diagramme, Klassen usw.) und verhalten sich folgendermaßen:

- Sie können an jeder Stelle der Modell-Struktur erstellt werden.
- Sie können in andere Pakete (und auch in gültige Modelldiagramme) verschoben und kopiert werden, siehe Umbenennen, Verschieben und Kopieren von Elementen[116].
- Sie können beim Generieren oder Synchronisieren von Code und Modell als Quell- oder Zielelemente verwendet werden, siehe Forward Engineering (Modell zu Code)[65] und Reverse Engineering (Code zu Modell)[75].

Wenn Sie ein neues UModel-Projekt erstellen, stehen standardmäßig zwei Pakete zur Verfügung: "Root" und "Component View". Diese beiden Pakete sind die einzigen, die nicht umbenannt oder gelöscht werden können. Das "Root"-Paket dient als Ausgangspunkt für die Modellierung aller anderen Elemente, während das

"Component View"-Paket für das Code Engineering benötigt wird.



UModel-Standardpakete

5.1.2 Einfügen von Elementen aus dem Modell in ein Diagramm

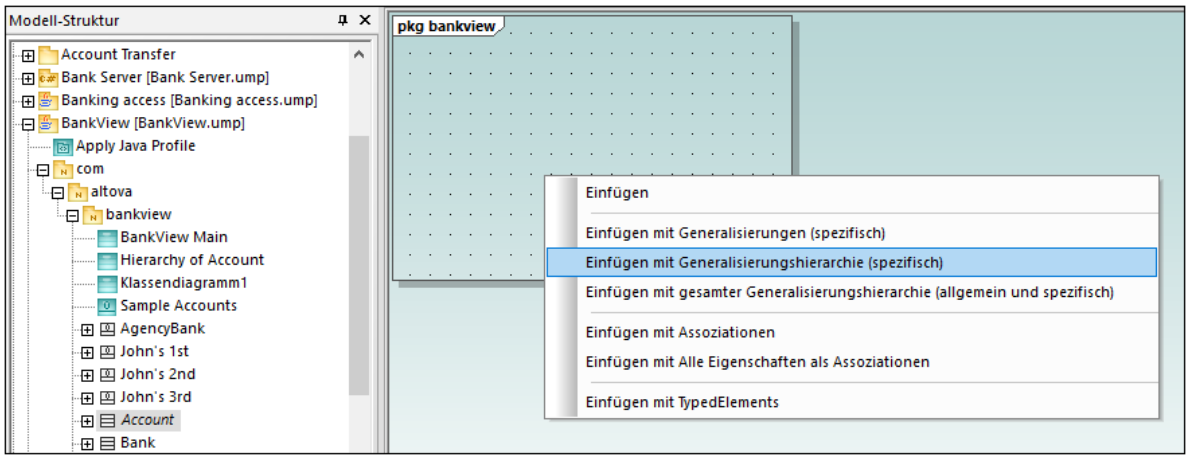
Im Modell bereits vorhandene Elemente können einzeln oder als Gruppe in ein Diagramm eingefügt werden. Um mehrere Elemente aus dem Fenster "Modell-Struktur" zu markieren, halten Sie die **Strg**-Taste gedrückt und klicken Sie auf die einzelnen Elemente. Zum Einfügen der Elemente in das Diagramm stehen zwei verschiedene Methoden zur Verfügung: links ziehen und rechts ziehen.

- **Links ziehen** (Gedrückt-halten der linken Maustaste und Loslassen der Maustaste im Diagramm): Fügt Elemente direkt an der Cursorposition ein. In diesem Fall werden automatisch alle Assoziationen, Abhängigkeiten usw., die zwischen den aktuell eingefügten und dem neuen Element bestehen, angezeigt.
- **Rechts ziehen** (Gedrückt-halten der rechten Maustaste und Loslassen der Maustaste im Diagramm): Öffnet ein Kontextmenü, aus dem Sie die Assoziationen und Generalisierungen, die angezeigt werden sollen, auswählen können.

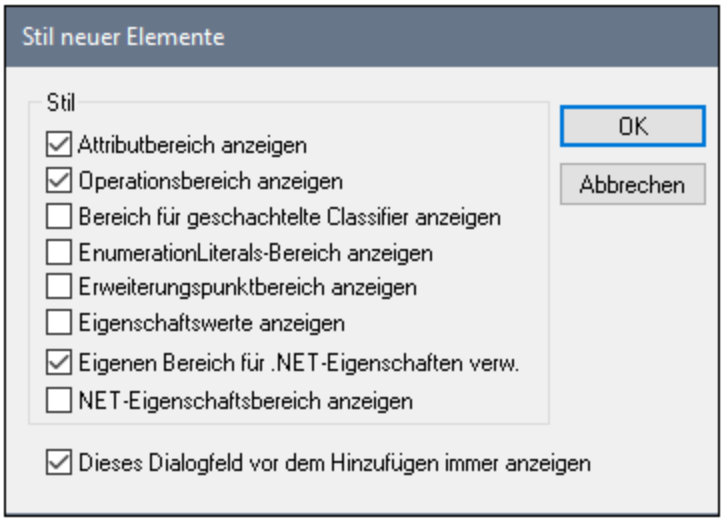
Angenommen, Sie möchten anhand einer bereits im Modell vorhandenen Klasse ein neues Klassendiagramm erstellen. Öffnen Sie dazu das Beispielprojekt **Bank_MultiLanguage.ump** aus dem folgenden Ordner: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**. Um z.B. das Diagramm "Account Hierarchy" in einem neuen Klassendiagramm zu replizieren, gehen Sie folgendermaßen vor:

1. Rechtsklicken Sie auf das **bankview**-Paket und wählen Sie **Neues Diagramm | Klassendiagramm**.
2. Suchen Sie in der Modell-Struktur die abstrakte Klasse `Account` und verwenden Sie den "**Rechts ziehen**"-Vorgang, um sie in das neue Diagramm zu platzieren. In diesem Beispiel soll die Klasse zusammen mit ihren abgeleiteten Klassen angezeigt werden. Wählen Sie dazu im Kontextmenü den Befehl **Einfügen mit Generalisierungshierarchie (spezifisch)**.

© 2016-2022 Altova GmbH



3. Aktivieren oder deaktivieren Sie die Kontrollkästchen für die Einträge, die in den Elementen angezeigt werden sollen.



4. Klicken Sie auf OK. Die Klasse `Account` und **ihre drei Unterklassen** werden alle auf dem Diagrammregister eingefügt. Die Generalisierungspfeile werden ebenfalls automatisch angezeigt. Um die Klassen automatisch im Diagramm anzuordnen, starten Sie den Befehl **Layout | Automatisches Layout | Hierarchisch**.

Hätten Sie anstelle des Befehls **Einfügen mit Generalisierungshierarchie (spezifisch)** den Befehl **Einfügen** gewählt, wäre die Klasse ohne abgeleitete Klassen zum Diagramm hinzugefügt worden. Beachten Sie, dass Sie die Generalisierungshierarchie auch später anzeigen können. Gehen Sie dazu folgendermaßen vor:

- Klicken Sie mit der rechten Maustaste im Diagramm auf die Klasse `Account` und wählen Sie im Kontextmenü den Befehl **Anzeigen | Generalisierungshierarchie**. Dadurch werden auch die abgeleiteten Klassen in das Diagramm eingefügt.

5.1.3 Umbenennen, Verschieben und Kopieren von Elementen

Sie können Elemente im Fenster Modell-Struktur[86] und innerhalb von Diagrammen desselben Typs ausschneiden, kopieren, umbenennen und verschieben. Diese Aktionen sind, falls anwendbar, auch zwischen Diagrammen unterschiedlichen Typs möglich. Sie können Elemente auch aus dem Fenster "Modell-Struktur" in ein Diagramm kopieren oder verschieben, vorausgesetzt das entsprechende Element ist gemäß der UML-Spezifikation in diesem Diagramm zulässig.

So benennen Sie ein Element um:

- Doppelklicken Sie auf den Elementnamen und bearbeiten Sie ihn.
- Klicken Sie alternativ dazu auf das Element und drücken Sie **F2**.

Die oben beschriebenen Verfahren gelten für jedes Fenster, in dem das Element angezeigt wird, darunter auch die Fenster "Modell-Struktur", "Eigenschaften" und das Diagrammfenster.

> Die Pakete "Root" und "Component View" werden immer im Fenster "Modell-Struktur" angezeigt und können nicht umbenannt oder gelöscht werden.

So kopieren oder verschieben Sie Elemente:

- Verwenden Sie die Windows-Standardbefehle **Ausschneiden, Kopieren** oder **Einfügen**. Diese Befehle können über die Tastaturkürzel (**Strg+X**, **Strg+C** bzw. **Strg+V**), über die entsprechenden Symbolleisten-Schaltflächen sowie über das Menü **Bearbeiten** aufgerufen werden.
- Alternativ dazu können Sie ein Element in ein Zielpaket (oder Element) ziehen. Durch das Ziehen eines Elements verschieben Sie es. Halten Sie die **Strg**-Taste beim Ziehen gedrückt, um es zu kopieren.

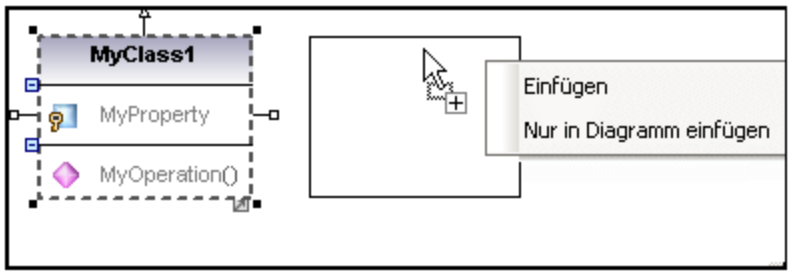
So können Sie etwa ein Mitglied einer Klasse in einem Diagramm in eine andere Klasse verschieben, indem Sie es mit der Maus von der Quellklasse in die Zielklasse ziehen. Um das Mitglied der Klasse zu kopieren, anstatt es zu verschieben, wählen Sie es aus und ziehen Sie es in die Zielklasse, während Sie die **Strg**-Taste gedrückt halten.

Wenn Sie eine Klasse in dasselbe Paket einfügen, wird eine laufende Nummer an die neue Klasse angehängt, z.B. "MyClass1". Wenn Sie eine Eigenschaft innerhalb derselben Klasse einfügen, wird ebenfalls eine laufende Nummer an das Ende der Eigenschaft angehängt, z.B. "MyProperty1". Dies gilt auch für andere Klassenmitglieder wie Operationen und Enumerationen oder beim Einfügen von Elementen in dasselbe Diagramm, vorausgesetzt das Diagramm gehört zum selben Paket wie die eingefügten Elemente.

Wenn Sie eine Klasse in ein anderes Paket einfügen, erhält die neue Klasse denselben Namen wie die ursprüngliche Klasse. Dies gilt auch für das Kopieren von Klassenmitgliedern (wie Eigenschaften, Operationen, usw.) in eine andere Klasse.

Standardmäßig wird jedes Element, das in ein Diagramm eingefügt wird, automatisch auch zum Modell hinzugefügt (und daher auch im Fenster "Modell-Struktur" angezeigt). Sie können ein Element aber auch nur in das aktuelle Diagramm kopieren, ohne es zum Modell hinzuzufügen. Kopieren Sie dazu zuerst das Element, klicken Sie mit der rechten Maustaste in das Diagramm und wählen Sie im Kontextmenü den Befehl **Nur in Diagramm einfügen**. Der Befehl **Nur in Diagramm einfügen** wird auch angezeigt, wenn Sie ein vorhandenes Element in dasselbe Diagramm ziehen, während Sie die **Strg**-Taste gedrückt halten.

© 2016-2022 Altova GmbH



Im obigen Beispiel erstellt **Einfügen** die neue Klasse im Diagramm und fügt sie auch zur Modell-Struktur hinzu, während mit **Nur in Diagramm einfügen** nur eine zweite Ansicht der Klasse im Diagramm angezeigt wird. Beachten Sie, dass es sich bei Kopien, die mit der zweiten Methode erstellt wurden, nur um zusätzliche Ansichten des Originalelements handelt, zu dem eine Verknüpfung erstellt wird, und nicht um eine eigenständige Kopie. (Wenn Sie z.B. eine Eigenschaft in der duplizierten Klasse umbenennen, wird diese Änderung automatisch auf die Originalklasse angewendet.)

5.1.4 Löschen von Elementen

Elemente können auf die folgenden Arten gelöscht werden:

- Über das Fenster "Modell-Struktur". Verwenden Sie diese Methode, wenn das Element sowohl aus dem Projekt als auch aus allen Diagrammen, in denen es vorkommt, gelöscht werden soll.
- Direkt über die Diagramme, in denen sie vorkommen. In diesem Fall können Sie auswählen, ob das Element nur aus dem Diagramm entfernt oder auch aus dem Modell (Projekt) gelöscht werden soll.

So löschen Sie Elemente aus dem Projekt und allen dazugehörigen Diagrammen (Methode 1):

1. Klicken Sie im Fenster "Modell-Struktur" auf das gewünschte Element. Halten Sie die **Strg**-Taste gedrückt, um mehrere Elemente auszuwählen.
2. Drücken Sie **Entf**.

So löschen Sie Elemente aus dem Projekt und allen dazugehörigen Diagrammen (Methode 2):

1. Öffnen Sie ein Diagramm und klicken Sie auf das gewünschte Element. Halten Sie die **Strg**-Taste gedrückt, um mehrere Elemente auszuwählen.
2. Drücken Sie **Entf**. Daraufhin wird ein Dialogfeld angezeigt, in dem Sie gefragt werden, ob Sie das Element sowohl aus dem Projekt als auch aus dem Diagramm löschen möchten.
3. Klicken Sie auf **Ja**. Das Element wird sowohl aus dem Diagramm als auch aus dem Projekt gelöscht.

So löschen Sie Elemente aus dem Diagramm, nicht aber aus dem Projekt:

1. Öffnen Sie ein Diagramm und klicken Sie auf das/die gewünschte/n Element/e. Halten Sie die **Strg**-Taste gedrückt, um mehrere Elemente auszuwählen.
2. Halten Sie die **Strg**-Taste gedrückt und drücken Sie **Entf**. Die Elemente werden aus dem Diagramm, nicht aber aus dem Projekt gelöscht.

Bevor Sie Elemente aus einem Projekt löschen, sollten Sie eventuell überprüfen, ob sie in einem Diagramm verwendet werden.

- Klicken Sie mit der rechten Maustaste auf ein Element in der Modell-Struktur und wählen Sie im Kontextmenü den Befehl **Element in allen Diagrammen anzeigen**.

Auf dieselbe Weise können Sie, während ein Diagramm geöffnet ist, schnell ein Element in der Modell-Struktur auswählen:

- Klicken Sie mit der rechten Maustaste auf ein Element im Diagramm und wählen Sie im Kontextmenü den Befehl **In Modell-Struktur auswählen**.
- Klicken Sie alternativ dazu auf das Element im Diagramm und drücken Sie **F4**.

5.1.5 Konvertieren von Elementen

Einige der Elemente unterstützen eine schnelle Konvertierung in eine andere Elementart. Dies kann sich als nützlich erweisen, wenn Sie z.B. begonnen haben, eine Klasse zu erstellen, diese später aber in eine Schnittstelle umwandeln möchten oder umgekehrt. Die folgenden Arten von Elementen unterstützen die Konvertierung in ein beliebiges anderes Element in der Liste:

- Klasse
- Schnittstelle
- Enumeration
- Primitivtyp
- Datentyp

Die oben aufgelisteten Elementarten können entweder über das Diagrammfenster[90] oder die Modell-Struktur[86] konvertiert werden.

So konvertieren Sie Elemente:

1. Öffnen Sie ein Diagramm, das Klassen, Schnittstellen, Enumerationen, Primitivtypen oder Datentypen enthält (z.B. ein Klassendiagramm). Navigieren Sie alternativ dazu in der Modell-Struktur zu einer dieser Elementarten.
2. Klicken Sie mit der rechten Maustaste auf das gewünschte Element (z.B. eine Klasse) und wählen Sie im Kontextmenü den Befehl **Konvertieren in | <Elementart>**.

Der Name des Elements bleibt nach der Konvertierung erhalten. Wenn möglich, bleiben auch die mit dem Element verknüpften Daten erhalten. So bleiben etwa bei einer Konvertierung von einer Schnittstelle in eine Klasse oder einer Klasse in eine Schnittstelle Daten wie Eigenschaften oder Operationen erhalten. Bei einer Konvertierung von einer Klasse oder Schnittstelle in eine Enumeration gehen hingegen Daten verloren. In einem solchen Fall können Sie den vorherigen Zustand des Elements, falls nötig, mit Hilfe des Befehls **Rückgängig** (**Strg+Z**) wiederherstellen.

5.1.6 Suchen und Ersetzen von Text

Sie können in jedem der folgenden Fenster nach Modellierungselementen, Diagrammen, Text, usw. suchen.:

- Diagrammfenster

© 2016-2022 Altova GmbH

- Fenster "Modell-Struktur"
- Fenster "Diagramm-Struktur"
- Fenster "Favoriten"
- Fenster "Dokumentation"
- Fenster "Meldungen"

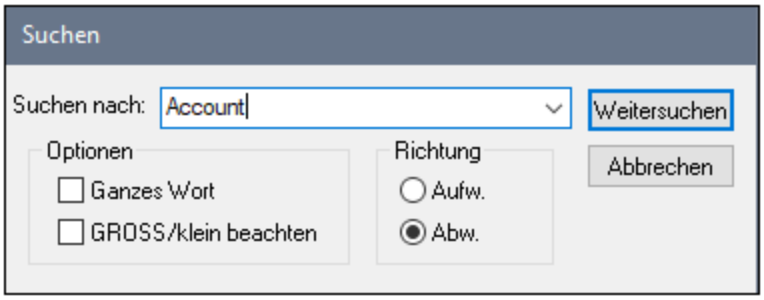
Die Suche wird dabei im Geltungsbereich des Fensters, in dem sich der Cursor gerade befindet, durchgeführt. Wenn Sie also in einem Diagramm nach Text suchen möchten, klicken Sie zuerst in das Diagramm. Wenn Sie nach einem Objekt im UModel-Projekt suchen möchten, klicken Sie zuerst in das Fenster "Modell-Struktur".

So suchen Sie nach Text oder Elementen:

1. Klicken Sie in das Fenster, in dem Sie Text suchen möchten.
2. Wählen Sie eine der folgenden Methoden:
   a. Geben Sie den Suchtext in das Textfeld der Hauptsymbolleiste ein und klicken Sie dann auf **Weitersuchen** oder drücken Sie **F3**. Um zum vorherigen Treffer zu gelangen, drücken Sie **Umschalt+F3**.



   b. Klicken Sie im Menü **Bearbeiten** auf **Suchen** (oder drücken Sie **Strg+F**).



Suchen und ersetzen

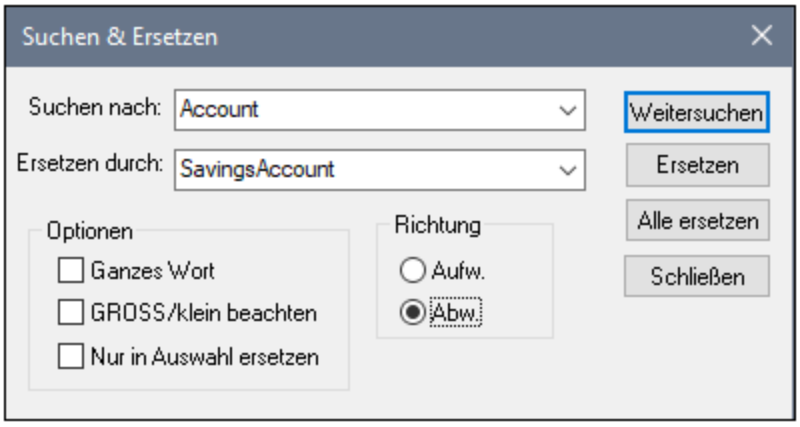
Sie können Text auch suchen und ersetzen (z.B. um Modellierungselemente schnell umzubenennen). Wenn das Element gefunden wird, wird es sowohl im Diagramm als auch in der Modell-Struktur markiert. Die Such- und Ersetzungsfunktion funktioniert in den folgenden Fenstern:

- Diagrammfenster
- Fenster "Modell-Struktur"
- Fenster "Diagramm-Struktur"
- Fenster "Favoriten"
- Fenster "Dokumentation"

So suchen und ersetzen Sie Text:

1. Klicken Sie in das Fenster, in dem Sie Text suchen und ersetzen möchten.
3. Wählen Sie eine der folgenden Methoden:
   a. Klicken Sie auf die Symbolleisten-Schaltfläche **Ersetzen**.

b. Wählen Sie im Menü **Bearbeiten** den Befehl **Ersetzen** (oder drücken Sie **Strg+H**).



5.1.7 Überprüfen, wo und ob Elemente verwendet werden

Manchmal ist es beim Navigieren durch die Elemente in der Modell-Struktur hilfreich zu sehen, wo oder ob das Element in einem Modelldiagramm vorkommt. Um herauszufinden, wo Elemente verwendet werden, wählen Sie eine der folgenden Methoden:

- Klicken Sie mit der rechten Maustaste auf das Element im Fenster "Modell-Struktur" und wählen Sie den Befehl **Element in allen Diagrammen anzeigen** (oder, falls ein Diagramm bereits geöffnet ist, auf **Element in aktivem Diagramm anzeigen**).

Sie können auch im gesamten Projekt oder einzelnen Paketen nach Elementen suchen, die in keinem Diagramm verwendet werden.

So suchen Sie im gesamten Projekt nicht verwendete Elemente:

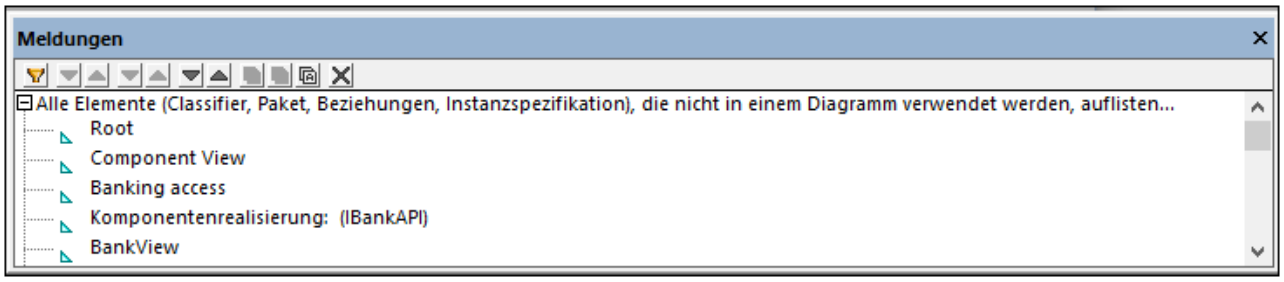
- Klicken Sie im Menü **Projekt** auf **In keinem Diagramm verwendete Elemente auflisten.**

So suchen Sie in einem bestimmten Paket nicht verwendete Elemente:

- Klicken Sie mit der rechten Maustaste auf das gewünschte Paket und wählen Sie den Befehl **In keinem Diagramm verwendete Elemente auflisten.**

Daraufhin wird im Fenster "Meldungen" eine Liste nicht verwendeter Elemente angezeigt. Beachten Sie, dass die nicht verwendeten Elemente für das aktuell ausgewählte Paket und seine Unterpakete angezeigt werden. Objekte innerhalb von Klammern sind Elemente, die über **Extras | Optionen |** Register "**Ansicht**" als in der Liste der nicht verwendeten Elemente anzuzeigende Elemente konfiguriert wurden.

© 2016-2022 Altova GmbH



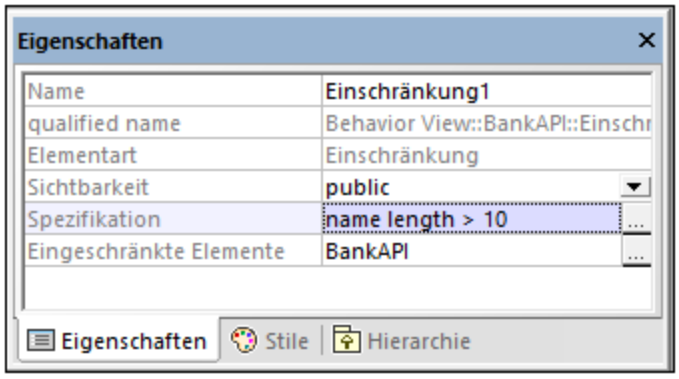
Klicken Sie im Fenster "Meldungen" auf den Elementnamen, um das Element in der Modell-Struktur zu finden.

5.1.8 Einschränken von Elementen

Es können für die meisten Modellelemente in UModel Einschränkungen definiert werden. Beachten Sie bitte, dass diese bei der Syntaxüberprüfung nicht berücksichtigt werden, da Einschränkungen nicht Teil der Java-Codegenerierung sind.

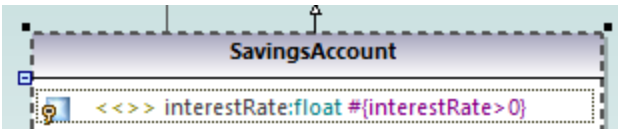
So schränken Sie ein Element (über die Modell-Struktur) ein:

1. Rechtsklicken Sie auf das gewünschte Element und wählen Sie **Neues Element | Einschränkungen | Einschränkung**.
2. Geben Sie den Namen der Einschränkung ein und drücken Sie die **Eingabetaste**.
3. Geben Sie im Fenster "Eigenschaften" in das Feld "Spezifikation" die Einschränkung ein, `name length > 10`.



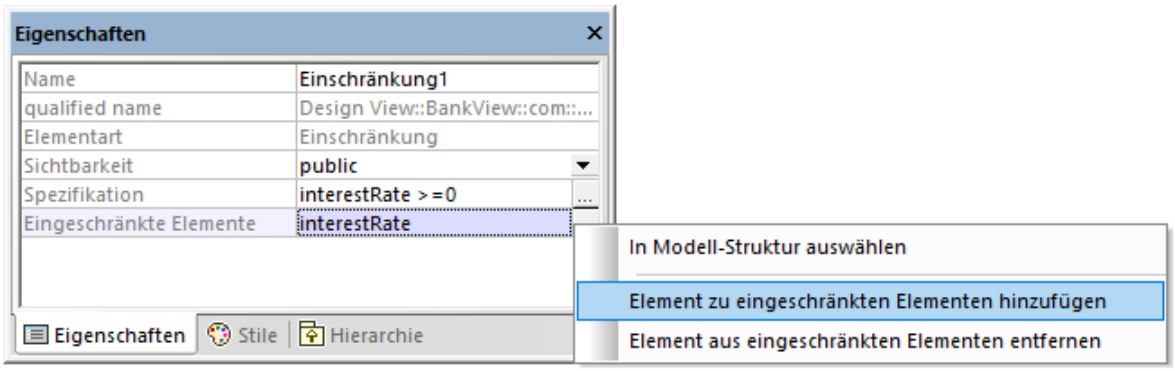
So schränken Sie ein Element (über ein Diagramm) ein:

1. Doppelklicken Sie auf das jeweilige Element, um es editieren zu können.
2. Geben Sie #, gefolgt vom Einschränkungstext innerhalb einer geschweiften Klammer ein z.B. `#{interestRate >=0}`.

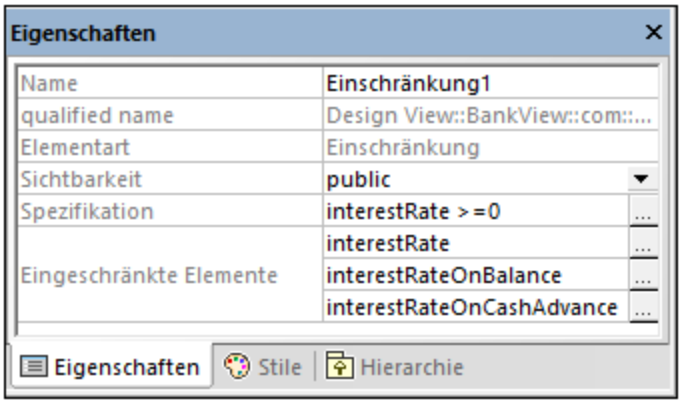


So weisen Sie eine Einschränkung mehreren Modellierungselementen zu:

1. Wählen Sie eine Einschränkung in der Modell-Struktur aus.
2. Klicken Sie im Fenster "Eigenschaften" mit der rechten Maustaste auf die Eigenschaft "eingeschränkte Elemente" und wählen Sie den Befehl **Element eingeschränkten Elementen hinzufügen**.



3. Wählen Sie das jeweilige Element aus, dem Sie die aktuelle Einschränkung zuweisen möchten. Halten Sie die **Strg**-Taste gedrückt, um mehrere Elemente auszuwählen.



Das Feld "Eingeschränkte Elemente" enthält die Namen der Modellierungselemente, denen es zugewiesen wurde. In der obigen Abbildung wurde etwa `Einschränkung1` den folgenden Eigenschaften zugewiesen: `interestRate`, `interestRateOnBalance`, `interestRateOnCashAdvance`.

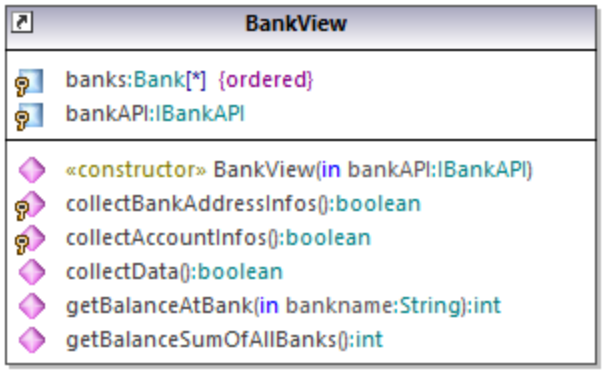
5.1.9 Erstellen von Hyperlinks zu Elementen

Sie können zwischen den meisten Modellierungselementen (mit Ausnahme von Linien) und jedem der folgenden Elemente Hyperlinks erstellen:

© 2016-2022 Altova GmbH

- Andere Elemente (entweder im Diagramm oder in der Modell-Struktur)
- Diagramme
- Externe Dateien, die nicht im Projekt enthalten sind (z.B. PDF-, Word- oder Excel-Dokumente, grafische Dateien, usw.)
- Webseiten

Ein einzelnes Element kann einen oder mehrere Hyperlinks jeder der oben aufgelisteten Arten haben. In einem Diagramm werden Elemente, die Hyperlinks enthalten, durch ein Hyperlink-Symbol neben dem Element (entweder in der rechten oder linken Ecke) gekennzeichnet. Um das Ziel des Hyperlink zu öffnen, klicken Sie mit der rechten Maustaste auf das Hyperlink-Symbol des Elements und wählen Sie das Ziel aus. Wenn nur ein Hyperlink definiert ist, können Sie auch einfach auf klicken und das Ziel direkt aufrufen.



Klasse, die Hyperlinks enthält

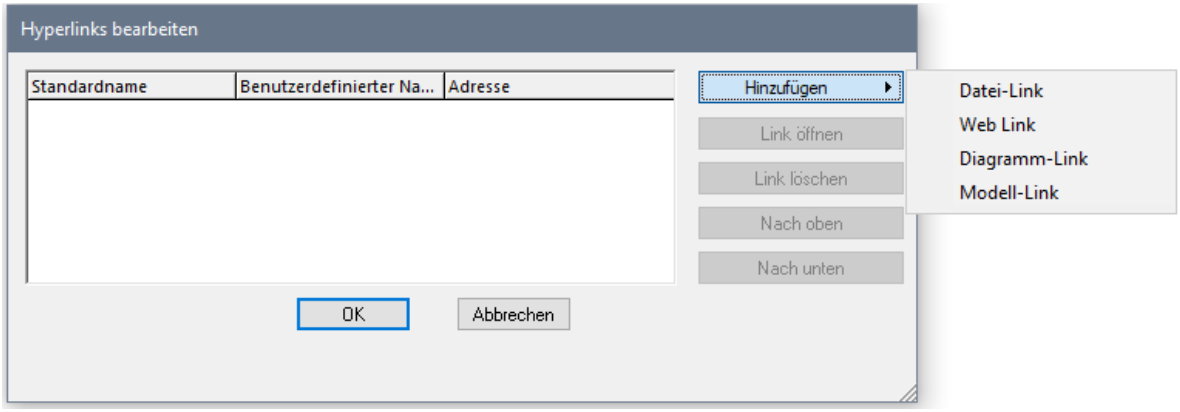
Tipp: Während Sie entweder über Hyperlinks oder auf andere Weise über die grafische Benutzeroberfläche von UModel navigieren, können Sie über die Symbolleisten-Schaltflächen **Zurück** bzw. **Weiter** einfach von einer Ansicht zur anderen weiter- oder zurückgehen.

Wenn Sie Quellcode oder Binärdateien in ein Modell importieren, können Sie automatisch Hyperlinks zwischen abhängigen Paketen und Diagrammen generieren, vorausgesetzt, Sie haben im Import-Dialogfeld die entsprechenden Einstellungen ausgewählt. Nähere Informationen dazu finden Sie unter Importieren von Quellcode 209 und Importieren von Java-, C#- und VB.NET-Binärdateien 225. Außerdem können Sie beim Generieren von UML-Dokumentation anhand des Projekts auswählen, ob Hyperlinks in die generierte Ausgabe inkludiert werden sollen, siehe Generieren von UML-Dokumentation 345.

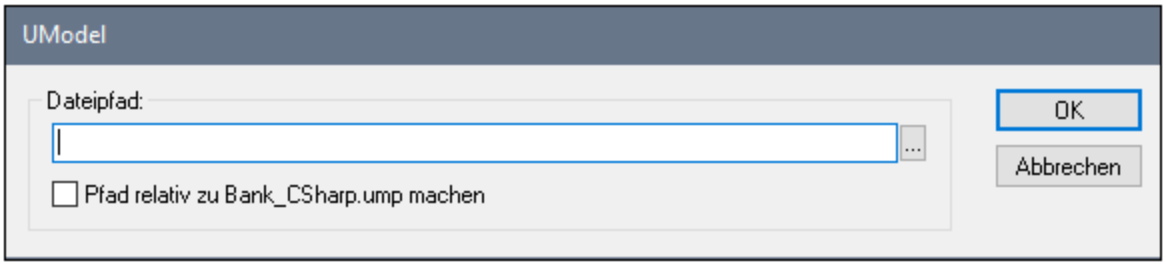
Sie können Hyperlinks nicht nur anhand der im Diagramm oder der Modell-Struktur angezeigten Elemente, sondern auch anhand von Text in Anmerkungen sowie anhand von Text im Fenster "Dokumentation" generieren. Eine Anleitung dazu finden Sie weiter unten.

So erstellen Sie einen Hyperlink anhand eines Elements:

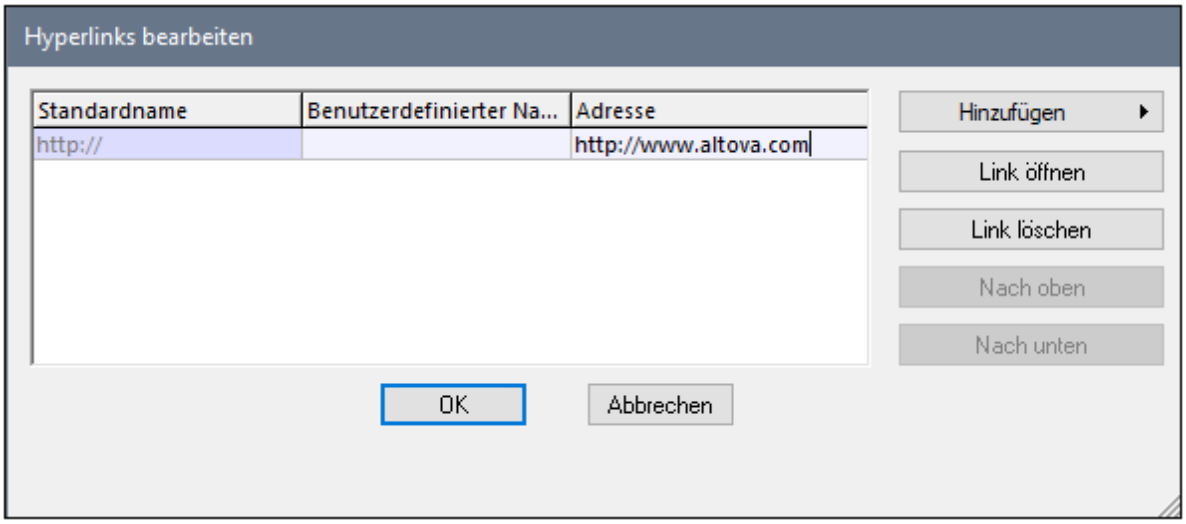
1. Klicken Sie mit der rechten Maustaste in einem Diagramm oder im Fenster "Modell-Struktur" auf ein Element und wählen Sie im Kontextmenü den Befehl **Hyperlinks | Hyperlinks einfügen/bearbeiten**.
2. Klicken Sie auf **Hinzufügen** und wählen Sie eine Hyperlinkart (Element-, Diagramm-, Datei- oder Web Link aus).



3. Wählen Sie eine der folgenden Methoden:
   - Um ein Diagramm oder einen Hyperlink zu erstellen, wählen Sie das Zielelement oder -diagramm aus, wenn Sie dazu aufgefordert werden.
   - Um einen Datei-Hyperlink zu erstellen, klicken Sie auf die Auslassungszeichen und navigieren Sie zur Zieldatei.



   - Um einen Weblink zu erstellen, geben Sie die gewünschte URL in die Spalte "Adresse" des Dialogfelds ein, z.B:



4. Geben Sie optional einen benutzerdefinierten Linknamen in die Spalte "Benutzerdefinierter Name" ein. Falls einer definiert ist, wird anstelle des Zielpfads (oder der Adresse) dieser Name auf der grafischen Benutzeroberfläche von UModel angezeigt.

© 2016-2022 Altova GmbH

So erstellen Sie einen Hyperlink in einer Anmerkung:

- Wählen Sie einen Textbereich in der Anmerkung aus, klicken Sie mit der rechten Maustaste darauf und wählen Sie im Kontextmenü den Befehl **Hyperlinks einfügen/bearbeiten**. Auf dieselbe Art und Weise kann auch im Fenster "Dokumentation" ein Hyperlink erstellt werden.



So ändern oder entfernen Sie einen Hyperlink:

- Klicken Sie mit der rechten Maustaste auf das Hyperlinksymbol im Element (oder auf den mit einem Hyperlink versehenen Text) und verwenden Sie den entsprechenden Befehl im Dialogfeld "Hyperlinks bearbeiten".

5.1.10 Dokumentieren von Elementen

Sie können Dokumentationskommentare folgendermaßen zu Modellierungselementen hinzufügen:

- Klicken Sie (entweder im Diagramm oder in der Modell-Struktur) auf das Element.
- Geben Sie im Fenster "Dokumentation" Text ein.

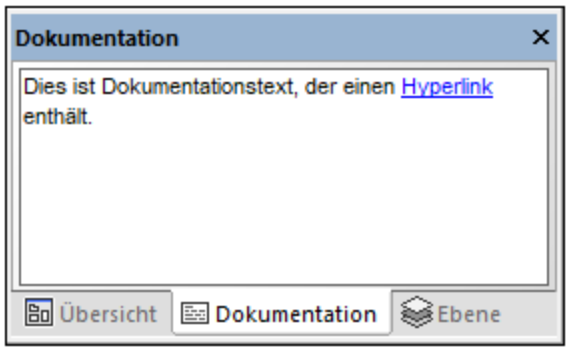
Dokumentationstext wird zusammen mit dem Projekt gespeichert.

Wenn ein Element ausgewählt ist, wird die Dokumentation dazu, falls vorhanden, immer im Fenster "Dokumentation" angezeigt. Sie können auch ein automatisches Layout für alle Elemente im Diagramm durchführen.

- Klicken Sie mit der rechten Maustaste auf das Element im Diagramm und wählen Sie im Kontextmenü den Befehl **Anzeigen | Anmerkende Kommentare**.

Dokumentations-Hyperlinks

Um im Fenster "Dokumentation" einen Hyperlink zu erstellen, wählen Sie Text im Fenster aus, klicken Sie mit der rechten Maustaste darauf und wählen Sie im Kontextmenü den Befehl **Hyperlinks einfügen/bearbeiten**. Beim Ziel des Hyperlink kann es sich um eine Webseite, ein Diagramm, eine Datei oder ein anderes Element handeln, siehe auch Erstellen von Hyperlinks zu Elementen 122.



Fenster "Dokumentation"

Codegenerierung und Dokumentationskommentare

Wenn Sie anhand von Klassendiagrammen Code generieren, können alle auf Klassen und deren Mitglieder angewendete Kommentare (in Klassendiagrammen) ebenfalls in den generierten Code exportiert werden. Aktivieren Sie dazu vor der Generierung von Programmcode das Kontrollkästchen **Dokumentation als JavaDocs schreiben** (für Java) oder **Dokumentation als DocComments verfassen** (für C#, VB.NET), siehe auch Codegenerierungsoptionen[186].

Ebenso können Codekommentare auch beim Reverse Engineering von Programmcode zu einem Modell in das Modell importiert werden. Aktivieren Sie dazu vor dem Reverse Engineering das Kontrollkästchen **JavaDocs als Dokumentation** (für Java) oder **DocComments als Dokumentation** (für C#, VB.NET), siehe auch Optionen für den Code-Import[212].

Informationen zu den Entsprechungen der Kommentare in Programmcode (oder XML-Schemas) für UModel-Kommentare finden Sie in den Entsprechungstabellen zu den einzelnen Sprachen:

- C#-Entsprechungen[253]
- VB.NET-Entsprechungen[273]
- Java-Entsprechungen[288]
- XML Schema-Entsprechungen[294]

5.1.11 Ändern des Stils von Elementen

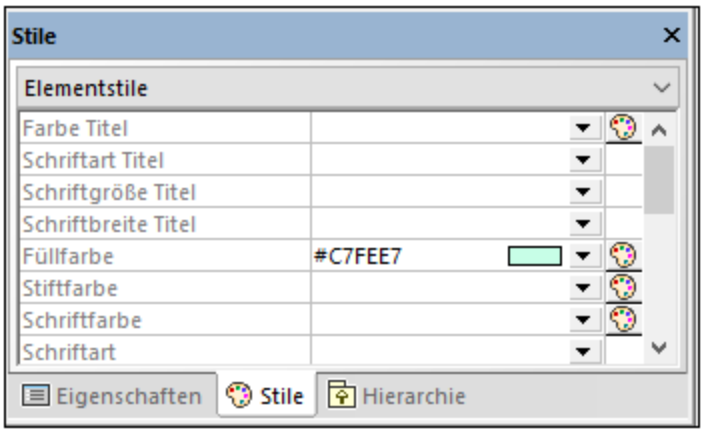
Sie können das Aussehen (den Stil) von Modellierungselementen, wie Farbe, Schriftgröße und -breite, Hintergrundfarbe, Liniendicke und andere Einstellungen ändern. Das Aussehen von Elementen kann auf verschiedenen Ebenen geändert werden: global für alle Elemente im Projekt, selektiv für alle Elemente derselben Familie (z.B. Klassen) oder für jedes einzelne Element. Informationen dazu, wie Sie den Stil des Diagramms selbst ändern können, finden Sie unter Ändern des Stils von Diagrammen[134].

Wenn Sie anstelle der konventionellen Elementdarstellung in Diagrammen Ihre eigenen Bilder verwenden möchten, können Sie Ihr Projekt um benutzerdefinierte Profile und Stereotypen erweitern. Nähere Informationen dazu finden Sie unter Beispiel: Anpassen von Symbolen und Stilen[487].

© 2016-2022 Altova GmbH

So ändern Sie das Aussehen von Elementen:

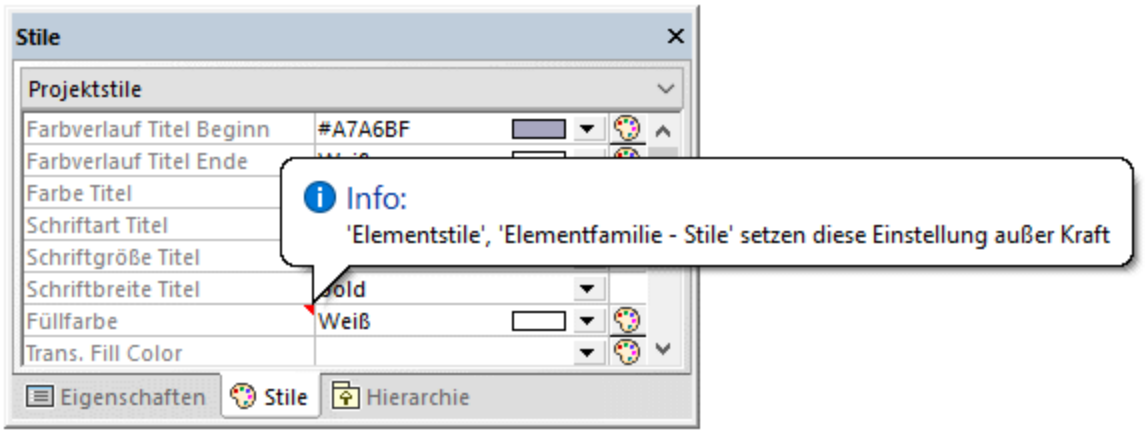
1. Klicken Sie auf das Element in einem Diagramm.
2. Am oberen Rand des Fensters "Stile" befindet sich eine Dropdown-Liste. Hier haben Sie folgende Auswahlmöglichkeiten:
   a. Um nur die Eigenschaften des aktuellen Elements zu bearbeiten, wählen Sie in der Liste den Eintrag "Elementstile".



   b. Um die Eigenschaften aller Elemente derselben Art (z.B. Klassen) zu bearbeiten, wählen Sie in der Liste "Elementfamilie - Stile" aus.
   c. Um die Eigenschaften aller Elemente global auf Projektebene zu bearbeiten, wählen Sie "Projektstile".
   d. Um die Eigenschaften aller Linien im Projekt, einschließlich aller Assoziations-, Abhängigkeits- und Realisierungslinien zu bearbeiten, wählen Sie "Linienarten". (Dieser Wert ist nur sichtbar, wenn es sich beim aktuell ausgewählten Element um eine Linie handelt.)
   e. Um im gesamten Projekt die Eigenschaften aller Elemente, bei denen es sich nicht um Linien handelt (die so genannten Knoten), zu bearbeiten, wählen Sie "Knotenstile". (Dieser Wert ist nur sichtbar, wenn es sich beim aktuell ausgewählten Element nicht um eine Linie handelt.)
3. Ändern Sie den Wert der gewünschten Eigenschaft (z.B. "Füllfarbe").

Spezifischere Stile setzen generischere Stile außer Kraft, d.h. Stile, die auf einer Elementebene angewendet werden, setzen solche, die auf Elementfamilienebene angewendet werden, außer Kraft. Ebenso setzen auf Elementfamilienebene angewendete Stile auf Projektebene angewendete Stile außer Kraft.

Wenn ein Stil außer Kraft gesetzt wird, erscheint in der rechten oberen Ecke der außer Kraft gesetzten Eigenschaft ein kleines rotes Dreieck. Bewegen Sie den Cursor über das Dreieck, um einen Tooltipp mit Informationen über diese Stilpriorität zu sehen.



Außer Kraft gesetzter Elementstil

© 2016-2022 Altova GmbH

5.2 Diagramme

5.2.1 Erstellen von Diagrammen

In Diagrammen wird visuell dargestellt, wie Modellierungselemente miteinander interagieren, welche Struktur, Abhängigkeiten, Hierarchie, usw. sie haben. Diagramme müssen zu einem Paket im Projekt gehören und daher im Fenster "Modell-Struktur" unterhalb von einem bestehenden Paket erstellt werden. Sie können Diagramme jederzeit durch Ziehen mit dem Maus von einem Paket in ein anderes verschieben.

So erstellen Sie ein neues Diagramm:

1. Klicken Sie im Fenster "Modell-Struktur" 86 mit der rechten Maustaste auf ein Paket.
2. Wählen Sie **Neues Diagramm | <Diagrammart>**.

Sie können ein neues Diagramm auch über das Fenster Diagramm-Struktur 90 erstellen. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie im Fenster "Diagramm-Struktur" mit der rechten Maustaste auf den Root-Node ("Diagramme").
2. Wählen Sie ein Paket für das Diagramm aush und klicken Sie auf **OK**.

Wenn das Diagrammfenster aktiv ist, werden in den Symbolleisten nur Modellierungselemente angezeigt, die auf die aktuelle Diagrammart angewendet werden können. Die Diagrammart wird im Fenster "Eigenschaften" angezeigt, wenn Sie in einen leeren Bereich des Diagramms klicken. Zusätzlich dazu wird die Art des Diagramms durch die folgenden Symbole gekennzeichnet.

Symbol	Beschreibung
	Aktivitätsdiagramm
	BPMN 1 (Business Process Modeling Notation)-Geschäftsprozessdiagramm
	BPMN 2-Geschäftsprozessdiagramm
	BPMN 2-Choreographiediagramm
	BPMN 2-Kollaborationsdiagramm
	Klassendiagramm
	Kommunikationsdiagramm
	Komponentendiagramm
	Kompositionsstrukturdiagramm
	Datenbankdiagramm
	Deployment-Diagramm

Symbol	Beschreibung
	Interaktionsübersichtsdiagramm
	Objektdiagramm
	Paketdiagramm
	Profildiagramm
	Protokoll-Zustandsdiagramm
	Sequenzdiagramm
	Zustandsdiagramm
	SysML-Diagramme (9 Diagrammtypen)
	Zeitverlaufsdiagramm
	Use Case-Diagramm
	XML-Schema-Diagramm

5.2.2 Generieren von Diagrammen

Anstatt Diagramme von Grund auf neu zu erstellen, können Sie bestimmte Diagramme auch automatisch anhand vorhandener Modellierungselemente oder anhand von Programmcode erstellen. In diesem Kapitel wird erläutert, wie Sie Diagramme anhand von vorhandenen Modellierungselementen generieren. Informationen darüber, wie Sie Diagramme anhand von Quellcode generieren, finden Sie hier:

- Generieren von Klassendiagrammen 462
- Generieren von Sequenzdiagrammen anhand von Quellcode 426
- Generieren von Paketdiagrammen 473

Um Diagramme anhand von vorhandenen Elementen zu generieren, klicken Sie mit der rechten Maustaste auf ein Element (z.B. Paket) in der Modell-Struktur und wählen Sie anschließend im Kontextmenü die Option **In neuem Diagramm anzeigen | <Option>**. Unten finden Sie einige Beispiele:

So erstellen Sie ein Diagramm, in dem der Inhalt eines vorhandenen Pakets angezeigt wird:

- Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf ein Paket und wählen Sie im Kontextmenü den Befehl **In neuem Diagramm anzeigen | Inhalt**.

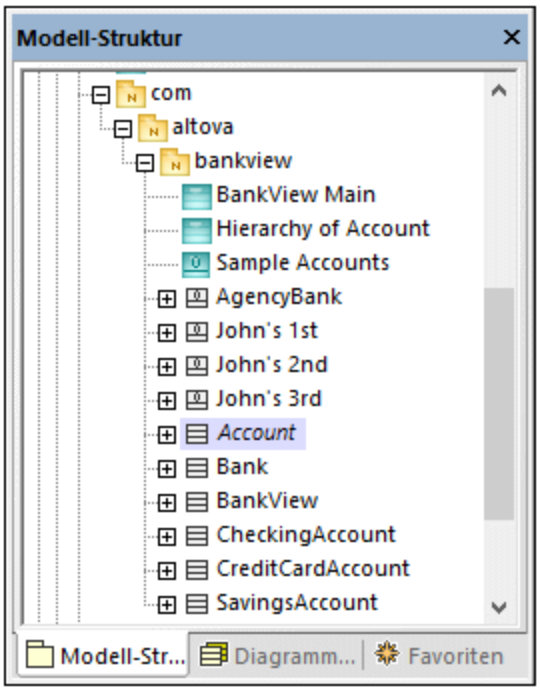
© 2016-2022 Altova GmbH

So erstellen Sie ein Diagramm, in dem die Abhängigkeiten eines vorhandenen Pakets angezeigt werden:

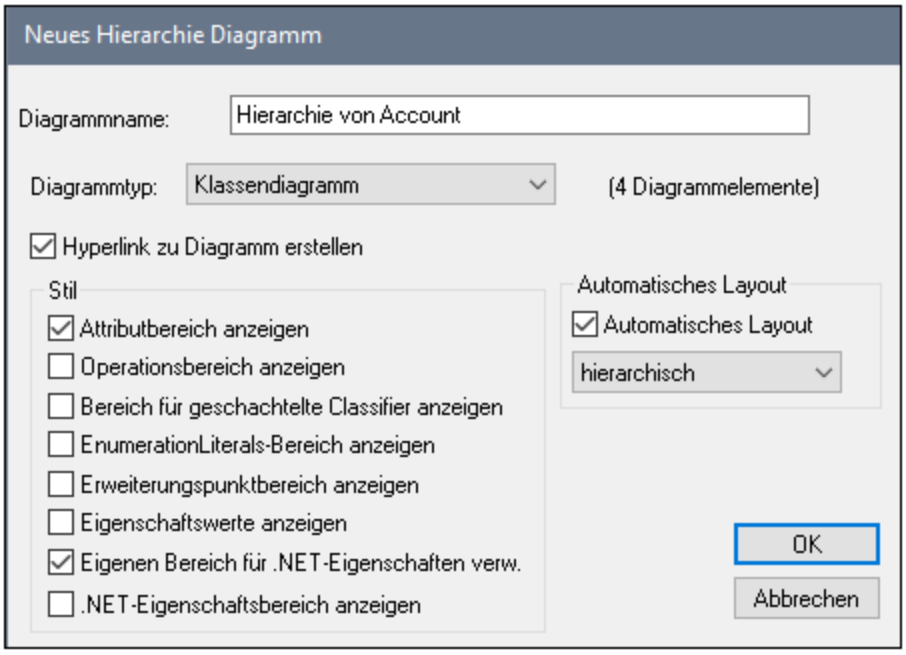
- Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf ein Paket und wählen Sie im Kontextmenü den Befehl **In neuem Diagramm anzeigen | Paketabhängigkeiten**.

So erstellen Sie ein Diagramm, in dem die Generalisierungshierarchie einer Klasse angezeigt wird:

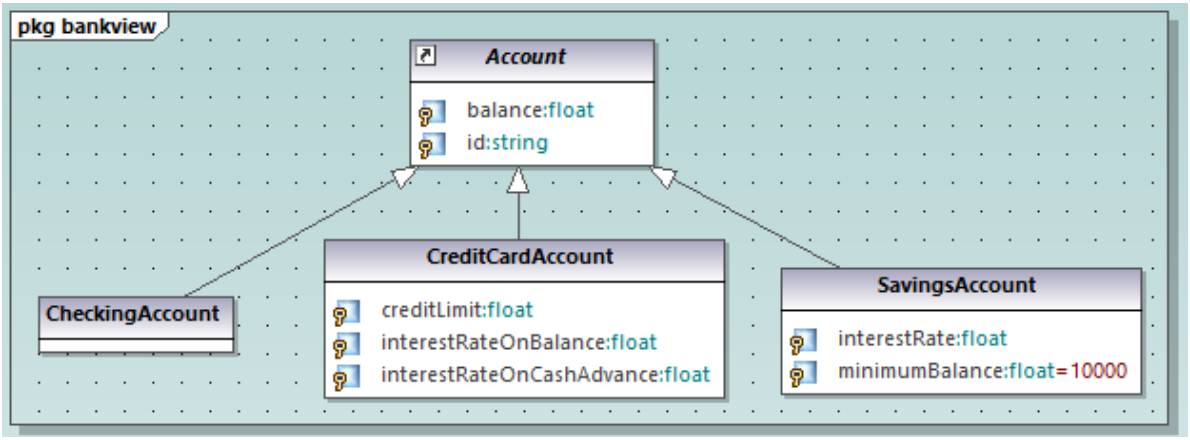
1. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf eine Klasse, die eine Generalisierungsbeziehung zu oder von anderen Klassen aufweist (z.B. auf die Klasse `Account` aus dem Beispielprojekt **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_CSharp.ump**).



2. Wählen Sie im Kontextmenü die Option **In neuem Diagramm anzeigen | Generalisierungshierarchie**. Daraufhin wird ein Dialogfeld angezeigt, in dem Sie die Einstellungen für das zu erstellende Diagramm, darunter auch den Diagrammtyp anpassen können. Beachten Sie den Text "N Diagrammelemente", der die Anzahl der Elemente angibt, die zum Diagramm hinzugefügt werden sollen. Im Beispiel unten wird als Diagrammtyp "Klassendiagramm" ausgewählt. Das Diagramm enthält vier Diagrammelemente (Klassen): die Klasse `Account` und die davon abgeleiteten Klassen.



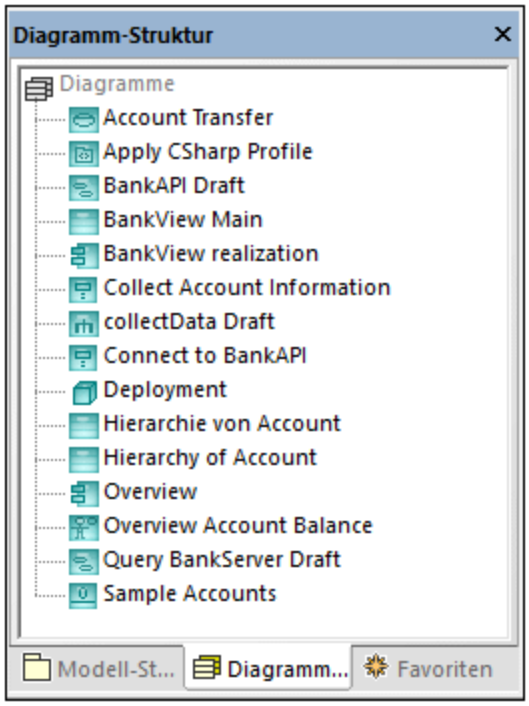
3. Klicken Sie auf **OK**. Das anhand der ausgewählten Optionen generierte Diagramm wird im Diagrammfenster geöffnet, z.B.:



© 2016-2022 Altova GmbH

5.2.3 Öffnen von Diagrammen

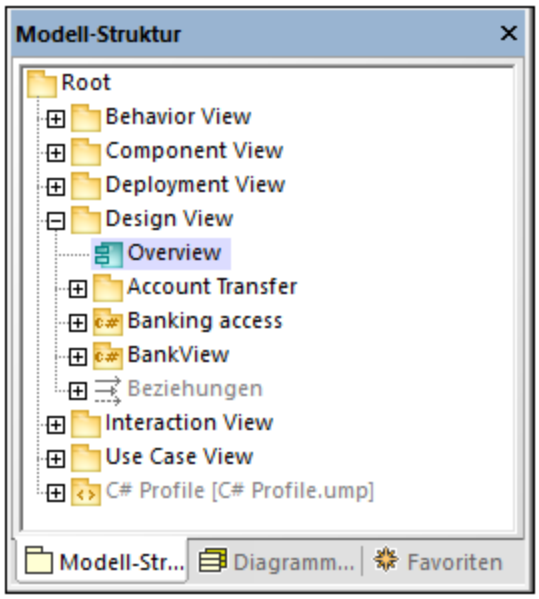
Wenn das UModel-Projekt Diagramme enthält, werden diese im Fenster "Diagramm-Struktur" angezeigt.



Fenster "Diagramm-Struktur"

Anmerkung: BStandardmäßig werden Diagramm im Fenster "Diagramm-Struktur" nach Typ gruppiert. Um nur Diagramme (ohne die übergeordnete Struktur) zu sehen, klicken Sie mit der rechten Maustaste in das Fenster und deaktivieren Sie im Kontextmenü die Option **Nach Diagrammtyp gruppieren**.

Die Diagramme werden auch im Fenster "Modell-Struktur" unterhalb der Pakete, zu denen sie gehören, angezeigt., z.B.:



So öffnen Sie ein vorhandenes Diagramm:

- Doppelklicken Sie im Fenster "Modell-Struktur" (oder im Fenster "Diagramm-Struktur" oder im Fenster "Favoriten") auf das Diagrammsymbol.
- Klicken Sie mit der rechten Maustaste auf das Diagramm und wählen Sie im Kontextmenü den Befehl **Diagramm öffnen**.

5.2.4 Löschen von Diagrammen

UModel-Diagramme können auf die folgenden Arten gelöscht werden:

- Klicken Sie im Fenster "Modell-Struktur" (oder im Fenster "Diagramm-Struktur" oder im Fenster "Favoriten") mit der rechten Maustaste auf das Diagramm und wählen Sie im Kontextmenü den Befehl **Löschen**.
- Klicken Sie in einem der oben erwähnten Fenster auf das Diagramm und drücken Sie die **Entf**-Taste.

Durch das Löschen eines Diagramms wird nur das Diagramm selbst aus dem Projekt gelöscht, nicht aber die Elemente. Um zu überprüfen, ob Elemente in einem Diagramm verwendet werden, klicken Sie mit der rechten Maustaste auf das gewünschte Paket und wählen Sie den Befehl **In keinem Diagramm verwendete Elemente auflisten**, siehe auch Überprüfen, wo und ob Elemente verwendet werden 120.

Wie Sie Elemente aus einem Diagramm oder Projekt löschen können, finden Sie unter Löschen von Elementen 117.

5.2.5 Ändern des Stils von Diagrammen

Sie können das Aussehen (den Stil) von Diagrammen, wie Hintergrundfarbe, Rasterfarbe, Rastergröße und Farbe sowie das Aussehen der Diagrammüberschrift ändern. Dabei können Sie entweder den Stil einzelner

 © 2016-2022 Altova GmbH

Diagramme im Projekt ändern oder dieselben Eigenschaften auf alle Diagramme im Projekt anwenden. Informationen darüber, wie Sie den Stil von Elemente in einem Diagramm ändern, finden Sie unter Ändern des Stils von Elementen 126.

Die Größe von Diagrammen ist durch die Elemente und ihre Anordnung definiert. Um die Größe eines Diagramms anzupassen, ziehen Sie ein Element an einen der Diagrammränder und die Größe wird automatisch angepasst.

So ändern Sie das Aussehen von Diagrammen:

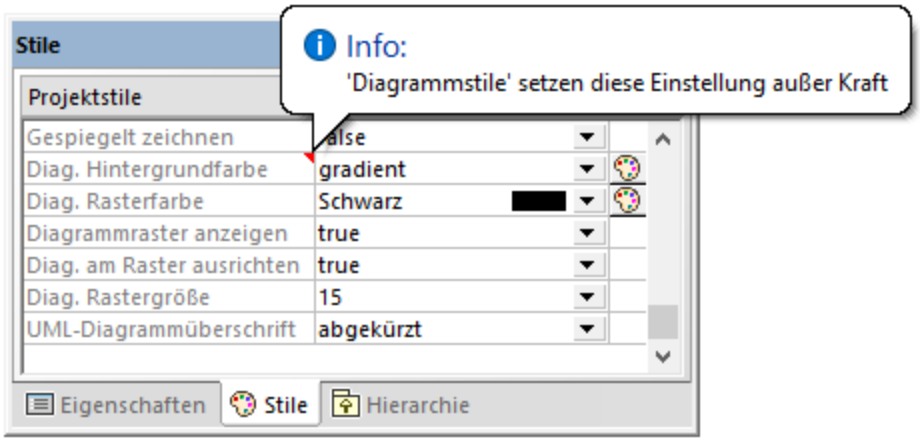
1. Öffnen Sie ein Diagramm (siehe Öffnen von Diagrammen 133).
2. Beachten Sie die Dropdown-Liste am oberen Rand des Fensters "Stile" und wählen Sie nach Bedarf eine der folgenden Methoden:
   a. Um nur die Eigenschaften des aktuellen Diagramms zu bearbeiten, wählen Sie in der Liste den Eintrag "Diagrammstile". Dieser Wert wird standardmäßig ausgewählt, wenn Sie auf den leeren Hintergrund des Diagramms klicken (d.h., wenn kein Diagrammelement ausgewählt ist).



   b. Um Änderungen auf alle Diagramme im Projekt anzuwenden, wählen Sie den Eintrag "Projektstile" aus. Scrollen Sie in diesem Fall bis zum Ende des Fensters "Stile", bis Sie die zu den Stilen gelangen, die auf Diagramme angewendet werden (d.h. zu den Stilen, die mit "Diag." beginnen).
3. Ändern Sie den Wert der gewünschten Eigenschaft (z.B. "Diag. Hintergrundfarbe").

> Stile, die auf Diagrammebene angewendet wird, setzen diejenigen, die auf Projektebene angewendet werden, außer Kraft.

Wenn ein Stil außer Kraft gesetzt wird, erscheint in der rechten oberen Ecke der außer Kraft gesetzten Eigenschaft ein kleines rotes Dreieck. Bewegen Sie den Cursor über das Dreieck, um einen Tooltipp mit Informationen über die Stilpriorität zu sehen



Außer Kraft gesetzter Diagrammstil

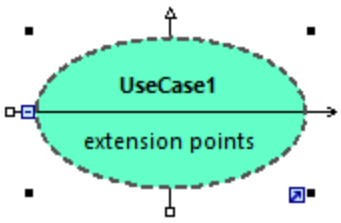
Die folgenden diagrammspezifischen Eigenschaften stehen in Form von Symbolleisten-Schaltflächen zur Verfügung. Wenn Sie die Eigenschaft im Fenster "Stile" ändern, wird der Status der Symbolleisten-Schaltfläche aktualisiert und umgekehrt.

	Raster anzeigen	Blendet das Diagrammraster ein bzw. aus.
	Diagrammüberschrift anzeigen	Blendet die Diagrammüberschrift ein bzw. aus.
	Am Raster ausrichten	Wenn diese Eigenschaft aktiviert ist, werden alle Elemente am Raster ausgerichtet. Bei deaktivierter Eigenschaft werden die Elemente beliebig unabhängig vom Diagrammraster positioniert.

5.2.6 Ausrichten von Modellierungselementen und Anpassen der Größe

Sie können die Größe von Elementen im Diagramm folgendermaßen anpassen:

1. Klicken Sie auf ein Element im Diagramm. Daraufhin werden rund um das Diagramm herum mehrere schwarze Punkte angezeigt..

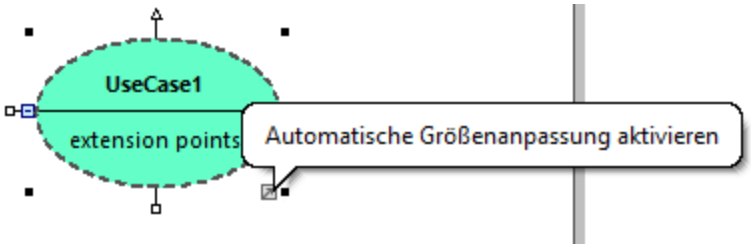


2. Ziehen Sie einen der schwarzen Punkte in die Richtung, in die das Diagramm vergrößert werden soll.

Um die Elementgröße wieder auf die Originalgröße zurückzusetzen, wählen Sie eine der folgenden Methoden:

© 2016-2022 Altova GmbH

- Klicken Sie an der rechten unteren Ecke des Elements auf das Symbol **Automatische Größenanpassung aktivieren**.



- Klicken Sie mit der rechten Maustaste auf ein Element im Diagramm und wählen Sie im Kontextmenü den Befehl **Größe automatisch anpassen.**.
- Wählen Sie ein oder mehrere Elemente aus. Klicken Sie im Menü **Layout** auf **Größe automatisch anpassen**.

Wenn in Diagramm mindestens zwei Modellierungselemente ausgewählt wurden, können diese aneinander ausgerichtet werden (so können z.B. beide waagrecht oder senkrecht aneinander ausgerichtet werden, oder in der Größe angeglichen werden). Die Befehle zum Ausrichten oder Anpassen der Größe von Elementen finden Sie im Menü **Layout** sowie in der gleichnamigen Symbolleiste.



Layout-Symbolleiste

Wenn Sie mehrere Elemente markieren, wird für die Aktion, die Sie daran ausführen (Ausrichten oder Größe anpassen), als Vorlage das **zuletzt** ausgewählte Element verwendet. Wenn Sie z.B. drei Klassenelemente auswählen und den Befehl **Breite angleichen** aufrufen, dann werden alle drei Elemente in der Breite an die zuletzt ausgewählte Klasse angepasst. Das zuletzt ausgewählte Element wird immer mit einer strichlierten Umrandung angezeigt.

In der folgenden Tabelle finden Sie die Befehle für die Ausrichtung von Elementen und die Anpassung ihrer Größe:

Symbol	Befehl	Anmerkungen
	Linksbündig	
	Rechtsbündig	
	Bündig oben	
	Bündig unten	
	Vertikal zentrieren	
	Horizontal zentrieren	
	Waagrecht verteilen	Dieser Befehl steht zur Verfügung, wenn drei oder mehr Elemente ausgewählt sind. Die waagrechten Abstände zwischen den

Symbol	Befehl	Anmerkungen
		ausgewählten Elementen werden dadurch gleich groß gemacht.
	Senkrecht verteilen	Dieser Befehl steht zur Verfügung, wenn drei oder mehr Elemente ausgewählt sind. Die senkrechten Abstände zwischen den ausgewählten Elementen werden dadurch gleich groß gemacht.
	Horizontal anordnen	Mit diesem Befehl werden alle ausgewählten Elemente im Diagramm neu positioniert, so dass sie waagrecht nebeneinander angeordnet werden.
	Vertikal anordnen	Mit diesem Befehl werden alle ausgewählten Elemente im Diagramm neu positioniert, so dass sie senkrecht übereinander angeordnet werden.
	Breite angleichen	
	Höhe angleichen	
	Größe angleichen	

Sie können auch ein automatisches Layout für alle Elemente im Diagramm durchführen. Gehen Sie dazu folgendermaßen vor:

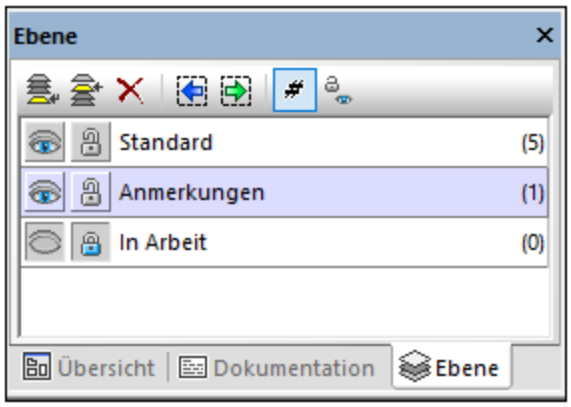
- Klicken Sie im Menü **Layout** auf **Automatisches Layout** und wählen Sie eine der folgenden Optionen aus: **Zentriert**, **Hierarchisch** oder **Block**.

Zentriert	Zeigt die Modellierungselemente in der Mitte zentriert an.
Hierarchisch	Zeigt die Elemente nach ihren hierarchischen Beziehungen an. So wird z.B. eine übergeordnete Klasse oberhalb der davon abgeleiteten Klassen angezeigt. Die Optionen für hierarchisches Layout können über das Menü **Extras | Optionen,** Register **Ansicht,** Gruppe **Hierarchisch anordnen** angepasst werden.
Block	Zeigt die Elemente rechteckig nach Elementgröße gruppiert an.

5.2.7 Hinzufügen von Ebenen zu Diagrammen

Standardmäßig besteht ein Diagramm aus einer einzigen Ebene, in der alle Elemente im Diagrammzeichenbereich angezeigt werden. Sie haben jedoch die Möglichkeit, mehrere Ebenen zu einem Diagramm hinzuzufügen. Mit Hilfe von Ebenen können Sie Modellierungselemente im selben Diagramm logisch gruppieren und dadurch separat halten. So können Sie zusätzlich zur Standardebene einige zusätzliche Ebenen z.B. für Anmerkungen mit internen Informationen oder nicht fertige Klassen erstellen. Ebenen können über das Fenster "Ebene" angezeigt und verwaltet werden.

© 2016-2022 Altova GmbH



Fenster "Ebene"

In der Abbildung oben wurden drei Ebenen im Diagramm definiert. Die Ebene "Anmerkungen" ist derzeit ausgewählt. Die dritte Ebene "In Arbeit" ist derzeit gesperrt. Die Zahl, die rechts in Klammern neben jeder Ebene angezeigt wird, zeigt an, wie viele Elemente jede Ebene hat.

Sie können jeder Ebene jedes beliebige UML-Element zuweisen. Standardmäßig werden neue Elemente zur aktiven Ebene hinzugefügt. Diese Ebene erscheint im Fenster "Ebenen" markiert. Wenn alle Ebenen sichtbar sind, können Sie zwischen Elementen auf verschiedenen Ebenen Beziehungen wie z.B. Assoziationen, Generalisierungen, usw. erstellen.

Beim Drucken von Diagrammen oder Speichern der Diagramme in einem Bild, werden nur Elemente aus den gerade sichtbaren Ebenen gedruckt. Die maximale Anzahl der Ebenen pro Diagramm beträgt 20.

Das Fenster "Ebene" enthält die folgenden Schaltflächen:

Symbol	Befehl	Anmerkungen
	**Ebene anhängen**	Hängt eine neue Ebene an die Liste der aktuellen Ebenen an und weist ihr einen Standardnamen zu, den Sie sofort oder über die Kontextmenüoption "Umbenennen" ändern können.
	**Ebene einfügen**	Fügt eine neue Ebene oberhalb der aktiven Ebene zur Ebenenliste hinzu.
	**Ebene löschen**	Löscht die aktive Ebene. Bevor die Ebene gelöscht wird, wird ein Dialogfeld angezeigt, in dem Sie gefragt werden, wohin die Elemente der aktiven Ebene (falls vorhanden) verschoben (zusammengeführt) werden sollen.
	**Fokus auf vorhergehendes Element in aktiver Ebene**	Wählt das vorherige Element auf der aktiven Ebene aus. Dieser Befehl ist nur aktiv, wenn die Ebene Elemente enthält.
	**Fokus auf nächstes Element in aktiver Ebene**	Wählt das nächste Element auf der aktiven Ebene aus. Dieser Befehl ist nur aktiv, wenn die Ebene Elemente enthält.

Symbol	Befehl	Anmerkungen
	**Anzahl der Ebenenelemente**	Blendet die Anzahl der Elemente auf den einzelnen Ebenen ein bzw. aus.
	**Zustand aller Ebenen zurücksetzen**	Setzt alle Ebenen in den Zustand "sichtbar" und "entsperrrt" zurück.

Einige der obigen Befehle stehen auch als Kontextmenübefehle zur Verfügung, wenn Sie mit der rechten Maustaste in das Fenster "Ebene" klicken.

So verschieben Sie Elemente von einer Ebene auf eine andere:

- Klicken Sie mit der rechten Maustaste auf das Element im Diagramm und wählen Sie im Kontextmenü den Befehl **Ebene | <Ebenenname>**. Dieser Befehl kann auch nach Auswahl mehrerer Elemente angewendet werden. In diesem Fall werden alle auf die gewünschte Ebene verschoben.
- Wählen Sie alternativ dazu ein oder mehrere Elemente im Diagramm aus und ziehen Sie diese mit der Maus auf die gewünschte Ebene im Fenster "Ebene".
- Um alle Elemente einer Ebene auf eine andere zu verschieben, klicken Sie mit der rechten Maustaste auf die Ebene, und wählen Sie im Kontextmenü den Befehl **Zusammenführen zu | <Ebenenname>**.

So können Sie einzelne oder mehrere Ebenen auf einmal einblenden, ausblenden oder sperren:

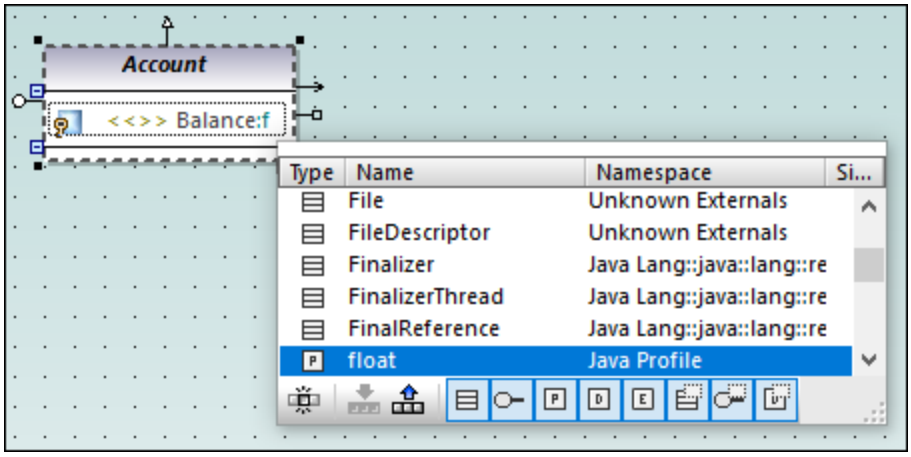
- Klicken Sie mit der rechten Maustaste in das Fenster "Ebene" und wählen Sie den Befehl **Anzeigen, Ausblenden** bzw. **Sperren**. Mit Hilfe der Untermenübefehle **Ausgewählte Ebene** und **Andere** können Sie den Befehl entweder auf die aktive Ebene oder auf alle Ebenen mit Ausnahme der gerade ausgewählten anwenden.
- Klicken Sie alternativ dazu mit der rechten Maustaste auf die Ebene und wählen Sie den Befehl **Ein-/Ausblenden** bzw. **Sperre ein/aus**. Dadurch wird/werden die zuvor angezeigte/n Ebene/n ausgeblendet, wenn sie zuvor angezeigt wurden bzw. gesperrt, wenn sie zuvor entsperrt war/en (und umgekehrt).

5.2.8 Typ-Autokomplettierung in Klassen

Die Autokomplettierung von Datentypen ist in UModel standardmäßig beim Hinzufügen von Operationen und Attributen zu einer Klasse aktiviert. Auf diese Art können Sie den Datentyp der Operation oder Eigenschaft direkt im Diagramm definieren, z.B.:

1. Klicken Sie mit der rechten Maustaste auf eine Klasse und wählen Sie im Kontextmenü den Befehl **Neu | Operation**.
2. Geben Sie nach den doppelten spitzen Klammern << >> den Namen der Operation, gefolgt von einem Doppelpunkt (:) ein.
3. Daraufhin wird ein Autokomplettierungsfenster geöffnet.

© 2016-2022 Altova GmbH



Autokomplettierungsfenster

Das Autokomplettierungsfenster bietet die folgenden Funktionalitäten:

- Durch Klick auf einen Spaltennamen wird das Fenster in auf- oder absteigender Reihenfolge nach diesem Attribut sortiert.
- Durch Ziehen der rechten unteren Ecke können Sie die Größe des Fensters anpassen.
- Der Inhalt des Fensters kann durch Auswahl der entsprechenden Filter (Kategorien) am unteren Rand des Fensters gefiltert werden: Klasse, Schnittstelle, Primitivtyp, Datentyp, Enumeration, Klassen-Vorlage, Schnittstellen-Vorlage, Datentyp-Vorlage.

So aktivieren Sie nur einen der Filter auf einmal:

- Klicken Sie auf die Schaltfläche **Einzelmodus** . In der obigen Abbildung sehen Sie das Autokomplettierungsfenster im "Mehrfachmodus", d.h. alle Filter sind aktiviert. Die Schaltfläche "Einzelmodus" ist deaktiviert.

So aktivieren oder deaktivieren Sie alle Filter auf einmal:

- Klicken Sie auf **Alle Kategorien definieren** bzw. **Alle Kategorien löschen** .

So deaktivieren Sie die Autokomplettierung:

1. Klicken Sie im Menü **Extras** auf **Optionen** und anschließend auf das Register **Diagrammbearbeitung**.
2. Deaktivieren Sie das Kontrollkästchen **Automatische Eingabehilfe aktivieren.**

So rufen Sie die Autokomplettierung bei Bedarf auf (wenn sie deaktiviert ist):

1. Stellen Sie sicher, dass sich der Cursor innerhalb von einem Attribut, einer Operation oder Klasse hinter dem Doppelpunkt (:) befindet.
2. Drücken Sie **Strg+Leerzeichen**.

5.2.9 Vergrößern und Verkleinern von Diagrammen

Um ein Diagramm auf dem Bildschirm zu vergrößern bzw. zu verkleinern, wählen Sie eine der folgenden Methoden:

- Rufen Sie den Menübefehl **Ansicht | Vergrößern (Strg+Umschalt+I)** oder **Ansicht | Verkleinern (Strg+Umschalt+O)** auf.
- Wählen Sie in der Symbolleiste Vergrößern/Verkleinern einen vordefinierten Prozentwert aus.



- Halten Sie die Strg-Taste gedrückt, während Sie das Mausrad drehen.

So passen Sie den Diagrammbereich an die Größe des sichtbaren Fensters an:

- Rufen Sie den Menübefehl **Ansicht | An Fenstergröße anpassen** auf (oder klicken Sie auf die Symbolleisten-Schaltfläche **An Fenstergröße anpassen**).

© 2016-2022 Altova GmbH

5.3 Beziehungen

5.3.1 Erstellen von Beziehungen

Für eine Beziehung werden normalerweise zwei Elemente benötigt, daher muss Ihr Diagramm die Elemente, zwischen denen Sie Beziehungen hinzufügen möchten, bereits enthalten. Beziehungen können folgendermaßen erstellt werden:

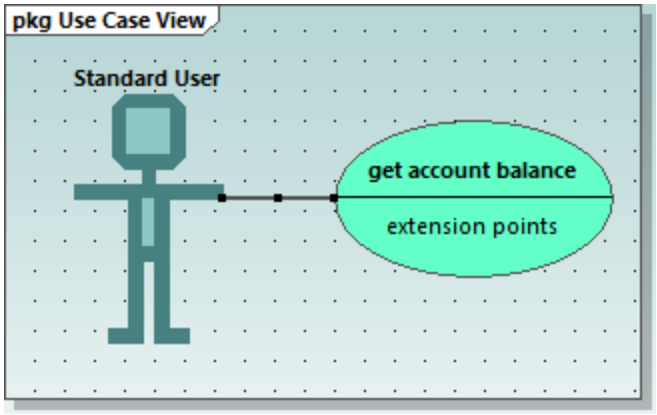
1. Über eine Symbolleisten-Schaltfläche für die gewünschte Beziehung (z.B. Assoziation).
2. Über die Ziehpunkte, die angezeigt werden, wenn Sie auf ein Element im Diagramm klicken.

Erstellen von Beziehungen mit Hilfe von Symbolleisten-Schaltflächen

Wenn im Hauptfenster von UModel ein Diagrammfenster aktiv (im Fokus) ist, werden in der Symbolleiste alle von diesem Diagramm unterstützten Elemente und Beziehungen angezeigt. So bietet etwa ein Klassendiagramm Symbolleisten-Schaltflächen für alle unterstützten Beziehungen. Dazu gehören etwa Assoziation , Collection-Assoziation , Aggregation , Komposition , Realisierung , Generalisierung und andere. Ein Use Case-Diagramm bietet z.B. Symbolleisten-Schaltflächen für Assoziationen , Generalisierungen sowie Include- und Extend- Beziehungen.

In der Anleitung unten wird beschrieben, wie Sie eine Assoziationsbeziehung zwischen einem Akteur und einem Use Case (Anwendungsfall) erstellen. Auf dieselbe Art können Sie bei Bedarf auch andere Beziehungen erstellen.

1. Klicken Sie auf ein Element im Diagramm (Akteur "Standardbenutzer" in der Abbildung unten).
2. Klicken Sie auf die Symbolleisten-Schaltfläche für die gewünschte Beziehung (in diesem Fall Assoziation).
3. Platzieren Sie die Maus über "Standardbenutzer" und ziehen Sie sie auf ein Zielelement (den Use Case "get account balance"). Beachten Sie, dass das Zielelement grün markiert wird und die Beziehung nur akzeptiert, wenn diese gemäß der UML-Spezifikation sinnvoll ist.



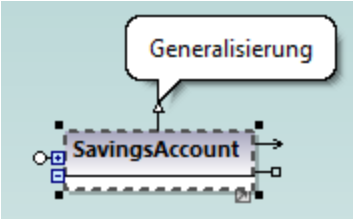
Assoziation in einem Use Case-Diagramm

Erstellen von Beziehungen mit Hilfe von Ziehpunkten

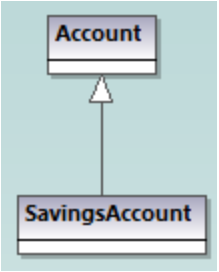
Wenn Sie auf ein Element in einem Diagramm klicken, werden eventuell links, rechts, oberhalb und unterhalb des Elements mehrere Ziehpunkte angezeigt. Die Ziehpunkte werden nur bei Elementen angezeigt, die Beziehungen unterstützen. Jeder Ziehpunkt entspricht einer Beziehungsart. Klassenelemente haben z.B. die folgenden Ziehpunkte:

- Schnittstellenrealisierung
- Generalisierung
- Assoziation
- Collection-Assoziation

Um zu sehen, für welche Art von Beziehung ein Ziehpunkt verwendet wird, platzieren Sie den Cursor über den Ziehpunkt. In der Abbildung oben dient der ausgewählte obere Ziehpunkt z.B. zum Erstellen einer Generalisierungsbeziehung.



Um die Beziehung zu erstellen, klicken Sie auf den Ziehpunkt und ziehen Sie den Cursor über das gewünschte Elemente. Dadurch wird die entsprechende Beziehung (in diesem Fall eine Generalisierung) erstellt.



Generalisierungsbeziehung zwischen zwei Klassen

5.3.2 Ändern des Stils von Linien und Beziehungen

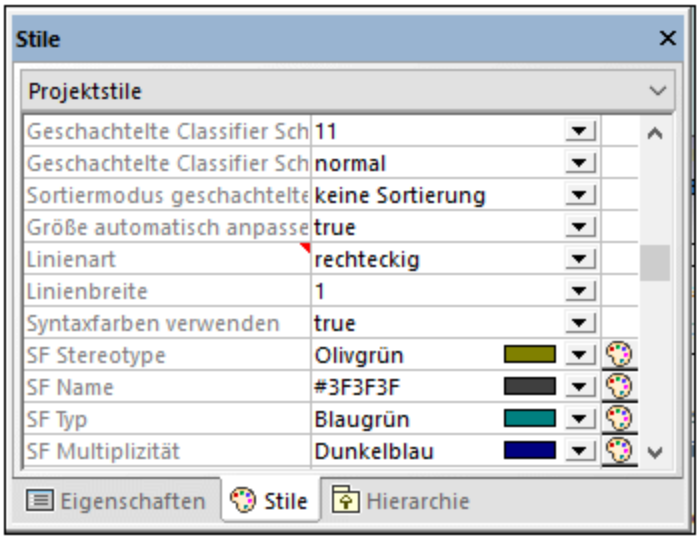
Sie können die Dicke, Farbe und den Krümmungsstil von Linien über das Fenster "Stile" ändern. Außerdem können Sie Text (Beschriftungen) zu Beziehungen hinzufügen, Beschriftungen neu positionieren und diese im Diagramm entweder für jede Beziehung einzeln oder alle gemeinsam ein- und ausblenden.

Anmerkung: Achten Sie in der Anleitung unten auf den Unterschied zwischen "Linien" (jede Linie im Diagramm) und "Beziehungen" wie Assoziation, Generalisierung, Komposition, usw. Alle Beziehungen sind Linien, dies gilt aber nicht für den umgekehrten Fall. So ist ein Kommentar oder eine Anmerkung nur eine Linie, nicht aber eine Beziehung.

© 2016-2022 Altova GmbH

So ändern Sie Linieneigenschaften:

1. Klicken Sie im Diagramm auf eine Linie.
2. Definieren Sie im Fenster "Stile" die gewünschte Eigenschaft (z.B. "Linienbreite").



Die für die Eigenschaft "Linienart" verfügbaren Werte stehen auch als Befehle im Menü **Layout | Linienart** und als Symbolleisten-Schaltflächen zur Verfügung. Wenn Sie diese Eigenschaft ändern, wird die entsprechende Symbolleiste aktiv und umgekehrt.

	Orthogonale Linie	Eine Linie, die nur im rechten Winkel abgeknickt wird.
	Gerade Linie	Eine Linie dieser Art bildet eine gerade Verbindung ohne Wegpunkte zwischen zwei Elementen.
	Benutzerdefinierte Linie	Eine Linie dieser Art kann in jedem Winkel gekrümmt werden. Um die Linie zu verschieben, ziehen Sie einen beliebigen Wegpunkt (kleine schwarze Punkte) der Linie. Um neue Wegpunkte zu erstellen, klicken Sie an eine Stelle zwischen zwei Wegpunkten und ziehen Sie die Linie. Um Wegpunkte zu löschen, ziehen Sie einen Wegpunkt direkt auf einen vorhandenen.

Linienarten können wie andere Elementstile für jede einzelne Linie oder auf einer allgemeineren Ebene (z.B. auf Projektebene) definiert werden. Der spezifischere Stil setzt den allgemeineren außer Kraft. Wenn ein Stil außer Kraft gesetzt wurde, wird dies durch ein rotes Dreieck neben der betreffenden Eigenschaft im Fenster "Stile" angezeigt, siehe auch Ändern des Stils von Elementen 126.

So fügen Sie Beschriftungstext zu einer Beziehung hinzu:

- Klicken Sie auf eine Beziehung im Diagramm und geben Sie Text ein.

So verschieben Sie Beschriftungstext:

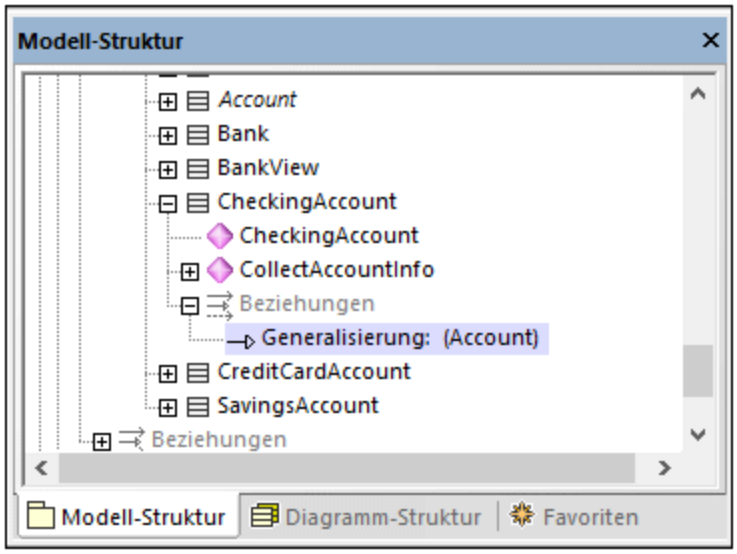
- Klicken Sie auf die Beschriftung und ziehen Sie sie an eine andere Stelle im Diagramm.
- So verschieben Sie die Beschriftung wieder an die Standardposition: Klicken Sie mit der rechten Maustaste auf die Beziehung und wählen Sie im Kontextmenü den Befehl **Textlabels | Textlabels neu positionieren.**
- So positionieren Sie mehrere Beschriftungen gleichzeitig neu: Wählen Sie eine oder mehrere Beziehungen im Diagramm aus und starten Sie den Menübefehl **Layout | Textlabels neu positionieren.**

So blenden Sie den Beschriftungstext ein oder aus:

- Klicken Sie mit der rechten Maustaste auf die Beziehung und wählen Sie im Kontextmenü den Befehl **Textlabels | Alle Textlabels anzeigen/ausblenden**.

5.3.3 Anzeigen von Elementbeziehungen

Standardmäßig werden die Beziehungen eines Elements in der Modell-Struktur unter dem jeweiligen Element angezeigt. So hat etwa die unten gezeigte Klasse `CheckingAccount` eine Generalisierungsbeziehung zur Klasse `Account`:



Beziehung im Fenster "Modell-Struktur"

Anmerkung: Um Beziehungen aus dem Fenster "Modell-Struktur" auszublenden, klicken Sie mit der rechten Maustaste in das Fenster und deaktivieren Sie die Option **Beziehungen in Struktur anzeigen.**

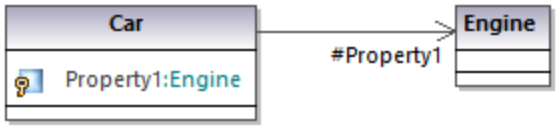
Um die Beziehungen eines Elements im Diagramm anzuzeigen, klicken Sie mit der rechten Maustaste auf das Element im Diagramm und wählen Sie im Kontextmenü die Option **Anzeigen | <Beziehungsart>**.

© 2016-2022 Altova GmbH

5.3.4 Assoziationen

Eine Assoziation ist eine begriffliche Verbindung zwischen zwei Elementen. Assoziationsbeziehungen werden wie alle anderen Beziehungen in UModel erstellt, siehe Erstellen von Beziehungen 143.

Wenn Sie eine Assoziation zwischen zwei Klassen erstellen, wird in die Ursprungsklasse automatisch ein neues Attribut eingefügt. Wenn Sie z.B. eine Assoziation zwischen den Klassen `Car` und `Engine` erstellen, wird eine Eigenschaft vom Typ `Engine` zur Klasse `Car` hinzugefügt.



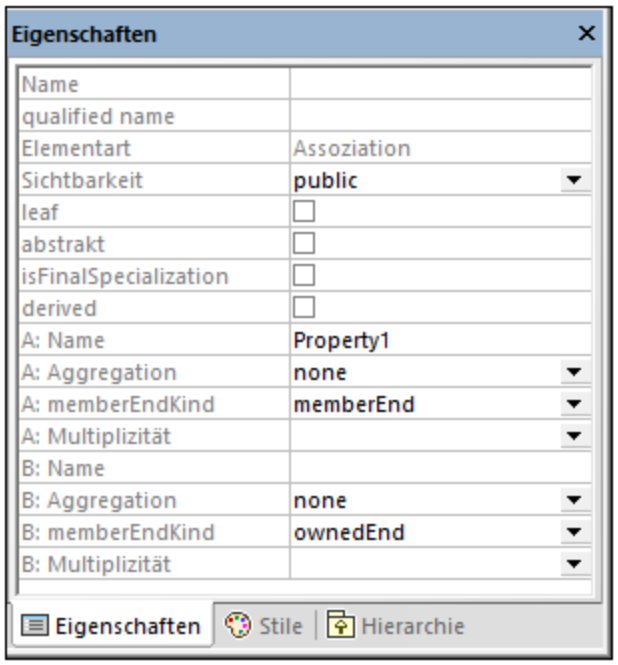
Wenn eine Klasse zu einem Diagramm hinzugefügt wird, werden ihre Assoziationen im Diagramm automatisch angezeigt, vorausgesetzt, die folgenden Bedingungen treffen zu:

- Die Option **Assoziationen automatisch erstellen** wurde unter **Extras | Optionen |** Register **Diagrammbearbeitung** aktiviert.
- Der Typ des Attributs wurde definiert (im Bild oben hat `Property1` den Typ `Engine`)
- Die Klasse des referenzierten "Typs" ist auch im aktuellen Diagramm vorhanden (in der Abbildung oben die Klasse `Engine`).

Sie können die Klasseneigenschaften jeder Klasse explizit als Assoziationen im Diagramm anzeigen. Klicken Sie dazu mit der rechten Maustaste auf eine Klasseneigenschaft und wählen Sie einen der folgenden Befehle:

- **Anzeigen | <Eigenschaft> als Assoziation**
- **Anzeigen | Alle Eigenschaften als Assoziationen**

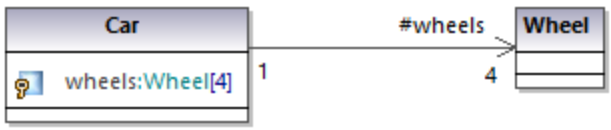
Wenn Sie im Diagramm auf eine Assoziation klicken, können ihre Eigenschaften bei Bedarf über das Fenster "Eigenschaften" geändert werden.



Beachten Sie besonders die unten aufgelisteten Eigenschaften. Wenn Sie diese Eigenschaften ändern, ändert sich das Aussehen der Assoziation im Diagramm oder es werden verschiedene informative Textbeschriftungen hinzugefügt. Informationen zum Anzeigen oder Ausblenden von Textbeschriftungen oder zum Ändern des Aussehens der Beziehung (z.B. Farben oder Linienbreite) finden Sie unter Ändern des Stils von Linien und Beziehungen 144.

Eigenschaft	Zweck
A: Name	Der Name des Mitglieds am Ende A der Beziehung. Im Car-Beispiel oben ist es `Property1`.
A: Aggregation	Damit können Sie den Typ der Assoziation am Ende A ändern. Wenn Sie diese Eigenschaft ändern, wird auch die Darstellung der Assoziation im Diagramm geändert. Gültige Werte: **none** Kennzeichnet eine normale Assoziation **shared** Ändert die Assoziation in eine Aggregation **composite** Ändert die Assoziation in eine Komposition
A: memberEndKind	Attribute, die an einer Beziehung beteiligt sind, können entweder zu einer Klasse oder zur Assoziation gehören. Diese Eigenschaft definiert, wer der Inhaber dieses Endes der Beziehung ist und ob dieses Ende der Beziehung "navigierbar" ist. (Mit "navigierbar" ist gemeint, dass das Ende einen "Pfeil" aufweist). Gültige Werte: **memberEnd** Das Mitglied an diesem Ende gehört zur Klasse.

© 2016-2022 Altova GmbH

Eigenschaft	Zweck
	ownedEnd Das Mitglied an diesem Ende gehört zur Assoziation **navigableOwnedEnd** Das Mitglied an diesem Ende gehört zur Assoziation und diese Ende wird navigierbar. Wenn Sie sowohl das Ende A als auch das Ende B auf **ownedEnd** setzen, wird die Assoziation bidirektional.
A: Multiplizität	Multiplizität definiert die Anzahl der Objekte an diesem Ende der Beziehung. Wenn ein Auto z.B. vier Räder hat, wäre die Multiplizität an einem Ende der Beziehung 1 und am anderen Ende 4. 

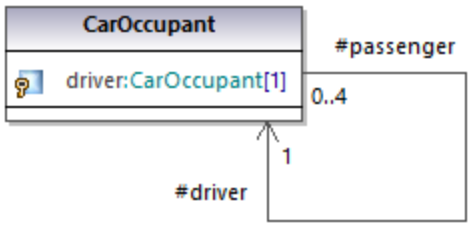
Dieselbe Gruppe von Attributen steht auch für das Ende B der Beziehung zur Verfügung.

Wenn im Fenster "Stile" die Eigenschaft **Show Assoc. Ownership Punkt anzeigen** aktiviert ist, werden Ownership-Punkte für die ausgewählte Beziehung angezeigt. Standardmäßig ist diese Eigenschaft auf **False** gesetzt. Im Folgenden sehen Sie ein Beispiel für eine Klasse, für die **Show Assoc. Ownership Punkt anzeigen** auf **True** gesetzt ist:



Erstellen reflexiver Assoziationen

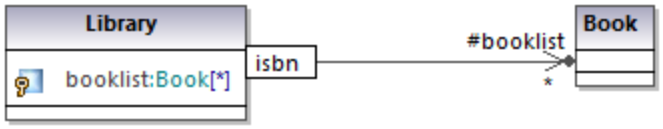
Assoziationen können mit derselben Klasse als Quelle und Ziel erstellt werden. Dies ist ein sogenanntes "self link" oder eine reflexive Assoziation. Damit kann z.B. die Fähigkeit eines Objekts, zum Zweck rekursiver Aufrufe eine Nachricht an sich selbst zu senden, beschrieben werden. Um ein self-link zu erstellen, klicken Sie auf die Symbolleisten-Schaltfläche "Assoziation" und ziehen Sie diese dann vom Element an eine andere Stelle im selben Element.



Erstellen von Assoziations-Qualifiern

Assoziationen können optional mit Assoziations-Qualifiern versehen werden. Qualifier sind Attribute einer Assoziation. Im Beispiel unten definiert der Assoziations-Qualifier `isbn`, dass ein Buch anhand dieses Attributs aus der Liste der Bücher abgerufen werden kann. So fügen Sie einen Qualifier hinzu:

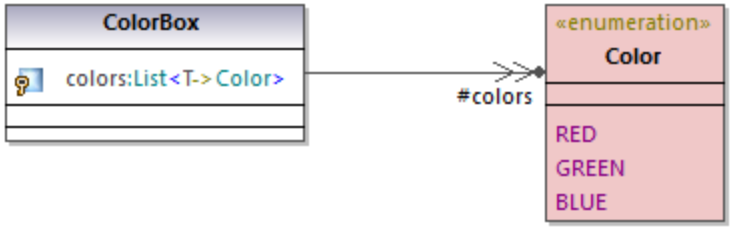
1. Erstellen Sie eine Assoziation zwischen zwei Klassen.
2. Klicken Sie mit der rechten Maustaste auf die Assoziation und wählen Sie **Neu | Qualifier**.



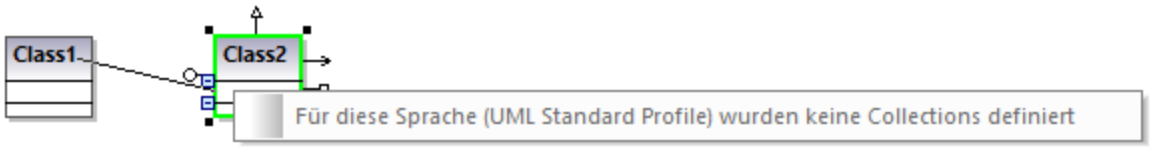
Um Assoziations-Qualifier umzubenennen oder zu löschen, gehen Sie auf dieselbe Weise vor wie bei anderen Elementen, siehe Umbenennen, Verschieben und Kopieren von Elementen 116 und Löschen von Elementen 117.

5.3.5 Collection-Assoziationen

Eine Collection-Assoziationsbeziehung eignet sich, um anzuzeigen, dass eine Klasseneigenschaft eine Collection irgendeiner Art ist. So ist etwa die Eigenschaft `colors` der Klasse `ColorBox` im Diagramm unten eine Liste von Farben. Dieser Typ ist in diesem Fall als Enumeration definiert; es kann sich dabei jedoch auch um eine andere Klasse oder sogar eine Schnittstelle handeln.



Bevor Sie Collection-Assoziationen erstellen, muss das UModel-Projekt die Collection-Vorlagen für die gewünschte Projektsprache (wie z.B. Java, C# oder VB.NET) enthalten. Andernfalls wird ein Tooltipp mit dem Text "Für diese Sprache wurden keine Collections definiert" angezeigt, wenn Sie versuchen, die Collection-Assoziation zu erstellen.



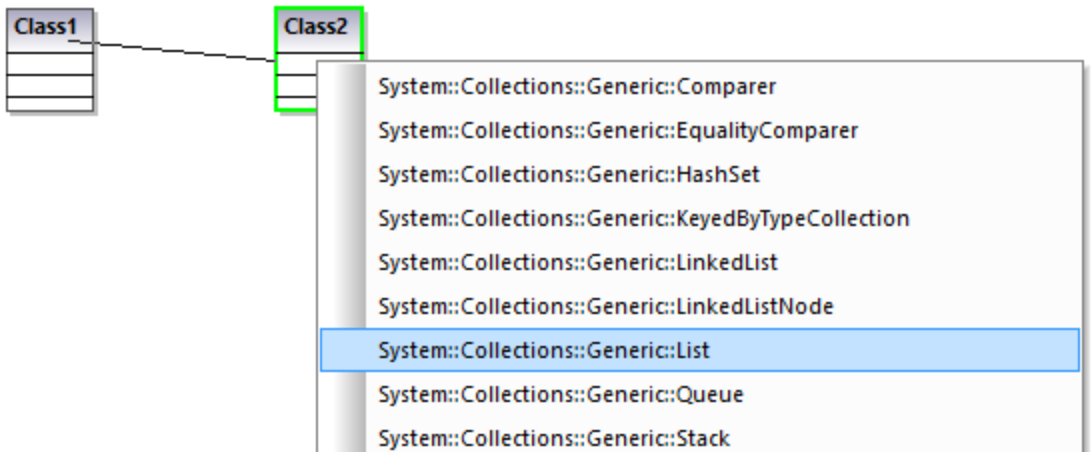
Wenn es sich bei Ihrem Projekt um ein reines UML-Projekt handelt (ohne Unterstützung für eine bestimmte Code Engineering-Sprache), können Sie über das Menü **Extras | Optionen | Diagrammbearbeitung | Collection-Vorlagen |** Register **UML** Collection-Vorlagen definieren.

© 2016-2022 Altova GmbH

Wenn Ihr Projekt bereits einen Sprach-Namespace (wie z.B. Java, C#, VB.NET) enthält, sind die Collection-Vorlagen über das Profil dieser Sprache vordefiniert. Sie können über das Menü **Extras | Optionen | Diagrammbearbeitung | Collection-Vorlagen** zusätzliche Vorlagen hinzufügen.

So erstellen Sie (z.B. zwischen zwei Klassen) eine Collection-Assoziation:

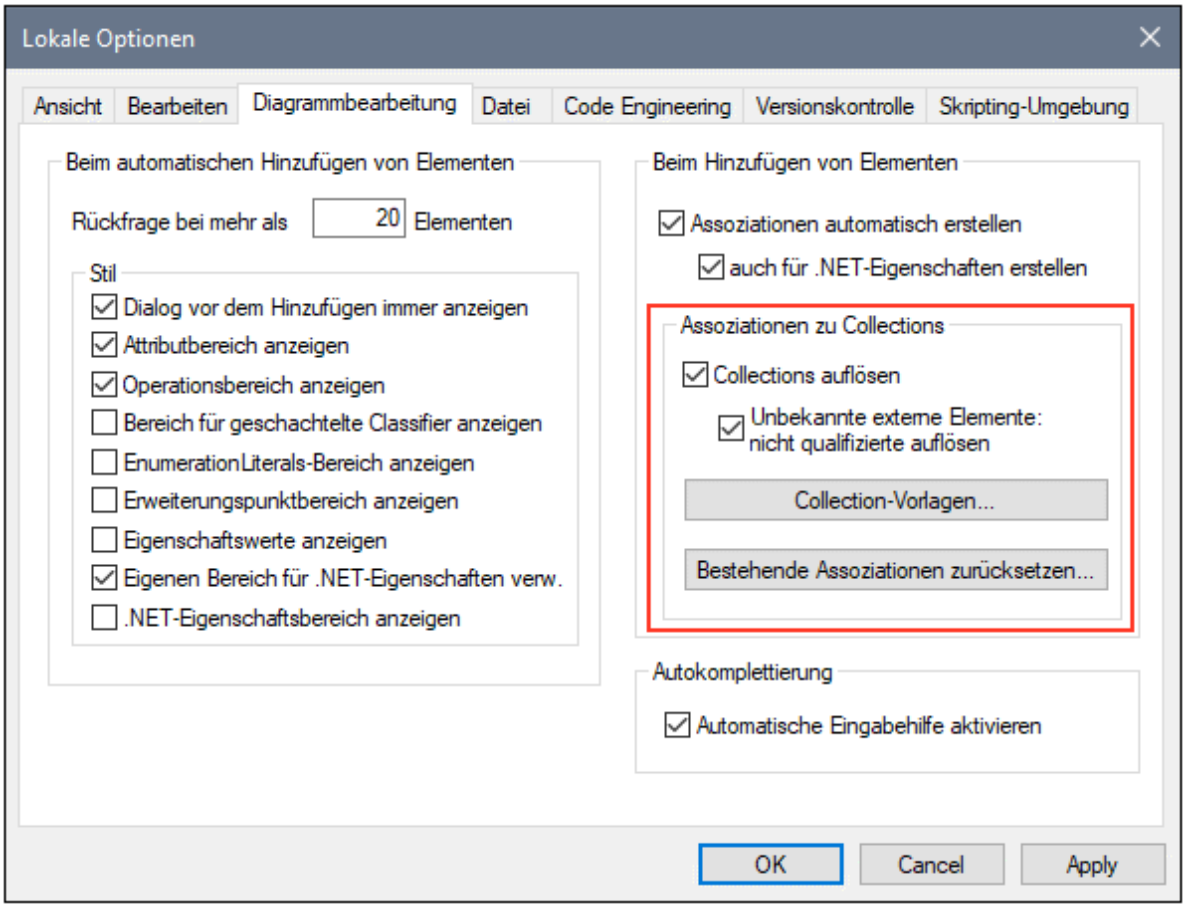
1. Fügen Sie zwei Klassen zum Diagramm hinzu.
2. Klicken Sie auf die Symbolleisten-Schaltfläche **Collection-Assoziation** .
3. Ziehen Sie die Maus von der ersten Klasse auf die zweite Klasse. Daraufhin werden im Kontextmenü die für das Projekt definierten Collection-Vorlagen angezeigt und Sie können die gewünschte auswählen.



Collection-Assoziationen und Code Engineering

Wenn Sie Programmcode in das Modell importieren, werden standardmäßig automatisch Collection-Assoziationen auf Basis der vordefinierten Collection-Vorlagen erstellt. So aktivieren bzw. deaktivieren Sie diese Option:

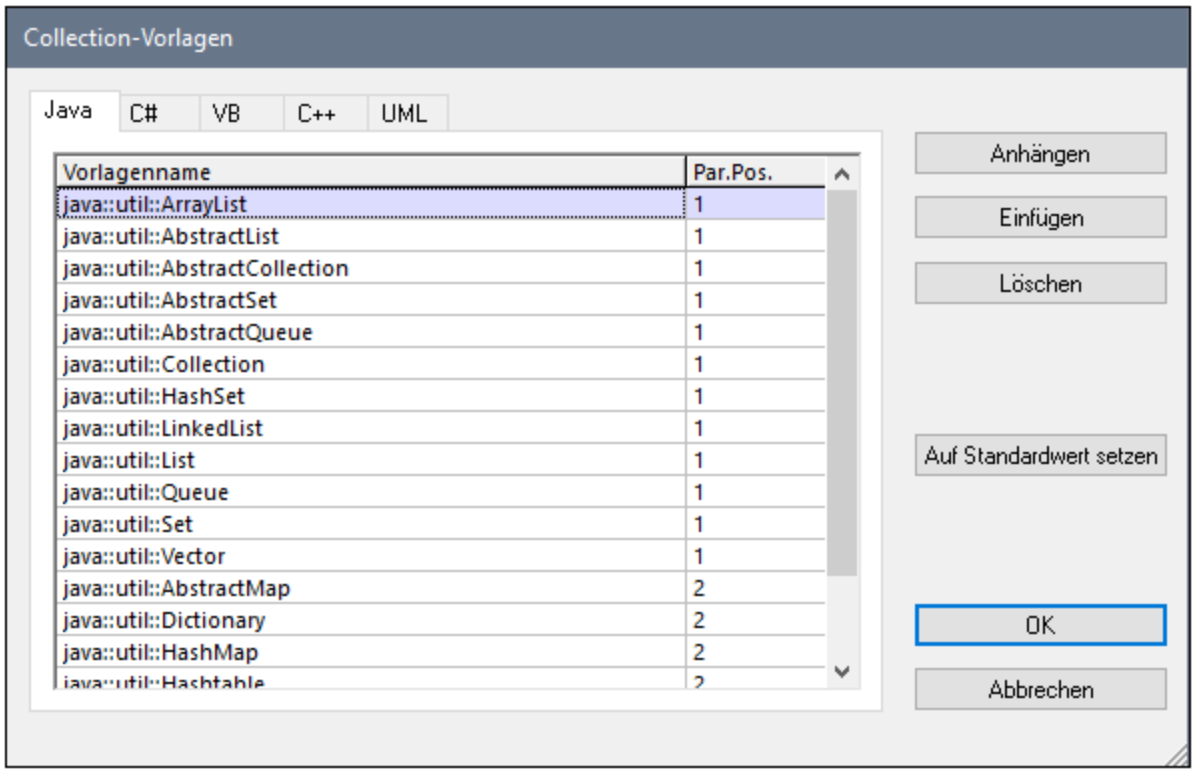
1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Diagrammbearbeitung**.
3. Aktivieren bzw. deaktivieren Sie je nach Bedarf das Kontrollkästchen **Collections auflösen.**



Die Collection-Assoziationen werden standardmäßig auf Basis einer Liste vordefinierter Collection-Vorlagen aufgelöst. Um die vordefinierten Collection-Vorlagen anzuzeigen oder zu ändern, klicken Sie auf **Collection-Vorlagen**.

Mit Hilfe der Dialogfeld-Schaltflächen Anhängen, Einfügen bzw. Löschen können Sie benutzerdefinierte Collection-Arten einfügen. Die Spalte **Par.Pos.** gibt die Position des Parameters, der den Wertetyp der Collection enthält, an.

© 2016-2022 Altova GmbH



Dialogfeld "Collection-Vorlagen"

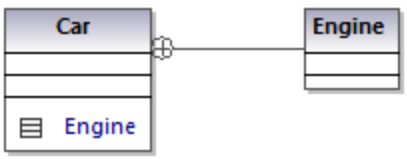
Um die Collection-Vorlagen auf ihre Standardwerte zurückzusetzen, klicken Sie auf **Auf Standardwerte setzen**.

5.3.6 Enthältbeziehung

Mit Hilfe einer Enthältbeziehungsassoziation werden z.B. Beziehungen Parent-Child-Beziehungen zwischen zwei Klassen oder Paketen dargestellt.

So stellen Sie eine Enthältbeziehung zwischen zwei Klassen dar:

1. Klicken Sie (in einer Klasse oder einem Paketdiagramm) auf die Symbolleisten-Schaltfläche **Enthältbeziehung** .
2. Ziehen Sie die Linie mit der Maus von der Klasse, die enthalten sein soll, auf die enthaltende Klasse.



Beachten Sie, dass die enthaltene Klasse - in diesen Fall `Engine` - jetzt in einem Bereich von `Car` zu sehen ist. Dadurch wird die enthaltene Klasse in denselben Namespace platziert wie die enthaltende Klasse.

© 2016-2022 Altova GmbH

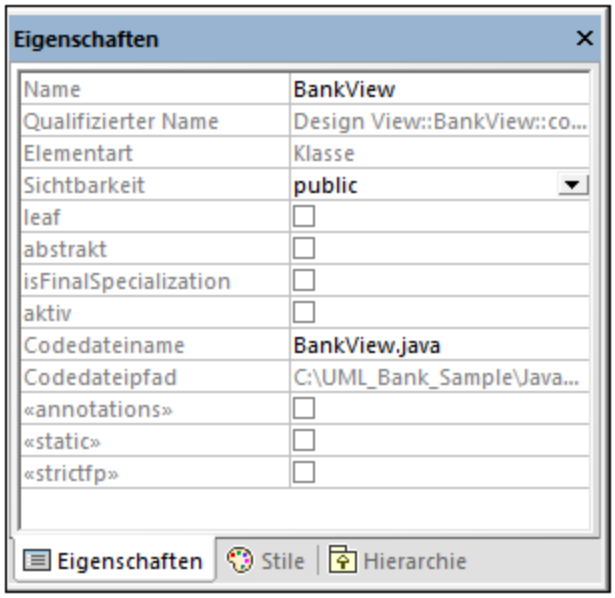
5.4 Stereotype und Eigenschaftswerte

Ein Stereotyp ist ein Erweiterungsmechanismus, mit dem ein vorhandenes XML-Element auf flexible Art erweitert werden kann, wodurch einige in Standard-UML nicht behandelten Aspekte abgedeckt werden können. Wenn Stereotype auf ein Element angewendet werden, bedeutet dies, dass das Element zu einem speziellen Zweck dient. Die vordefinierten UModel-Profile (C#, Java, VB.NET, usw.) enthalten alle für das Modellieren von Projekten in den entsprechenden Sprachen erforderlichen Stereotype. Sie können jedoch auch Ihre eigenen Profile (und die entsprechenden Stereotype) erstellen, siehe Erstellen und Anwenden von benutzerdefinierten Profilen 477.

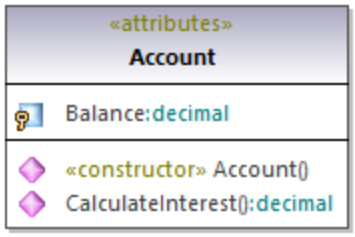
Wenn Sie Quellcode oder Binärdateien in das Modell importieren, wendet UModel automatisch auf Basis der Struktur des ursprünglichen Codes Stereotype auf Elemente an. Wenn z.B. im importierten Java-Quellcode annotations Modifier vorhanden sind, erhalten die entsprechenden Elemente im Modell das Stereotyp «annotations». Informationen zu den Elementensprechungen für verschiedene Sprachkonstrukte in UModel und deren Darstellung als Stereotype finden Sie unter UModel-Elementzuordnungen 247.

Sie können Stereotype auch manuell auf Elemente anwenden, während Sie diese modellieren. So können Sie etwas das Stereotyp «attributes» auf eine C#-Klasse anwenden, wodurch angegeben wird, dass die Klasse im generierten Code mit Attributen versehen werden muss. Um die Attributwerte im generierten Code zu definieren, können Sie in UModel so genannte "Eigenschaftswerte" hinzufügen, wie unter Anwenden von Stereotypen 157 gezeigt. Auch bei der XML-Schemamodellierung kommen sehr häufig Stereotype zum Einsatz, um Elemente wie simpleTypes, complexTypes, Facets, usw. zu definieren. Ebenso werden Stereotype auch zur Modellierung von Datenbanken verwendet, um Elemente wie Tabellen, Spalten, Indizes, usw. zu modellieren, siehe Erstellen von Datenbankobjekten 565.

Auf der grafischen Benutzeroberfläche von UModel werden Stereotype innerhalb von doppelten spitzen Anführungszeichen (z.B. «static») angezeigt. Alle in den vordefinierten UModel-Profilen enthaltenen Stereotype, werden im Fenster "Eigenschaften" angezeigt, wenn Sie auf ein Element klicken. Wenn Sie z.B. in der Modell-Struktur auf eine Java-Klasse klicken, werden im Fenster "Eigenschaften" ausschließlich Klassenstereotype für das Java-Profil angezeigt (in diesem Beispiel, «annotations», «static», «strictfp»).



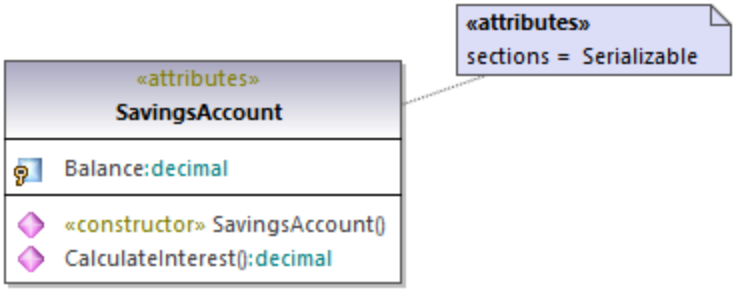
In Klassendiagrammen werden Stereotype oberhalb des Namens der Klasse angezeigt. Die unten gezeigte Klasse hat z.B. das Stereotyp «attributes».



Bei Methoden oder Eigenschaften werden Stereotype inline angezeigt, wie z.B. das Stereotyp «constructor», das in der oben gezeigten Klasse auf die Methode **Account()** angewendet wurde.

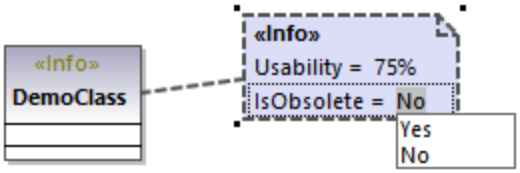
5.4.1 Eigenschaftswerte

Stereotype können mit Attributen (Eigenschaftswerten) verknüpft sein. Eigenschaftswerte sind Name-Wert-Paare, die Zusatzinformationen zum entsprechenden Stereotyp enthalten. So wurde z.B. auf die unten gezeigte Klasse das Stereotyp «attributes» angewendet. Beachten Sie, dass mit dem Stereotyp «attributes» Eigenschaftswerte verknüpft sind: ein Schlüssel(name) namens "sections" und der Wert "Serializable".



Eigenschaftswerte

Ein Stereotyp kann mehrere Eigenschaftswertpaare haben. Ein Wert kann auch aus einer Gruppe von Enumerationswerten ausgewählt werden.



Sie können die Art, wie Eigenschaftswerte im Diagramm angezeigt werden, ändern oder diese komplett ausblenden, siehe Anzeigen oder Ausblenden von Eigenschaftswerten[159]. Informationen darüber, wie Sie die Eigenschaftswerte eines Stereotyps ändern können, finden Sie unter Anwenden von Stereotypen[157]. Ein Beispiel für die Erstellung eines Stereotyps mit Eigenschaftswerten finden Sie unter Beispiel: Erstellen und Anwenden von Stereotypen[482].

© 2016-2022 Altova GmbH

5.4.2 Anwenden von Stereotypen

Durch Anwendung eines Stereotyps auf ein Element geben Sie an, dass das Element einen bestimmten Verwendungszweck hat. Im Falle der von UModel unterstützten Codesprachen (wie z.B. C#, VB.NET, Java) werden Stereotype normalerweise angewendet, um die Grammatik dieser Sprache abzubilden. So kann etwa auf eine Java-Klasse das Stereotyp «static» angewendet werden.

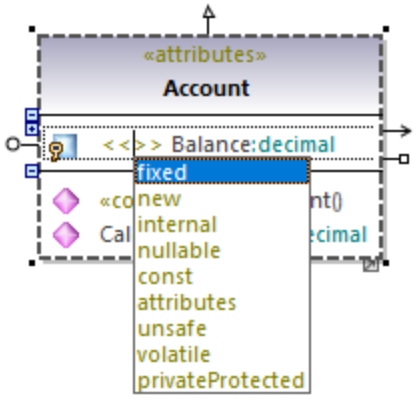
Bevor Sie Stereotype anwenden können, muss zuerst das entsprechende Profil auf Ihr(e) Paket(e) angewendet werden. Dies wird in UModel automatisch durchgeführt, wenn Sie mit der rechten Maustaste auf ein Paket klicken und den Befehl **Code Engineering | Als {language} Namespace Root definieren** auswählen. Nähere Informationen dazu finden Sie unter Anwenden von UModel-Profilen 170.

Wenn Sie benutzerdefinierte Profile erstellt haben, so müssen diese manuell auf das Paket angewendet werden, siehe Erstellen und Anwenden von benutzerdefinierten Profilen 477.

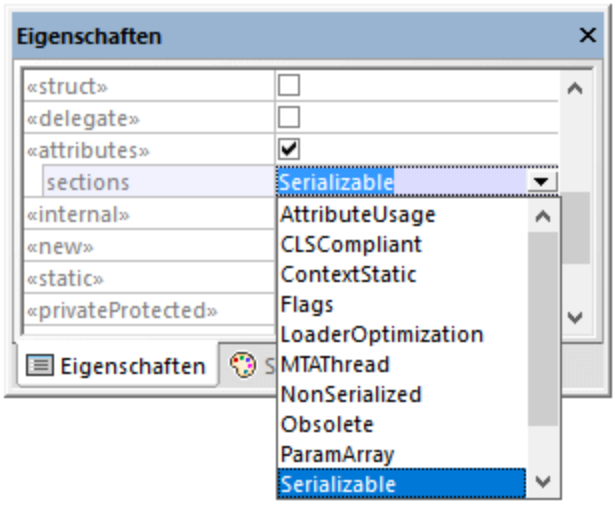
So wenden Sie ein Stereotyp auf ein Element an:

1. Klicken Sie im Fenster "Modell-Struktur" auf das Element. Wenn das Element durch Stereotype erweitert werden kann, werden diese im Fenster "Eigenschaften" als innerhalb von doppelte spitze Anführungszeichen ("«" und "»") gesetzte Eigenschaften angezeigt.
2. Aktivieren Sie im Fenster "Eigenschaften" das Kontrollkästchen des Stereotyps (z.B. «static»).

Sie können Stereotype auch anwenden, während Sie Elemente innerhalb eines Klassendiagramms erstellen. Klicken Sie dazu auf eine Eigenschaft einer Klasse und geben Sie innerhalb der Zeichen "<< und ">>" die ersten Buchstaben des Texts ein.

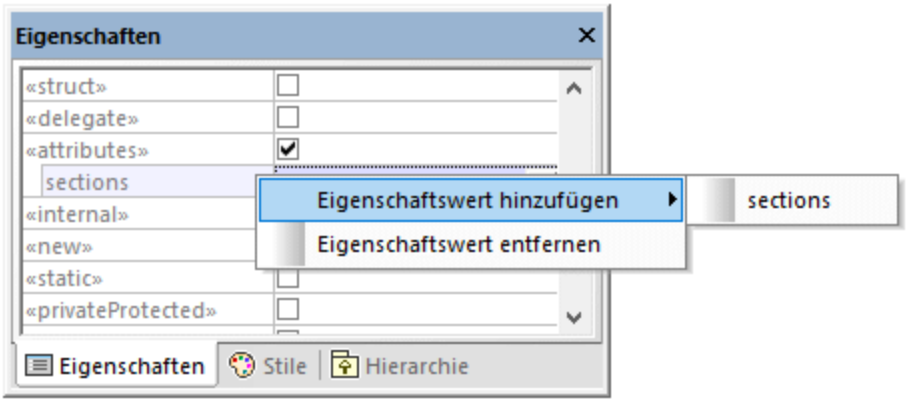


Einige Stereotype sind mit einer in UML als "Eigenschaftswerte" bezeichneten Liste von Name-Wert-Paaren verknüpft. Um ein Stereotyp mit Eigenschaftswerten auf ein Element anzuwenden, aktivieren Sie im Fenster "Eigenschaften" das Kontrollkästchen dieses Stereotyps (in diesem Beispiel «attributes»). Dadurch wird ein eingerückter Eintrag hinzugefügt, wo Sie den erforderlichen Wert aus einer vordefinierten Liste auswählen können.

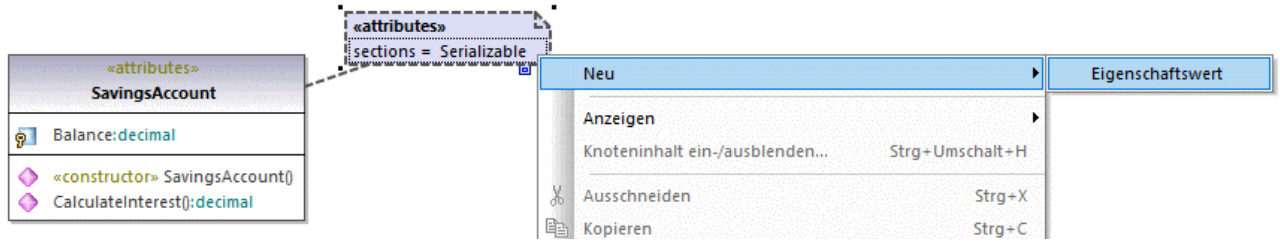


Eigenschaftswerte

Sie können auch mehrere Werte zum selben Schlüssel hinzufügen. Klicken Sie dazu mit der rechten Maustaste auf den eingerückten Eintrag und wählen Sie im Kontextmenü den Befehl **Eigenschaftswert hinzufügen | <name>**.



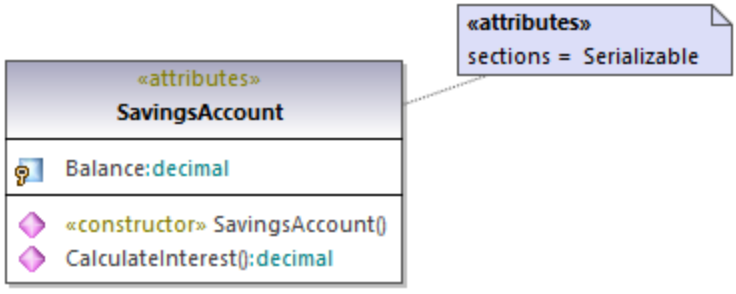
Alternativ dazu können Sie Eigenschaftswerte auch direkt über das Diagramm hinzufügen, indem Sie mit der rechten Maustaste auf einen Wert klicken und im Kontextmenü den Befehl **Neu | Eigenschaftswert** auswählen.



 © 2016-2022 Altova GmbH

5.4.3 Anzeigen oder Ausblenden von Eigenschaftswerten

Wenn ein Element Eigenschaftswerte hat, können Sie alle jeweiligen Eigenschaftswerte entweder in einem eigenen Kasten oder inline, als Bereich, anzeigen. Eigenschaftswerte können komplett ausgeblendet werden. Um auszuwählen, wie Eigenschaftswerte angezeigt werden sollen, klicken Sie mit der rechten Maustaste auf das Element im Diagramm und wählen Sie die Option **Eigenschaftswerte | <Anzeigeoption>** aus. Um z.B. alle Eigenschaftswerte außerhalb der Klasse anzuzeigen, klicken Sie im Diagramm mit der rechten Maustaste auf die Klasse und wählen Sie **Eigenschaftswerte | alle**. Um alle Eigenschaftswerte einer Klasse auszublenden, klicken Sie mit der rechten Maustaste auf die Klasse im Diagramm und wählen Sie den Befehl **Eigenschaftswerte | Keine**.

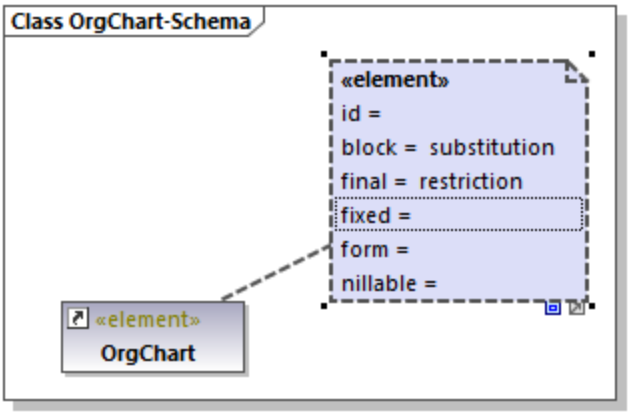


Außerhalb einer Klasse angezeigte Eigenschaftswerte

Ein- und Ausschalten des Kompaktmodus

Wenn einige Werte in einem Eigenschaftswertekasten leer sind, können Sie nur die leeren Werte folgendermaßen ausblenden:

1. Wählen Sie einen Eigenschaftswertekasten (mit sowohl leeren als auch nicht leeren Werten) im Diagramm aus.



2. Klicken Sie in der rechten unteren Ecke des Kastens auf den Ziehpunkt **Kompaktmodus ein-/ausschalten**.

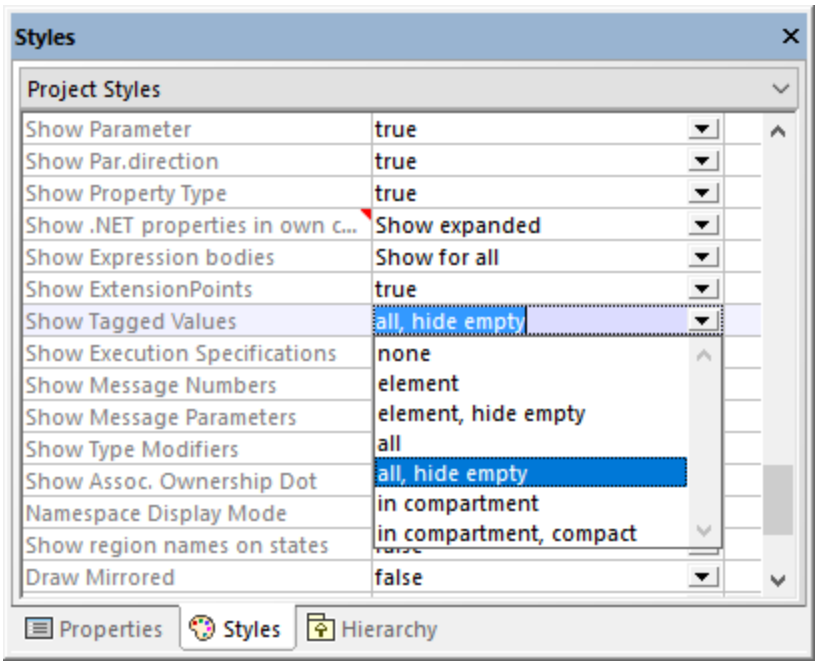
Im erweiterten Zustand werden auch die leeren Werte angezeigt. Im reduzierten Zustand werden die leeren Werte ausgeblendet.

Globale Änderung der Anzeige von Eigenschaftswerten

Sie können die Anzeigen von Eigenschaftswerten entweder, wie oben gezeigt, für einzelne Elemente oder global auf Projektebene ändern.

So ändern Sie die Anzeige von Eigenschaftswerten auf Projektebene:

1. Wählen Sie aus der Liste am oberen Rand des Fensters "Stile" [93] **Projektstile** aus.
2. Scrollen Sie hinunter bis zur Eigenschaft **Eigenschaftswerte anzeigen** und wählen Sie die gewünschte Option aus der Liste aus (z.B. **alle, leere ausblenden**).



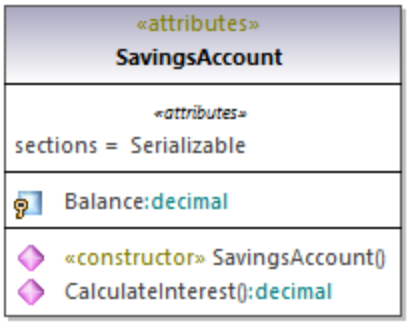
Informationen darüber, wie Sie Stile auf verschiedenen Ebenen ändern, finden Sie unter Ändern des Stils von Elementen [126].

Mögliche Anzeigeoptionen

In der Tabelle unten sehen Sie eine Liste möglicher Optionen für die Anzeige von Eigenschaftswerten. Diese Optionen sind ähnlich, wenn Sie Eigenschaftswerte global oder nur für einzelne Elemente ändern.

Keines	Blendet alle Eigenschaftswerte aus.
Alle	Zeigt die Eigenschaftswerte eines Elements (z.B. einer Klasse) sowie diejenigen von Elemente im Eigentum der Klasse wie z.B. Attributen und Operationen an.
Alle, leere ausblenden	Zeigt nur die Eigenschaftswerte, für die ein Wert vorhanden ist, an.

© 2016-2022 Altova GmbH

Element	Zeigt die Eigenschaftswerte eines Elements (z.B. einer Klasse) an, nicht aber diejenigen von Attributen, Operationen, usw. im Eigentum der Klasse.
Element, leere ausblenden	Zeigt nur die Eigenschaftswerte eines Elements, für die ein Wert vorhanden ist, an.
Im Bereich	Zeigt die Eigenschaftswerte in einem separaten Bereich an. So hat etwa die unten gezeigte Klasse einen Bereich *«attributes»*, der Eigenschaftswerte enthält. 
Im Bereich, leere ausblenden	Zeigt nur die Eigenschaftswerte, für die ein Wert vorhanden ist, in einem Bereich an.
Im Bereich, kompakt	Wie oben.

6 Projekte und Code Engineering

Dieses Kapitel enthält Informationen zur Erstellung von UModel-Projekten (entweder von neuen Projekten oder durch Import von Daten aus Quellcode oder Binärdateien). Außerdem werden darin verschiedene Operationen im Zusammenhang mit Code Engineering mit UModel beschrieben. Dazu zählen:

- Forward Engineering (Generieren von Code anhand eines UModel-Projekts)
- Reverse Engineering (Importieren von Quellcode in ein UModel-Projekt)
- Roundtrip Engineering (Synchronisieren von Modell und Code in beide Richtungen je nach Bedarf)

Die Menübefehle zum Code Engineering stehen im Menü **Projekt** zur Verfügung. So können Sie etwa mit dem Menübefehl **Projekt | Quellprojekt importieren** C#-, C++- oder VB.NET Visual Studio-Lösungen oder Java-Code importieren und anhand dieses Codes UModel-Diagramme generieren. Wenn keine Projektlösung zur Verfügung steht, verwenden Sie den Menübefehl **Projekt | Quellverzeichnis importieren**, siehe Importieren von Quellcode (Reverse Engineering) 209. Auch Java-, C#- und VB.NET-Binärdateien können importiert werden, vorausgesetzt, es werden einige grundlegende Voraussetzungen erfüllt, siehe Importieren von Java-, C#- und VB.NET-Binärdateien 225.

Die oben erwähnten Code Engineering-Operationen können nicht nur auf Programmiersprachen, sondern auch auf Datenbanken und XML-Schemas angewendet werden. So können Sie etwa mit dem Menübefehl **Projekt | XML-Schema-Datei importieren** ein Reverse Engineering eines vorhandenen XML-Schemas durchführen und automatisch ein Klassendiagramm anhand dieser Datei generieren.

Unter UModel-Elementzuordnungen 247 finden Sie eine Liste von Zuordnungen zwischen UModel-Elementen und Elementen in den einzelnen unterstützten Sprachprofilen (darunter auch Datenbanken und XML-Schemas). Informationen zur Verbindung mit Datenbanken und den dazugehörigen Operationen finden Sie unter UModel und Datenbanken 555.

 © 2016-2022 Altova GmbH

6.1 Verwalten von UModel-Projekten

Ein UModel-Projekt dient als Container für UML-Modellierungselemente, Diagramme und verschiedene von Ihnen definierbare Einstellungen zum Projekt. UModel-Projekte werden mit der Dateierweiterung .ump (UModel-Projektdatei) gespeichert.
UModel zwingt Sie nicht, sich bei der Modellierung an eine bestimmte vordefinierte Reihenfolge zu halten. Sie können jeden Modellelementtyp zum Projekt hinzufügen: UML-Diagramm, Paket, Akteur usw. - und zwar in jeder beliebigen Reihenfolge bzw. an jeder beliebigen Stelle. Alle Modellelemente können auf dem Register "Modell-Struktur" eingefügt, umbenannt und gelöscht werden, d.h. Sie müssen diese Elemente nicht unbedingt als Teil eines Diagramms erstellen.

6.1.1 Erstellen, Öffnen und Speichern von Projekten

Wenn Sie UModel zum ersten Mal starten, wird automatisch ein neues Projekt geöffnet. Bei späteren Ausführungen wird das zuletzt verwendete Projekt geöffnet.

Anmerkung: UModel enthält eine Reihe von Beispielprojekten, anhand welcher Sie die Grundlagen der Modellierung erlernen und die grafische Benutzeroberfläche von UModel kennenlernen können. Diese Projekte befinden sich im folgenden Ordner: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**.

So erstellen Sie ein neues Projekt:

- Klicken Sie im Menü **Datei** auf **Neu** (oder klicken Sie auf die Symbolleistenschaltfläche **Neu**).

Daraufhin wird ein neues Projekt mit dem Standardnamen **NeuesProjekt1** erstellt. Außerdem werden die folgenden Pakete automatisch zum Projekt hinzugefügt und im Fenster "Modell-Struktur" angezeigt.

- Root
- Component View

Diese beiden Pakete haben einen speziellen Zweck und sind die einzigen, die, wie im Tutorial unter Forward Engineering (Modell zu Code) 65 erläutert, nicht umbenannt oder gelöscht werden können.

Nachdem Sie das Projekt erstellt haben, können Sie Modellierungselemente wie UML-Pakete und Diagramme dazu hinzufügen, siehe Erstellen von Elementen 113 und Erstellen von Diagrammen 129.

So fügen Sie ein neues Paket hinzu:

1. Rechtsklicken Sie auf das Paket, unter dem das neue Paket angezeigt werden soll (in einem neuen Projekt entweder "Root" oder "Component View").
2. Wählen Sie im Kontextmenü den Befehl **Neues Element | Paket**.

Beachten Sie, dass Paket sowie andere Modellierungselemente auch über UML-Diagramme hinzugefügt werden können. Sie werden in diesem Fall automatisch im Fenster "Modell-Struktur" angezeigt.

So fügen Sie ein neues Diagramm hinzu:

- Klicken Sie mit der rechten Maustaste auf ein Paket in der Modell-Struktur und wählen Sie **Neues Diagramm**.

So fügen Sie Elemente zu einem Diagramm hinzu:

- Wählen Sie eine der folgenden Methoden:
  - Klicken Sie mit der rechten Maustaste auf das Diagramm und Wählen Sie **Neues Element | <Elementart>** aus dem Kontextmenü aus.
  - Ziehen Sie das gewünschte Element aus der Symbolleiste.

Ein Arbeitsbeispiel zur Erstellung eines Projekts und zum Generieren von Programmcode anhand dieses Projekts finden Sie unter Forward Engineering (Modell zu Code) 65.

So öffnen Sie ein vorhandenes Projekt:

- Klicken Sie im Menü **Datei** auf **Öffnen** und navigieren Sie zur .ump-Projektdatei.

Anmerkung: Externe Änderungen, die an der Projektdatei oder inkludierten Dateien extern vorgenommen wurden, werden automatisch erkannt und es erscheint eine entsprechende Meldung, ob das Projekt neu geladen werden soll. Diese Funktion kann über **Extras | Optionen** | Register **Datei** deaktiviert werden.

So speichern Sie ein Projekt:

- Klicken Sie im Menü Datei auf Speichern (oder **Speichern unter**).

Alle projektrelevanten Daten werden in der UModel-Projektdatei (Dateierweiterung *.**ump** (UModel-Projektdatei) gespeichert.

Anmerkung: Die *.ump-Datei ist ein XML-Dateiformat, auf das beim Speichern die Option "Pretty Print" aus dem Menü **Extras | Optionen |** Register **Datei** angewendet werden kann.

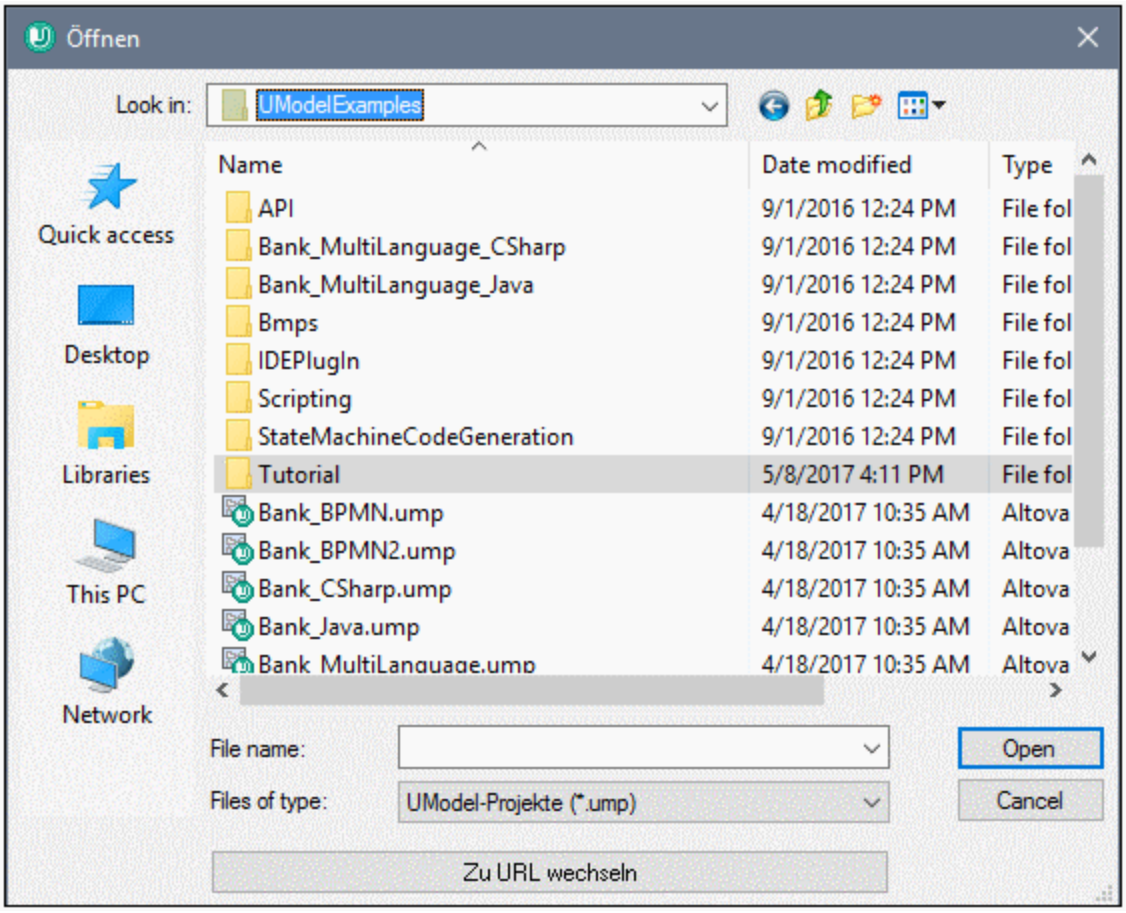
6.1.2 Öffnen von Projekten über eine URL

Zusätzlich zu lokalen Projekt-Dateien können Sie Dateien auch über eine URL öffnen. Es werden die Protokolle HTTP, HTTPS und FTP unterstützt. Beachten Sie, dass von URLs geladene Dateien nicht wieder unter ihrem ursprünglichen Pfad gespeichert werden können (d.h. Sie haben nur Lesezugriff auf die Datei), es sei denn, Sie haben diese, wie unten gezeigt, von einem Microsoft® SharePoint® Server ausgecheckt.

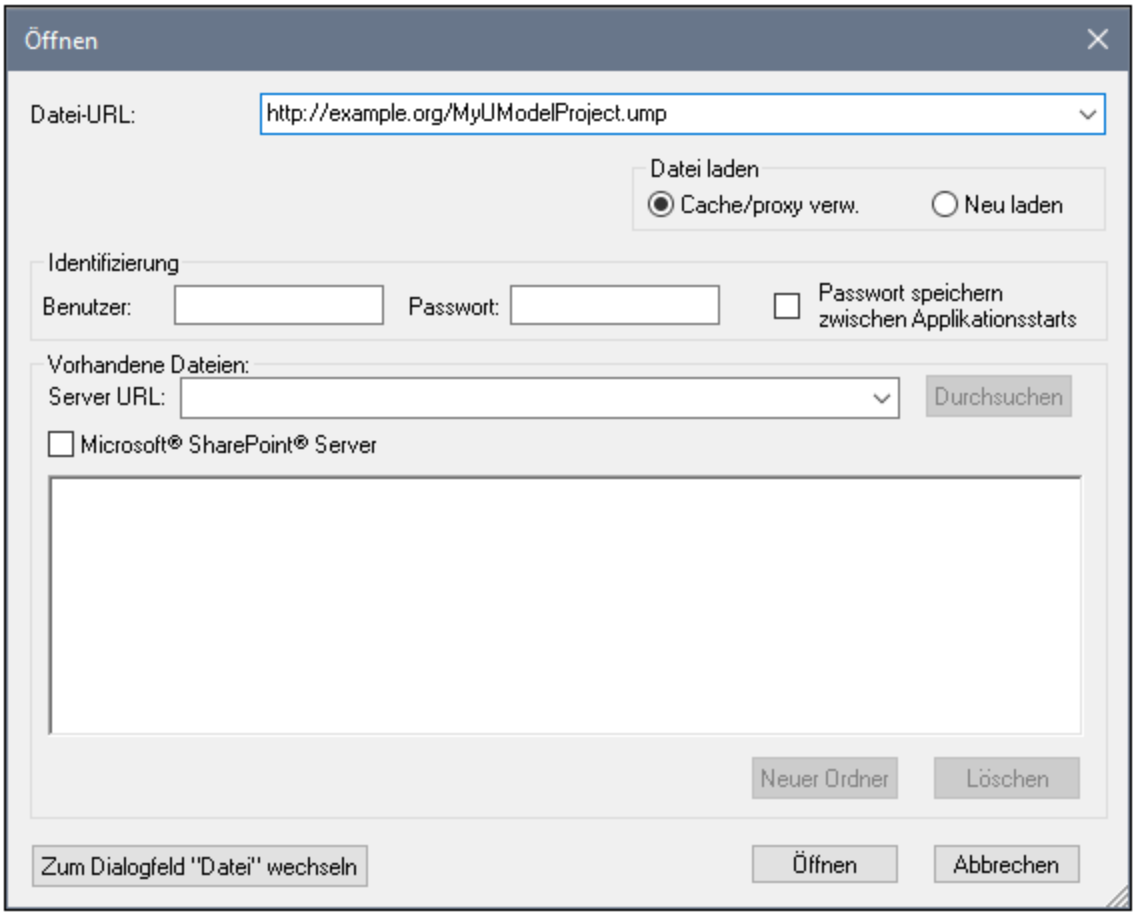
So öffnen Sie eine Datei über eine URL:

1. Klicken Sie im Dialogfeld **Öffnen** auf **Zu URL wechseln**.

© 2016-2022 Altova GmbH



2. Geben Sie die URL der Datei in das Textfeld **Datei-URL** ein und klicken Sie auf **Öffnen**.



Falls für den Server eine Passwort-Authentifizierung erforderlich ist, werden Sie aufgefordert, den Benutzernamen und das Passwort einzugeben. Wenn Sie möchten, dass sich UModel den Benutzernamen und das Passwort für das nächste Mal merkt, geben Sie diese in das Dialogfeld "Öffnen" ein und aktivieren Sie das Kontrollkästchen **Passwort speichern zwischen Applikationsstarts**.

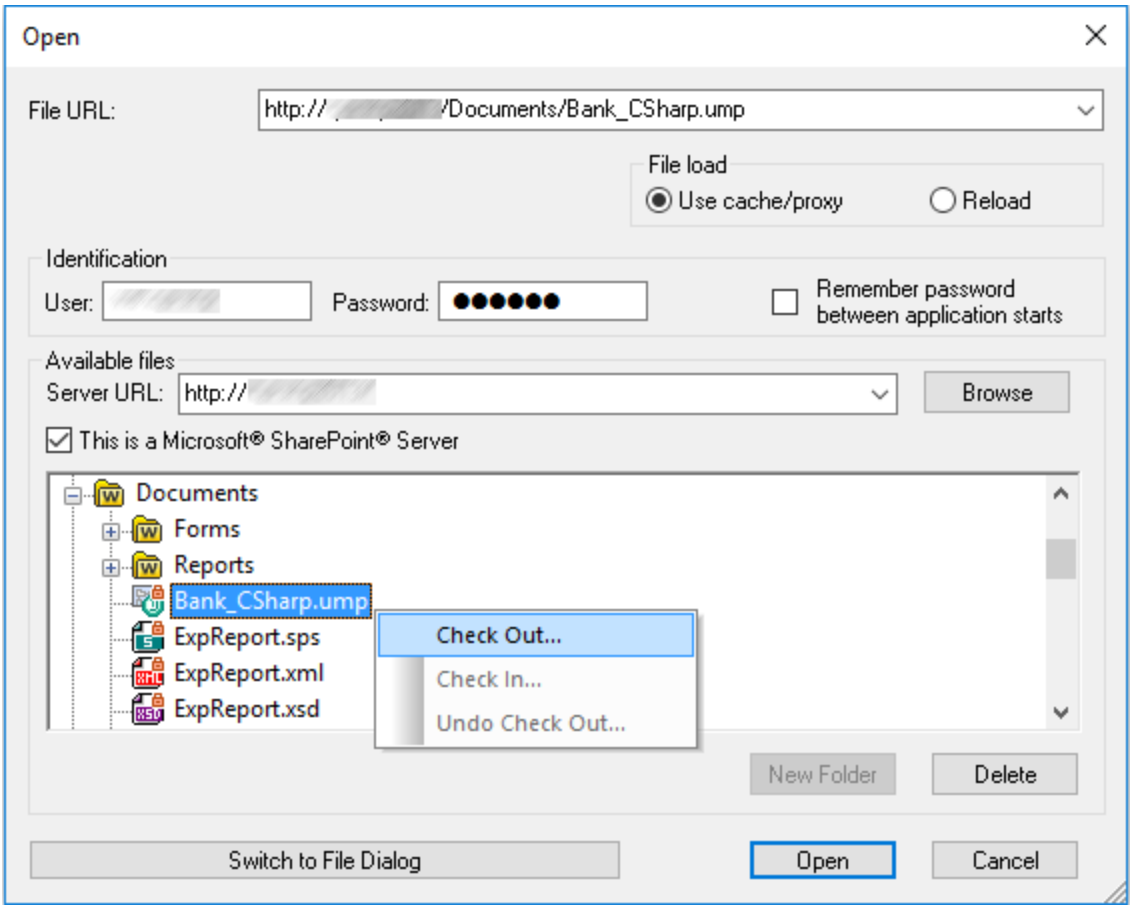
Wenn sich die zu ladende Datei wahrscheinlich nicht ändern wird, aktivieren Sie die Option **Cache/proxy verwenden,** damit die Daten im Cache gespeichert werden und die Datei schneller geladen wird. Wenn die Datei bei jedem Start von UModel neu geladen werden soll, wählen Sie **Neu laden**.

Für Server mit Unterstützung für Web Distributed Authoring and Versioning (WebDAV) können Sie die Verzeichnisse nach Eingabe der **Server-URL** in das Textfeld Server URL und Klicken auf **Durchsuchen** durchsuchen.

Anmerkung: Die Funktion **Durchsuchen** steht nur auf Servern, die WebDAV unterstützen und auf Microsoft SharePoint Servern zur Verfügung.

Wenn es sich beim Server um einen Microsoft® SharePoint® Server handelt, aktivieren Sie das Kontrollkästchen **Microsoft® SharePoint® Server**. Daraufhin wird im Vorschaubereich angezeigt, ob die Datei eingecheckt oder ausgecheckt ist.

© 2016-2022 Altova GmbH



Dateien können einen der folgenden Status haben:

	Eingecheckt. Die Datei kann ausgecheckt werden.
	Von einem anderen Benutzer ausgecheckt. Kann nicht ausgecheckt werden.
	Lokal ausgecheckt. Kann bearbeitet und eingecheckt werden.

Um die Datei in UModel ändern zu können, klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie den Befehl **Auschecken**. Wenn eine Datei von Microsoft® SharePoint® ausgecheckt wurde, werden die Änderungen in der Datei beim Speichern der Datei in UModel zurück an den Server gesendet. Um die Datei am Server wieder einzuchecken, klicken Sie im Dialogfeld oben mit der rechten Maustaste auf die Datei und wählen Sie im Kontextmenü den Befehl **Einchecken** (Melden Sie sich alternativ dazu am Server an und führen Sie diese Operation direkt vom Browser aus aus). Um Änderungen, die seit dem letzten Check-out an der Datei vorgenommen wurden, zu verwerfen, klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie den Befehl **Auschecken rückgängig** (oder führen Sie diese Operation über den Browser aus).

Beachten Sie dazu Folgendes:

- Wenn eine Datei bereits von einem anderen Benutzer ausgecheckt wurde, steht Sie nicht zum Auschecken zur Verfügung.
- Wenn Sie eine Datei in einer Altova-Applikation auschecken, können Sie sie nicht in einer anderen

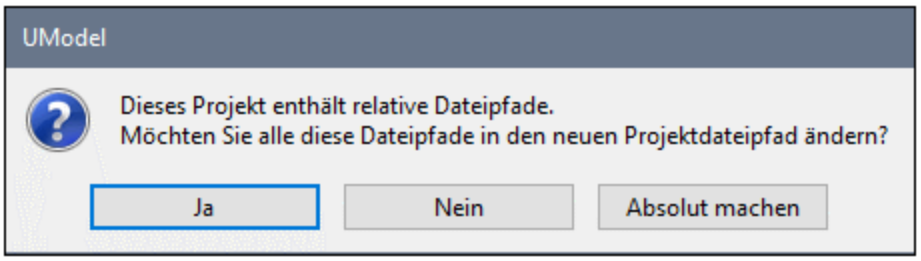
Altova-Applikation auschecken. Die Datei gilt bereits als für Sie ausgecheckt.

6.1.3 Verschieben von Projekten in ein neues Verzeichnis

UModel-Projekte und generierter Code können ganz einfach in ein anderes Verzeichnis (oder auf einen anderen Computer) verschoben und dort erneut synchronisiert werden.

Dazu gibt es zwei Methoden:

- Wählen Sie die Menüoption **Datei | Speichern unter...** und klicken Sie auf **Ja**, wenn Sie gefragt werden, ob die Dateipfade an den neuen Projektordner angepasst werden sollen.

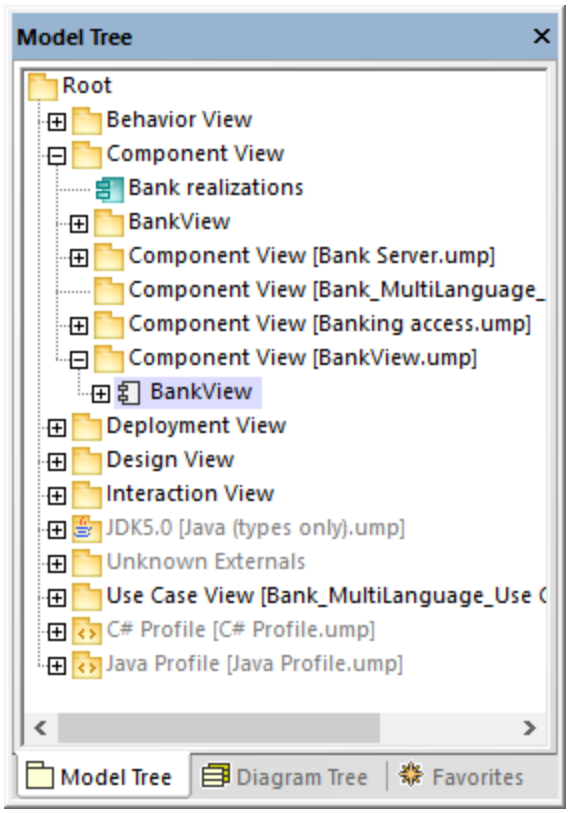


- Kopieren Sie das UModel-Projekt (*.ump) in einen neuen Ordner und passen Sie die Codegenerierungspfade für die einzelnen beteiligten Komponenten an.

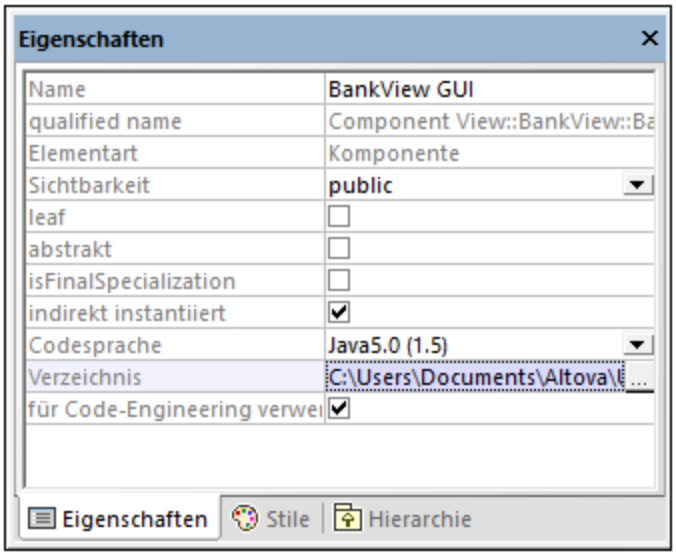
Als Beispiel für die zweite Methode öffnen Sie das folgende Beispielprojekt: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_Multilanguage.ump**.

1. Suchen Sie in der Modell-Struktur die Komponente `BankView`.

© 2016-2022 Altova GmbH



2. Suchen Sie im Fenster "Eigenschaften" die Eigenschaft **Verzeichnis**, und geben Sie dort den neuen Pfad an.



3. Synchronisieren Sie das Modell erneut mit dem Code.

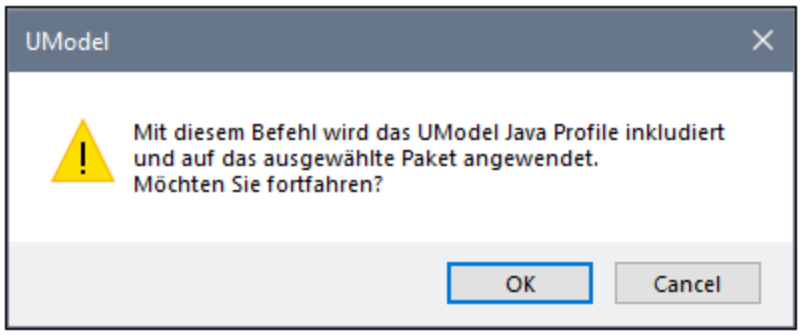
6.1.4 Anwenden von UModel-Profilen

Wenn Sie in UModel ein neues Modellierungsprojekt beginnen, sind im Projekt standardmäßig noch keine Geschäftsanwendungen oder Code Engineering-Sprachen definiert, die Sie benötigen werden. Um daher für Ihr UML-Projekt eine Domain oder Sprache einzustellen, müssen Sie ein *Profil darauf anwenden*.

Es wird zwischen zwei Profilarten unterschieden:

- In UModel vordefinierte Profile (dazu gehören C++, C#, VB.NET, Java, BPMN 1.0, BPMN 2.0, SysML, usw.).
- Benutzerdefinierte Profile, die Sie erstellen können, um UML für Ihre spezifische Domain bzw. Ihre Bedürfnisse zu erweitern.

Über den Menübefehl **Projekt | Unterprojekt inkludieren** können Sie jedes der vordefinierten Profile zu Ihrem Projekt hinzufügen. Außerdem werden Sie, immer, wenn Sie eine Aktion durchführen, für die ein bestimmtes Profil benötigt wird, von UModel aufgefordert, ein vordefiniertes Profil anzuwenden. Wenn Sie z.B. mit der rechten Maustaste auf ein neues Paket klicken und im Kontextmenü die Option **Code Engineering | Als Java Namespace Root definieren** auswählen, werden Sie aufgefordert, das Java-Profil darauf anzuwenden.



Um die vollständige Liste der vordefinierten UModel-Profile anzuzeigen oder diese manuell zu Ihrem Modell hinzuzufügen, wählen Sie den Menübefehl **Projekt | Unterprojekt inkludieren**. Siehe auch Inkludieren von Unterprojekten 174.

Eine Anleitung zum Erstellen von benutzerdefinierten Profilen zur Erweiterung oder Anpassung von UML finden Sie unter Erstellen und Anwenden von benutzerdefinierten Profilen 477.

6.1.5 Aufteilen von UModel-Projekten

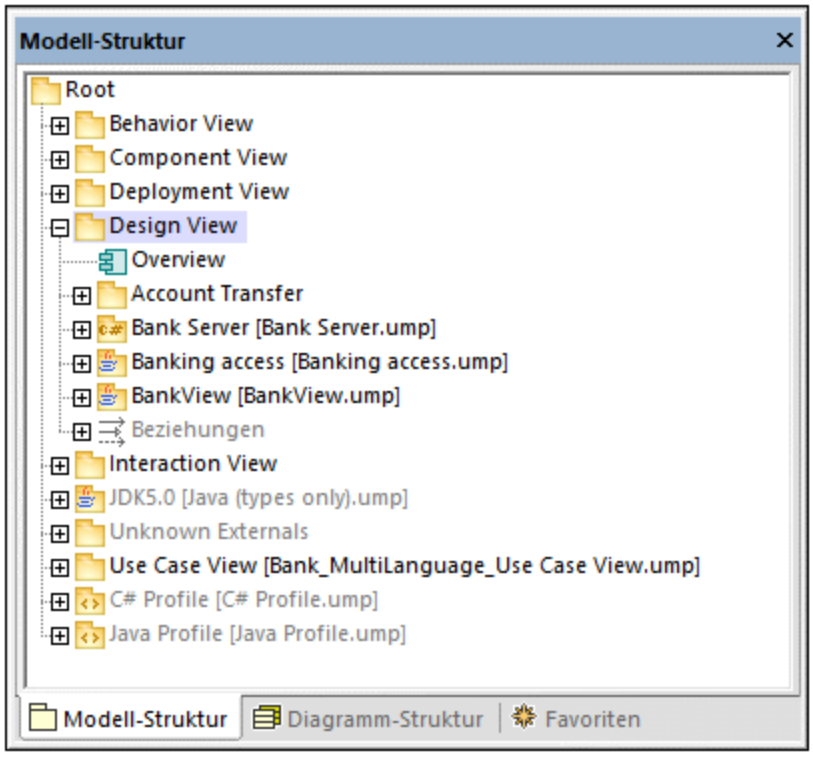
UModel-Projekte können in mehrere Unterprojekte aufgeteilt werden, sodass mehrere Entwickler unterschiedliche Teile eines einzigen Projekts gleichzeitig bearbeiten können. Unterprojekte sind genau wie UModel-Standardprojektdateien und haben dieselbe *.ump-Erweiterung. Die einzelnen Unterprojekte können zu einem Versionskontrollsystem hinzugefügt werden. Das Projekt der obersten Ebene wird als als Hauptprojekt bezeichnet.

Unterprojekte können anhand beinahe jeden Pakets im Hauptprojekt erstellt werden. Sie können auswählen, ob das Unterprojekt vom Hauptprojekt aus bearbeitet werden können soll oder ob es schreibgeschützt sein soll. Wenn es schreibgeschützt ist, kann das Unterprojekt nur bearbeitet werden, wenn es als eigenständiges Projekt geöffnet wird.

© 2016-2022 Altova GmbH

Unterprojekte können beliebig strukturiert werden, in einer flachen oder einer hierarchischen Struktur oder in Form einer Kombination aus beiden. Dadurch kann praktisch jedes Paket eines Hauptprojekts in Unterprojektdateien aufgeteilt werden.

Im Fenster "Modell-Struktur" 86 werden Unterprojekte innerhalb von eckigen Klammern mit dem jeweiligen .ump-Dateinamen rechts davon angezeigt. So enthält etwa das unten gezeigte Projekt (**Bank_MultiLanguage.ump** aus dem Verzeichnis **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**) mehrere Unterprojekte.

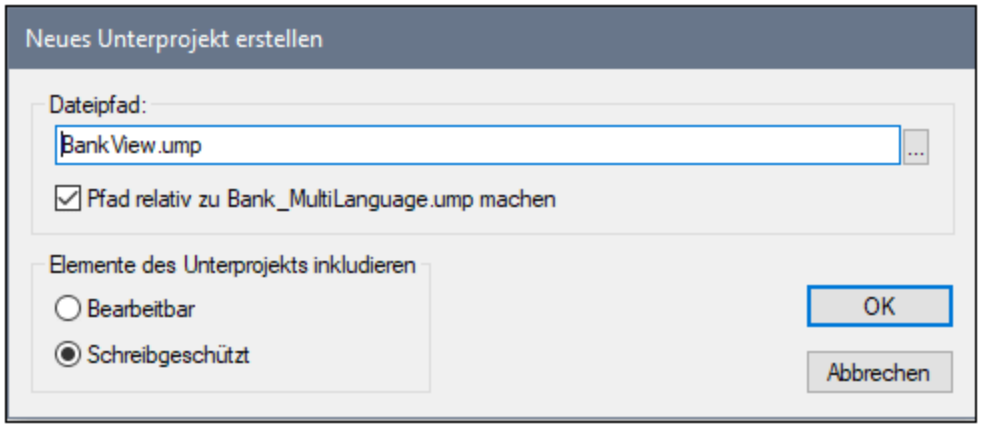


Beim Code Engineering werden alle untergeordneten Komponenten eines Unterprojekts berücksichtigt. Es besteht kein Unterschied zwischen einer einzelnen Projektdatei oder einer Projektdatei, die aus mehreren editierbaren Unterprojekten besteht. Dies gilt auch für UML-Diagramme. Auch diese Diagramme können auf der Haupt- oder Unterprojektebene bearbeitet werden.

Anmerkung: Sie können Pakete und darin enthaltene UML-Diagramme auch in verschiedenen Projekten gemeinsam verwenden. Nähere Informationen dazu finden Sie unter Freigeben von Paketen und Diagrammen 176.

Erstellen von Unterprojekten

Um ein Unterprojekt zu erstellen, klicken Sie mit der rechten Maustaste auf ein Paket und wählen Sie im Kontextmenü den Befehl **Unterprojekt | Neues Unterprojekt erstellen**.



Klicken Sie auf die Schaltfläche **Durchsuchen** und wählen Sie das Verzeichnis, in dem das Unterprojekt gespeichert werden soll, aus.

Aktivieren Sie das Optionsfeld **Bearbeitbar**, damit das Unterprojekt vom Hauptprojekt aus bearbeitet werden kann. (Wenn Sie "Schreibgeschützt" auswählen, kann es im Hauptprojekt nicht bearbeitet werden.)

Anmerkung: Der Dateipfad des Unterprojekts kann jederzeit durch Rechtsklick auf das Unterprojekt und Auswahl des Befehls **Unterprojekt | Dateipfad bearbeiten** geändert werden.

Öffnen und Bearbeiten von Unterprojektdateien

Sie können ein Unterprojekt direkt vom Hauptprojekt aus als eigenes UModel-Projekt öffnen. Dazu sollte es keine nicht aufgelösten Referenzen auf andere Elemente geben. UModel führt bei Erstellung eines Unterprojekts anhand des Hauptprojekts und beim Speichern einer Datei automatisch eine Überprüfung durch.

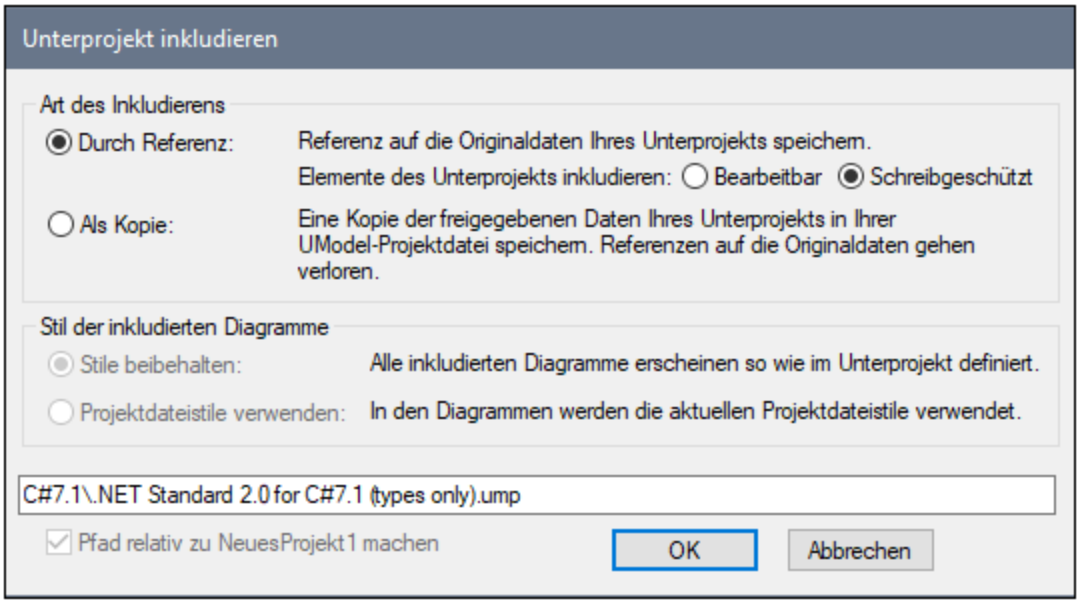
Um ein Unterprojekt als eigenständiges Projekt zu öffnen, klicken Sie mit der rechten Maustaste im Hauptprojekt auf das Unterprojektpaket und wählen Sie den Befehl **Unterprojekt | Als Projekt öffnen.** Daraufhin wird eine weitere Instanz von UModel gestartet und das Unterprojekt wird als Hauptprojekt geöffnet. Alle nicht aufgelösten Referenzen werden im Fenster "Meldungen" angezeigt.

Wiederverwendung von Unterprojekten

Unterprojekte, die von einem Hauptprojekt abgespalten wurden, können in jedem beliebigen anderen Hauptprojekt verwendet werden.

1. Öffnen Sie ein Projekt und wählen Sie den Befehl **Projekt | Unterprojekt inkludieren**.
2. Klicken Sie auf die Schaltfläche "Durchsuchen" und wählen Sie die gewünschte .ump-Datei aus.

© 2016-2022 Altova GmbH



3. Wählen Sie aus, wie die Datei inkludiert werden soll; durch eine Referenz oder als Kopie.

Speichern von Projekten

Wenn Sie die Hauptprojektdatei speichern, werden auch alle editierbaren Unterprojektdateien gespeichert. Sie sollten daher außerhalb der freigegebenen/Unterprojektstruktur keine Daten (Komponenten) erstellen/hinzufügen, wenn das Unterprojekt in einer Hauptprojektdatei als "editierbar" definiert ist. Wenn es noch Daten außerhalb der Unterprojektstruktur gibt, wird im Fenster "Meldungen" eine Warnmeldung angezeigt.

Speichern von Unterprojektdateien

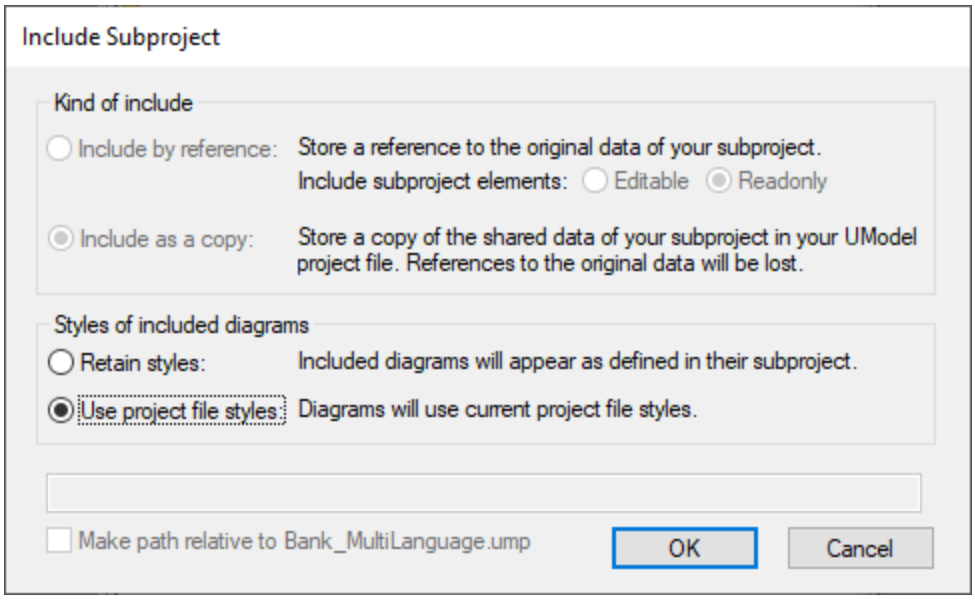
Beim Speichern von Unterprojekten (von der Hauptprojektebene aus) werden alle Referenzen auf gleichrangige und untergeordnete Unterprojekte berücksichtigt und gespeichert. Wenn z.B. zwei gleichrangige Unterprojekte, nämlich sub1 und sub2 existieren und in sub1 Elemente aus sub2 verwendet werden, dann wird sub1 so gespeichert, dass darin automatisch auch Referenzen auf sub2 enthalten sind.

Wenn sub1 als ein Hauptprojekt geöffnet wurde, so wird es als unabhängiges Projekt betrachtet und kann ohne Referenz auf das eigentliche Hauptprojekt bearbeitet werden.

Wiedereingliedern von Unterprojekten in das Hauptprojekt

Zuvor definierte Unterprojekte können wieder zurück in das Hauptprojekt kopiert werden. Wenn das Unterprojekt keine Diagramme enthält, so wird es sofort wieder eingegliedert. Falls das Unterprojekt Diagramme enthält, wird das folgende Dialogfeld angezeigt:

1. Klicken Sie mit der rechten Maustaste auf das Unterprojekt und wählen Sie den Befehl **Unterprojekt | Als Kopie inkludieren**. Daraufhin wird das Dialogfeld "Unterprojekt inkludieren" geöffnet, in dem Sie die Diagrammstile, die Sie beim Inkludieren des Unterprojekts verwenden möchten, definieren können.



2. Wählen Sie die gewünschte Stiloption aus und klicken Sie anschließend auf **OK**.

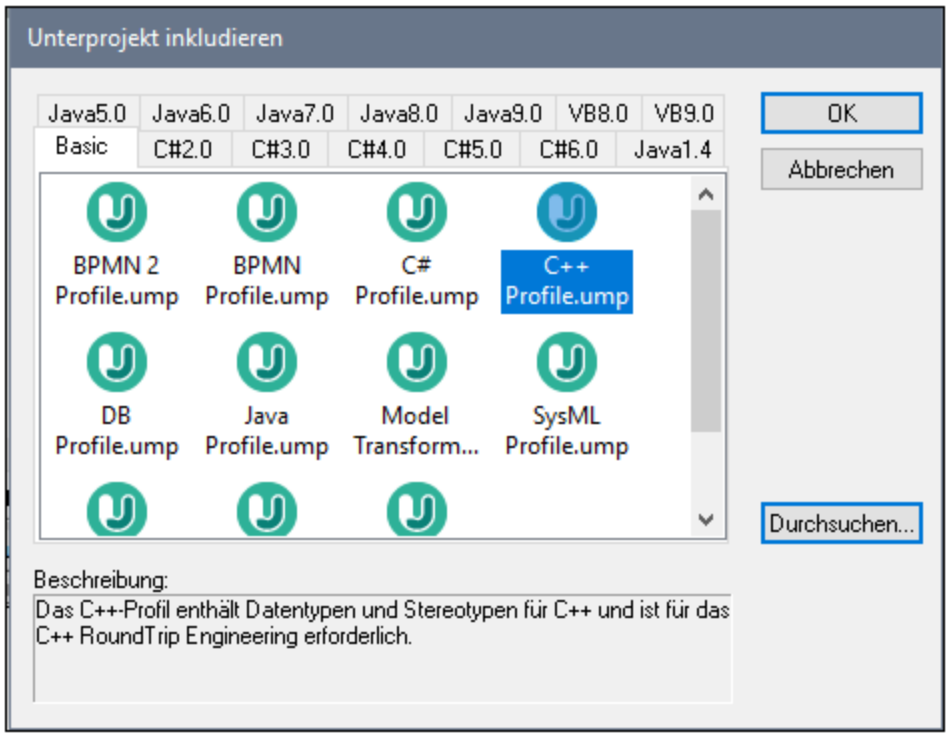
6.1.6 Inkludieren von Unterprojekten

Wenn Sie Code anhand eines Modells generieren oder Quellcode in ein Modell importieren möchten, muss ein Profilprojekt für die jeweilige Sprachen (z.B. für C#, Java, VB.NET) in Ihr UModel-Projekt inkludiert werden.

Um ein UModel-Projekt als Unterprojekt eines anderen UModel-Projekts zu inkludieren, wählen Sie den Menübefehl **Projekt | Unterprojekt inkludieren.** Wie unten gezeigt, stehen auf dem Register **Basic** diverse .ump-Unterprojekte (für das Code Engineering erforderliche Sprachprofile) zur Verfügung. Zusätzlich dazu steht eine Reihe von .ump-Unterprojekten mit C#-, Java- und VB.NET-Typen nach Version geordnet auf den entsprechenden Registern zur Verfügung.

Damit beim Code Engineering alle Typen richtig erkannt werden, müssen sowohl das Sprachprofil (z.B. das **C#-Profil**) als auch das Typprojekt der entsprechenden Sprachversion (z.B. **.NET 5 für C# 9.0**) inkludiert werden, da sonst im Projekt ein Paket namens "Unbekannte externe Elemente", das alle nicht erkannten Typen enthält, erstellt wird. Beachten Sie, dass es für C++ keine Typprojekte sondern nur ein C++-Sprachprofil gibt.

© 2016-2022 Altova GmbH

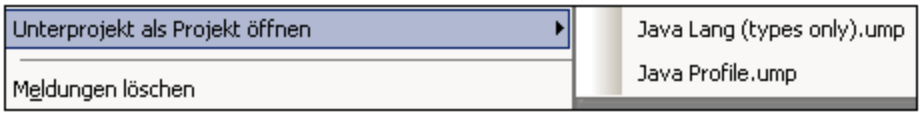


Dialogfeld "Unterprojekt inkludieren"

Die Register und UModel-Projekte im Dialogfeld "Unterprojekt inkludieren" sind konfigurierbar. UModel liest diese Informationen aus dem folgenden Pfad (relativ zum Ordner "Programme" Ihres Betriebssystems): **\Altova\UModel2023\UModelInclude**. Beachten Sie, dass sich die Projektdateien auf dem Register **Basic** direkt unterhalb des Ordners **UModelInclude** befinden, während sich die Projekte auf den verschiedenen Java-, VB- und C#-Registern in Unterordnern des Ordners **UModelInclude** befinden.

So zeigen Sie alle derzeit importierten Projekte an:

- Wählen Sie die Menüoption **Projekt | Unterprojekt separat öffnen**.
  Das Menü, das erscheint, enthält die aktuell inkludierten Unterprojekte.



So erstellen Sie im Dialogfeld "Unterprojekt inkludieren" ein benutzerdefiniertes Register:

- Navigieren Sie zum Ordner **\Altova\UModel2023\UModelInclude** (relativ zu Ihrem Ordner "Programme") und erstellen Sie Ihren benutzerdefinierten Ordner unterhalb, z.B. **\UModelInclude\myfolder.** Als Name des Registers im Dialogfeld "Unterrprojekt inkludieren" wird der Name, den Sie dem Ordner geben, angezeigt.

- Kopieren Sie alle .ump-Dateien, die Sie auf dem entsprechenden Register zur Verfügung stellen möchten, in Ihren benutzerdefinierten Ordner.

So erstellen Sie beschreibenden Text zu jeder UModel Projektdatei:

- Erstellen Sie eine Textdatei mit demselben Namen wie die *.ump-Datei und platzieren Sie sie in denselben Ordner. So wird z.B. für die Datei **MyModel.ump** eine beschreibende Datei names **MyModel.txt** benötigt. Stellen Sie bitte sicher, dass die Codierung dieser Textdatei UTF-8 ist.

So entfernen Sie ein inkludiertes Projekt:

1. Klicken Sie in der Modell-Strukturansicht auf das inkludierte Paket und drücken Sie die Entf-Taste.
2. Wenn Sie gefragt werden, ob Sie mit dem Löschen fortfahren wollen, klicken Sie auf OK, um die inkludierte Datei aus dem Projekt zu löschen.

So löschen oder entfernen Sie ein Projekt aus dem Dialogfeld "Unterprojekt inkludieren":

- Löschen oder entfernen Sie die .ump-Datei (MyModel.ump) aus dem entsprechenden Ordner.

6.1.7 Freigeben von Paketen und Diagrammen

Pakete (und eventuell darin enthaltene UML-Diagramme) können freigegeben und in verschiedenen UMode-Projekten verwendet werden. Pakete können mittels Referenz oder in Form einer Kopie in andere UModel-Projekte inkludiert werden.

Beachten Sie bitte außerdem, dass Unterprojektdateien jederzeit von einem Haupt- oder Unterprojekt abgeteilt werden können. Die Unterprojektdateien können vom Hauptprojekt aus als editierbare oder als schreibgeschützte Projekte inkludiert werden; die einzelnen Pakete werden freigegeben und als Unterprojektdatei gespeichert und können zu einem Versionskontrollsystem hinzugefügt werden. Nähere Informationen dazu finden Sie unter Aufteilen von UModel-Projekten 170.

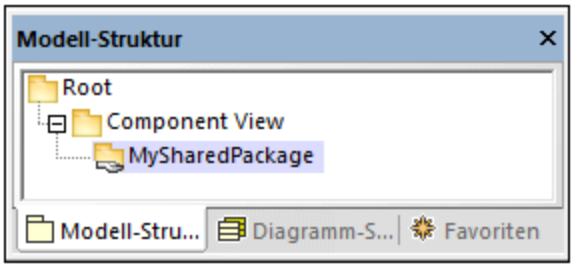
Anmerkungen

- Damit ein Paket freigegeben werden kann, darf es keine Links zu externen Elementen (Elementen außerhalb des freigegebenen Bereichs) enthalten.
- Wenn Sie UModel-Projektdateien erstellen, verwenden Sie nicht eine einzige Projektdatei als "Vorlage/Kopie" für eine andere Projektdatei, in der Sie ein Paket freigeben möchten. Dies kann zu Konflikten führen, da jedes Element global nur einmal vorhanden sein sollte (siehe uuid 664). Dies wäre dann nicht der Fall, da zwei Projekte dann Elemente mit identischen uuids hätten.

So verwenden Sie ein Paket in mehreren Projekten:

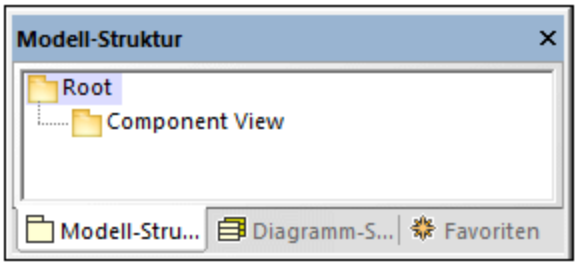
- Rechtsklicken Sie auf dem Register "Modell-Struktur" auf ein Paket und wählen Sie **Unterprojekt | Paket freigeben**. In der Modell-Struktur wird unterhalb des freigegebenen Pakets ein "freigegeben"-Symbol angezeigt. Dieses Paket kann nun in jedes beliebige andere UModel-Projekt inkludiert werden.

© 2016-2022 Altova GmbH

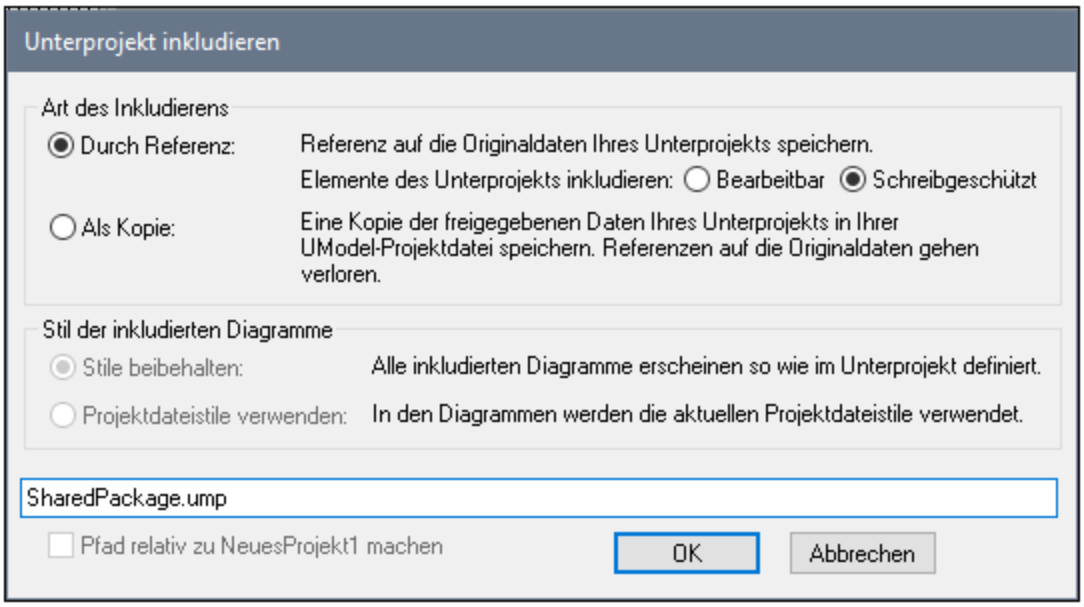


So inkludieren/importieren Sie einen freigegebenen Ordner in ein Projekt:

1. Öffnen Sie das Projekt, das das freigegebene Paket enthalten soll (in diesem Beispiel ein leeres Projekt).

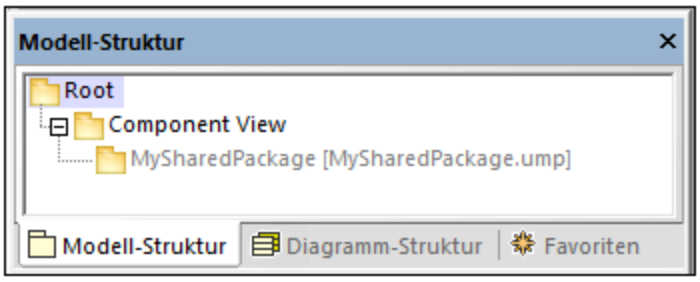


2. Wählen Sie den Menüeintrag **Projekt | Unterprojekt inkludieren...**
3. Klicken Sie auf die **Durchsuchen**-Schaltfläche, wählen Sie das Projekt, das das freigegebene Paket enthält, aus und klicken Sie auf **Öffnen**. Im Dialogfeld "Unterprojekt inkludieren" können Sie das Paket/Projekt in Form einer Referenz oder als Kopie inkludieren.



4. Wählen Sie die gewünschte Option (In diesem Beispiel "Durch Referenz") und klicken Sie auf OK.

Im neuen Paket ist nun das Paket "Deployment View" zu sehen. Das Quellprojekt des Pakets wird in Klammern angezeigt (in diesem Beispiel **SharedPackage.ump**).

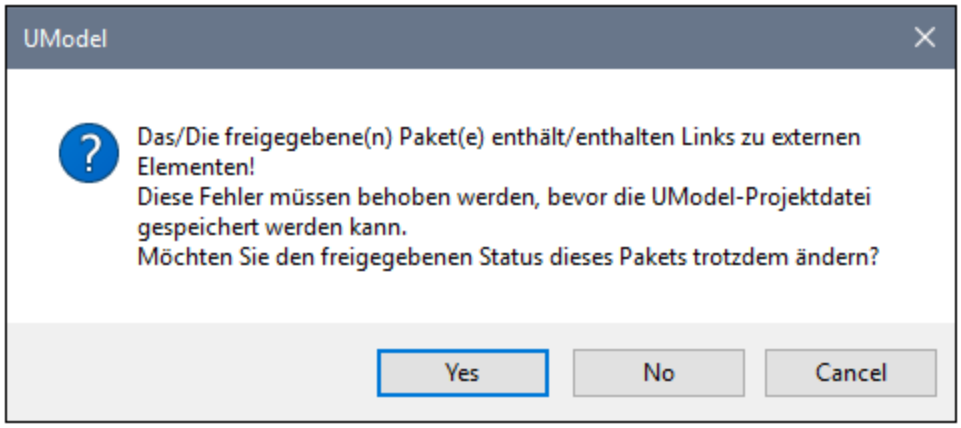


Anmerkungen:

- Wenn Sie ein Quellprojekt inkludieren, das Unterprojekte enthält, werden auch alle Unterprojekte des Quellprojekts in das Zielprojekt inkludiert.
- Freigegebene Ordner, die in Form einer Referenz inkludiert wurden, können jederzeit durch Rechtsklick auf den Ordner und Auswahl von **Unterprojekt | Als Kopie** in "als Kopie" geändert werden.

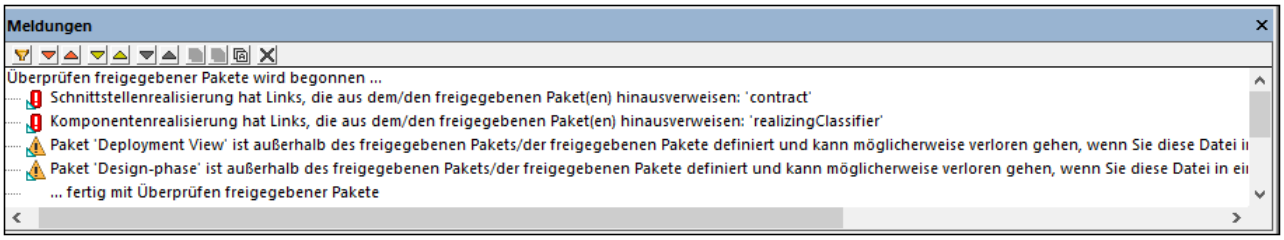
Auflösen von Links zu externen Elementen

Wenn Sie versuchen ein Paket freizugeben, das Links zu externen Elementen aufweist, erscheint ein Dialogfeld mit einer Warnmeldung. So wird etwa die folgende Meldung angezeigt, wenn Sie versuchen, das Paket "Deployment View" des Beispielprojekts **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial\BankView-start.ump** freizugeben.



Klicken Sie auf **Ja**, um das Paket trotz der Fehler freizugeben. Das Fenster "Meldungen" enthält Informationen über die einzelnen externen Links.

© 2016-2022 Altova GmbH



Klicken Sie im Fenster "Meldungen" auf einen Eintrag, um das entsprechende Element im Fenster "Modell-Struktur" anzuzeigen.

6.1.8 Verbesserung der Performance

Da Modellierungsprojekte manchmal relativ groß werden können, gibt es Möglichkeiten, die Performance beim Modellieren zu verbessern:

- Stellen Sie sicher, dass Sie die neuesten Treiber für Ihre jeweilige Grafikkarte verwenden (tun Sie dies, bevor Sie die folgenden Tipps befolgen)
- Deaktivieren Sie die Syntaxfärbung - **Register "Stile" | Syntaxfarben verwenden** = false.
- Deaktivieren Sie "gradient" als Hintergrundfarbe für Diagramme, verwenden Sie eine einheitliche Farbe (Setzen Sie z.B. im Fenster "Stile" die Eigenschaft **Diag. Hintergrundfarbe** auf eine einheitliche Farbe, z.B. Weiß).
- Die automatisch aktivierte Autokomplettierung kann über **Extras | Optionen | Diagrambearbeitung** und Deaktivieren des Kontrollkästchens **Automatische Eingabehilfe aktivieren** deaktiviert werden.

6.2 Generieren von Programmcode

Nachdem Sie das Modell Ihrer Applikation in UModel erstellt haben (z.B. ein oder mehrere Klassendiagramme), können Sie schnell ein Prototyp-Projekt generieren, das alle definierten Schnittstellen, Klassen, Operationen, usw. in der Sprache Ihrer Wahl enthält. Sie können mit UModel auf Basis der in Ihrem UModel-Projekt gefundenen Elemente (wie z.B. Schnittstellen, Klassen, Operationen, usw.) anhand Ihres Modells C++-, C#-, VB.NET- oder Java-Programmcode generieren. Dieser Vorgang wird auch als "Forward Engineering" bezeichnet. Im generierten Code werden alle Objekte genauso generiert, wie sie im Modell definiert wurden, sodass Sie zu ihrer eigentlichen Implementierung übergehen können.

Die Codegenerierung kann auch auf XML-Schemas und Datenbanken angewendet werden*. So könnten Sie etwa ein XML-Schema oder eine Datenbank mit UModel erstellen und anschließend die entsprechende Datei (oder das entsprechende SQL-Skript im Fall von Datenbanken) anhand des Modells generieren. Konsultieren Sie dazu die Zuordnungstabellen, um herauszufinden, welche Schema- oder Datenbankelemente UModel-Elementen zugeordnet werden, siehe UModel-Elementzuordnungen 247.

** Für die Generierung von Datenbanken benötigen Sie die UModel Enterprise oder Professional Edition.*

Voraussetzungen

Damit Programmcode generiert werden kann, muss das UModel-Projekt die folgenden Mindestvoraussetzungen erfüllen:

- Eines der Pakete in Ihrem Projekt muss als Namespace Root definiert sein. Bei der Namespace Root kann es sich um einen C++, C#-, Java-, VB.NET-, XSD- oder Datenbank-Namespace handeln. Dieses Paket muss alle Klassen und Schnittstellen enthalten, anhand welcher der Code generiert werden soll. Nähere Informationen dazu finden Sie unter Definieren eines Pakets als Namespace Root 180.
- Zum Projekt muss eine Code Engineering-Komponente hinzugefügt werden. Diese muss von allen Klassen oder Schnittstellen, für die Code generiert werden soll, realisiert werden. Nähere Informationen dazu finden Sie unter Hinzufügen einer Code Engineering-Komponente 181.
- Im Fall von Datenbanken muss zuerst über die Menüoption **Projekt | SQL-Datenbank importieren** eine Verbindung zur Zieldatenbank hergestellt werden. Sobald die Verbindung hergestellt ist, können Sie eine Datenbankstruktur im Modell erstellen oder ändern und die Änderungen über ein SQL-Skript in der Datenbank übernehmen (siehe auch UModel und Datenbanken 555).

Zusätzlich dazu sollten Sie eines der vordefinierten UModel-Unterprojekte für die entsprechende Sprache (oder Sprachversion) inkludieren, siehe Inkludieren anderer UModel-Projekte 174. Wenn Ihre Applikation z.B. für eine bestimmte Version von C#, Java oder VB.NET geschrieben wird, könnten Sie dadurch beim Erstellen Ihrer UML-Klassen, Schnittstellen, usw. die entsprechenden Datentypen verwenden.

Ein Beispiel zum Erstellen eines Projekts von Grund auf und Generieren von Code anhand des Projekts finden Sie unter Beispiel: Generieren von Java-Code anhand des Modells 193 und Beispiel: Generieren von C++-Code 202.

6.2.1 Definieren eines Pakets als Namespace Root

Um anhand Ihres UModel-Projekts Programmcode zu generieren, muss ein Paket in Ihrem Modell als Namespace Root definiert werden.

 © 2016-2022 Altova GmbH

So definieren Sie ein Paket als Namespace Root:

- Klicken Sie mit der rechten Maustaste auf ein Paket im Fenster-Modell-Struktur[86] und wählen Sie im Kontextmenü den Befehl **Code Engineering | Als <...> Namespace Root definieren,** wobei **<...>** für C++, C#, Java, VB.NET, XSD, Datenbank steht.

Wenn Sie ein Paket als Namespace Root definieren, informiert Sie UModel darüber, dass auch das UML-Profil der entsprechenden Sprache zum Projekt hinzugefügt und auf das ausgewählte Paket angewendet wird. Klicken Sie auf OK, um dies zu bestätigen, wenn Sie in einem Dialogfeld wie dem unten gezeigten danach gefragt werden.



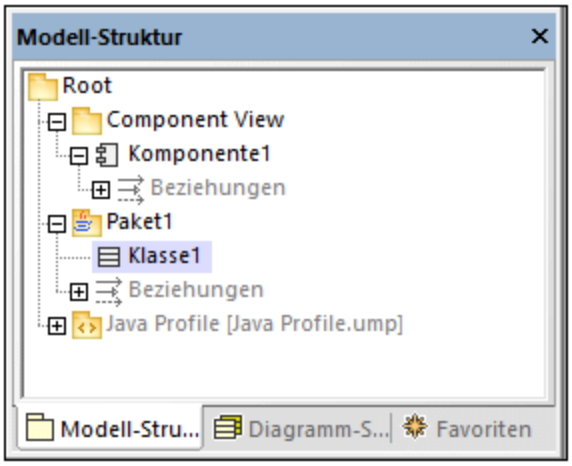
6.2.2 Hinzufügen einer Code Engineering-Komponente

Um Programmcode zu generieren, muss Ihr UModel-Projekt eine Code Engineering-Komponente enthalten, in der alle Einzelheiten zur Codegenerierung definiert sind (z.B. welche Klassen aus dem Projekt bei der Codegenerierung berücksichtigt werden sollen und in welchem Zielverzeichnis der Code gespeichert werden soll). Wie unten gezeigt, muss die Komponente die folgenden Kriterien erfüllen, damit Code generiert werden kann:

- Der Komponente muss ein physischer Ordner zugewiesen werden, in dem der Code generiert wird.
- Die Klassen oder Schnittstellen, die am Code Engineering beteiligt sind, müssen von der Komponente realisiert werden.
- Für die Komponente muss die Eigenschaft **für Code Engineering verwenden** aktiviert sein.

So fügen Sie eine Komponente hinzu, die die gewünschten Klassen oder Schnittstellen realisiert:

1. Klicken Sie in der Modell-Struktur mit der rechten Maustaste auf ein Paket und wählen Sie im Kontextmenü den Befehl **Neues Element | Komponente**. Daraufhin wird die Komponente zum Modell hinzugefügt.
2. Klicken Sie in der Modell-Struktur auf die Klasse oder Schnittstelle, die von der Komponente realisiert werden soll und ziehen Sie diese anschließend mit dem Cursor auf die Komponente (in diesem Beispiel wurde `Klasse1` aus `Paket1` auf `Komponente1` gezogen). Dadurch wird in der Modell-Struktur automatisch eine `Komponentenrealsierungsbeziehung` erstellt.



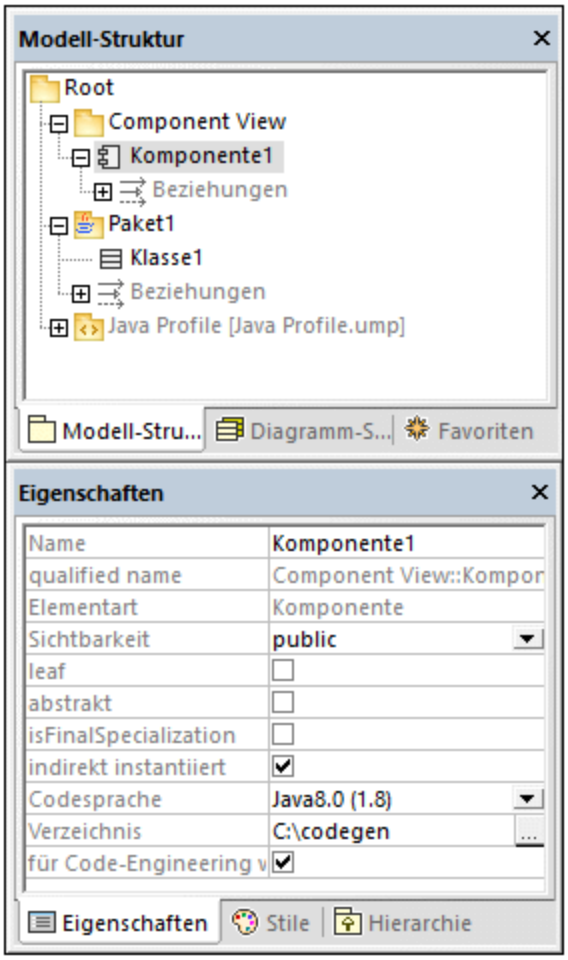
Sie können dies auch auf eine andere Art tun, nämlich indem Sie ein Komponentendiagramm erstellen und anschließend zwischen der Komponente und den Klassen oder Schnittstellen eine `Komponentenrealisierungsbeziehung` ziehen. Nähere Informationen dazu finden Sie unter Komponentendiagramme[54].

So bereiten Sie eine Komponente für das Code Engineering vor:

1. Wählen Sie die Komponente in der Modell-Struktur aus (es wird davon ausgegangen, dass diese Komponente, wie oben erläutert, bereits von mindestens einer Klasse oder Schnittstelle realisiert wird).
2. Suchen Sie im Fenster "Eigenschaften" die Eigenschaft **Verzeichnis** und definieren Sie dafür den Pfad, unter dem Sie Code generieren möchten.
3. Aktivieren Sie im Fenster "Eigenschaften" das Kontrollkästchen **für Code Engineering verwenden**.

So sehen Sie etwa in der Abbildung unten, dass die Komponente **Komponente1** aus dem Paket **Component View** so konfiguriert ist, dass damit Java 8.0 Code im Verzeichnis **C:\codegen**: generiert wird.

© 2016-2022 Altova GmbH



6.2.3 Überprüfen der Projektsyntax

Es ist wichtig, dass Sie die Syntax des Projekts überprüfen, bevor Sie anhand des Modells Code generieren. Bei der Syntaxüberprüfung werden Sie über alle Probleme, durch die eine Codegenerierung verhindert würde, informiert. Die Projektsyntaxüberprüfung erfolgt über den Menübefehl **Projekt | Projektsyntax überprüfen** (oder drücken Sie alternativ dazu **F11**). Auch bevor Code anhand des Modells aktualisiert wird, wird automatisch eine Syntaxüberprüfung durchgeführt. Die Ergebnisse (Fehler, Warnungen und Informationsmeldungen) werden im Fenster "Meldungen" angezeigt.

Bei der Syntaxüberprüfung wird die Projektdatei, wie aus den Tabellen unten ersichtlich, auf mehreren Ebenen überprüft. Beachten Sie die folgenden Punkte:

- Informationen zur Behebung häufiger Syntaxfehler finden Sie im Kapitel Codegenerierung "Voraussetzungen" 180.
- Die unten aufgelisteten Überprüfungen werden nur dann für Komponenten durchgeführt, wenn für die Komponente im Fenster "Eigenschaften" die Eigenschaft **für Code Engineering verwenden** aktiviert ist.

- Für Klassen, Schnittstellen und Enumerationen werden die unten angeführten Überprüfungen nur durchgeführt, wenn die Klasse, Schnittstelle, Enumeration im Codesprachen-Namespace enthalten ist, d.h. sie muss sich unterhalb eines Pakets befinden, das als Namespace Root definiert wurde.
- Constraints für Modellelemente werden nicht überprüft, da sie nicht Teil der Codegenerierung bilden, siehe Einschränken von Elementen 121.

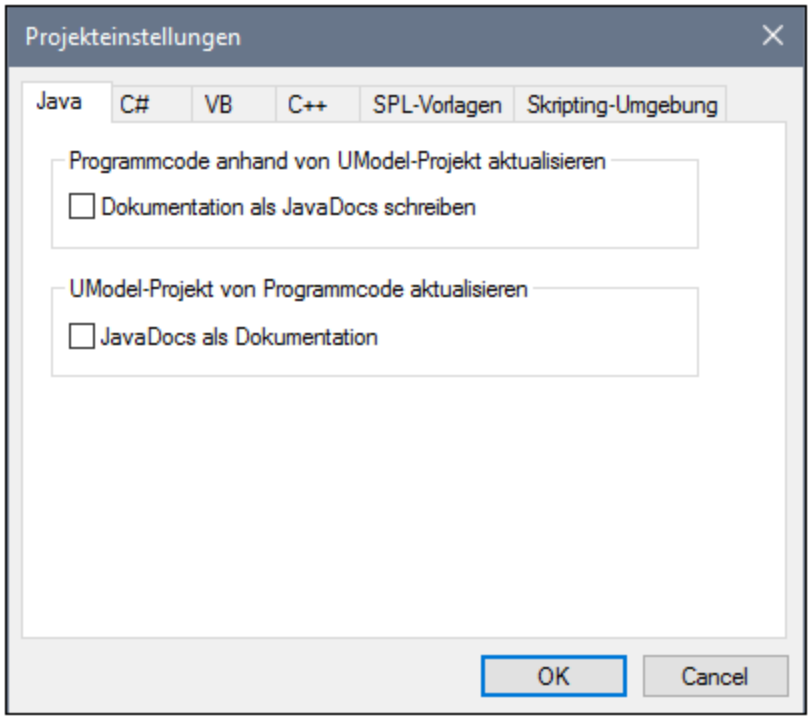
Ebene	Überprüfung	Fehlerschweregrad, falls die Überprüfung fehlschlägt
Projekt	...ob mindestens ein Namespace Root-Paket vorhanden ist.	Fehler
Komponente	...ob die Projektdatei oder das Verzeichnis definiert ist.	Fehler
	...ob diese Komponente eine `Komponentenrealisierungsbeziehung` zu mindestens einer Klasse oder Schnittstelle aufweist.	Fehler
Klasse	...ob der Codedateiname definiert ist. **Anmerkung:** Diese Überprüfung gilt nicht für verschachtelte Klassen.	*Fehler,* wenn die Option **Fehlende Codedateinamen generieren** auf dem Register **Extras | Optionen | Code Engineering** nicht aktiviert wurde. *Warnung,* wenn die Option aktiviert ist.
	...ob für Operationsparameter der Typ definiert ist.	Fehler
	...ob für Eigenschafte der Typ definiert ist.	Fehler
	...ob der Operationsrückgabetyp definiert ist.	Fehler
	...ob Operationen (Namen + Parametertypen) doppelt vorhanden sind.	Fehler
	...ob eine `Komponentenrealisierungsbeziehung` zu einer Komponente besteht. **Anmerkung:** Diese Überprüfung gilt nicht für verschachtelte Klassen.	Warnung
	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler
	...ob eine Mehrfachvererbung vorkommt	Fehler
Klassenoperation	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler
	...ob ein Rückgabeparameter vorhanden ist.	Fehler
Klassen-operations-	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler

© 2016-2022 Altova GmbH

Ebene	Überprüfung	Fehlerschweregrad, falls die Überprüfung fehlschlägt
parameter	...ob der Typ gültig ist	Fehler
Schnittstelle	...ob der Codedateiname definiert ist.	*Fehler,* wenn die Option **Fehlende Codedateinamen generieren** auf dem Register **Extras | Optionen | Code Engineering** nicht aktiviert wurde. *Warnung,* wenn die Option aktiviert ist.
	..ob die Schnittstelle im Codesprachen-Namespace enthalten ist.	Fehler
	...ob für Eigenschaften der Typ definiert ist.	Fehler
	...ob für Operationsparameter der Typ definiert ist.	Fehler
	...ob der Operationsrückgabetyp definiert ist.	Fehler
	...ob Operationen (Namen + Parametertypen) doppelt vorhanden sind.	Fehler
	...ob Schnittstellen an einer Komponentenrealisierung beteiligt sind.	Warnung
	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler
Schnittstellen-operation	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler
Schnittstellen-operations-parameter	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler
Schnittstellen-eigenschaften	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort)	Fehler
Paket	...ob der Name gültig ist (keine verbotenen Zeichen, Name ist kein Schlüsselwort) **Anmerkung:** Diese Überprüfung wird angewendet, wenn sich das Paket innerhalb eines Namespace Root-Pakets befindet und darauf über das Fenster "Eigenschaften" der <<namespace>> Stereotyp angewendet wurde.	Fehler
Enumeration	...ob eine `Komponentenrealisierungsbeziehung` zu einer Komponente besteht.	Warnung

6.2.4 Codegenerierungsoptionen

Wenn Sie anhand von Programmcode ein UModel-Projekt generieren, können Sie die unten aufgelisteten Optionen definieren bzw. ändern. Diese Optionen stehen zur Verfügung, wenn Sie den Menübefehl **Projekt | Projekteinstellungen** auswählen. Sie werden zusammen mit dem Projekt gespeichert.



Die Optionen sind folgendermaßen auf Registern gruppiert.

Register	Optionen
Java	Aktivieren Sie das Kontrollkästchen **Dokumentation als JavaDocs schreiben,** um die Dokumentation von UModel-Elementen im generierten Code in die entsprechende Dokumentation für JavaDocs zu konvertieren.
C#	Aktivieren Sie das Kontrollkästchen **Dokumentation als DocComments verfassen,** um die Dokumentation von UModel-Elementen im generierten Code in Kommentare zu konvertieren.
VB	Aktivieren Sie das Kontrollkästchen **Dokumentation als DocComments verfassen,** um die Dokumentation von UModel-Elementen im generierten VB.NET-Code in Kommentare zu konvertieren.
C++	Siehe Optionen für den Code-Import 212.
SPL-Vorlagen	Wenn UModel SPL-Vorlagen über einen benutzerdefinierten Pfad auslesen soll, der nicht der Standardpfad ist, muss der benutzerdefinierte Pfad hier eingegeben werden. Siehe auch SPL-Vorlagen 207.

© 2016-2022 Altova GmbH

Register	Optionen
Skripting-Umgebung	Optionen auf diesem Register werden nur angewendet, wenn Sie UModel-Skripting-Projekte so definiert haben, dass sie verschiedene Ereignisse behandeln, oder das Verhalten Ihrer UModel-Projekte anpassen. Nähere Informationen dazu finden Sie unter Skript-Editor 809

Zusätzlich zu den oben angeführten Einstellungen gibt es noch einige weitere Einstellungen, die sich auf die Codegenerierung auswirken. Um diese aufzurufen, wählen Sie den Menübefehl **Extras | Optionen** und klicken Sie auf das Register **Code Engineering**. Die Einstellungen zum Generieren von Code anhand eines Modells befinden sich unter **Programmcode anhand von UModel-Projekt aktualisieren**. Beachten Sie, dass diese Einstellungen lokal sind (sie wirken sich nur auf die aktuelle Installation von UModel aus und werden nicht mit dem Projekt gespeichert).

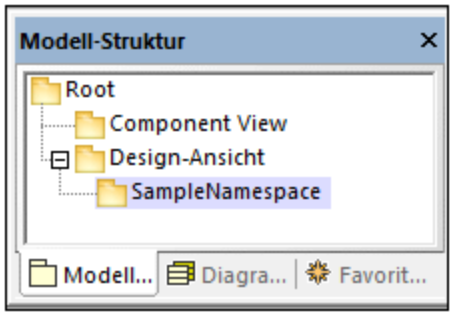
6.2.5 Beispiel: Generieren von C#-Code

In diesem Beispiel wird gezeigt, wie Sie mit UModel C#-Code generieren. Zuerst werden wir einen C#-Beispiel-Namespace, der einige Klassen enthält, erstellen, das Projekt für die Codegenerierung konfigurieren und anschließend den eigentlichen Code generieren.

Die Zielplattform in diesem Beispiel ist *.NET Standard 2.0 für C# 7.1*. Dies ist dank eines in UModel vordefinierten Profils, in dem alle Typen des *.NET Standard 2.0 für C# 7.1* vordefiniert sind, möglich. UModel enthält auch vordefinierte Profile für bestimmte .NET Framework-Versionen, siehe auch Inkludieren von Unterprojekten 174.

Erstellen eines neue Projekts und seiner Struktur

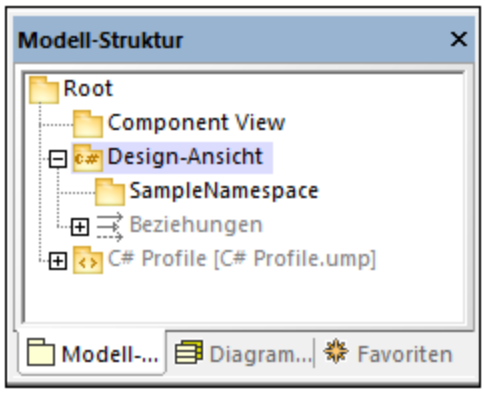
Im ersten Schritt wird ein leeres Projekt erstellt, das zwei Standardpakete hat (`Root` und `Component View`): Klicken Sie im Menü **Datei** oder in der Symbolleiste auf **Neu**. Klicken Sie als nächstes auf das `Root`-Paket und erstellen Sie einige weitere Pakete, wie unten gezeigt. Wenn Sie mit der grafischen Benutzeroberfläche von UModel noch nicht vertraut sind, lesen Sie zuerst die Kapitel UModel Tutorial 18 und Anleitung zur Modellierung von... 111.



In diesem Beispiel dient das Paket `Design-Ansicht` als Container für den Design-Teil Ihres Modells (z.B. Klassen und Klassendiagramme), während das Paket `SampleNamespace` als Namespace für alle Klassen, die erstellt werden sollen, verwendet wird. Sie können Ihre Pakete bei Bedarf auch anders strukturieren.

Code Engineering

Im nächsten Schritt wird C# für Ihr Paket definiert. Klicken Sie mit der rechten Maustaste auf das Paket `Design-Ansicht` und wählen Sie im Kontextmenü den Befehl **Code Engineering | Als C# Namespace Root definieren**. UModel informiert Sie darüber, dass das C#-Profil auf das Paket angewendet wird. Klicken Sie auf **OK.** Das in UModel vordefinierte C#-Profil wird nun in das Projekt inkludiert (*siehe Abbildung unten*).



Definieren von SampleNamespace als Namespace

Klicken Sie als nächstes auf das Paket `SampleNamespace` und aktivieren Sie im Fenster **Eigenschaften** das Kontrollkästchen `<<namespace>>`. Dadurch wird das `namespace`-Stereotyp auf das Paket angewendet und sein Symbol ändert sich in . Sie können jetzt unter diesem Namespace Klassen erstellen.

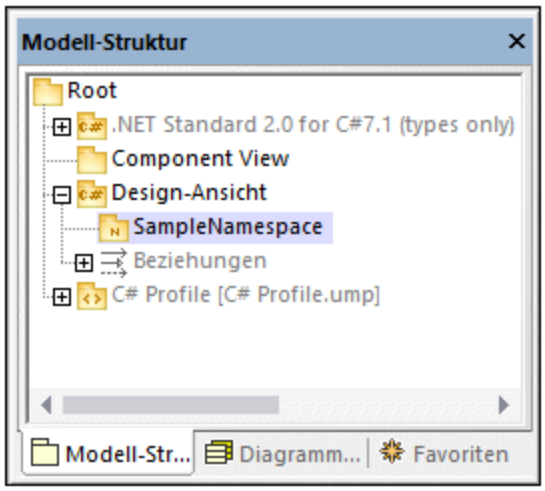
Inkludieren eines Unterprojekts

Das Modell enthält bisher das C#-Profil, das die Datentypen für C# enthält. Allerdings enthält es die .NET Standard 2.0-spezifischen Datentypen noch nicht (diese stehen in einem separaten UModel-Profil zur Verfügung). Um dieses Profil zum Projekt hinzuzufügen, gehen Sie folgendermaßen vor:

1. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.
2. Wechseln Sie zum Register **C#** und wählen Sie den Eintrag *.NET Standard 2.0 for C# 7.1 (types only)* aus.
3. Wählen Sie im Dialogfeld **Unterprojekt inkludieren** *Durch Referenz* aus und klicken Sie auf **OK**.

Das zusätzliche Profil wurde nun zum Projekt hinzugefügt (*siehe unten*).

© 2016-2022 Altova GmbH



Erstellen von C#-Klassen

Im nächsten Schritt werden Klassen erstellt. Sie können dies entweder direkt über das Fenster **Modell-Struktur** oder über ein Klassendiagramm tun. In diesem Beispiel haben wir uns für die zweite Option entschieden. Gehen Sie folgendermaßen vor:

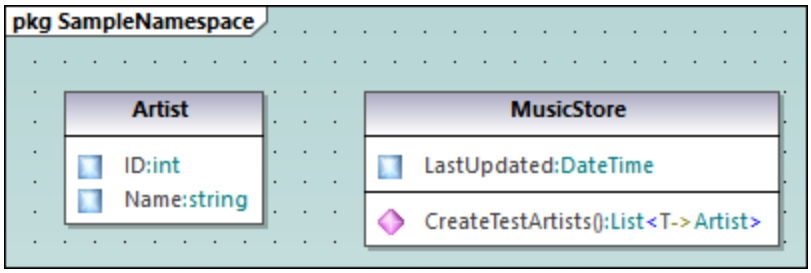
1. Öffnen Sie das Fenster **Diagramm-Struktur**.
2. Klicken Sie mit der rechten Maustaste auf **Klassendiagramme** und wählen Sie **Neues Diagramm | Klassendiagramm**.

In diesem Beispiel wird vorausgesetzt, dass alle Ihre Klassen unter dem Namespace `SampleNamespace` generiert werden müssen. Wenn Sie daher aufgefordert werden, einen Owner für das Diagramm auszuwählen, wählen Sie das Paket `SampleNamespace` aus. Wenn Sie ein anderes Paket auswählen, gehören alle Elemente, die Sie zum Diagramm hinzufügen zum selben Paket wie das Diagramm (was nicht notwendigerweise beabsichtigt ist).

Erstellen von Klassen und ihrer Struktur

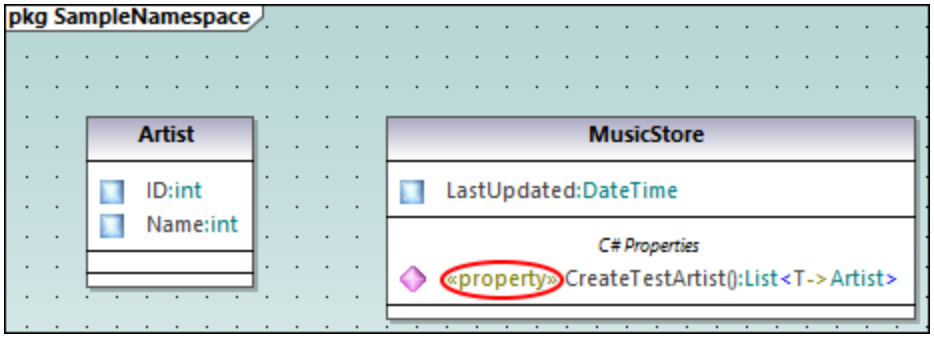
Erstellen Sie als nächstes Klassen, Typen und andere in Ihrem Modell erforderlichen Elemente. Sie können für unser Beispiel ein einfaches Diagramm, das eine `Artist`- und eine `MusicStore`-Klasse enthält, erstellen (*siehe Abbildung unten*). Gehen Sie folgendermaßen vor:

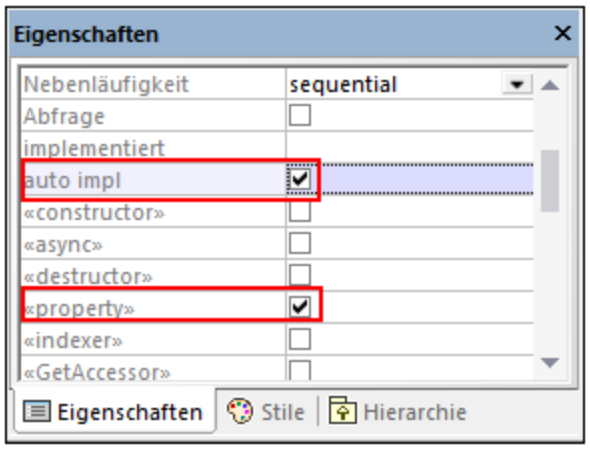
1. Klicken Sie mit der rechten Maustaste in das *pkg SampleNamespace*-Fenster und wählen Sie **Neu | Klasse**.
2. Geben Sie dieser Klasse den Namen `Artist`.
3. Klicken Sie mit der rechten Maustaste in das Kästchen `Artist` und erstellen Sie zwei Eigenschaften: `ID` von Typ `int` und `Name` vom Typ `string`.
4. Erstellen Sie die zweite Klasse mit dem Namen `MusicStore`.
5. Erstellen Sie eine Eigenschaft namens `LastUpdated` vom Typ `DateTime`.
6. Erstellen Sie eine Operation und geben Sie ihren Namen und ihre Definition, wie unten gezeigt, ein.



Nähere Informationen zum Erstellen von Klassen und ihren Mitgliedern finden Sie in den Kapiteln Klassendiagramme[31] und Anleitung zur Modellierung von...[111].

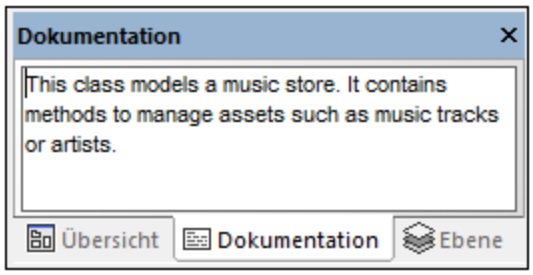
Über automatisch implementierte C#-Eigenschaften
Sie sehen in UModel, ob C#-Eigenschaften automatisch implementiert wurden. Die Autoimplementierungsoption steht zur Verfügung, nachdem im Fenster **Eigenschaften** das Kontrollkästchen `property` (in unserem Beispiel für `CreateTestArtist()`) aktiviert wurde (*siehe Abbildungen unten*).





Hinzufügen von Dokumentation (optional)
Klicken Sie optional im Diagramm auf die Klasse `MusicStore` und fügen Sie durch Eingabe von Text in das Fenster "Dokumentation"[97] Dokumentation hinzu (*siehe Abbildung unten*). Auf diese Art können Sie Codekommentare zu dieser Klasse generieren.

© 2016-2022 Altova GmbH

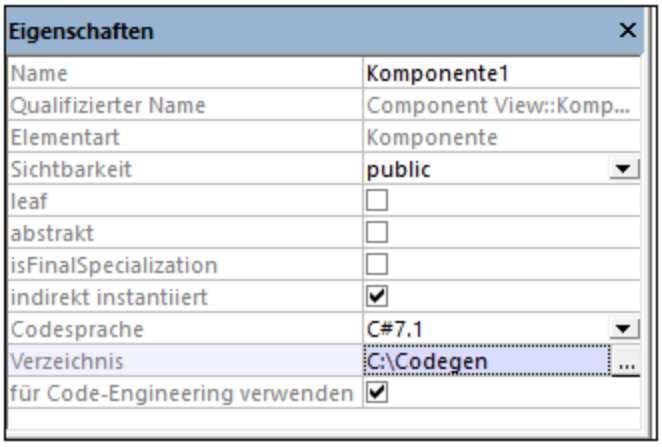


Konfigurieren des Projekts für das Code Engineering

Im nächste Schritt müssen wir nun Code Engineering-Einstellungen definieren. Gehen Sie folgendermaßen vor:

1. Speichern Sie das Projekt in einem Verzeichnis.
2. Klicken Sie anschließend im **Fenster Modell-Struktur** mit der rechten Maustaste auf das Paket `Component View` und fügen Sie eine neue **Komponente** (d.h. eine Software-Komponente) hinzu.
3. Klicken Sie auf die neue Software-Komponente und definieren Sie im Fenster **Eigenschaften** die folgenden Eigenschaften (*siehe Abbildung unten*):

   - Setzen Sie die Codesprache der Komponente z.B. auf C# 7.1.
   - Wählen Sie das Verzeichnis für die Codegenerierung aus (`c:\codegen` in unserem Beispiel).
   - Aktivieren Sie das Kontrollkästchen *für Code Engineering verwenden*.



Erstellen einer Komponentenrealisierungsbeziehung

Erstellen Sie als nächstes zwischen den Klassen, anhand welcher C#-Code generiert werden muss, eine `Komponentenrealisierungsbeziehung`. Sie können dies folgendermaßen tun: Klicken Sie im Fenster **Modell-Struktur** auf die Klasse, die von der Komponente realisiert werden soll (in diesem Beispiel `Artist`) und ziehen Sie sie mit der Maus in die Code Engineering-Komponente (`Komponente1`) (*siehe Abbildung unten*). Führen Sie denselben Schritt auch für die Klasse `MusicStore` durch.

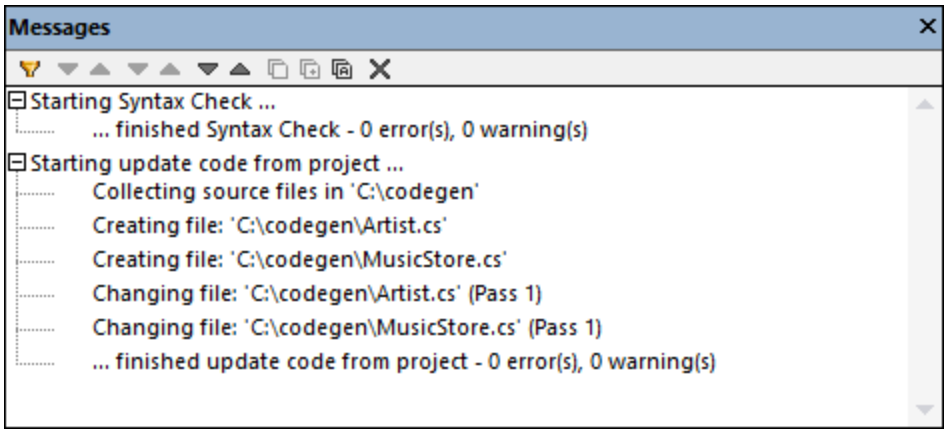


Anmerkung: Falls Sie vergessen haben, für eine Klasse eine `Komponentenrealisierungsbeziehung` zu erstellen, generiert UModel dennoch die entsprechende Codedatei, gibt im Fenster **Meldungen** aber Warnungen aus. Diese Einstellung kann über **Extras | Optionen | Register Code Engineering** konfiguriert werden (Kontrollkästchen *Fehlende Komponentenrealisierungen generieren*).

Generieren von C#-Code

Im letzten Schritt wird nun der eigentliche C#-Code generiert. Gehen Sie folgendermaßen vor:

1. Klicken Sie im Menü **Projekt** auf **Merge Programmcode aus UModel-Projekt**. Daraufhin wird ein Dialogfeld angezeigt, in dem Sie einstellen können, ob Änderungen im Modell mit denjenigen im Code zusammengeführt oder diese gegebenenfalls überschrieben werden sollen. Wählen Sie in diesem Beispiel **...überschreiben**, da ein neues Projekt generiert wird.
2. Damit die Klassendokumentation im generierten Code in Form von Kommentaren inkludiert wird, klicken Sie auf **Projekt | Projekteinstellungen** und aktivieren Sie das Kontrollkästchen **Dokumentation als DocComments verfassen**. Nähere Informationen dazu finden Sie unter Codegenerierungsoptionen 186.
3. Klicken Sie auf **OK.** Das Ergebnis des Code Engineering wird im Fenster **Meldungen** angezeigt (*siehe unten*).



Wenn Sie zur Klasse `MusicStore` Dokumentation hinzugefügt haben, sehen Sie diese im generierten Code in Form von Codekommentaren:

```
using System;
using System.Collections.Generic;
namespace SampleNamespace
{
   /// This class models a music store. It contains methods to manage assets such as
music tracks or artists.
   public class MusicStore
   {
```

 © 2016-2022 Altova GmbH

```
        public DateTime LastUpdated;
        public List<Artist> CreateTestArtists()
        {
            // TODO add implementation
        }
    }

}
```

6.2.6 Beispiel: Generieren von Java-Code

In diesem Beispiel wird gezeigt, wie Sie ein neues UModel-Projekt erstellen und Programmcode anhand dieses Projekts generieren (ein Prozess, der als "Forward Engineering" bezeichnet wird). Der Einfachheit halber besteht das Projekt aus nur einer Klasse. Außerdem lernen Sie, wie Sie das Projekt für die Codegenerierung vorbereiten und sicher stellen, dass im Projekt die richtige Syntax verwendet wird. Nachdem Sie Programmcode generiert haben, werden Sie diesen außerhalb von UModel ändern, indem Sie eine neue Methode zur Klasse hinzufügen. Zum Abschluss werden Sie die Codeänderungen wieder im ursprünglichen UModel-Projekt zusammenführen (ein als "Reverse Engineering" bekannter Prozess).

In diesem Tutorial wird als Codegenerierungssprache Java verwendet. Die Vorgangsweise ist aber auch bei anderen Codegenerierungssprachen ähnlich.

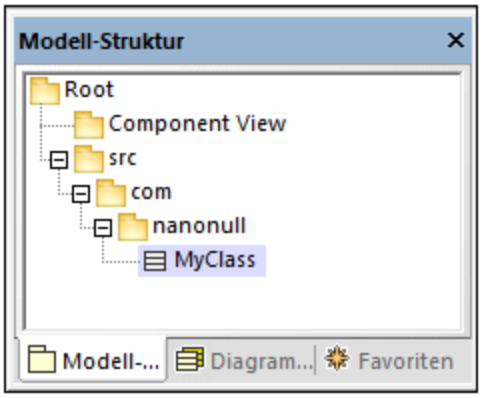
Erstellen eines neue UModel-Projekts

Sie können folgendermaßen ein neues UModel-Projekt erstellen:

- Klicken Sie im Menü **Datei** auf **Neu**. (Oder drücken Sie alternativ dazu **Strg+N** oder klicken Sie auf die Symbolleisten-Schaltfläche "Neu" .)

Das Projekt enthält zu diesem Zeitpunkt nur die Standardpakete "Root" und "Component View". Diese beiden Pakete können nicht gelöscht oder umbenannt werden. "Root" bildet die oberste Hierarchieebene für alle anderen Pakete und Elemente im Projekt. "Component View" wird für das Code Engineering benötigt; normalerweise enthält es eine oder mehrere UML-Komponenten, die von den Klassen oder Schnittstellen Ihres Projekts realisiert werden; wir haben zu diesem Zeitpunkt jedoch noch keine Klassen erstellt. Erstellen wir also zuerst die Struktur unseres Programms. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf das Paket "Root" und wählen Sie im Kontextmenü den Befehl **Neues Element | Paket**. Benennen Sie das neue Paket in "src" um.
2. Klicken Sie mit der rechten Maustaste auf "src" und wählen Sie im Kontextmenü den Befehl **Neues Element | Paket.** Benennen Sie das neue Paket in "com" um.
3. Klicken Sie mit der rechten Maustaste auf "com" wählen Sie im Kontextmenü den Befehl **Neues Element | Paket**. Benennen Sie das neue Paket in "nanonull" um.
4. Klicken Sie mit der rechten Maustaste auf "nanonull" wählen Sie im Kontextmenü den Befehl **Neues Element | Klasse**. Benennen Sie die neue Klasse in "MyClass" um.



Vorbereiten des Projekts für die Codegenerierung

Um Code anhand eines UModel-Modells zu generieren, müssen die folgenden Voraussetzungen gegeben sein:

- Es muss ein C#- oder VB.NET-Namespace-Root-Pakte definiert sein.
- Es muss eine Komponente geben, die von allen Klassen oder Schnittstellen, für die Code generiert werden muss, realisiert wird.
- Der Komponente muss ein physischer Pfad (Verzeichnis) zugewiesen worden sein. Der Code wird in diesem Verzeichnis generiert.
- Für die Komponente muss die Eigenschaft **für Code Engineering verwenden** aktiviert sein.

Alle diese Anforderungen werden weiter unten ausführlicher beschrieben. Beachten Sie, dass Sie jederzeit überprüfen können, ob das Projekt alle Voraussetzungen für die Codegenerierung erfüllt, indem Sie es validieren:

- Klicken Sie im Menü **Projekt** auf **Projektsyntax überprüfen**. (Drücken Sie alternativ dazu **F11**.)

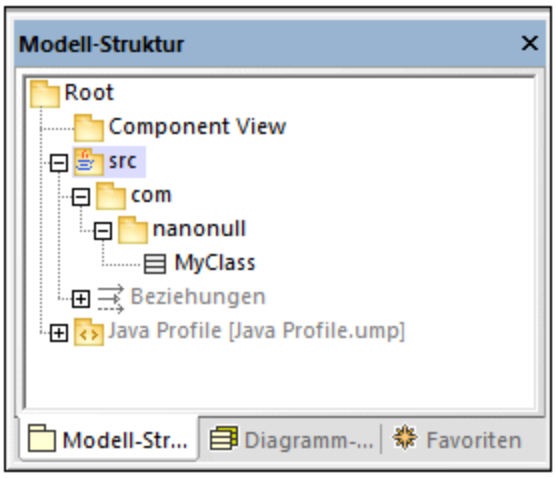
Wenn Sie das Projekt in dieser Phase validieren, wird im Fenster "Meldungen" ein Validierungsfehler angezeigt ("*Es wurde keine Namespace Root gefunden! Verwenden Sie bitte das Kontextmenü in der Modell-Struktur, um ein Paket als Namespace Root zu definieren*"). Um diesen Fehler zu beheben, weisen Sie das Paket "src" als Namespace Root zu:

- Klicken Sie mit der rechten Maustaste auf das Paket "src" und wählen Sie im Kontextmenü den Befehl **Code Engineering | Als Java Namespace Root definieren**.
- Wenn Sie informiert werden, dass das UModel Java-Profil inkludiert wird, klicken Sie auf **OK**.



© 2016-2022 Altova GmbH

Beachten Sie, dass sich das Paketsymbol nun in das Symbol geändert hat, was bedeutet, dass es sich beim Paket um eine Java Namespace Root handelt. Zusätzlich dazu wurde ein Java-Profil zum Projekt hinzugefügt.



Der eigentliche Namespace kann folgendermaßen definiert werden:

1. Wählen Sie das Paket "com" im Fenster **Modell-Struktur** aus.
2. Aktivieren Sie im Fenster **Eigenschaften** die Eigenschaft **<<namespace>>**.



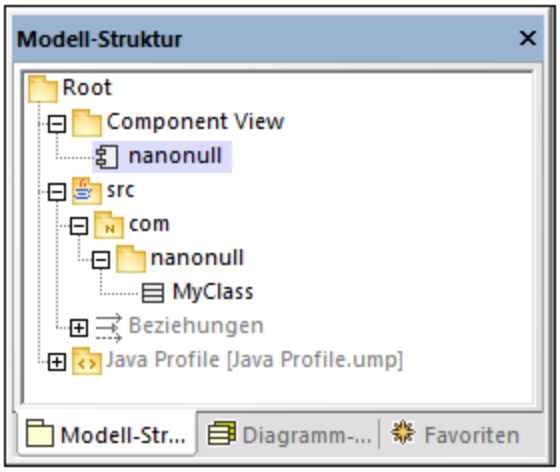
3. Wiederholen Sie die obigen Schritte für das Paket "nanonull".

Beachten Sie, dass sich das Symbol der beiden Pakete "com" und "nanonull" nun in geändert hat, was bedeutet, dass es sich nun um einen Namespace handelt.

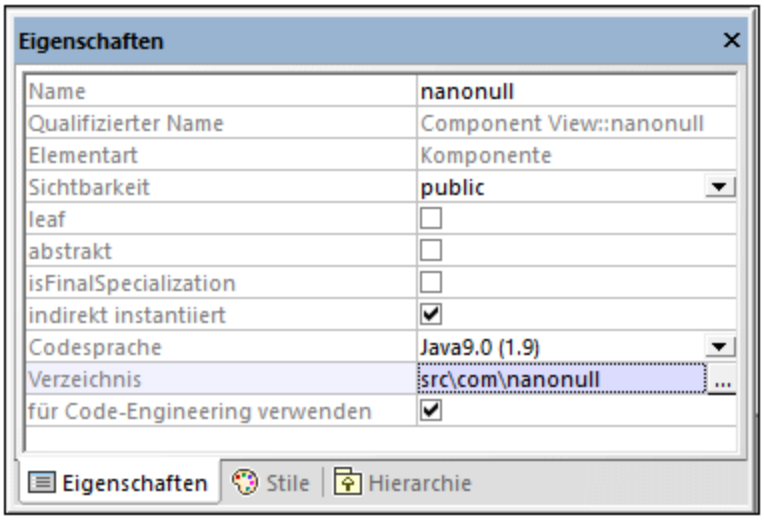
Eine weitere Voraussetzung für die Codegenerierung ist, dass eine Komponente von mindestens einer Klasse oder Schnittstelle realisiert werden muss. Eine Komponente ist in UML ein Teilstück des Systems. In UModel können Sie über die Komponente das Verzeichnis für die Codegenerierung und andere Einstellungen definieren. Andernfalls wäre die Codegenerierung nicht möglich. Wenn Sie das Projekt zu diesem Zeitpunkt validieren, wird im Fenster **Meldungen** eine Warnung angezeigt: "*MyClass hat keine Komponentenrealisierung zu einer Komponente - es wird kein Code generiert*". Um diesen Fehler zu beheben, muss eine Komponente zum Projekt hinzugefügt werden. Gehen Sie folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf "Component View" und wählen Sie im Kontextmenü den Befehl **Neues Element | Komponente**.

2. Benennen Sie die neue Komponente in "nanonull" um.



3. Ändern Sie im Fenster **Eigenschaften** die Eigenschaft **Verzeichnis** in ein Verzeichnis, in dem der Code generiert werden soll (in diesem Beispiel, "src\com\nanonull"). Beachten Sie dass die Eigenschaft **für Code Engineering verwenden** aktiviert ist, was eine weitere Vorbedingung für die Codegenerierung ist.



4. Speichern Sie das UModel-Projekt in einem Verzeichnis und geben Sie ihm einen beschreibenden Namen (in diesem Beispiel **C:\UModelDemo\Tutorial.ump**).

Anmerkung: Der Pfad für die Codegenerierung kann absolut oder relativ zum .ump-Projekt sein. Wenn er, wie in diesem Beispiel, relativ ist, würden mit dem Pfad **src\com\nanonull** alle Verzeichnisse im selben Verzeichnis, in dem auch das UModel-Projekt gespeichert ist, erstellt werden.

Wir haben uns absichtlich entschieden, Code in einem Verzeichnis zu generieren, das den Namespace-Namen enthält; andernfalls würde es zu Warnungen kommen. UModel zeigt standardmäßig Projektvalidierungswarnungen an, wenn die Komponente so konfiguriert ist, das Java-Code in einem Verzeichnis generiert wird, das nicht denselben Namen wie der Namespace hat. Die Komponente "nanonull" in diesem Beispiel hat den Pfad "C:\UModelDemo\src\com\nanonull", sodass es zu keinen Validierungswarnungen kommt. Wenn Sie eine ähnliche Überprüfung für C# oder VB.NET implementieren möchten oder wenn Sie die Namespace-Validierung für Java deaktivieren möchten, gehen Sie folgendermaßen

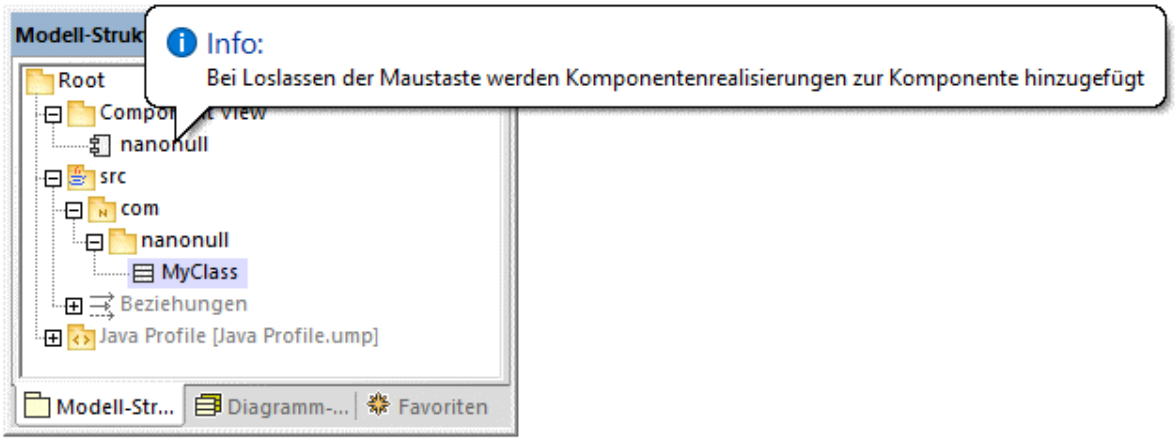
 © 2016-2022 Altova GmbH

vor:

1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Code Engineering**.
3. Aktivieren Sie unter **Namespace für Codedateipfad verwenden** das entsprechende Kontrollkästchen.

Die Komponentenrealisierungsbeziehung kann folgendermaßen erstellt werden:

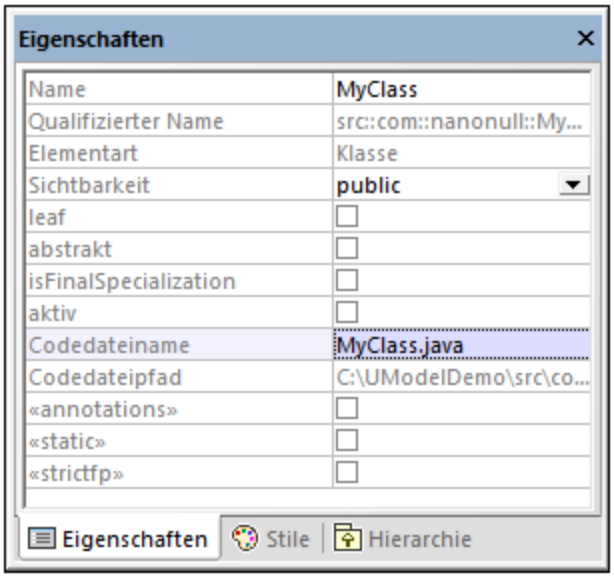
- Ziehen Sie die Maus im Fenster Modell-Struktur bei gedrückter Maustaste von der zuvor erstellten Klasse `MyClass` auf die Komponente `nanonull`.



Die Komponente wird nun von der einzigen Klasse des Projekts, nämlich `MyClass` realisiert. Beachten Sie, dass die oben beschriebene Methode nun eine von mehreren Möglichkeiten ist, die Komponentenrealisierung zu erstellen. Eine weitere Methode ist, sie, wie im Tutorialabschnitt Komponentendiagramme 54 beschrieben, von einem Komponentendiagramm aus zu erstellen.

Als nächstes wird empfohlen, den an der Codegenerierung beteiligten Klassen oder Schnittstellen einen Dateinamen zu geben. Andernfalls generiert UModel die entsprechende Datei mit einem Standarddateinamen und im Fenster **Meldungen** wird eine Warnung angezeigt ("*'Codedateiname wurde nicht definiert - es wird ein Standardname generiert.*"). Um diese Warnung zu entfernen, gehen Sie folgendermaßen vor:

1. Wählen Sie die Klasse `MyClass` im Fenster **Modell-Struktur** aus.
2. Ändern Sie die Eigenschaft **Codedateiname** im Fenster **Eigenschaften** in den gewünschten Datennamen (in diesem Beispiel, `MyClass.java`).

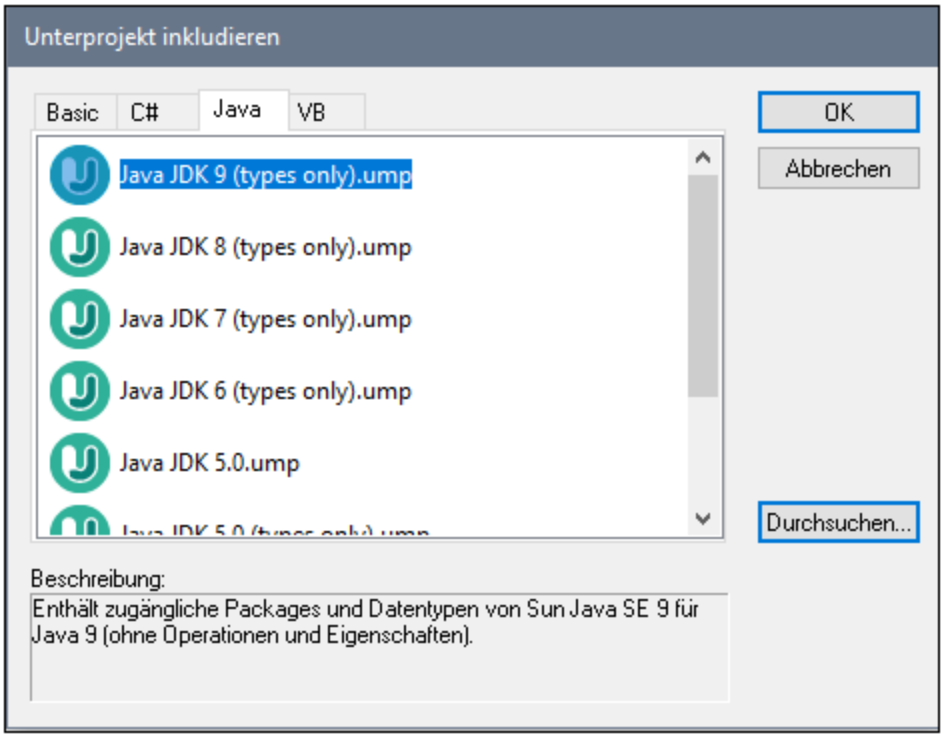


Inkludieren der JDK-Typen

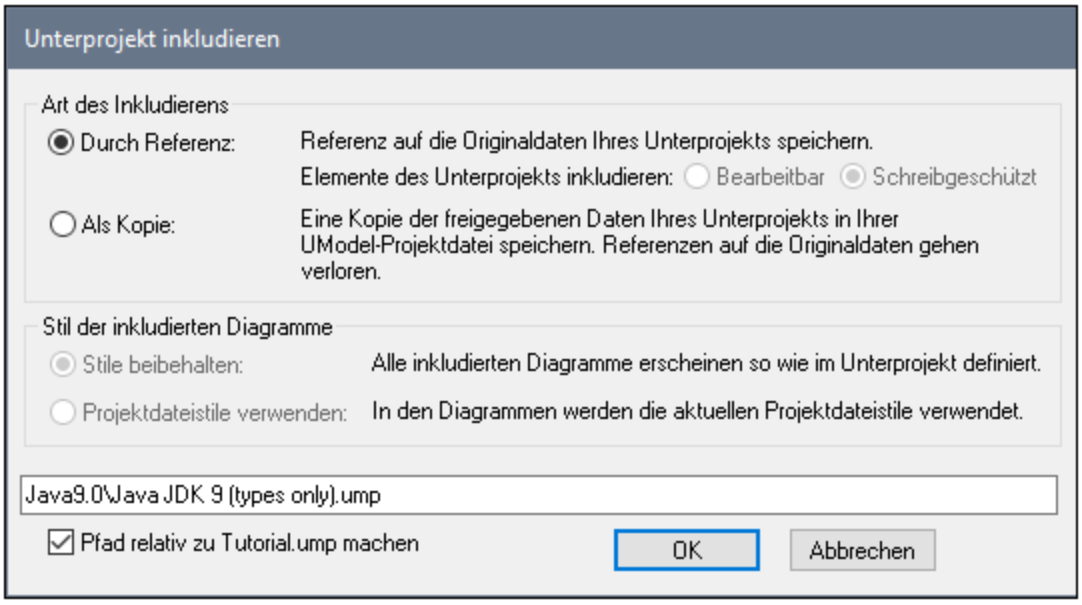
Dieser Schritt ist zwar optional, dennoch wird empfohlen, die Java Development Kit (JDK)-Sprachentypen als Unterprojekt Ihres aktuellen UModel-Projekts zu inkludieren. Andernfalls stehen die JDK-Typen beim Erstellen von Klassen oder Schnittstellen nicht zur Verfügung. Gehen Sie dazu folgendermaßen vor (die Vorgangsweise ist für C#, C++ und VB.NET ähnlich):

1. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.
2. Klicken Sie auf das Register **Java** und wählen Sie das Projekt **Java JDK 9 (types only)** aus.

© 2016-2022 Altova GmbH



3. Wenn Sie gefragt werden, ob das Projekt über eine Referenz oder als Kopie inkludiert werden soll, wählen Sie **Durch Referenz**.

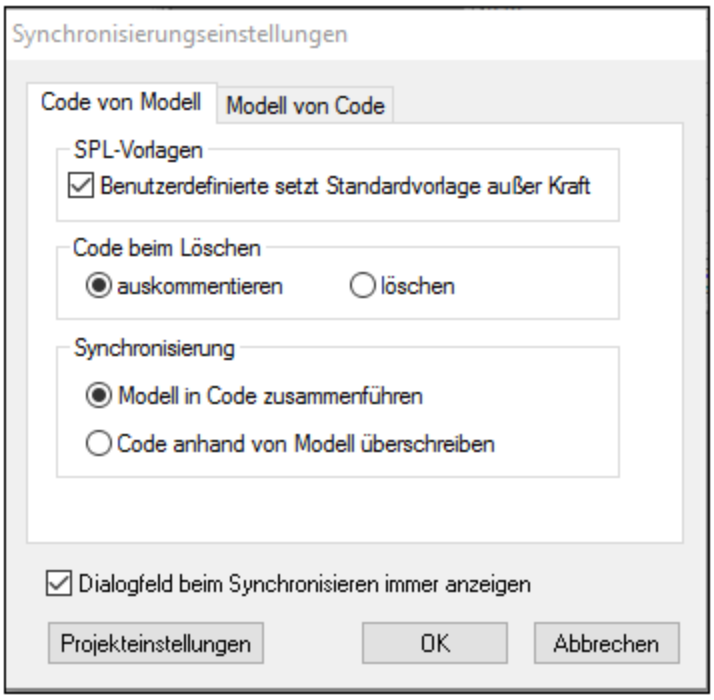


Generieren von Code

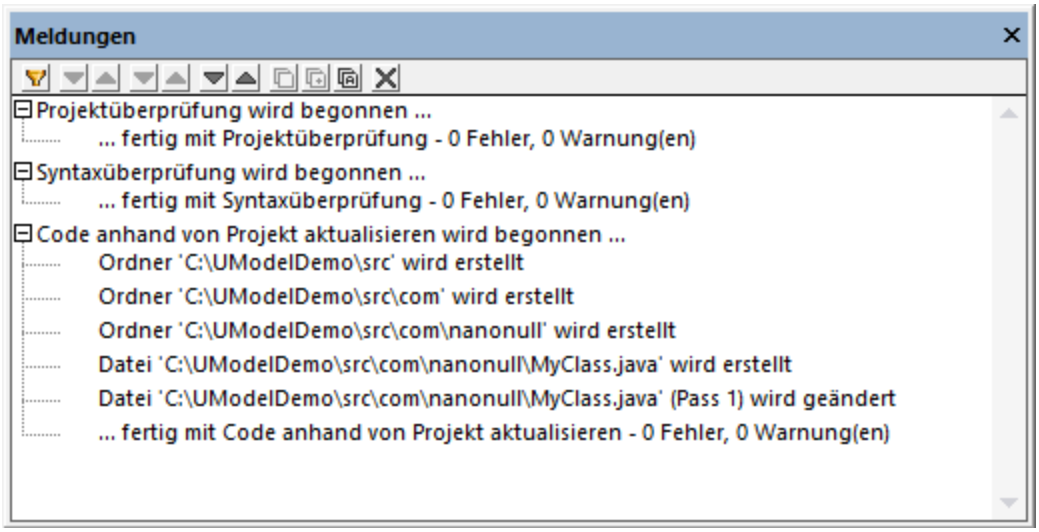
Nachdem nun alle Vorbedingungen erfüllt werden, kann Code folgendermaßen generiert werden:

1. Klicken Sie im Menü **Projekt** auf **Merge Programmcode aus UModel-Projekt**. (Drücken Sie

alternativ dazu **F12**). Beachten Sie, dass dieser Befehl den Namen **Überschreibe Programmcode aus UModel-Projekt** hat, wenn zuvor im unten gezeigten Dialogfeld "Synchronisierungseinstellungen" die Option **Code anhand von Modell überschreiben** aktiviert wurde.



2. Lassen Sie die Standardsynchronisierungseinstellungen unverändert und klicken Sie auf **OK**. Es wird automatisch eine Projektsyntaxüberprüfung durchgeführt, deren Ergebnis im Fenster **Meldungen** angezeigt wird:



Ändern von Code außerhalb von UModel

Die Generierung von Programmcode ist erst der erste Schritt bei der Entwicklung Ihrer Software-Applikation oder Ihres Systems. In einem realen Szenario würde der Code mehrmals verändert, bevor das ein Programm

© 2016-2022 Altova GmbH

mit allen seinen Features fertig entwickelt ist. Öffnen Sie die generierte Datei **MyClass.java** zu diesem Zweck in einem Text-Editor und fügen Sie eine neue Methode zur Klasse hinzu, wie unten gezeigt. Die Datei **MyClass.java** sollte nun folgendermaßen aussehen:

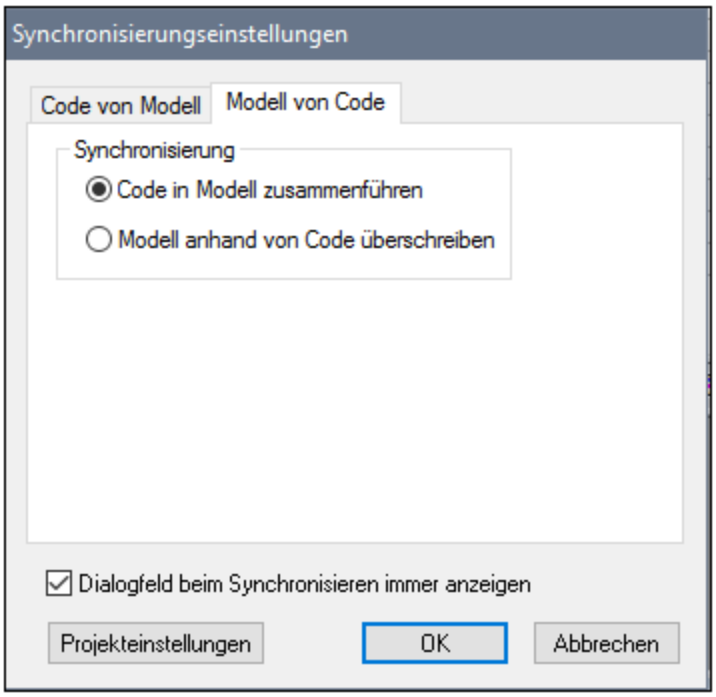
```
package com.nanonull;
public class MyClass{
   public float sum(float num1, float num2){
    return num1 + num2;
   }
}
```

MyClass.java

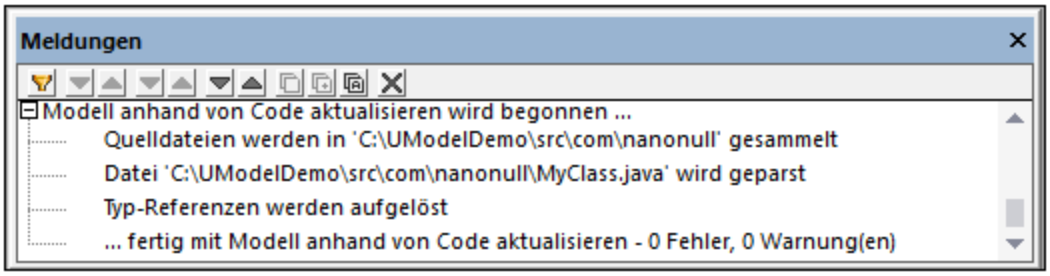
Zusammenführen der Codeänderungen im Modell

Sie können den geänderten Code nun wieder im Modell zusammenführen. Gehen Sie folgendermaßen vor:

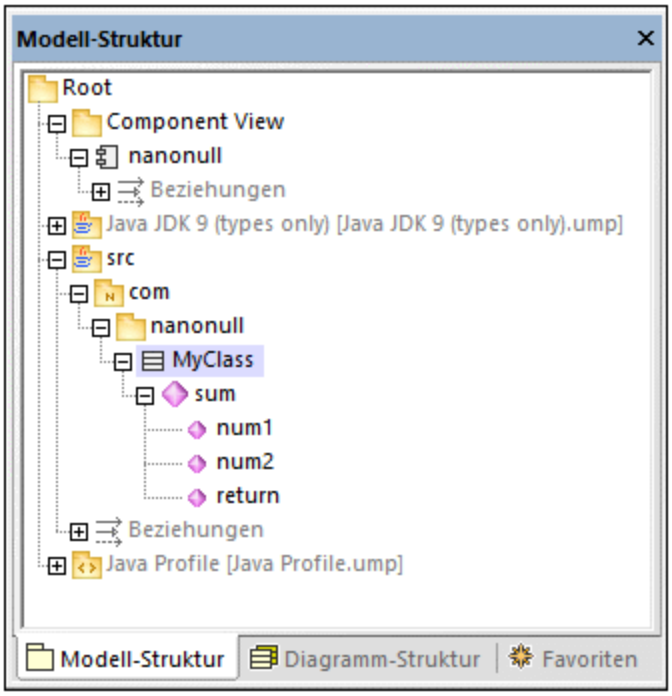
1. Klicken Sie im Menü **Projekt** auf **Merge UModel-Projekt aus Programmcode** (Drücken Sie alternativ dazu **Ctrl + F12**).



2. Lassen Sie die Standardsynchronisierungseinstellungen unverändert und klicken Sie auf **OK**. Es wird automatisch eine Codesyntaxüberprüfung durchgeführt, deren Ergebnis im Fenster **Meldungen** angezeigt wird:



Die Operation `sum` (die mit Reverse Engineering anhand des Codes generiert wurde) wird nun im Fenster "Modell-Struktur" angezeigt.



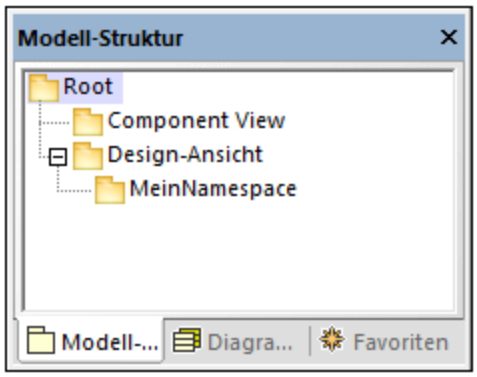
6.2.7 Beispiel: Generieren von C++-Code

In diesem Beispiel wird gezeigt, wie Sie mit UModel C++-Code generieren. Zuerst werden wir ein einfaches UModel-Projekt erstellen, es für die Codegenerierung konfigurieren und anschließend den eigentlichen Code generieren.

Erstellen eines neue UModel-Projekts und seiner Struktur

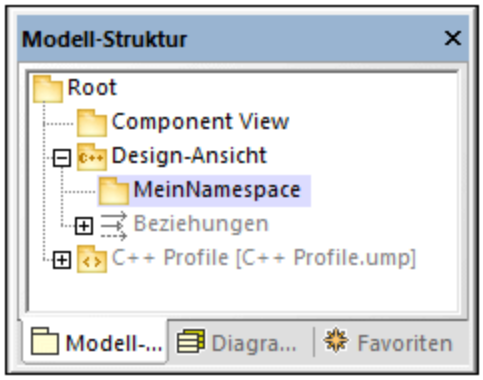
Klicken Sie im Menü **Datei** auf **Neu**. Daraufhin wird ein leeres Projekt mit zwei Standardpaketen ("Root" und "Component View") erstellt. Klicken Sie als nächstes auf das "Root"-Paket und erstellen Sie einige weitere Pakete, wie unten gezeigt. (Wenn Sie mit der grafischen Benutzeroberfläche von UModel noch nicht vertraut sind, lesen Sie zuerst die Kapitel UModel Tutorial 18 und Anleitung zur Modellierung von... 111.)

 © 2016-2022 Altova GmbH



In diesem Beispiel dient das Paket "Design-Ansicht" als Container für alles, was das Design Ihres Modells betrifft (z.B. Klassen und Klassendiagramme), während das Paket "MeinNamespace" als Namespace für alle Klassen, die erstellt werden sollen, verwendet wird. Die Paketstruktur ist jedoch nicht vorgeschrieben, Sie können Ihre Pakete bei Bedarf auch anders strukturieren.

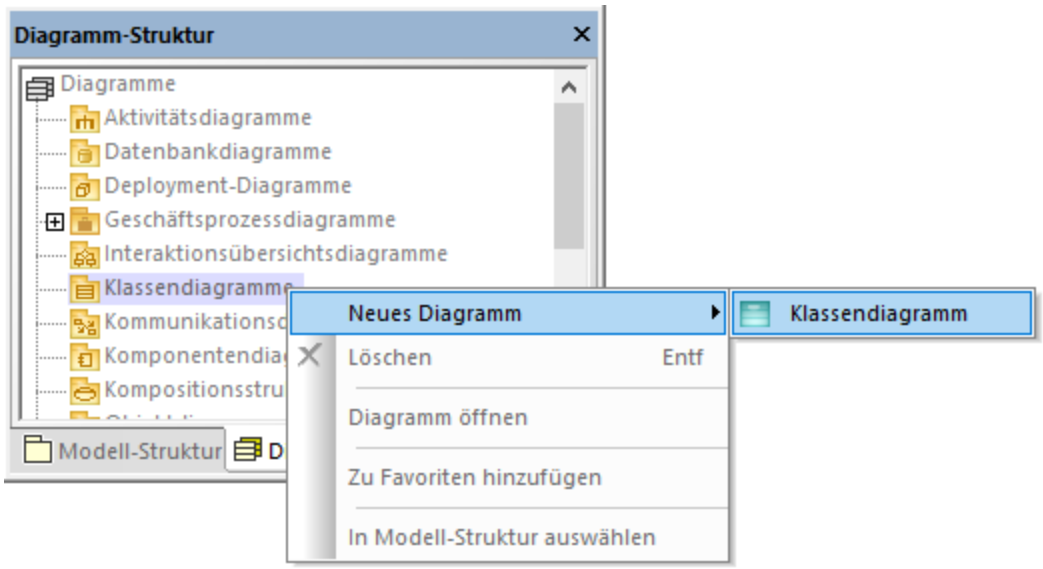
Klicken Sie mit der rechten Maustaste auf das Paket "Design-Ansicht" und wählen Sie im Kontextmenü den Befehl **Code Engineering | Als C++ Namespace Root definieren**. Wenn Sie gefragt werden, ob das C++-Profil auf das Paket angewendet werden soll, klicken Sie zur Bestätigung auf **OK**. Das in UModel vordefinierte C++-Profil wird nun in das Projekt inkludiert.



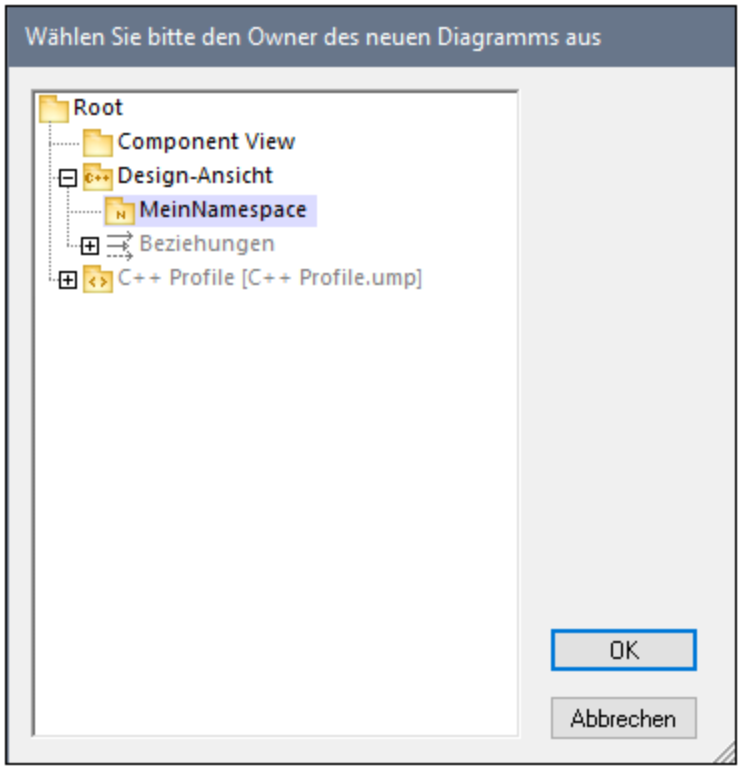
Klicken Sie als nächstes auf das Paket "MeinNamespace" und aktivieren Sie im Fenster "Eigenschaften" das Kontrollkästchen `<<namespace>>`. Dadurch wird das "namespace"-Stereotyp auf das Paket angewendet und sein Symbol ändert sich in . Sie können jetzt unter diesem Namespace Klassen erstellen.

Erstellen von C++-Klassen

Sie können Klassen entweder direkt über das Fenster "Modell-Struktur" oder über ein Klassendiagramm erstellen. In diesem Beispiel werden wir über das Fenster "Diagramm-Struktur" ein Klassendiagramm erstellen, wie unten gezeigt.

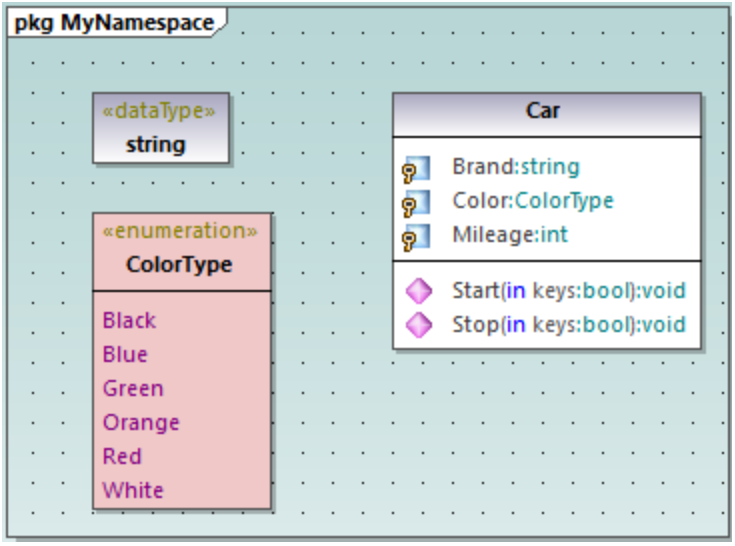


In diesem Beispiel wird vorausgesetzt, dass alle Ihre Klassen unter dem Namespace "MeinNamespace" generiert werden. Wenn Sie daher aufgefordert werden, einen Owner für das Diagramm auszuwählen, wählen Sie das Paket "MeinNamespace" (wie unten gezeigt) aus. Wenn Sie ein anderes Paket auswählen, gehören alle Elemente, die Sie zum Diagramm hinzufügen zum selben Paket wie das Diagramm (was nicht notwendigerweise beabsichtigt ist).



 © 2016-2022 Altova GmbH

Erstellen Sie als nächstes die Klassen, Typen und andere in Ihrem Modell benötigte Elemente, z.B. ein einfaches Diagramm, in dem eine Klasse namens `Car` dargestellt wird:



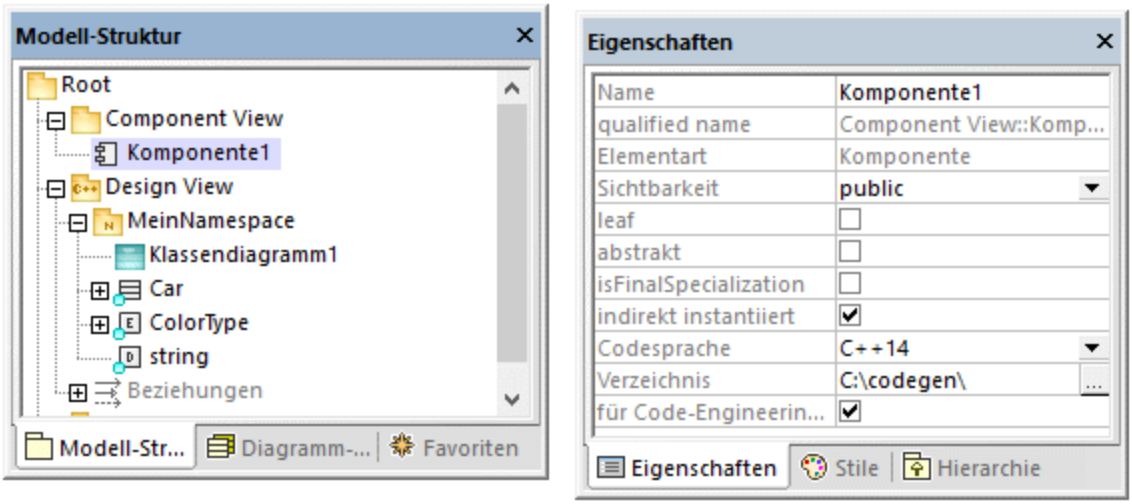
Beachten Sie im Diagramm oben die Enumeration `ColorType` und den Datentyp `string`. Diese Typen sind keine C++-Grundtypen, daher sind sie im C+-Profil, das in UModel vordefiniert ist, nicht enthalten. Aus diesem Grund müssen sie im Modell explizit mit Hilfe der Symbolleisten-Schaltflächen **Enumeration** bzw. **Datentyp** erstellt werden. Im Gegensatz dazu stehen Grundtypen (wie `int` oder `bool`) automatisch während der Eingabe zur Auswahl, siehe auch Typ-Autokomplettierung in Klassen 140. Eine schrittweise Anleitung zum Erstellen von Klassen und ihren Mitgliedern finden Sie in den Kapiteln Klassendiagramme 31 und Anleitung zur Modellierung von... 111.

Konfigurieren des Projektcodes für das Code Engineering

Klicken Sie mit der rechten Maustaste auf das Paket "Component View" und fügen Sie eine neue **Komponente** (d.h. eine Software-Komponente) hinzu. Klicken Sie auf die neue Software-Komponente und definieren Sie im Fenster "Eigenschaften" die folgenden Eigenschaften:

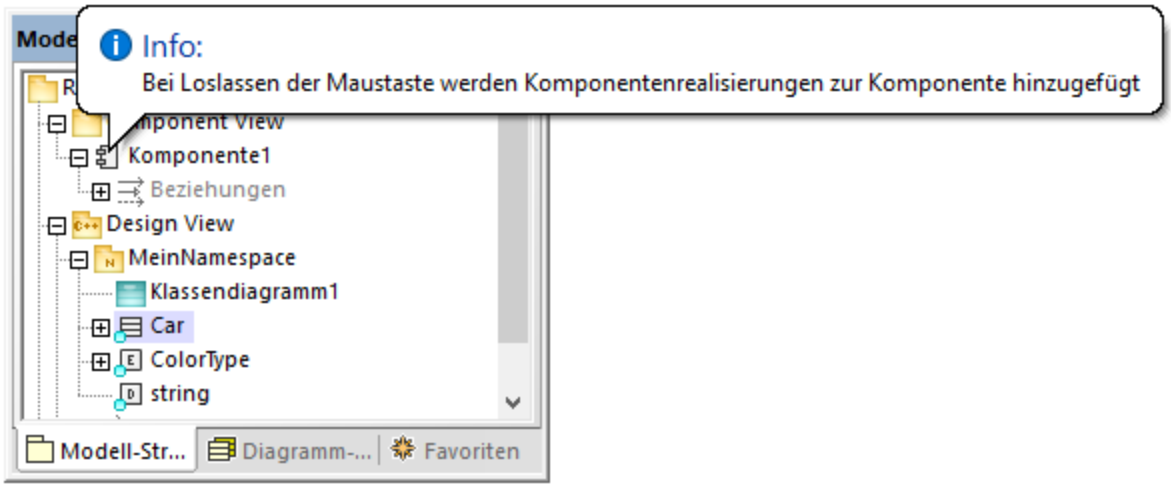
- Codesprache der Komponente (in diesem Beispiel "C++ 14")
- Codegenerierungsverzeichnis (in diesem Beispiel "C:\codegen").

Stellen Sie außerdem sicher, das die Eigenschaft "für Code-Engineering verwenden" auf **True** gesetzt ist.



Erstellen Sie als nächstes zwischen den Klassen, anhand welcher C++-Code generiert werden muss (in diesem Beispiel `Car` und `ColorType`) und der Code Engineering-Komponente eine **Komponentenrealisierungsbeziehung** . Sie können dies entweder von einem Komponentendiagramm 54 aus oder auf die folgende, einfachere Weise tun:

- Klicken Sie im Fenster "Modell-Struktur" auf die Klasse, die von der Komponente realisiert werden soll (in diesem Beispiel `Car` und `ColorType`) und ziehen Sie sie mit der Maus auf die Code Engineering-Komponente (`Komponente1`).



Führen Sie denselben Schritt auch für die Klasse `ColorType` durch.

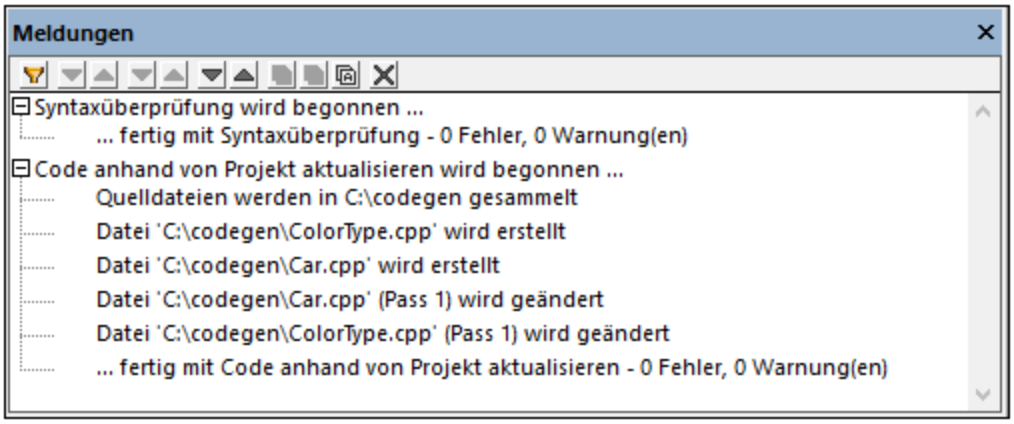
Anmerkung: Falls Sie vergessen haben, für eine Klasse eine **Komponentenrealisierungsbeziehung** zu erstellen, generiert UModel dennoch die entsprechende Codedatei, gibt im Fenster "Meldungen" aber Warnungen aus. Diese Einstellung kann über **Extras | Optionen |** Register **Code Engineering** konfiguriert werden (Name des Kontrollkästchens ist **Fehlende Komponentenrealisierungen generieren**).

© 2016-2022 Altova GmbH

Generieren von C++-Code

Sie können den eigentlichen C++-Code nun generieren. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie im Menü **Projekt** auf **Merge Programmcode aus UModel-Projekt**. (oder drücken Sie alternativ dazu **F12**). Daraufhin wird ein Dialogfeld angezeigt, in dem Sie einstellen können, ob Änderungen im Modell mit denjenigen im Code zusammengeführt oder diese gegebenenfalls überschreiben sollen. Die Codeeinstellungen können für dieses Beispiel unverändert übernommen werden, da der Code hier zum ersten Mal generiert wird. Nähere Informationen dazu finden Sie unter Codesynchronisierungseinstellungen[244].
2. Klicken Sie auf **OK**. Das Ergebnis des Code Engineering wird im Fenster "Meldungen" angezeigt.



6.2.8 SPL-Vorlagen

Bei der Generierung von C++, Java-, C#- oder VB.NET-Code sowie von XSD-Schemas verwendet UModel eine Vorlagensprache namens SPL (Spy Programming Language). Die SPL-Vorlagen geben die Syntax der generierten Codedateien vor. Die SPL-Vorlagen können angepasst werden, z.B. um die Syntax des generierten Codes geringfügig zu modifizieren. Das Bearbeiten von SPL-Vorlagen ist nur bei Codesprachen sinnvoll, die von UModel unterstützt werden. Wenn Sie völlig neue SPL-Vorlagen für andere Sprachen erstellen möchten, wäre es zwar möglich neuen Code zu generieren, doch wäre es nicht möglich, vorhandenen Code zu aktualisieren (da die Sprachsyntax UModel nicht bekannt wäre).

Die SPL-Standardvorlagen befinden sich relativ zum Programminstallationsverzeichnis im Verzeichnis **UModel\SPL**.

> Ändern Sie die vorhandenen SPL-Standardvorlagen nicht, da diese sich direkt auf die Standardcodegenerierung auswirken. Falls Sie die Codegenerierung anpassen müssen, erstellen Sie stattdessen, wie unten beschrieben, benutzerdefinierte Vorlagen.

SPL-Vorlagen werden nur verwendet, wenn neuer Code generiert wird (d.h. wenn neue Klassen, Operationen, usw. zum Modell hinzugefügt wurden und anschließend Code generiert wird). Vorhandener Code wird von SPL-Vorlagen nicht beeinflusst.

Eine Einführung in SPL finden Sie unter SPL-Referenz[1379].

So ändern Sie bereitgestellte SPL-Vorlagen:

1. Gehen Sie im UModel-Installationsverzeichnis ("Programme") zu den bereitgestellten SPL-Vorlagen, z.B.: **...\UModel2023\UModelSPL\Java\Default**.
2. Kopieren Sie die SPL-Dateien, die Sie ändern möchten, in das übergeordnete Verzeichnis. Wenn Sie z.B. das Auftreten der Java-Klasse im generierten Code ändern möchten, kopieren Sie die Datei **Class.spl** aus **...\UModel2023\UModelSPL\Java\Default** in **...\UModel2023\UModelSPL\Java**.
3. Nehmen Sie die Änderungen an der/den .spl-Datei(en) vor und speichern Sie diese.

So verwenden Sie die benutzerdefinierten SPL-Vorlagen:

1. Wählen Sie die Menüoption **Projekt | Synchronisierungseinstellungen**.
2. Aktivieren Sie in der Gruppe "SPL-Vorlagen" das Kontrollkästchen **Benutzerdefinierte setzt Standardvorlage außer Kraft**.

© 2016-2022 Altova GmbH

6.3 Importieren von Quellcode

Vorhandener Java-, C#-, C++- und VB.NET-Programmcode kann in UModel importiert werden (dieser Vorgang wird auch als "Reverse Engineering") bezeichnet. Die folgenden Projekttypen können in UModel importiert werden:

- Java-Projekte (Eclipse-.project-Dateien, NetBeans-project.xml-Dateien und JBuilder-.jpx-Dateien)
- C#- und VB.NET-Projekte (Visual Studio sln, csproj, csdprj..., vbproj, vbp sowie Borland .bdsproj-Projektdateien)
- C++98-, C++11-, C++14, C++17-Projekte (dazu gehören mit Visual Studio 2010, 2012, 2013, 2015, 2017 und 2019 erstellte Visual Studio .vcxproj- und .sln-Projektdateien).

Sie können nicht nur Quellcode aus einem Quellprojekt, sondern auch aus einem Quellverzeichnis importieren. Der Import aus einem Quellverzeichnis funktioniert ähnlich und ist vor allem dann nützlich, wenn in Ihrem Code keiner der oben aufgelisteten Projekttypen verwendet wird. Ein Beispiel zum Importieren eines Quellverzeichnisses finden Sie unter Reverse Engineering (Code zu Modell) 75.

Quellcode kann außerdem entweder in ein neues leeres UModel-Projekt oder in ein vorhandenes UModel-Projekt importiert werden. Sie können beim Import angeben, ob die importierten Elemente diejenigen im Modell selbst (falls vorhanden) überschreiben sollen, oder ob sie in das Modell zusammengeführt werden sollen. Optional können Klassen- und Paketdiagramme beim Importieren von Code automatisch generiert werden.

Der Importassistent enthält eine Reihe von Importoptionen für die einzelnen Plattformen (Java, .NET, C++). Wenn der importierte Java/C#/VB.NET-Code etwa Kommentare enthält, können diese optional in UModel-Dokumentation konvertiert werden. Eine vollständige Liste aller Optionen finden Sie unter Optionen für den Code-Import 212.

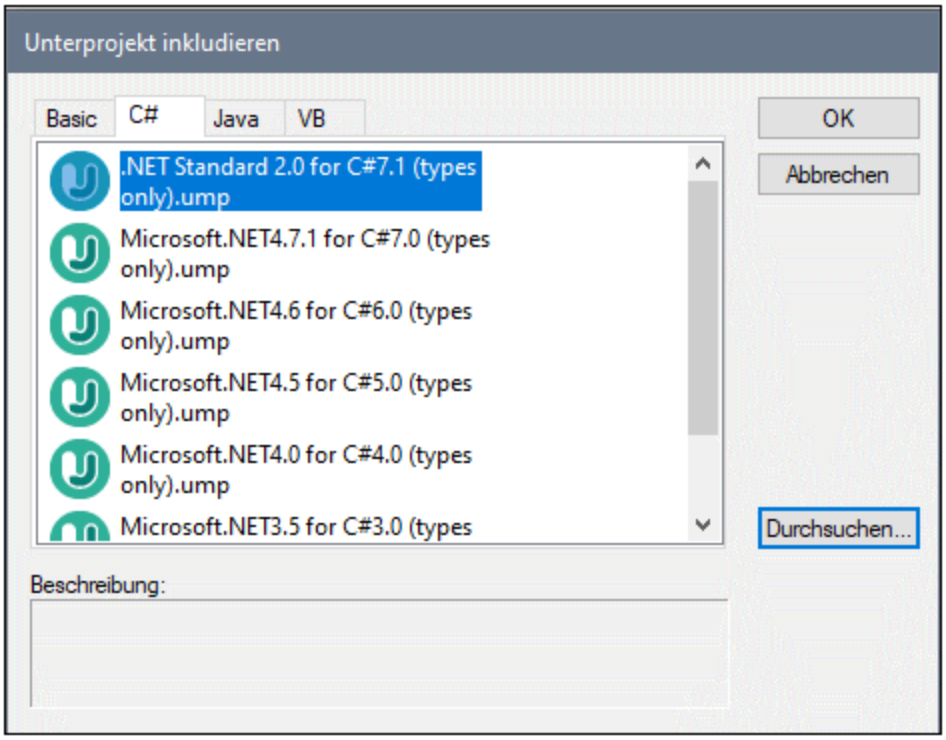
Nachdem Ihr C++, C#-, VB.NET- oder Java-Code in UModel importiert wurde, können Sie das Modell ändern (z.B. durch Hinzufügen neuer Klassen oder Umbenennen von Eigenschaften und Operationen) und es optional wieder mit dem Originalcode rücksynchronisieren, sodass ein vollständiges Round-Trip-Engineering durchgeführt wird, siehe Synchronisieren von Modell und Quellcode 240. Beachten Sie, dass die Code Engineering-Unterstützung für C++ sich auf das Reverse Engineering (d.h. auf das Importieren von Quellcode in ein Modell) beschränkt. Im Gegensatz zu anderen Sprachen können Änderungen am Modell nicht wieder zurück in C++-Code geschrieben werden.

Voraussetzungen

UModel enthält mehrere vordefinierte Unterprojekte, die speziell für das Code Engineering erstellt wurden und die die Datentypen für die jeweilige unterstützte Sprache und Plattform enthalten. Bevor Sie versuchen, Quellcode in ein UModel-Projekt zu importieren, sollten Sie das vordefinierte UModel-Unterprojekt für die entsprechende Programmiersprache und Plattform inkludieren, siehe Inkludieren von Unterprojekten 174. Andernfalls werden bestimmte Datentypen nicht erkannt und werden nach dem Import in ein separates Paket mit dem Namen "Unbekannte externe Elemente" platziert.

So inkludieren Sie ein Unterprojekt mit den erforderlichen Sprachdatentypen:

1. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.
2. Klicken Sie auf das Register für die entsprechende Quellsprache und Plattform (z.B., Java 8.0, C# 6.0, VB 9.0) und klicken Sie anschließend auf OK.



Beachten Sie die folgenden Punkte:

- Wenn Sie ein Datentyp-Unterprojekt für eine bestimmte Sprache inkludieren, fügt UModel automatisch auch das Profil dieser Sprache zu Ihrem Projekt hinzu. Das Profil-Unterprojekt (.ump) enthält nur die wichtigsten Typen und ist nicht mit dem Datentyp-Unterprojekt (auch .ump), welches ausführlichere Typdefinitionen enthält, identisch.
- Wenn Sie den Import ohne Inkludierung eines Datentyp-Unterprojekts durchführen, wird der Import dennoch durchgeführt und UModel inkludiert auch das Profil dieser Sprache in das Projekt. Alle unbekannten Typen werden jedoch in das Paket "Unbekannte externe Elemente" platziert. Um dies zu vermeiden, sollten Sie sicherstellen, dass das Datentyp-Unterprojekt für die erforderliche Sprache und Plattform, wie oben erklärt, inkludiert wird.
- Für C++ gibt es kein Unterprojekt mit allen möglichen C++-Datentypen aus der Standard Template Library (STL). Es gibt stattdessen ein C++-Sprachprofil mit grundlegenden Typen. Sie können dieses Unterprojekt entweder, wie oben gezeigt, manuell hinzufügen oder es wird beim Importieren von C++-Code oder wenn Sie mit der rechten Maustaste auf ein Paket klicken und im Kontextmenü den Befehl **Code Engineering | Als C++ Namespace Root definieren** auswählen, automatisch zum Projekt hinzugefügt.

Importieren von Quellcode aus einem Projekt

1. Klicken Sie im Menü **Projekt** auf **Quellprojekt importieren**. (Wenn Sie alternativ dazu Code aus einem bestehenden Verzeichnis importieren möchten, wählen Sie den Befehl **Quellverzeichnis importieren**.)
2. Wählen Sie die Sprachversion des Quellprojekts (z.B. Java 8.0, C# 6.0 oder C++14).
3. Klicken Sie auf **Durchsuchen** ... und wählen Sie die Quellprojektdatei aus.
4. Definieren Sie die erforderlichen Importoptionen oder ändern Sie diese, siehe auch Optionen für den Code-Import 212 (Beachten Sie, dass diese Optionen von der in Schritt 2 ausgewählten Sprache abhängig sind).

 © 2016-2022 Altova GmbH

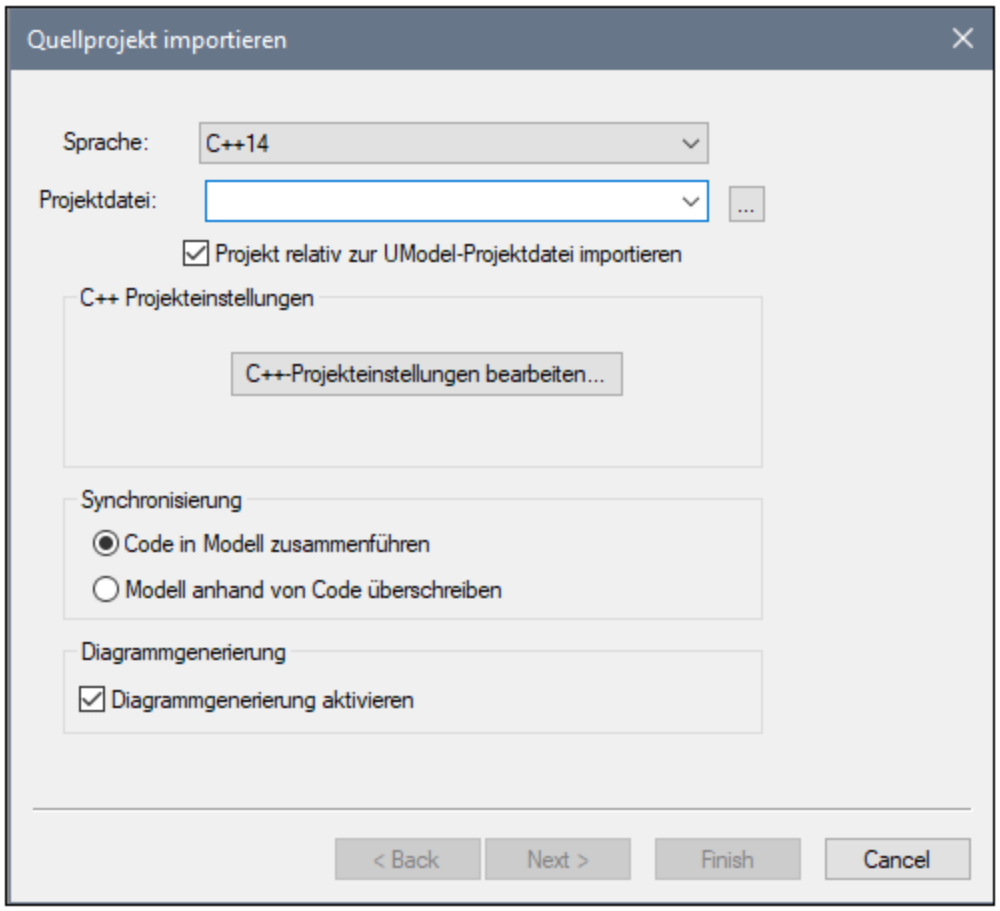
5. Klicken Sie auf **Fertig stellen**, um den Assistenten zu beenden.

Ein Schritt-für-Schritt-Beispiel dazu finden Sie unter Beispiel: Importieren eines C#-Projekts 218.

6.3.1 Reverse Engineering von C++ Code

C++-Projekte sind in punkto Reverse Engineering sehr groß im Vergleich zu Java-, C#- oder VB.NET-Projekten. Im Allgemeinen wird empfohlen, die Reverse Engineering-Funktion nur für kleine bis mittelgroße C++-Projekte zu verwenden. Bei großen C++-Projekten würde der Import sehr lange dauern (z.B. 15 Minuten und länger).

Verwenden Sie den Menübefehl **Projekt | Quellprojekt importieren**, um C++-Projekte in UModel zu importieren.



Um in einer anderen IDE als Visual Studio erstellte C++-Projekte zu importieren, verwenden Sie den Menübefehl **Projekt | Quellverzeichnis importieren** anstelle von **Projekt | Quellprojekt importieren**. Für solche Projekte müssen Sie über das Importdialogfeld Angaben bzgl. Vorverarbeitung machen sowie Pfade und Compiler-Einstellungen definieren, siehe Optionen für den Code-Import 212.

Die zu inkludierenden Verzeichnisse, die vom Parser durchsucht werden sollen, können entweder auf Projektebene über die Optionen für den Code-Import 212 oder global definiert werden. Um zu inkludierende Verzeichnisse global hinzuzufügen, setzen Sie die Umgebungsvariable `UMODEL_CPP_INCLUDE` auf eine Liste von

Verzeichnissen, die durch ";" getrennt werden. So können Sie z.B. den Include-Pfad "C:\example\include" folgendermaßen hinzufügen:

1. Rufen Sie die Systemsteuerung auf und geben Sie in das Suchfeld die ersten Buchstaben von "Umgebungsvariablen" ein.
2. Klicken Sie auf **Systemumgebungsvariablen bearbeiten.**
3. Klicken Sie auf **Umgebungsvariablen**.
4. Klicken Sie auf **Neu** und fügen Sie eine neue Variable mit dem Namen `UMODEL_CPP_INCLUDE` und dem Wert `C:\example\include` hinzu.
5. Klicken Sie auf **OK,** um alle Dialogfelder zu schließen.
6. Starten Sie UModel neu.

Bei mit Visual Studio erstellten C++-Projekten werden die Angaben zur Vorverarbeitung und Inkludierung von Pfaden automatisch anhand der .vcproj-Dateien eruiert. Die Kompatibilität mit dem Microsoft Visual C++ Compiler wird von Visual Studio 6.0 bis Visual Studio 2019 unterstützt (Beachten Sie, dass sich diese Kompatibilität auf den in den .cpp-Quelldateien verwendeten Codedialekt bezieht; Ihr Visual Studio-Projekt muss mit Visual Studio 2010, 2012, 2013, 2015, 2017 oder 2019 gespeichert worden sein, damit es importiert werden kann).

Beachten Sie die folgenden Punkte:

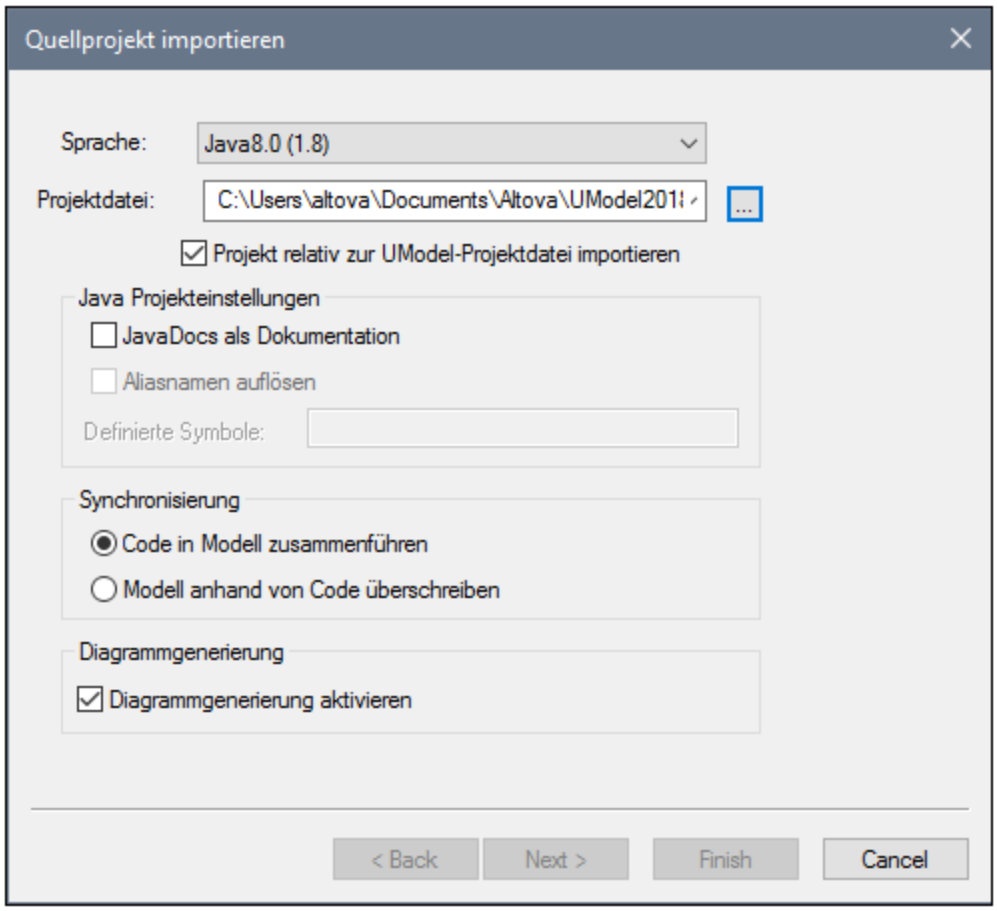
- Wenn UModel beim Import einen unbekannten Datentyp findet, wird im Fenster "Meldungen" eine Warnung angezeigt und der Typ wird im Modell als `int` angezeigt. Im Gegensatz dazu werden unbekannte Typen bei C# oder Java in das Paket "Unbekannte externe Elemente" importiert.
- Wenn Sie C++-Code im UModel importieren, wird automatisch ein vordefiniertes UModel-Profil für C++ zum Projekt hinzugefügt. Das Profil enthält die grundlegenden C++Datentypen und -Stereotype, die für das Code Engineering benötigt werden und ähnelt Profilen für andere Sprachen.
- Die Unterstützung von C++-Attributen ist eingeschränkt. Es werden nur vordefinierte Standardattribute wie `[[noreturn]]`, `[[carries_dependency]]`, `[[deprecated]]` erkannt. Benutzerdefinierte Attribute werden ignoriert.

Nachdem der C++-Code in UModel importiert wurde, können Sie über das Modell Änderungen daran vornehmen und diese Änderungen anschließend wieder in den Code überführen (Round-Trip Engineering). Wie bei anderen Code Engineering-Sprachen bleibt die ursprüngliche Quellcodeimplementierung (z.B. Methodenkörper) nach dem Round-Trip Engineering unverändert. Datentypen oder Member-Namen, die Sie im Modell geändert haben (z.B. umbenannte Klassen) werden jedoch im Code übernommen. Nähere Informationen dazu finden Sie unter Beispiel: Generieren von C++-Code 202 und Synchronisieren von Modell und Quellcode 240.

6.3.2 Optionen für den Code-Import

Wenn Sie Programmcode in ein UModel-Projekt importieren, müssen Sie die unten aufgelisteten Optionen eventuell definieren oder ändern. Diese Optionen stehen in dem Dialogfeld, das bei Aufruf des Menübefehls **Projekt | Quellprojekt importieren** oder **Projekt | Quellverzeichnis importieren** angezeigt wird, zur Verfügung.

 © 2016-2022 Altova GmbH



Dialogfeld "Quellprojekt importieren"

Die meisten der Optionen im Dialogfeld können auch später jederzeit geändert werden, siehe Codesynchronisierungseinstellungen[244].

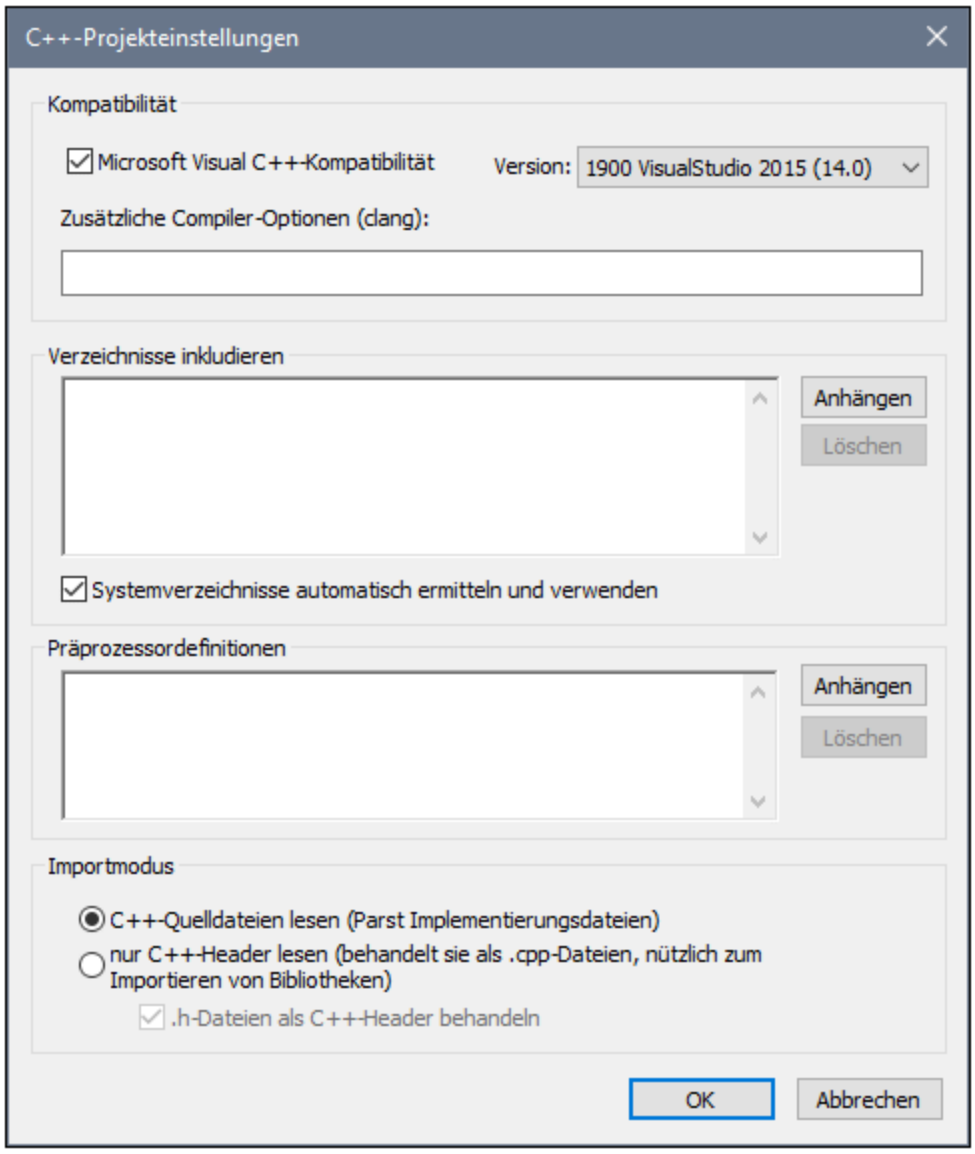
Die folgenden Optionen gelten unabhängig von Sprache oder Plattform für alle Projekttypen:

Option	Beschreibung
Projekt relativ zur UModel-Projektdatei importieren	Standardmäßig ist diese Option aktiviert, d.h. zwischen dem UModel-Projekt und dem importierten Quellcodeprojekt wird ein relativer Pfad definiert. Nach Import des Quellcodes wird im UModel-Projekt automatisch eine UML-Komponente generiert (sie steht in der Modell-Struktur als Child von "Component View" zur Verfügung). Diese Komponente realisiert die zu erzeugenden Schnittstellen oder Klassen; Sie definiert auch die Optionen für das Code Engineering, einschließlich Pfad zum Quellprojekt oder -verzeichnis. Wenn **Projekt relativ zur UModel-Projektdatei importieren** aktiviert ist, handelt es sich hierbei um einen relativen Pfad, andernfalls um einen absoluten.

Option	Beschreibung
Code in Modell zusammenführen / Modell anhand von Code überschreiben	Wenn **Code ...zusammenführen** ausgewählt ist, werden potenzielle Namenskonflikte (wie z.B. Paket- oder Klassennamen) durch Anhängen einer Nummer an das zu importierende Element gelöst. Wenn **...überschreiben** ausgewählt ist und es Namenskonflikte gibt, hat das importierte Element Vorrang vor einem im Projekt bereits vorhandenen (und überschreibt dieses).
Diagrammgenerierung aktivieren	Aktivieren Sie dieses Kontrollkästchen optional, wenn Sie anhand der importierten Klassen Klassen- und Paketdiagramme generieren möchten. Wenn dieses Kontrollkästchen aktiviert ist, inkludiert der Importassistent zusätzliche Schritte, über die sie das Aussehen der generierten Diagramme anpassen können.

Die folgenden Optionen gelten nur für C++-Projekte:

© 2016-2022 Altova GmbH



Dialogfeld für C++-Projekteinstellungen

Option	Beschreibung
Microsoft Visual C++-Kompatibilität	Diese Option ist nur beim Import von mit Visual Studio kompiliertem C++-Code anwendbar. Sie können damit die Microsoft Visual C++ Compiler-Kompatibilität definieren. Wählen Sie hier die von Ihrem Visual Studio C++-Projekt verwendete Compiler-Version (Codedialekt) aus. Beachten Sie, dass sich diese Einstellung auf den Codedialekt der Quellcodedateien bezieht. Das Visual Studio-Projekt (oder die Projektmappe) selbst muss mit Visual Studio-Versionen ab Visual Studio 2010 gespeichert werden, damit Code importiert werden kann. Um Quellcode zu importieren, der mit einer andere IDE als Visual Studio erstellt wurde, verwenden Sie den Befehl **Projekt | Quellverzeichnis importieren**.

Option	Beschreibung
Zusätzliche Compiler-Optionen (clang)	UModel verwendet intern zum Lesen von C++-Code die `clang` Compiler-Version 3.8. In diesem Textfeld können, falls nötig (wenn auf UModel anwendbar), zusätzliche Code Parsing-Optionen definiert werden, siehe auch `clang`-Dokumentation (http://releases.llvm.org/3.8.1/tools/docs/UsersManual.html#command-line-options).
Verzeichnisse inkludieren	Mit Hilfe dieser Option können Sie zusätzliche Verzeichnisse definieren, in denen UModel beim Reverse Engineering von C++-Code nach C++-Klassen suchen soll. Das Definieren der zu inkludierenden Verzeichnisse ist optional, wenn es sich um ein Visual Studio-Projekt handelt. Wenn Sie das Kontrollkästchen **Systemverzeichnisse automatisch ermitteln und verwenden** aktiviert haben, versucht UModel, zusätzlich zu den in diesem Dialogfeld explizit angeführten Verzeichnissen, alle systemweit definierten zu inkludierenden Verzeichnisse automatisch zu ermitteln. Die Systemverzeichnispfade können auch über die Systemumgebungsvariable `UMODEL_CPP_INCLUDE` definiert werden, siehe Reverse engineering von C++-Projekten 211. In diesem Fall ersetzen die in der Systemumgebungsvariablen `UMODEL_CPP_INCLUDE` definierten Verzeichnisse diejenigen, die sonst inkludiert würden, wenn das Kontrollkästchen **Systemverzeichnisse automatisch ermitteln und verwenden** aktiviert wäre.
Präprozessordefinitionen	Mit Hilfe dieser Option können Sie etwaige für die Codekompilierung benötigte C++-Vorverarbeitungsanweisungen definieren. Wenn es sich beim Quellprojekt um ein Visual Studio-Projekt handelt, werden die Vorverarbeitungsanweisungen automatisch ermittelt.
Importmodus	Mit der Option **C++-Quelldateien lesen** werden alle Dateien des Quellprojekts geparst. Dies ist die Standardoption. Wenn Sie nur C++-Bibliotheken importieren möchten, aktivieren Sie die Option **nur C++-Header lesen**, was auch den Import beschleunigt. Standardmäßig werden .h-Dateien als C++-Header behandelt. Deaktivieren Sie das Kontrollkästchen **.h-Dateien als C++-Header behandeln,** wenn im Quellprojekt eine andere Dateierweiterung für Header-Dateien verwendet wird.

Die folgenden Optionen gelten nur für C#- und VB.NET-Projekte:

Option	Beschreibung
DocComments als Dokumentation	Aktivieren Sie dieses Kontrollkästchen, um im C#-Code gefundene Kommentare in UModel-Elementdokumentation zu konvertieren (siehe auch Dokumentation 97).

© 2016-2022 Altova GmbH

Option	Beschreibung

Aliasnamen auflösen	Dieses Kontrollkästchen ist standardmäßig aktiviert. Wenn Ihr C#- oder VB.NET-Code Namespaces oder Klassenaliasse wie im Codefragment unten enthält, wird empfohlen, dieses Kontrollkästchen aktiviert zu lassen. Andernfalls werden Assoziationen und Abhängigkeiten im Zusammenhang mit Klassen und Namespaces in Ihrem Code beim Import von UModel eventuell nicht automatisch erkannt (und wären daher im Modell auch nicht vorhanden).
	
	`using Q = System.Collections.Generic.Queue<String>;`
	`Q myQueue;`
	
	Beispiel für ein Alias in C#-Code
	
	Potenziell Konflikte verursachende Aliasnamen werden beim Import zum Paket "Unbekannte externe Elemente" des UModel-Projekts hinzugefügt, wenn ihre Verwendung nicht klar ist.
	
	Wenn Sie den Code anhand des Modells wieder aktualisieren (Round-Trip Engineering), werden die Aliasse, so wie sie im generierten Code vorhanden sind, beibehalten.
	
	Die Option **Alialsnamen auflösen** kann später jederzeit wieder geändert werden, siehe Codesynchronisierungseinstellungen 244. Wenn Sie diese Option nach (nicht aber vor) dem Import aktivieren, fordert Sie UModel auf, das Projekt wieder anhand des Codes zu aktualisieren, da die Option sich auch auf das Forward Engineering auswirkt.

Definierte Symbole	Wenn Ihr C#- oder VB.NET-Code Symbole enthält, die über Vorverarbeitungsanweisungen wie `#if`, `#endif` definiert sind, können Sie UModel anweisen, diese beim Reverse Engineering von Code zu berücksichtigen.
	
	`#if DEBUG`
	`static void DisplayMessage()`
	`{`
	`Console.WriteLine("Please wait...");`
	`}`
	`#endif`
	
	Beispiel für ein Symbol für bedingte Kompilierung in C#-Code
	
	Wenn Sie z.B. am obigen Code ein Reverse Engineering durchführen, wird die Methode `DisplayMessage()` nur dann in das Modell importiert, wenn Sie das `DEBUG`-Symbol definiert haben.
	
	Um Symbole für bedingte Kompilierung zu definieren, geben Sie diese, getrennt durch ein Semikolon, in das Textfeld "Definierte Symbole" ein.
	
	Beim Reverse Engineering werden alle im Quellcode verwendeten Symbole von UModel im Fenster "Meldungen" ausgegeben.

Die folgenden Optionen gelten nur für Java-Projekte:

Option	Beschreibung
JavaDocs als Dokumentation	Aktivieren Sie dieses Kontrollkästchen, um im Code gefundene Kommentare im JavaDocs-Stil in UModel-Elementdokumentation zu konvertieren (siehe auch Dokumentation[97]). **Anmerkung:** Nur Kommentare zu Java-Klassen, Schnittstellen, Operationen und Eigenschaften werden konvertiert.

6.3.3 Beispiel: Importieren eines C#-Projekts

In diesem Beispiel wird gezeigt, wie Sie eine mit Visual Studio erstellte C#-Beispiellösung in UModel importieren. Die Lösung steht als .zip-Archiv unter dem folgenden Pfad zur Verfügung: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial\Anagram_CSharp.zip**. Sie müssen die Lösung vor dem Import nicht mit Visual Studio kompilieren, das Archiv **Anagram_CSharp.zip** jedoch in einen Ordner ihrer Wahl entpacken, bevor Sie mit den unten beschriebenen Schritten fortfahren.

Ziel in diesem Beispiel ist die Erstellung eines Reverse Engineering der C#-Lösung und die Erzeugung von eines UModel-Projekts anhand dieses Codes. Bei Importieren des Codes wählen wir die Option zum automatischen Generieren von Klassen- und Paketdiagrammen.

Schritt 1: Erstellen eines neuen Projekts

Sie können Quellcode entweder in ein vorhandenes oder in ein neues UModel-Projekt importieren. In diesem Beispiel werden wir Code in ein neues UModel-Projekt importieren.

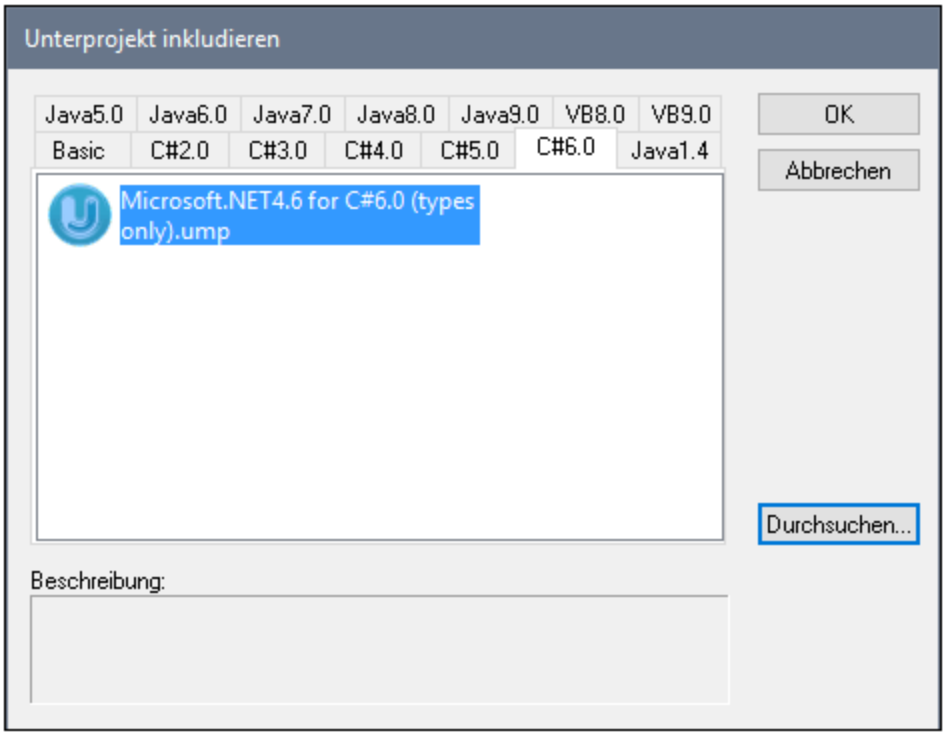
- Wählen Sie im Menü **Datei** den Befehl **Neu** (Drücken Sie alternativ dazu **Strg + N** oder klicken Sie in der Symbolleiste auf die Schaltfläche **Neu**).

Schritt 2: Inkludieren der C#-Sprachtypen

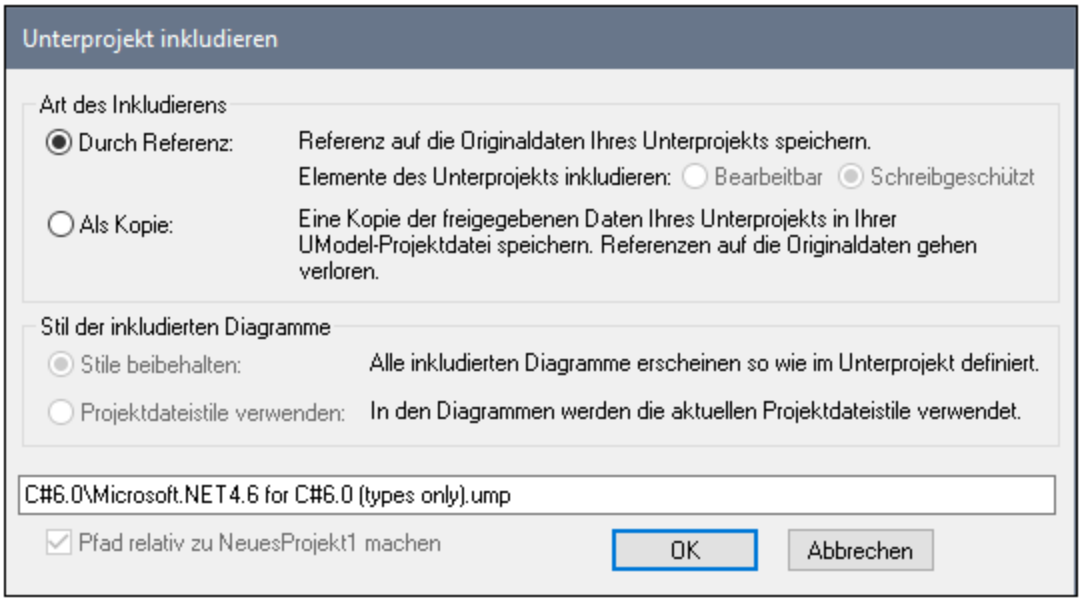
Das Quellprojekt wurde mit Visual Studio 2015 in C# erstellt, daher inkludieren wir ein vordefiniertes UModel-Projekt, das die C# 6.0-Sprachtypen enthält (da die C#-Sprachversion für Visual Studio 2015 Version 6.0 ist). Wahrscheinlich funktionieren auch frühere Versionen von C# mit unserer C#-Beispiellösung.

1. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.
2. Klicken Sie auf das Register **C#**.

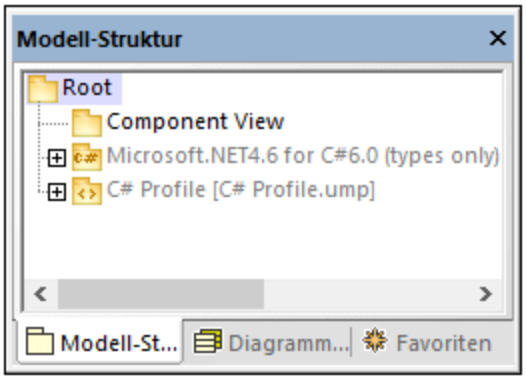
© 2016-2022 Altova GmbH



3. Wählen Sie das Projekt **Microsoft .NET 4.6 for C# 6.0 (types only).ump** aus und klicken Sie auf **OK**.
4. Wenn Sie aufgefordert werden, die Art der Inkludierung auszuwählen (durch Referenz oder als Kopie) belassen Sie die Standardoption unverändert.

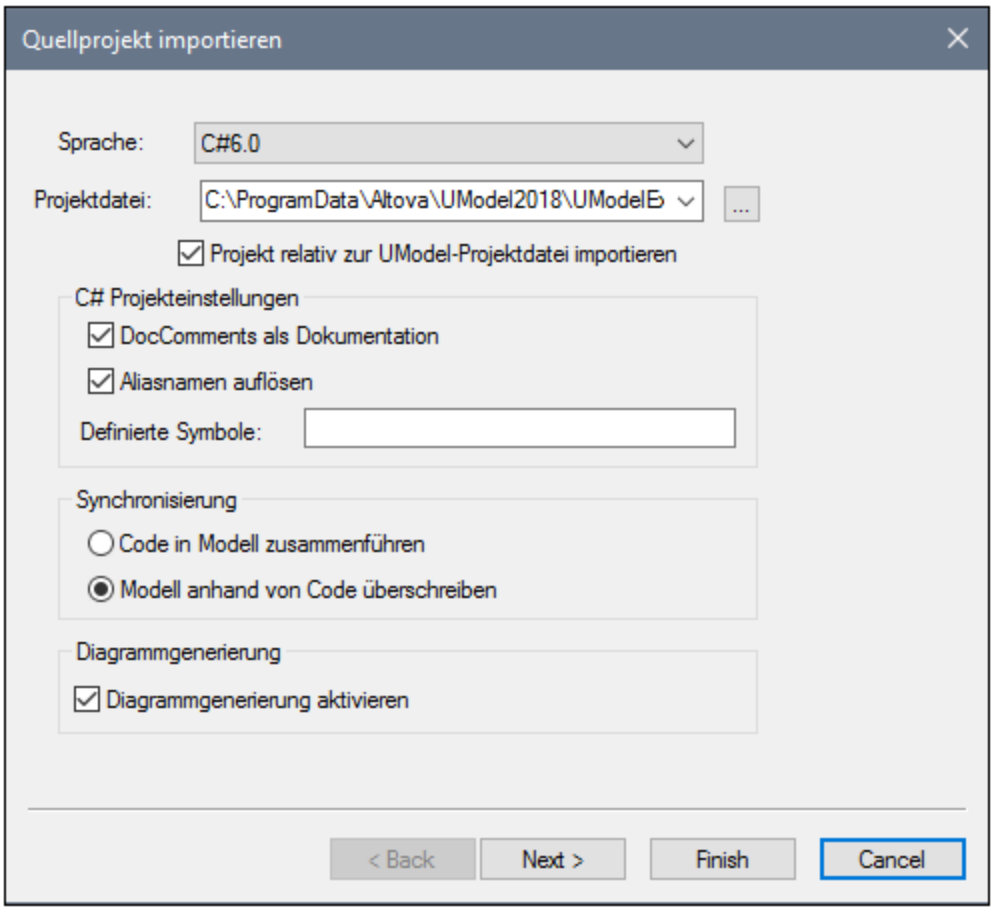


Dadurch werden sowohl die C#-Sprachtypen als auch das C#-Sprachprofil inkludiert und in der Modell-Struktur angezeigt:



Schritt 3: Importieren der C#-Lösung

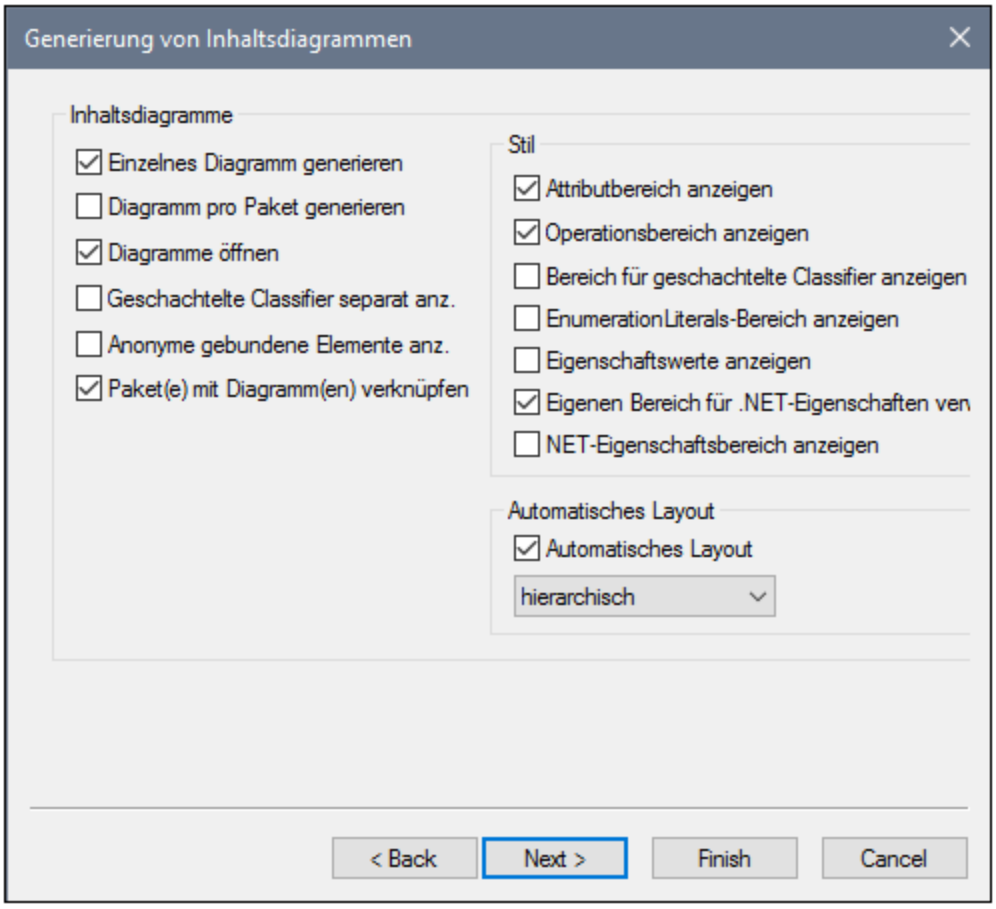
1. Klicken Sie im Menü **Projekt** auf **Quellprojekt importieren**.



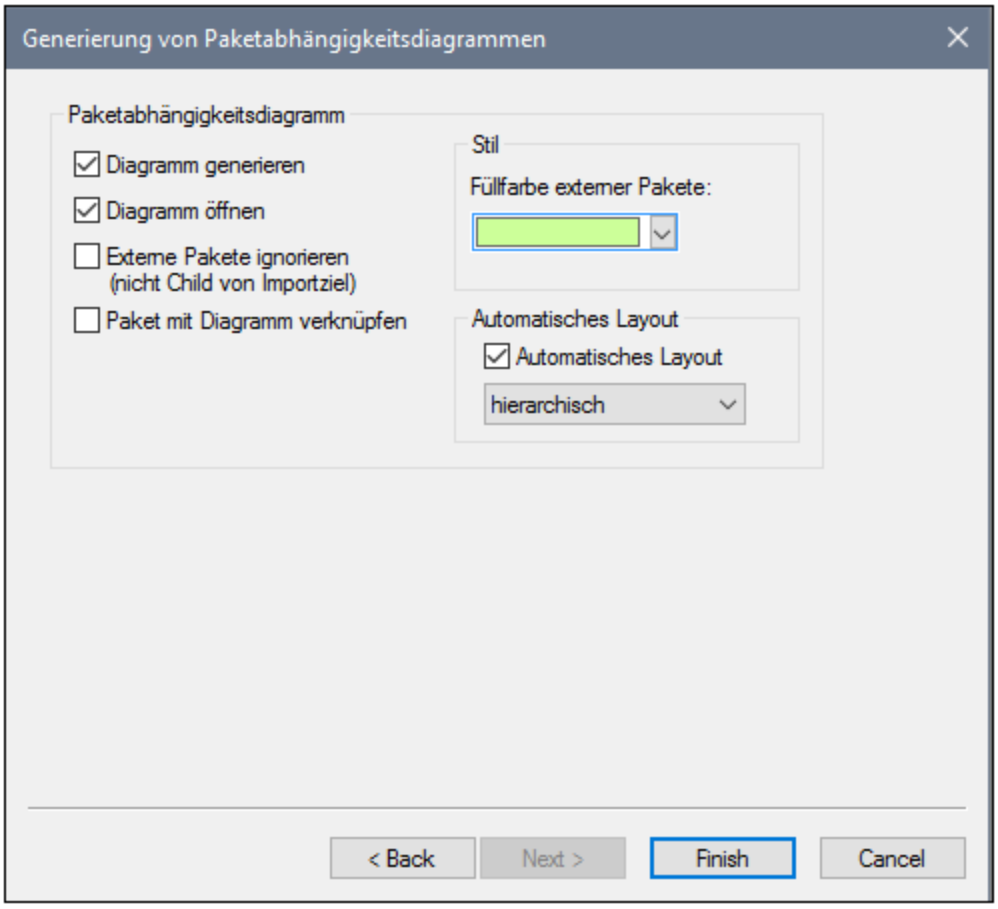
2. Wählen Sie als Sprache **C# 6.0** aus.
3. Klicken Sie neben **Projektdatei** auf **Durchsuchen** ... und navigieren Sie zur .sln-Lösungsdatei.
4. Aktivieren Sie das Kontrollkästchen **DocComments als Dokumentation** (Dadurch werden Codekommentare zu Operationen oder Eigenschaften in das Modell importiert).

© 2016-2022 Altova GmbH

5. Da wir Code in ein neues UModel-Projekt importieren, aktivieren Sie die Option **Modell anhand von Code überschreiben** (die andere Option **Code in Modell zusammenführen** eignet sich für den Import in ein vorhandenes Projekt).
6. Klicken Sie auf **Weiter**.
7. Aktivieren Sie die Diagrammgenerierungsoptionen, wie unten gezeigt, und klicken Sie auf **Weiter**. (Diese Optionen gelten für automatisch beim Codeimport generierte Klassendiagramme.)

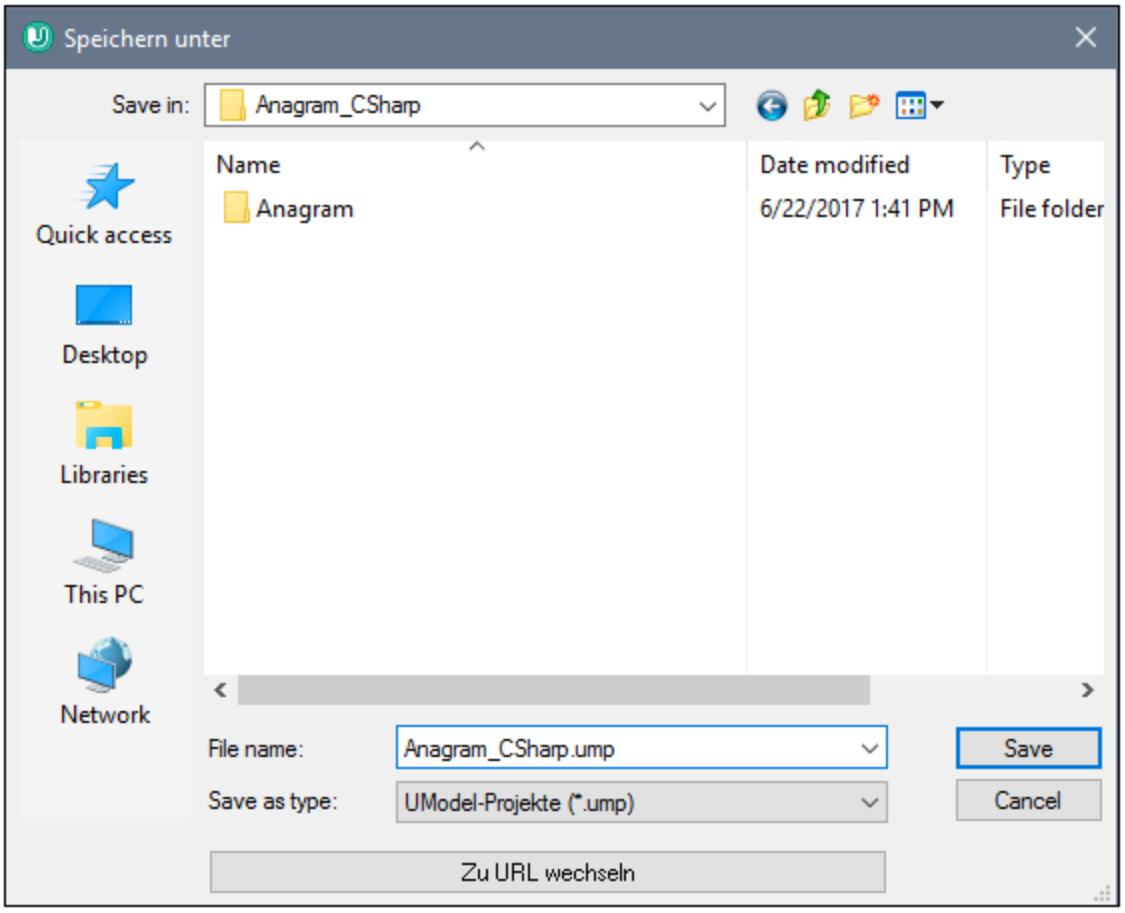


8. Wählen Sie die Diagrammgenerierungsoptionen aus, wie unten gezeigt, und klicken Sie auf **Fertig stellen**. (Diese Optionen gelten für automatisch beim Codeimport generierte Paketdiagramme.)

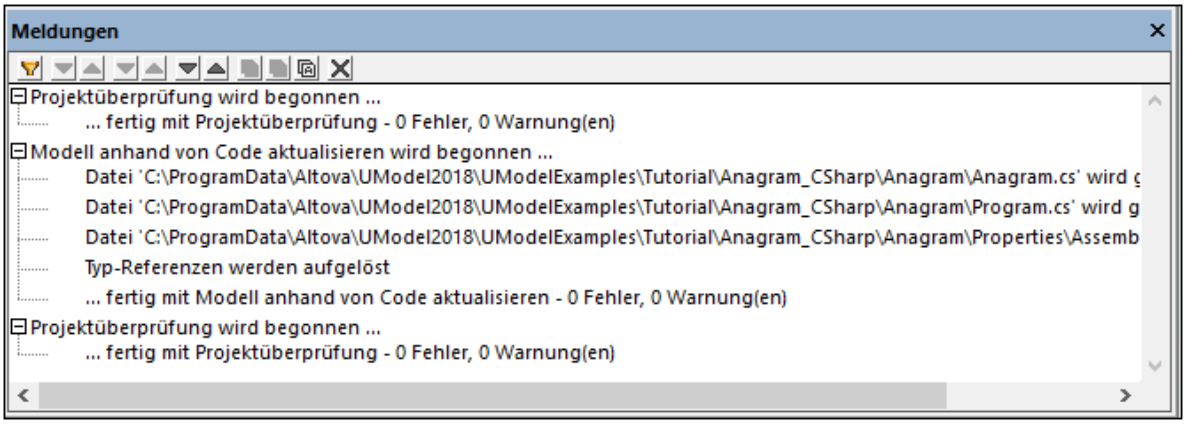


9. Geben Sie einen Namen ein, wählen Sie einen Zielordner für das neue UModel-Projekt aus und klicken Sie auf **Speichern** (Standardmäßig ist der in diesem Dialogfeld angezeigte Ordner derselbe Ordner, wie der aus, dem die Lösung importiert wurde).

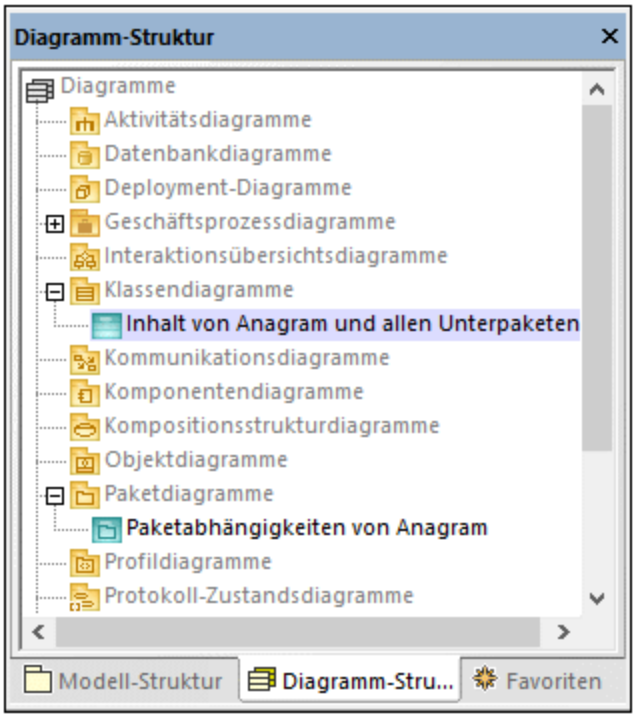
© 2016-2022 Altova GmbH



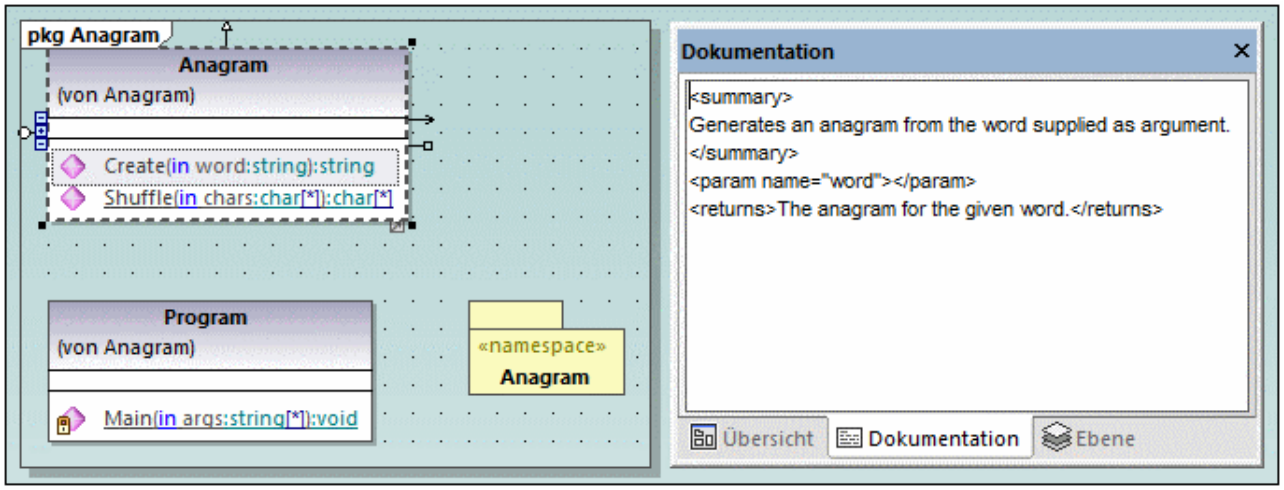
Im Fenster "Meldungen" sehen Sie die Fortschrittsanzeige für das Reverse Engineering.



Nach Abschluss des Codeimports werden alle Diagramme automatisch geöffnet, da diese Option vor der Codegenerierung ausgewählt wurde. Alle generierten Diagramme stehen in der Diagramm-Struktur zur Verfügung:



Da wir die Option zum Generieren von Dokumentation anhand des Quellcodes ausgewählt haben, wird die importierte Dokumentation im Fenster **Dokumentation** angezeigt, z.B. wenn Sie in der Klasse `Anagram` auf die Operation `create` klicken:



Anmerkung: Die Dokumentation wird nur hinzugefügt, wenn beim Import der C#-Lösung die Option **DocComments als Dokumentation** aktiviert war (siehe "Schritt 3: Importieren der C#-Lösung weiter oben).

© 2016-2022 Altova GmbH

6.4 Importieren von Java-, C#- und VB.NET-Binärdateien

UModel unterstützt den Import von C#-, Java- und VB.NET-Binärdateien. Diese Funktion erweist sich vor allem beim Arbeiten mit Binärdateien aus externen Quellen oder, wenn der Quellcode nicht mehr zur Verfügung steht, als besonders nützlich. Beachten Sie dazu Folgendes:

- Um Java-Binärdateien zu importieren, muss eine unterstützte Version[13] von Java Runtime Environment (JRE) oder Development Kit (JDK) installiert sein. Der Import von Typen wird für Java .class-Dateien oder .jar Class Archives, die den Java Virtual Machine Spezifikationen entsprechen, unterstützt. Dazu gehören Java Virtual Machines wie OpenJDK, SapMachine, Liberica JDK und andere, siehe Hinzufügen von benutzerdefinierten Java Runtimes[226].
- Um C#- oder VB.NET-Binärdateien zu importieren, muss je nach Bedarf .NET Framework, .NET Core, .NET 5 oder .NET 6 installiert sein. Die besten Ergebnisse erzielen Sie bei Auswahl der Option **beliebige (Disassembler verwenden)** aus dem Import-Dialogfeld. Alle nicht erkannten Typen werden nach dem Import in das Paket "Unbekannte externe Elemente" platziert. Damit keine (oder weniger") unbekannte externe Elemente importiert werden, wenden Sie *vor dem Import* das entsprechende UModel-Profil für die jeweilige Version Ihrer Code Engineering-Sprache an (z.B. ."NET 5 für C# 9.0"). Siehe auch Anwenden von UModel-Profilen[170].
- Der Import von Binärdateien, die mit Hilfe eines Obfuscators unleserlich gemacht wurden, wird nicht unterstützt.

In der unten stehenden Tabelle finden Sie eine Liste der verfügbaren Methoden für den Import von Binärdateitypen in ein UModel-Projekt.

C#, VB.NET	Java
Import einer Assembly-Datei (.dll, .exe)	Import eines Class File Archive (.jar, .zip)
Import einer Assembly aus dem Global Assembly Cache (GAC)	Import einer Klassendatei (.class) aus einem Paket-Root-Ordner
Import einer Assembly aus Visual Studio .NET-Referenzen	Import von Class Archives aus dem Klassenpfad
	Import von Class Archives aus Java Runtime (Nur für Java-Versionen bis einschließlich Java 8)

Binärdateien können mit dem Menübefehl **Projekt | Binärtypen importieren** importiert werden. Sie können UModel optional Klassen- und Paketdiagramme anhand der importierten Typen generieren lassen. Beispiele dazu finden Sie unter Beispiel: Import von .NET GAC Assemblys[230] und Beispiel: Import von Java .class-Dateien[233].

Zusätzlich dazu können Binärtypen auch über die Befehlszeile (siehe UModel Befehlszeilenschnittstelle[104]) oder programmatisch über die UModel API (siehe Programmatischer Import von Binärtypen)[879] importiert werden.

Wenn Sie Binärdateien in ein UModel-Projekt importieren, können Sie verschiedene Importoptionen, darunter die folgenden, definieren:

- Zusätzlich zu den in der Binärdatei definierten Typen können Sie jeden beliebigen referenzierenden Typ importieren. Außerdem können Sie den Import von referenzierenden Typen auf bestimmte Java-Pakete und .NET-Namespaces einschränken.
- Typmember können beim Import übersprungen werden. So können Sie etwa Klassen und Schnittstellen ohne deren Eigenschaften und Methoden importieren.
- Typen können entsprechend ihren Zugriffsmodifizierern (wie z.B. private oder public) importiert werden. So können Sie z.B. nur öffentliche Klassen importieren und private, geschützte (protected) und interne Klassen überspringen.

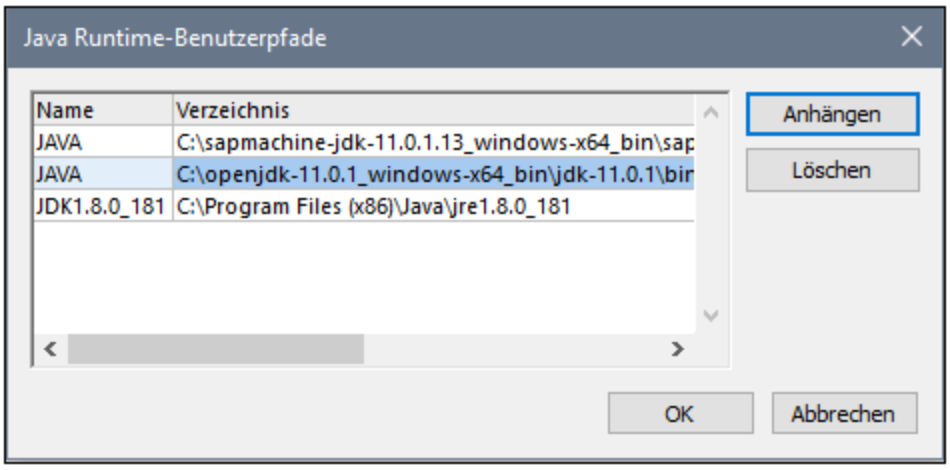
Nähere Informationen zu allen Optionen finden Sie unter Optionen für den Import von Binärdateien 227.

6.4.1 Hinzufügen von benutzerdefinierten Java Runtimes

UModel ermittelt standardmäßig JDKs und JREs, falls diese auf dem lokalen Rechner *installiert* sind. Diese werden folglich beim Start des Binärdateiimportassistenten in der Liste der Java Runtimes angezeigt. Dies ist bei JDKs und JREs von Oracle, die mit einem Installationsprogramm geliefert werden und sich bei der Installation im System selbst registrieren, der Fall. Andere Java Virtual Machine Distributionen hingegen, zu denen es kein Installationsprogramm gibt, müssen manuell zu UModel hinzugefügt werden. Dazu gehören Oracle OpenJDK, SapMachine und andere.

So fügen Sie benutzerdefinierte Java Runtimes zu UModel hinzu:

1. Klicken Sie im Menü **Projekt** auf **Binärtypen importieren**.
2. Wählen Sie als Sprache **Java** aus.
3. Erweitern Sie die **Runtime** Dropdown-Liste und klicken Sie auf **Java Runtime-Benutzerpfade bearbeiten...**.
4. Klicken Sie auf **Anhängen** und navigieren Sie zum Verzeichnis für das jeweilige JDK.



5. Klicken Sie auf **OK**.

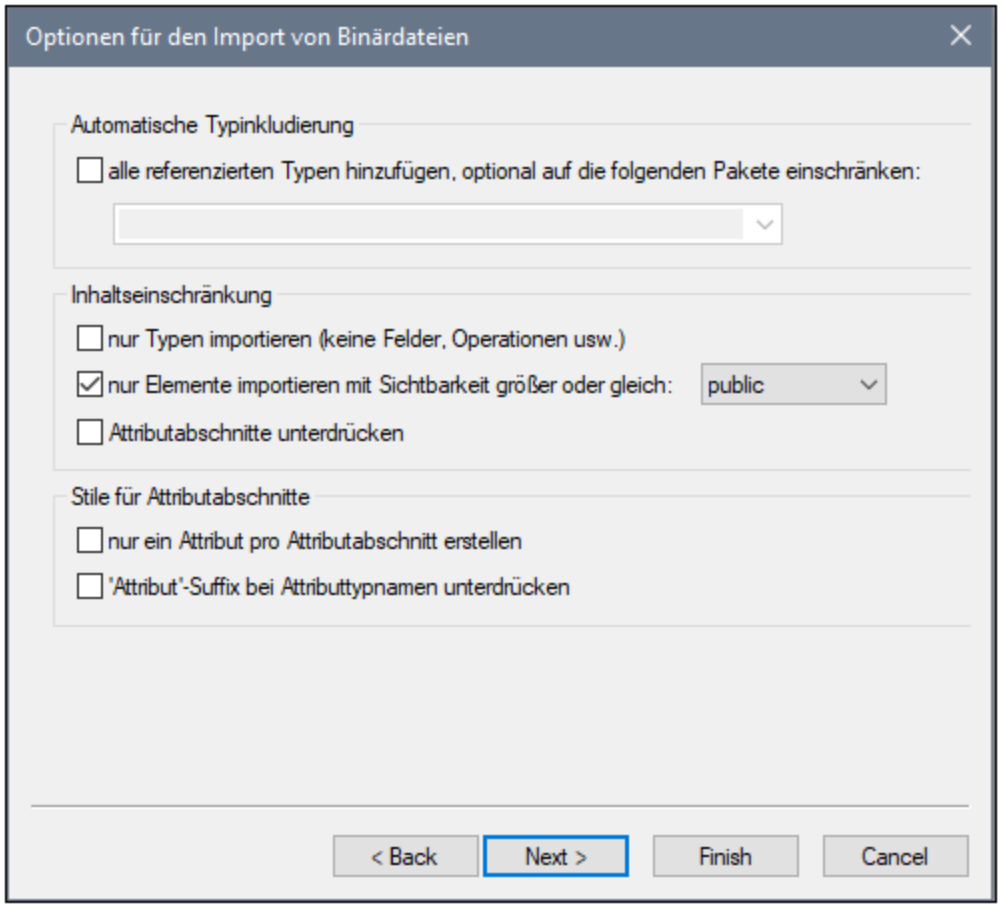
Die ausgewählte Runtime wird nun in der **Runtime**-Liste angezeigt und steht zur Auswahl bereit, wenn Sie Binärdateien für diese Runtime importieren müssen.

© 2016-2022 Altova GmbH

Beachten Sie, dass sich diese Einstellungen nur auf den Import von Binärdateien auswirken. Informationen über das Hinzufügen eines Java Virtual Machine-Pfads für JDBC-Verbindungen und die Generierung und den Import von Java-Code finden Sie unter Java Virtual Machine-Einstellungen 788.

6.4.2 Optionen für den Import von Binärdateien

Bei Aufruf des Menübefehls **Projekt | Binärtypen importieren** werden Sie in einem der Schritte des Assistenten aufgefordert, Optionen für den Import von Binärdateien zu definieren. Im Folgenden wird beschrieben, welche Optionen Sie definieren können. Beachten Sie, dass die Dialogfeldoptionen je nachdem, ob Sie .NET- oder Java-Binärdateien importieren, etwas unterschiedlich sein können.



Dialogfeld "Optionen für den Import von Binärdateien"

Automatische Typinkludierung

.NET- oder Java-Binärdateien können verschiedene externe Assemblys oder Pakete referenzieren. Aktivieren Sie die Option **alle referenzierten Typen hinzufügen...** , wenn alle Typen, die von den in der Binärdatei inkludierten Typen referenziert werden, importiert werden sollen.

Um referenzierte Typen nur für bestimmte Java-Pakete oder .NET-Namespaces zu importieren, geben Sie die entsprechenden Pakete oder Namespaces in das Textfeld daneben ein. Mehrere Pakete oder Namespaces können durch Kommas, Semikola oder Leerzeichen voneinander getrennt werden.

Angenommen, die Quell-.NET .dll-Datei referenziert Typen aus den Namespaces `System.Reflection` und `System.Data`. Wenn Typen aus dem Namespace `System.Reflection`, nicht aber solche aus dem Namespace `System.Data` importiert werden sollen, aktivieren Sie die Option **alle referenzierten Typen hinzufügen, optional auf die folgenden Pakete einschränken:** und geben Sie in das Textfeld "System.Reflection" ein.

Inhaltseinschränkung

Aktivieren Sie die Option **Nur Typen importieren,** um Member wie Felder, Operationen, Eigenschaften usw. zu überspringen.

Aktivieren Sie die Option **nur Elemente importieren mit Sichtbarkeit größer oder gleich:**, um Typen und Typmember nach ihrer Sichtbarkeit zu importieren. In der Tabelle unten ist die Sichtbarkeit von Typen beginnend mit Typen mit der geringsten Sichtbarkeit aufgelistet. Wenn Sie z.B. "private" auswählen, werden alle Typen importiert, während bei Auswahl von "public" nur öffentliche Typen und Typmember importiert werden.

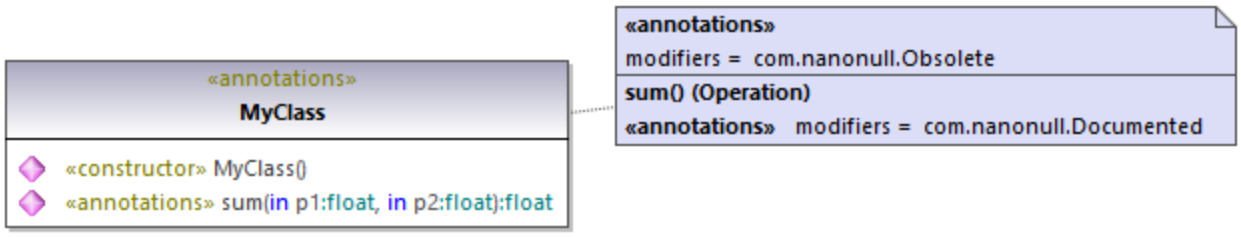
Anmerkung: Wenn das Kontrollkästchen deaktiviert ist, werden alle Typen unabhängig von ihrer Sichtbarkeit importiert.

.NET	Java
private	private
internal	package (Standardsichtbarkeit, wenn kein expliziter Modifier vorhanden ist)
protected	protected
public	public

Die Option **Attributabschnitte unterdrücken** ist auf .NET-Binärdateien anwendbar. Standardmäßig importiert UModel die in der Binärdatei gefundenen C#- oder VB.NET-Attribute. Aktivieren Sie die Option **Attributabschnitte unterdrücken,** wenn Attribute nicht importiert werden sollen. Andernfalls wird auf Member, die in Originalquellcode mit Attributen ausgestattet waren, nach Import der Binärdatei in das Modell das Stereotyp `<<attributes>>` angewendet. Wenn Attribute importiert werden, können Sie diese im Diagramm als Eigenschaftswerte anzeigen, Klicken Sie dazu mit der rechten Maustaste im Diagramm auf die Klasse und wählen Sie im Kontextmenü den Befehl **Eigenschaftswerte | Alle**. Nähere Informationen dazu finden Sie unter Stereotype und Eigenschaftswerte 155.

Die Option **Attributabschnitte unterdrücken** kann auch auf Java-Binärdateien angewendet werden. Standardmäßig importiert UModel die in der Binärdatei gefundenen Java-Annotationen, vorausgesetzt ihre Beibehaltungsrichtlinie wurde als `RUNTIME` (nicht `CLASS` oder `SOURCE`) definiert. Wenn keine Annotationen importiert werden sollen, aktivieren Sie die Option **Annotations-Modifier unterdrücken**. Wenn Annotationen importiert werden, erhalten Member, die in der Originaldatei Annotationen hatten, das Stereotyp `<<annotations>>` und Annotationen werden, wie unten gezeigt, als Eigenschaftswerte angezeigt.

© 2016-2022 Altova GmbH



Stile für Attributabschnitte

Diese Optionen sind nur auf .NET-Binärdateien anwendbar. Wie bereits zuvor erwähnt, werden Typen oder Typmember in UModel als Eigenschaftswerte importiert, falls diese im ursprünglichen Quellcode Attribute hatten.

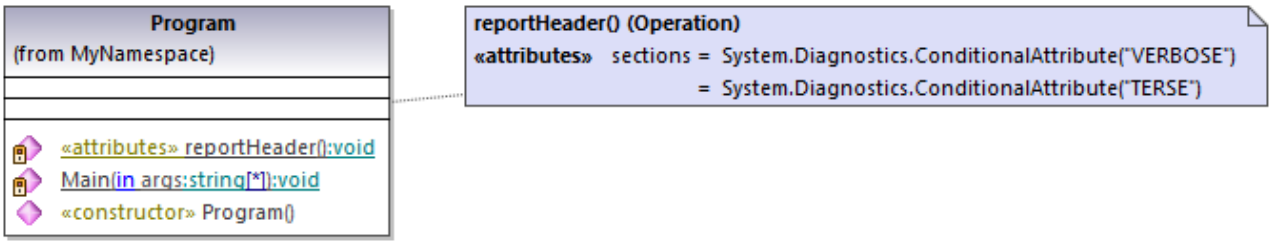
Am besten lässt sich die Option **nur ein Attribut pro Attributabschnitt erstellen** anhand eines Beispiels erläutern. Angenommen, im ursprünglichen C#-Quellcode ist eine Methode mit zwei Attributen definiert:

```
using System;
using System.Diagnostics;

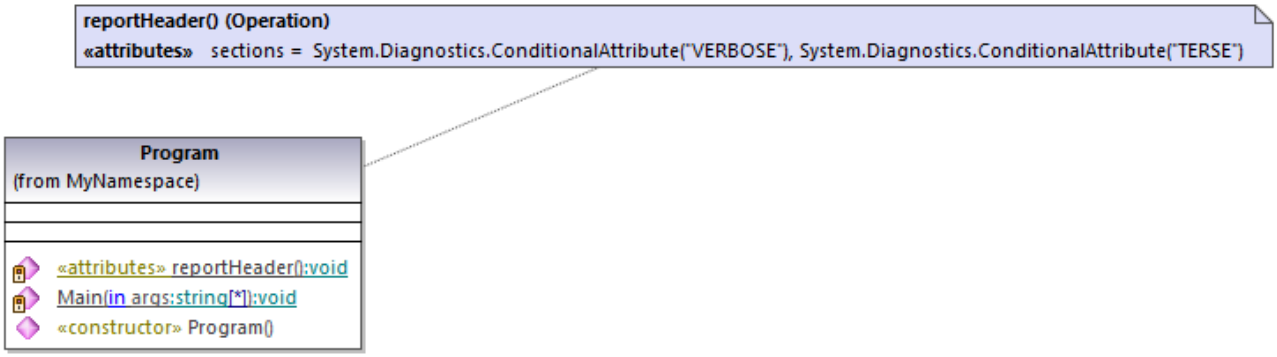
namespace MyNamespace
{
    class Program
    {
        [Conditional("VERBOSE"), Conditional("TERSE")]
        static void reportHeader()
        {
            Console.WriteLine("This is the header");
        }

        static void Main(string[] args)
        {
            reportHeader();
        }
    }
}
```

Wenn beim Import der Binärdatei die Option **nur ein Attribut pro Attributabschnitt erstellen** aktiviert ist, würde jedes Attribut im Element "Eigenschaftswerte" in einer eigenen Zeile angezeigt:



Andernfalls würde die Attribute durch Kommas getrennt:



Mit der Option **'Attribut'-Suffix bei Attributtypnamen unterdrücken** schließlich wird das 'Attribute'-Suffix eines Attributtyps entfernt. So würde z.B. ein im Quellcode definierter Attributtyp `System.Xml.Serialization.XmlTypeAttribute` als `System.Xml.Serialization.XmlType` importiert, wenn diese Option aktiviert ist.

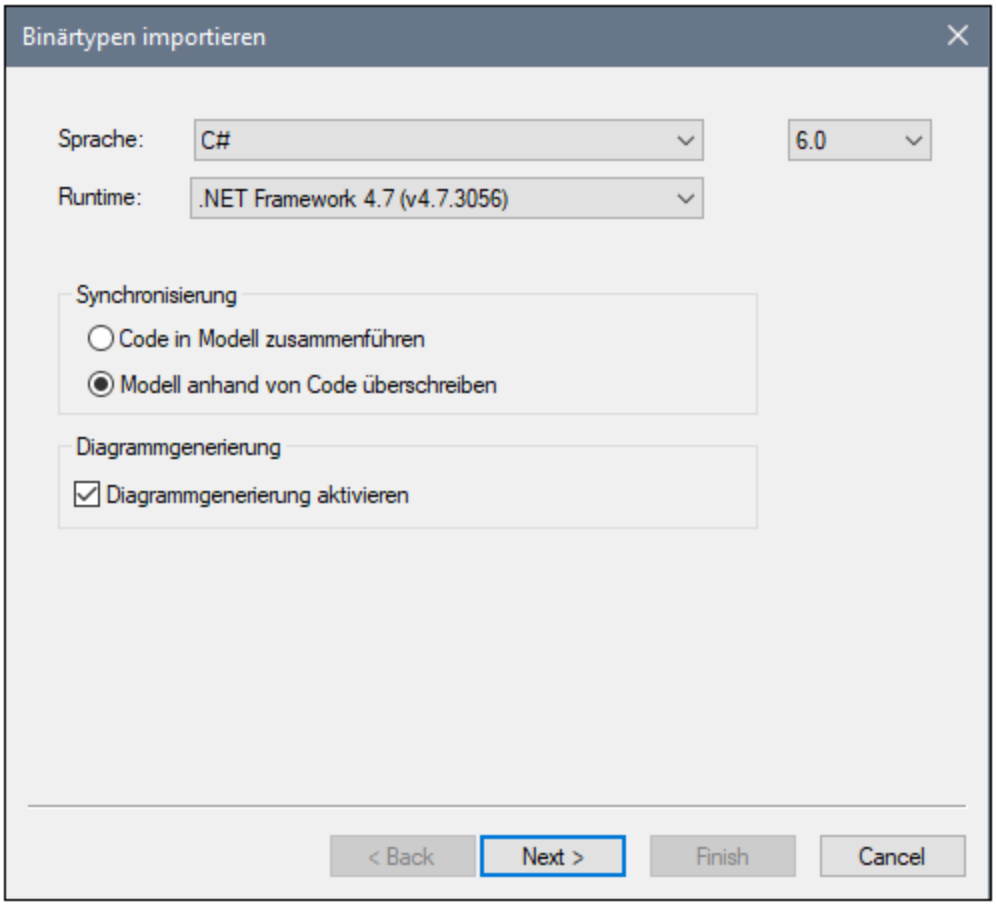
6.4.3 Beispiel: Import von .NET Assemblys

In diesem Beispiel wird gezeigt, wie Sie Binärtypen aus dem .NET Global Assembly Cache (GAC) in ein C#-UModel-Projekt importieren. Die Vorgangsweise ist ähnlich wie beim Import von Binärtypen aus einer Standalone `.dll`- oder `.exe`-Datei. Eine Anleitung zum Import von Java `.class`-Dateien finden Sie im nächsten Kapitel 233.

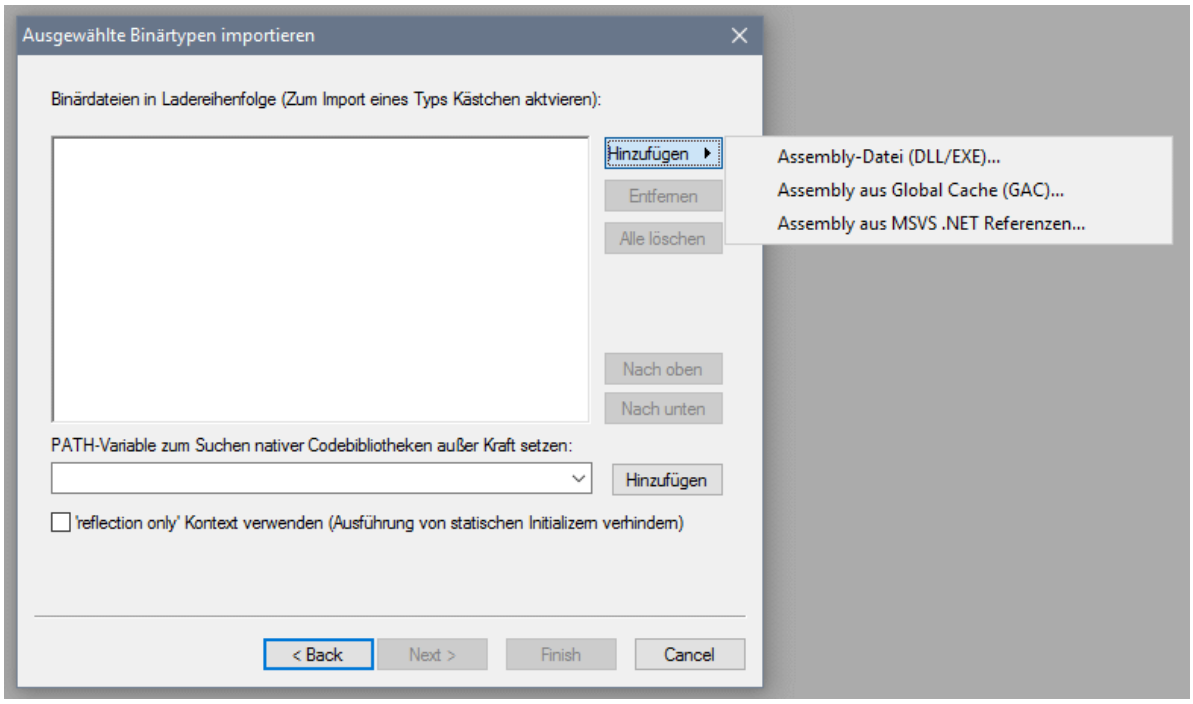
So importieren Sie Binärdateien aus dem .NET Global Assembly Cache:

1. Klicken Sie im Menü **Projekt** auf **Binärtypen importieren** (*siehe Abbildung unten*).

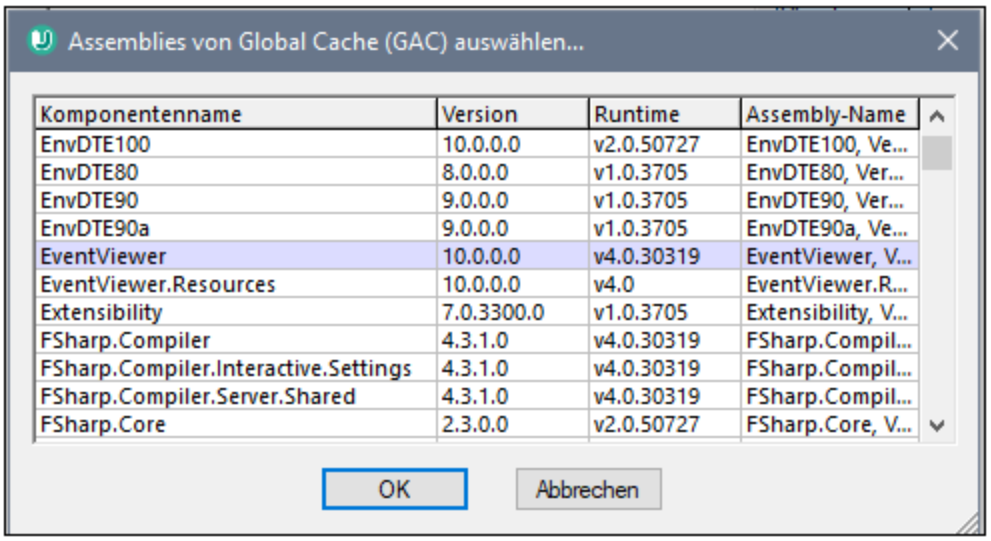
 © 2016-2022 Altova GmbH



2. Wählen Sie die Zielsprache des UModel-Projekts aus (C#, VB.NET, Java). In diesem Beispiel wird C# ausgewählt, da wir eine .NET GAC Assembly importieren.
3. Wenn Sie eine bestimmte Sprachversion für das importierte UModel-Projekt festlegen möchten, wählen Sie diese aus dem benachbarten Textfeld aus. In diesem Beispiel wird C# 7.3 ausgewählt.
4. Wählen Sie optional eine .NET Runtime-Version aus der **Runtime** Dropdown-Liste aus. Die Standardoption ist *beliebige (Disassembler verwenden)*. In diesem Fall wählt UModel die für die importierte Binärdatei am besten geeignete Reflection APIs aus.
5. Wenn Sie Binärtypen in ein neues Projekt importieren, wählen Sie entweder **Code in Modell zusammenführen** oder **Modell anhand von Code überschreiben** aus.
6. Um anhand der importierten Binärtypen optional Klassen- und Paketdiagramme zu generieren, aktivieren Sie das Kontrollkästchen **Diagrammgenerierung aktivieren**. Wenn Sie diese Option auswählen, stehen in den nächsten Schritten weitere Diagrammgenerierungsoptionen zur Verfügung. Siehe auch Generieren von Klassendiagrammen 462 und Generieren von Paketdiagrammen 473.
7. Klicken Sie auf **Weiter**.
8. Klicken Sie auf **Hinzufügen | Assembly aus Global Cache (GAC)** (*siehe Abbildung unten*). Beachten Sie, dass die Option **Assembly aus Global Cache (GAC)** nur für NET Framework 2.x-4.x zur Verfügung steht. Für .NET Core, .NET 5 und spätere Versionen ist der GAC nicht relevant. Nähere Informationen dazu finden Sie in der Microsoft-Dokumentation. Um Assembly-Dateien für .NET Core, .NET 5 und .NET 6 zu importieren, müssen Sie die erforderlichen Dateien aus dem GAC extrahieren. Klicken Sie anschließend auf **Hinzufügen | Assembly-Datei (DLL/EXE)**, wählen Sie die Assembly-Dateien manuell aus und fügen Sie sie zum Projekt hinzu.



9. Wählen Sie eine Assembly aus dem Dialogfeld aus. In diesem Beispiel wurde die Assembly *EventViewer* ausgewählt (*siehe Abbildung unten*).



10. Wählen Sie die zu importierenden Typen aus und klicken Sie auf **Weiter**. Nähere Informationen zu anderen Optionen des Dialogfelds **Ausgewählte Binärtypen importieren** finden Sie in den Anmerkungen weiter unten.
11. Wählen Sie die entsprechenden Importoptionen aus. Nähere Informationen dazu finden Sie unter Optionen für den Import von Binärdateien 227.
12. Wenn Sie in Schritt 6 die Diagrammgenerierung aktiviert haben, klicken Sie auf **Weiter** und konfigurieren Sie die entsprechenden Optionen dafür. Klicken Sie andernfalls auf **Fertig stellen**.

UModel führt die Konvertierung durch und zeigt im Fenster **Meldungen** die Fortschritte an. Wenn die

 © 2016-2022 Altova GmbH

Konvertierung von Binärtypen nicht durchgeführt werden kann, finden Sie in der Fehlermeldung eventuell zusätzliche Informationen dazu. So kann es z.B. vorkommen, dass die Binärdatei, die Sie versuchen zu importieren, für eine Runtime-Version gedacht ist, die neuer als die von Ihnen im Dialogfeld **Binärtypen importieren** ausgewählte ist. Wählen Sie in diesem Fall eine neuere Runtime-Version aus und versuchen Sie es erneut.

Anmerkungen:

- Das Textfeld **PATH-Variable ... außer Kraft setzen** im Dialogfeld **Ausgewählte Binärtypen importieren** gilt nur für Java. Fügen Sie optional hier alle Java-Klassenpfade ein, die zusätzlich zu den aus der Umgebungsvariablen `CLASSPATH` ausgelesenen abgefragt werden sollen. Klicken Sie alternativ dazu auf **Hinzufügen** und navigieren Sie zu den erforderlichen Ordnern.
- Das Kontrollkästchen **'reflection only'-Kontext verwenden...** im Dialogfeld **Ausgewählte Binärtypen importieren** gilt nur, wenn Sie eine C#- oder VB.NET-Binärdatei importieren. Dies ist nützlich, wenn Sie eine Bibliothek importieren, die Abhängigkeiten hat, die nicht aufgelöst oder geladen werden können. Wenn Sie dieses Kontrollkästchen aktivieren, wird statischer Initializer-Code, der beim Import zu Fehlern führen könnte, nicht ausgeführt.

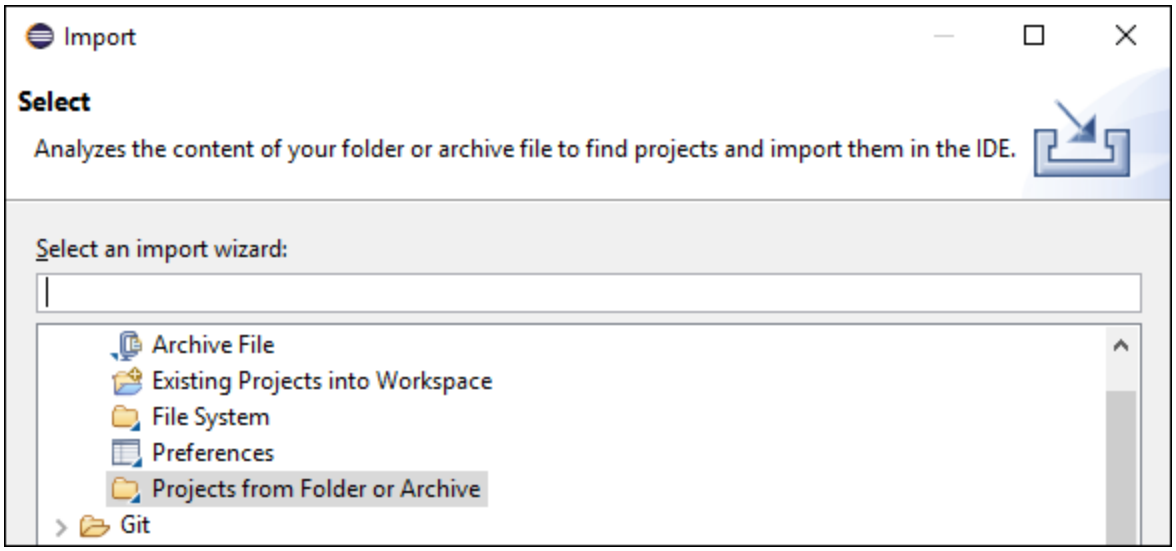
6.4.4 Beispiel: Import von Java .class-Dateien

In diesem Beispiel wird gezeigt, wie Sie kompilierte Java .class-Dateien in UModel importieren. Die Java .class-Quelldateien in diesem Beispiel stammen aus einem mit UModel erstellten Java-Tutorial-Projekt, Sie können jedoch alternativ dazu auch andere .class-Dateien verwenden.

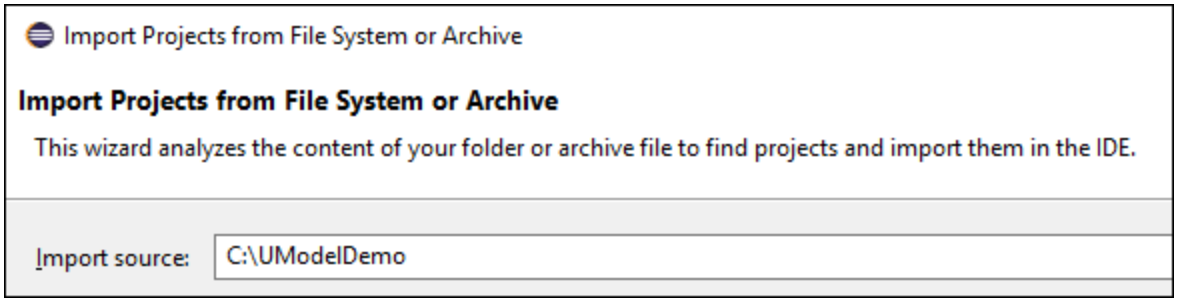
Kompilieren von mit UModel generiertem Java-Code (optional)

In diesem Abschnitt wird gezeigt, wie Sie ein mit UModel generiertes Java-Demo-Projekt mit Eclipse kompilieren. Beachten Sie, dass dies ein rein optionaler Schritt ist. Ziel ist es, eine kompilierte .class-Datei zu erhalten. Wenn Sie bereits Java .class-Dateien zur Verfügung haben, können Sie den Schritt auch überspringen. In diesem Beispiel wird aus praktischen Gründen Eclipse als Kompilierungsumgebung verwendet, Sie können dazu aber auch die Java-Befehlszeile oder eine Java-Entwicklungsumgebung verwenden.

1. Erstellen Sie, wie in Beispiel: Generieren von Java-Code 193 gezeigt, ein einfaches Java-Projekt mit UModel, falls Sie dies noch nicht getan haben. Es handelt sich hierbei um ein ganz einfaches Beispiel bestehend aus einem Java-Paket mit nur einer Klasse. Nach Fertigstellung des Beispiels enthält das Verzeichnis **C:\UModelDemo\src** den benötigten Java-Quellcode.
2. Starten Sie Eclipse. Klicken Sie im Menü **File** auf **Import**.

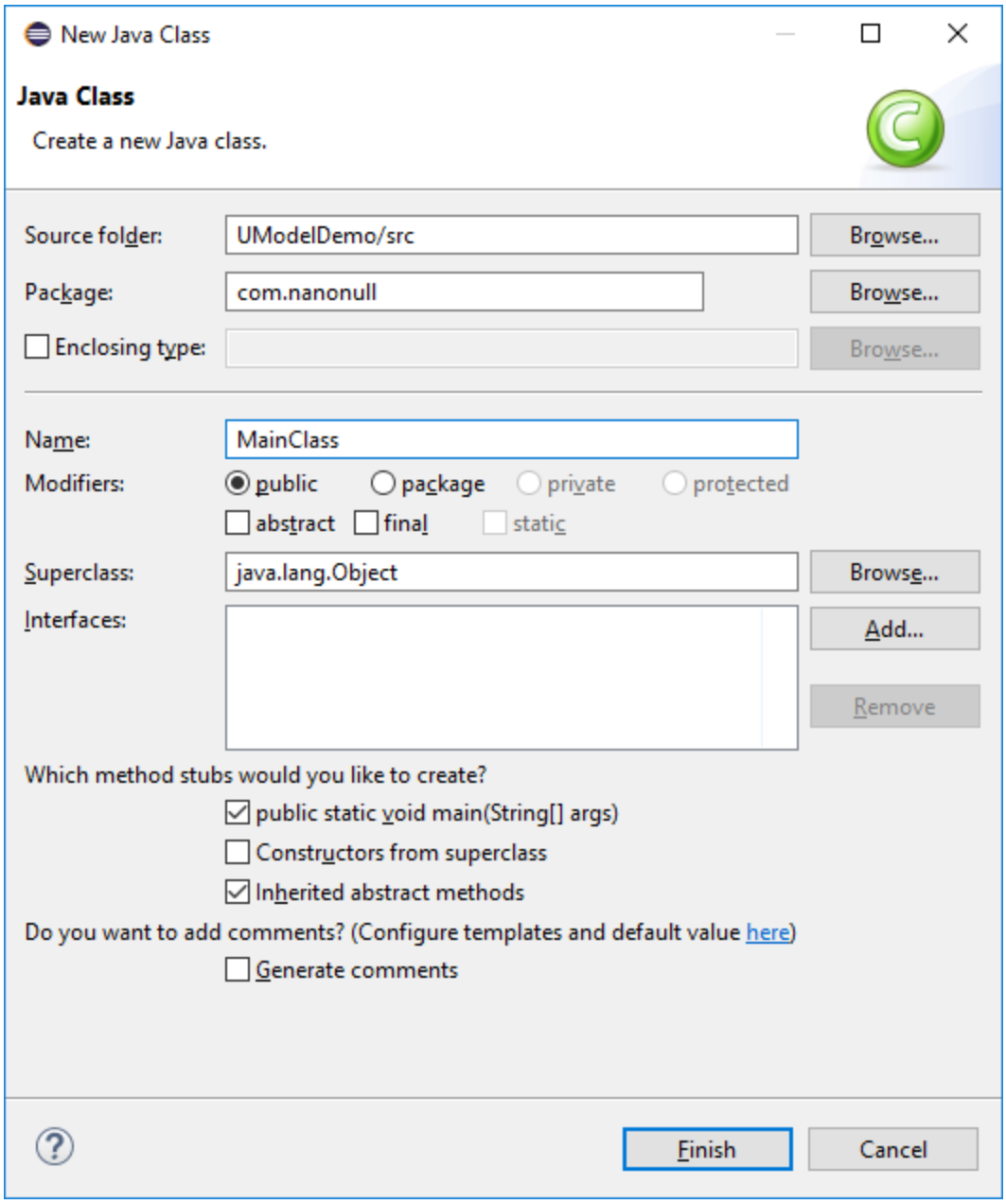


3. Wählen Sie **Projects from Folder or Archive** und klicken Sie auf **Next**.



4. Geben Sie als Verzeichnis **C:\UModelDemo** ein und klicken Sie auf **Finish**.
5. Klicken Sie mit der rechten Maustaste im Paket-Explorer von Eclipse auf das Paket **com.nanonull** und wählen Sie im Kontextmenü den Befehl **New | Class**.
6. Geben Sie einen Klassennamen ein (In diesem Beispiel "MainClass") und aktivieren Sie das Kontrollkästchen **public static void main...**.

© 2016-2022 Altova GmbH



7. Klicken Sie im Menü **Run** auf **Run**.

Sie haben nun das mit UModel generierte Java-Projekt kompiliert. Die kompilierten .class-Dateien sollten sich im Unterverzeichnis **bin** Ihres Projektverzeichnisses befinden.

Machen Sie sich eine Notiz, welche Java-Version Sie für die Kompilierung verwendet haben - Sie benötigen diese Version, wenn Sie beabsichtigen, die Binärtypen später zu importieren. Wenn Sie die Eigenschaften Ihres Eclipse-Projekts nicht geändert haben, wurde es wahrscheinlich mit der Standard-Java-Version für Eclipse kompiliert. Um die Standard-Java-Version zu sehen, gehen Sie in Eclipse folgendermaßen vor:

1. Klicken Sie im Menü **Window** auf **Preferences**.
2. Klicken Sie auf **Java** und anschließend auf **Installed JREs**.

Importieren von Java .class-Dateien

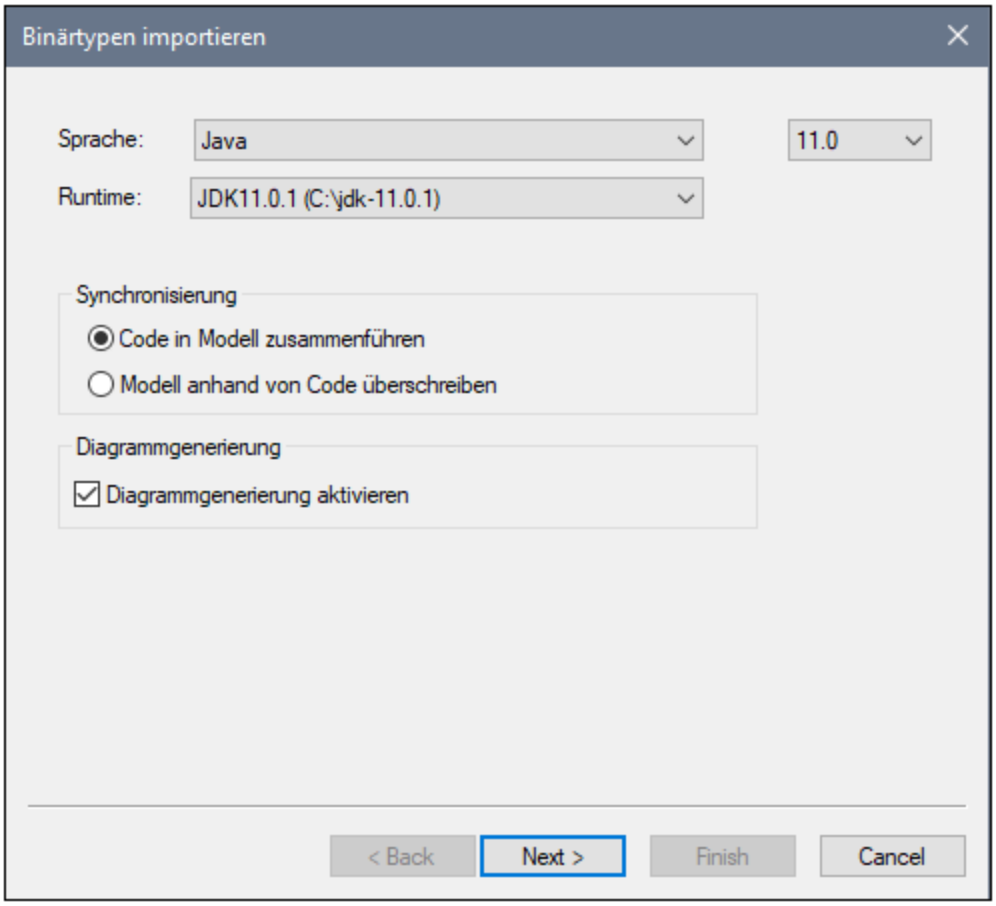
Wenn Sie bereits .class-Binärdateien wie die zuvor kompilierten zur Verfügung haben, können Sie diese nun in UModel importieren.

1. Erstellen Sie ein neues UModel-Projekt oder öffnen Sie ein vorhandenes. Wir importieren Binärtypen in diesem Beispiel in ein neues Projekt.
2. Wenn Ihr Projekt die Java JDK-Typen noch nicht enthält, gehen Sie folgendermaßen vor:

   a. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.
   b. Klicken Sie auf das Register **Java** und wählen Sie **Java JDK (types only)**.
   c. Wählen Sie **Durch Referenz** aus, wenn Sie gefragt werden.

Anmerkung: Dies ist ein optionaler Schritt, mit dem Sie normalerweise vermeiden, dass das Paket "Unbekannte externe Elemente" nach abgeschlossenem Import im Projekt angezeigt wird.

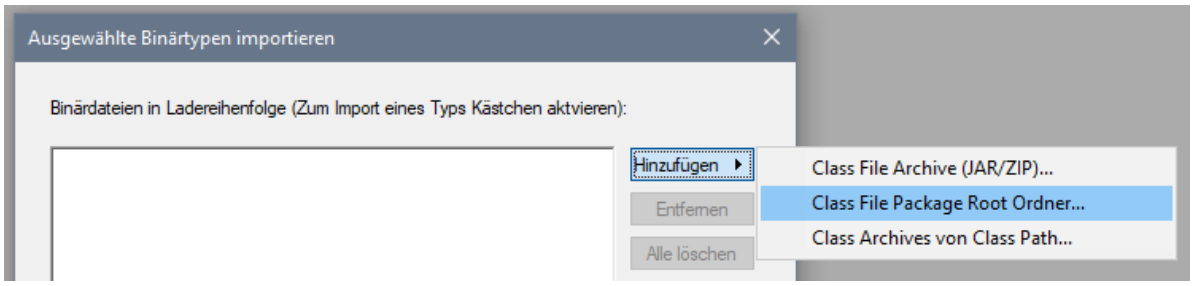
3. Klicken Sie im Menü **Projekt** auf **Binärtypen importieren.**
4. Wählen Sie unter Sprache **Java** sowie die Java-Version, mit der der Java-Code kompiliert wurde, aus (z.B. 11.0).
5. Wählen Sie die Java Runtime aus, die von UModel zum Extrahieren von Informationen aus den Binärdateien verwendet werden soll (die so genannte "Reflection"). Die Runtime-Version muss der im vorherigen Schritt ausgewählten Java-Version entsprechen oder neuer sein.



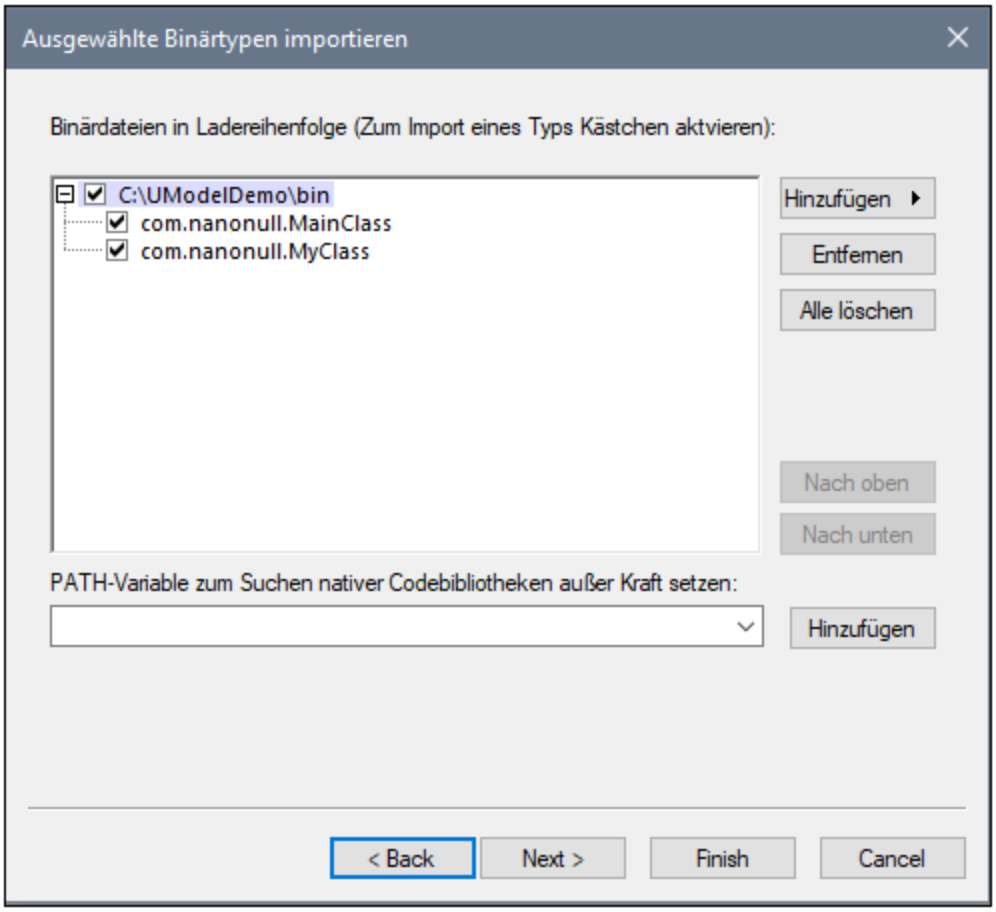
© 2016-2022 Altova GmbH

Anmerkung: Die **Runtime** Dropdown-Liste enthält nur automatisch ermittelte Java JDKs und JREs. Wenn Ihr JDK oder JRE nicht in der Liste aufscheint, wählen Sie den Eintrag **Java Runtime-Benutzerpfade bearbeiten** und navigieren Sie zu dem Verzeichnis, in dem die entsprechende Distribution auf Ihrem Rechner installiert wurde, siehe Hinzufügen von benutzerdefinierten Java Runtimes 226.

6. Wenn Sie Binärtypen in ein neues Projekt importieren, wählen Sie entweder **Code in Modell zusammenführen** oder **Modell anhand von Code überschreiben** aus. Wählen sie andernfalls **Code in Modell zusammenführen** aus.
7. Um anhand der importierten Binärtypen optional Klassen- und Paketdiagramme zu generieren, aktivieren Sie das Kontrollkästchen **Diagrammgenerierung aktivieren**. Wenn Sie diese Option auswählen, stehen in den darauf folgenden Schritten weitere Diagrammgenerierungsoptionen zur Verfügung, siehe auch Generieren von Klassendiagrammen 462 und Generieren von Paketdiagrammen 473.
8. Klicken Sie auf **Weiter**.

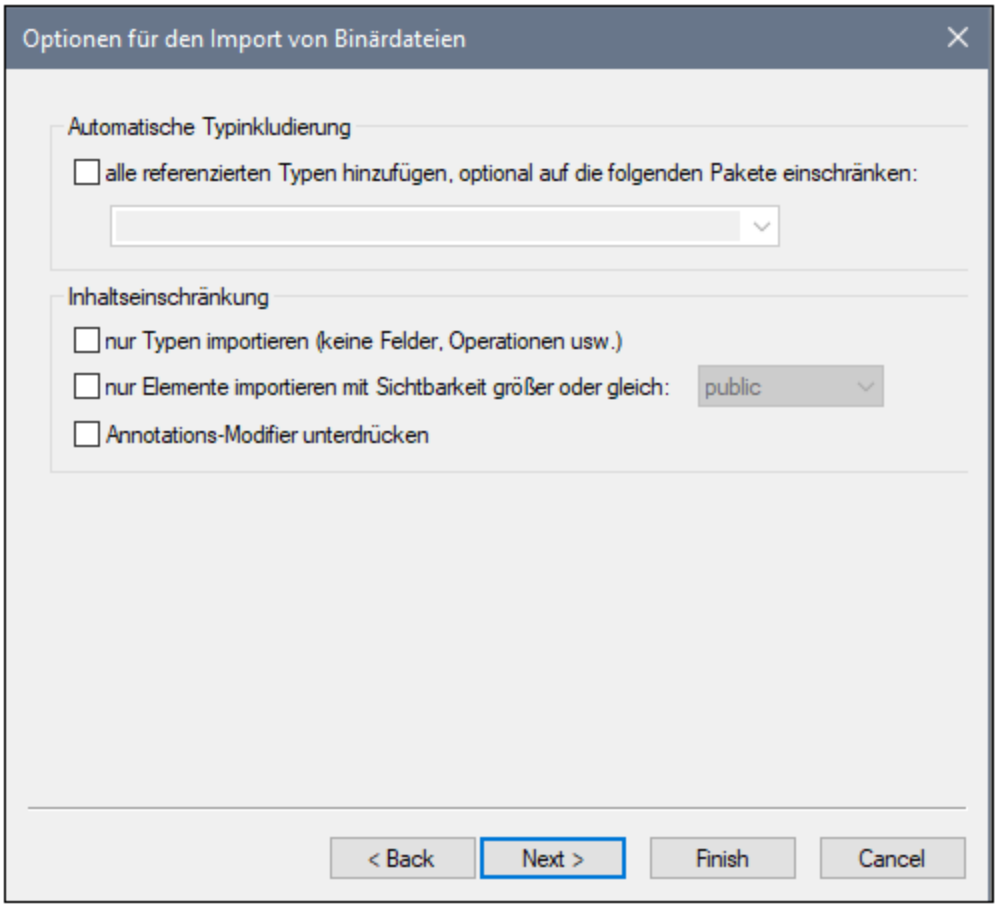


9. Wir importieren in diesem Beispiel Java .class-Dateien aus einer Paket-Root. Wählen Sie die Option **Hinzufügen | Class File Package Root Ordner** und navigieren Sie zum Verzeichnis **C:\UModelDemo\bin**. Falls dieses Verzeichnis fehlt, muss zuerst das Projekt kompiliert werden, wie im ersten Teil dieses Tutorials beschrieben.



10. Wählen Sie die gewünschten Klassen aus und klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH



11. Wählen Sie die entsprechenden Importoptionen aus, siehe Optionen für den Import von Binärdateien[227].
12. Wenn Sie in einem früheren Schritt die Diagrammgenerierung aktiviert haben, klicken Sie auf Weiter und konfigurieren Sie die entsprechenden Optionen dafür. Klicken Sie andernfalls auf **Fertig stellen**.

UModel führt die Konvertierung durch und zeigt im Fenster **Meldungen** die Fortschritte an. Wenn die Konvertierung von Binärtypen nicht durchgeführt werden kann, finden Sie in der Fehlermeldung eventuell zusätzliche Informationen dazu. So kann es z.B. vorkommen, dass die Binärdatei, die Sie versuchen zu importieren, für eine Runtime-Version gedacht ist, die neuer als die von Ihnen im Dialogfeld **Binärtypen importieren** ausgewählte ist. Wählen Sie in diesem Fall eine neuere Runtime-Version aus und versuchen Sie es erneut.

6.5 Synchronisieren von Modell und Quellcode

Sie können Modell und Code in beiden Richtungen und auf unterschiedlichen Ebenen (z.B. Projekt, Paket oder Klasse) synchronisieren.

> Wenn UModel (Enterprise oder Professional) als Eclipse- oder Visual Studio Plug-in ausgeführt wird, erfolgt die Synchronisierung zwischen Modell und Code automatisch. Eine manuelle Synchronisierung ist auf Projektebene möglich; die Option zum Aktualisieren einzelner Klassen oder Pakete steht nicht zur Verfügung. Nähere Informationen finden Sie unter UModel Plug-in für Visual Studio[665] und UModel Plug-in für Eclipse[676].

Bei Rechtsklick auf ein Element in der Modellstruktur (z.B. eine Klasse) werden im Kontextmenü unter dem Eintrag **Code Engineering** die Befehle zur Synchronisierung oder zum Zusammenführen von Code angezeigt:

- **Merge Programmcode von UModel Projekt *****
- **Merge UModel Projekt von Programmcode *****

> *** ist je nach aktueller Auswahl ein Projekt, Paket, eine Komponente oder eine Klasse usw.

Je nachdem, welche Einstellungen Sie unter **Projekt | Synchronisierungseinstellungen** definiert haben, können die Namen dieser beiden Befehle auch folgendermaßen lauten:

- **Überschreibe Programmcode von UModel Projekt *****
- **Überschreibe UModel Projekt von Programmcode *****

Um das gesamte Projekt (nicht aber Klassen, Pakete oder andere lokale Elemente) zu aktualisieren, können Sie auch die beiden folgenden Befehle aus dem **Projekt**-Menü von UModel verwenden:

- **Merge (oder Überschreibe) Programmcode aus UModel Projekt**
- **Merge (oder Überschreibe) UModel Projekt aus Programmcode**

Aus Gründen der Einfachheit werden die oben aufgelisteten Befehle in diesem Kapitel allgemein als Codesynchronisierungsbefehle bezeichnet.

Wählen Sie eine der folgenden Methoden, um eine Synchronisierung auf Projekt- oder Root-Paketebene durchzuführen:

- Klicken Sie mit der rechten Maustaste auf das Root-Paket in der Modell-Struktur und wählen Sie den gewünschten Codesynchronisierungsbefehl aus.
- Klicken Sie im Menü **Projekt** auf den gewünschten Codesynchronisierungsbefehl.

So führen Sie eine Synchronisierung auf Paketebene durch:

1. Drücken Sie Umschalt + Klick oder Strg + Klick, um das/die gewünschte(n) Paket(e) auszuwählen
2. Klicken Sie mit der rechten Maustaste auf die Auswahl und wählen Sie den gewünschten Codesynchronisierungsbefehl aus.

© 2016-2022 Altova GmbH

So führen Sie eine Synchronisierung auf Klassenebene durch:

1. Drücken Sie Umschalt + Klick oder Strg + Klick, um die gewünschte(n) Klasse(n) auszuwählen
2. Klicken Sie mit der rechten Maustaste auf die Auswahl und wählen Sie den gewünschten Codesynchronisierungsbefehl aus.

Um beim Synchronisieren von Modell und Code unerwünschte Ergebnisse zu vermeiden, betrachten Sie bitte die folgenden Szenarien:

Auswahl des Befehls **Überschreibe UModel Projekt aus Programmcode** im Menü **Projekt**	• Dabei werden alle Verzeichnisse (Projektdateien) aller verschiedenen von Ihnen in Ihrem Projekt definierten Codesprachen überprüft. • Neue Dateien werden ermittelt und zum Projekt hinzugefügt. • In Ihrem Meldungsfenster erscheint der Eintrag "Quelldateien werden in 'C:\UMTest' gesammelt".
Rechtsklick auf eine Klasse oder Schnittstelle in der Modell-Struktur und Auswahl von **Code Engineering | Überschreibe UModel Klasse von Programmcode**	• Daraufhin wird nur die ausgewählte Klasse (Schnittstelle,...) Ihres Projekts aktualisiert. • Wenn der Quellcode neue oder seit der letzten Synchronisierung geänderte Klassen enthält, werden diese Änderungen nicht zum Modell hinzugefügt.
Rechtsklick auf eine Komponente in der Modell-Struktur (im Component View-Paket) und Auswahl von **Code Engineering | Überschreibe UModel Komponente von Programmcode**	• Daraufhin wird nur das entsprechende Verzeichnis (bzw. die entsprechende Projektdatei) aktualisiert. • Neue Dateien im Verzeichnis (in der Projektdatei) werden ermittelt und zum Projekt hinzugefügt. • In Ihrem Meldungsfenster erscheint der Eintrag "Quelldateien werden in 'C:\UMTest' gesammelt".

Anmerkung:

Unter Umständen wird beim Synchronisieren von Code ein Dialogfeld angezeigt, in dem Sie aufgefordert werden, Ihr UModel-Projekt vor dem Synchronisieren zu aktualisieren. Dies kommt nur bei UModel-Projekten vor, die mit einer Vorversion der letzten Release erstellt wurden. Klicken Sie bitte auf **JA** um Ihr Projekt zu aktualisieren und Ihre Projektdatei zu speichern. Anschließend wird diese Aufforderung nicht mehr angezeigt.

6.5.1 Tipps zur Synchronisierung

Umbenennen von Classifiern und Reverse Engineering

Der unten beschriebenen Vorgang beim Reverse Engineering oder der automatischen Synchronisierung bezieht sich sowohl auf die Standalone-Applikation als auch auf die Plug-In-Versionen (Visual Studio oder Eclipse).

Wenn Sie einen Classifier im Code-Fenster Ihrer Programmierapplikation umbenennen, wird er gelöscht und als neuer Classifier in die **Modell-Struktur** eingefügt.

Der neue Classifier wird nur in diejenigen **Modellierungsdiagramme** eingefügt, die beim Reverse Engineering automatisch erstellt werden, oder wenn ein Diagramm mit dem Befehl **In neuem Diagramm anzeigen |**

Inhalt erstellt wird. Der neue Classifier wird an der Standardposition im Diagramm eingefügt. Diese ist wahrscheinlich nicht mit der vorherigen Position identisch.

Siehe auch Refactoring und Synchronisierung von Code 243.

Automatische Generierung von Komponentenrealisierungen

UModel kann in Rahmen der Codegenerierung automatisch Komponentenrealisierungen generieren. Komponentenrealisierungen werden nur generiert, wenn es absolut klar ist, welcher Komponente eine Klasse zugewiesen werden soll:

- Es ist nur eine Visual Studio-Projektdatei im .ump-Projekt vorhanden.
- Es gibt mehrere Visual Studio-Projekte, doch deren Klassen sind im Modell streng getrennt.

So aktivieren Sie die automatische Generierung von Komponentenrealisierungen:

1. Wählen Sie den Menübefehl **Extras | Optionen.**
2. Klicken Sie auf das Register **Code Engineering** und aktivieren Sie die Option **Fehlende Komponentenrealisierungen generieren.**

Automatische Komponentenrealisierungen werden für einen **Classifier** generiert, dem eine (und nur eine) Komponente zugewiesen werden kann.

- ohne Komponentenrealisierung oder
- eine im Namespace einer Codesprache enthaltene Komponente

Die Art, wie die Komponente gesucht wird, ist in den beiden Fällen unterschiedlich:

Komponente, die eine Code-Projektdatei repräsentiert (Eigenschaft "**projectfile**" ist aktiviert)

- wenn es EINE Komponente gibt, die Classifier im Paket, das sie enthält, hat/realisiert
- wenn es EINE Komponente gibt, die Classifier in einem Unterpaket, des Pakets, das sie enthält, hat/realisiert (von oben nach unten)
- wenn es EINE Komponente gibt, die Classifier in einem der übergeordneten Pakete hat/realisiert (von unten nach oben)
- wenn es EINE Komponente gibt, die Classifier in einem Unterpaket eines der übergeordneten Paket hat/realisiert (von oben nach unten)

Komponente, die ein Verzeichnis repräsentiert (Eigenschaft "**directory**" ist aktiviert)

- wenn es EINE Komponente gibt, die Classifier im Paket, das sie enthält, hat/realisiert
- wenn es EINE Komponente gibt, die Classifier in einem der übergeordneten Pakete hat/realisiert (von unten nach oben)

Anmerkungen:

- Die Option "**Code Engineering | Fehlende Komponentenrealisierungen generierten**" muss aktiviert sein.
- Sobald bei einem der obigen Schritte EINE verwendbare Komponente gefunden wird, wird diese Komponente verwendet und die restlichen Schritte werden ignoriert!

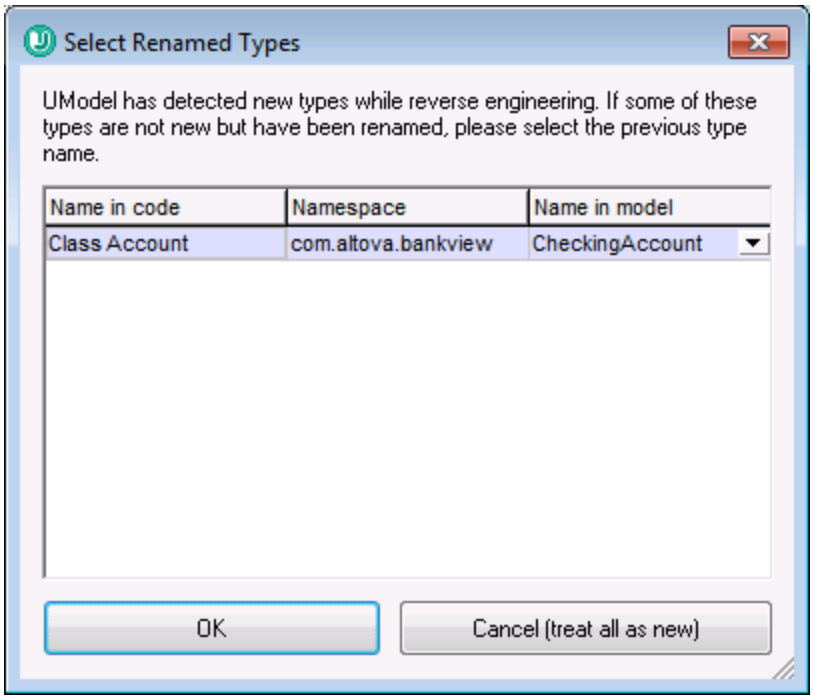
Fehler/Warnungen:

© 2016-2022 Altova GmbH

- Wenn keine verwendbare Komponente gefunden wird, wird eine Warnmeldung generiert (Meldungsprotokoll)
- Wenn mehrere verwendbare Komponenten gefunden werden, wird eine Fehlermeldung generiert (Meldungsprotokoll)

6.5.2 Refactoring und Synchronisierung von Code

Beim Refactoring von Code müssen oft Klassennamen im Code geändert oder aktualisiert werden. Falls in UModel ab Version 2009 beim Reverse Engineering festgestellt wird, dass neue Typen hinzugekommen oder umbenannt worden sind, wird ein Dialogfeld geöffnet. Die neuen Typen werden in der Spalte "Name im Code" geöffnet, während der vom Programm als ursprünglicher Typname angenommene Name in der Spalte "Name im Modell" aufgelistet wird. UModel ermittelt den Originalnamen anhand von Namespace, Klasseninhalt, Basisklassen und anderen Daten.



Wenn eine Klasse umbenannt wurde, wählen Sie den früheren Klassennamen über die Auswahlliste in der Spalte "Name im Modell" aus, z.B. C1. Damit stellen Sie sicher, dass alle damit in Zusammenhang stehenden Daten beibehalten werden und das Code Engineering korrekt erfolgt.

Ändern von Klassennamen im Modell und Neugenerierung von Code

Wenn Sie ein Modell erstellt haben und anhand dieses Modells Code generiert haben, möchten Sie unter Umständen nochmals Änderungen am Modell vornehmen, bevor Sie die Synchronisierung vornehmen.

Beispiel: Sie haben beschlossen, Klassennamen zu ändern, bevor Sie Code zum zweiten Mal generieren. Da Sie jeder Klasse zuvor einen Dateinamen zugewiesen haben, würde der neue Name der Klasse und der Datei nun im Fenster "Eigenschaften" im Feld "Codedateiname" nicht mehr übereinstimmen.

UModel fragt Sie, ob der Name der Codedatei an den neuen Namen der Klasse angepasst werden soll, bevor Sie die Synchronisierung starten. Beachten Sie, dass Sie auch die Möglichkeit haben, auch die Klassenkonstruktoren zu ändern.

Round-Trip Engineering und Beziehungen zwischen Modellelementen:

Beim Aktualisieren des Modells anhand von Code werden Assoziationen zwischen Modellelementen automatisch angezeigt, wenn die Option **Bearbeiten | Assoziationen automatisch erstellen** im Dialogfeld **Extras | Optionen** aktiviert wurde. Assoziationen werden für jene Elemente angezeigt, bei denen der Attributtyp definiert ist und bei denen sich das referenzierte "Typ"-Modellelement im selben Diagramm befindet.

Die Schnittstellenrealisierungen sowie die Generalisierungen werden beim Aktualisieren des Modells anhand von Code automatisch angezeigt.

6.5.3 Codesynchronisierungseinstellungen

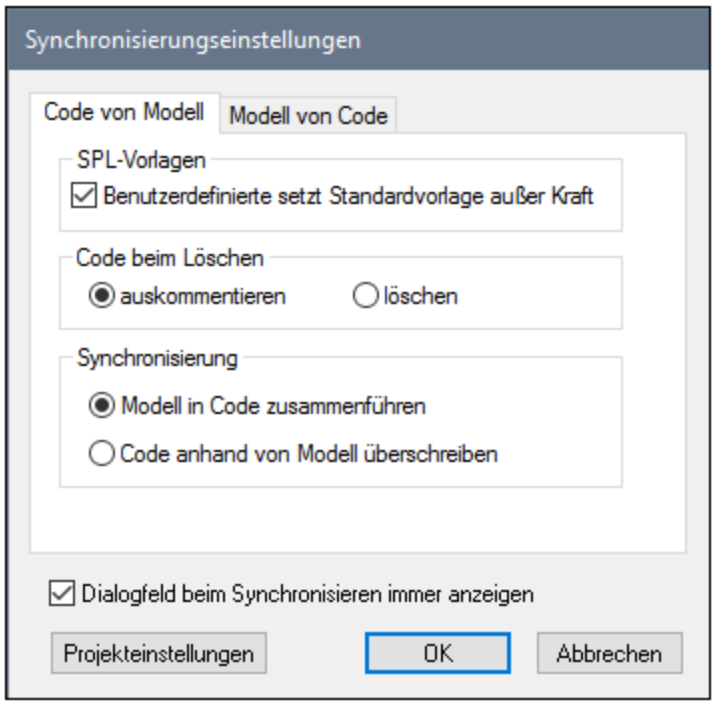
Die Codesynchronisierungseinstellungen sind in den folgenden Szenarien von Bedeutung:

- Wenn anhand des Modells Programmcode generiert wird (d.h. bei Auswahl des Menübefehls **Projekt | Merge Programmcode aus UModel-Projekt** oder **Projekt | Überschreibe Programmcode aus UModel-Projekt**)
- Wenn Quellcode in das Modell importiert wird (d.h. bei Auswahl des Befehls **Projekt | Merge UModel-Projekt aus Programmcode** oder **Projekt | Überschreibe UModel-Projekt aus Programmcode**)
- Wenn eine automatische Synchronisierung in eine der beiden Richtungen durchgeführt wird (Dies gilt für die UModel Enterprise und die Professional Edition, wenn UModel als Visual Studio- oder Eclipse Plug-in ausgeführt wird).

So ändern Sie die Codesynchronisierungseinstellungen:

- Klicken Sie im Menü **Projekt** auf **Synchronisierungseinstellungen**.

© 2016-2022 Altova GmbH



Dialogfeld "Synchronisierungseinstellungen"

Standardmäßig wird das Dialogfeld "Synchronisierungseinstellungen" automatisch jedes Mal angezeigt, wenn Sie einen der Befehle zur Codesynchronisierung auswählen. Damit das Dialogfeld nicht mehr angezeigt wird, deaktivieren Sie das Kontrollkästchen **Dialogfeld beim Synchronisieren immer anzeigen**.

Die verfügbaren Optionen sind auf zwei Registern gruppiert:

- **Code von Modell** (die Optionen auf diesem Register gelten für die Generierung von Programmcode anhand des Modells)
- **Modell von Code** (die Optionen auf diesem Register gelten für den Import von Programmcode in das Modell).

Option	Beschreibung
SPL-Vorlagen	Diese Option ist nur bei Generierung von Programmcode anwendbar. Aktivieren Sie das Kontrollkästchen **Benutzerdefinierte setzt Standardvorlage außer Kraft,** wenn Sie benutzerdefinierte Spy Programming Language (SPL)-Vorlagen erstellt haben, die die mit UModel bereitgestellten Standardvorlagen außer Kraft setzen sollen (siehe auch SPL-Vorlagen 207).
Code beim Löschen	Diese Option ist nur bei Generierung von Programmcode anwendbar. Wählen Sie aus, ob Programmcode bei der Synchronisierung gelöscht oder auskommentiert werden soll (vorausgesetzt die entsprechenden Objekte sind im Modell nicht mehr vorhanden).

Option	Beschreibung
Synchronisierung	Diese Option ist sowohl auf die Generierung als auch auf den Import von Programmcode anwendbar. Sie können damit festlegen, ob Änderungen zusammengeführt oder überschrieben werden sollen. Angenommen, es wurde einmal Code anhand eines Modells generiert und es wurden sowohl am Modell als auch am Code Änderungen vorgenommen, z.B.: • wurde eine neue Klasse X in UModel hinzugefügt • wurde eine neue Klasse Y zum externen Code hinzugefügt. **Modell in Code zusammenführen** bedeutet, dass • die im externen Code neu hinzugefügte Klasse Y beibehalten wird • die über UModel neu hinzugefügte Klasse X zum Code hinzugefügt wird. **Code anhand von Modell überschreiben** bedeutet, dass • die neu hinzugefügte Klasse Y im externen Code gelöscht (oder, je nach aktueller Einstellung, auskommentiert) wird • die neu hinzugefügte Klasse X aus UModel zum Code hinzugefügt wird. **Code in Modell zusammenführen** bedeutet, dass • die neu hinzugefügte Klasse X in UModel beibehalten wird • die neu hinzugefügte Klasse Y aus dem externen Code zum Modell hinzugefügt wird **Modell anhand von Code überschreiben** bedeutet, dass • die neu hinzugefügte Klasse X in UModel gelöscht (oder, je nach aktueller Einstellung, auskommentiert) wird • die neu hinzugefügte Klasse Y aus dem externen Code zum Modell hinzugefügt wird.
Projekteinstellungen	Öffnet das Dialogfeld "Projekteinstellungen", wo Sie die Code Engineering-Einstellungen für die einzelnen Sprachen ändern können. Nähere Informationen zu allen Einstellungen finden Sie unter Optionen für den Code-Import 212 bzw. Codegenerierungsoptionen 186. Sie können das Dialogfeld "Projekteinstellungen" auch über den Menübefehl **Projekt | Projekteinstellungen** aufrufen. Beachten Sie, dass die Projekteinstellungen in diesem Dialogfeld global sind (Sie werden zusammen mit dem Projekt gespeichert und von jedem Rechner, auf dem das UModel-Projekt geöffnet wird, angewendet), während die Optionen, die Sie über das Menü **Extras | Optionen** definieren, lokal sind (Sie sind nur in der aktuellen Installation von UModel anwendbar).

© 2016-2022 Altova GmbH

6.6 UModel-Elemententsprechungen

In diesem Abschnitt sehen Sie die UModel-Elementenentsprechungen für Elemente (Konstrukte) in verschiedenen Programmiersprachen (C++, C#, Java, VB.NET) sowie Datenbanken und XML-Schemas. Die Zuordnungen sind nach Sprache gruppiert und gelten für den Import von Code in das Modell oder die Generierung von Code anhand des Modells.

- C++-Entsprechungen 247
- C#-Entsprechungen 253
- VB.NET-Entsprechungen 273
- Java-Entsprechungen 288
- XML Schema-Entsprechungen 294
- Datenbank-Entsprechungen 304

6.6.1 C++-Entsprechungen

In der Tabelle unten sehen Sie die 1:1-Entsprechungen zwischen C++-Codeelementen und UModel-Modellelementen beim Import von C++-Code in das Modell.

> Die Unterstützung von C++-Attributen ist eingeschränkt. Es werden nur vordefinierte Standardattribute wie `[[noreturn]]`, `[[carries_dependency]]`, `[[deprecated]]` erkannt. Benutzerdefinierte Attribute werden ignoriert.

C++-Projekt

C++		UModel	
Projekt	Projektdatei	Projektdatei	Komponente
	Verzeichnis	Verzeichnis	

C++ Namespace

C++		UModel	
Namespace	Name	Name	Paket <<namespace>>

C++-Klasse / Struktur / Union

C++			UModel		
Klasse / Struktur / Union	Name		Name		Klasse / <<struct>> Klasse /
	Zugriffsmodifikator	public	Sichtbarkeit	public	

C++			UModel		
	protected			protected	<<union>> Klasse
	private			private	
Dateiname			Codedateiname		
verknüpfte Projektdatei / Verzeichnis			Komponentenrealisierung		
base specifier	base types		Generalisierung		
	virtual		Generalisierung <<virtual>>		
	access		Generalisierung <<visibility>> Wert		
Attribute			<<attributes>>		
final			isFinalSpecialization		
Vorlagenparameter	Name		Name		Vorlagenparameter
	template parameter pack		parameterPack		
	type		property @type		
	default		default		
Template Specialization	Argumente		Parameter		<<specialization>>
Feld	Name		Name		Eigenschaft
	Zugriffsmodifikator	public	Sichtbarkeit	public	
		protected		protected	
		private		private	
	type		Typ		
	type modifiers		Typ-Modifier		
	static		static		
	mutable		<<mutable>>		
	thread_local		<<thread_local>>		
	const		<<const>>		
	constexpr		<<constexpr>>		
	in class initializer		default		
	attributes		<<attributes>>		
	volatile		<<volatile>>		
	variable template		<<varTemplate>>		

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="4">C++</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="30"></td><td rowspan="30">Methode</td><td colspan="2">Name</td><td colspan="2">Name</td><td rowspan="30">Operation</td><td rowspan="30"></td></tr>
<tr><td rowspan="3">Zugriffsmodifikator</td><td>public</td><td rowspan="3">Sichtbarkeit</td><td>public</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td colspan="2">static</td><td colspan="2">static</td></tr>
<tr><td colspan="2">virtual</td><td colspan="2"><<virtual>></td></tr>
<tr><td colspan="2">= 0</td><td colspan="2"><<purevirtual>></td></tr>
<tr><td colspan="2">const</td><td colspan="2"><<const>></td></tr>
<tr><td colspan="2">inline</td><td colspan="2"><<inline>></td></tr>
<tr><td colspan="2">= delete</td><td colspan="2"><<delete>></td></tr>
<tr><td colspan="2">= default</td><td colspan="2"><<default>></td></tr>
<tr><td colspan="2">override</td><td colspan="2"><<overrride>></td></tr>
<tr><td colspan="2">final</td><td colspan="2"><<final>></td></tr>
<tr><td colspan="2">volatile</td><td colspan="2"><<volatile>></td></tr>
<tr><td colspan="2">constexpr</td><td colspan="2"><<constexpr>></td></tr>
<tr><td>noexcept</td><td></td><td colspan="2"><<noexcept>></td></tr>
<tr><td>throw</td><td>exceptions</td><td><<throw >></td><td>specification</td></tr>
<tr><td colspan="2">Attribute</td><td colspan="2"><<attributes>></td></tr>
<tr><td rowspan="4">Vorlagenparameter</td><td>Name</td><td>Name</td><td rowspan="4">Vorlagenparameter</td></tr>
<tr><td>template parameter pack</td><td>parameterPack</td></tr>
<tr><td>type</td><td>property @type</td></tr>
<tr><td>default</td><td>default</td></tr>
<tr><td>Template specialization</td><td>Argumente</td><td>Parameter</td><td><<specialization>></td></tr>
<tr><td rowspan="5">Parameter</td><td>Name</td><td>Name</td><td rowspan="5">Parameter</td></tr>
<tr><td>Typ</td><td>Typ</td></tr>
<tr><td>type modifiers</td><td>type modifier</td></tr>
<tr><td>const</td><td><<const>></td></tr>
<tr><td>volatile</td><td><volatile>></td></tr>
</table>

C++				UModel			
			attributes	<<attributes>>			
			varArgList	varArgList			
			default value	default			
	Konstruktor	Name		Name		Operation <<constructor>>	
		Zugriffsmodifikator	public	Sichtbarkeit	public		
			protected		protected		
			private		private		
		explicit		<<explicit>>			
		= delete		<<delete>>			
		inline		<<inline>>			
		= default		<<default>>			
		noexcept		<<noexcept>>			
		throw	exceptions	<<throw >>	specification		
		attributes		<<attributes>>			
		Vorlagenparameter	Name	Name	Vorlagenparameter		
			template parameter pack	parameterPack			
			type	property @type			
			default	default			
		Template specialization	Argumente	Parameter	<<specialization>>		
		Parameter	Name	Name	Parameter		
			type	Typ			
			type modifiers	Typ-Modifier			
			const	<<const>>			
			volatile	<volatile>>			
			attributes	<<attributes>>			
			varArgList	varArgList			
			default value	default			
	Destruktor	Name		Name		Operation	

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="4">C++</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="19"></td><td rowspan="6"></td><td rowspan="3">Zugriffsmodifikator</td><td>public</td><td rowspan="3">Sichtbarkeit</td><td>public</td><td rowspan="6"><<destructor>></td><td rowspan="19"></td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td colspan="2">inline</td><td colspan="2"><<inline>></td></tr>
<tr><td colspan="2">noexcept</td><td colspan="2"><<noexcept>></td></tr>
<tr><td>throw</td><td>exceptions</td><td><<throw>></td><td>specification</td></tr>
<tr><td colspan="2">Attribute</td><td colspan="2"><<attributes>></td></tr>
<tr><td rowspan="13">Operator</td><td colspan="2">Name</td><td colspan="2">'operator' Name</td><td rowspan="13">Operation</td></tr>
<tr><td rowspan="3">Zugriffsmodifikator</td><td>public</td><td rowspan="3">Sichtbarkeit</td><td>public</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td colspan="2">static</td><td colspan="2">static</td></tr>
<tr><td colspan="2">virtual</td><td colspan="2"><<virtual>></td></tr>
<tr><td colspan="2">= 0</td><td colspan="2"><<purevirtual>></td></tr>
<tr><td colspan="2">const</td><td colspan="2"><<const>></td></tr>
<tr><td colspan="2">inline</td><td colspan="2"><<inline>></td></tr>
<tr><td colspan="2">= delete</td><td colspan="2"><<delete>></td></tr>
<tr><td colspan="2">= default</td><td colspan="2"><<default>></td></tr>
<tr><td colspan="2">override</td><td colspan="2"><<overrride>></td></tr>
<tr><td colspan="2">final</td><td colspan="2"><<final>></td></tr>
<tr><td colspan="2">volatile</td><td colspan="2"><<volatile>></td></tr>
<tr><td colspan="2">constexpr</td><td colspan="2"><<constexpr>></td></tr>
<tr><td colspan="2">noexcept</td><td colspan="2"><<noexcept>></td></tr>
<tr><td>throw</td><td>exceptions</td><td><<throw>></td><td>specification</td></tr>
<tr><td colspan="2">Attribute</td><td colspan="2"><<attributes>></td></tr>
<tr><td rowspan="4">Vorlagenparameter</td><td>Name</td><td>Name</td><td rowspan="4">Vorlagenparameter</td></tr>
<tr><td>template parameter pack</td><td>parameterPack</td></tr>
<tr><td>type</td><td>property @type</td></tr>
<tr><td>default</td><td>default</td></tr>
</table>

C++				UModel			
		Template specialization	arguments	Parameter	<<specialization>>		
		Parameter	Name	Name	Parameter		
			type	Typ			
			type modifiers	Typ-Modifier			
			const	<<const>>			
			volatile	<volatile>>			
			attributes	<<attributes>>			
			varArgList	varArgList			
			default value	default			

C++ Typedef

C++		UModel	
Typedef	Name	Name	Klasse <<typedef>>
	Dateiname	Codedateiname	
	verknüpfte Projektdatei / Verzeichnis	Komponentenrealisierung	
	Typ	@type property	
	Attribute	<<attributes>>	

C++ Type alias

C++			UModel		
Typalias	Name		Name		Klasse <<typealias>>
	Dateiname		Codedateiname		
	verknüpfte Projektdatei / Verzeichnis		Komponentenrealisierung		
	Typ		@type property		
	Attribute		<<attributes>>		
	Vorlagenparameter	Name	Name	Vorlagenparameter	
		template parameter pack	parameterPack		
		type	property @type		

© 2016-2022 Altova GmbH

C++			UModel		
		default	default		

C++ Enum

C++			UModel		
Enum	Name		Name		Enumeration
	Dateiname		Codedateiname		
	verknüpfte Projektdatei / Verzeichnis		Komponentenrealisierung		
	base type		<<basetype>> Wert		
	Attribute		<<attributes>>		
	Enumerator	Name	Name	Enumeration Literal	
		default value	default		
		attribute sections	<<attributes>>		

6.6.2 C#-Entsprechungen

In der Tabelle unten sehen Sie die 1:1-Entsprechungen zwischen:

- UModel-Elementen und C#-Codeelementen bei der Ausgabe von Modell in Code
- C#-Codeelementen und UModel-Modellelementen beim Import von Code in das Modell

C#-Projekt

C#		UModel	
Projekt	Projektdatei	Projektdatei	Komponente
	Verzeichnis	Verzeichnis	

C# Namespace

C#		UModel	
Namespace	Name	Name	Paket <<namespace>>

C#-Klasse

<table>
<tr><th colspan="5">C#</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="20">Klasse</td><td colspan="3">Name</td><td colspan="2">Name</td><td>Klasse</td></tr>
<tr><td rowspan="6">modifiers</td><td colspan="2">internal</td><td rowspan="5">Sichtbarkeit</td><td>Paket</td><td rowspan="6"></td></tr>
<tr><td colspan="2">protected internal</td><td>protected <<internal>></td></tr>
<tr><td colspan="2">public</td><td>public</td></tr>
<tr><td colspan="2">protected</td><td>protected</td></tr>
<tr><td colspan="2">private</td><td>private</td></tr>
<tr><td colspan="2">sealed</td><td colspan="2">leaf</td></tr>
<tr><td colspan="2">abstract</td><td colspan="2">abstract</td></tr>
<tr><td colspan="2">static</td><td colspan="2"><<static>></td></tr>
<tr><td colspan="2">unsafe</td><td colspan="2"><<unsafe>></td></tr>
<tr><td colspan="2">partial</td><td colspan="2"><<partial>></td></tr>
<tr><td colspan="2">new</td><td colspan="2"><<new >></td></tr>
<tr><td colspan="3">Dateiname</td><td colspan="2">Codedateiname</td></tr>
<tr><td colspan="3">verknüpfte Projektdatei / Verzeichnis</td><td colspan="2">Komponentenrealisierung</td></tr>
<tr><td colspan="3">base types</td><td colspan="2">Generalisierung, Schnittstellenrealisierung(en)</td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="2"><<attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="2">Kommentar(->Dokumentation)</td></tr>
<tr><td rowspan="11">Feld</td><td colspan="2">Name</td><td colspan="2">Name</td><td rowspan="11">Eigenschaft</td></tr>
<tr><td rowspan="5">modifiers</td><td>internal</td><td rowspan="5">Sichtbarkeit</td><td>Sichtbarkeit</td></tr>
<tr><td>protected internal</td><td>protected <<internal>></td></tr>
<tr><td>public</td><td>public</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td>static</td><td colspan="2">static</td></tr>
<tr><td>readonly</td><td colspan="2">readonly</td></tr>
<tr><td>volatile</td><td colspan="2"><<volatile>></td></tr>
<tr><td>unsafe</td><td colspan="2"><<unsafe>></td></tr>
<tr><td>new</td><td colspan="2"><<new >></td></tr>
</table>

© 2016-2022 Altova GmbH

C#				UModel			
		type		Typ			
		type dimensions		Multiplizität			
		type pointer		Typ-Modifier			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Kommentar(->Dokumentation)			
	Konstante	Name		Name		Eigenschaft <<const>>	
		modifiers	internal	Sichtbarkeit	Paket		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			new	<<new >>			
		type		Typ			
		type dimensions		Multiplizität			
		type pointer		Typ-Modifier			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Kommentar(->Dokumentation)			
	Methode	Name		Name		Operation	
		modifiers	internal	Sichtbarkeit	Paket		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			static	static			
			abstract	abstract			
			sealed	leaf			

C#					UModel				
			override		<<override>>				
			partial		<<partial>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		implemented interfaces			implements				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifiers	ref	direction	inout			
				out		out			
				params	varArgList				
			type		Typ				
			type dimensions		Multiplizität				
			type pointer		Typ-Modifier				
			this		<<this>>				
			nullable		<<nullable>>				
		Typparameter	Name		Name		Vorlagen-parameter		
			constraint		Einschränkender Classifier				
			predefined constraint	struct	<<ValueTypeConstraint>>				
				Klasse	<<ReferenceTypeConstraint>>				
				new ()	<<ConstructorConstraint>>				
			attribute sections		<<attributes>>				
	Konstruktor	Name			Name			Operation <<constructor>>	
		modifiers	internal		Sichtbarkeit	Paket			
			protected internal			protected <<internal>>			
			public			public			

© 2016-2022 Altova GmbH

C#					UModel			
			protected			protected		
			private			private		
			static		static			
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Kommentar(->Dokumentation)			
		Parameter	Name		Name		Parameter	
			modifiers	ref	direction	inout		
				out		out		
				params	varArgList			
			type		Typ			
			type dimensions		Multiplizität			
			type pointer		Typ-Modifier			
			nullable		<<nullable>>			
	Destruktor	Name			Name		Operation <<destructor>>	
		modifiers	private		Sichtbarkeit	private		
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Kommentar(->Dokumentation)			
	Eigenschaft	Name			Name		Operation <<property>>	
		modifiers	internal		Sichtbarkeit	Paket		
			protected internal			protected <<internal>>		
			public			public		
			protected			protected		
			private			private		
			static		static			
			abstract		abstract			
			sealed		leaf			
			override		<<override>>			

C#					UModel				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		type dimensions			Multiplizität				
		nullable			<<nullable>>				
		Get Accessor	modifiers	internal	Sichtbarkeit	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	Sichtbarkeit	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Operator	Name			Name			Operation <<operator>>	
		modifiers	public		Sichtbarkeit	public			
			static		static				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifier	params	varArgList				
			type		Typ				
			type dimensions		Multiplizität				
			type pointer		Typ-Modifier				

© 2016-2022 Altova GmbH

C#					UModel				
			nullable		<<nullable>>				
	Indexer	Name (="this")			Name (="this")			Operation <<indexer>>	
		modifiers	internal		Sichtbarkeit	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				
			sealed		leaf				
			override		<<override>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifier	params	varArgList				
			type		type				
			type dimensions		Multiplizität				
			type pointer		Typ-Modifier				
			nullable		<<nullable>>				
		Get Accessor	modifiers	internal	Sichtbarkeit	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	Sichtbarkeit	internal	<<SetAccessor>>		
				protected internal		protected internal			

C#					UModel				
				protected		protected			
				private		private			
	Event	Name			Name			Operation <<event>>	
		modifiers	internal		Sichtbarkeit	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				
			sealed		leaf				
			override		<<override>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		type dimensions			Multiplizität				
		nullable			<<nullable>>				
		Add Accessor			<<AddRemoveAccessor>>				
		Remove Accessor							
	Typparameter	Name			Name			Vorlagen-parameter	
		constraint			Einschränkender Classifier				
		predefined constraint	struct		<<ValueTypeConstraint>>				
			Klasse		<<ReferenceTypeConstraint>>				
			new ()		<<ConstructorConstraint>>				
		attribute sections			<<attributes>>				

© 2016-2022 Altova GmbH

C# Struct

C#				UModel			
Struct	Name			Name			Klasse <<struct>>
	modifiers	internal		Sichtbarkeit	package		
		protected internal			protected <<internal>>		
		public			public		
		protected			protected		
		private			private		
		unsafe		<<unsafe>>			
		partial		<<partial>>			
		new		<<new >>			
	Dateiname			Codedateiname			
	verknüpfte Projektdatei / Verzeichnis			Komponentenrealisierung			
	base types			InterfaceRealization(s)			
	attribute sections			<<attributes>>			
	doc comments			Kommentar(->Dokumentation)			
	Feld	Name		Name		Property	
		modifiers	internal	Sichtbarkeit	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			static	static			
			readonly	readonly			
			volatile	<<volatile>>			
			unsafe	<<unsafe>>			
			new	<<new >>			
		type		type			
		type dimensions		Multiplizität			
		type pointer		Typ-Modifier			

C#				UModel			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Kommentar(->Dokumentation)			
	Konstante	Name		Name		Eigenschaft <<const>>	
		modifiers	internal	Sichtbarkeit	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			new	<<new >>			
		type		Typ			
		type dimensions		Multiplizität			
		type pointer		Typ-Modifier			
		nullable		<<nullable>>			
		default value		default			
		attribute sections		<<attributes>>			
		doc comments		Kommentar(->Dokumentation)			
	Fixedsize Buffer	Name		Name		Eigenschaft <<fixed>>	
		modifiers	internal	Sichtbarkeit	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			unsafe	<<unsafe>>			
			new	<<new >>			
		type		Typ			
		type pointer		Typ-Modifier			
		nullable		<<nullable>>			
		buffer size		default			

© 2016-2022 Altova GmbH

C#					UModel				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
	Methode	Name			Name			Operation	
		modifiers	internal		Sichtbarkeit	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				
			sealed		leaf				
			override		<<override>>				
			partial		<<partial>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		implemented interfaces			implements				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifiers	ref	direction	inout			
				out		out			
				params	varArgList				
			type		type				
			type dimensions		Multiplizität				
			type pointer		Typ-Modifier				
			this		<<this>>				
			nullable		<<nullable>>				

C#					UModel			
		Typparameter	Name		Name	Vorlagen-parameter		
			constraint		Einschränkender Classifier			
			predefined constraint	struct	<<ValueTypeConstraint>>			
				Klasse	<<ReferenceTypeConstraint>>			
				new ()	<<ConstructorConstraint>>			
			attribute sections		<<attributes>>			
	Konstruktor	Name			Name		Operation <<constructor>>	
		modifiers	internal		Sichtbarkeit	package		
			protected internal			protected <<internal>>		
			public			public		
			protected			protected		
			private			private		
			static		static			
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Kommentar(->Dokumentation)			
		Parameter	Name		Name	Parameter		
			modifiers	ref	direction	inout		
				out		out		
				params	varArgList			
			type		Typ			
			type dimensions		Multiplizität			
			type pointer		Typ-Modifier			
			nullable		<<nullable>>			
	Destruktor	Name			Name		Operation <<destructor>>	
		modifiers	private		Sichtbarkeit	private		
			unsafe		<<unsafe>>			

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="5">C#</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="30"></td><td rowspan="2"></td><td colspan="2">attribute sections</td><td colspan="2"><<attributes>></td><td rowspan="2"></td><td rowspan="30"></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="2">Kommentar(->Dokumentation)</td></tr>
<tr><td rowspan="28">Eigenschaft</td><td colspan="2">Name</td><td colspan="2">Name</td><td rowspan="28">Operation
<<property>></td></tr>
<tr><td rowspan="8">modifiers</td><td>internal</td><td rowspan="5">Sichtbarkeit</td><td>package</td></tr>
<tr><td>protected internal</td><td>protected <<internal>></td></tr>
<tr><td>public</td><td>public</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td>static</td><td colspan="2">static</td></tr>
<tr><td>abstract</td><td colspan="2">abstract</td></tr>
<tr><td>sealed</td><td colspan="2">leaf</td></tr>
<tr><td>override</td><td colspan="2"><<override>></td></tr>
<tr><td>virtual</td><td colspan="2"><<virtual>></td></tr>
<tr><td>new</td><td colspan="2"><<new >></td></tr>
<tr><td>unsafe</td><td colspan="2"><<unsafe>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="2">Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="2">type</td><td>direction</td><td>return</td><td rowspan="3">Parameter</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">Multiplizität</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<nullable>></td></tr>
<tr><td rowspan="4">Get Accessor</td><td rowspan="4">modifiers</td><td>internal</td><td rowspan="4">Sichtbarkeit</td><td>internal</td><td rowspan="4"><<GetAccessor>></td></tr>
<tr><td>protected internal</td><td>protected internal</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
<tr><td rowspan="4">Set Accessor</td><td rowspan="4">modifiers</td><td>internal</td><td rowspan="4">Sichtbarkeit</td><td>internal</td><td rowspan="4"><<SetAccessor>></td></tr>
<tr><td>protected internal</td><td>protected internal</td></tr>
<tr><td>protected</td><td>protected</td></tr>
<tr><td>private</td><td>private</td></tr>
</table>

C#				UModel			
	Operator	Name		Name			Operation <<operator>>
		modifiers	public	Sichtbarkeit	public		
			static	static			
			unsafe	<<unsafe>>			
		attribute sections		<<attributes>>			
		doc comments		Kommentar(->Dokumentation)			
		type		direction	return	Parameter	
		Parameter	Name	Name			
			modifier params	varArgList			
			type	Typ			
			type dimensions	Multiplizität			
			type pointer	Typ-Modifier			
			nullable	<<nullable>>			
	Indexer	Name (="this")		Name (="this")			Operation <<indexer>>
		modifiers	internal	Sichtbarkeit	package		
			protected internal		protected <<internal>>		
			public		public		
			protected		protected		
			private		private		
			static	static			
			abstract	abstract			
			sealed	leaf			
			override	<<override>>			
			virtual	<<virtual>>			
			new	<<new >>			
			unsafe	<<unsafe>>			
		attribute sections		<<attributes>>			
		doc comments		Kommentar(->Dokumentation)			
		type		direction	return	Parameter	

© 2016-2022 Altova GmbH

C#					UModel				
		Parameter	Name		Name				
			modifier	params	varArgList				
			type		Typ				
			type dimensions		Multiplizität				
			type pointer		Typ-Modifier				
			nullable		<<nullable>>				
		Get Accessor	modifiers	internal	Sichtbarkeit	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	Sichtbarkeit	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Event	Name			Name			Operation <<event>>	
		modifiers	internal		Sichtbarkeit	package			
			protected internal			protected <<internal>>			
			public			public			
			protected			protected			
			private			private			
			static		static				
			abstract		abstract				
			sealed		leaf				
			override		<<override>>				
			virtual		<<virtual>>				
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				

C#				UModel				
		type		direction	return	Parameter		
		type dimensions		Multiplizität				
		nullable		<<nullable>>				
		Add Accessor		<<AddRemoveAccessor>>				
		Remove Accessor						
	Typparameter	Name		Name			Vorlagen-parameter	
		constraint		Einschränkender Classifier				
		predefined constraint	struct	<<ValueTypeConstraint>>				
			Klasse	<<ReferenceTypeConstraint>>				
			new ()	<<ConstructorConstraint>>				
		attribute sections		<<attributes>>				

C#-Schnittstelle

C#				UModel		
Schnittstelle	Name			Name		Schnittstelle
	modifiers	internal		Sichtbarkeit	package	
		protected internal			protected <<internal>>	
		public			public	
		protected			protected	
		private			private	
		unsafe		<<unsafe>>		
		partial		<<partial>>		
		new		<<new >>		
	Dateiname			Codedateiname		
	verknüpfte Projektdatei / Verzeichnis			Komponentenrealisierung		
	base types			Generalisierung(en)		
	attribute sections			<<attributes>>		
	doc comments			Kommentar(->Dokumentation)		

© 2016-2022 Altova GmbH

C#					UModel			
	Methode	Name			Name		Operation	
		modifiers	public		Sichtbarkeit	public		
			new		<<new >>			
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			
		doc comments			Kommentar(->Dokumentation)			
		type			direction	return	Parameter	
		Parameter	Name		Name			
			modifiers	ref	direction	inout		
				out		out		
				params	varArgList			
			type		Typ			
			type dimensions		Multiplizität			
			type pointer		Typ-Modifier			
			this		<<this>>			
			nullable		<<nullable>>			
		Typparameter	Name		Name		Vorlagen-parameter	
			constraint		Einschränkender Classifier			
			predefined constraint	struct	<<ValueTypeConstraint>>			
				Klasse	<<ReferenceTypeConstraint>>			
				new ()	<<ConstructorConstraint>>			
			attribute sections		<<attributes>>			
	Property	Name			Name		Operation <<property>>	
		modifiers	public		Sichtbarkeit	public		
			new		<<new >>			
			unsafe		<<unsafe>>			
		attribute sections			<<attributes>>			

C#					UModel				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		type dimensions			Multiplizität				
		nullable			<<nullable>>				
		Get Accessor	modifiers	internal	Sichtbarkeit	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
		Set Accessor	modifiers	internal	Sichtbarkeit	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Indexer	Name (="this")			Name (="this")			Operation <<indexer>>	
		modifiers	public		Sichtbarkeit	public			
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifier	params	varArgList				
			type		type				
			type dimensions		Multiplizität				
			type pointer		Typ-Modifier				
			nullable		<<nullable>>				
		Get Accessor	modifiers	internal	Sichtbarkeit	internal	<<GetAccessor>>		
				protected internal		protected internal			
				protected		protected			

© 2016-2022 Altova GmbH

C#					UModel				
				private		private			
		Set Accessor	modifiers	internal	Sichtbarkeit	internal	<<SetAccessor>>		
				protected internal		protected internal			
				protected		protected			
				private		private			
	Event	Name			Name			Operation <<event>>	
		modifiers	public		Sichtbarkeit	public			
			new		<<new >>				
			unsafe		<<unsafe>>				
		attribute sections			<<attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		type dimensions			Multiplizität				
		nullable			<<nullable>>				
		Add Accessor			<<AddRemoveAccessor>>				
		Remove Accessor							
	Typparameter	Name			Name			Vorlagen-parameter	
		constraint			Einschränkender Classifier				
		predefined constraint	struct		<<ValueTypeConstraint>>				
			Klasse		<<ReferenceTypeConstraint>>				
			new ()		<<ConstructorConstraint>>				
		attribute sections			<<attributes>>				

C#-Delegat

C#			UModel		
Delegat	Name		Name		Klasse <<delegate>>
	modifiers	internal	Sichtbarkeit	package	
		protected internal		protected <<internal>>	
		public		public	

C#				UModel			
	protected				protected		
	private				private		
	unsafe			<<unsafe>>			
	new			<<new >>			
Dateiname				Codedateiname			
verknüpfte Projektdatei / Verzeichnis				Komponentenrealisierung			
attribute sections				<<attributes>>			
doc comments				Kommentar(->Dokumentation)			
type				direction	return	Parameter	Operation
Parameter	Name			Name			
	modifiers	ref		direction	inout		
		out			out		
		params		varArgList			
	type			Typ			
	type dimensions			Multiplizität			
	type pointer			Typ-Modifier			
	nullable			<<nullable>>			
Typparameter	Name			Name			Vorlagen-parameter
	constraint			Einschränkender Classifier			
	predefined constraint	struct		<<ValueTypeConstraint>>			
		Klasse		<<ReferenceTypeConstraint>>			
		new ()		<<ConstructorConstraint>>			
	attribute sections			<<attributes>>			

© 2016-2022 Altova GmbH

C#-Enum

C#			UModel		
Enum	Name		Name		Enumeration
	modifiers	internal	Sichtbarkeit	package	
		protected internal		protected <<internal>>	
		public		public	
		protected		protected	
		private		private	
		new	<<new >>		
	Dateiname		Codedateiname		
	verknüpfte Projektdatei / Verzeichnis		Komponentenrealisierung		
	base type		Typ	<<BaseType>>	
	attribute sections		<<attributes>>		
	doc comments		Kommentar(->Dokumentation)		
	Enum Constant	Name	Name	Enumeration Literal	
		default value	default		
		attribute sections	<<attributes>>		
		doc comments	Kommentar(->Dokumentation)		

Parametrisierter C#-Typ

C#	UModel
Parametrisierter Typ	Anonymes gebundenes Element

6.6.3 VB.NET-Entsprechungen

In der Tabelle unten sehen Sie die 1:1-Entsprechungen zwischen:

- UModel-Elementen und VB.NET-Codeelementen bei der Ausgabe von Modell in Code
- VB.NET-Codeelementen und UModel-Modellelementen beim Import von Code in das Modell

VB.NET				UModel			
Projekt	Projektdatei			Projektdatei			Komponente
	Verzeichnis			Verzeichnis			
Namespace	Name			Name			Paket <<namespace>>
Klasse	Name			Name			Klasse
	modifiers	Friend		Sichtbarkeit	Paket		
		Protected Friend			protected <<Friend>>		
		Public			public		
		Protected			protected		
		Private			private		
		NotInheritable		leaf			
		MustInherit		abstract			
		Partial		<<Partial>>			
		Shadows		<<Shadows>>			
	Dateiname			Codedateiname			
	verknüpfte Projektdatei/Verzeichnis			Komponentenrealisierung			
	base types			Generalisierung, InterfaceRealization(s)			
	attribute sections			<<Attributes>>			
	doc comments			Kommentar(->Dokumentation)			
	Feld	Name		Name		Eigenschaft	
		modifiers	Friend	Sichtbarkeit	Paket		
			Protected Friend		protected <<Friend>>		
			Public		public		
			Protected		protected		
			Private		private		
			Shared	static			
			ReadOnly	readonly			
			Shadows	<<Shadows>>			
		type		Typ			
		type dimensions		Multiplizität			

© 2016-2022 Altova GmbH

VB.NET				UModel			
		nullable		<<Nullable>>			
		default value		default			
		attribute sections		<<Attributes>>			
		doc comments		Kommentar(->Dokumentation)			
	Konstante	Name		Name		Eigenschaft <<Const>>	
		modifiers	Friend	Sichtbarkeit	Paket		
			Protected Friend		protected <<Friend>>		
			Public		public		
			Protected		protected		
			Private		private		
			Shadows	<<Shadows>>			
		type		Typ			
		type dimensions		Multiplizität			
		nullable		<<Nullable>>			
		default value		default			
		attribute sections		<<Attributes>>			
		doc comments		Kommentar(->Dokumentation)			
	Methode	Name		Name		Operation	
		modifiers	Friend	Sichtbarkeit	Paket		
			Protected Friend		protected <<Friend>>		
			Public		public		
			Protected		protected		
			Private		private		
			Shared	static			
			MustOverride	abstract			
			NotOverridable	leaf			
			Overrides	<<Overrides>>			
			Overridable	<<Overridable>>			
			Partial	<<Partial>>			
			Shadows	<<Shadows>>			

VB.NET					UModel				
			Overloads		<<Overloads>>				
		attribute sections			<<Attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		implemented interfaces			implements				
		type (function)			direction	return	Parameter		
		Parameter	Name		Name				
			modifiers	ByRef	direction	inout			
				ByVal		in			
				ParamArray	varArgList				
				Optional	default				
			type		Typ				
			type dimensions		Multiplizität				
			nullable		<<Nullable>>				
		Typparameter	Name		Name		Vorlagen-parameter		
			constraint		Einschränkender Classifier				
			predefined constraint	Structure	<<ValueTypeConstraint>>				
				Klasse	<<ReferenceTypeConstraint>>				
				New	<<ConstructorConstraint>>				
			attribute sections		<<Attributes>>				
	Konstruktor	Name			Name			Operation <<Constructor>>	
		modifiers	Friend		Sichtbarkeit	Paket			
			Protected Friend			protected <<Friend>>			
			Public			public			
			Protected			protected			
			Private			private			
			Shared		static				
		attribute sections			<<Attributes>>				

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="5">VB.NET</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="13"></td><td rowspan="9"></td><td colspan="3">doc comments</td><td colspan="3">Kommentar(->Dokumentation)</td><td rowspan="9"></td><td rowspan="13"></td></tr>
<tr><td rowspan="8">Parameter</td><td colspan="2">Name</td><td colspan="2">Name</td><td rowspan="8">Parameter</td></tr>
<tr><td rowspan="4">modifiers</td><td>ByRef</td><td rowspan="2">direction</td><td>inout</td></tr>
<tr><td>ByVal</td><td>in</td></tr>
<tr><td>ParamArray</td><td colspan="2">varArgList</td></tr>
<tr><td>Optional</td><td colspan="2">default</td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">Multiplizität</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td rowspan="4">Eigenschaft</td><td colspan="3">Name</td><td colspan="3">Name</td><td rowspan="4">Operation
<<Property>></td></tr>
<tr><td rowspan="3">modifiers</td><td>Friend</td><td rowspan="3">Sichtbarkeit</td><td>Paket</td></tr>
<tr><td>Protected Friend</td><td>protected <<Friend>></td></tr>
<tr><td>Public</td><td>public</td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="3">Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="3">type</td><td>direction</td><td>return</td><td>Parameter</td></tr>
</table>

VB.NET					UModel				
		type dimensions			Multiplizität				
		nullable			<<Nullable>>				
		Get Accessor	modifiers	Friend	Sichtbarkeit	Friend	<<GetAccessor>>		
				Protected Friend		Protected Friend			
				Protected		Protected			
				Private		Private			
		Set Accessor	modifiers	Friend	Sichtbarkeit	Friend	<<SetAccessor>>		
				Protected Friend		Protected Friend			
				Protected		Protected			
				Private		Private			
	Operator	Name			Name			Operation <<Operator>>	
		modifiers	Public		Sichtbarkeit	Public			
			Shared		static				
			Narrowing		Name <= Narrowing				
			Widening		Name <= Widening				
		attribute sections			<<Attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifier	ByVal	direction	in			
			type		Typ				
			type dimensions		Multiplizität				
			nullable		<<Nullable>>				
	Event	Name			Name			Operation <<Event>>	
		modifiers	Friend		Sichtbarkeit	Paket			
			Protected Friend			protected <<Friend>>			
			Public			public			
			Protected			protected			

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="5">VB.NET</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="16"></td><td rowspan="15"></td><td rowspan="10"></td><td>Private</td><td></td><td>private</td><td rowspan="15"></td><td rowspan="16"></td></tr>
<tr><td>Shared</td><td colspan="2">static</td></tr>
<tr><td>MustOverride</td><td colspan="2">abstract</td></tr>
<tr><td>NotOverridable</td><td colspan="2">leaf</td></tr>
<tr><td>Overrides</td><td colspan="2"><<Overrides>></td></tr>
<tr><td>Overridable</td><td colspan="2"><<Overridable>></td></tr>
<tr><td>Shadows</td><td colspan="2"><<Shadows>></td></tr>
<tr><td>Overloads</td><td colspan="2"><<Overloads>></td></tr>
<tr><td rowspan="3">kind</td><td>without specifying a delegate type</td><td colspan="2"><<Event>> (Type <= Simple)</td></tr>
<tr><td>with specifying a delegate type</td><td colspan="2"><<Event>> (Type <= Regular)</td></tr>
<tr><td>with custom accessors</td><td colspan="2"><<Event>> (Type <= Custom)</td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<Attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td colspan="2">Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="2">type</td><td>direction</td><td>return</td><td rowspan="3">Parameter</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">Multiplizität</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td>Typparameter</td><td colspan="2">Name</td><td colspan="3">Name</td><td rowspan="5">Vorlagen-parameter</td></tr>
<tr><td colspan="2">constraint</td><td colspan="3">Einschränkender Classifier</td></tr>
<tr><td rowspan="3">predefined constraint</td><td>Structure</td><td colspan="3"><<ValueTypeConstraint>></td></tr>
<tr><td>Klasse</td><td colspan="3"><<ReferenceTypeConstraint>></td></tr>
<tr><td>New</td><td colspan="3"><<ConstructorConstraint>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
<tr><td rowspan="7">Struktur</td><td colspan="2">Name</td><td colspan="3">Name</td><td rowspan="7">Klasse <<Structure>></td></tr>
<tr><td rowspan="6">modifiers</td><td>Friend</td><td rowspan="6">Sichtbarkeit</td><td>Paket</td></tr>
<tr><td>Protected Friend</td><td>protected <<Friend>></td></tr>
<tr><td>Public</td><td>public</td></tr>
<tr><td>Protected</td><td>protected</td></tr>
<tr><td>Private</td><td>private</td></tr>
<tr><td>Partial</td><td colspan="2"><<Partial>></td></tr>
</table>

VB.NET			UModel		
	Shadows		<<Shadows>>		
Dateiname			Codedateiname		
verknüpfte Projektdatei/Verzeichnis			Komponentenrealisierung		
base types			InterfaceRealization(s)		
attribute sections			<<Attributes>>		
doc comments			Kommentar(->Dokumentation)		
Feld	Name		Name		Eigenschaft
	modifiers	Friend	Sichtbarkeit	Paket	
		Public		public	
		Private		private	
		Shared	static		
		ReadOnly	readonly		
		Shadows	<<Shadows>>		
	type		type		
	type dimensions		Multiplizität		
	nullable		<<Nullable>>		
	default value		default		
	attribute sections		<<Attributes>>		
	doc comments		Kommentar(->Dokumentation)		
Konstante	Name		Name		Eigenschaft <<Const>>
	modifiers	Friend	Sichtbarkeit	Paket	
		Public		public	
		Private		private	
		Shadows	<<Shadows>>		
	type		type		
	type dimensions		Multiplizität		
	nullable		<<Nullable>>		
	default value		default		
	attribute sections		<<Attributes>>		
	doc comments		Kommentar(->Dokumentation)		

© 2016-2022 Altova GmbH

VB.NET					UModel				
	Methode	Name			Name			Operation	
		modifiers	Friend		Sichtbarkeit	Paket			
			Public			public			
			Private			private			
			Shared		static				
			MustOverride		abstract				
			NotOverridable		leaf				
			Overrides		<<Overrides>>				
			Overridable		<<Overridable>>				
			Partial		<<Partial>>				
			Shadows		<<Shadows>>				
			Overloads		<<Overloads>>				
		attribute sections			<<Attributes>>				
		doc comments			Kommentar(->Dokumentation)				
		implemented interfaces			implements				
		type (function)			direction	return	Parameter		
		Parameter	Name		Name				
			modifiers	ByRef	direction	inout			
				ByVal		in			
				ParamArray	varArgList				
				Optional	default				
			type		Typ				
			type dimensions		Multiplizität				
			nullable		<<Nullable>>				
		Typparameter	Name		Name		Vorlagen-parameter		
			constraint		Einschränkender Classifier				
			predefined constraint	Structure	<<ValueTypeConstraint>>				
				Klasse	<<ReferenceTypeConstraint>>				

VB.NET					UModel			
				New	<<ConstructorConstraint>>			
			attribute sections		<<Attributes>>			
	Konstruktor	Name			Name			Operation <<Constructor>>
		modifiers	Friend		Sichtbarkeit	Paket		
			Public			public		
			Private			private		
			Shared		static			
		attribute sections			<<Attributes>>			
		doc comments			Kommentar(->Dokumentation)			
		Parameter	Name		Name		Parameter	
			modifiers	ByRef	direction	inout		
				ByVal		in		
				ParamArray	varArgList			
				Optional	default			
			type		type			
			type dimensions		Multiplizität			
			nullable		<<Nullable>>			
	Eigenschaft	Name			Name			Operation <<Property>>
		modifiers	Friend		Sichtbarkeit	Paket		
			Public			public		
			Private			private		
			Shared		static			
			Default		<<Property>> (Default <= IsDefault)			
			MustOverride		abstract			
			NotOverridable		leaf			
			Overrides		<<Overrides>>			
			Overridable		<<Overridable>>			
			Shadows		<<Shadows>>			
			Overloads		<<Overloads>>			

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="5">VB.NET</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="15"></td><td rowspan="11"></td><td rowspan="2"></td><td colspan="2">ReadOnly</td><td colspan="3"><<GetAccessor>> (without <<SetAccessor>>)</td><td rowspan="11"></td><td rowspan="15"></td></tr>
<tr><td colspan="2">WriteOnly</td><td colspan="3"><<SetAccessor>> (without <<GetAccessor>>)</td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="3">Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="3">type</td><td>direction</td><td>return</td><td rowspan="3">Parameter</td></tr>
<tr><td colspan="3">type dimensions</td><td colspan="2">Multiplizität</td></tr>
<tr><td colspan="3">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td rowspan="2">Get Accessor</td><td rowspan="2">modifiers</td><td>Friend</td><td rowspan="2">Sichtbarkeit</td><td>Friend</td><td rowspan="2"><<GetAccessor>></td></tr>
<tr><td>Private</td><td>Private</td></tr>
<tr><td rowspan="2">Set Accessor</td><td rowspan="2">modifiers</td><td>Friend</td><td rowspan="2">Sichtbarkeit</td><td>Friend</td><td rowspan="2"><<SetAccessor>></td></tr>
<tr><td>Private</td><td>Private</td></tr>
<tr><td rowspan="12">Operator</td><td colspan="3">Name</td><td colspan="3">Name</td><td rowspan="12">Operation <<Operator>></td></tr>
<tr><td rowspan="4">modifiers</td><td colspan="2">Public</td><td>Sichtbarkeit</td><td>Public</td></tr>
<tr><td colspan="2">Shared</td><td colspan="3">static</td></tr>
<tr><td colspan="2">Narrowing</td><td colspan="3">Name <= Narrowing</td></tr>
<tr><td colspan="2">Widening</td><td colspan="3">Name <= Widening</td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="3">Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="3">type</td><td>direction</td><td>return</td><td rowspan="6">Parameter</td></tr>
<tr><td rowspan="5">Parameter</td><td colspan="2">Name</td><td colspan="2">Name</td></tr>
<tr><td>modifier</td><td>ByVal</td><td>direction</td><td>in</td></tr>
<tr><td colspan="2">type</td><td colspan="2">type</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">Multiplizität</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td>Event</td><td colspan="3">Name</td><td colspan="3">Name</td><td>Operation <<Event>></td></tr>
<tr><td rowspan="3">modifiers</td><td colspan="2">Friend</td><td rowspan="3">Sichtbarkeit</td><td>Paket</td></tr>
<tr><td colspan="2">Public</td><td>public</td></tr>
<tr><td colspan="2">Private</td><td>private</td></tr>
</table>

<table>
<tr><th colspan="5">VB.NET</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="16"></td><td rowspan="14"></td><td rowspan="8"></td><td>Shared</td><td>static</td><td rowspan="14"></td><td rowspan="16"></td></tr>
<tr><td>MustOverride</td><td>abstract</td></tr>
<tr><td>NotOverridable</td><td>leaf</td></tr>
<tr><td>Overrides</td><td><<Overrides>></td></tr>
<tr><td>Overridable</td><td><<Overridable>></td></tr>
<tr><td>Shadows</td><td><<Shadows>></td></tr>
<tr><td>Overloads</td><td><<Overloads>></td></tr>
<tr><td rowspan="3">kind</td><td>without specifying a delegate type</td><td><<Event>> (Type <= Simple)</td></tr>
<tr><td>with specifying a delegate type</td><td><<Event>> (Type <= Regular)</td></tr>
<tr><td>with custom accessors</td><td><<Event>> (Type <= Custom)</td></tr>
<tr><td colspan="2">attribute sections</td><td><<Attributes>></td></tr>
<tr><td colspan="2">doc comments</td><td>Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="2">type</td><td>direction</td><td>return</td><td>Parameter</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="3">Multiplizität</td></tr>
<tr><td colspan="2">nullable</td><td colspan="3"><<Nullable>></td></tr>
<tr><td>Typparameter</td><td colspan="2">Name</td><td colspan="3">Name</td><td>Vorlagen-parameter</td></tr>
<tr><td colspan="2">constraint</td><td colspan="3">Einschränkender Classifier</td></tr>
<tr><td rowspan="3">predefined constraint</td><td>Structure</td><td colspan="3"><<ValueTypeConstraint>></td></tr>
<tr><td>Klasse</td><td colspan="3"><<ReferenceTypeConstraint>></td></tr>
<tr><td>New</td><td colspan="3"><<ConstructorConstraint>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="3"><<Attributes>></td></tr>
<tr><td rowspan="8">Schnittstelle</td><td colspan="3">Name</td><td colspan="3">Name</td><td rowspan="8">Schnittstelle</td></tr>
<tr><td rowspan="6">modifiers</td><td colspan="2" rowspan="5">Friend</td><td rowspan="6">Sichtbarkeit</td><td rowspan="5">Paket</td></tr>
<tr><td colspan="2">Protected Friend</td><td>protected <<Friend>></td></tr>
<tr><td colspan="2">Public</td><td>public</td></tr>
<tr><td colspan="2">Protected</td><td>protected</td></tr>
<tr><td colspan="2">Private</td><td>private</td></tr>
<tr><td colspan="2">Shadows</td><td colspan="2"><<Shadows>></td></tr>
<tr><td colspan="3">Dateiname</td><td colspan="3">Codedateiname</td></tr>
</table>

© 2016-2022 Altova GmbH

<table>
<tr><th colspan="5">VB.NET</th><th colspan="5">UModel</th></tr>
<tr><td rowspan="30"></td><td colspan="4">verknüpfte Projektdatei/Verzeichnis</td><td colspan="4">Komponentenrealisierung</td><td rowspan="30"></td></tr>
<tr><td colspan="4">base types</td><td colspan="4">Generalisierung(en)</td></tr>
<tr><td colspan="4">attribute sections</td><td colspan="4"><<Attributes>></td></tr>
<tr><td colspan="4">doc comments</td><td colspan="4">Kommentar(->Dokumentation)</td></tr>
<tr><td rowspan="13">Methode</td><td colspan="3">Name</td><td colspan="2">Name</td><td rowspan="13">Operation</td></tr>
<tr><td rowspan="2">modifiers</td><td colspan="2">Public</td><td>Sichtbarkeit</td><td>public</td></tr>
<tr><td colspan="2">Shadows</td><td colspan="2"><<Shadows>></td></tr>
<tr><td colspan="3">attribute sections</td><td colspan="2"><<Attributes>></td></tr>
<tr><td colspan="3">doc comments</td><td colspan="2">Kommentar(->Dokumentation)</td></tr>
<tr><td colspan="3">type (function)</td><td>direction</td><td>return</td><td rowspan="9">Parameter</td></tr>
<tr><td>Parameter</td><td colspan="2">Name</td><td colspan="2">Name</td></tr>
<tr><td rowspan="3">modifiers</td><td>ByRef</td><td rowspan="2">direction</td><td>inout</td></tr>
<tr><td>ByVal</td><td>in</td></tr>
<tr><td>ParamArray</td><td colspan="2">varArgList</td></tr>
<tr><td>Optional</td><td colspan="2">default</td></tr>
<tr><td colspan="2">type</td><td colspan="2">Typ</td></tr>
<tr><td colspan="2">type dimensions</td><td colspan="2">Multiplizität</td></tr>
<tr><td colspan="2">nullable</td><td colspan="2"><<Nullable>></td></tr>
<tr><td rowspan="6">Typparameter</td><td colspan="2">Name</td><td colspan="2">Name</td><td rowspan="6">Vorlagen-parameter</td></tr>
<tr><td colspan="2">constraint</td><td colspan="2">Einschränkender Classifier</td></tr>
<tr><td rowspan="3">predefined constraint</td><td>Structure</td><td colspan="2"><<ValueTypeConstraint>></td></tr>
<tr><td>Klasse</td><td colspan="2"><<ReferenceTypeConstraint>></td></tr>
<tr><td>New</td><td colspan="2"><<ConstructorConstraint>></td></tr>
<tr><td colspan="2">attribute sections</td><td colspan="2"><<Attributes>></td></tr>
<tr><td rowspan="2">Eigenschaft</td><td colspan="3">Name</td><td colspan="2">Name</td><td rowspan="2">Operation
<<Property>></td></tr>
<tr><td>modifiers</td><td colspan="2">Public</td><td>Sichtbarkeit</td><td>public</td></tr>
</table>

VB.NET				UModel				
			Default	<<Property>> (Default <= IsDefault)				
			Shadows	<<Shadows>>				
			ReadOnly	<<GetAccessor>> (without <<SetAccessor>>)				
			WriteOnly	<<SetAccessor>> (without <<GetAccessor>>)				
		attribute sections		<<Attributes>>				
		doc comments		Kommentar(->Dokumentation)				
		type		direction	return	Parameter		
		type dimensions		Multiplizität				
		nullable		<<Nullable>>				
	Event	Name		Name			Operation <<Event>>	
		modifiers	Public	Sichtbarkeit	public			
			Shadows	<<Shadows>>				
		kind	without specifying a delegate type	<<Event>> (Type <= Simple)				
			with specifying a delegate type	<<Event>> (Type <= Regular)				
		attribute sections		<<Attributes>>				
		doc comments		Kommentar(->Dokumentation)				
		type		direction	return	Parameter		
		type dimensions		Multiplizität				
		nullable		<<Nullable>>				
	Typparameter	Name		Name			Vorlagen-parameter	
		constraint		Einschränkender Classifier				
		predefined constraint	Structure	<<ValueTypeConstraint>>				
			Klasse	<<ReferenceTypeConstraint>>				
			New	<<ConstructorConstraint>>				
		attribute sections		<<Attributes>>				
Delegat	Name			Name				Klasse <<Delegate>>
	modifiers	Friend		Sichtbarkeit	Paket			

© 2016-2022 Altova GmbH

VB.NET				UModel				
		Protected Friend			protected <<Friend>>			
		Public			public			
		Protected			protected			
		Private			private			
		Shadows		<<Shadows>>				
	Dateiname			Codedateiname				
	verknüpfte Projektdatei/Verzeichnis			Komponentenrealisierung				
	attribute sections			<<Attributes>>				
	doc comments			Kommentar(->Dokumentation)				
	type			direction	return	Parameter	Operation	
	Parameter	Name		Name				
		modifiers	ByRef	direction	inout			
			ByVal		in			
		type		type				
		type dimensions		Multiplizität				
		nullable		<<Nullable>>				
	Typparameter	Name		Name			Vorlagen-parameter	
		constraint		Einschränkender Classifier				
		predefined constraint	struct	<<ValueTypeConstraint>>				
			Klasse	<<ReferenceTypeConstraint>>				
			new()	<<ConstructorConstraint>>				
		attribute sections		<<Attributes>>				
Enum	Name			Name				Enumeration
	modifiers	Friend		Sichtbarkeit	Paket			
		Protected Friend			protected <<Friend>>			
		Public			public			
		Protected			protected			
		Private			private			
		Shadows		<<Shadows>>				
	Dateiname			Codedateiname				

VB.NET			UModel		
	verknüpfte Projektdatei/Verzeichnis		Komponentenrealisierung		
	base type		Typ	<<BaseType>>	
	attribute sections		<<Attributes>>		
	doc comments		Kommentar(->Dokumentation)		
	Enum Constant	Name	Name	Enumeration Literal	
		default value	default		
		attribute sections doc comments	<<Attributes>>		
			Kommentar(->Dokumentation)		
Parametrisierter Typ			Anonymes gebundenes Element		

6.6.4 Java-Entsprechungen

In der Tabelle unten sehen Sie die 1:1-Entsprechungen zwischen:

- UModel-Elementen und Java-Codeelementen bei der Ausgabe von Modell in Code
- Java-Codeelementen und UModel-Modellelementen beim Import von Code in das Modell

Java			UModel		
Projekt	Projektdatei		Projektdatei		Komponente
	Verzeichnis		Verzeichnis		
Paket	Name		Name		Paket <<namespace>>
Klasse	Name		Name		Klasse
	modifiers	Paket	visibility	Paket	
		public		public	
		protected		protected	
		private		private	
		abstract	abstract		
		strictfp	<<strictfp>>		
		final	<<final>>		
	Dateiname		Codedateiname		

© 2016-2022 Altova GmbH

Java			UModel		
verknüpfte Projektdatei/Verzeichnis			Komponentenrealisierung		
extends clause			Generalization		
implements clause			Schnittstellenrealisierung(en)		
java docs			Kommentar(->Dokumentation)		
Feld	Name		Name		Eigenschaft
	modifiers	Paket	Sichtbarkeit	Paket	
		public		public	
		protected		protected	
		private		private	
		static	static		
		transient	<<transient>>		
		volatile	<<volatile>>		
		final	<<final>>		
	type		Typ		
	type dimensions		Multiplizität		
	default value		default		
	java docs		Kommentar(->Dokumentation)		
Methode	Name		Name		Operation
	modifiers	Paket	Sichtbarkeit	Paket	
		public		public	
		protected		protected	
		private		private	
		static	static		
		abstract	abstract		
		final	<<final>>		
		native	<<native>>		
		strictfp	<<strictfp>>		
		synchronized	<<synchronized>>		
	throws clause		Ausnahmeereignisse		
	java docs		Kommentar(->Dokumentation)		

Java					UModel				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifier	final	<<final>>				
			...		varArgList				
			type		Typ				
			type dimensions		Multiplizität				
		Typparameter	Name		Name		Vorlagen-parameter		
			bound		Einschränkender Classifier				
	Konstruktor	Name			Name			Operation <<constructor>>	
		modifiers	public		Sichtbarkeit	public			
			protected			protected			
			private			private			
		throws clause			Ausnahmeereignisse				
		java docs			Kommentar(->Dokumentation)				
		Parameter	Name		Name		Parameter		
			modifier	final	<<final>>				
			...		varArgList				
			type		Typ				
			type dimensions		Multiplizität				
		Typparameter	Name		Name		Vorlagen-parameter		
			bound		Einschränkender Classifier				
	Typparameter	Name			Name			Vorlagen-parameter	
		bound			Einschränkender Classifier				
Schnittstelle	Name				Name				Schnittstelle
	modifiers	Paket			Sichtbarkeit	Paket			
		public				public			
		protected				protected			
		private				private			
		abstract			abstract				

© 2016-2022 Altova GmbH

Java				UModel			
	strictfp			<<strictfp>>			
Dateiname				Codedateiname			
verknüpfte Projektdatei/Verzeichnis				Komponentenrealisierung			
extends clause				Generalization(s)			
java docs				Kommentar(->Dokumentation)			
Feld	Name			Name			Eigenschaft
	modifiers	public		Sichtbarkeit	public		
		static		static			
		final		<<final>>			
	type			type			
	type dimensions			Multiplizität			
	default value			default			
	java docs			Kommentar(->Dokumentation)			
Methode	Name			Name			Operation
	modifiers	public		Sichtbarkeit	public		
		abstract		abstract			
	throws clause			Ausnahmeereignisse			
	java docs			Kommentar(->Dokumentation)			
	type			direction	return	Parameter	
	Parameter	Name		Name			
		modifier	final	<<final>>			
		...		varArgList			
		type		Typ			
		type dimensions		Multiplizität			
	Typparameter	Name		Name		Vorlagen-parameter	
		bound		Einschränkender Classifier			
Typparameter	Name			Name			Vorlagen-parameter
	bound			Einschränkender Classifier			

Java				UModel			
Enum	Name			Name			Enumeration
	modifiers	Paket		Sichtbarkeit	Paket		
		public			public		
		protected			protected		
		private			private		
	Dateiname			Codedateiname			
	verknüpfte Projektdatei/Verzeichnis			Komponentenrealisierung			
	java docs			Kommentar(->Dokumentation)			
	Enum Constant	Name		Name		Enumerations-Literal	
	Feld	Name		Name		Eigenschaft	
		modifiers	Paket	Sichtbarkeit	Paket		
			public		public		
			protected		protected		
			private		private		
			static	static			
			transient	<<transient>>			
			volatile	<<volatile>>			
			final	<<final>>			
		type		Typ			
		type dimensions		Multiplizität			
		default value		default			
		java docs		Kommentar(->Dokumentation)			
	Methode	Name		Name		Operation	
		modifiers	Paket	Sichtbarkeit	Paket		
			public		public		
			protected		protected		
			private		private		
			static	static			
			abstract	abstract			

© 2016-2022 Altova GmbH

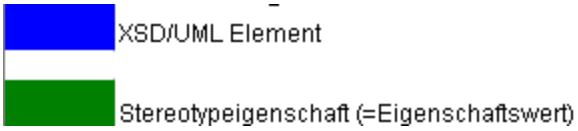
Java					UModel				
			final		<<final>>				
			native		<<native>>				
			strictfp		<<strictfp>>				
			synchronized		<<synchronized>>				
		throws clause			Ausnahmeereignisse				
		java docs			Kommentar(->Dokumentation)				
		type			direction	return	Parameter		
		Parameter	Name		Name				
			modifier	final	<<final>>				
			...		varArgList				
			type		Typ				
			type dimensions		Multiplizität				
		Typparameter	Name		Name		Vorlagen-parameter		
			bound		Einschränkender Classifier				
	Constructor	Name			Name			Operation <<constructor>>	
		modifiers	public		Sichtbarkeit	public			
			protected			protected			
			private			private			
		throws clause			Ausnahmeereignisse				
		java docs			Kommentar(->Dokumentation)				
		Parameter	Name		Name		Parameter		
			modifier	final	<<final>>				
			...		varArgList				
			type		Typ				
			type dimensions		Multiplizität				
		Typparameter	Name		Name		Vorlagen-parameter		
			bound		Einschränkender Classifier				
Parametrisierter Typ					Anonymes gebundenes Element				
Annotation					<<annotations> modifiers				

6.6.5 XML Schema-Entsprechungen

In der Tabelle unten sehen Sie die 1:1-Entsprechungen zwischen:

- UModel-Elementen und XML-Schema-Elementen bei der Ausgabe von Modell in Code
- XML-Schema-Elementen und UModel-Modellelementen beim Import von Code in das Modell

Legende:



XSD					UModel			
Dateipfad					Projektdatei			Komponente
Schema	Ziel-Namespace				Name			Paket <<namespace>>
	attributeFormDefault				attributeFormDefault			Klasse <<schema>>
	blockDefault				blockDefault			
	elementFormDefault				elementFormDefault			
	finalDefault				finalDefault			
	version				version			
	xml:lang				xml:lang			
	xmlns				xmlns			
	Annotation	source			source			
		appinfo					Kommentar <<appinfo>>	
		documentation	xml:lang		xml:lang		Kommentar <<documentation>>	
	attributeGroup	Name			Name			Klasse <<attributeGroup>>
		annotation	appinfo			Kommentar <<appinfo>>		

© 2016-2022 Altova GmbH

XSD					UModel			
			documentation			Kommentar <<documentation>>		
		attribute	Name		Name	Eigenschaft <<attribute>>		
			form		form			
			use		use			
			ref		type			
			type					
			default		default			
			fixed			fixed		
		attributeGroup	ref		type	Eigenschaft <<attributeGroup>>		
		anyAttribute	namespace		namespace	Eigenschaft <<anyAttribute>>		
			processContents		processContents			
	Attribut	Name			Name		Klasse <<attribute>>	
		form			form			
		use			use			
		type			type	Eigenschaft		
		default			default			
		fixed				fixed		
		annotation	appinfo			Kommentar <<appinfo>>		
			documentation			Kommentar <<documentation>>		
		simpleType			Name (= Name der Klasse + "_anonymousType[n]")	DataType <<simpleType>>		
	Element	Name			Name		Klasse <<element>>	
		abstract			abstract			

XSD					UModel				
		block			block				
		final			final				
		form			form				
		nillable			nillable				
		type			type		Eigenschaft		
		default			default				
		fixed				fixed			
		substitutionGroup			general		Generalisierung <<substitution>>		
		Annotation	appinfo				Kommentar <<appinfo>>		
			documentation				Kommentar <<documentation>>		
		simpleType			Name (= Name der Klasse + "_anonymousType[n]")		DataType <<simpleType>>		
		complexType			Name (= name of Class + "_anonymousType[n]")		Klasse <<complexType>>		
	group	Name			Name			Klasse <<group>>	
		annotation	appinfo				Kommentar <<appinfo>>		
			documentation				Kommentar <<documentation>>		
		all			Name (= "_all")		Eigenschaft		
					Name (= "mg"_ + "all")		Klasse <<all>>		
			annotation	appinfo		Kommentar <<appinfo>>			

© 2016-2022 Altova GmbH

XSD					UModel				
				documentation		Kommentar <<documentation>>			
			element	Name	Name	Eigenschaft <<element>>			
				ref	type				
				type					
		choice			Name (= "_choice")		Eigenschaft		
					Name (= "mg"_ + "choice")		Klasse <<choice>>		
			annotation	appinfo		Kommentar <<appinfo>>			
				documentation		Kommentar <<documentation>>			
			element	Name	Name	Eigenschaft <<element>>			
				ref	type				
				type					
			group			Eigenschaft <<group>>			
			any	namespace	namespace	Eigenschaft <<any>>			
				processContents	processContents				
			choice			Eigenschaft			
						Klasse <<choice>>			
			sequence			Eigenschaft			
						Klasse <<sequence>>			

<table>
<tr><th colspan="5">XSD</th><th colspan="4">UModel</th></tr>
<tr><td rowspan="11"></td><td rowspan="10"></td><td rowspan="10">sequence</td><td colspan="2"></td><td>Name (= "_sequence")</td><td>Eigenschaft</td><td rowspan="10"></td><td rowspan="11"></td></tr>
<tr><td colspan="2"></td><td>Name (= "mg"_ + "sequence")</td><td rowspan="9">Klasse <<sequence>></td></tr>
<tr><td rowspan="2">annotation</td><td>appinfo</td><td></td><td>Kommentar <<appinfo>></td></tr>
<tr><td>documentation</td><td></td><td>Kommentar <<documentation>></td></tr>
<tr><td rowspan="3">element</td><td>Name</td><td>Name</td><td rowspan="3">Eigenschaft <<element>></td></tr>
<tr><td>ref</td><td rowspan="2">type</td></tr>
<tr><td>type</td></tr>
<tr><td>group</td><td></td><td></td><td>Eigenschaft <<group>></td></tr>
<tr><td rowspan="2">any</td><td>namespace</td><td>namespace</td><td rowspan="2">Eigenschaft <<any>></td></tr>
<tr><td>processContents</td><td>processContents</td></tr>
<tr><td>choice</td><td></td><td></td><td>Eigenschaft</td></tr>
<tr><td></td><td></td><td></td><td>Klasse <<choice>></td></tr>
<tr><td>sequence</td><td></td><td></td><td>Eigenschaft</td></tr>
<tr><td></td><td></td><td></td><td>Klasse <<sequence>></td></tr>
<tr><td rowspan="4">notation</td><td colspan="2">Name</td><td colspan="3">Name</td><td rowspan="4">DataType <<notation>></td></tr>
<tr><td colspan="2">system</td><td colspan="3">system</td></tr>
<tr><td colspan="2">public</td><td colspan="3">public</td></tr>
<tr><td>annotation</td><td>appinfo</td><td></td><td></td><td></td><td>Kommentar <<appinfo>></td></tr>
</table>

© 2016-2022 Altova GmbH

XSD					UModel				
			documentation				Kommentar <<documentation>>		
	complexType	Name			Name			Klasse <<complexType>>	
		abstract			abstract				
		block			block				
		final			final				
		mixed			mixed				
		annotation	source		source				
			appinfo				Kommentar <<appinfo>>		
			documentation	xml:lang	xml:lang		Kommentar <<documentation>>		
		group			Name (= "_ref[n]")		Eigenschaft <<group>>		
			maxOccurs		Multiplizität				
			minOccurs						
			ref		type				
		all			Name (= "mg"_ + "all")		Klasse <<all>>		
					Name (= "_all")		Eigenschaft		
			maxOccurs		Multiplizität				
			minOccurs						
		choice			Name (= "mg"_ + "choice[n]")		Klasse <<choice>>		
					Name (= "_choice[n]")		Eigenschaft		
			maxOccurs		Multiplizität				
			minOccurs						

XSD					UModel				
		sequence			Name (= "mg"_ + "sequence[n]")		Klasse <<sequence>>		
					Name (= "_sequence[n]")		Eigenschaft		
			maxOccurs		Multiplizität				
			minOccurs						
		attribute	Name		Name		Eigenschaft <<attribute>>		
			ref		type				
			type						
		attributeGroup	ref		type		Eigenschaft <<attributeGroup>>		
		anyAttribute	namespace		namespace		Eigenschaft <<anyAttribute>>		
			processContents		processContents				
		complexContent	restriction	base	general		Generalisierung <<restriction>>		
			extension				Generalisierung <<extension>>		
	simpleType	Name			Name			DataType <<simpleType>> Enumeration <<simpleType>>	
		final			final				
		annotation	source		source				
			appinfo				Kommentar <<appinfo>>		
			documentation	xml:lang	xml:lang		Kommentar <<documentation>>		
		list	itemType		Name (= "_itemType")	Eigenschaft <<itemType>>	<<list>>		

© 2016-2022 Altova GmbH

XSD					UModel			
			simpleType		DataType <<simpleType>>			
		union	memberTypes		Name (= "memberType[n]")	Eigenschaft <<memberType>>	<<union>>	
			simpleType		DataType <<simpleType>>			
		minExclusive	Wert		Wert		<<minExclusive>>	
			fixed		fixed			
		minInclusive	Wert		Wert		<<minInclusive>>	
			fixed		fixed			
		maxExclusive	Wert		Wert		<<maxExclusive>>	
			fixed		fixed			
		maxInclusive	Wert		Wert		<<maxInclusive>>	
			fixed		fixed			
		totalDigits	Wert		Wert		<<totalDigits>>	
			fixed		fixed			
		fractionDigits	Wert		Wert		<<fractionDigits>>	
			fixed		fixed			
		length	Wert		Wert		<<length>>	
			fixed		fixed			
		minLength	Wert		Wert		<<minLength>>	
			fixed		fixed			
		maxLength	Wert		Wert		<<maxLength>>	
			fixed		fixed			
		whitespace	Wert		Wert		<<whitespace>>	
			fixed		fixed			
		pattern	Wert		Wert		<<whitespace>>	
		enumeration	Wert		Name		Enumerations-literal	
		simpleType					DataType <<simpleT	

XSD					UModel				
							ype>>		
		restriction	base		general		Generalisierung <<restriction>>		
	complexType simpleContent	Name			Name			DataType <<complexType>> <<simpleContent>>	
		annotation	source		source				
			appinfo				Kommentar <<appinfo>>		
			documentation	xml:lang	xml:lang		Kommentar <<documentation>>		
		minExclusive	Wert		Wert		<<minExclusive>>		
			fixed		fixed				
		minInclusive	Wert		Wert		<<minInclusive>>		
			fixed		fixed				
		maxExclusive	Wert		Wert		<<maxExclusive>>		
			fixed		fixed				
		maxInclusive	Wert		Wert		<<maxInclusive>>		
			fixed		fixed				
		totalDigits	Wert		Wert		<<totalDigits>>		
			fixed		fixed				
		fractionDigits	Wert		Wert		<<fractionDigits>>		
			fixed		fixed				
		length	Wert		Wert		<<length>>		
			fixed		fixed				
		minLength	Wert		Wert		<<minLength>>		
			fixed		fixed				
		maxLength	Wert		Wert		<<maxLength>>		
			fixed		fixed				

© 2016-2022 Altova GmbH

XSD					UModel				
		whitespace	Wert		Wert		<<whitespace>>		
			fixed		fixed				
		pattern	Wert		Wert		<<whitespace>>		
		attribute	Name		Name		Eigenschaft <<attribute>>		
			ref		type				
			type						
		attributeGroup	ref		type		Eigenschaft <<attributeGroup>>		
		anyAttribute	namespace		namespace		Eigenschaft <<anyAttribute>>		
			processContents		processContents				
		simpleType					DataType <<simpleType>>		
		restriction	base		general		Generalisierung <<restriction>>		
		extension	base		general		Generalisierung <<extension>>		
	import	schemaLocation			schemaLocation			ElementImport <<import>>	
		namespace			namespace				
	include	schemaLocation			schemaLocation			ElementImport <<include>>	
	redefine	schemaLocation			schemaLocation			ElementImport <<redefine>>	
		simpleType			<<redefine>>			DataType <<simpleType>>	

XSD				UModel		
		complexType			Klasse <<complexType>>	
		attributeGroup			Klasse <<attributeGroup>>	
		group			Klasse <<group>>	

6.6.6 Datenbank-Entsprechungen

In der Tabelle unten sehen Sie die 1:1-Entsprechungen zwischen:

- UModel-Elementen und Datenbankelementen bei der Ausgabe von Modell in Code
- Datenbankelementen und UModel-Modellelementen beim Import von Code in das Modell

Datenbank					UModel				
Datenbank	Verbindung				Verbindung				Komponente
Datenbank	Name				Name				Paket <<name space>> <<Database>>
	Schema	Name			Name			Paket <<name space>> <<Schema>>	
		Tabelle	Name		Name		Klasse <<Table>>		
			Spalte	Name	Name	Eigenschaft			
				Data Type	type				
				Not Null	<<not_null>>				
				Null	<<nullable>>				
				Length	Multiplizität				
				Precision					
				Scale					
				Default	default				
				Autoincrement	<<autoincrement>>				
				Primärschlüsselteil	<<PK>>				
				Sekundärschlüsselteil	<<FK>>				

© 2016-2022 Altova GmbH

Datenbank						UModel				
				Teil des eindeutigen Schlüssels		<<unique>>				
			Primär-schlüssel	Name		Name		Klasse <<PrimaryKey>>		
				Spalte	Name	Name	Eigenschaft			
			Sekundär-schlüssel	Name		Name		Klasse <<ForeignKey>>		
				Spalte	Name	Name	Eigenschaft			
				Sekundär-spalte	Name	Name	Eigenschaft			
					Sekundär-tabelle	type				
			Eindeutiger Schlüssel	Name		Name		Klasse <<UniqueKey>>		
				Spalte	Name	Name	Eigenschaft			
			Index	Name		Name		Klasse <<Index>>		
				Spalte	Name		Eigenschaft			
					Reihenfolge: aufsteigend	<<ascending>>				
					Reihenfolge: absteigend	<<descending>>				
			CheckConstraint	Name		Name		Klasse <<CheckConstraint>>		
				definition		definition				
		Ansicht	Name			Name		Klasse <<View>>		
			definition			definition				
			Spalte	Name		Name		Eigenschaft		
				Datentyp		type				
				Not Null		<<not_null>>				
				Null		<<nullable>>				
				Length		Multiplizität				

Datenbank					UModel				
				Precision					
				Scale					
				Default	default				
				Autoincrement	<<autoincrement>>				
		Gespeicherte Prozedur	Name		Name		Operation <<StoredProcedure>>	Klasse <<StoredProcedures>>	
			definition		definition				
			Parameter	Name	Name	Parameter			
				Richtungsmodus	direction				
				Datentyp	type				
		Funktion	Name		Name		Operation <<Function>>	Klasse <<Functions>>	
			definition		definition				
			Parameter	Name	Name	Parameter			
				Richtungsmodus	direction				
				Datentyp	type				
		Trigger	Name		Name			Klasse <<Trigger>>	
			definition		definition				

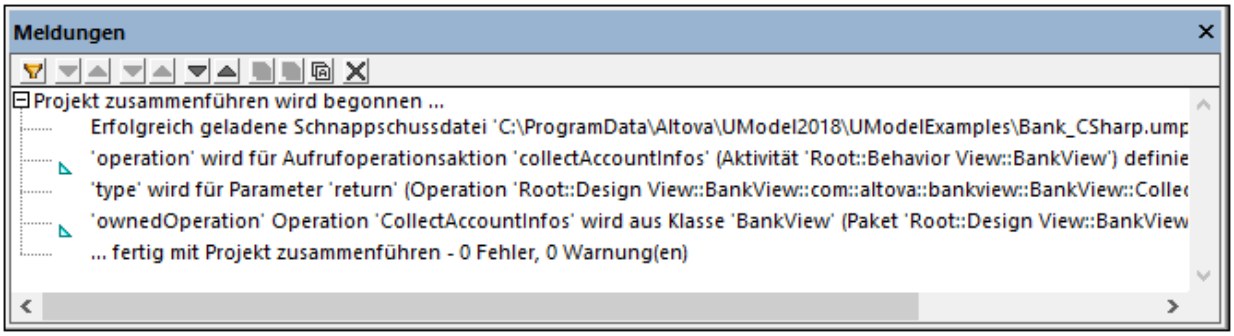
© 2016-2022 Altova GmbH

6.7 Zusammenführen von UModel-Projekten

UModel unterstützt 2-Weg- und 3-Weg-Projektzusammenführungen. In beiden Fällen werden unterschiedliche UModel-Projektdateien zu einem gemeinsamen UModel *.ump Modell zusammengeführt. Dies ist hilfreich, wenn mehrere Personen gleichzeitig am selben Projekt arbeiten oder wenn Sie Ihre Arbeit einfach in einem einzigen Modell zusammenführen möchten.

So führen Sie zwei UML-Projekte zusammen:

1. Öffnen Sie die UML-Datei, in die das zweite Modell überführt werden soll.
2. Wählen Sie die Menüoption **Projekt | Projekt zusammenführen....**
3. Wählen Sie das zweite UML-Projekt aus, also das Projekt, das in die erste Datei überführt werden soll. Im Meldungsfenster sehen Sie Meldungen über den Ablauf der Zusammenführung und die relevanten Einzelheiten werden protokolliert.



Anmerkung: Wenn Sie auf einen der Einträge im Meldungsfenster klicken, wird das entsprechende Modellierungselement in der Modell-Struktur angezeigt.

Ergebnisse der Zusammenführung:

- Neue Modellierungselemente, d.h. Elemente, die in der Quelle nicht vorhanden sind, werden zum Projekt hinzugefügt.
- Unterschiede in denselben Modellierungselementen; die Elemente aus dem **zweiten** Modell haben Vorrang. So kann es z.B. nur einen Standardwert für ein Attribut geben. Es wird der Standardwert aus der zweiten Datei verwendet.
- Diagrammunterschiede: UModel überprüft zuerst, ob es Unterschiede zwischen den Diagrammen der beiden Modelle gibt.
  Wenn ja, so wird das neue/unterschiedliche Diagramm zum Modell hinzugefügt (und eine fortlaufende Nummerierung wird an den Namen des Diagramms angehängt, z.B: activity1 usw.) und das ursprüngliche Diagramm wird beibehalten.
  Falls keine Unterschiede vorhanden sind, werden identische Diagramme ignoriert und es werden keine Änderungen vorgenommen.
  Sie können anschließend entscheiden, welches der Diagramme Sie beibehalten oder löschen möchten. Natürlich können Sie auch beide Diagramme beibehalten, wenn Sie möchten.
- Die gesamte Zusammenführung kann Schritt für Schritt rückgängig gemacht werden. Klicken Sie dazu in der Symbolleiste auf das Symbol "**Rückgängig**" oder drücken Sie **Strg + Z**.

- Wenn Sie im Meldungsfenster auf einen Eintrag klicken, wird das entsprechende Element in der Modell-Struktur angezeigt.
- Der Dateiname der zusammengeführten Datei, also der ersten Datei, die Sie geöffnet haben, wird beibehalten!

6.7.1 3-Weg-Projektzusammenführung

UModel unterstützt nun die Zusammenführung mehrerer UModel-Projekte, die gleichzeitig von mehreren Entwicklern bearbeitet wurden, in einer 3-Weg-Projektzusammenführung.

Die 3-Weg-Projektzusammenführung funktioniert bei UModel-Projekten der **obersten Ebene**, d.h. Projekten die auch Unterprojekte enthalten können. Nicht unterstützt wird die Zusammenführung einzelner **Dateien**, wenn diese Dateien nicht aufgelöste Referenzen auf andere Dateien enthalten.

Bei der Zusammenführung von **Hauptprojekten** werden auch alle editierbaren Unterprojekte automatisch zusammengeführt. Die Unterprojekte müssen nicht separat zusammengeführt werden. Ein Beispiel dazu finden Sie unter Beispiel: Manuelle 3-Weg-Projektzusammenführung 310.

- Die gesamte Projektzusammenführung kann durch Klicken auf die "Rückgängig"-Schaltfläche in der Symbolleiste oder Drücken von **Strg+Z** Schritt für Schritt rückgängig gemacht werden.
- Wenn Sie im Fenster "Meldungen" auf einen Eintrag klicken, wird dieses Element in der Modell-Struktur angezeigt.
- Der Dateiname der zusammengeführten Datei, also der ersten Datei, die Sie geöffnet haben, wird beibehalten.

Ergebnisse der Zusammenführung

Mit "Quelle" wird im Folgenden die Anfangsprojektdatei bezeichnet, also die Datei, die Sie geöffnet haben, bevor Sie mit der Zusammenführung begonnen haben.

- Neue Modellierungselemente in der zweiten Datei, also Elemente, die in der Quelle nicht vorhanden sind, werden zum zusammengeführten Modell hinzugefügt.
- Neue Modellierungselemente in der Quelldatei, also Elemente, die in der zweiten Datei nicht vorhanden sind, bleiben im zusammengeführten Modell erhalten.
- Gelöschte Modellierungselemente aus der zweiten Datei, also Elemente, die in der Quelle noch immer vorhanden sind, werden aus dem zusammengeführten Modell entfernt.
- Gelöschte Modellierungselemente aus der Quelldatei, also Elemente, die in der zweiten Datei noch vorhanden sind, bleiben im zusammengeführten Modell gelöscht.

Unterschiede im selben Modellierungselement:

- Wenn eine Eigenschaft (z.B. die Sichtbarkeit einer Klasse) in der Quelldatei oder der zweiten Datei geändert wurde, so wird im zusammengeführten Modell der aktualisierte Wert verwendet
- Wenn eine Eigenschaft (z.B. die Sichtbarkeit einer Klasse) sowohl in der Quelle als auch in der zweiten Datei geändert wurde, so wird der Wert aus der zweiten Datei verwendet (und im Meldungsfenster unterhalb wird eine Warnmeldung angezeigt)

Verschobene Elemente:

- Wenn ein Element in der Quelle oder der zweiten Datei verschoben wurde, so wird das Element im zusammengeführten Modell ebenfalls verschoben

© 2016-2022 Altova GmbH

- Wenn ein Element sowohl in der Quelle als auch in der zweiten Datei (in unterschiedliche Parent-Elemente) verschoben wird, so erscheint eine Meldung, in der Sie aufgefordert werden, das Parent-Element im zusammengeführten Modell manuell auszuwählen.

Diagrammunterschiede:

UModel überprüft zuerst, ob zwischen den Diagrammen der beiden Modelle Unterschiede bestehen. Falls dies der Fall ist, so wird das neue/geänderte Diagramm zum zusammengeführten Modell (mit einer fortlaufenden Nummer, wie activity1 usw.) hinzugefügt und das Original wird beibehalten. Wenn es keine Unterschiede gibt, so werden identische Diagramme ignoriert, d.h. es gibt keine Änderungen. Sie können später entscheiden, welches der Diagramme beibehalten oder gelöscht werden soll. Natürlich können Sie auch beide Diagramme beibehalten.

Versionskontrollunterstützung für 3-Weg-Zusammenführungen

Beim Ein-/Auschecken von Projektdateien generiert UModel automatisch "gemeinsame Vorgängerdateien" (oder Schnappschussdateien), anhand derer anschließend die 3-Weg-Projektzusammenführung durchgeführt wird. Dies ermöglicht viel genauere Zusammenführungsresultate als die normale 2-Weg-Zusammenführung.

Es hängt vom jeweiligen Versionskontrollsystem, das Sie verwenden, ab, ob UModel die automatische 3-Weg-Zusammenführung über die Schnappschussdatei unterstützt. Eine manuelle 3-Weg-Zusammenführung ist jedoch immer möglich.

- Versionskontrollsysteme, die die Dateizusammenführung durchführen, ohne dass der Benutzer darauf Einfluss nehmen kann, werden eine automatische 3-Weg-Zusammenführung wahrscheinlich nicht unterstützen.
- Versionskontrollsysteme, in denen Sie bei einem geänderten Projekt zwischen Ersetzen und Zusammenführen wählen können, unterstützen eine 3-Weg-Zusammenführung im Allgemeinen. Nachdem das Versionskontroll-Plugin die Datei ersetzt hat, wird bei Auswahl des Befehls "Ersetzen" eine UModel-Dateiwarnung aktiviert, die Ihnen dann gestattet, eine 3-Weg-Zusammenführung durchzuführen. Das Ein- und Auschecken muss über UModel erfolgen.
- Haupt- und Unterprojekte können unter Versionskontrolle gestellt werden. Wenn Daten in einem Unterprojekt geändert wurden, werden Sie automatisch gefragt, ob das/die Unterprojekte ausgecheckt werden sollen.
- Bei jedem Ein-/Auscheckvorgang wird eine gemeinsame Vorgänger- oder Schnappschussdatei erstellt, die dann für die 3-Weg-Projektzusammenführung verwendet wird.

Anmerkung: Schnappschussdateien werden nur mit den Standalone-Versionen von UModel automatisch erstellt und verwendet, d.h. diese Funktionen stehen in der Eclipse- und der Visual Studio-Plugin-Version nicht zur Verfügung.

Beispiel

Benutzer A bearbeitet eine UModel-Projektdatei und ändert den Namen einer Klasse im BankView-Hauptdiagramm. Benutzer B öffnet dieselbe Projektdatei und ändert die Sichtbarkeit derselben Klasse.

Anhand der Schnappschussdateien, die für die einzelnen Benutzer erstellt wurden, wird ein Bearbeitungsverlauf erstellt, der eine Zusammenführung der einzelnen Änderungen im Projekt ermöglicht. Sowohl die Änderungen am Namen als auch an der Sichtbarkeit werden bei der 3-Weg-Zusammenführung in der Projektdatei zusammengeführt.

6.7.2 Beispiel: Manuelle 3-Weg-Projektzusammenführung

In diesem Beispiel wird eine einfache 3-Weg-Projektzusammenführung gezeigt. Angenommen zwei Personen, Tom und Alice, haben jeweils eine eigene Kopie eines UModel-Projekts erstellt und Änderungen daran vorgenommen. Es gibt nun drei Versionen desselben Projekts: das Originalprojekt, die Kopie von Tom und die Kopie von Alice. Bei einer 3-Weg-Projektzusammenführung bildet das Originalprojekt die "gemeinsame Vorgängerdatei".

Als gemeinsame Vorgängerdatei verwenden wir für dieses Beispiel das Projekt **Bank_CSharp.ump** aus dem Ordner **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**. Die Kopien von Tom und Alice müssen manuell erstellt werden. Erstellen wir daher zuerst in Unterordnern des Ordners ... \UModelExamples zwei Kopien des Projekts **Bank_Csharp.ump.** Nennen wir die Unterordner **Alice** und **Tom**; und lassen wir den Projektnamen unverändert.

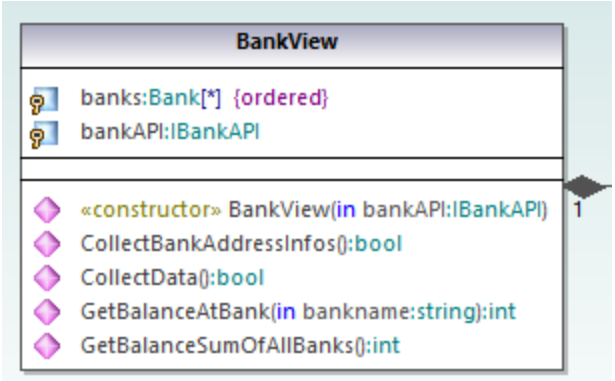
> Verwenden Sie zum Erstellen der Kopien von Tom und Alice den Befehl **Projekt | Speichern unter.** Wenn Sie gefragt werden, ob die relativen Pfade angepasst werden sollen, klicken Sie auf **Ja**. Dadurch vermeiden Sie Syntaxfehler in den Projektkopien.

Wir wollen in diesem Beispiel zeigen, wie Alice Änderungen nicht nur aus dem Originalprojekt **Bank_CSharp.ump,** sondern auch aus Toms Projekt in einem neuen Modell (in einer so genannten 3-Weg-Zusammenführung) zusammenführt.

Schritt 1: Vorbereiten von Toms Projekt

Tom öffnet die Projektdatei **Bank_CSharp.ump** im Ordner **Tom,** öffnet das Diagramm "BankView Main" und nimmt Änderungen an der Klasse `BankView` vor.

1. Die Operation `CollectAccountInfos():bool` wird aus der `BankView`-Klasse gelöscht.
2. Die Sichtbarkeit (**visibility)** der Operation `CollectBankAddressInfos():bool` wird von "protected" in "public" geändert.



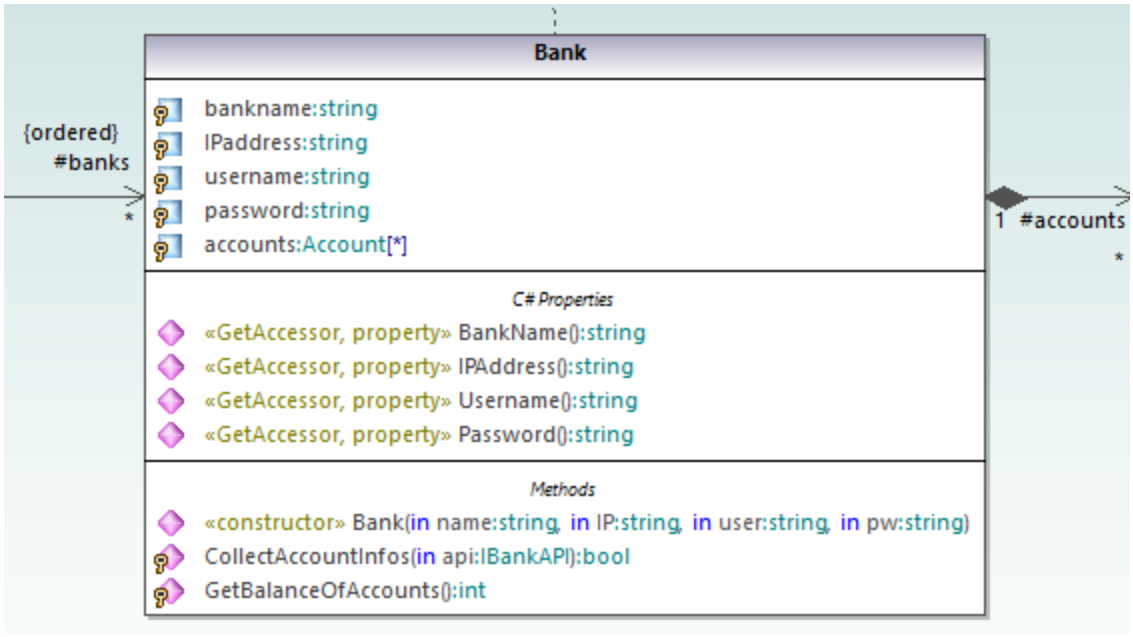
3. Anschließend wird das Projekt gespeichert.

Schritt 2: Vorbereiten des Projekts von Alice

Alice öffnet die Projektdatei **Bank_CSharp.ump** im Ordner **Alice** öffnet das Diagramm "BankView Main" und nimmt Änderungen an der Klasse `Bank` vor.

© 2016-2022 Altova GmbH

1. Die Operationen `CollectAccountInfos` und `GetBalanceOfAccounts` werden beide von "public" in "protected" geändert.



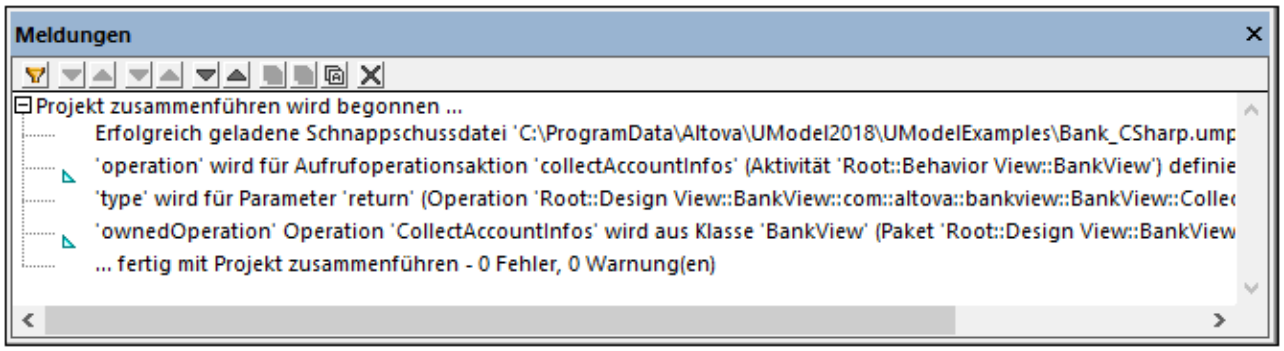
2. Anschließend wird das Projekt gespeichert.

Schritt 3: Durchführung einer 3-Weg-Zusammenführung

Alice beginnt nun eine 3-Weg-Projektzusammenführung:

1. Öffnen Sie das Projekt von Alice aus dem Ordner **Alice.**
2. Wählen Sie im Menü **Projekt** den Befehl **Projekt zusammenführen (3-Weg)** und wählen Sie die von Tom geänderte Projektdatei aus dem Ordner **Tom** aus.
3. Sie werden nun aufgefordert, die gemeinsame Vorgängerdatei zu öffnen. Wählen Sie die Originalprojektdatei **Bank_CSharp.ump** aus dem Ordner **...\UModelExamples** aus.

Die 3-Weg-Zusammenführung wird gestartet und sie kehren zur Projektdatei, von der aus Sie die 3-Weg-Zusammenführung gestartet haben, also zur Projektdatei im Ordner Alice, zurück. Im Fenster "Meldungen" wird die Zusammenführung im Detail angezeigt.



Das Resultat der 3-Weg-Zusammenführung ist das folgende:

- Die von Tom am Projekt vorgenommenen Änderungen werden in der Projektdatei von Alice repliziert.
- Die von Alice am Projekt vorgenommenen Änderungen werden in der Projektdatei beibehalten.

Anmerkung: Für zukünftige 3-Weg-Zusammenführungen zwischen den Projektdateien in den Ordnern **Tom** und **Alice** sollte nun in Zukunft die Projektdatei im Ordner Alice als gemeinsame Vorgängerdatei verwendet werden.

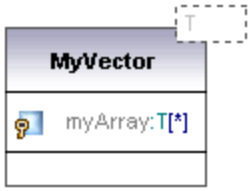
© 2016-2022 Altova GmbH

6.8 UML-Vorlagen

UModel unterstützt die Verwendung von UML-Vorlagen (Templates) und das Mappen dieser Vorlagen auf oder von Java-, C#- und Visual Basic Generics.

- Vorlagen (Templates) sind "potentielle" Modellelemente mit nicht gebundenen formalen Parametern.
- Diese parametrisierten Modellelemente beschreiben eine Gruppe von Modellelementen eines bestimmten Typs: Classifier oder Operationen.
- Vorlagen können nicht direkt als Typen verwendet werden. Die Parameter müssen gebunden sein.
- Instantiieren bedeutet, die Vorlagenparameter an aktuelle Werte zu binden.
- Aktuelle Werte für Parameter sind Ausdrücke.
- Durch die Bindung zwischen einer Vorlage und einem Modellelement wird ein neues Modellelement (ein gebundenes Element) auf Basis der Vorlage erzeugt.
- Bei Vorhandensein mehrerer einschränkender Classifier in C# können die Vorlagenparameter direkt auf dem Register "Eigenschaften" bearbeitet werden, wenn der Vorlagenparameter ausgewählt ist.

Anzeige der **Vorlagensignatur** in UModel:



- Klassenvorlage mit dem Namen **MyVector** mit dem formalen Vorlagenparameter "**T**", der in einem gestrichelten Rechteck angezeigt wird.
- Formale Parameter ohne Typinfo (T) sind implizit Classifier: Class, Datatype, Enumeration, PrimitiveType, Interface. Alle anderen Parametertypen müssen explizit angezeigt werden z.B. Integer.
- Eigenschaft **myArray** mit einer unbeschränkten Anzahl an Elementen vom Typ T.

  Wenn Sie mit der rechten Maustaste auf die Vorlage klicken und den Eintrag **Anzeigen | Gebundene Elemente** auswählen, werden die gebundenen Elemente angezeigt.

Anzeige der **Vorlagenverwendung**:



- eine gebundene benannte Vorlage **intvector**
- Vorlage vom Typ **MyVector**, wobei
- der Parameter **T** durch **int** ersetzt wird
- **"Ersetzt durch"** wird angezeigt durch **- >**.

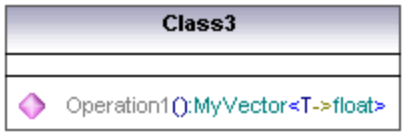
Verwendung von Vorlagen in Eigenschaften/Operationen:



Eine anonyme Vorlagenverwendung:

- Eigenschaft MyFloatVector vom Typ **MyVector<T->float>**

Vorlagen können auch beim Definieren von Eigenschaften oder Operationen definiert werden. Die Autokomplettierungsfunktion hilft Ihnen, die Syntaxvorgaben einzuhalten.



- Operation1 gibt einen Vektor von floats zurück.

6.8.1 Vorlagensignaturen

Eine Vorlagensignatur ist ein String, der die formalen Vorlagenparameter definiert. Eine Vorlage (Template) ist ein parametrisiertes Element, das zum Generieren neuer Modellelemente verwendet wird, indem die formalen Parameter durch tatsächliche Parameter (Werte) ersetzt werden bzw. daran gebunden werden.

Formaler Vorlagenparameter

T
Vorlage mit einem einzigen formalen Parameter, ohne Typenkonzept
(speichert Elemente vom Typ T)

Multiple formale Vorlagenparameter

KeyType:DateType, ValueType

Parameterersetzung

T>aBaseClass

Die Parametersubstitution muss vom Typ "aBaseClass" oder davon abgeleitet sein.

Standardwerte für Vorlagenparameter

T=aDefaultValue

Ersetzen von Classifiern

T>{contract}aBaseClass

allowsSubstitutable is true
Der Parameter muss ein Classifier sein, der an die Stelle des Classifiers gesetzt werden kann, der durch den Classifier-Namen definiert ist.

Einschränken von Vorlagenparametern

T:Interface>anInterface

Wenn Sie Parameter auf etwas anderes als eine Klasse einschränken wollen (Schnittstelle, Datentyp), wird die Einschränkung nach dem ":"-Zeichen angezeigt. So wird T z.B. an eine Schnittstelle gebunden (T:Interface) die vom Typ "anInterface" (>anInterface) sein muss.

Verwendung von Platzhalterzeichen in Vorlagensignaturen

T>vector<T->?<aBaseClass>

© 2016-2022 Altova GmbH

Der Vorlagenparameter T muss vom Typ "vector" sein, der Objekte enthält, die ein übergeordneter Typ von aBaseClass sind.

Erweitern von Vorlagenparametern

T>Comparable<T->T>

6.8.2 Vorlagenverwendung

Bei der Vorlagenverwendung werden die formalen Parameter durch tatsächliche Werte ersetzt, d.h. die Vorlage wird instantiiert. UModel generiert in diesem Fall automatisch anonym gebundene Klassen. Vorlagenverwendungen können wie unten gezeigt im Klassennamen-Feld definiert werden.



Ersetzen/Binden von formalen Parametern

Vektor <T->int>

Erstellen von Vorlagenverwendungen über den Klassennamen

a_float_vector:vector<T->float>

Binden von mehreren Vorlagen gleichzeitig

Class5:vector<T->int, map<KeyType->int, ValueType<T->int>

Verwendung von Platzhalterzeichen ? als Parameter (Java 5.0)

vector<T->?>

Einschränken von Platzhalterzeichen - upper bounds (UModel Erweiterung)

vector<T->?>aBaseClass>

Einschränken von Platzhalterzeichen - lower bounds (UModel Erweiterung)

vector<T->?<aDerivedClass>

6.8.3 Vorlagenverwendung in Operationen und Eigenschaften

Operation, die eine gebundene Vorlage zurückgibt

Class1
Operation1():vector<T->int>

Der Parameter T ist an "int" gebunden. Operation1 gibt einen Vektor von ints zurück.

Klasse, die eine Vorlagenoperation enthält

Class1
Operation1<T>(in T):T

Verwenden von Platzhalterzeichen

Class1
Property1:vector<T->?>

Diese Klasse enthält einen generischen Vektor eines nicht spezifizierten Typs (? ist das Platzhalterzeichen).

Typisierte Eigenschaften können folgendermaßen als Assoziationen angezeigt werden:

- Rechtsklicken Sie auf eine Eigenschaft und wählen Sie **Anzeigen | PropertyX als Assoziation** oder
- Ziehen Sie eine Eigenschaft auf den Diagrammhintergrund.

© 2016-2022 Altova GmbH

7 Transformieren von UML-Modellen

Sie können jedes vorhandene UML-Paket von einer Modellierungssprache in eine andere transformieren. Mit der Transformation werden alle relevanten Elemente, wie Klassen, Schnittstellen, Attribute, Operationen, Generalisierungen usw. von der Quell- in die Zielsprache transformiert. Bei der Quell- und Zielsprache kann es sich um jede beliebige von UModel unterstützte handeln (C++, C#, Java, VB.NET, UML sowie Datenbanken und XML-Schemas).

Für eine Transformation werden ein "Quellmodell" (d.h. das zu transformierende Paket) und ein "Zielmodell" (ein Zielpaket) benötigt. Da das Zielpaket unter Umständen bereits Elemente enthält, können Sie eine Modelltransformation auf zwei verschiedene Arten durchführen:

1. Sie können die Änderungen aus dem Quellmodell im Zielmodell überschreiben.
2. Sie können die Änderungen aus dem Quellmodell mit dem Zielmodell zusammenführen.

(Wenn es sich beim Ziel um ein neues Paket handelt, spielt es bei der ersten Transformation keine Rolle, ob Sie "überschreiben" oder "Zusammenführen (Mergen)" auswählen.)

Wenn das Quellmodell Klassendiagramme enthält, können diese optional in das Zielmodell transformiert werden (dies gilt für C#, Java, VB.NET und UML). Diagramme, die im Zielmodell vorhanden sind, werden gemäß den Transformationseinstellungen aktualisiert, die Elemente darin werden durch diejenigen im Quellmodell überschrieben oder mit diesen zusammengeführt.

Während der Transformation können Sie die einzelnen Datentypen in der Quellsprache über ein Dialogfeld der Assistenten optional auf Datentypen in der Zielsprache mappen. Wenn Sie diesen Schritt überspringen, verwendet UModel standardmäßig vordefinierte Typentsprechungen. Die Typentsprechungen können auch zu einem späteren Zeitpunkt geändert werden, Sie müssen die Transformation jedoch erneut ausführen, damit die Änderungen im Zielmodell übernommen werden.

Wenn Sie Modelltransformationen automatisch durchführen, nimmt UModel die folgenden Änderungen automatisch vor:

- Wenn im UML-Quellmodell das UML-Stereotyp «create» auf eine Klassenoperation angewendet wurde, wird im Zielmodell (C++, C#, Java, VB) das Stereotyp «constructor» angewendet. Wenn eine Operation umgekehrt in C++, C#, Java oder VB.NET das Stereotyp «constructor» hat, erhält dieselbe Operation im UML-Zielmodell das Stereotyp «create».
- Wenn es sich beim Zielmodell um eine Datenbank handelt, wird eine Eigenschaft namens "id" im Quellmodell im Zielmodell in einen Primär- oder Sekundärschlüssel des passenden Datentyps konvertiert.

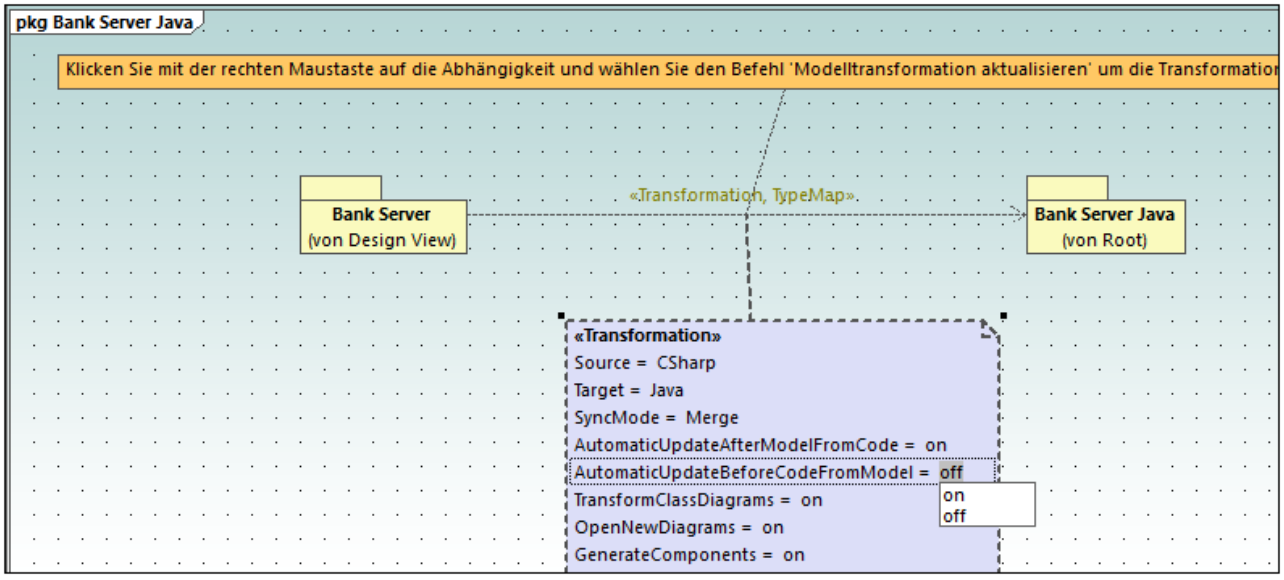
UModel unterstützt die laufende Aktualisierung von transformierten Modellen, d.h. Sie können im Quellmodell weiterarbeiten und die Modelltransformation so oft wie nötig durchführen, um das Zielmodell auf aktuellem Stand zu halten. Die Modelltransformation kann auch automatisch durchgeführt werden, siehe Transformationseinstellungen 320.

So führen Sie eine Modelltransformation durch:

1. Öffnen Sie das UModel-Projekt, das das Paket, das Sie als Quellmodell verwenden möchten, enthält.
2. Klicken Sie im Menü "Projekt" auf **Modelltransformation**.

3. Wählen Sie das Quellprojekt (das Sie in eine andere Sprache transformieren möchten) aus und klicken Sie auf **Weiter**.
4. Wählen Sie eine Zielsprache aus und klicken Sie auf **Weiter**. (Um alle Elemente in ein neues Zielpaket zu platzieren, aktivieren Sie das Kontrollkästchen **In neues Paket transformieren**.)
5. Wählen Sie die Art der Transformation aus (z.B. **Java in C#**). Informationen zu allen anderen Einstellungen finden Sie unter Transformationseinstellungen 320.
6. Wählen Sie eine der folgenden Methoden:
   a. Um die Transformation mit den Standardtypmappings durchzuführen, klicken Sie auf **Fertig stellen**.
   b. Um die Typmappings vor der Transformation zu überprüfen, klicken Sie auf **Weiter**, ändern Sie die Datenmapppings nach Bedarf und klicken Sie anschließend auf **Fertig stellen**.

Wenn die Transformation erfolgreich fertig gestellt wurde, wird automatisch ein neues Paket namens "Modelltransformation von <Quellpaket> in <Zielpaket>" generiert. Das Diagramm wird im *Zielpaket* generiert. Im Diagramm unten sehen Sie das Quellpaket, das Zielpaket, die Abhängigkeitsbeziehung zwischen den beiden Paketen und eine Liste der Eigenschaftswerte.



Modelltransformations-Beispieldiagramm

In diesem Diagramm wird nicht nur die Modelltransformation dargestellt, Sie können darin auch die Modelltransformationseinstellungen ändern. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie auf die Abhängigkeitsbeziehung im Diagramm (oder in der Modell-Struktur - Sie finden diese unter *Beziehungen*).
2. Ändern Sie im Fenster "Eigenschaften" die erforderlichen Optionen.

Doppelklicken Sie alternativ dazu direkt im Diagramm auf einen Eigenschaftswert, um ihn zu ändern.

Nachdem Sie die Transformationseinstellungen geändert haben, führen Sie die Transformation erneut durch, um das Zielmodell zu aktualisieren. Sie können dies auf folgende Arten tun:

- Klicken Sie im Diagramm mit der rechten Maustaste auf die Abhängigkeitsbeziehung und wählen Sie in Kontextmenü den Befehl **Modelltransformation aktualisieren.**

© 2016-2022 Altova GmbH

- Klicken Sie im Fenster "Modell-Struktur" mit der rechten Maustaste auf das Quellpaket und wählen Sie im Kontextmenü den Befehl **Modelltransformation aktualisieren.**

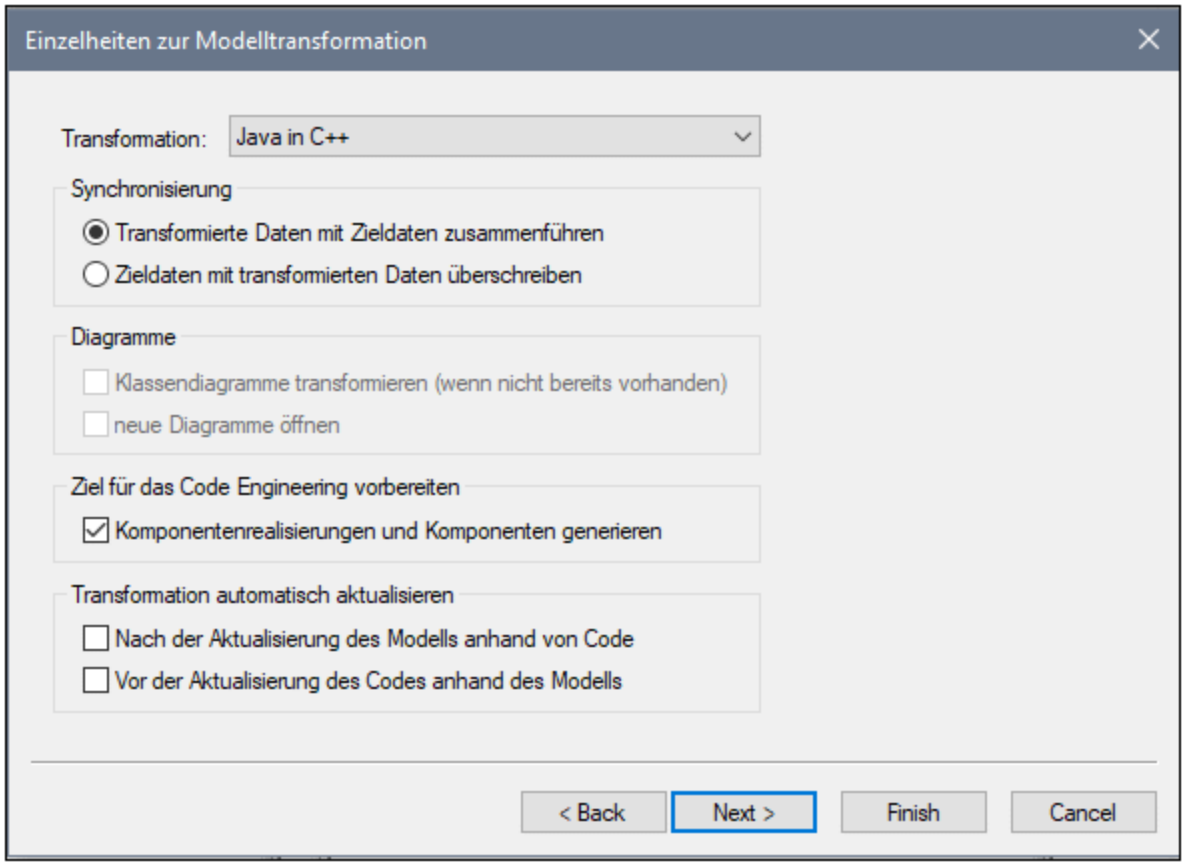
Schritt-für-Schritt-Beispiele für Transformationen finden Sie unter:

- Beispiel: Transformieren von Java in C++ 322
- Beispiel. Transformieren von C# in Java 329
- Beispiel: Transformieren einer Access-Datenbank in SQLite 335

7.1 Transformationseinstellungen

Über das Dialogfeld **Einzelheiten zur Modelltransformation** können Sie die Einstellungen, wie ein Modell in ein anderes Modell transformiert werden soll, definieren bzw. ändern. Dieses Dialogfeld wird angezeigt, wenn Sie eine neue Modelltransformation durchführen oder eine vorhandene aktualisieren, siehe Transformieren von UML-Modellen 317.

Anmerkung: Es gibt bei der Transformation eines Modells in C# eine Option, um Felder in automatisch implementierte C#-Eigenschaften zu transformieren. Diese Option steht als Kontrollkästchen im Dialogfeld **Typ-Mapping** zur Verfügung. Um das Dialogfeld **Typ-Mapping** aufzurufen, klicken Sie im Dialogfeld **Einzelheiten zur Modelltransformation** auf **Weiter**.



Weiter unten finden Sie eine Beschreibung der verfügbaren Optionen.

Transformation

Wählen Sie Quell- und die Zielsprache der Transformation aus. Welche Optionen in der Liste zur Verfügung stehen, hängt von der Code Engineering-Sprache das Pakets ab, das Sie als Quellpaket auswählen. Beachten Sie, dass diese Option nicht zur Verfügung steht (deaktiviert ist), wenn Sie eine vorhandene Transformation erneut ausführen.

© 2016-2022 Altova GmbH

Synchronisierung

Mit Hilfe dieser Option können Sie angeben, ob die Quelldaten mit den Zieldaten zusammengeführt oder ob die Zieldaten durch die Daten aus dem Quelldokument überschrieben werden sollen. Angenommen, eine Klasse im Quelldokument enthält `OperationA`, während dieselbe Klasse im Zieldokument `OperationA` und `OperationB` enthält. Wenn Sie sich für eine Zusammenführung entscheiden, bleiben im Zielmodell beide Operationen erhalten. Wenn Sie sich für Überschreiben entscheiden, wird `OperationB` aus dem Zielmodell gelöscht.

Diagramme

Mit der Option **Klassendiagramme transformieren (wenn nicht bereits vorhanden)** werden neue Klassendiagramme generiert, falls sie im Zielmodell noch nicht vorhanden waren. Um alle neuen Diagramme nach Abschluss der Modelltransformation zu öffnen, aktivieren Sie das Kontrollkästchen **neue Diagramme öffnen**.

Ziel für das Code Engineering vorbereiten

Aktivieren Sie die Option **Komponentenrealisierungen und Komponenten generieren,** wenn Sie vorhaben, das Code Engineering im Zielpaket zu aktivieren. Wenn dieses Kontrollkästchen aktiviert ist, erstellt UModel automatisch **Komponentenrealisierungsbeziehungen** und Code Engineering-Komponenten im Zielmodell. Bevor Code anhand des Zielmodells generiert werden kann, muss auch ein Verzeichnis für die Codegenerierung definiert werden:

1. Klicken Sie im Paket "Component View" auf die automatisch von UModel generierte Komponente.
2. Suchen Sie im Fenster "Eigenschaften" die Eigenschaft **Verzeichnis** und geben Sie einen Verzeichnispfad ein.

Nähere Informationen dazu finden Sie unter Generieren von Programmcode 180.

Transformation automatisch aktualisieren

Mit Hilfe dieser Einstellung können Sie das Quellmodell mit dem Zielmodell synchron halten. Diese Einstellung ist dann sinnvoll, wenn Ihr Quellmodell für die Generierung von Code (oder die Aktualisierung anhand von Code) konfiguriert wurde. Wenn Sie häufig Änderungen am Quellmodell (oder dessen Quellcode) vornehmen, nachdem dieses bzw. dieser in ein Zielmodell transformiert wurde, besteht die Möglichkeit, alle Änderungen automatisch im Zielmodell zu übernehmen. Sie haben dabei folgende Möglichkeiten:

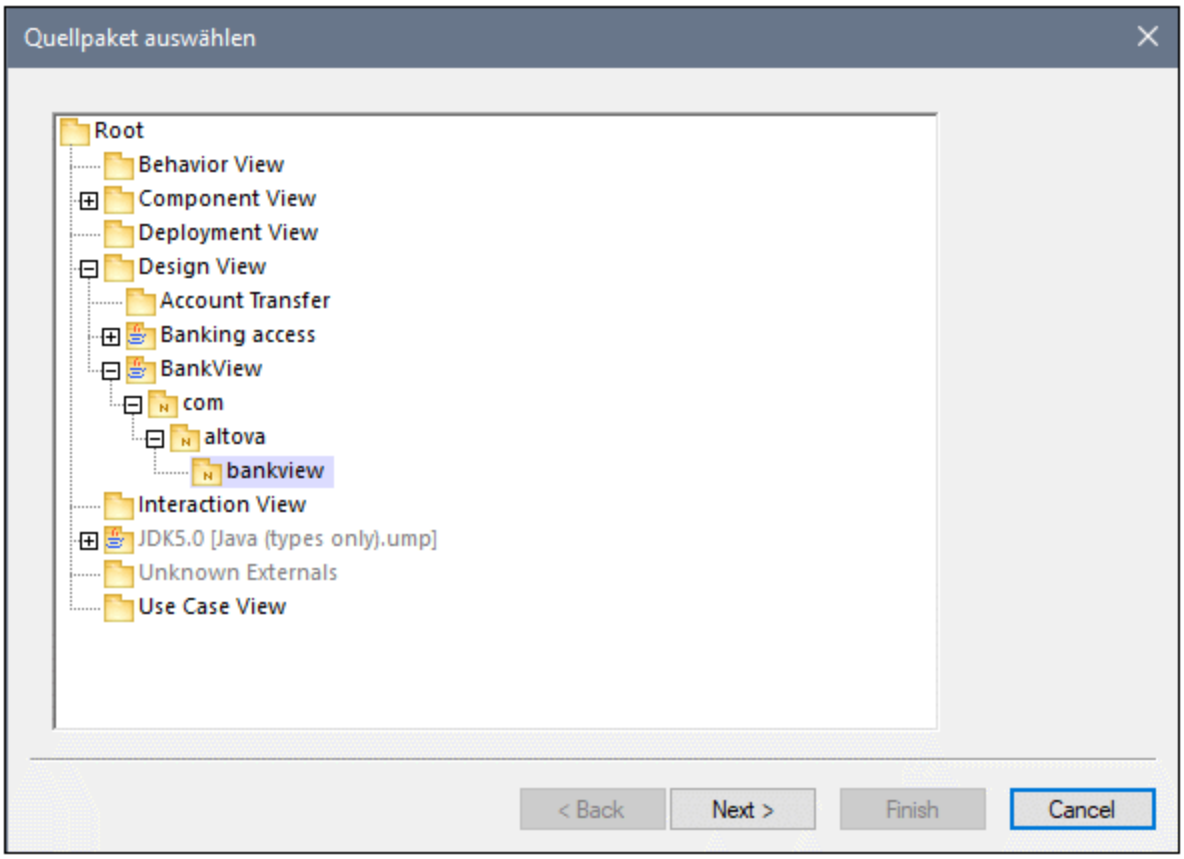
a) jedes Mal, nachdem Sie das Quellmodell anhand des Programmquellcodes aktualisiert haben
b) jedes Mal, bevor Sie Programmcode anhand des Quellmodells generieren
c) in beiden oben beschriebenen Fällen

Angenommen, Ihr Projekt enthält ein ursprünglich für das C#-Code Engineering erstelltes Paket. Dieses Paket wurde anschließend mit dem Menübefehl **Projekt | Modelltransformation** in ein Java-Paket transformiert. Nach der Transformation enthält Ihr Projekt zwei Pakete: das C#-Quellpaket und das Java-Zielpaket. Wenn die Option (a) aktiviert ist, wird die Transformation von C# in Java automatisch jedes Mal, nachdem eine Änderung am C#-Code vorgenommen wurde und das Modell mit der Änderung aktualisiert wurde, durchgeführt. Ebenso erfolgt die Transformation von C# in Java automatisch jedes Mal, bevor Sie C#-Programmcode generieren, wenn Option (b) aktiviert ist und Sie das C#-Modell geändert haben. Ein ausführlicheres Beispiel dazu finden Sie unter Beispiel: Transformieren von C# in Java 329.

7.2 Beispiel: Transformieren von Java in C++

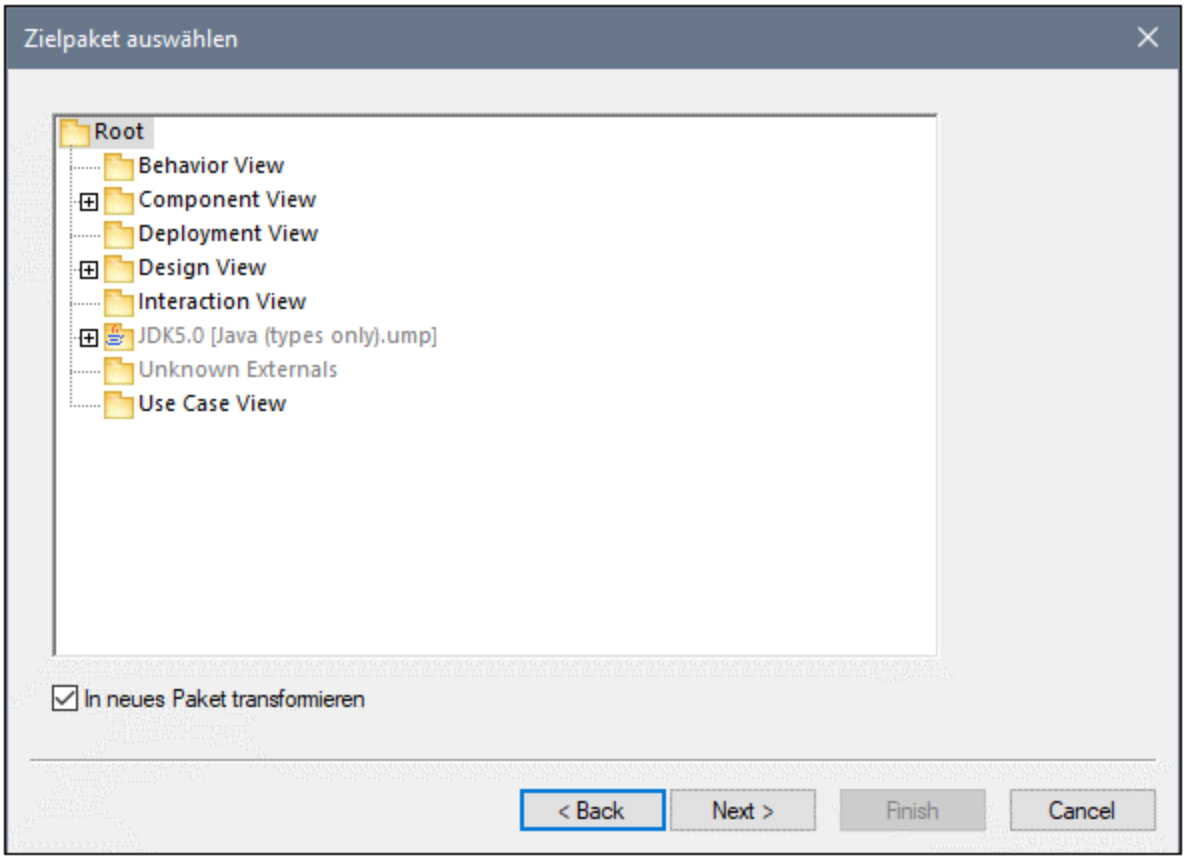
In diesem Beispiel wird gezeigt, wie Sie eine einfache Transformation von einem Java-Modell in ein C++-Modell durchführen. Außerdem erfahren Sie, wie Sie anhand des transformierten (Ziel)modells C++-Code generieren.

1. Öffnen Sie die Beispieldatei **Bank_Java.ump** aus dem Ordner **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples**.
2. Klicken Sie im Menü **Projekt** auf **Modelltransformation**.
3. Wenn Sie aufgefordert werden, ein Quellpaket auszuwählen, wählen Sie den Namespace "bankview" aus. Der vollständige Pfad zu diesem Paket im Fenster "Modell-Struktur" lautet **Root \ DesignView \ BankView \ com \ altova \ bankview**.

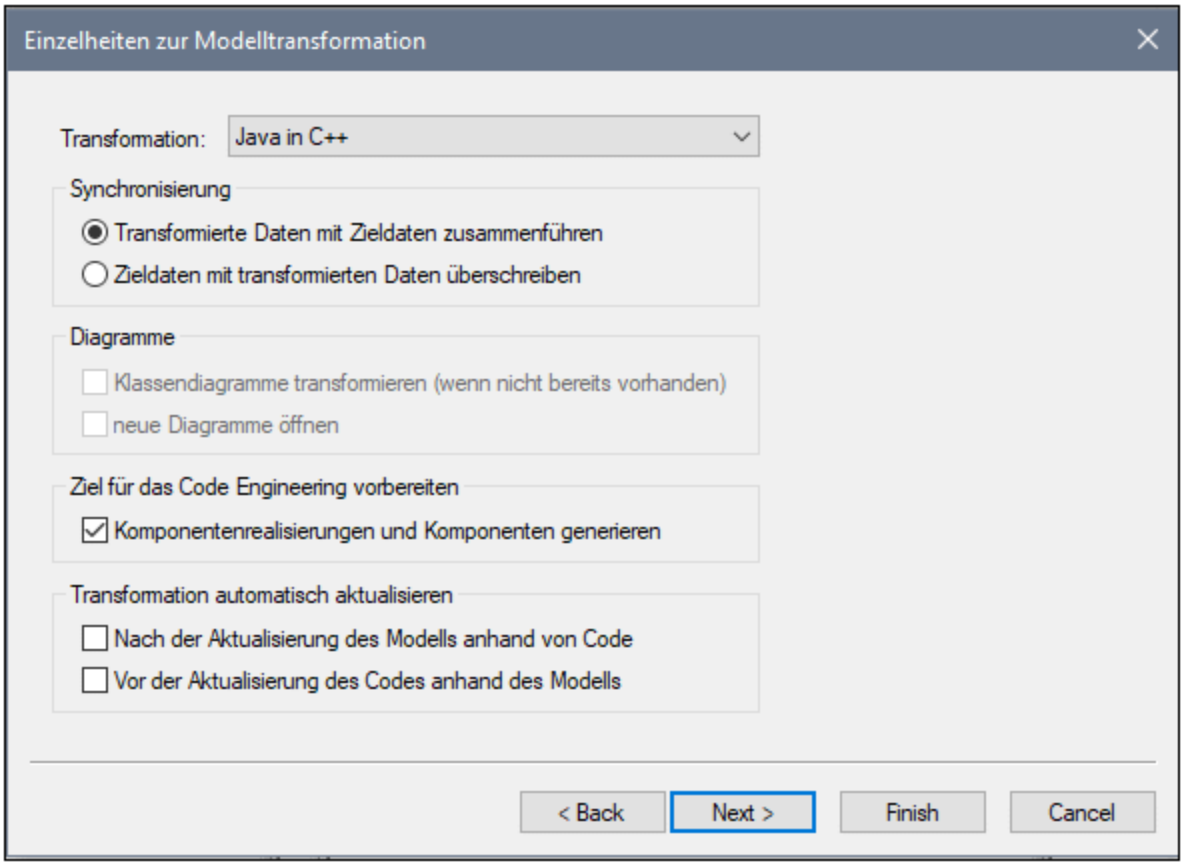


4. Klicken Sie auf **Weiter**. Wenn Sie aufgefordert werden, ein Zielpaket auszuwählen, aktivieren Sie das Kontrollkästchen **In neues Paket transformieren**..

© 2016-2022 Altova GmbH

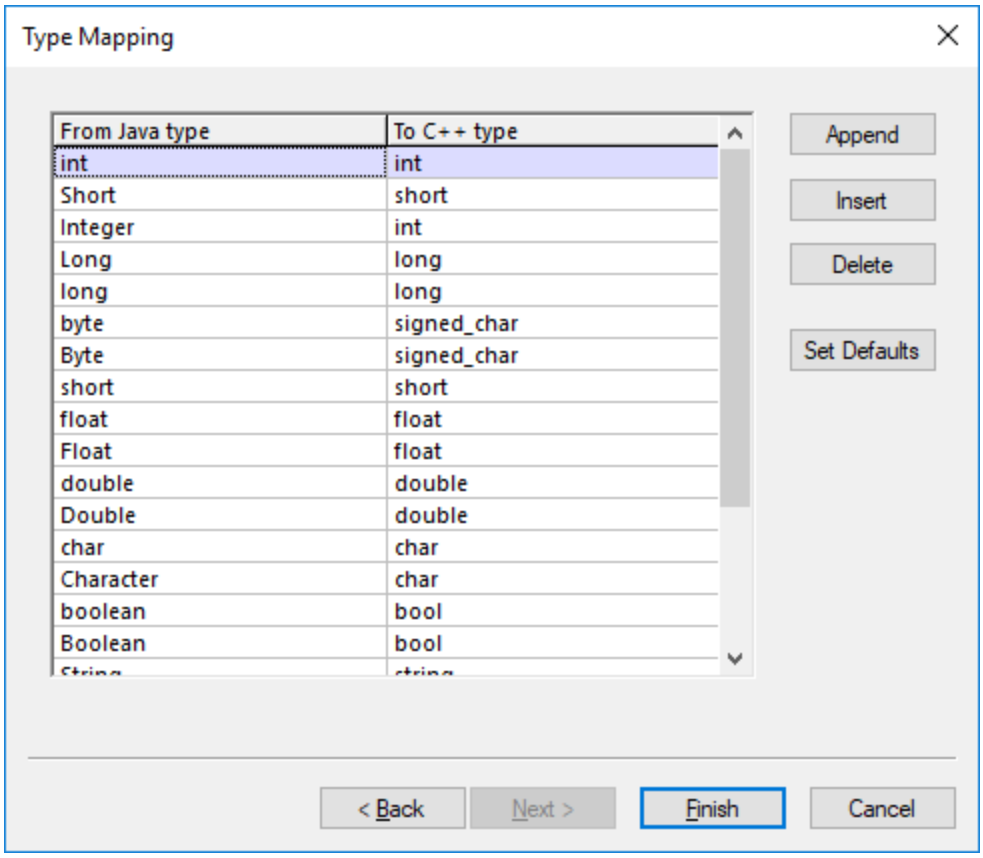


5. Kicken Sie auf **Weiter**. Wählen Sie im daraufhin angezeigten Dialogfeld als Transformationsart **Java in C++**. Informationen zu allen anderen Einstellungen finden Sie unter Transformationseinstellungen 320 .



6. Klicken Sie auf **Weiter**. Daraufhin wird das Dialogfeld "Typ-Mapping" angezeigt, wo Sie die Typ-Entsprechungen zwischen Java und C# definieren können. Klicken Sie auf **Fertig stellen**, um die Standardeinstellungen zu übernehmen.

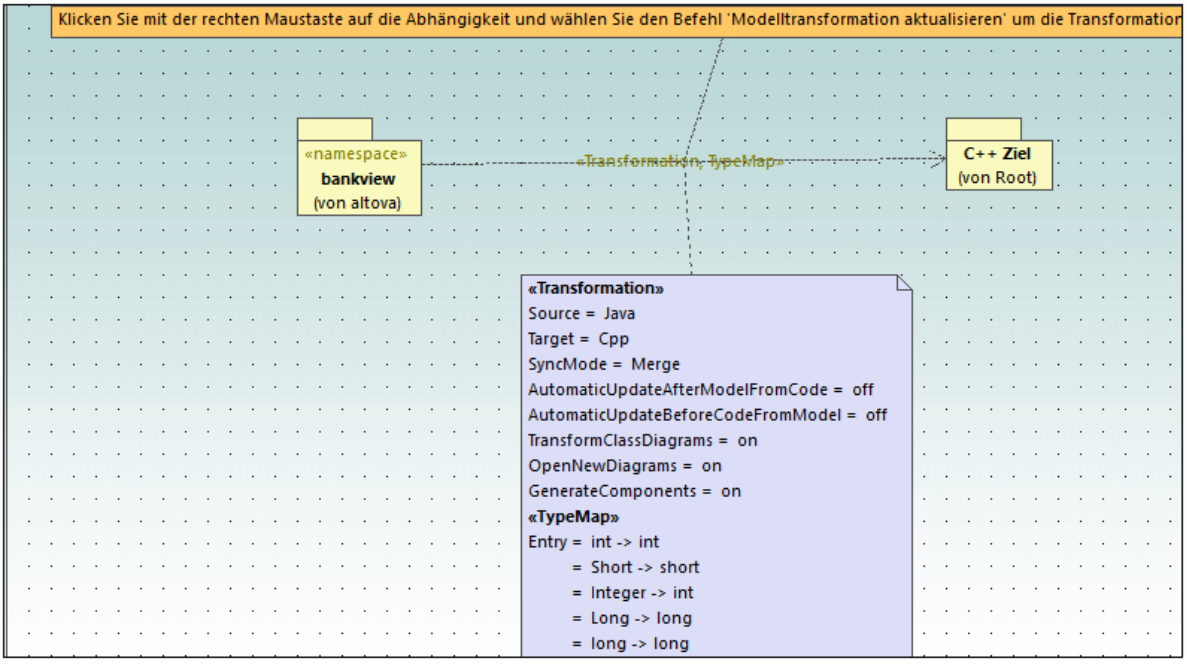
© 2016-2022 Altova GmbH



7. Wenn angezeigt wird, dass das *UModel Model Transformation Profile* inkludiert wird, klicken Sie auf **OK**.

Die Transformation wird durchgeführt und im Projekt werden die folgenden Änderungen vorgenommen:

- Im *Zielmodell* wird ein Paketdiagramm namens "Modelltransformation von bankview in C++-Ziel" generiert und automatisch geöffnet. In diesem Diagramm wird die soeben durchgeführte Transformation dargestellt. Bei Bedarf können Sie darin die zuvor definierten Einstellungen ändern, siehe Transformieren von UML-Modellen 317.

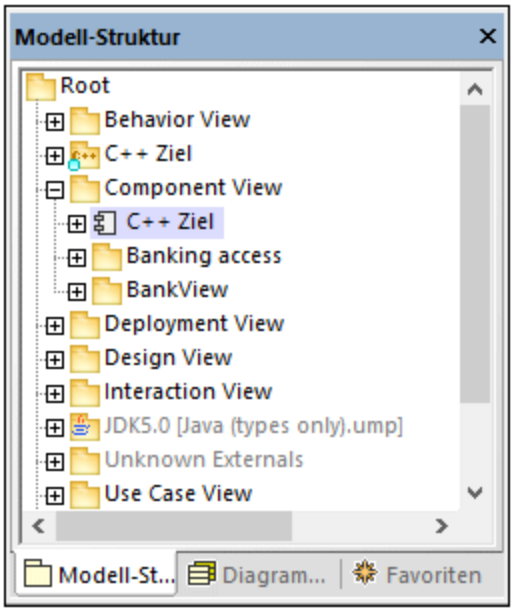


- Das Fenster "Modell-Struktur" enthält nun ein Paket namens "C++-Ziel". Dieses Paket enthält alle anhand des Java-Quellmodells transformierten und für C++ angepassten Elemente. Wenn Sie z.B. das Diagramm "BankView Main" öffnen, werden Sie sehen, dass es anstelle des Java-Typs `boolean` den Typ `bool` enthält.
- Das Paket "Component View" im Fenster "Modell-Struktur" enthält eine neue Komponente "C++-Ziel". Diese Komponente wurde automatisch generiert, da die Einstellung **Komponentenrealisierungen und Komponenten generieren** aktiviert war. In der neuen Komponente sind die Code Engineering-Einstellungen für das Zielmodell (in diesem Fall C++) definiert.

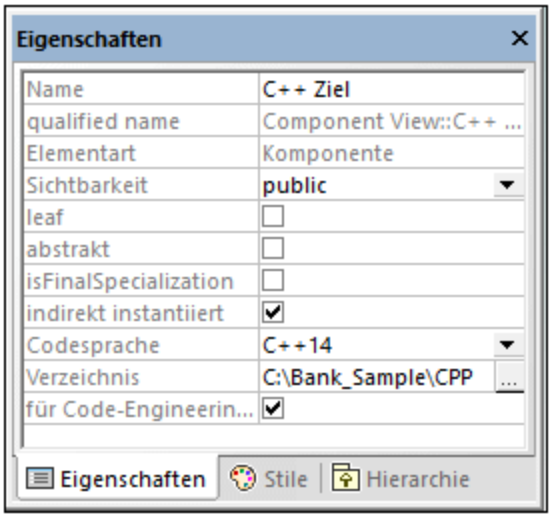
Sie können nun anhand des Zielmodells C++-Code generieren:

1. Klicken Sie im Paket "Component View" auf die Komponente "C++-Ziel".

© 2016-2022 Altova GmbH

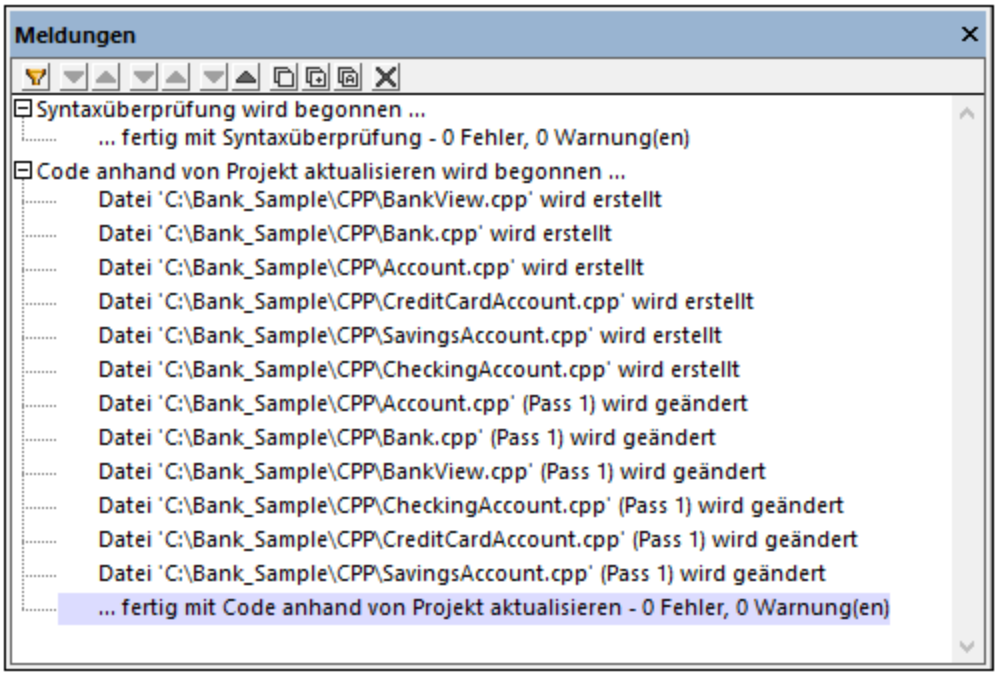


2. Gehen Sie im Fenster "Eigenschaften" zur Eigenschaft **Verzeichnis** und geben Sie das Verzeichnis ein, in dem der C++-Code generiert werden soll (z.B. **C:\Bank_Sample\CPP**, vorausgesetzt, dieses Verzeichnis existiert).



3. Klicken Sie mit der rechten Maustaste auf das C++-Zielpaket und wählen Sie den Befehl **Code Engineering | Merge Programmcode von UModel-Paket**.

Im Fenster "Meldungen" wird das Ergebnis der C++-Codegenerierung angezeigt:



Nähere Informationen zum Generieren von Code anhand eines UModel-Projekts finden Sie unter Generieren von Programmcode 180.

 © 2016-2022 Altova GmbH

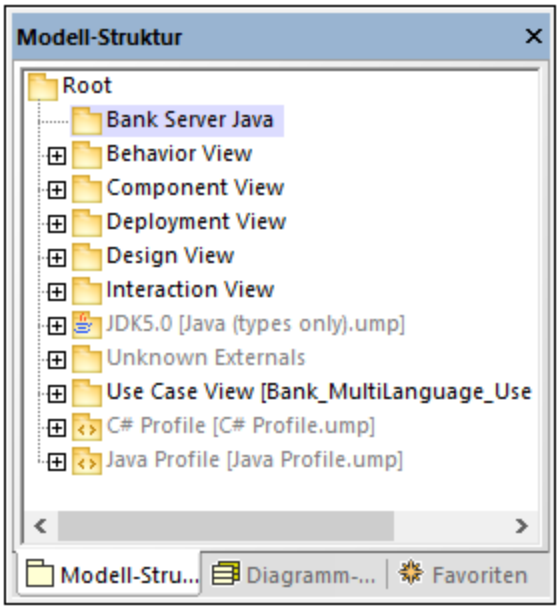
7.3 Beispiel: Transformieren von C# in Java

In diesem Beispiel wird gezeigt, wie Sie eine Transformation von einem C#-Modell in ein Java-Modell durchführen. Außerdem erfahren Sie auch, wie Sie Quell- und Zielmodell manuell oder automatisch synchron halten.

Das in diesem Beispiel verwendete UModel-Projekt steht unter dem folgenden Pfad zur Verfügung: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_multiLanguage.ump**. Wenn Sie das Paket "Design View" öffnen, sehen Sie, dass das Modell zwei in Java geschriebene Pakete und ein in C# geschriebenes Paket enthält. In diesem Beispiel wird angenommen, dass sich die Anforderungen nun geändert haben und das dritte Paket nun ebenfalls in Java implementiert werden muss.

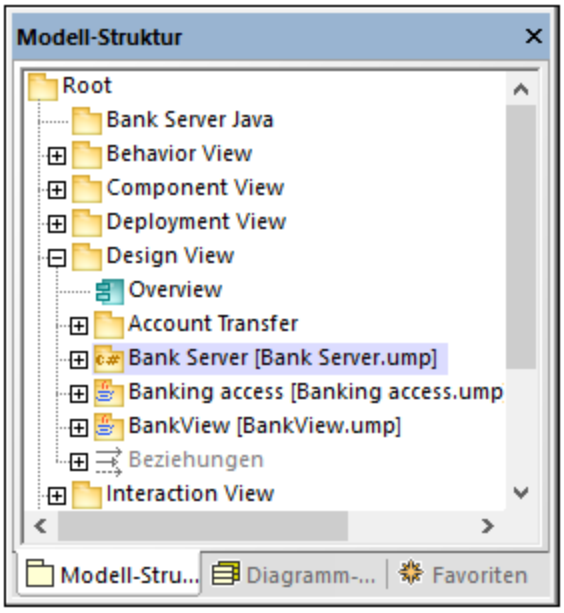
Erstellen wir zuerst das Paket, in dem alle Elemente des neue Java-Zielmodells gespeichert werden sollen.

1. Klicken Sie mit der rechten Maustaste auf das "Root"-Paket und wählen Sie im Kontextmenü den Befehl **Neues Element | Paket**.
2. Geben Sie dem Paket den Namen "Bank Server Java".

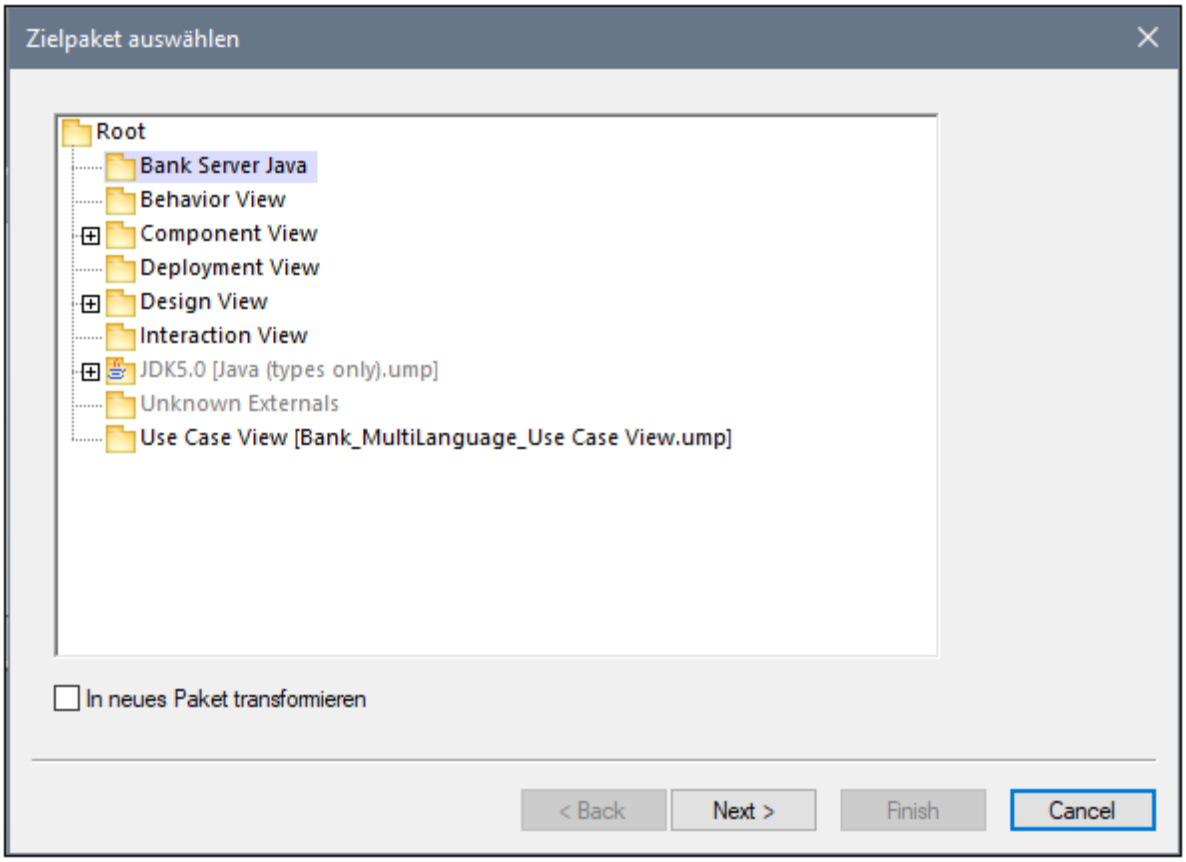


Sie können nun die Transformation von C# in Java folgendermaßen durchführen:

1. Klicken Sie mit der rechten Maustaste auf das Quellpaket "Bank Server" und wählen Sie im Kontextmenü den Befehl **Modelltransformation.**

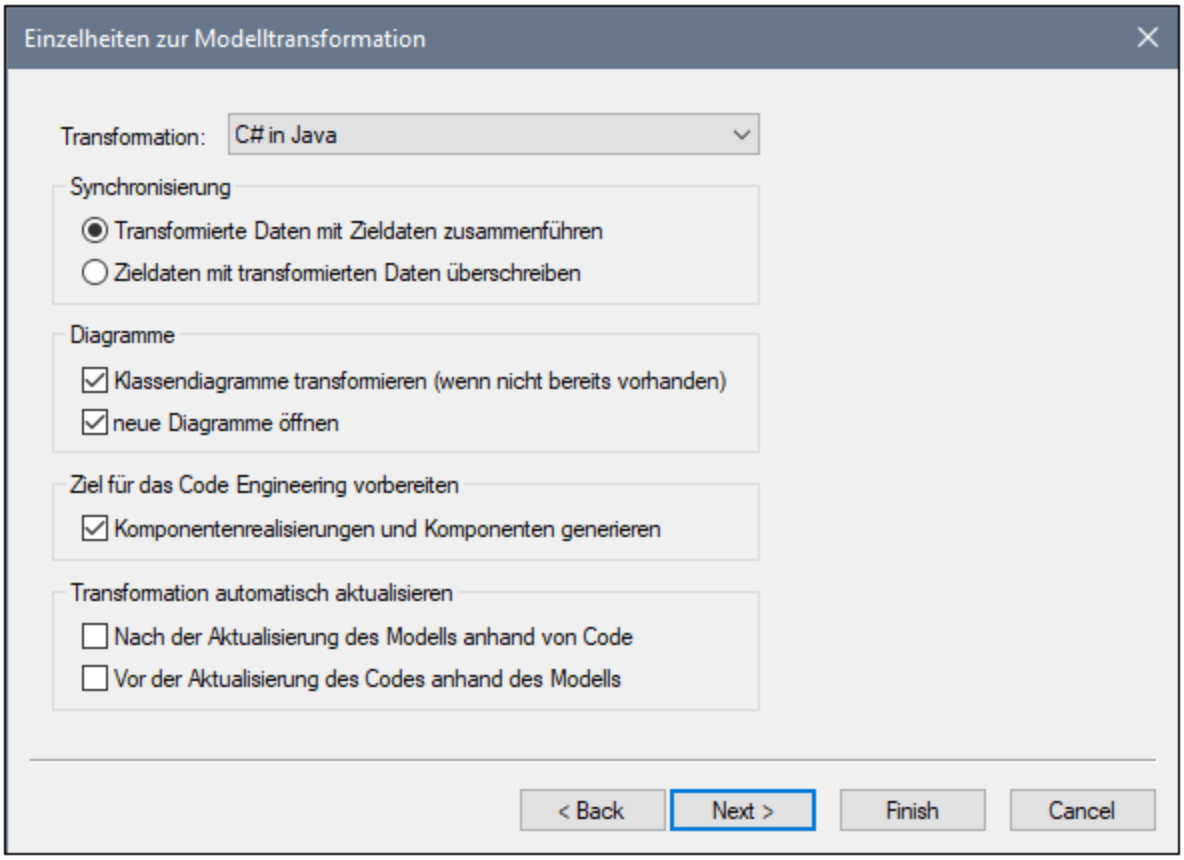


2. Wenn Sie aufgefordert werden, ein Zielpaket auszuwählen, wählen Sie das zuvor erstellte Paket "Bank Server Java" aus und klicken Sie auf **Weiter**.



3. Wählen Sie als Transformationstyp **C# in Java** aus und behalten Sie alle anderen Einstellungen unverändert bei.

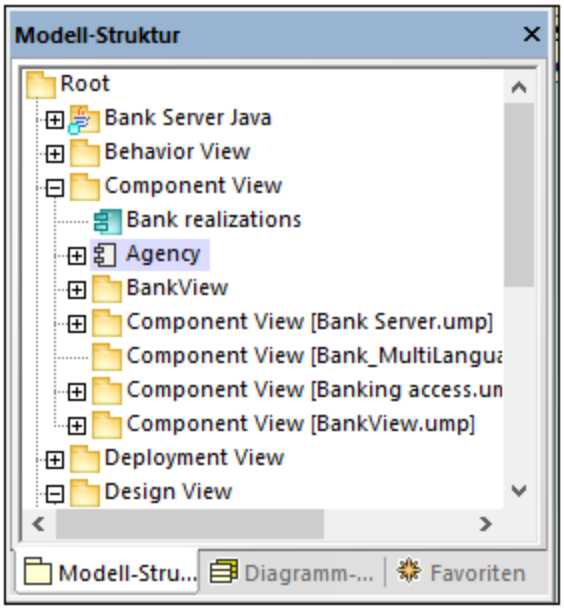
© 2016-2022 Altova GmbH



4. Klicken Sie auf **Fertig stellen**. Wenn Sie darüber informiert werden, dass das "UModel Model Transformation Profile" inkludiert wird, klicken Sie zur Bestätigung auf **OK**.

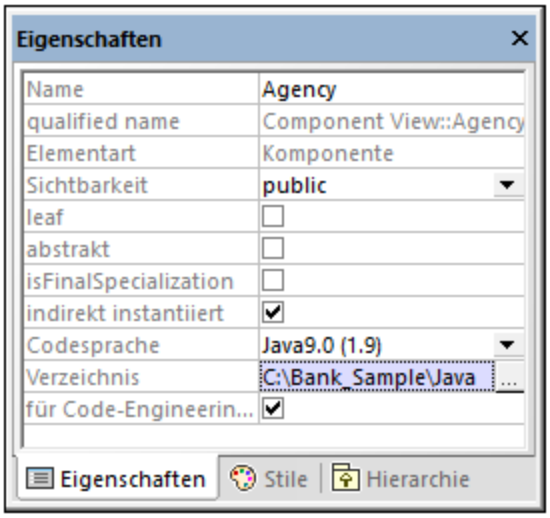
Die Transformation wird fertig gestellt und im Projekt wurden die folgenden Änderungen vorgenommen:

- Im *Zielpaket* wird ein Paketdiagramm namens "Modelltransformation von Bank Server in Bank Server Java" generiert und automatisch geöffnet. In diesem Diagramm wird die soeben durchgeführte Transformation dargestellt. Bei Bedarf können Sie darin die zuvor definierten Einstellungen ändern, wie unten beschrieben.
- Das Zielpaket "Bank Server Java" enthält alle anhand des C#-Quellmodells transformierten und für Java angepassten Elemente. Wenn Sie z.B. das Diagramm "Bank Server" öffnen, werden Sie sehen, dass es anstelle des in C# verwendeten Typs `bool` den Typ `boolean` enthält.
- Das Paket "Component View" im Fenster "Modell-Struktur" enthält eine neue Komponente "Agency". Diese Komponente wurde automatisch generiert, da die Einstellung **Komponentenrealisierungen und Komponenten generieren** aktiviert war und das Quellpaket den Namespace `Agency` enthält. In der neuen Komponente sind die Code Engineering-Einstellungen für das Zielmodell (in diesem Fall Java) definiert.



Sie können nun das Java-Zielmodell für das Code Engineering konfigurieren:

1. Klicken Sie im Paket "Component View" auf die Komponente "Agency".
2. Gehen Sie im Fenster "Eigenschaften" zur Eigenschaft **Verzeichnis** und geben Sie das Verzeichnis ein, in dem der Code generiert werden soll (z.B. **C:\Bank_Sample\Java**, vorausgesetzt, dieses Verzeichnis existiert).

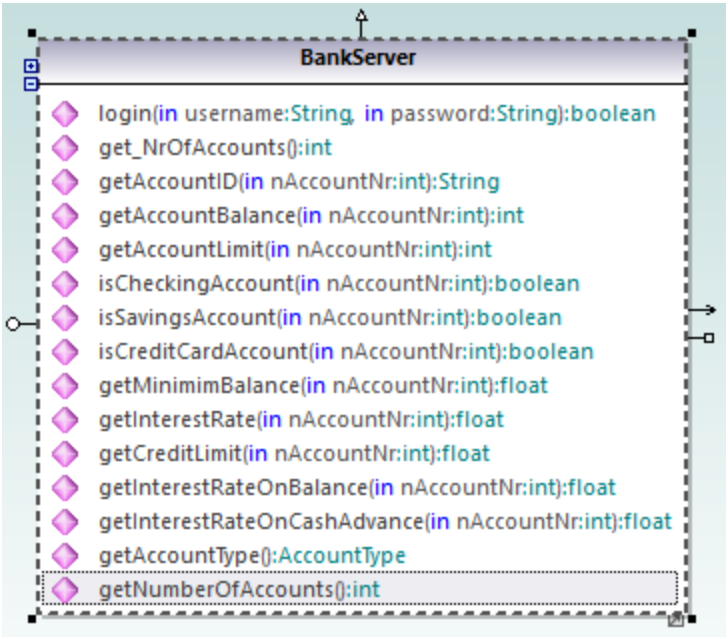


Wir werden nun Java-Code anhand des Zielmodells generieren:

1. Klicken Sie mit der rechten Maustaste auf das "Bank Server Java"-Paket und wählen Sie den Befehl **Code Engineering | Merge Programmcode von UModel-Paket**.
2. Klicken Sie auf **OK,** um die Synchronisierungseinstellungen zu bestätigen.

© 2016-2022 Altova GmbH

Ihr UModel-Projekt enthält zu diesem Zeitpunkt sowohl das Bank Server-Quellmodell in C# als auch das Zielmodell in Java (und beide Modelle sind für die Codegenerierung konfiguriert). Sie können ab jetzt beide Modelle (manuell oder automatisch) synchron halten, selbst wenn Sie am C#-Modell weiterarbeiten. Öffnen Sie als Beispiel dafür das Diagramm "Bank Server" im C#-Quellpaket und fügen Sie zur Klasse `BankServer` eine neue Operation namens `getNumberOfAccounts` hinzu, deren Rückgabewert ein `int`-Wert ist.



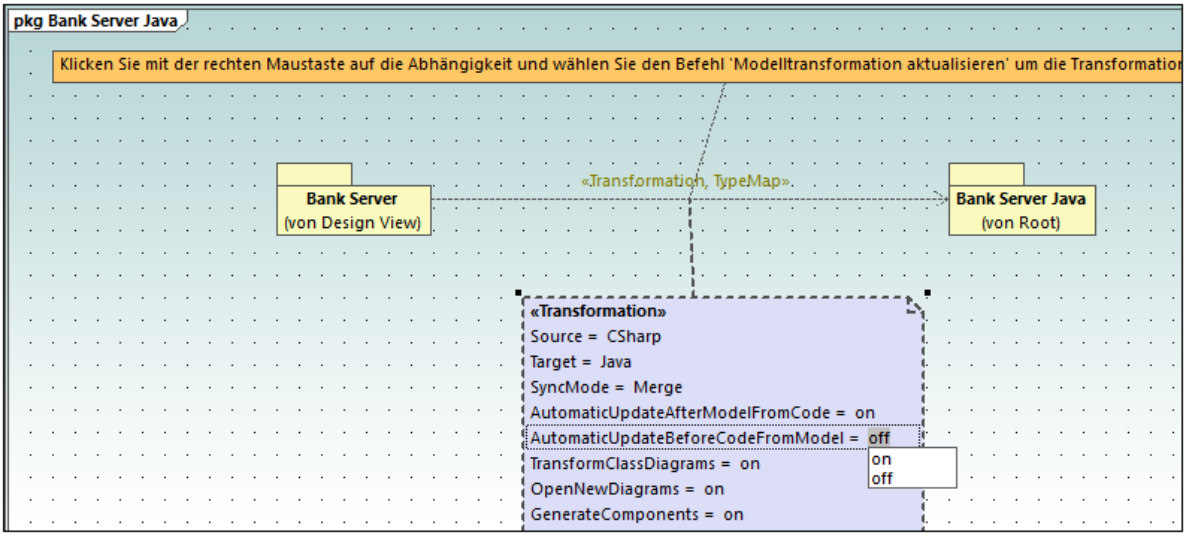
Diese Änderung kann manuell im Zielmodell übernommen werden. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste auf das Quellpaket "Bank Server" und wählen Sie den Befehl **Modelltransformation aktualisieren | von 'Bank Server' in 'Bank Server Java'**.
2. Klicken Sie auf **Fertig stellen**.

Die zuvor über das C#-Modell hinzugefügte Operation `getNumberOfAccounts` wurde nun in das Java-Zielmodell zusammengeführt.

Wir wollen die Transformationseinstellungen nun so konfigurieren, dass Aktualisierungen in C# automatisch in Java übernommen werden, wenn Sie das C#-Quellmodell in das C#-Modell importieren oder Änderungen aus dem Modell im C#-Code zusammenführen.

1. Öffnen Sie das Paketdiagramm "Modelltransformation von Bank Server in Bank Server Java".
2. Doppelklicken Sie auf den Eigenschaftswert `AutomaticaUpdateAfterModelFromCode` und setzen Sie ihn auf "on".
3. Wiederholen Sie die obigen Schritt für den Eigenschaftswert `AutomaticUpdateBeforeCodeFromModel`.



Um die automatischen Aktualisierungen auszulösen,

1. gehen Sie zurück zur Klasse `BankServer` im C#-Quellmodell und löschen Sie die Operation `getNumberOfAccounts`.
2. Klicken Sie mt der rechten Maustaste auf das Paket Bank Server C# und wählen Sie entweder den Befehl **Merge Programcode aus UModel-Paket** oder **Merge UModel-Paket aus Programmcode**.

Da automatische Aktualisierungen nun aktiviert sind, wird diese Änderung nun auch automatisch in der Java-Zielklasse `BankServer` übernommen.

© 2016-2022 Altova GmbH

7.4 Beispiel: Transformieren einer Access-Datenbank in SQLite

In diesem Beispiel wird gezeigt, wie Sie ein Datenbankmodell von einer Datenbankart in eine andere konvertieren. Dabei wird insbesondere gezeigt, wie die Struktur einer Microsoft Access-Datenbank in ein UML-Modell eingelesen und anschließend mit einer vorhandenen SQLite-Datenbank zusammengeführt wird. Nach Fertigstellung dieses Beispiels wird die Struktur der Access-Quelldatenbank in der SQLite-Zieldatenbank übernommen. Beachten Sie, dass die Microsoft Access- und die SQLite-Datenbanken hier nur als Beispiel dienen; auch jede andere von UModel unterstützte Datenbank (siehe UModel und Datenbanken 555) kann auf diese Art konvertiert werden.

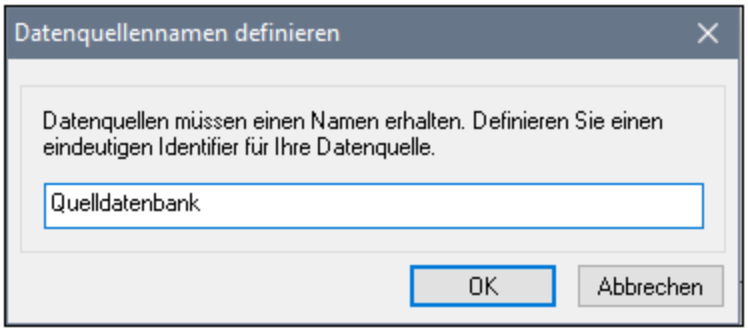
In diesem Beispiel werden folgenden Dateien aus dem Verzeichnis **C:\Users\... \Documents\Altova\UModel2023 \UModelExamples\Tutorial** verwendet:

- **Nanonull.mdb** - die Microsoft Access-Quelldatenbank
- **Nanonull.sqlite** - die SQLite-Zieldatenbank

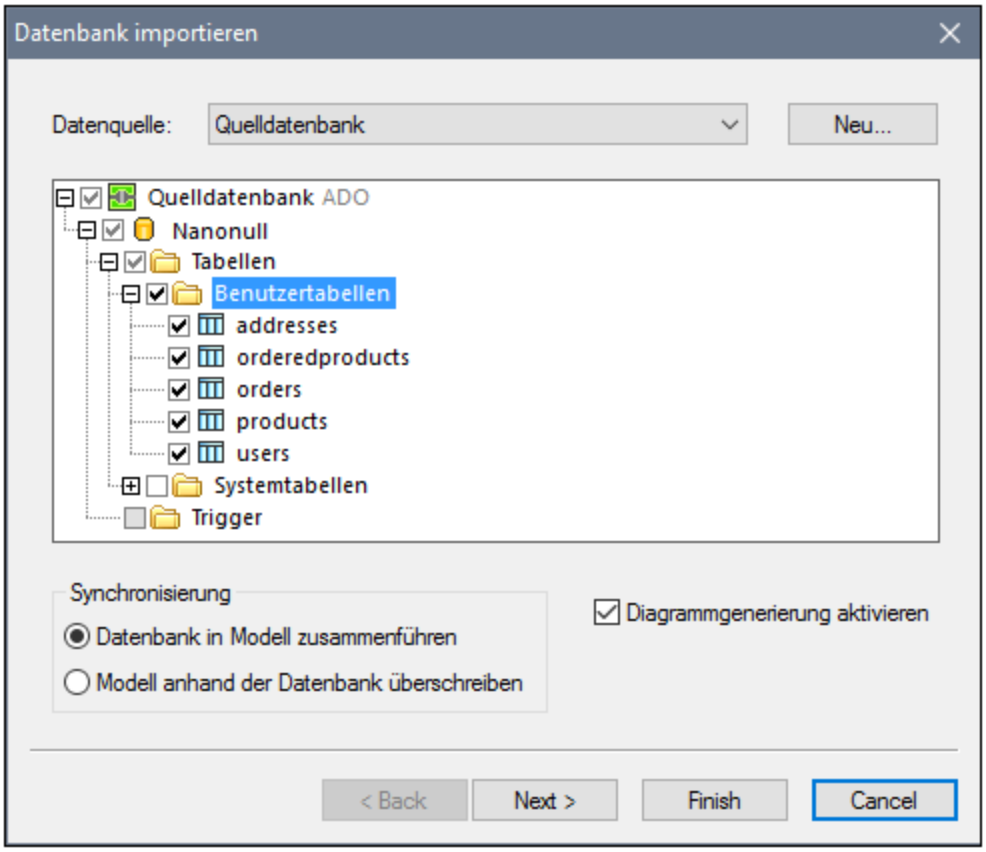
Anmerkung: Bevor Sie beginnen, wird empfohlen, ein Backup der Beispieldatenbankdatei **Nanonull.sqlite** anzulegen, da ihr Inhalt durch das unten beschriebene Verfahren verändert wird.

Schritt 1: Import der Quelldatenbank in UModel

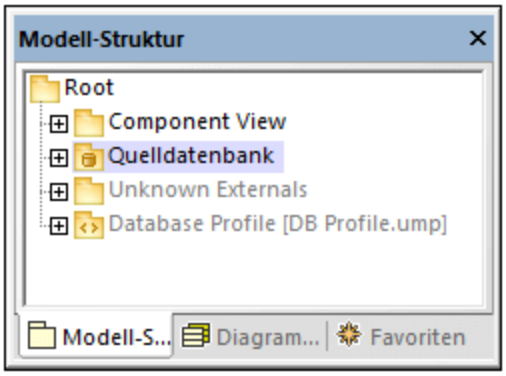
1. Klicken Sie im Menü **Projekt** auf **SQL-Datenbank importieren** und befolgen Sie die Anweisungen des Assistenten, um eine Verbindung zur Microsoft Access-Quelldatenbank (**Nanonull.mdb**) herzustellen. Nähere Informationen dazu finden Sie unter Herstellen einer Verbindung zu einer Datenbank 577.
2. Wenn Sie aufgefordert werden, einen Namen für die Datenquelle anzugeben, geben Sie ihr einen beschreibenden Namen (z.B. "Quelldatenbank").



3. Wählen Sie die Datenbankobjekte aus, die in das Modell importiert werden sollen, und klicken Sie auf **Fertig stellen**.



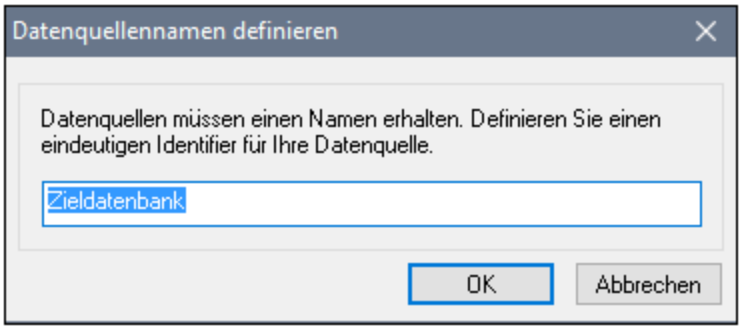
Beachten Sie, dass im Fenster "Modell-Struktur" nun unter dem Paket "Root" ein Paket namens "Quelldatenbank" zur Verfügung steht.



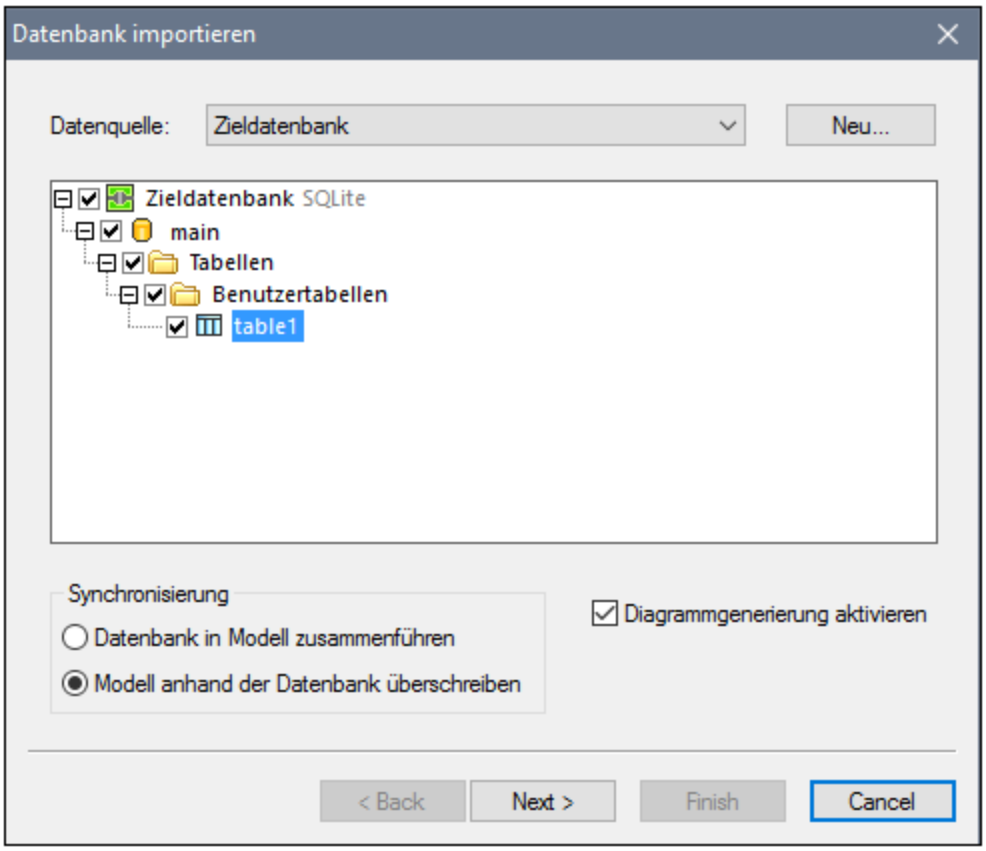
Schritt 2: Import der Zieldatenbank in UModel

1. Klicken Sie im Menü **Projekt** auf **SQL-Datenbank importieren** und befolgen Sie die Anweisungen des Assistenten, um eine Verbindung zur SQLite-Zieldatenbank (**Nanonull.sqlite**) herzustellen.
2. Wenn Sie aufgefordert werden, einen Namen für die Datenquelle anzugeben, geben Sie ihr einen beschreibenden Namen (z.B. "Zieldatenbank").

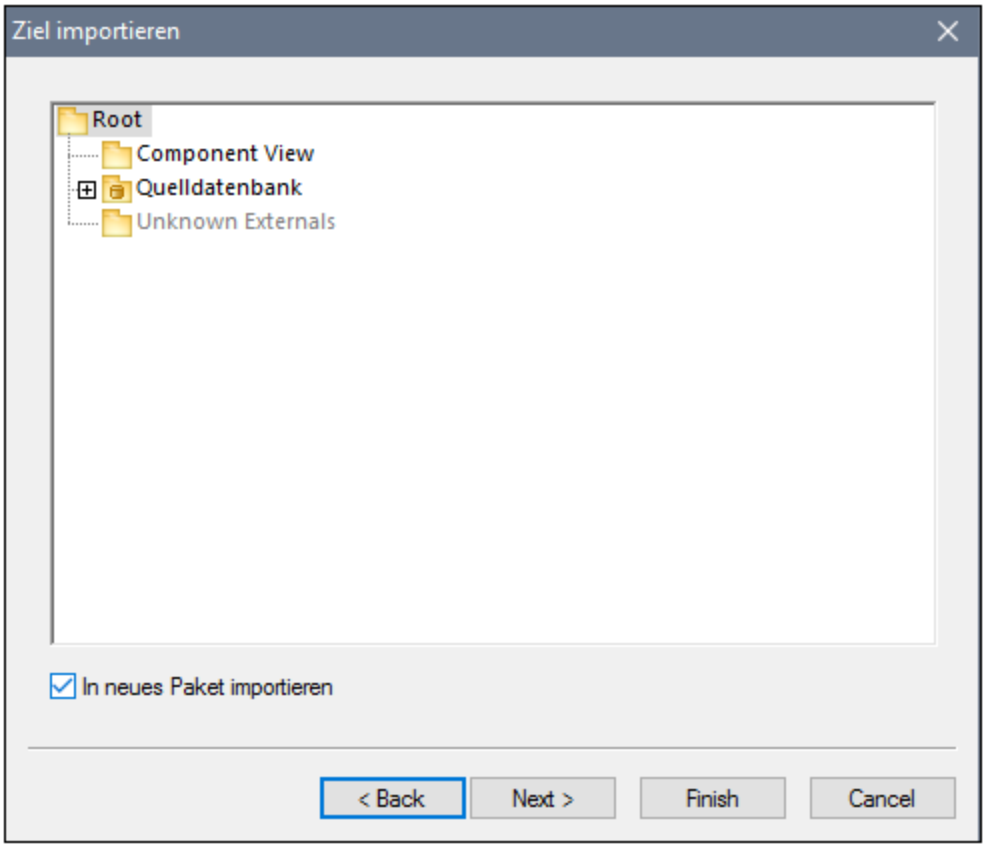
© 2016-2022 Altova GmbH



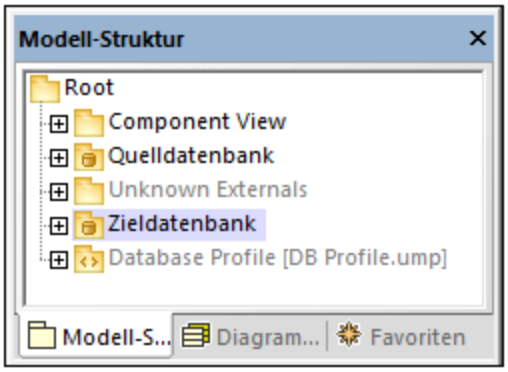
3. Wählen Sie die Datenbankobjekte, die Sie in das Modell importieren möchten, aus und klicken Sie auf **Weiter**.



4. Wenn Sie aufgefordert werden, ein Zielpaket auszuwählen, aktivieren Sie das Kontrollkästchen **In neues Paket importieren** und klicken Sie auf **Fertig stellen**.



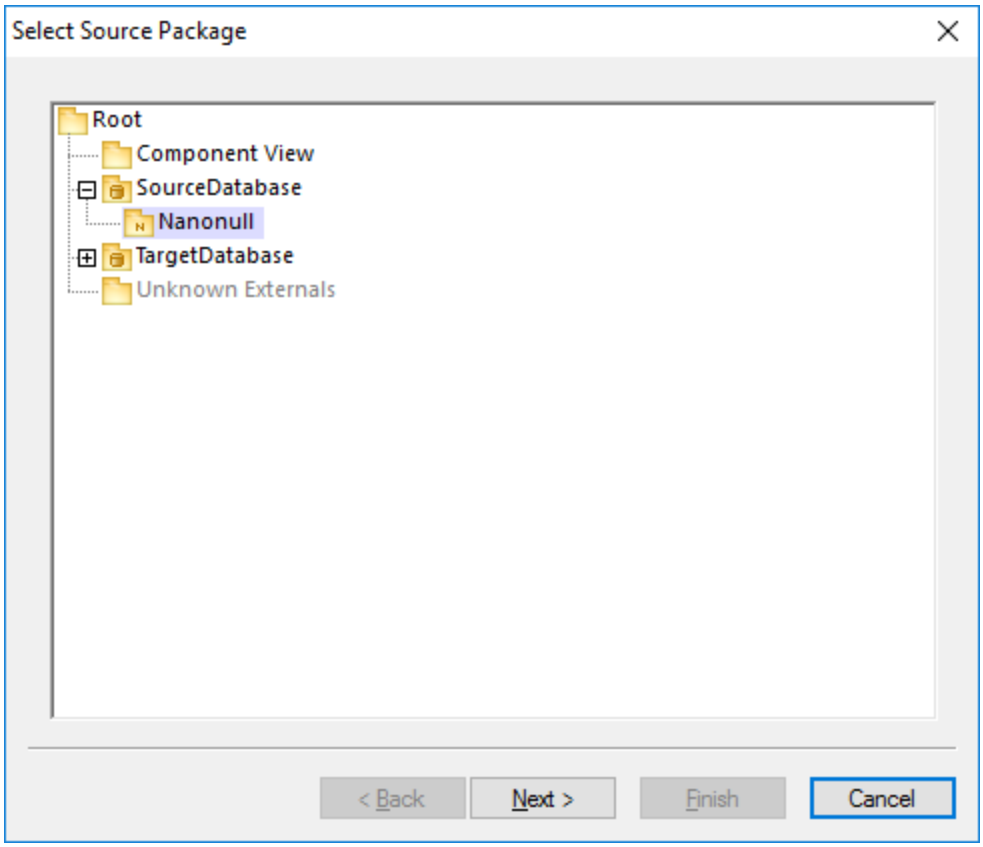
Daraufhin wird im Fenster "Modell-Struktur" unterhalb des Root-Pakets ein neues Paket namens "Zieldatenbank" hinzugefügt.



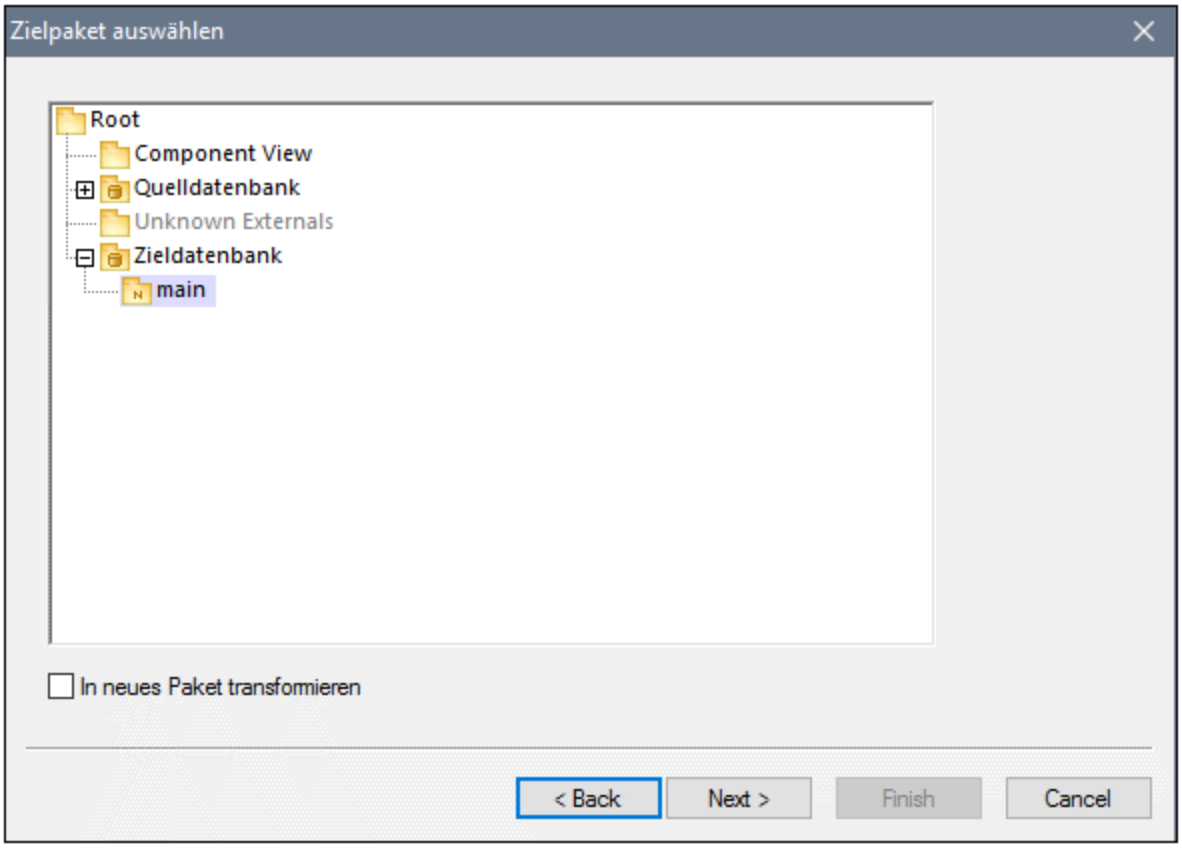
Schritt 3: Ausführung der Modelltransformation von der Quell- in die Zieldatenbank

1. Klicken Sie im Menü **Projekt** auf **Modelltransformation**.
2. Wählen Sie im Dialogfeld "Quellpaket auswählen" "Quelldatenbank / Nanonull" als Paket aus und klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH

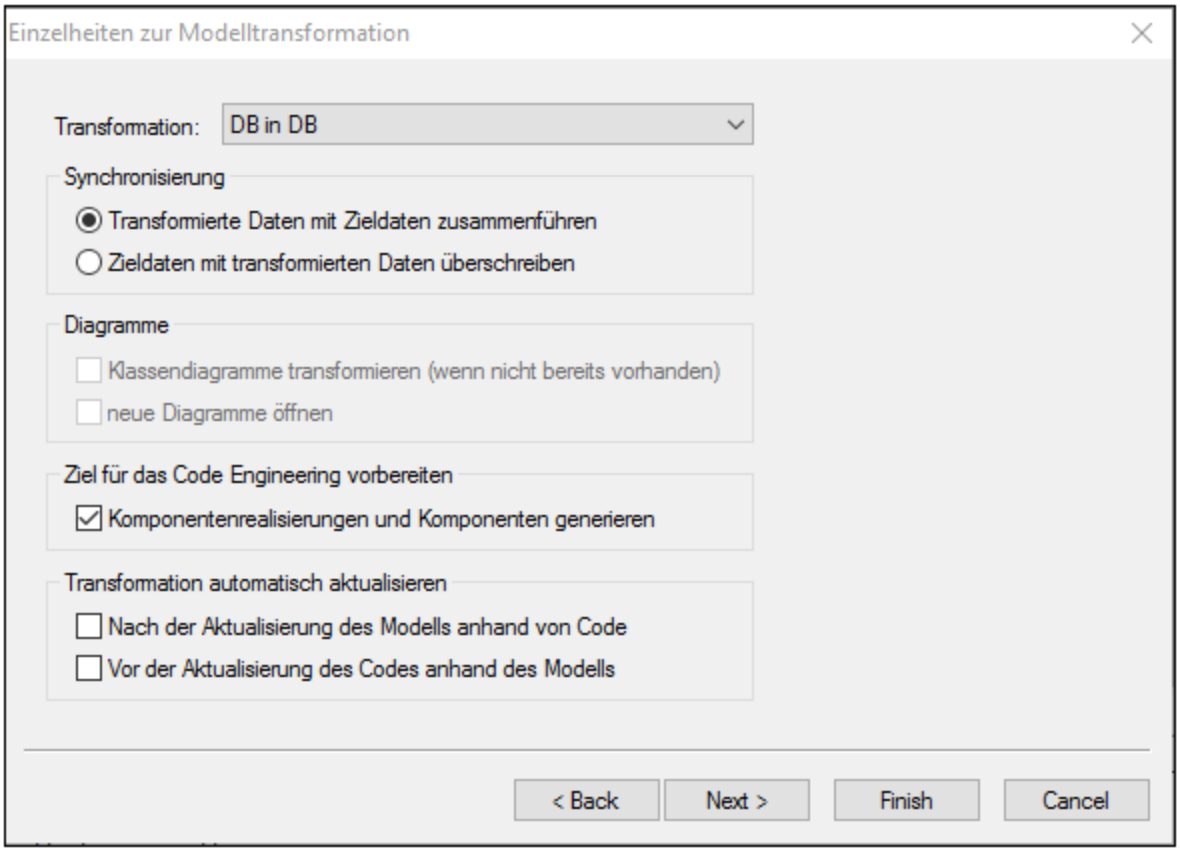


3. Wählen Sie im Dialogfeld "Zielpaket auswählen" "Zieldatenbank / main" als Paket aus und klicken Sie auf **Weiter**.

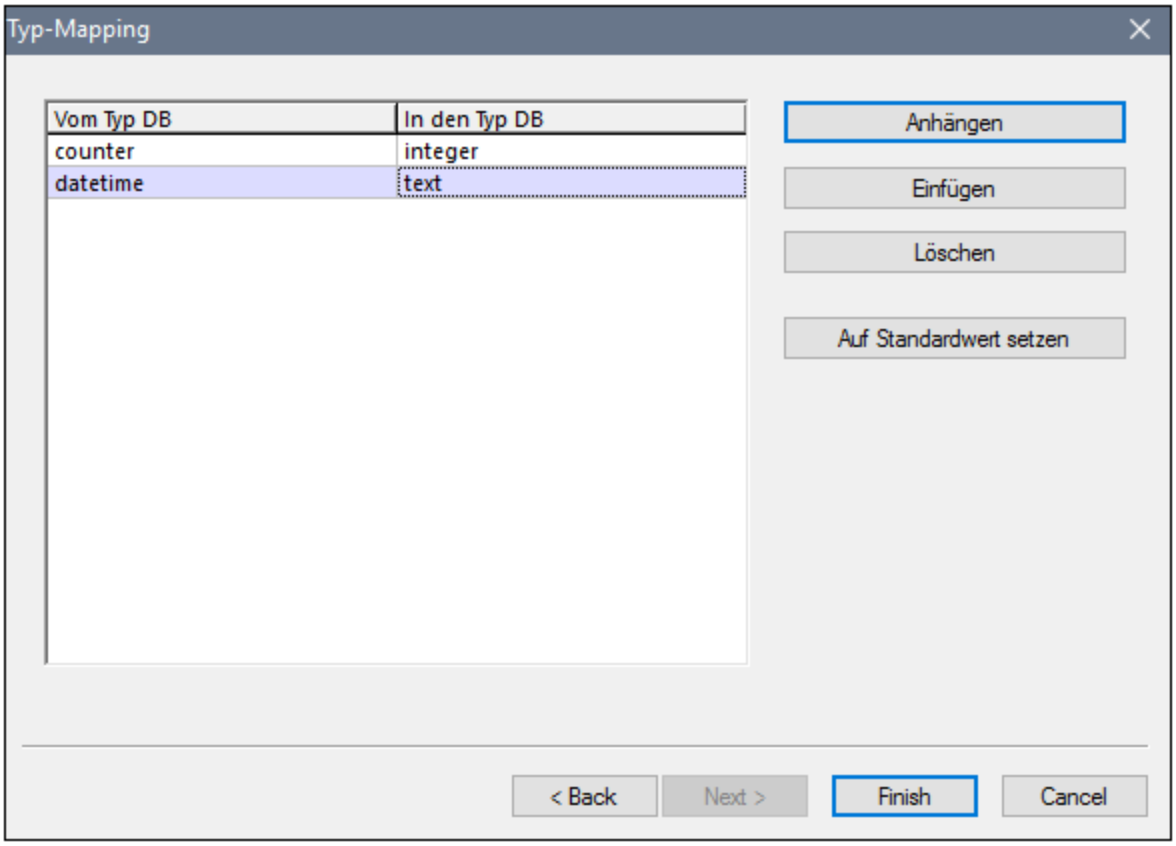


4. Wählen Sie im Dialogfeld "Einzelheiten zur Modelltransformation" als Transformationstyp **DB in DB** aus und klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH

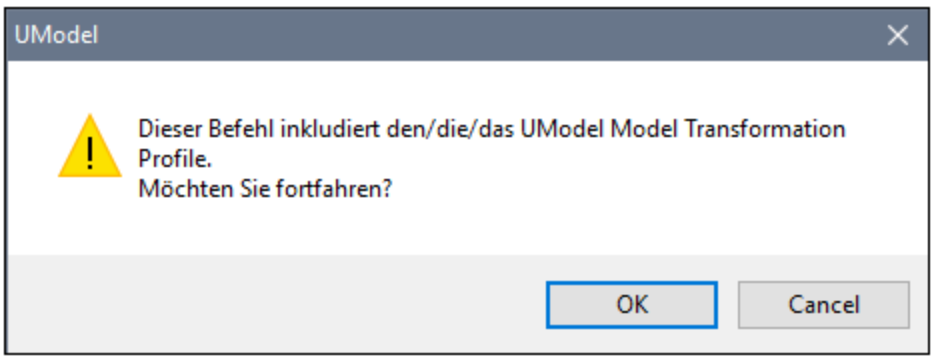


5. Überprüfen Sie im Dialogfeld "Typ-Mapping" die Datentypen und ändern Sie diese nach Bedarf. In diesem Beispiel wollen wir nur einige Microsoft Access-spezifische Datentypen, die in SQLite nicht existieren, mappen (siehe Abb. unten):



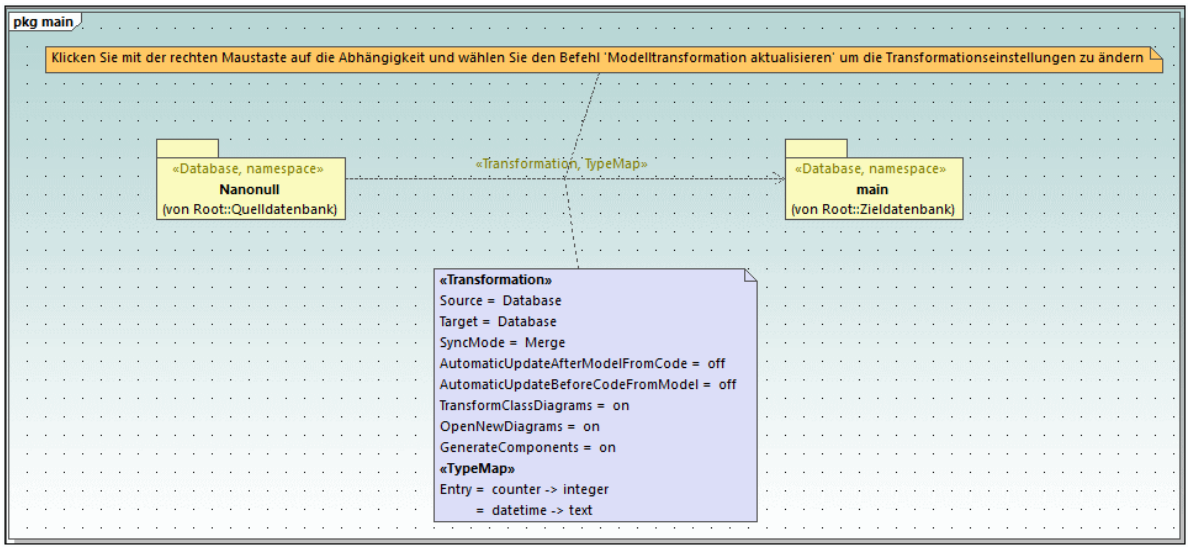
Im Allgemeinen sollte sichergestellt werden, dass die linke Spalte einen mit der Quelldatenbank kompatiblen Datentyp und die rechte Spalte einen mit der Zieldatenbank kompatiblen Datentyp enthält. Um neue Mappings hinzuzufügen oder zu löschen, klicken Sie auf die Schaltflächen **Anhängen**, **Einfügen** und **Löschen**.

6. Klicken Sie auf **Fertig stellen**. Bestätigen Sie die Meldung, die nun angezeigt wird, mit **OK**.



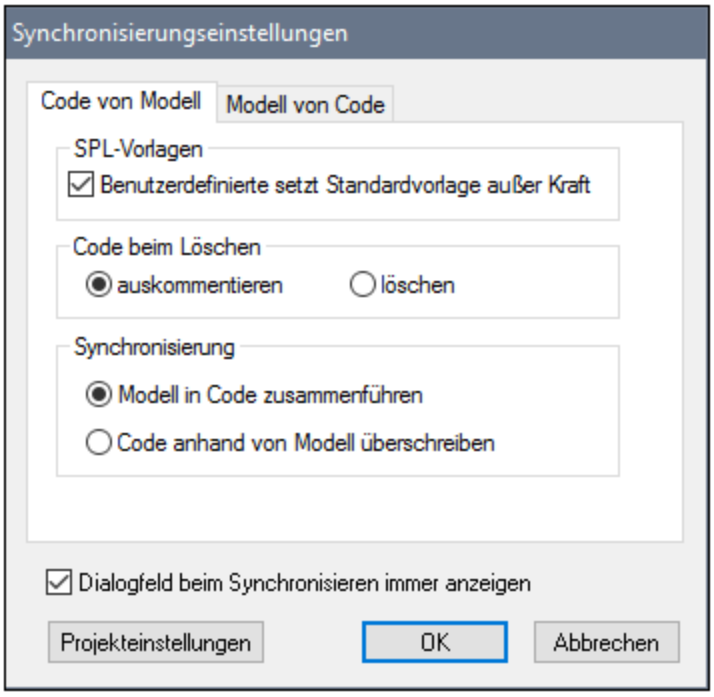
Es wird nun ein Abhängigkeitsdiagramm generiert, in dem Sie alle der zuvor definierten Einstellungen, darunter auch die Datentyp-Mappings überprüfen (und gegebenenfalls ändern können). In diesem Beispiel übernehmen wir die Standardeinstellungen unverändert.

© 2016-2022 Altova GmbH

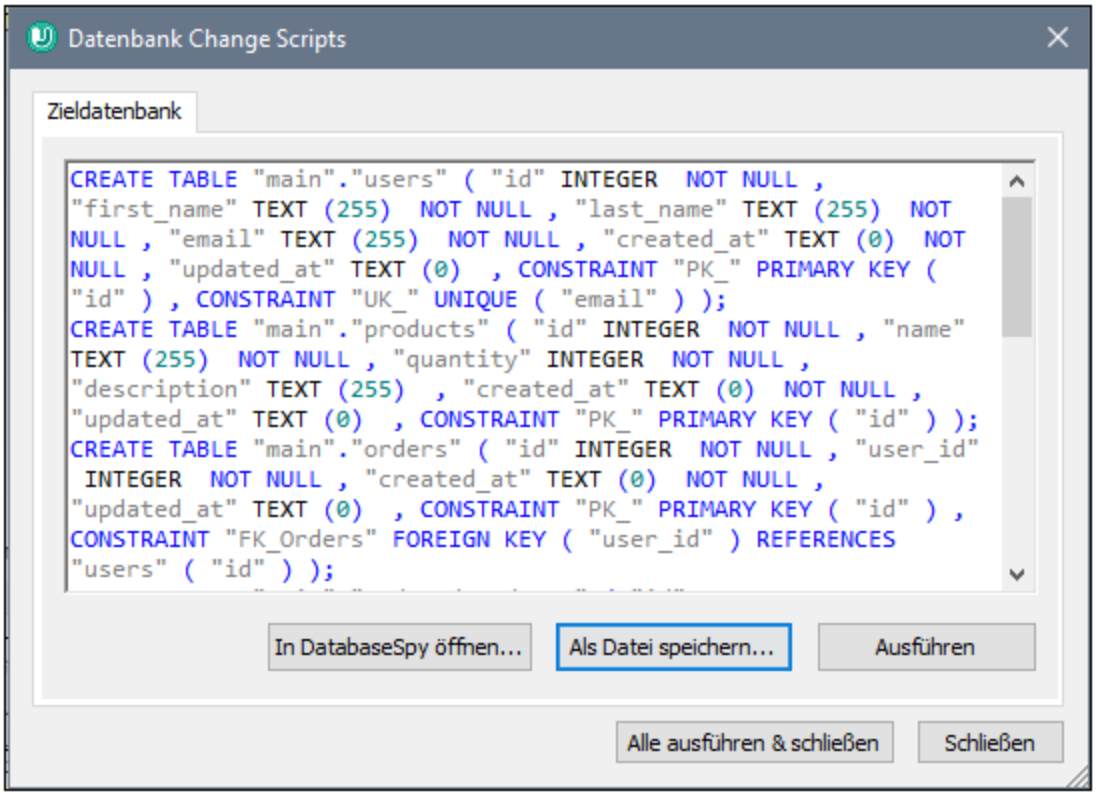


Schritt 4: Zusammenführen von Programmcode aus dem UModel-Projekt

1. Klicken Sie im Menü **Projekt** auf **Merge Programmcode aus UModel-Projekt**.
2. Belassen Sie die Standardeinstellungen unverändert und klicken Sie auf OK.



Daraufhin wird ein Datenbankaktualisierungsskript generiert und, wie unten gezeigt, in einem Dialogfeld angezeigt. Sie können das Skript nun direkt in UModel ausführen oder es in einer Datei speichern. Wenn Sie Altova DatabaseSpy installiert haben, können Sie das Skript auch in DatabaseSpy, das bessere Datenbankverwaltungsfunktionen hat, öffnen und ausführen.



Es wird empfohlen, das generierte Skript unbedingt zu überprüfen und gegebenenfalls zu ändern, bevor Sie es an der Zieldatenbank ausführen.

Wenn eine Quelldatenbank Objektnamen (wie z.B. Indizes oder Sekundärschlüssel) enthält, die auf Datenbankebene nicht eindeutig sind, kann das Datenbankaktualisierungsskript nicht erfolgreich ausgeführt werden. So könnte eine Microsoft Access-Datenbank etwa mehrere Indizes mit demselben Namen enthalten. Wenn die Zieldatenbank doppelt vorhandene Namen für Indizes nicht akzeptiert, müssen Sie das Aktualisierungsskript bearbeiten, sodass alle erforderlichen Objektnamen eindeutig sind.

Eventuell müssen Sie das Skript auch aktualisieren, um die Größe von Spalten den Anforderungen der Zieldatenbank entsprechend zu ändern.

Nachdem Sie das Skript ausgeführt haben (entweder direkt in UModel oder extern in einem Tool wie z.B. DatabaseSpy), werden die erforderlichen Tabellen, Spalten sowie Indizes und Constraints in der SQLite-Zieldatenbank neu erstellt. Beachten Sie, dass SQLite (Version 3.6.19) die von der SQL-Anweisung bereitgestellten Namen von Sekundärschlüssel-Constraints akzeptiert, aber keine Möglichkeit bietet, diese aus der Datenbank abzurufen (Sekundärschlüssel-Constraints werden mit einem beliebigen Namen, der nicht der tatsächliche Name ist, abgerufen). Um sicherzustellen, dass in Ihrem Datenbankmodell die tatsächlichen Objektnamen, so wie sie von der Datenbank bereitgestellt werden, angezeigt werden, führen Sie eine umgekehrte Aktualisierung vom Modell in die Datenbank durch. Führen Sie zu diesem Zweck den Menübefehl **Projekt | Merge UModel-Projekt aus Programmcode**. Daraufhin wird das Modell aktualisiert, sodass die Objektnamen, wie sie von der Datenbank bereitgestellt werden, angezeigt werden.

© 2016-2022 Altova GmbH

8 Generieren von UML-Dokumentation

Altova Website: UML-Projektdokumentation

Mit dem Befehl **Projekt | Dokumentation generieren** wird detaillierte Dokumentation zu Ihrem UML-Projekt in den Formaten HTML, Microsoft Word, RTF oder PDF generiert. Mit diesem Befehl generierte Dokumentation kann beliebig geändert und verwendet werden; Sie benötigen dazu keine Genehmigung von Altova.

Anmerkungen

- Um Dokumentation im PDF-Format zu generieren oder die generierte Dokumentation anzupassen, muss Altova StyleVision (https://www.altova.com/de/stylevision) auf Ihrem Rechner installiert und lizenziert sein.
- Um Dokumentation im Format MS Word zu generieren, muss MS Word (Version 2000 oder höher) installiert sein.

Die Dokumentation wird für die Modellierungselemente generiert, die Sie im Dialogfeld "Dokumentation generieren" ausgewählt haben. Sie können entweder das vordefinierte Design oder ein benutzerdefiniertes StyleVision Power Stylesheet (SPS) für das Design verwenden. Bei Verwendung eines StyleVision Power Stylesheet können Sie die Ausgabe der generierten Dokumentation anpassen, siehe Anpassen der Ausgabe mit StyleVision 354.

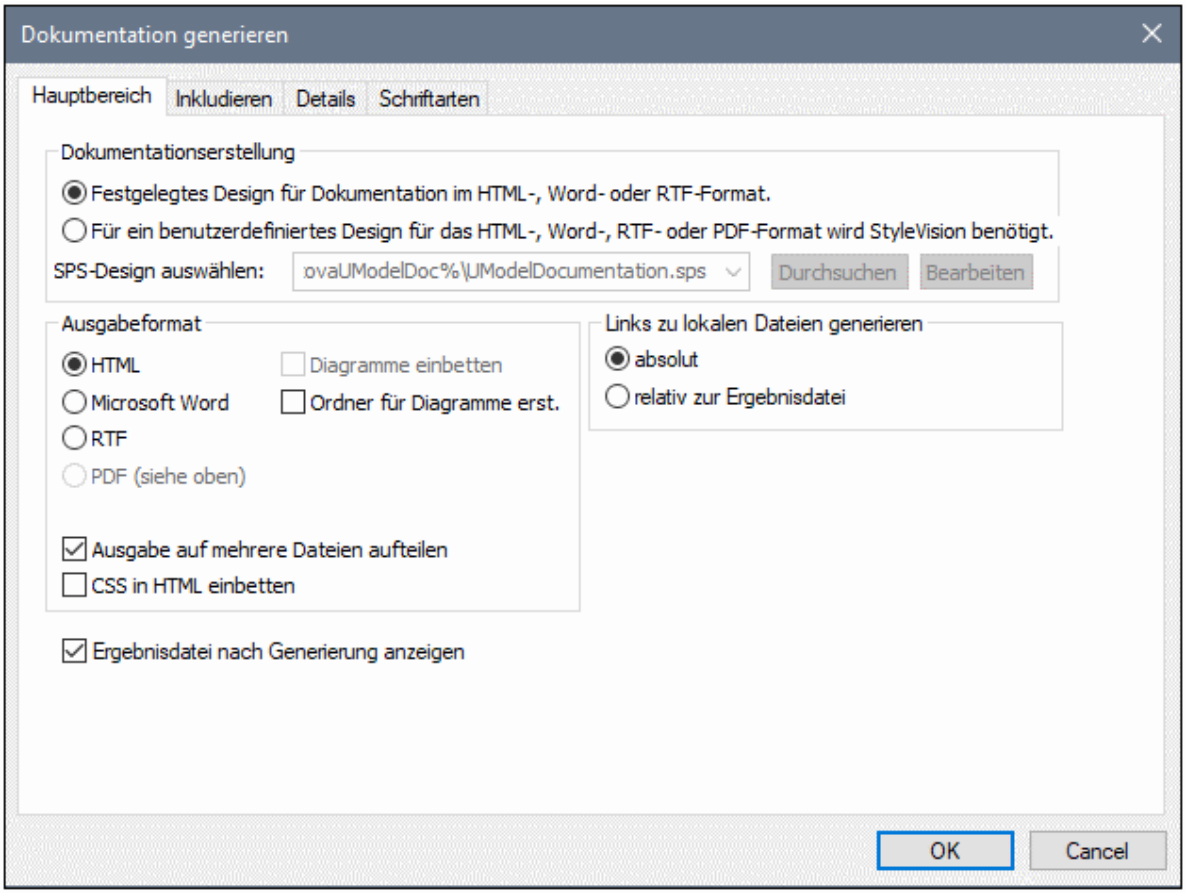
Sie können auch einzelne Teile eines Projekts dokumentieren. Rechtsklicken Sie dazu in der Modellstruktur auf ein Element (bzw. mit Strg + Klick auf mehrere Elemente) und wählen Sie den Befehl **Dokumentation generieren**. Beim Element kann es sich um einen Ordner, eine Klasse, eine Schnittstelle, usw. handeln. Die Dokumentationsoptionen sind in beiden Fällen dieselben.

Miteinander in Beziehung stehende Elemente werden in der generierten Ausgabe durch Hyperlinks miteinander verbunden, sodass Sie von Komponente zu Komponente navigieren können. In der Dokumentation sind auch alle manuell erstellten Hyperlinks enthalten.

Wenn Ihr Projekt UModel-Profile (wie C#, Java, VB.NET, usw.) enthält, werden diese in die generierte Dokumentation inkludiert, wenn die Option **Inkludierte Unterprojekte** auf dem Register **Inkludieren** ausgewählt ist, siehe Optionen zur Generierung von Dokumentation 349.

So generieren Sie Dokumentation:

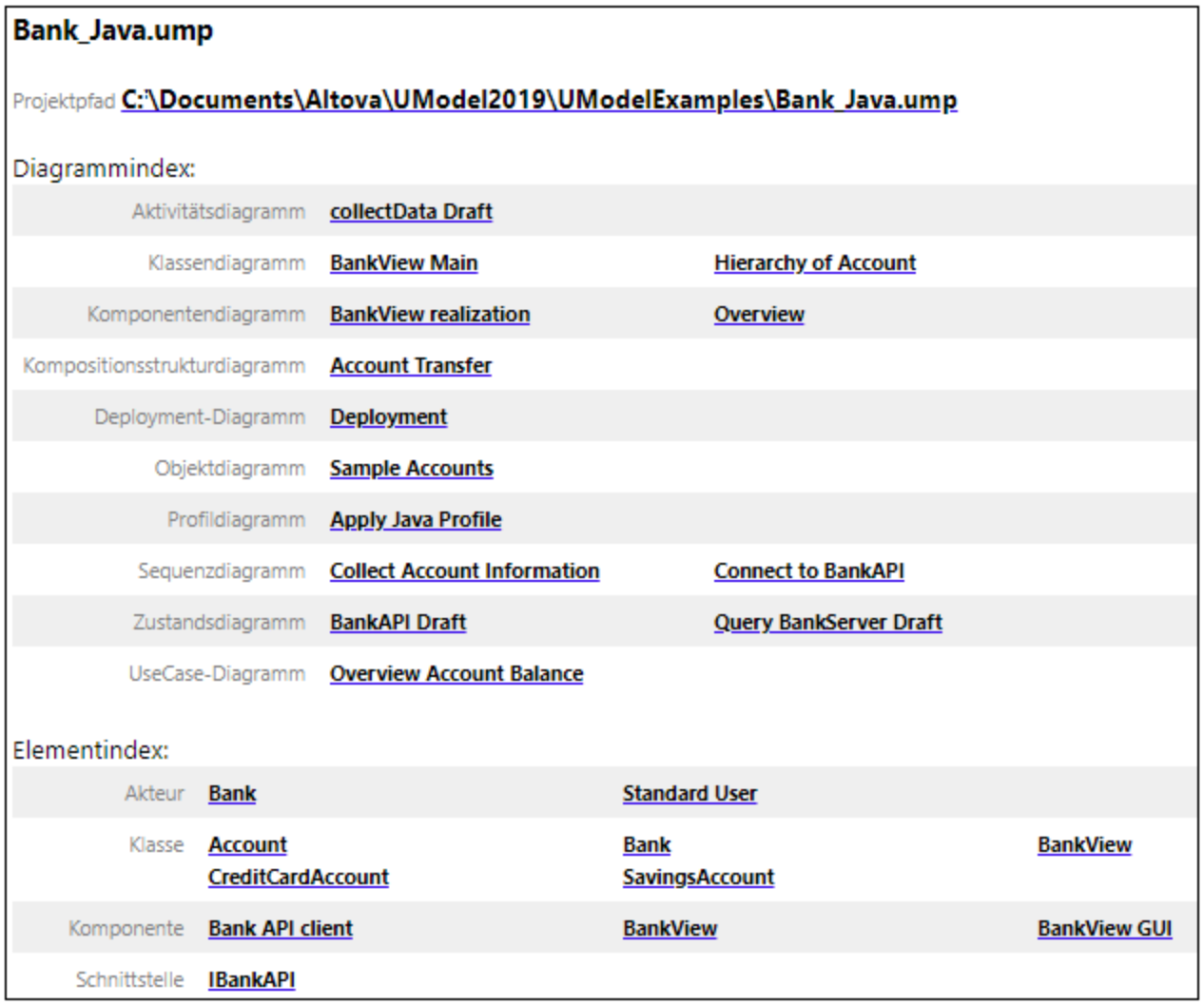
1. Öffnen Sie ein Projekt (z.B. **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_Java.ump**).
2. Klicken Sie im Menü **Projekt** auf **Dokumentation generieren.**



3. Wählen Sie ein Ausgabeformat aus (HTML, Word, RTF, PDF).
4. Passen Sie optional die Generierungsoptionen an, siehe Optionen zur Generierung von Dokumentation 349.
5. Klicken Sie auf **OK** und wählen Sie einen Zielordner für die Ausgabe.

In den folgenden Abbildungen sehen Sie einen Ausschnitt aus einer UModel-Dokumentation mit vordefiniertem Design, die anhand der Projektdatei **Bank_Java.ump** generiert wurde.

© 2016-2022 Altova GmbH



Bank_Java.ump

Projektpfad **C:\Documents\Altova\UModel2019\UModelExamples\Bank_Java.ump**

Diagrammindex:

Aktivitätsdiagramm	collectData Draft	
Klassendiagramm	BankView Main	Hierarchy of Account
Komponentendiagramm	BankView realization	Overview
Kompositionsstrukturdiagramm	Account Transfer	
Deployment-Diagramm	Deployment	
Objektdiagramm	Sample Accounts	
Profildiagramm	Apply Java Profile	
Sequenzdiagramm	Collect Account Information	Connect to BankAPI
Zustandsdiagramm	BankAPI Draft	Query BankServer Draft
UseCase-Diagramm	Overview Account Balance	

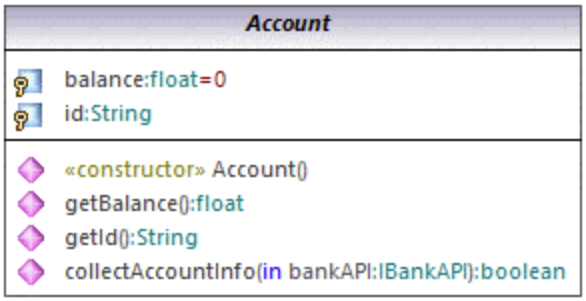
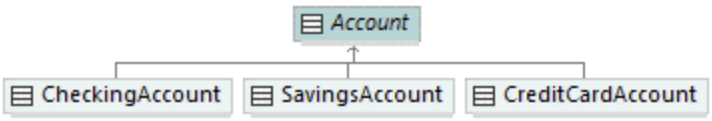
Elementindex:

Akteur	Bank	Standard User	
Klasse	Account CreditCardAccount	Bank SavingsAccount	BankView
Komponente	Bank API client	BankView	BankView GUI
Schnittstelle	IBankAPI		

Wie oben gezeigt, enthält die generierte Dokumentation im oberen Bereich der HTML-Datei einen Diagramm- und Elementindex (mit Links).

In der Abbildung unten sehen Sie einen Ausschnitt aus der generierten Dokumentation für die Klasse `Account`. Beachten Sie, dass die einzelnen Mitglieder in Klassendiagrammen auch mit Hyperlinks zu ihren Definitionen versehen sind. Wenn Sie z.B. auf eine Eigenschaft oder Operation klicken, gelangen Sie zu ihrer Definition. Auch die Hierarchieklassen sowie unterstrichener Text sind mit Hyperlinks versehen.

Klasse **Account**

Diagramm	
Hierarchie	
Owner	**bankview**
Eigenschaften	Qualifizierter Name Design View::BankView::com::altova::bankview::Account Sichtbarkeit public leaf false abstrakt true isFinalSpecialization false aktiv false Codedateiname Account.java Codedateipfad C:\UML_Bank_Sample\JavaCode\com\altova\bankview\Account.java «annotations» false «static» false «strictfp» false

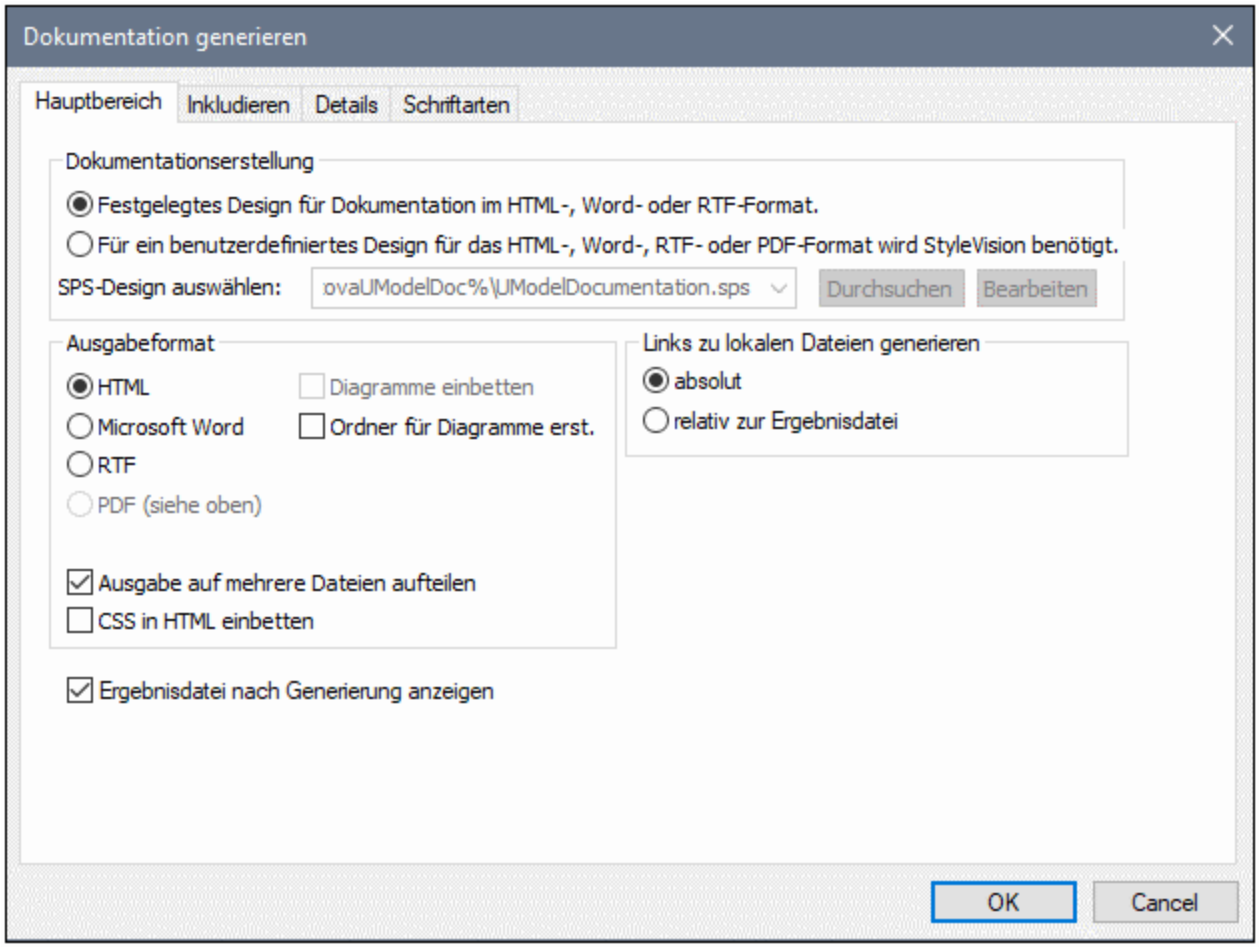
© 2016-2022 Altova GmbH

8.1 Optionen zur Generierung von Dokumentation

Vor der Generierung von Dokumentation anhand von UModel-Projekten können Sie verschiedene Option, wie unten beschrieben, definieren. Die Optionen sind nach dem Register, auf dem sie im Dialogfeld "Dokumentation generieren" angezeigt werden, angeordnet.

Register "Hauptbereich"

Das Register **Hauptbereich** enthält die allgemeinen Dokumentationsgenerierungsoptionen.



Dokumentationserstellung:

- Wählen Sie die Option **Festgelegtes Design ...verwenden,** um das vordefinierte UModel-Dokumentationdesign zu verwenden.
- Wählen Sie die Option **Für ein benutzerdefiniertes Design..,**. um Dokumentation zu generieren, die mit Hilfe eines in StyleVision erstellten StyleVision Power Stylesheet (.sps-Datei) formatiert wird. Anmerkung: Für diese Option muss Altova StyleVision installiert sein, siehe Anpassen der Ausgabe mit StyleVision 354.
- Klicken Sie auf **Durchsuchen,** um zu einer vordefinierten Stylesheet-Datei zu navigieren.

- Klicken Sie auf **Bearbeiten**, um StyleVision zu starten und die ausgewählte Stylesheet-Datei in einem StyleVision-Fenster zu öffnen.

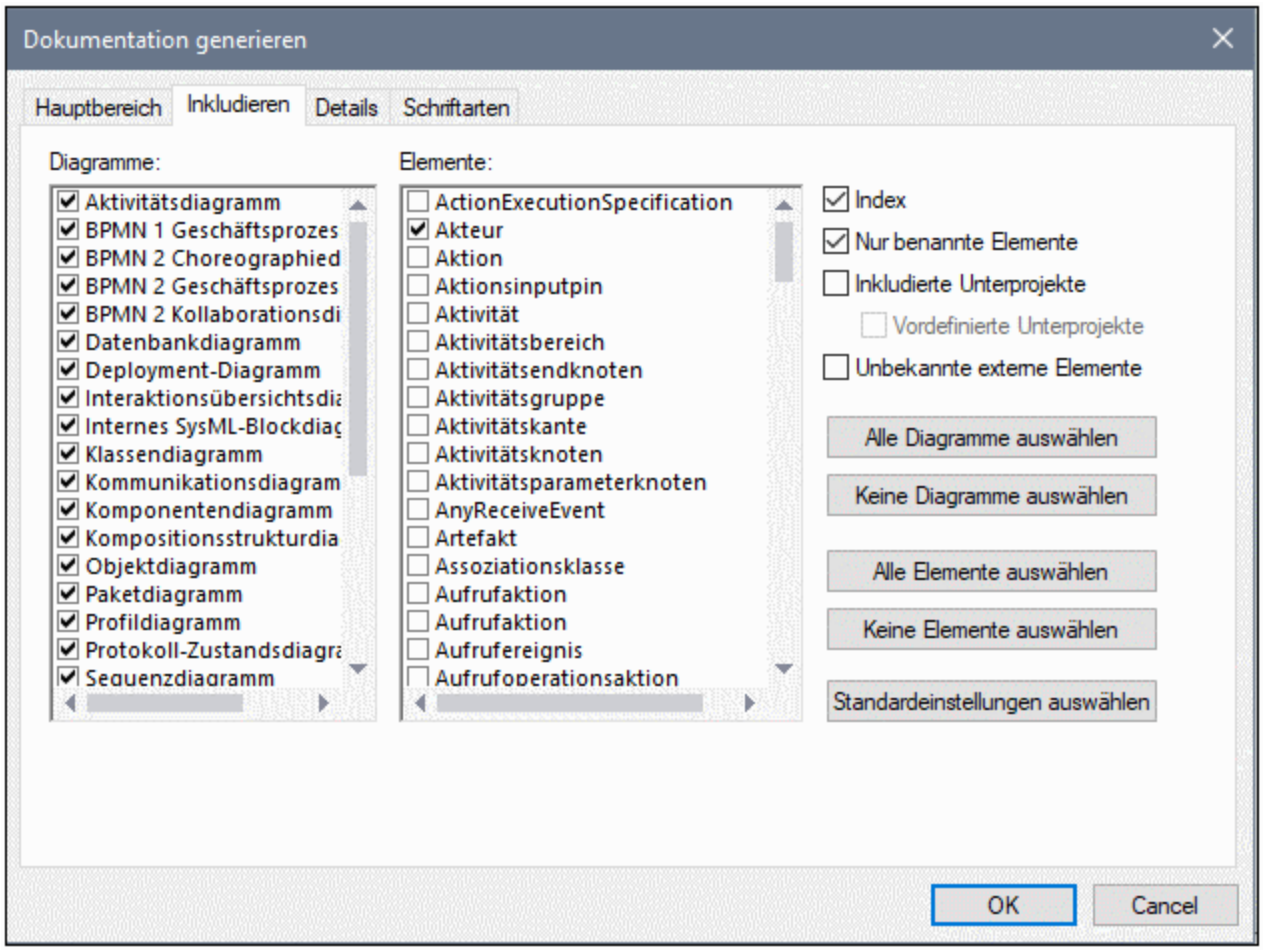
Ausgabeformat:

- Als Ausgabeformat kann eines der folgenden definiert werden: HTML, Microsoft Word, RTF oder PDF. Microsoft Word-Dokumente werden mit der Dateierweiterung .doc angelegt, wenn sie anhand eines festgelegten Designs erstellt werden, und mit der Dateierweiterung .docx, wenn sie anhand eines StyleVision Power Stylesheet generiert werden. Für das Ausgabeformat PDF muss Altova StyleVision installiert sein.
- Mit der Option **Ausgabe auf mehrere Dateien aufteilen** wird für jedes Modellierungselement (Klasse, Schnittstelle, Diagramm, usw.) eine Ausgabedatei erstellt. Deaktivieren Sie dieses Kontrollkästchen, um eine einzige globale Datei mit allen Modellierungselementen zu generieren.
- Mit der Option **CSS in HTML einbetten** können Sie den generierten CSS-Code in die HTML-Dokumentation einbetten. Deaktivieren Sie dieses Kontrollkästchen, damit die CSS-Datei extern bleibt.
- Die Option **Diagramme einbetten** ist für die Ausgabeoptionen Microsoft Word und RTF aktiviert. Wenn dieses Kontrollkästchen aktiviert ist, werden Diagramm in die generierte Datei eingebettet. Diagramme werden als PNG-Dateien erstellt und über Objektlinks in der Ergebnisdatei angezeigt.
- Mit der Option **Ordner für Diagramme erstellen** wird unterhalb des ausgewählten Ausgabeordners ein Unterordner erstellt, der alle Diagramme enthält.
- Die Option **Ergebnisdatei nach Generierung anzeigen** ist für alle Ausgabeformate aktiviert. Wenn dieses Kontrollkästchen aktiviert ist, wird die generierte Hauptdatei (bei HTML-Dateien) im Standard-Browser, (bei Word-Dateien) in Microsoft Word oder (für .pdf- oder .rtf-Dateien) in der Standardapplikation für angezeigt.
- Über die Option **Links zu lokalen Dateien generieren** können Sie festlegen, ob die generierten Links absolute oder relative Links zur Ausgabedatei sein sollen.

Register "Inkludieren"

Auf dem Register **Inkludieren** können Sie auswählen, welche Diagramme und Modellierungselemente in der Dokumentation aufscheinen sollen.

© 2016-2022 Altova GmbH



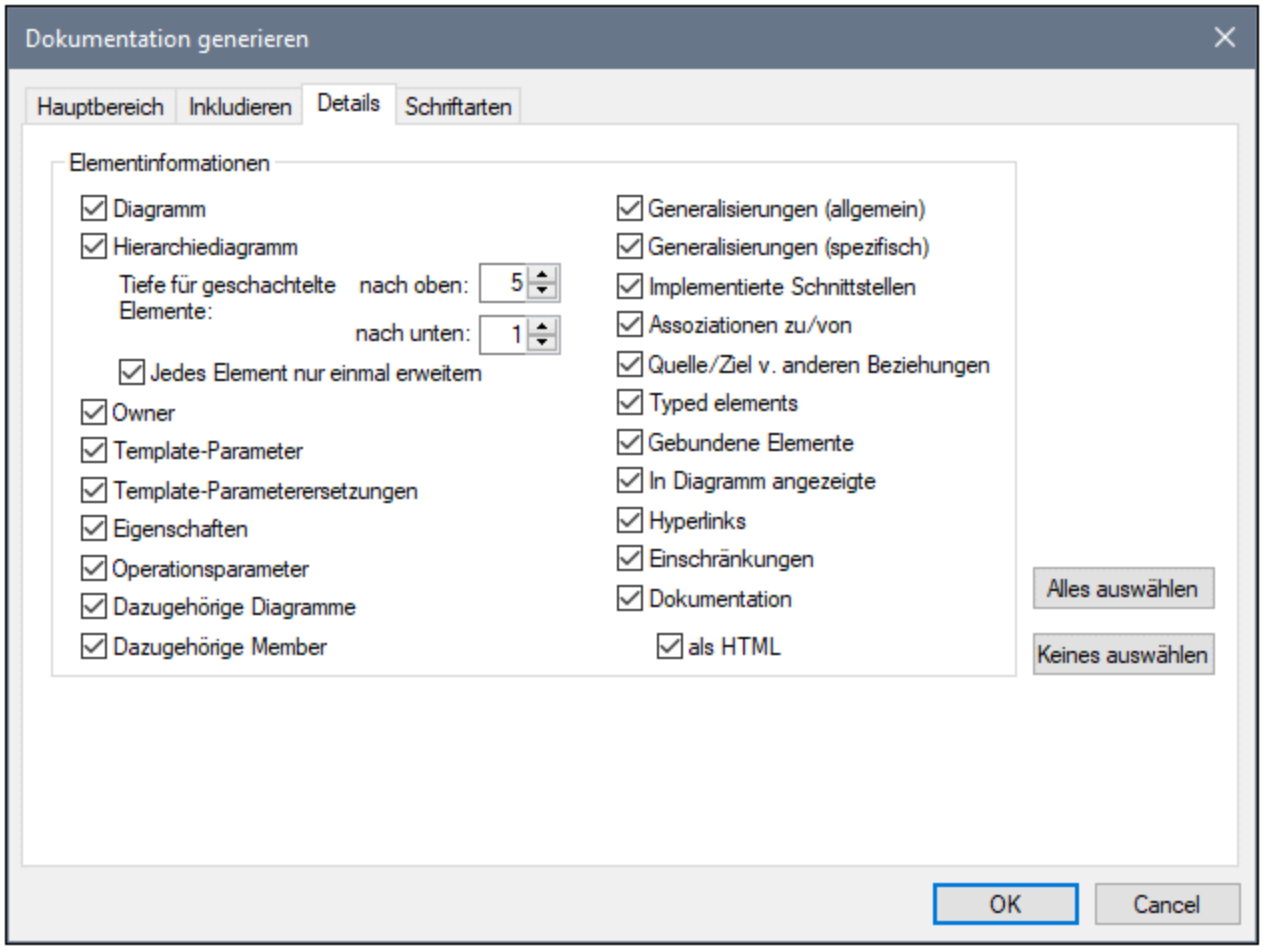
Um zu verhindern, dass Unterprojekte oder Profile dokumentiert werden, deaktivieren Sie das Kontrollkästchen **Inkludierte Unterprojekte**. Bedenken Sie, dass Elemente oder Diagramme in Unterprojekten nicht in die generierte Dokumentation inkludiert werden, wenn dieses Kontrollkästchen deaktiviert ist. Aktivieren Sie das Kontrollkästchen **Vordefinierte Unterprojekte,** um vordefinierte UModel-Profile wie C#- oder Java-Profile zu inkludieren. Beachten Sie jedoch, dass das Generieren von Dokumentation anhand von vordefinierten Projekten sehr lange dauert. **Unbekannte externe Elemente** bezieht sich auf Elemente, deren Art nicht identifiziert werden konnte. Dies kommt normalerweise dann vor, wenn Sie Quellcode in UModel importieren, ohne zuerst die vordefinierten Unterprojekte für diese Sprache oder Sprachversion zu inkludieren, siehe Inkludieren von Unterprojekten 174.

Register "Details"

Auf dem Register **Details** können Sie die Elementdetails auswählen, die in der Dokumentation aufscheinen sollen.

- Wenn Sie den Text von **XML-Tags** in Ihre Dokumentation importieren möchten, deaktivieren Sie unter der Option **Dokumentation** die Option **als HTML**.
- In den Feldern **Nach oben** / **Nach unten** können Sie die Verschachtelungstiefe festlegen, die im Hierarchiediagramm oberhalb / unterhalb der aktuellen Klasse angezeigt werden soll.

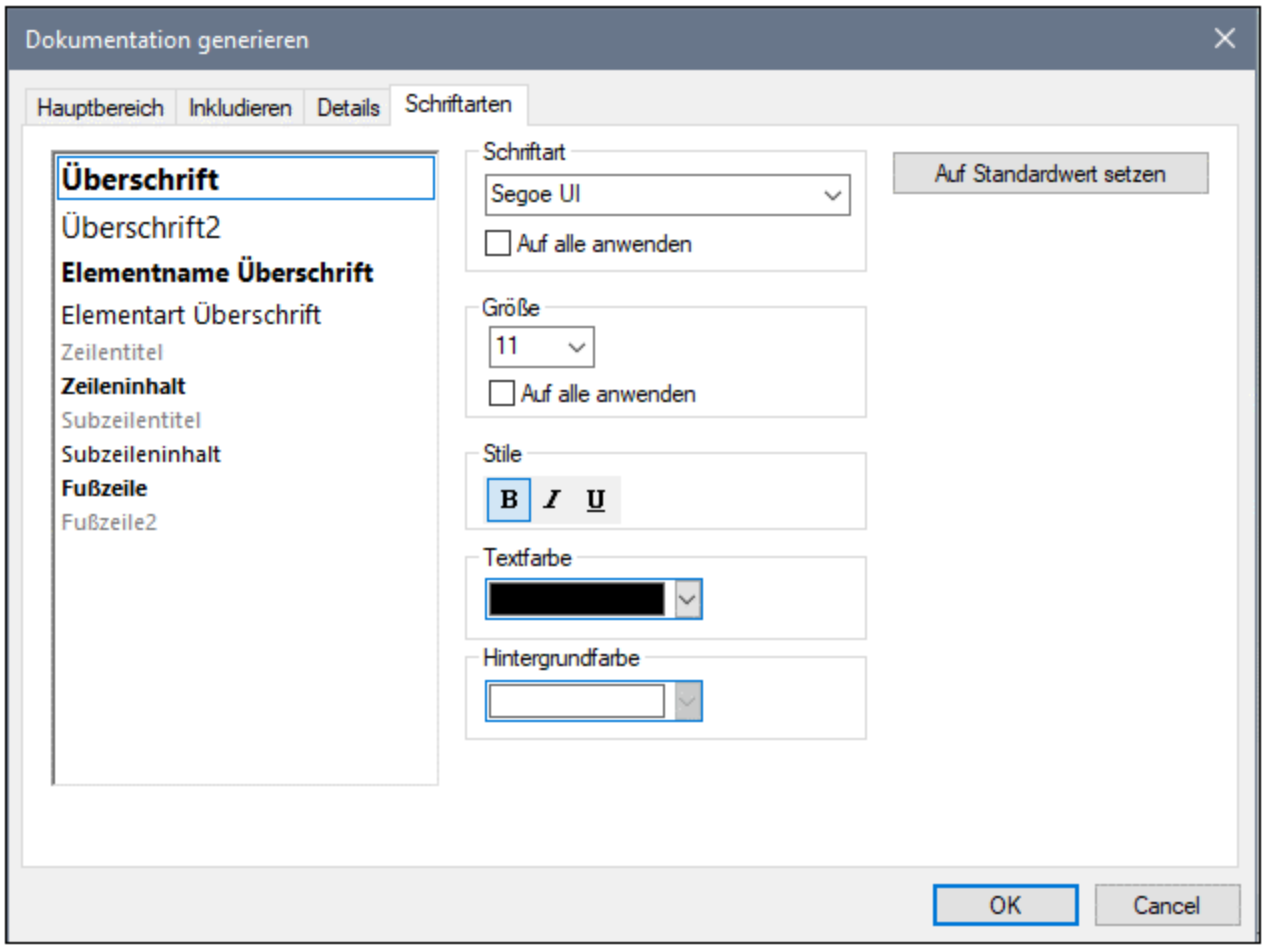
- Mit der Option **Jedes Element nur einmal erweitern** wird im selben Bild oder Diagramm nur immer jeweils ein Classifier erweitert.



Register "Schriftarten"

Auf dem Register "Schriftarten" können Sie die Schriftarteinstellungen für die verschiedenen Überschriften und Textinhalte anpassen.

© 2016-2022 Altova GmbH

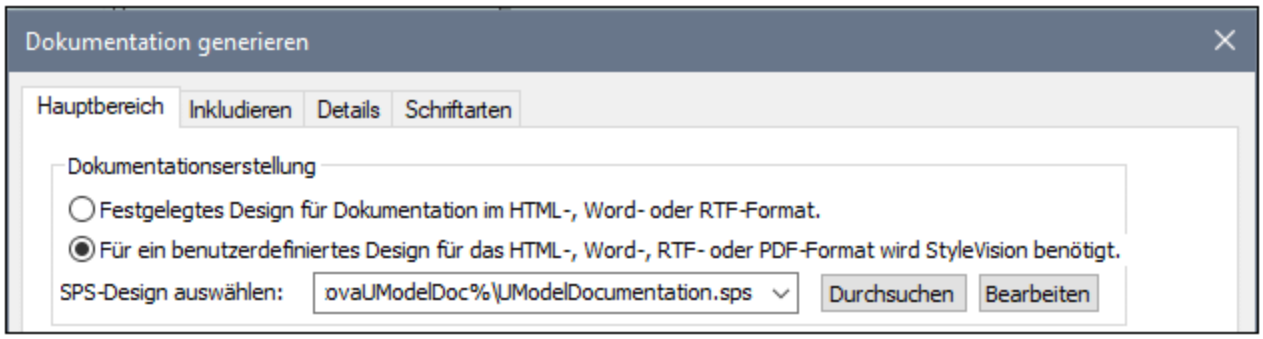

Dokumentation generieren
Hauptbereich
Inkludieren
Details
Schriftarten
Überschrift
Überschrift2
Elementname Überschrift
Elementart Überschrift
Zeilentitel
Zeileninhalt
Subzeilentitel
Subzeileninhalt
Fußzeile
Fußzeile2
Schriftart
Segoe UI
Auf alle anwenden
Auf Standardwert setzen
Größe
11
Auf alle anwenden
Stile
B
I
U
Textfarbe
Hintergrundfarbe
OK
Cancel

8.2 Anpassen der Ausgabe mit StyleVision

Sie können das Design von mit UModel generierter Dokumentation mit Hilfe von StyleVision Power Stylesheet (.sps)-Dateien anpassen. Solche Dateien werden in Altova StyleVision (https://www.altova.com/de/stylevision) generiert. Der Vorteil bei der Generierung der Dokumentation anhand einer .sps-Datei ist, dass Sie völlig freie Hand bei der Gestaltung der Dokumentation haben. Außerdem kann die Dokumentation bei Verwendung einer .sps-Datei auch im PDF-Format generiert werden.

> Damit Sie mit Hilfe von .sps-Dateien Dokumentation generieren können, muss Altova StyleVision auf Ihrem Rechner installiert und lizenziert sein.

Im Lieferumfang von UModel ist eine vordefinierte .sps-Datei inkludiert, die unter folgendem Pfad zur Verfügung steht: **C:\Benutzer\<Benutzername\Dokumente\UModel2023\Documentation\UModel\UModelDocumentation.sps**. Um die generierte Dokumentation mit Hilfe einer benutzerdefinierten .sps-Datei zu formatieren, wählen Sie bei der Generierung von Dokumentation diese Option aus, z.B.:

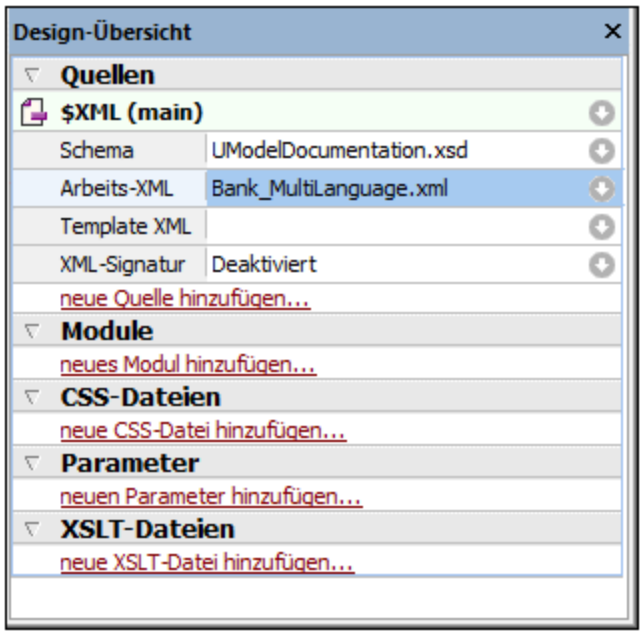


Als Ausgangsbasis für Ihr benutzerdefiniertes Stylesheet erstellen Sie eine Kopie der Standard-.sps-Datei **UModelDocumentation.sps** und bearbeiten Sie diese in StyleVision. Sie können z.,B. das vorhandene Format ändern oder Links und Bilder zum Design hinzufügen.

Jedem StyleVision Power Stylesheet liegt ein XML-Schema zugrunde. Im Fall von Stylesheets zur Definition des Designs von UModel-generierter Dokumentation befindet sich das Schema unter dem folgenden Pfad: **C:\Benutzer\<Benutzername\Dokumente\UModel2023\Documentation\UModel\UModelDocumentation.xsd**. Beachten Sie, dass die Datei **ModelDocumentation.xsd** die Datei **Documentation.xsd** im übergeordneten Ordner referenziert.

Wenn Sie benutzerdefinierte .sps-Dateien für die UModel-Dokumentation in StyleVision erstellen, muss als Schema die Datei **UModelDocumentation.xsd** verwendet werden. In der Abbildung unten sehen Sie das Fenster "Design-Übersicht" von StyleVision, nachdem Sie die Datei **UModelDocumentation.sps** geöffnet haben. Beachten Sie, dass darin die Schema-Datei **UModelDocumentation.xsd** sowie eine XML-Arbeitsdatei für die Design-Vorschau verwendet werden. Die XML-Arbeitsdatei steht relativ zur Schema-Datei im Unterordner **SampleData** zur Verfügung.

© 2016-2022 Altova GmbH



Eine Anleitung, wie Sie .sps-Dateien bearbeiten, finden Sie in der Dokumentation zu StyleVision (https://www.altova.com/de/documentation).

9 UML-Diagramme

Altova Website: UML-Diagramme

UML-Diagramme werden in zwei große Gruppen eingeteilt: Strukturdiagramme, in denen eine statische Ansicht des Modells zu sehen ist, und Verhaltensdiagramme, in denen die dynamische Ansicht dargestellt wird. UModel unterstützt alle vierzehn Diagramme der UML 2.5-Spezifikation sowie weitere zusätzliche Diagramme.

Zu den Verhaltensdiagrammen[357] gehören Aktivitäts-, Zustands-, Protokoll-Zustands- und Use Case-Diagramme sowie Interaktionsdiagramme Kommunikations-, Interaktionsübersichtsdiagramme, Sequenzdiagramme und Zeitverlaufsdiagramme.

Zu den Strukturdiagrammen[449] gehören: Klassen-, Kompositionsstruktur-, Komponenten-, Deployment-, Objekt- und Paketdiagramme.

Zusätzliche Diagramme[490]: XML-Schema-Diagramme und BPMN-Diagrammen (Geschäftsprozessdiagramme), SysML-Diagramme, Datenbankdiagramme.

Anmerkung:
Mit **Strg+Eingabetaste** können Sie mehrzeilige Beschriftungen für die meisten Modellierungsdiagramme erstellt werden, z.B. Lebenslinienbeschriftungen in Sequenzdiagrammen, Zeitverlaufsdiagrammen; Guard-Bedingungen, Zustandsnamen, Aktivitätsnamen usw.

© 2016-2022 Altova GmbH

9.1 Verhaltensdiagramme

In diesen Diagrammen werden Verhaltensaspekte eines Systems oder Geschäftsvorgangs beschrieben. Sie enthalten eine Untergruppe von Diagrammen zur Darstellung der Wechselbeziehungen zwischen Objekten.

Verhaltensdiagramme

- Aktivitätsdiagramm
- Zustandsdiagramm
- Protokoll-Zustandsdiagramm
- Use Case-Diagramm 402

Eine Untergruppe der Verhaltensdiagramme bilden jene zur Darstellung von Wechselwirkungen zwischen Objekten:

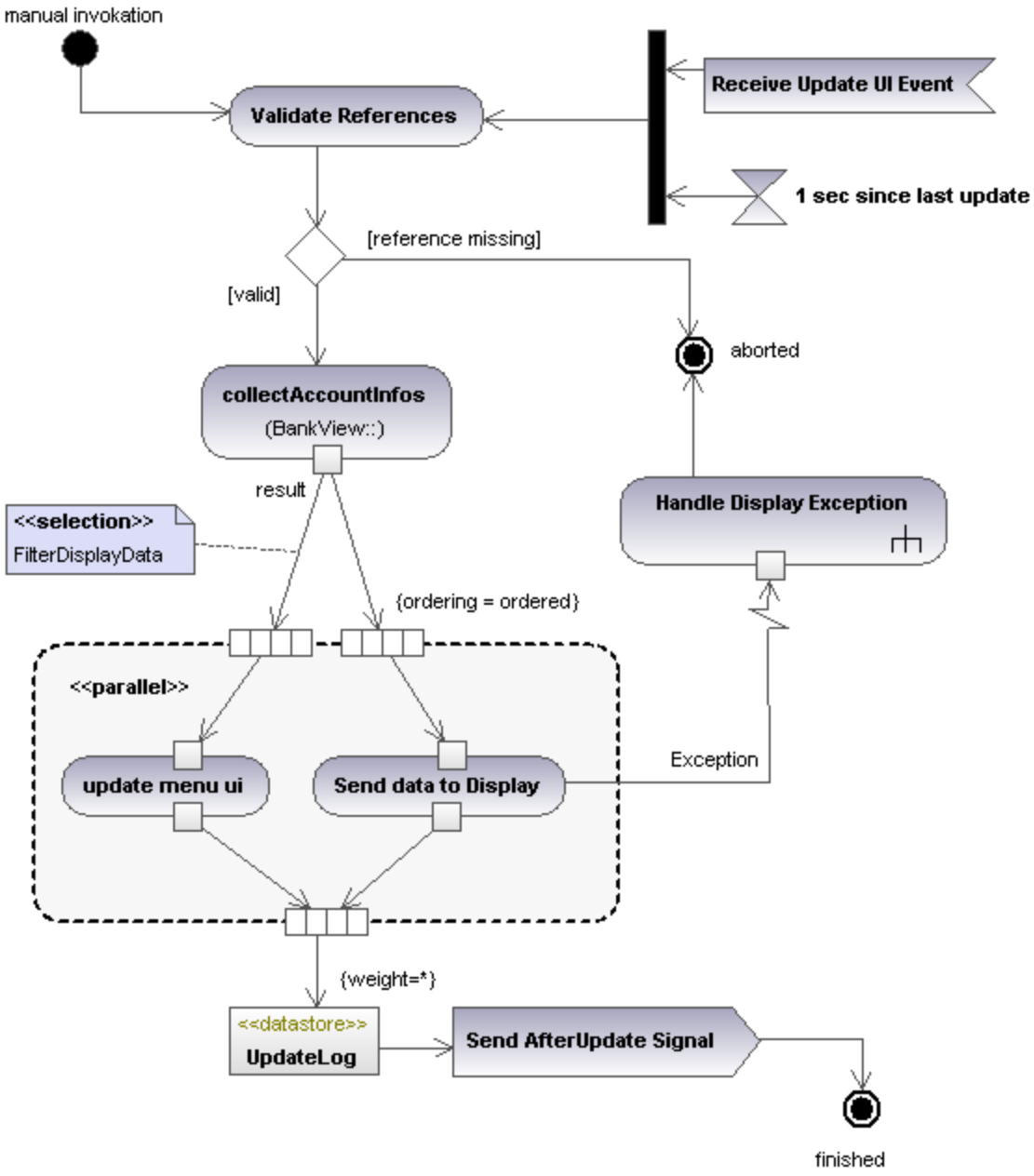
- Kommunikationsdiagramm
- Interaktionsübersichtsdiagramm
- Sequenzdiagramm
- Zeitverlaufsdiagramm 440

9.1.1 Aktivitätsdiagramm

Altova Website: UML-Aktivitätsdiagramme

Aktivitätsdiagramme eignen sich, um reale Arbeitsabläufe von Geschäftsprozessen zu modellieren und um anzuzeigen, welche Aktionen dabei erforderlich sind und welche Abhängigkeiten diese haben. Im Aktivitätsdiagramm wird die Reihenfolge von Aktivitäten beschrieben. Es wird sowohl bedingte als auch parallele Verarbeitung unterstützt. Das Aktivitätsdiagramm ist eine Variante des Zustandsdiagramms, wobei die Zustände hier Aktivitäten sind.

Das unten gezeigte Aktivitätsdiagramm steht im UModel-Ordner **...\UModelExamples** im Beispiel **Bank_MultiLanguage**.ump zur Verfügung.



9.1.1.1 Einfügen von Aktivitätsdiagrammelementen

So fügen Sie Elemente zum Diagramm hinzu:

1. Klicken Sie in der Symbolleiste "Aktivitätsdiagramm" auf die Symbolleisten-Schaltfläche des Elements.

© 2016-2022 Altova GmbH



2. Klicken Sie in das Aktivitätsdiagramm, um das Element einzufügen.

Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

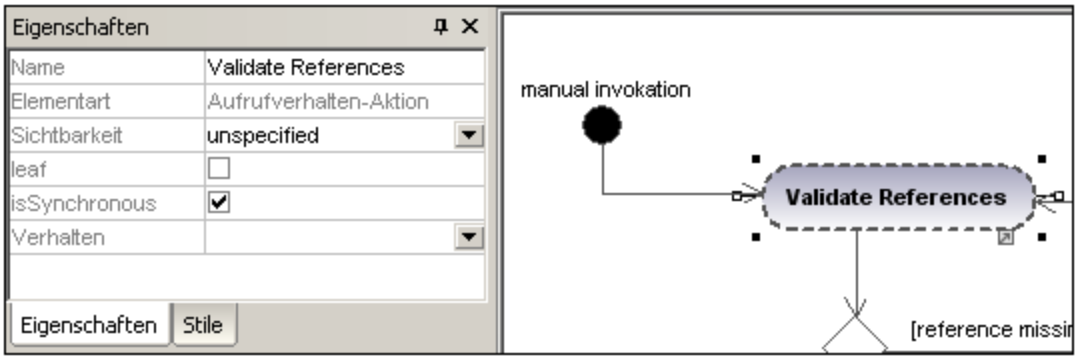
Ziehen bestehender Elemente in das Aktivitätsdiagramm

Die meisten Elemente, die in anderen Aktivitätsdiagrammen vorkommen, können in ein bestehendes Aktivitätsdiagramm eingefügt werden.

1. Suchen Sie das gewünschte Element im Fenster "Modell-Struktur" 86 (Sie können dazu das Suchfunktionstextfeld verwenden oder **Strg + F** drücken).
2. Ziehen Sie das/die Element(e) in das Aktivitätsdiagramm.

Einfügen einer Aktion (Aufrufverhalten)

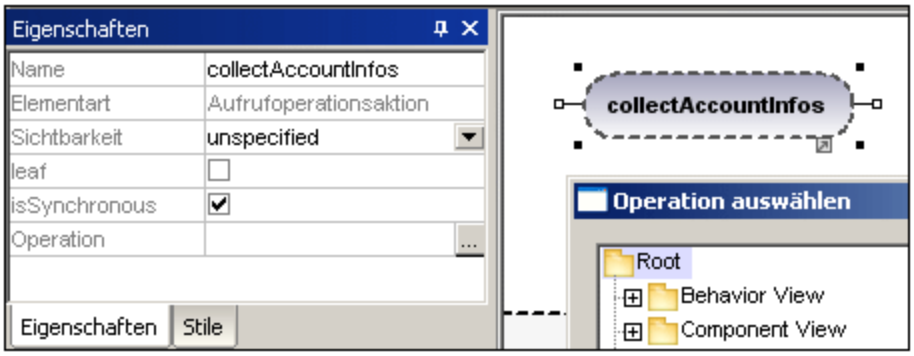
1. Klicken Sie auf die Symbolleisten-Schaltfläche **Aktion (Aufrufverhalten)**  und anschließend in das Aktivitätsdiagramm, um es einzufügen.
2. Geben Sie den Namen der Aktion ein, z.B. "Validate References" (=Referenzen validieren) und drücken Sie zur Bestätigung die **Eingabetaste**.



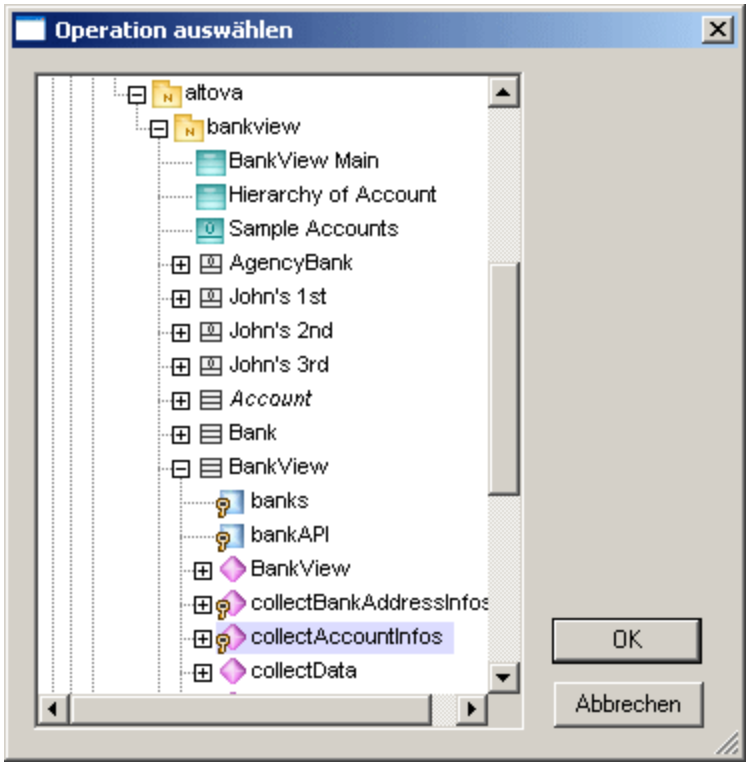
Anmerkung: Durch Drücken von **Strg+Eingabetaste** können Sie einen mehrzeiligen Namen erstellen.

Einfügen einer Aktion (Aufrufoperation) und Auswählen einer bestimmten Operation

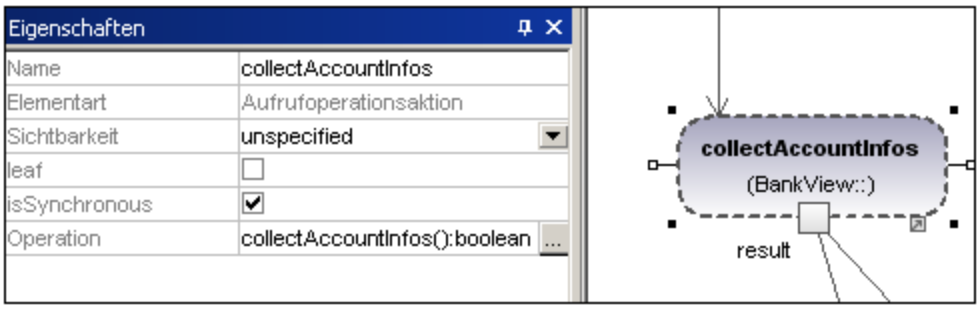
1. Klicken Sie in der Symbolleiste auf das Symbol **Aktion (Aufrufoperation)**  und anschließend in das Aktivitätsdiagramm um es einzufügen.
2. Geben Sie den Namen der Aktion ein, z.B. collectAccountInfo und drücken Sie zur Bestätigung die **Eingabetaste**.
3. Klicken Sie auf dem Register **Eigenschaften** auf die **Durchsuchen**-Schaltfläche rechts vom Feld "Operation". Daraufhin wird das Dialogfeld "Operation auswählen" geöffnet. Hier können Sie die gewünschte Operation auswählen.



4. Navigieren Sie zur gewünschten Operation und bestätigen Sie die Auswahl mit **OK**.



In diesem Beispiel befindet sich die Operation "collectAccountInfos" in der Klasse `BankView`.



© 2016-2022 Altova GmbH

9.1.1.2 Erstellen von Verzweigungen und Merges

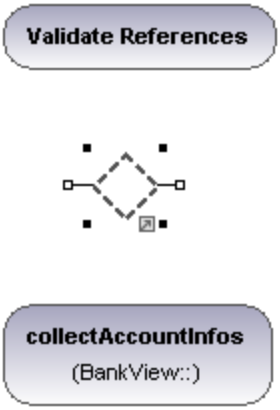
Erstellen einer Verzweigung (alternativer Fluss)
Eine Verzweigung hat einen einzigen eingehenden Fluss und mehrere mit "Guards" versehene ausgehende Flüsse. Es kann nur einer der ausgehenden Flüsse ausgewählt werden, daher sollten die Guards einander gegenseitig ausschließen.

In diesem Beispiel müssen die (BankView) Referenzen validiert werden.

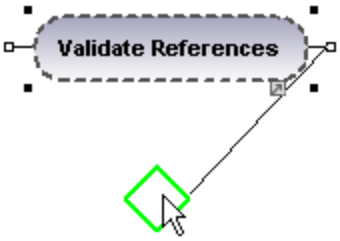
- branch1 (Verzweigung1) hat den Guard "reference missing" (=Referenz fehlt). Diese Transition endet im Abbruch der Aktivität.
- branch2 hat den Guard "valid" (= gültig). Diese Transition führt zur Aktivität collectAccountInfos (Kontoinformationen abrufen).

Erstellen einer Verzweigung (alternate flow)

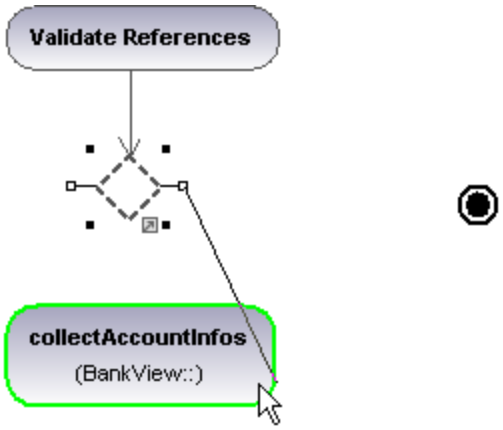
1. Klicken Sie in der Symbolleiste auf das Symbol **Verzweigungsknoten**  und fügen Sie das Element im Aktivitätsdiagramm ein.



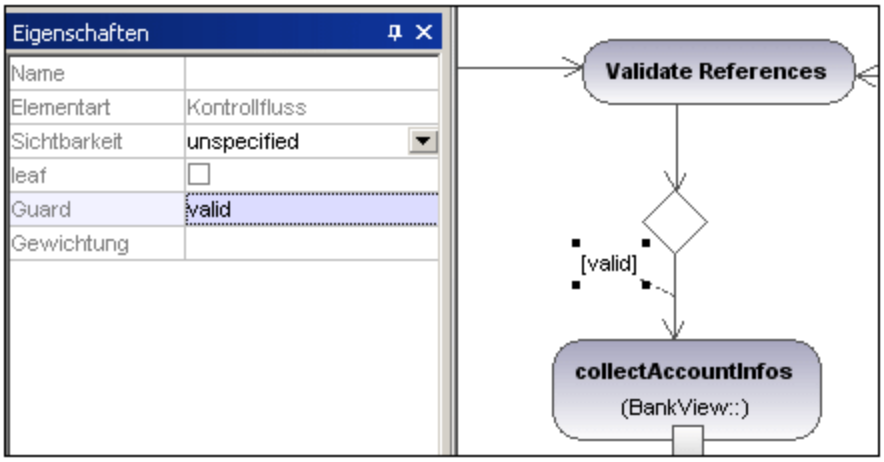
2. Klicken Sie auf das Symbol "Aktivitätsendknoten"  , welches für die Aktivität "Abbruch" steht und fügen Sie es in das Aktivitätsdiagramm ein.
3. Klicken Sie auf die Aktivität "Validate References", um sie auszuwählen, anschließend auf den rechten Ziehpunkt **Kontrollfluss** und ziehen Sie den Konnektor, der angezeigt wird, auf das Verzweigungsknoten-Element.



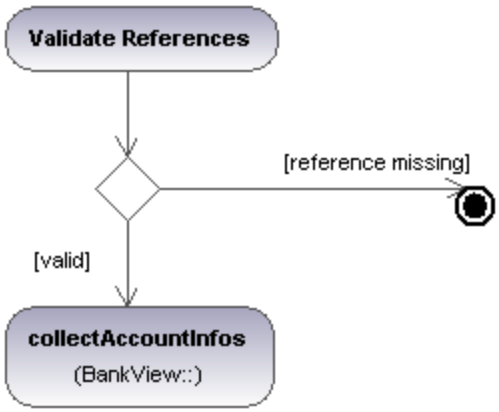
Sobald das Element markiert angezeigt wird, können Sie die Maustaste loslassen.
4. Klicken Sie auf das Verzweigungsknoten-Element, klicken Sie auf den rechten Konnektor **Kontrollfluss** und ziehen Sie ihn auf die Aktion "collectAccountInfos". Nähere Informationen dazu finden Sie unter "Einfügen einer Aktion (Aufrufoperation) 359".



5. Geben Sie auf dem Register "Eigenschaften" die Guard-Bedingung "valid" ein.



6. Klicken Sie auf das **Verzweigungsknotenelement** und ziehen Sie den rechten Ziehpunkt **Kontrollfluss** auf das Element "Aktivitätsendknoten".
Die Guard-Bedingung zu dieser Transition wird automatisch als "else" definiert. Doppelklicken Sie im Diagramm auf die Guard-Bedingung und ändern Sie diese z.B. in "reference missing".

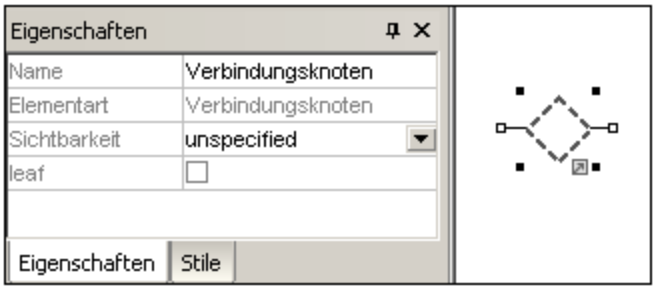


© 2016-2022 Altova GmbH

Bitte beachten Sie, dass UModel die Anzahl der Kontroll- / Objektflüsse in einem Diagramm nicht validiert oder überprüft.

Erstellen eines Merge

1. Klicken Sie in der Symbolleiste auf das Verbindungsknotensymbol und anschließend in das Aktivitätsdiagramm, um es einzufügen.

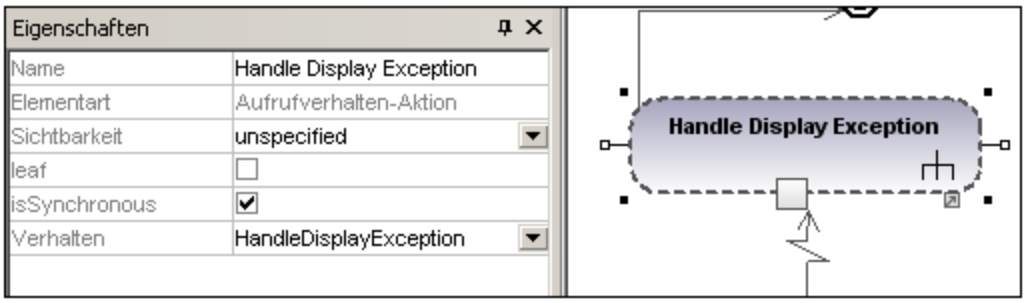


2. Klicken Sie auf die Kontrollfluss (Objektfluss)-Ziehpunkte der zu vereinigenden Aktionen und ziehen Sie die Pfeile auf das Verbindungsknotensymbol.

9.1.1.3 Aktivitätsdiagramm-Elemente

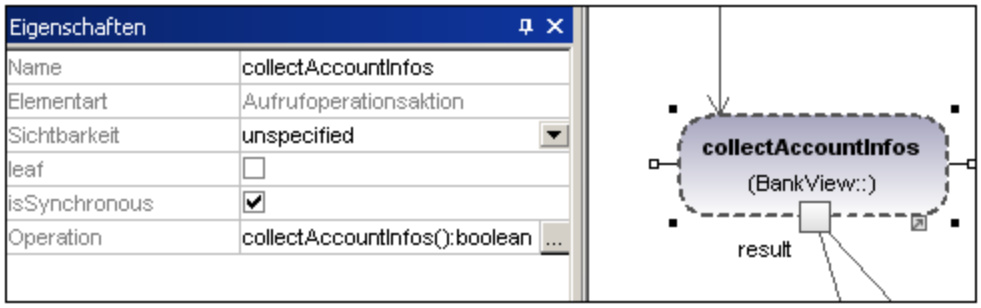
Aktion (Aufrufverhalten)

Fügt das Aktionselement **Aufrufverhalten-Aktion** ein, welches ein bestimmtes Verhalten direkt aufruft. Wenn Sie über die Auswahlliste **Verhalten** ein vorhandenes Verhalten auswählen, z.B. HandleDisplayException, wird innerhalb des Elements das Symbol eines Rechens angezeigt.



Aktion (Aufrufoperation)

Fügt eine **Aufrufoperationsaktion** ein, welche ein bestimmtes Verhalten als Methode direkt aufruft. Nähere Informationen finden Sie unter "Einfügen einer Aktion (Aufrufoperation) 359".



Aktion (OpaqueAction)

Eine Art von Aktion, mit der Implementierungsinformationen definiert werden. Kann als Platzhalter verwendet werden, bis fest steht, welche spezifische Aktion verwendet werden soll.

Aktion (ValueSpecificationAction)

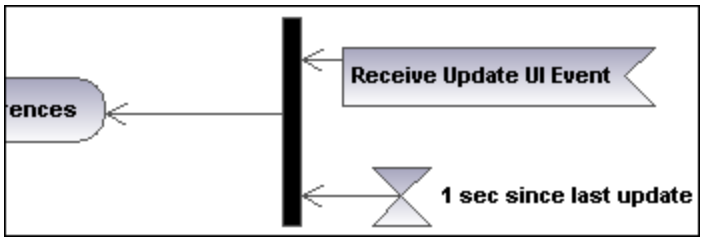
Eine Art von Aktion zum Auswerten (/Generieren)

Ereignisannahmeaktion

Fügt die Ereignisannahmeaktion ein, welche auf das Eintreten eines Ereignisses wartet, das bestimmte Bedingungen erfüllt.

Ereignisannahmeaktion (Zeitereignis)

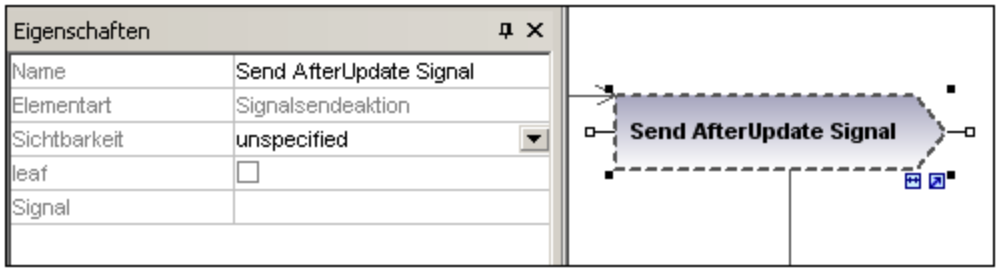
Fügt eine **Ereignisannahmeaktion** ein, die durch ein Zeitereignis ausgelöst wird, welches ein Zeitintervall - z.B. 1 sec since last update - mit einem Ausdruck definiert.



Signalsendeaktion

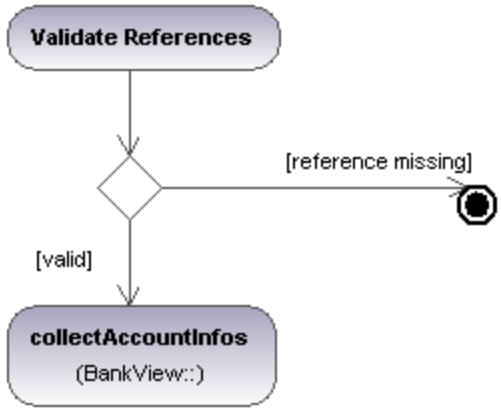
Fügt die **Signalsendeaktion** ein. Dabei wird ein Signal von den Inputs erstellt. Dieses Signal wird an das Zielobjekt übertragen, wo es die Ausführung einer Aktivität auslösen kann.

© 2016-2022 Altova GmbH



Verzweigungsknoten

Fügt einen Verzweigungsknoten ein, welcher eine einzige eingehende Transition hat und mehrere ausgehende mit Guards versehene Transitionen. Nähere Informationen dazu finden Sie unter "Erstellen einer Verzweigung[361]".



Verbindungsknoten

Fügt einen Verbindungsknoten ein, der mehrere durch einen Verzweigungsknoten definierte alternative Transitionen zusammenführt. Die gleichzeitig ablaufenden Prozesse werden dabei nicht synchronisiert, sondern es wird einer der Prozesse ausgewählt.

Startknoten

Der Anfang der Aktivität. Eine Aktivität kann mehrere Startknoten haben.

Aktivitätsendknoten

Das Ende der Aktivität. Eine Aktivität kann mehrere Endknoten haben. Alle Flüsse in der Aktivität werden beendet, wenn der "erste" Endknoten erreicht wird.

Endknoten für Kontrollflüsse

Fügt den Endknoten für Kontrollflüsse ein, d.h. ein Fluss wird dadurch beendet. Diese Beendung hat keinen Einfluss auf andere Flüsse in der Aktivität.

Parallelisierungsknoten

Fügt einen vertikalen Parallelisierungsknoten ein.
Dient zum Aufteilen von Flüssen in mehrere gleichzeitige Flüsse.

Parallelisierungsknoten (horizontal)

Fügt einen horizontalen Parallelisierungsknoten ein.
Dient zum Aufteilen von Flüssen in mehrere gleichzeitige Flüsse.

Synchronisationsknoten

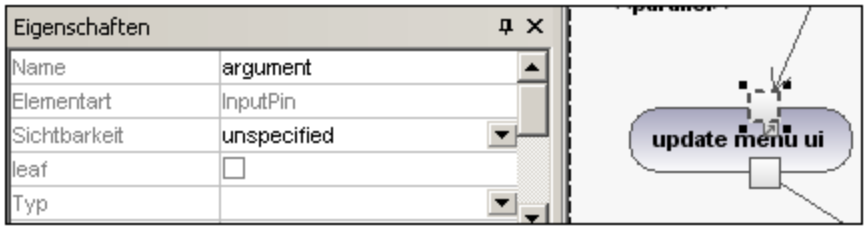
Fügt einen vertikalen Parallelisierungsknoten ein.
Ein Synchronisationsknoten synchronisiert mehrere durch einen Parallelisierungsknoten definierte Flüsse.

Synchronisationsknoten (horizontal)

Fügt einen horizontalen Parallelisierungsknoten ein.
Ein Synchronisationsknoten synchronisiert mehrere durch einen Parallelisierungsknoten definierte Flüsse.

InputPin

Fügt einen Input Pin in ein Aufrufverhalten oder eine Aufrufoperation ein. Input Pins liefern Eingabewerte, die von einer Aktion verwendet werden. Ein Input Pin erhält automatisch den Standardnamen "argument".

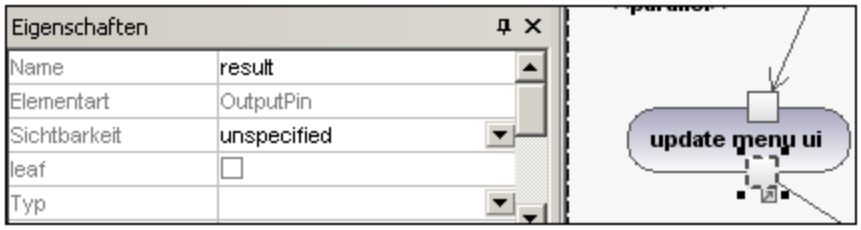


Das Input Pin-Symbol kann nur auf jene Aktivitätselemente platziert werden, bei denen der Mauszeiger sich in das Symbol einer Hand verwandelt . Wenn Sie das Symbol mit der Maus ziehen, wird es am Rand des Elements neu positioniert.

OutputPin

Fügt eine Output Pin-Aktion ein. Output Pins enthalten Werte, die von einer Aktion erzeugt werden. Dem Output Pin wird automatisch ein Name zugewiesen, der der UML-Eigenschaft dieser Aktion - z.B. "result" - entspricht.

© 2016-2022 Altova GmbH



Das Output Pin-Symbol kann nur auf jene Aktivitätselemente platziert werden, bei denen der Mauszeiger sich in eine Hand verwandelt . Wenn Sie das Symbol mit der Maus ziehen, wird es am Rand des Elements neu positioniert.

Ausnahmepin

Ein Output Pin kann durch Klicken auf den Pin und Auswahl der Option "isExceptionPin" im Fenster "Eigenschaften" in einen Ausnahmepin umgewandelt werden.

Wertpin

Fügt einen Wertpin ein. Dabei handelt es sich um einen Input Pin, der für eine Aktion einen Wert bereitstellt, der nicht aus einem eingehenden Objektfluss stammt. Er wird als ein Input Pin-Symbol dargestellt und hat dieselben Eigenschaften wie ein Input Pin.

Objektknoten

Fügt einen Objektknoten ein. Ein Objektknoten ist ein abstrakter Aktivitätsknoten, der den Objektfluss in einer Aktivität definiert. Objektknoten können nur Werte zur Laufzeit enthalten, die dem Typ des Objektknotens entsprechen.

Pufferknoten

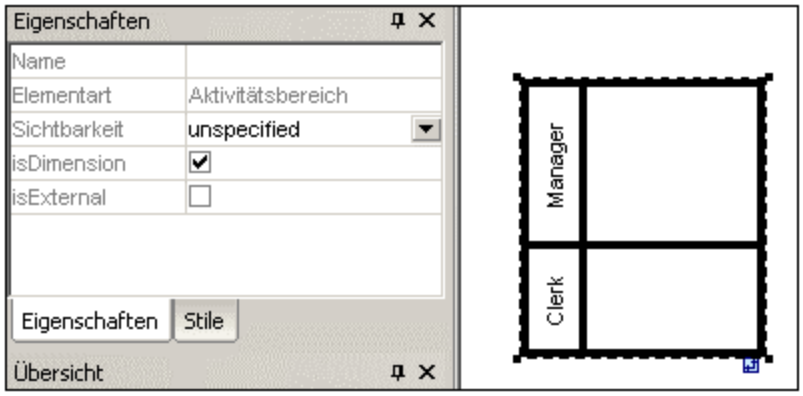
Fügt einen Pufferknoten ein, der als Puffer für mehrere eingehende und ausgehende Flüsse von anderen Objektknoten dient.

Datenspeicherknoten

Fügt einen Datenspeicherknoten ein. Dabei handelt es sich um einen speziellen Pufferknoten, der zum Speichern dauerhafter (d.h. nicht temporärer) Daten dient.

Aktivitätsbereich (horizontal)

Fügt einen horizontalen Aktivitätsbereich ein. Dabei handelt es sich um eine Art von Aktivitätsgruppe, die zum Kennzeichnen von Aktionen dient, die einige Eigenschaften gemeinsam haben. Dies entspricht oft Organisationseinheiten in einem Geschäftsmodell.



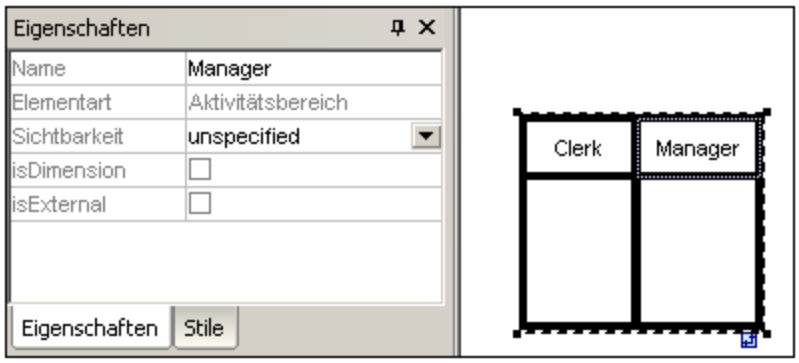
Wenn Sie auf eine Beschriftung doppelklicken, lässt sich diese direkt editieren; bei Drücken der Eingabetaste wird der Text korrekt ausgerichtet.

Bitte beachten Sie, dass Aktivitätsbereiche in UML 2.0 neu eingeführt wurden und die "swimlane" Funktion früherer UML-Versionen ersetzen.

- Elemente, die innerhalb einer ActivityPartition platziert werden, werden Teil davon, sobald die Umrandung markiert erscheint.
  - Objekte innerhalb einer ActivityPartition können mit Hilfe von Strg+Klick oder durch Aufziehen eines Rechtecks in der Umrandung einzeln ausgewählt werden.
  - Klicken Sie auf die Umrandung oder den titel der ActivityPartition und ziehen Sie sie/ihn, um diese an eine andere Stelle zu verschieben.

Aktivitätsbereich (vertikal)

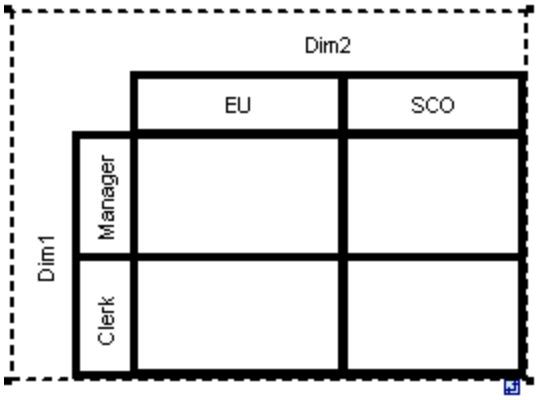
Fügt einen vertikalen Aktivitätsbereich ein. Dabei handelt es sich um eine Art von Aktivitätsgruppe, die zum Kennzeichnen von Aktionen dient, die einige Eigenschaften gemeinsam haben. Dies entspricht oft Organisationseinheiten in einem Geschäftsmodell.



Aktivitätsbereich (2 Dimensionen)

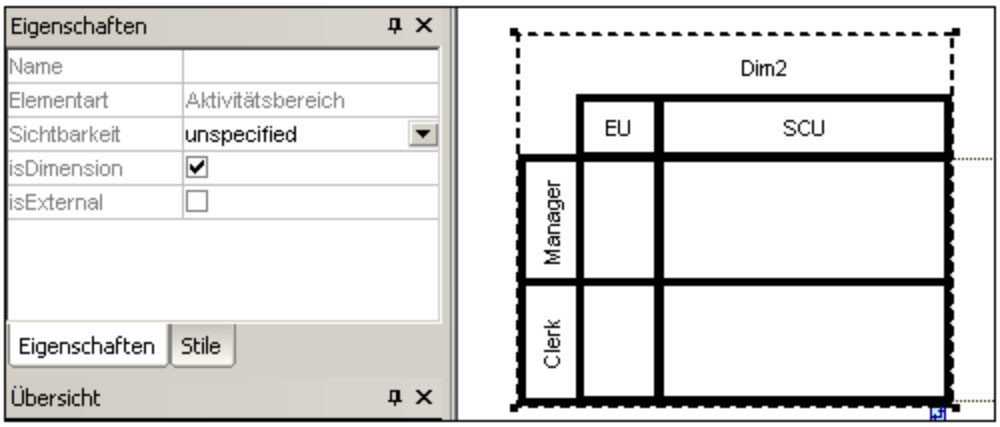
Fügt einen zweidimensionalen Aktivitätsbereich ein. Dabei handelt es sich um eine Art von Aktivitätsgruppe, die zum Kennzeichnen von Aktionen dient, die einige Eigenschaften gemeinsam haben. Beide Achsen haben editierbare Beschriftungen.

© 2016-2022 Altova GmbH



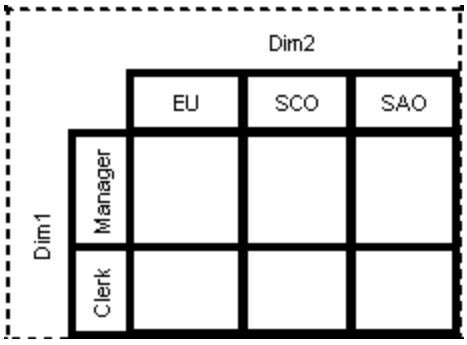
So entfernen Sie die Dim1, Dim2-Beschriftungen:

1. Klicken Sie auf die Dimensionsbezeichnung, die Sie entfernen möchten z.B. Dim1
2. Doppelklicken Sie auf dem Register "Eigenschaften" auf den Eintrag Dim1, löschen Sie ihn und bestätigen Sie den Vorgang mit der Eingabetaste.



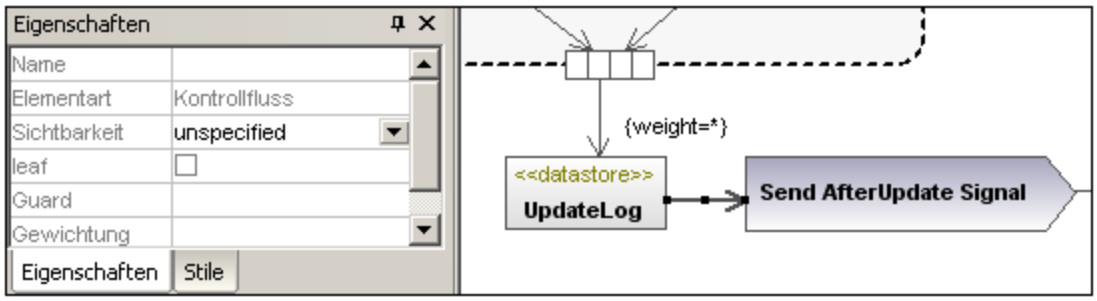
Beachten Sie, dass Aktivitätsbereiche geschachtelt sein können:

1. Rechtsklicken Sie auf die Beschriftung, wo Sie einen neuen Bereich einfügen möchten.
2. Wählen Sie **Neu | Aktivitätsbereich**.



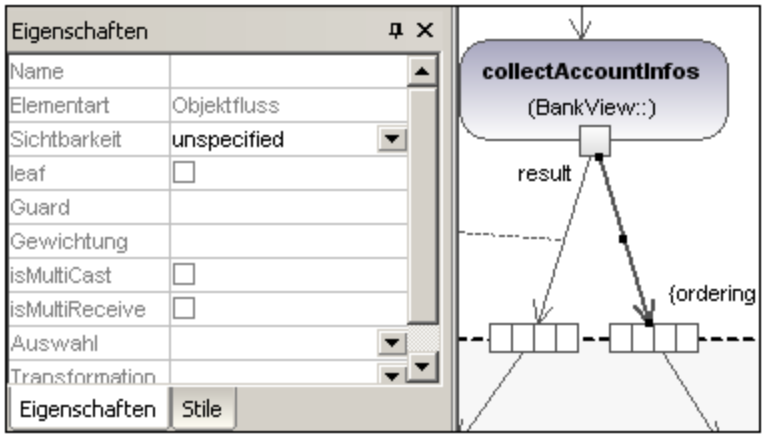
Kontrollfluss

Ein Kontrollfluss ist eine Kante, d.h. ein Pfeil, der zwei Aktivitäten/Verhalten verbindet und eine Aktivität startet, nachdem die vorherige zu Ende geführt wurde.



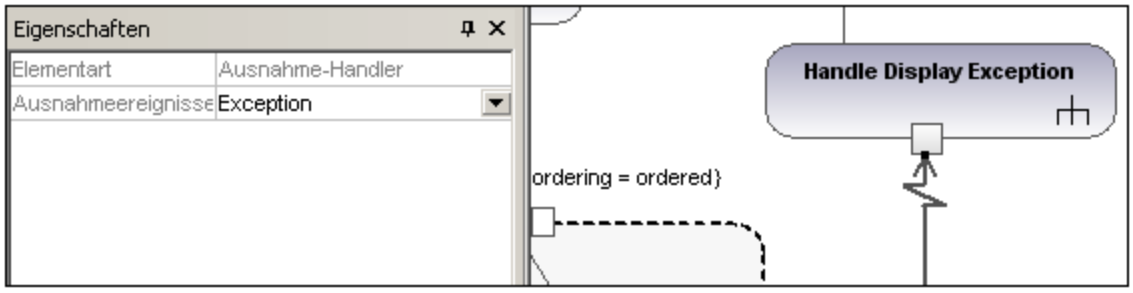
Objektfluss

Ein Objektfluss ist eine Kante, d.h. ein Pfeil, der zwei Aktionen/Objektknoten verbindet und eine Aktivität startet, sobald die vorherige zu Ende geführt worden ist. Entlang eines Objektflusses können Objekte oder Daten übergeben werden.



Ausnahme-Handler

Ein Ausnahme-Handler ist ein Element, das festlegt, welche Aktion ausgeführt werden soll, wenn während der Ausführung des geschützten Knotens ein bestimmtes Ausnahmeereignis auftritt.

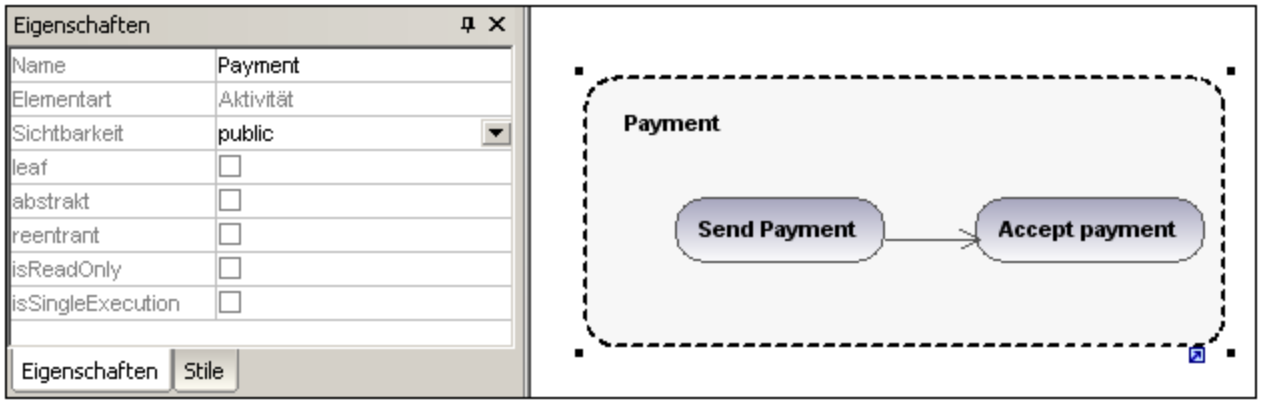


© 2016-2022 Altova GmbH

Ein Ausnahme-Handler kann nur auf einen Inputpin einer Aktion gezogen werden.

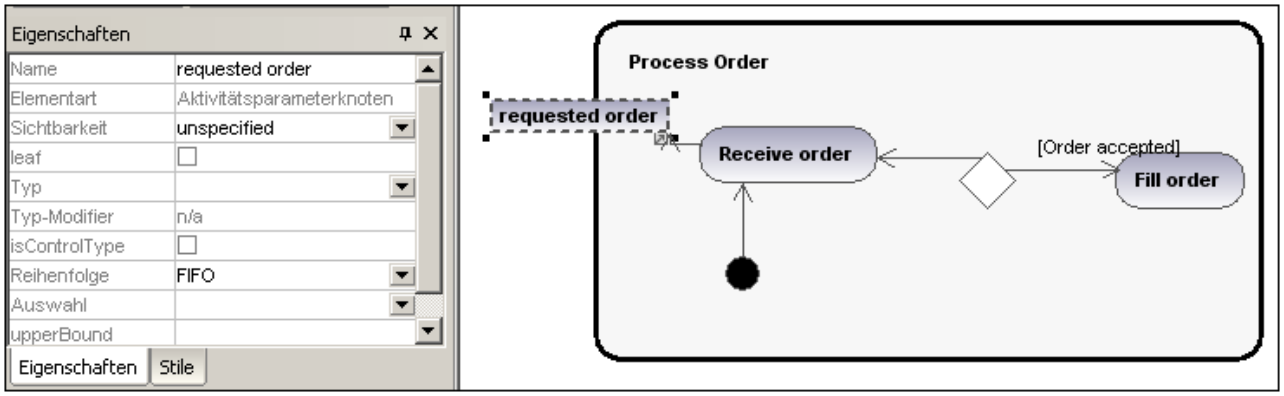
Aktivität

Fügt eine Aktivität in das Aktivitätsdiagramm ein.



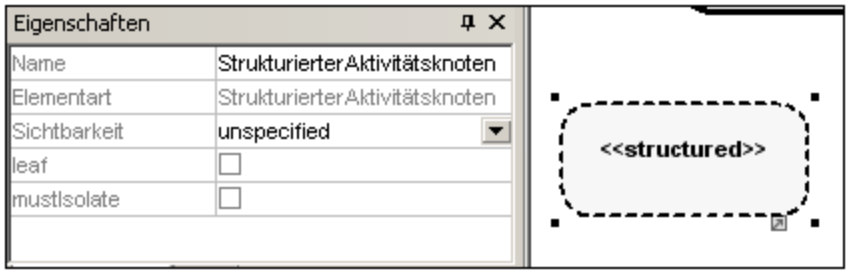
Aktivitätsparameterknoten

Fügt einen Aktivitätsparameterknoten in eine Aktivität ein. Wenn Sie an eine beliebige Stelle in der Aktivität klicken, wird der Parameterknoten auf den Rand der Aktivität platziert.



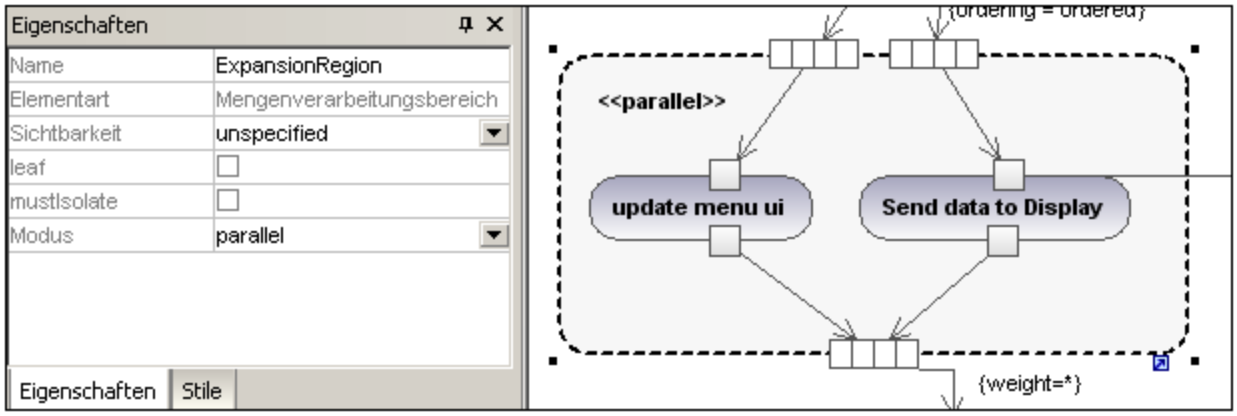
StrukturierterAktivitätsknoten

Fügt einen strukturierten Aktivitätsknoten ein. Dabei handelt es sich um einen strukturierten Teil der Aktivität, der mit keinem anderen strukturierten Knoten geteilt wird.



Mengenverarbeitungsbereich

Ein Mengenverarbeitungsbereich ist ein Bereich einer Aktivität mit expliziten Inputs und Outputs (über Erweiterungsknoten). Jeder Input ist eine Sammlung von Werten.

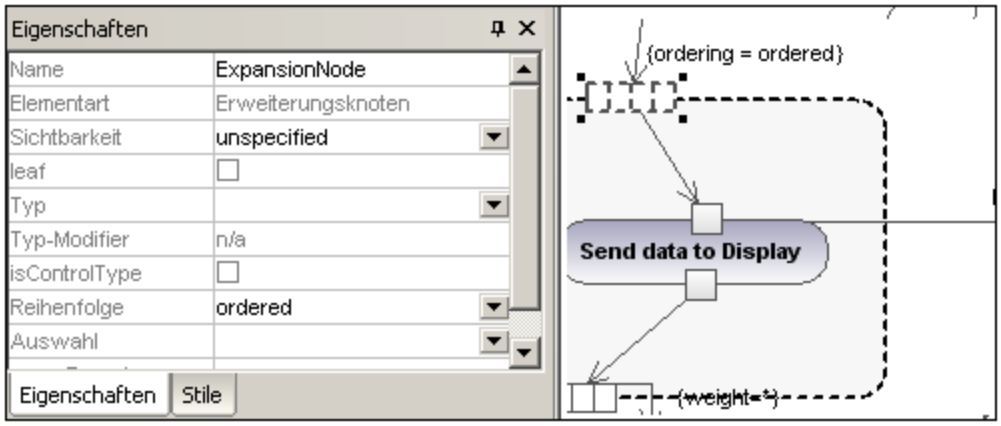


Der Mengenverarbeitungsbereichsknoten wird als Schlüsselwort angezeigt und kann durch Klicken auf die Auswahlliste "Modus" auf dem Register "Eigenschaften" geändert werden. Zur Auswahl stehen die Einstellungen: parallel, iterative oder stream.

Erweiterungsknoten

Fügt einen Erweiterungsknoten in einen Mengenverarbeitungsbereich ein. Erweiterungsknoten sind Eingabe- und Ausgabeknoten für den Mengenverarbeitungsbereich, wobei jeder Input/Output eine Sammlung von Werten ist. Die Pfeile, die in den Mengenverarbeitungsbereich hinein oder daraus heraus weisen , legen die jeweilige Art des Erweiterungsknotens fest.

© 2016-2022 Altova GmbH



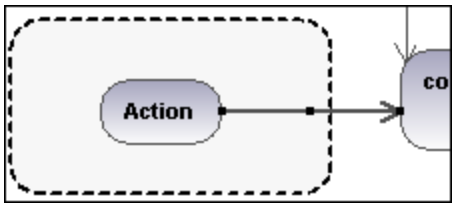
Unterbrechungsbereich

Ein Unterbrechungsbereich enthält Aktivitätsknoten. Wenn ein Kontrollfluss einen Unterbrechungsbereich verlässt, werden alle Flüsse und Verhalten im Bereich beendet.

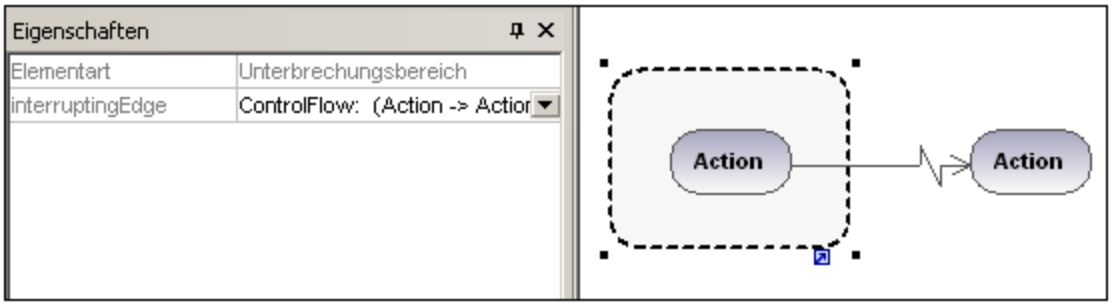
So fügen Sie eine InterruptingEdge hinzu:

Stellen Sie sicher dass:

- der Unterbrechungsbereich ein Aktionselement sowie einen ausgehenden Kontrollfluss zu einer anderen Aktion enthält:



1. Rechtsklicken Sie auf den Kontrollflusspfeil und wählen Sie **Neu | InterruptingEdge**.



Anmerkung:

Sie können eine InterrupingEdge auch durch Klicken auf den Unterbrechungsbereich, Rechtsklick in das Fenster "Eigenschaften" und Auswahl des Popup-Menübefehls "InterruptingEdge hinzufügen" hinzufügen.

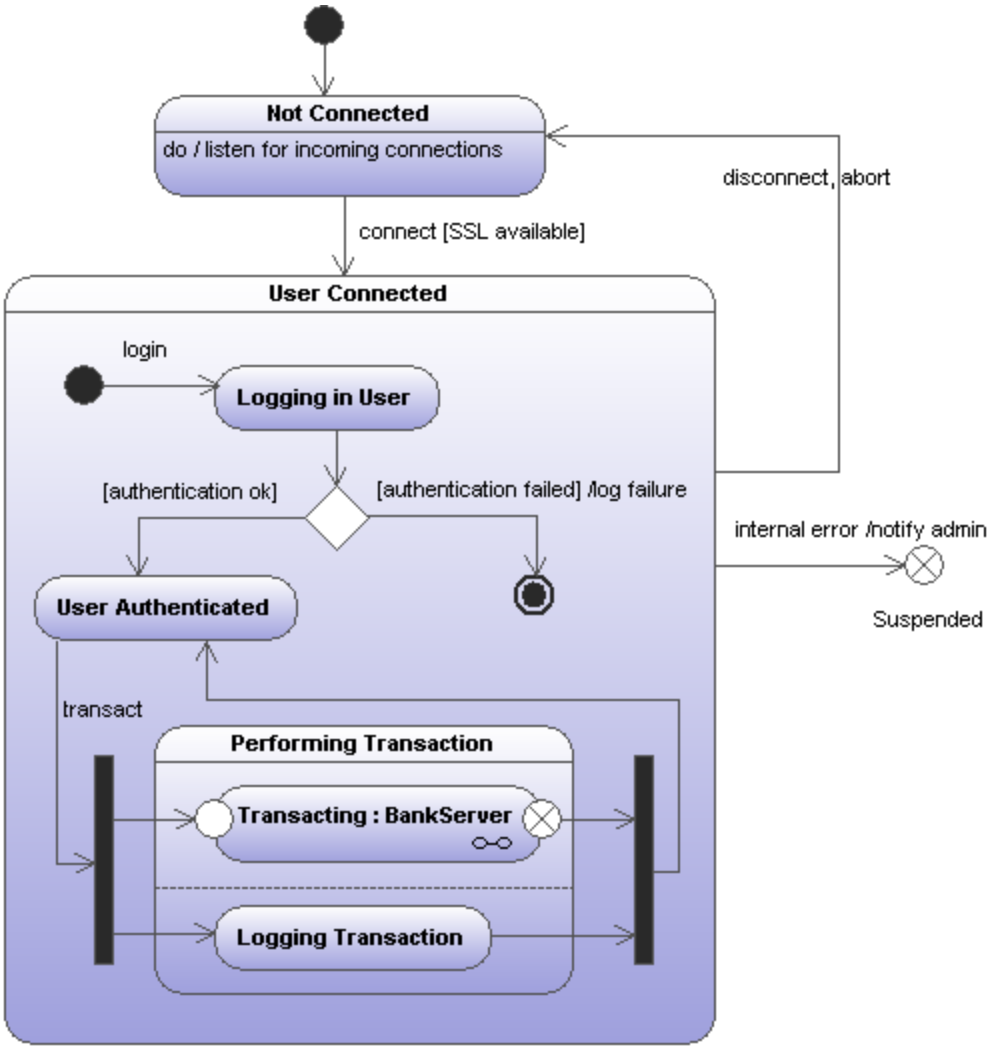
9.1.2 Zustandsdiagramm

Im Zustandsdiagramm wird das Verhalten eines Systems modelliert, indem die Zustände, in denen sich ein Objekt befinden kann und die Übergänge zwischen diesen Zuständen, beschrieben werden. Im Allgemeinen dienen diese Diagramme dazu, das Verhalten eines Objekts in verschiedenen Anwendungsfällen (Use Cases) zu beschreiben.

Dies kann in Form von zwei Arten von Prozessen erfolgen:

1. **Aktionen,** die mit **Transitionen** verknüpft sind, sind kurzfristige Prozesse, die nicht unterbrochen werden können (z.B. **internal error /notify admin** im Diagramm unten)
2. **Zustandsaktivitäten** (Verhalten), welche mit **Zuständen** verknüpft sind, sind länger andauernde Prozesse, die durch andere Ereignisse unterbrochen werden können (z.B. **listen for incoming connections** im Diagramm unten)

Ein Zustandsautomat kann in UModel beliebig viele Zustandsdiagramme haben.



© 2016-2022 Altova GmbH

Zustandsbeispieldiagramm

Das oben gezeigte Zustandsdiagramme steht im folgenden UModel-Beispielprojekt zur Verfügung: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_MultiLanguage.ump**.

9.1.2.1 Einfügen von Zustandsdiagrammelementen

Einfügen über die Symbole der Symbolleiste:

1. Klicken Sie in der Zustandsdiagramm-Symbolleiste auf das entsprechende Zustandsdiagramm-Symbol.



2. Klicken Sie in das Zustandsdiagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen von bestehenden Elementen in das Zustandsdiagramm

Die meisten Elemente, die in anderen Zustandsdiagrammen vorkommen, können in ein bestehendes Zustandsdiagramm eingefügt werden.

1. Suchen Sie das gewünschte Element auf dem Register Modell-Struktur (Sie können dazu das Suchfunktionstextfeld verwenden oder Strg + F drücken, um nach einem beliebigen Element zu suchen).
2. Ziehen Sie das/die Element(e) in das Zustandsdiagramm.

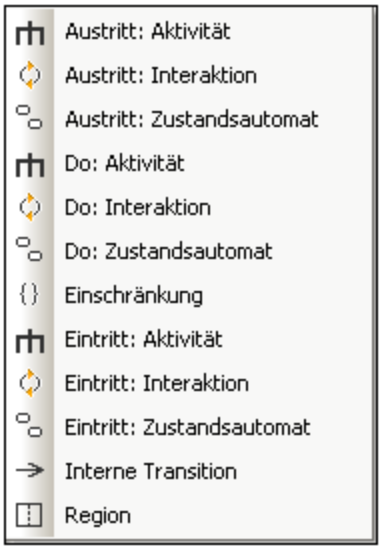
9.1.2.2 Erstellen von Zuständen, Aktivitäten und Transitionen

So fügen Sie einen einfachen Zustand hinzu:

1. Klicken Sie in der Symbolleiste auf das Symbol **Zustand** () und anschließend in das Diagramm.
2. Geben Sie den Namen des Zustands ein und drücken Sie zur Bestätigung die **Eingabetaste**.

So fügen Sie eine Aktivität zu einem Zustand hinzu:

- Rechtsklicken Sie auf das Element "Zustand", wählen Sie "Neu" und anschließend einen der Einträge aus dem Kontextmenü.



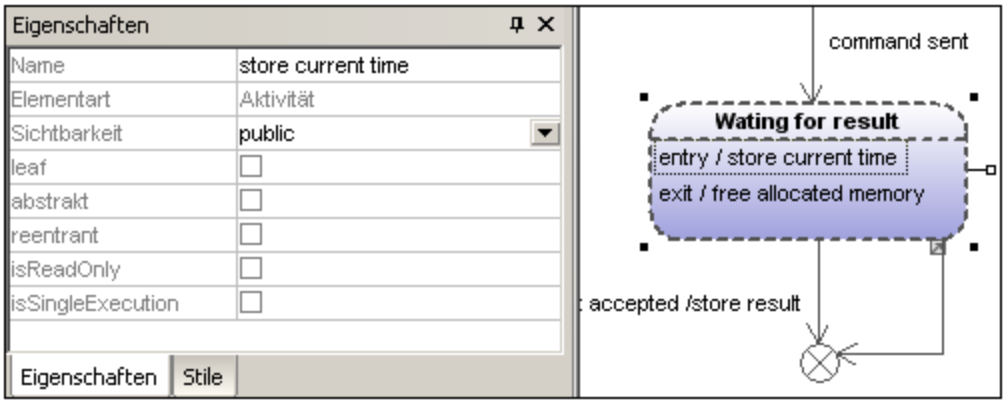
Die Aktivitäten **Eintritt**, **Austritt** und **Do** sind mit einem der folgenden möglichen Verhalten verknüpft: "Aktivität", "Interaktion" und "Zustandsautomat". Daher stehen im Kontextmenü die folgenden Optionen zur Verfügung:

- Austritt: Aktivität
- Austritt: Interaktion
- Austritt: Zustandsautomat
- Do: Aktivität
- Do: Interaktion
- Do: Zustandsautomat
- Eintritt: Aktivität
- Eintritt: Interaktion
- Eintritt: Zustandsautomat

Diese Optionen stammen aus der UML-Spezifikation. Bei jeder dieser internen Aktionen handelt es sich um ein Verhalten und in der UML-Spezifikation sind drei Klassen von der Klasse "Behavior" (Verhalten) abgeleitet: Activity (Aktivität), StateMachine (Zustandsautomat) und Interaction (Interaktion). Es macht im generierten Code keinen Unterschied, welches spezifische Verhalten (Aktivität, Zustandsautomat oder Interaktion) ausgewählt wurde.

Sie haben die Wahl zwischen Aktionen aus der Kategorie: Eintritts, Austritt und Do. Aktivitäten werden im Zustandselement in ihren eigenen Bereich - wenn auch nicht in eine separate Region - platziert. Die Art der ausgewählten Aktivität wird als Präfix für die Aktivität verwendet z.B: **entry / store current time** (=Eintritt / aktuelle Zeit speichern).

© 2016-2022 Altova GmbH

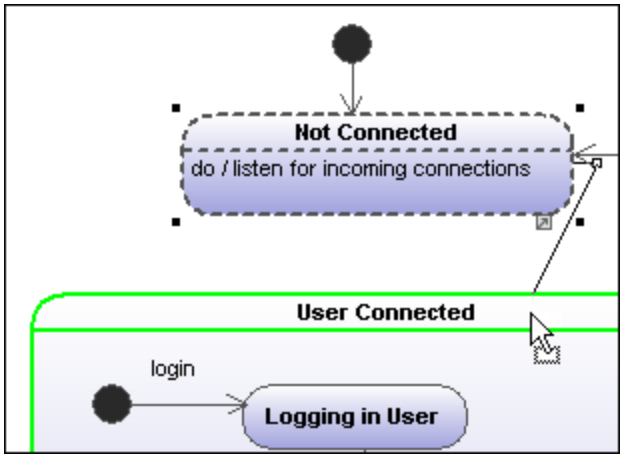


So löschen Sie eine Aktivität:

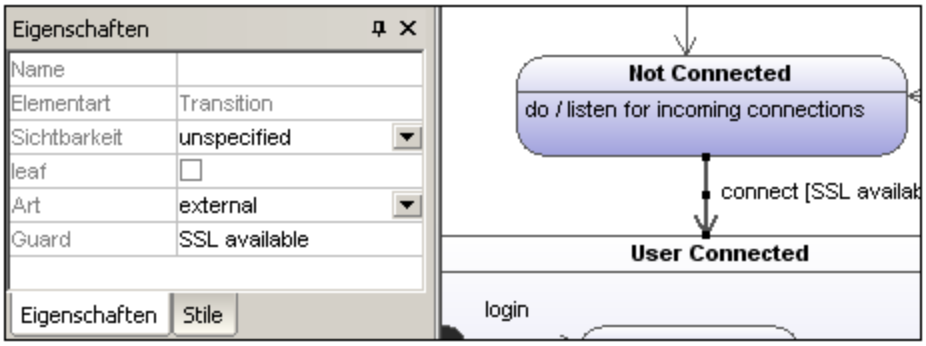
- Klicken Sie auf die entsprechende Aktivität im Zustandselement und drücken Sie die Entf-Taste.

So erstellen Sie eine Transition zwischen zwei Zuständen:

1. Klicken Sie auf den Transition-Ziehpunkt des Ausgangszustands (auf der rechten Seite des Elements).
2. Ziehen Sie den Transition-Pfeil mit der Maus auf den Ziel-Zustand.



Die Eigenschaften der Transition werden nun auf dem Register "Eigenschaften" angezeigt. Wenn Sie auf die Auswahlliste "Art" klicken, können Sie den Typ der Transition definieren: extern, intern oder lokal.



Transitionen können einen Event Trigger, eine Guard-Bedingung und eine Aktion in der Form **eventTrigger [Guard-Bedingung] /Aktivität** haben.

So erstellen Sie automatisch Operationen von Transitionen aus

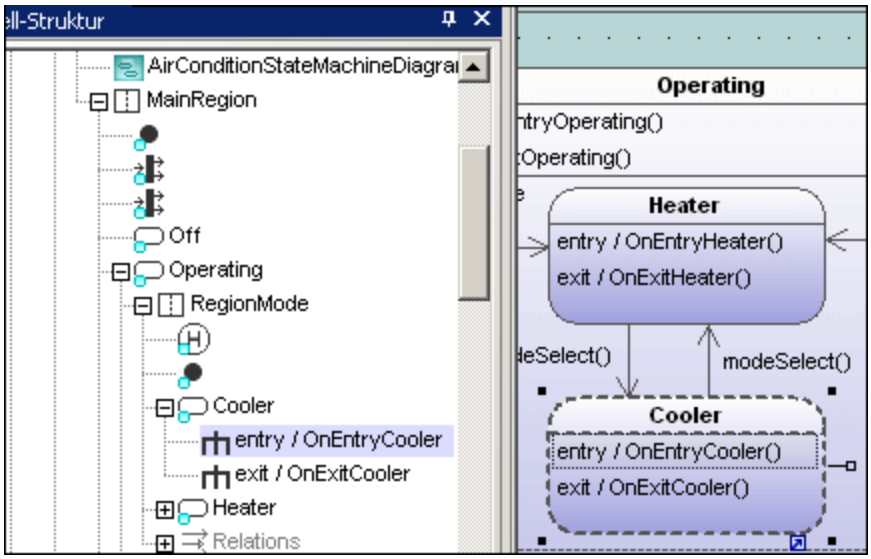
Wenn Sie die Schaltfläche "Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten"  aktivieren, wird die entsprechende Operation in der referenzierten Klasse automatisch erstellt, wenn Sie eine Transition erstellen und einen Namen,z. B. myOperation() eingeben.

Anmerkung: Operationen können nur dann automatisch erstellt werden, wenn sich der Zustandsautomat innerhalb einer Klasse oder Schnittstelle befindet.

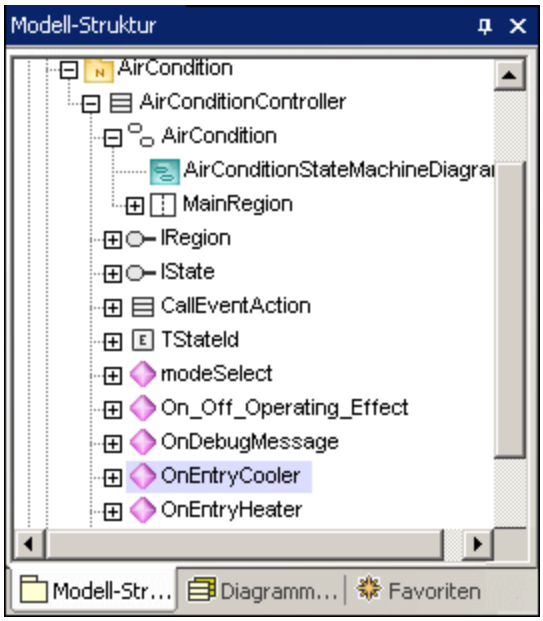
So erstellen Sie automatisch Operationen von Aktivitäten aus:

1. Klicken Sie mit der rechten Maustaste auf den Zustand und wählen Sie die entsprechende Aktion/Aktivität aus, z.B. Neu | Eintritt:Aktivität.
2. Geben Sie den Namen der Aktivität ein und stellen Sie sicher, dass Sie am Schluss das "Klammer auf" und "Klammer zu"-Zeichen "**()**", also z.B. **Eintritt / OnEntryCooler()** eingeben.

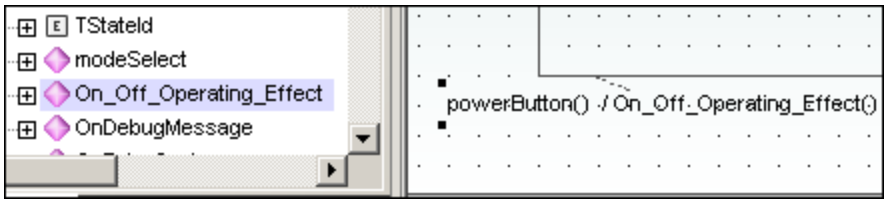
© 2016-2022 Altova GmbH



Das neue Element wird auch in der Modell-Struktur angezeigt.
Wenn Sie einen Bildlauf nach unten durch die Modell-Struktur durchführen, sehen Sie, dass die Operation OnEntryCooler zur übergeordneten Klasse AirConditionController hinzugefügt wurde.



Anmerkung:
Operationen werden automatisch für: Do:Aktivität, Eintritt:Aktivität, Austritt:Aktivität sowie als Guard-Bedingungsaktivitäten und Auswirkungen (in Transitionen) hinzugefügt.



So erstellen Sie einen Transition Trigger:

1. Rechtsklicken Sie auf eine zuvor erstelle Transition (Pfeil).
2. Wählen Sie **Neu | Trigger**.

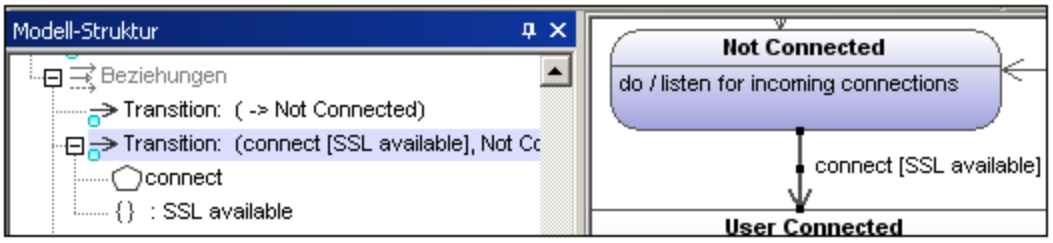


Über dem Transition-Pfeil wird in der Transition-Bezeichnung der Buchstabe "a" angezeigt, falls es sich um den ersten Trigger im Zustandsdiagramm handelt. Den Triggern werden Standardwerte in der Form: Buchstabe des Alphabets, Ausgangszustand -> Zielzustand zugewiesen.

3. Doppelklicken Sie auf den neuen Buchstaben und geben Sie die Eigenschaften der Transition in der Form **eventTrigger [Guard-Bedingung] /Aktivität** ein.

> Syntax der Eigenschaft der Transition:
> Der Text vor der eckigen Klammer ist der Trigger, innerhalb der eckigen Klammer befindet sich die Guard-Bedingung und hinter dem Schrägstrich folgt die Aktivität. Bei Bearbeitung des Strings werden automatisch die entsprechenden Elemente in der Modell-Struktur erstellt bzw. gelöscht.

Anmerkung:
Um die Eigenschaften der einzelnen Transition zu sehen, rechtsklicken Sie auf die Transition (also auf den Pfeil) und wählen Sie die Option "In Modell-Struktur auswählen". Das Ereignis, die Aktivität und die Einschränkungselemente werden alle unterhalb der ausgewählten Transition angezeigt.

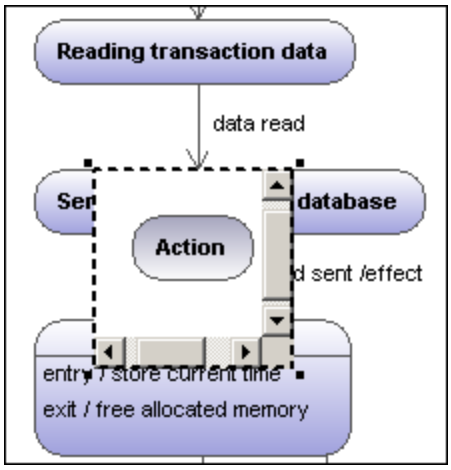


Hinzufügen eines Aktivitätsdiagramms zu einer Transition

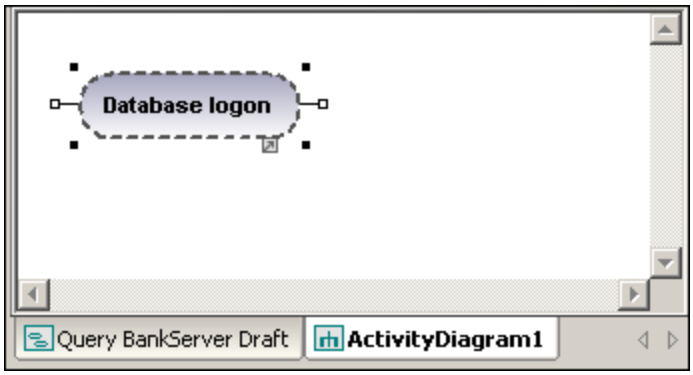
UModel bietet die einzigartige Möglichkeit, zur näheren Beschreibung der Transition ein Aktivitätsdiagramm zu einer Transition hinzuzufügen.

1. Rechtsklicken Sie auf einen Transitionspfeil im Diagramm und wählen Sie **Neu | Aktivitätsdiagramm**. Daraufhin wird an der Position des Transitionspfeils ein Aktivitätsdiagrammfenster in das Diagramm eingefügt.
2. Klicken Sie auf das eingefügte Fenster, um es aktiv zu machen. Über die Bildlaufleisten können Sie durch das Fenster scrollen.

© 2016-2022 Altova GmbH

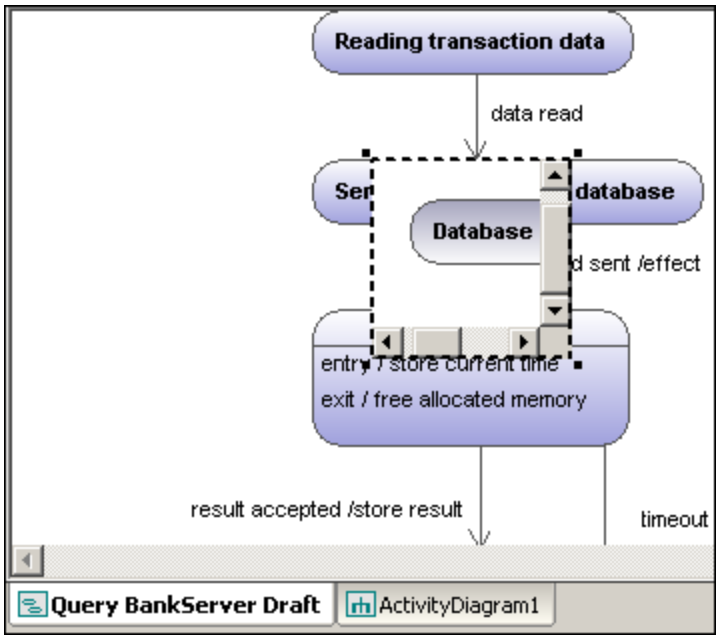


3. Doppelklicken Sie auf das Aktionsfenster um in das Aktivitätsdiagramm zu wechseln und die Transition näher zu definieren, z.B. um den Namen der Aktion in "Database logon" zu ändern.

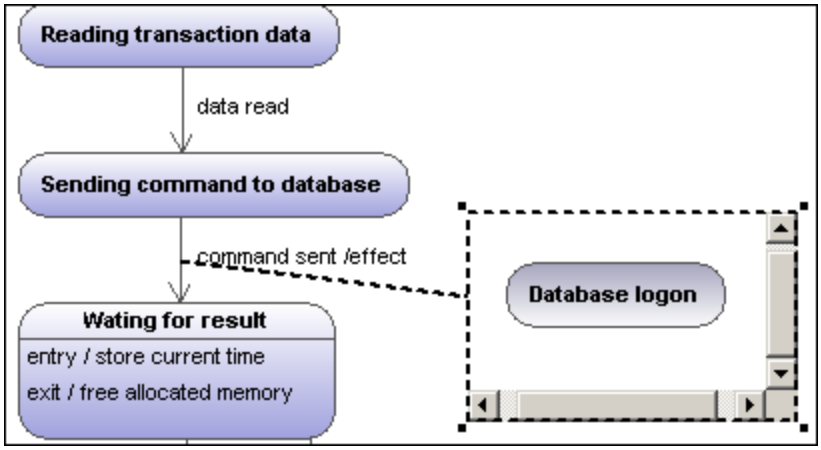


Beachten Sie, dass ein neues Aktivitätsdiagrammregister zum Projekt hinzugefügt wurde. Sie können jedes beliebige Aktivitätsdiagramm-Modellelement zum Diagramm hinzufügen. Nähere Informationen dazu finden Sie unter "Aktivitätsdiagramm 357".

4. Klicken Sie auf das Zustandsdiagrammregister um zurückzuwechseln und die aktualisierte Transition zu sehen.

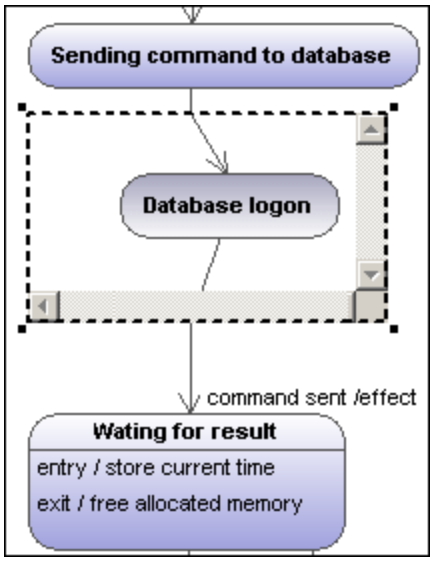


5. Ziehen Sie gegebenenfalls das Aktivitätsfenster an die gewünschte Stelle im Diagramm und klicken Sie auf den Ziehpunkt zur Größenanpassung.



Wenn Sie das Aktivitätsfenster zwischen die beiden Zustände ziehen, wird die Transition in die Aktivität und aus der Aktivität heraus angezeigt.

© 2016-2022 Altova GmbH



9.1.2.3 Zusammengesetzte Zustände

Zusammengesetzter Zustand

Diese Art von Zustand enthält einen zweiten Bereich bestehend aus einer einzigen Region. Innerhalb dieser Region können beliebig viele Zustände platziert werden.

So fügen Sie eine Region zu einem zusammengesetzten Zustand hinzu:

1. Rechtsklicken Sie auf den zusammengesetzten Zustand und wählen Sie im Kontextmenü den Befehl **Neu | Region**.
   Daraufhin wird eine neue Region zum Zustand hinzugefügt. Regionen werden durch strichlierte Linien voneinander getrennt.

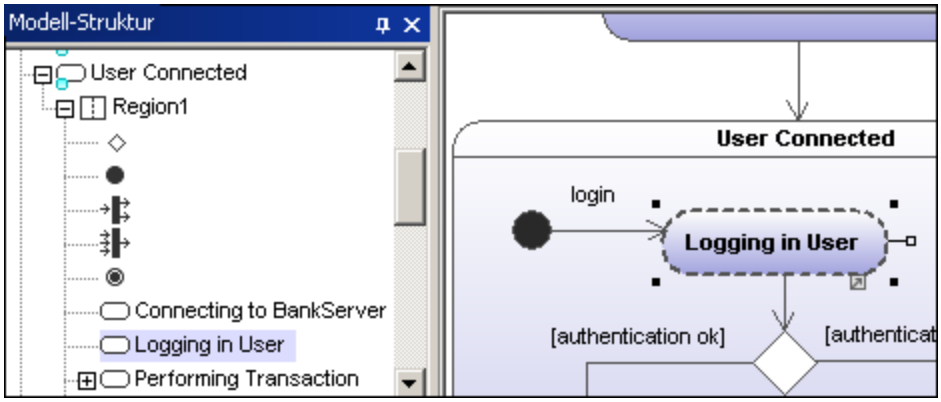
So löschen Sie eine Region:

1. Klicken Sie auf die gewünschte Region im zusammengesetzten Zustand und drücken Sie die Entf-Taste.
   Wenn Sie eine Region eines orthogonalen Zustands löschen, so wird dieser wieder zu einem zusammengesetzten Zustand; sobald die letzte Region eines zusammengesetzten Zustands gelöscht wurde, wird dieser wieder zu einem einfachen Zustand.

So platzieren Sie einen Zustand in einen zusammengesetzten Zustand:

1. Klicken Sie auf das gewünschte Zustandselement (z.B. Logging in User) und ziehen Sie es in den Regionsbereich des zusammengesetzten Zustands.

Der Regionsbereich erscheint markiert, sobald Sie die Maustaste loslassen können. Das eingefügte Element ist nun Teil der Region und wird im Fenster "Modell-Struktur" als Child-Element der Region angezeigt.

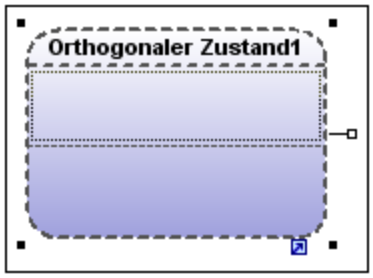


Wenn Sie den zusammengesetzten Zustand verschieben, werden alle darin enthaltenen Zustände mitverschoben.

Orthogonaler Zustand

Diese Art von Zustand enthält einen zweiten Bereich bestehend aus zwei oder mehr Regionen, wobei die einzelnen Regionen auf Gleichzeitigkeit hinweisen.

Wenn Sie mit der rechten Maustaste auf einen Zustand klicken und **Neu | Region** auswählen, können Sie neue Regionen hinzufügen.



So blenden Sie Regionsnamen ein/aus:

Klicken Sie auf das Register "Stile", scrollen Sie zum Eintrag "Regionsnamen in Zustandselementen anzeigen" und wählen Sie true/false aus.

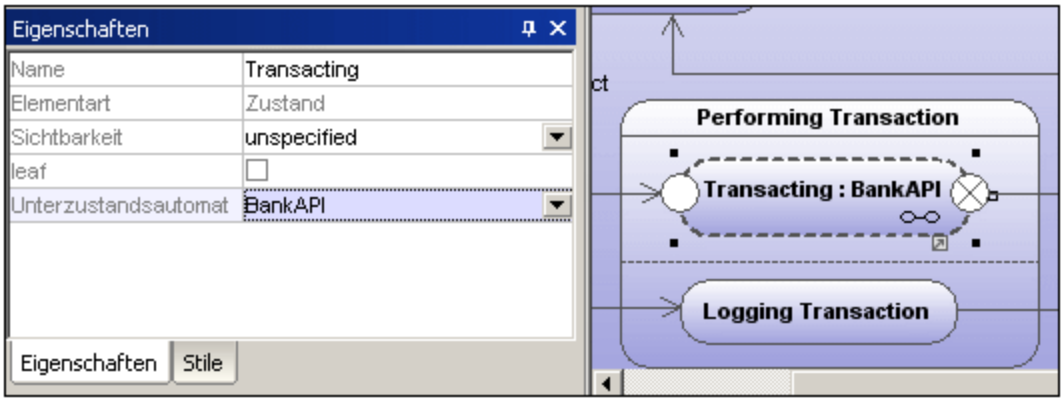
Unterautomatenzustand

Dieser Zustand dient zum Ausblenden der Einzelheiten eines Zustandsautomaten. Dieser Zustand hat keine Regionen, sondern ist mit einem separaten Zustandsautomaten verknüpft.

© 2016-2022 Altova GmbH

So definieren Sie einen Unterautomatenzustand:

1. Wählen Sie zuerst einen Zustand aus und klicken Sie auf dem Register "Eigenschaften" auf die Auswahlliste **Unterautomatenzustand.**
   Es erscheint eine Liste mit den derzeit definierten Zustandsautomaten.
2. Wählen Sie den Zustandsautomaten aus, den dieser Unterautomat referenzieren soll.

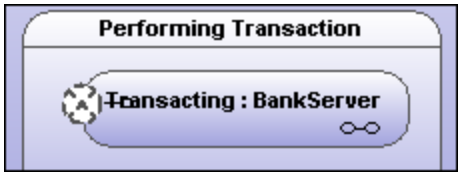


Beachten Sie bitte: Im Unterautomat wird automatisch ein Hyperlink angezeigt. Wenn Sie darauf klicken, wird der referenzierte Zustandsautomat, in diesem Fall BankServer, geöffnet.

So fügen Sie Eintritts- / Austrittspunkte zu einem Unterautomatenzustand hinzu:

- Der Zustand, mit dem der Punkt verbunden ist, muss selbst einen Unterautomatenzustand (sichtbar auf dem Register "Eigenschaften") referenzieren.
- Dieser Unterautomat muss einen oder mehrere Eintritts- und Austrittspunkte enthalten

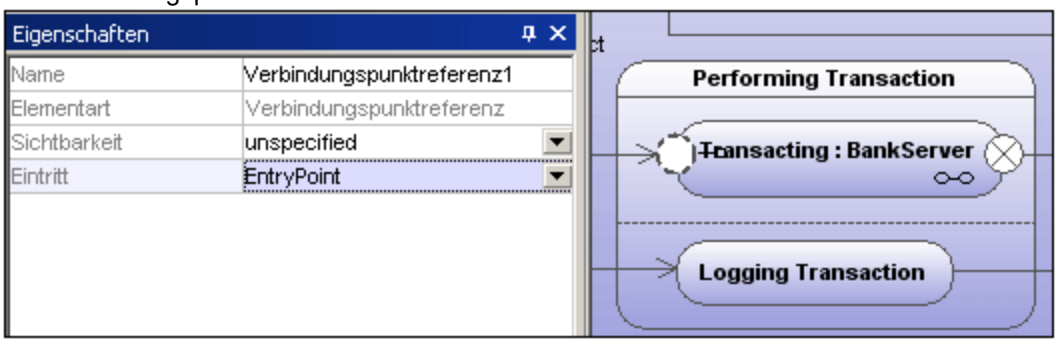
1. Klicken Sie in der Titelleiste auf das Symbol **Verbindungspunktreferenz**  und klicken Sie anschließend auf den Unterautomatenzustand, zu dem Sie den Eintritts- / Austrittspunkt hinzufügen möchten.



2. Rechtsklicken Sie auf das Register "Eigenschaften" und wählen Sie den Befehl "Eintritt hinzufügen". Bitte beachten Sie, dass es an einer anderen Stelle im Diagramm einen weiteren Eintritts- oder Austrittspunkt geben muss, damit dieses Popup-Menü aktiviert wird.



Daraufhin wird eine Eintrittspunktzeile zum Register "Eigenschaften" hinzugefügt und das Aussehen des Verbindungspunktreferenz-Elements ändert sich.



3. Fügen Sie auf dieselbe Weise einen Austrittspunkt hinzu. Wählen Sie dazu im Kontextmenü den Befehl "Austritt hinzufügen".

9.1.2.4 Generieren von Code anhand von Zustandsdiagrammen

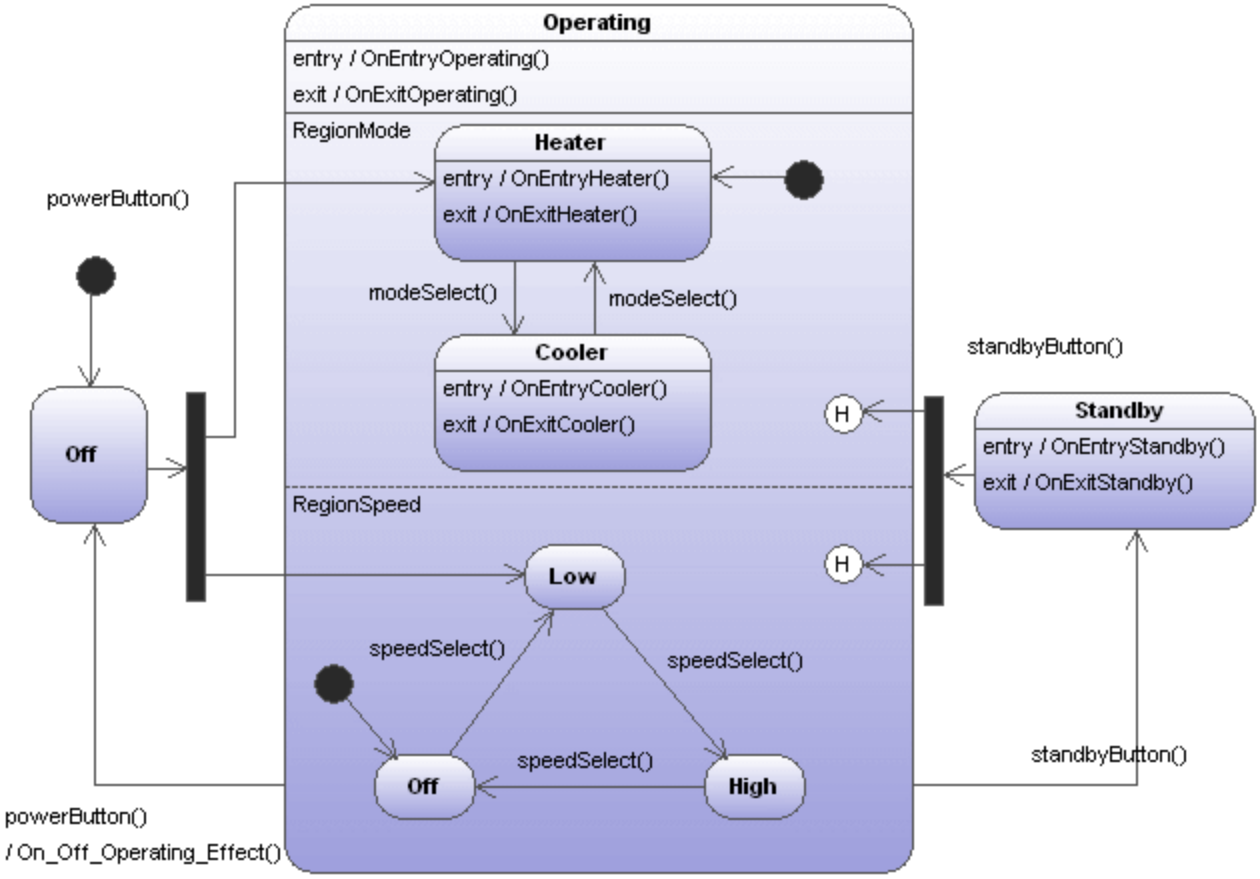
Sie können in UModel ausführbaren Code (C++, Java, VB.NET) anhand von Zustandsdiagrammen generieren. Es werden beinahe alle Zustandsdiagrammelemente und -funktionen unterstützt:

- Zustand
- Zusammengesetzter Zustand, mit jeder hierarchischen Ebene
- Orthogonaler Zustand, mit beliebig vielen Regionen
- Region
- Anfangszustand
- Endzustand
- Transition
- Guard
- Trigger
- Aufrufereignis
- Gabelung
- Vereinigung
- Entscheidung
- Kreuzung
- DeepHistory
- ShallowHistory

© 2016-2022 Altova GmbH

- Eintritts-/Austritts-/Do-Aktion
- Effekt

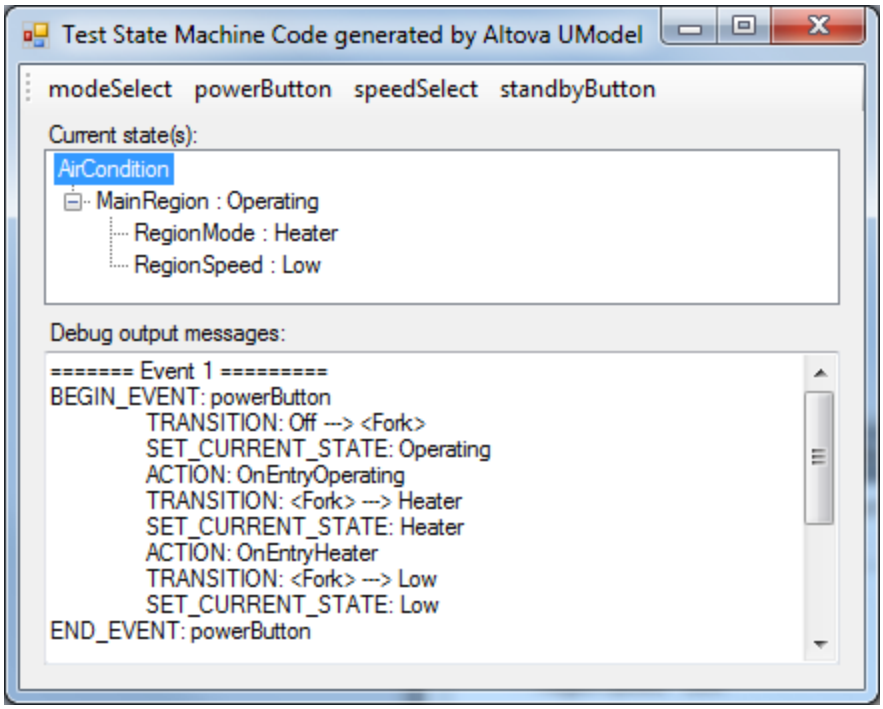
Die Generierung von Zustandsautomatencode ist in das normale Round Trip Engineering integriert, d.h. der Zustandsautomatencode kann bei jedem Forward Engineering automatisch aktualisiert werden.



In der Abbildung oben sehen Sie das Zustandsdiagramm AirCondition aus dem Verzeichnis **..\StateMachineCodeGeneration** unter**...\UModelExamples**. Für jede der Sprachen, in der in UModel Code generiert werden kann, gibt es ein eigenes Verzeichnis.

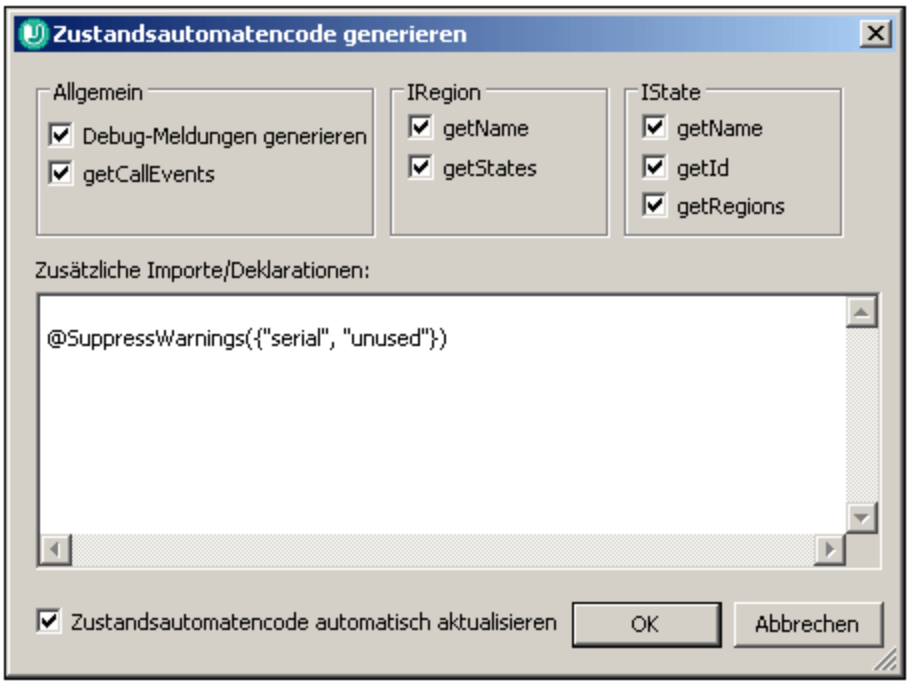
Jedes Verzeichnis enthält jeweils einen Ordner AirCondition und einen Ordner Complex, die jeweils das entsprechende UModel-Projekt, die Programmiersprachen-Projektdateien sowie die generierten Quelldateien enthalten. Die Projektdatei Complex.ump enthält beinahe alle der von UModel beim Generieren von Zustandsdiagrammcode unterstützten Modellierungselemente und Funktionalitäten.

Außerdem enthält jedes Verzeichnis eine Testapplikation, z.B. TestSTMAirCondition.sln für C#, sodass Sie mit den generierten Quellcodedateien sofort arbeiten können.



So generieren Sie anhand eines Zustandsdiagramms Code:

- **Klicken Sie mit der rechten Maustaste** in das Zustandsdiagramm und wählen Sie den Befehl "Zustandsautomatencode generieren" oder
- Wählen Sie die Menüoption **Projekt | Zustandsautomatencode generieren.**



© 2016-2022 Altova GmbH

In der Abbildung oben sehen Sie die Standardeinstellungen. Klicken Sie zur Codegenerierung auf OK.

Der Zustandsautomatencode wird automatisch aktualisiert, wenn Sie das Forward Engineering starten. Sie können diese Einstellung allerdings ändern, indem Sie auf den Hintergrund des Zustandsdiagramms klicken und das Kontrollkästchen "Code automatisch aktualisieren" deaktivieren.

Am generierten Code sollten keine manuellen Änderungen vorgenommen werden, da diese Änderungen beim Reverse Engineering nicht im Zustandsdiagramm übernommen werden.



Wenn Sie neben dem Feld "Code automatisch aktualisieren" auf das Symbol ... klicken, wird das Dialogfeld "Zustandsautomatencode generieren" geöffnet, wo Sie die Codegenerierungseinstellungen ändern können.

So führen Sie in einem Zustandsdiagramm eine Syntaxüberprüfung durch:

- Klicken Sie mit der rechten Maustaste auf das Diagramm und wählen Sie den Befehl **Zustandsautomatensyntax überprüfen**.

9.1.2.5 Arbeiten mit Zustandsdiagrammcode

Die übergeordnete Klasse des Zustandsdiagramms (d.h. die "Controller-Klasse" oder "Kontextklasse") bildet die einzige Schnittstelle zwischen dem Benutzer des Zustandsdiagramms und der Zustandsdiagrammimplementierung.

Die Controller-Klasse bietet Methoden, mit Hilfe derer die Zustandsdiagramme von "außen" her geändert werden können (z.B. nachdem externe Ereignisse eingetreten sind).

Bei der Implementierung des Zustandsdiagramms werden dagegen Controller-Klassenmethoden ("Callbacks") aufgerufen, um den Benutzer des Zustandsdiagramms über Zustandsänderungen (OnEntry, OnExit, ...), Transitionseffekte und die Möglichkeit Methoden für Bedingungen (Guards) außer Kraft zu setzen und zu implementieren, zu informieren.

UModel kann einfache Operationen (ohne Parameter) für Entry/Exit/Do-Verhalten, Transitionseffekte, ...automatisch erstellen, wenn die entsprechende Option aktiviert ist. (siehe auch Erstellen von Zuständen, Aktivitäten und Transitionen) 375. Diese Methoden können in UModel beliebig (durch Hinzufügen von Parametern, Definieren der Parameter als abstrakt, usw.) geändert werden.

Ein Zustandsdiagramm (d.h. seine Controller-Klasse) kann mehrmals instantiiert werden. Alle Instanzen sind unabhängig voneinander.

- Die Ausführung des UML-Zustandsdiagramms ist für das Modell: Bis zur Fertigstellungen ausführen" konzipiert.
- UML-Zustandsautomaten gehen davon aus, dass jedes Ereignis abgeschlossen ist, bevor das nächste verarbeitet wird.
- Dies bedeutet auch, dass keine Entry/Exit/Do-Aktion oder kein Transitionseffekt eine neue Transition/eine Zustandsänderung direkt auslösen darf.

Initialisierung

- Jede Region eines Zustandsdiagramms muss einen Anfangszustand haben.
- Der von UModel generierte Code initialisiert alle Regionen des Zustandsdiagramms automatisch (oder bei Aufruf der Initialize()-Methode der Controller-Klasse).
- Wenn OnEntry-Ereignisse bei der Initialisierung nicht erwünscht sind, können Sie die Initialize()-Methode manuell aufrufen und OnEntry-Ereignisse beim Start ignorieren.

Abrufen des/der aktuellen Zustands/Zustände

UModel unterstützt sowohl zusammengesetzte Zustände als auch orthogonale Zustände, d.h. es gibt nicht nur einen aktuellen Zustand, sondern jede Region (auf jeder hierarchischen Ebene) kann einen aktuellen Zustand haben.

Im Beispiel "AirCondition" wird gezeigt, wie die Regionen des aktuellen Zustands / der aktuellen Zustände durchlaufen werden.

```
TreeNode rootNode = m_CurrentStateTree.Nodes.Add(m_STM.getRootState().getName());
UpdateCurrentStateTree(m_STM.getRootState(), rootNode);

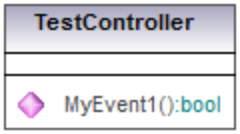
private void UpdateCurrentStateTree(AirCondition.AirConditionController.IState state,
TreeNode node)
{
    foreach (AirCondition.AirConditionController.IRegion r in state.getRegions())
    {
        TreeNode childNode = node.Nodes.Add(r.getName() + " : " + r.getCurrentState().getName());
        UpdateCurrentStateTree(r.getCurrentState(), childNode);
    }
}
```

Beispiel 1 - eine einfache Transition



Die entsprechende Operation wird in UModel automatisch generiert.

© 2016-2022 Altova GmbH

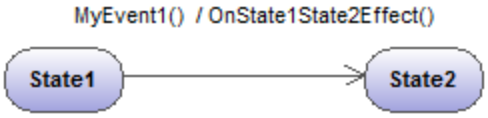


Generierte Methode im Code:

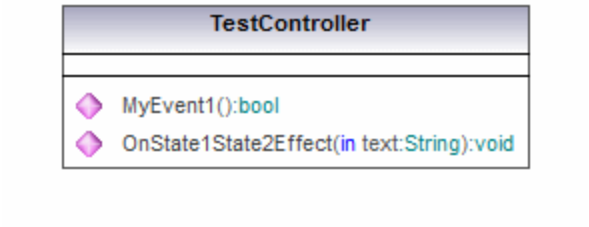
```
private class CTestStateMachine : IState
{
        …
        public bool MyEvent1()
        {
                …
        }
}
```

- Der Benutzer des Zustandsdiagramms sollte die generierte Methode "MyEvent1" aufrufen, wenn das entsprechende Ereignis (außerhalb des Zustandsautomaten) eintritt.
- Der Rückgabeparameter dieser Ereignismethoden liefert Informationen, ob das Ereignis eine Zustandsänderung verursacht hat (d.h. ob es eine Auswirkung auf das Zustandsdiagramm hat) oder nicht. Wenn sich der Automat z.B. im "State1" befindet und "MyEvent1()" eintritt, so ändert sich der aktuelle Status in "State2" und "MyEvent1()" gibt "true" zurück. Wenn "State2" aktiv ist und "MyEvent1()" eintritt, ändert sich nichts am Zustandsdiagramm und MyEvent1() gibt "false" zurück.

Beispiel 2 - eine einfache Transition mit einem Effekt



Die entsprechende Operation wird in UModel automatisch generiert.



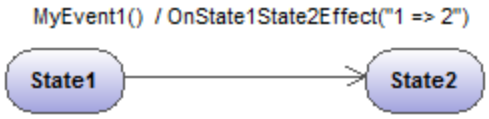
Generierte Methode im Code:

```
private class CTestStateMachine : IState
{
        …
        // Override to handle entry/exit/do actions, transition effects,...:
        public virtual void OnState1State2Effect() {}
}
```

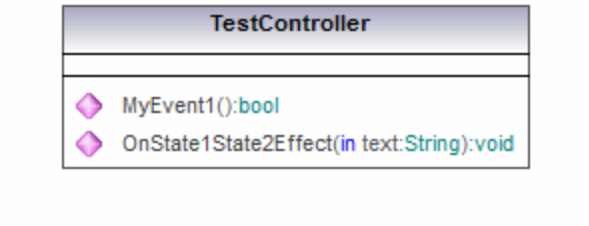
- "OnState1State2Effect()" wird immer dann von der Zustandsdiagrammimplementierung aufgerufen, wenn die Transition zwischen "State1" und "State2" ausgelöst wird.
- Als Reaktion auf diesen Effekt sollte "OnState1State2Effect()" in einer abgeleiteten Klasse von "CTestStateMachine" außer Kraft gesetzt werden.
- "CTestStateMachine:: OnState1State2Effect()" kann auch auf "abstract" gesetzt werden und Sie erhalten Kompilierfehler bis die Methode außer Kraft gesetzt wird.
- Wenn "OnState1State2Effect()" nicht "abstract" ist und die Option "Debug-Meldungen generieren" aktiv ist, generiert UModel die folgende Debug-Ausgabe:

```
// Override to handle entry/exit/do actions, transition effects,...:
public virtual void OnState1State2Effect() {OnDebugMessage("ACTION:
OnState1State2Effect");}
```

Beispiel 3 - eine einfache Transition mit einem Effekt-Parameter



Die entsprechende Operation wird in UModel automatisch generiert.



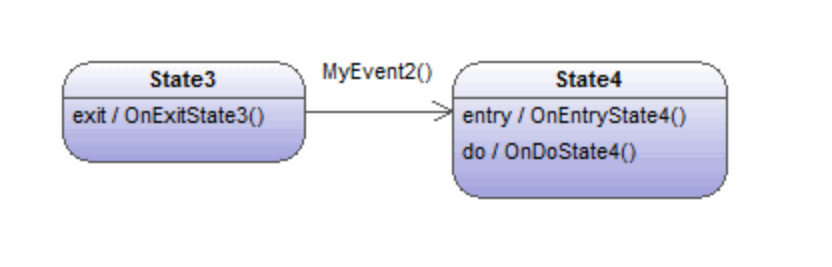
Generierte Methode im Code:

```
private class CTestStateMachine : IState
{
        …
        // Additional defined operations of the controller class:
        public virtual void OnState1State2Effect(String text)
        {
        }
}
```

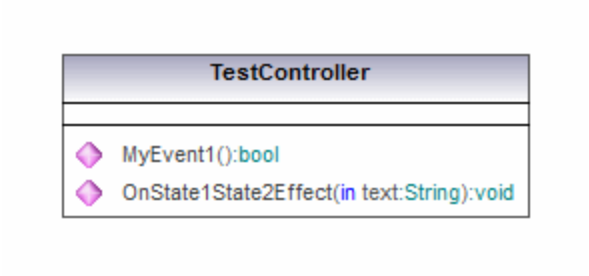
- Zur Durchführung von (automatisch mit UModel erzeugten) Operationen können Parameter manuell hinzugefügt werden (UModel kann nicht wissen, welcher Typ benötigt wird).
- In diesem Beispiel wurde der Parameter "text:String" zur Effekt-Methode in TestController hinzugefügt. Beim Aufruf dieser Methode muss ein ordnungsgemäßes Argument definiert werden (hier: "1 => 2").
- Eine andere Möglichkeit wäre z.B. die folgende: Aufruf von statischen Methoden ("MyStatic.OnState1State2Effect("1 => 2")") oder Methoden von Singletons ("getSingleton().OnState1State2Effect("1 => 2")").

© 2016-2022 Altova GmbH

Beispiel 4 - entry/exit/do-Aktionen



Die entsprechenden Operationen werden in UModel automatisch generiert.



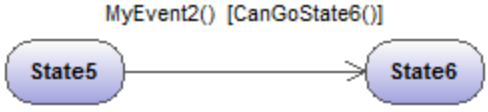
Generierte Methode im Code:

```
private class CTestStateMachine : IState
{
        …
        // Override to handle entry/exit/do actions, transition effects,...:
        public virtual void OnExitState3() {}
        public virtual void OnEntryState4() {}
        public virtual void OnDoState4() {}
}
```

- Zustände können entry/exit/do-Verhalten aufweisen. UModel generiert automatisch die entsprechenden Operationen zu deren Behandlung.
- Wenn im Beispiel oben "MyEvent2()" eintritt, ruft die Zustandsdiagrammimplementierung "OnExitState3()" auf. Wenn "MyEvent2" einen Effekt hätte, würde dieser in der Folge aufgerufen werden. Anschließend würden "OnEntryState4" und "OnDoState4" aufgerufen werden.
- Normalerweise sollten diese Methoden außer Kraft gesetzt werden. Wenn sie nicht abstrakt sind und die Option "Debug-Meldungen generieren" aktiv ist, liefert UModel eine Standard-Debug-Ausgabe, wie in Beispiel 2 beschrieben.
- Diese Methoden können auch Parameter haben, wie in Beispiel 3 gezeigt.

Beispiel 5 - Guards

Transitionen können Guards (Wächterausdrücke) haben, die ermitteln, ob die Transition wirklich ausgelöst werden kann.



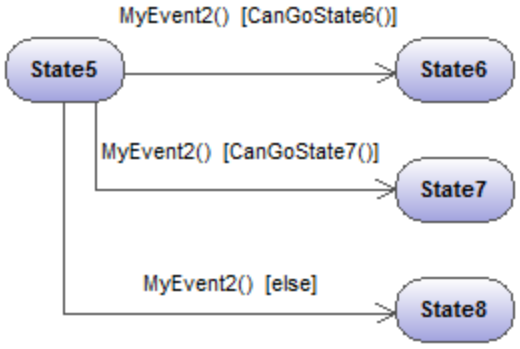
Die entsprechende Operation wird in UModel automatisch generiert.



Generierte Methode im Code:

```
private class CTestStateMachine : IState
{
        …
        // Additional defined operations of the controller class:
        public virtual bool CanGoState6()
        {
            return true; // Override!
        }
}
```

- Wenn "State5" der aktive Zustand ist und "MyEvent2" eintritt, so ruft die Zustandsdiagrammimplementierung "CanGoState6" auf und je nach Ergebnis wird die Transition ausgelöst oder nicht.
- Normalerweise sollten diese Methoden außer Kraft gesetzt werden. Wenn sie nicht abstrakt sind und die Option "Debug-Meldungen generieren" aktiv ist, liefert UModel eine Standard-Debug-Ausgabe, wie in Beispiel 2 beschrieben.
- Diese Methoden können auch Parameter haben, wie in Beispiel 3 gezeigt.
- Es sind mehrere Transitionen mit demselben Ereignis aber unterschiedlichen Guards möglich. Die Reihenfolge, in der die verschiedenen Guards abgefragt werden, ist nicht definiert. Wenn eine Transition keinen Guard hat oder der Guard "else" ist, wird sie als der letzte Guard betrachtet (d.h. dieser Guard wird nur dann ausgelöst, wenn alle anderen Transitions-Guards "false" zurückgeben). So ist z.B. im Diagramm unten nicht definiert, ob `CanGoState6()` oder `CanGoState7()` zuerst aufgerufen wird. Die dritte Transition wird nur ausgelöst, wenn `CanGoState6()` und `CanGoState7()` false zurückgeben.



Weitere Konstrukte und Funktionalitäten finden Sie in den Beispielen **AirCondition.ump** und **Complex.ump**.

© 2016-2022 Altova GmbH

9.1.2.6 Zustandsdiagramm-Elemente

 Startzustand (Pseudozustand)
Der Beginn eines Prozesses.

 Endzustand
Das Ende der Abfolge von Prozessen.

 Eintrittspunkt (Pseudozustand)
Der Eintrittspunkt eines Zustandsautomaten oder eines zusammengesetzten Zustands.

 Austrittspunkt (Pseudozustand)
Der Austrittspunkt eines Zustandsautomaten oder eines zusammengesetzten Zustands.

 Entscheidung
Dieses Element stellt eine dynamische bedingte Verzweigung dar, wobei einander gegenseitig ausschließende Guard Trigger ausgewertet werden (OR Operation).

 Kreuzung (Pseudozustand)
Dieses Element stellt das Ende der OR-Operation dar, die durch das Element "Entscheidung" definiert ist.

 Beendung (Pseudozustand)
Das Anhalten der Ausführung des Zustandsautomaten.

 Gabelung (Pseudozustand)
Fügt eine vertikale Gabelungsleiste ein.
Dient zum Aufteilen von Sequenzen in nebenläufige Untersequenzen.

 Gabelung horizontal (Pseudozustand)
Fügt eine horizontale Gabelungsleiste ein.
Dient zum Aufteilen von Sequenzen in nebenläufige Untersequenzen.

 Vereinigung (Pseudozustand)
Vereinigt/führt zuvor definierte Untersequenzen zusammen. Alle Aktivitäten müssen abgeschlossen sein, bevor Sie fortfahren können.

Vereinigung, horizontal (Pseudozustand)

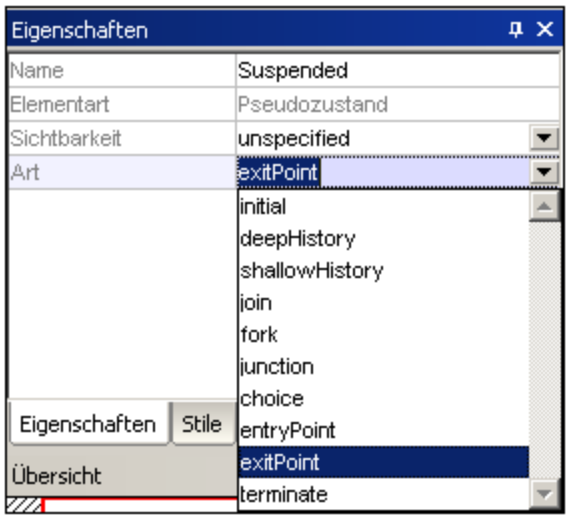
Vereinigt/führt zuvor definierte Untersequenzen zusammen. Alle Aktivitäten müssen abgeschlossen sein, bevor Sie fortfahren können.

DeepHistory

Ein Pseudozustand, der den zuvor aktiven Zustand in einem zusammengesetzten Zustand wiederherstellt.

ShallowHistory

Ein Pseudozustand, der den Ausgangszustand eines zusammengesetzten Zustands wiederherstellt. Alle Pseudozustandselemente können in einen anderen "Typ" geändert werden, indem Sie auf dem Register "Eigenschaften" den Eintrag in der Auswahlliste "**Art**" ändern.



Verbindungspunktreferenz

Eine Verbindungspunktreferenz stellt eine Verwendung (als Teil eines Unterautomatenzustands) eines Eintritts-/Austrittspunkts dar, der in der Zustandsautomatenreferenz durch den Unterautomatenzustand definiert ist.

So fügen Sie Eintritts- oder Austrittspunkte zu einer Verbindungspunktreferenz hinzu:

- Der Zustand, mit dem der Punkt verbunden ist, muss selbst einen Unterautomatenzustand referenzieren. (sichtbar auf dem Register "Eigenschaften").
- Dieser Unterautomat muss einen oder mehrere Eintritts- oder Austrittspunkte enthalten.

Transition

Eine direkte Beziehung zwischen zwei Zuständen. Ein Objekt im ersten Zustand führt eine oder mehrere Aktionen durch und tritt anschließend abhängig von einem Ereignis und der Erfüllung etwaiger Guard-

© 2016-2022 Altova GmbH

Bedingungen in den zweiten Zustand ein. Transitionen haben einen Event-Trigger, (eine) Guard-Bedingung(en), eine Aktion (Verhalten) und einen Zielzustand. Die folgenden Ereignis-Unterelemente werden unterstützt:

- ReceiveSignalEvent
- SignalEvent
- SendSignalEvent
- ReceiveOperationEvent
- SendOperationEvent
- ChangeEvent.



Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten

Wenn Sie die Schaltfläche "Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten" aktivieren, wird die entsprechende Operation in der referenzierten Klasse automatisch erstellt, wenn Sie eine Transition erstellen und einen Namen myOperation() eingeben.

Anmerkung: Operationen können nur dann automatisch erstellt werden, wenn sich der Zustandsautomat innerhalb einer Klasse oder Schnittstelle befindet.

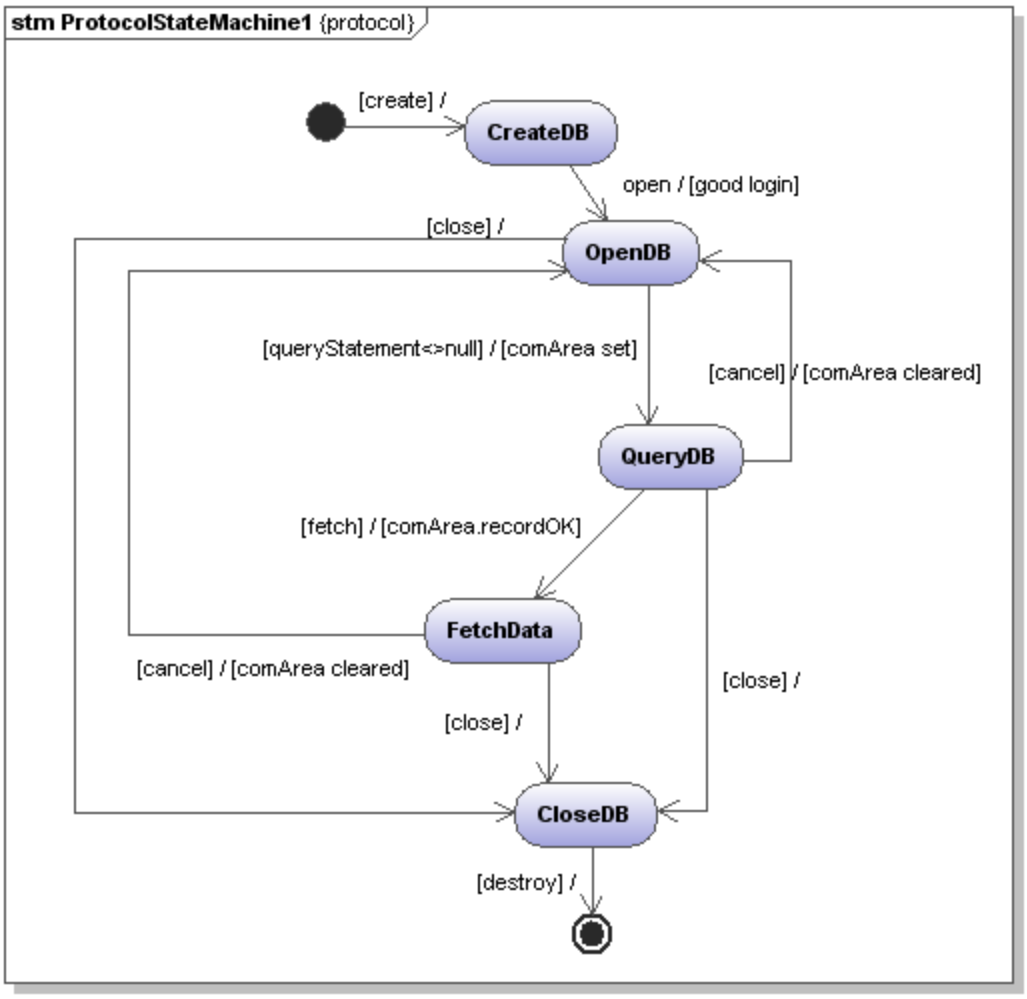
9.1.3 Protokoll-Zustandsautomat

Altova Website: UML-Protokoll-Zustandsdiagramme

Mit Hilfe von Protokoll-Zustandsautomaten wird eine **Sequenz** von Ereignissen dargestellt, auf die ein Objekt reagiert, ohne dass das spezifische Verhalten dargestellt werden muss. In diesem Diagramm werden die benötigte Ereignissequenz und die resultierenden Änderungen am Zustand des Objekts modelliert.

Meist dienen Protokoll-Zustandsautomaten zur Beschreibung komplexer Protokolle, z.B. zur Beschreibung des Datenbankzugriffs über eine bestimmte Schnittstelle oder von Kommunikationsprotokollen wie TCP/IP.

Protokoll-Zustandsautomaten werden auf dieselbe Weise wie Zustandsdiagramme erstellt, haben aber weniger Modellierungselemente. Die Protokoll-Übergänge zwischen Zuständen können Vor- und Nachbedingungen haben, die definieren, was zutreffen, also "true" sein muss, damit der Übergang in einen anderen Zustand erfolgen kann, oder was der resultierende Zustand nach dem Übergang sein muss.



9.1.3.1 Einfügen von Protokoll-Zustandsdiagramm-Elementen



Einfügen über die Symbole der Symbolleiste:

1. Klicken Sie in der Symbolleiste auf das Symbol "Protokoll-Zustandsautomat".

2. Klicken Sie in das Protokoll-Zustandsdiagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen von bestehenden Elementen in das Protokoll-Zustandsdiagramm:

Die meisten Elemente, die in anderen Protokoll-Zustandsdiagrammen vorkommen, können in ein bestehendes Diagramm eingefügt werden.

© 2016-2022 Altova GmbH

1. Suchen Sie das gewünschte Element auf dem Register Modell-Struktur (Sie können dazu das Suchfunktionstextfeld verwenden oder Strg + F drücken, um nach einem beliebigen Element zu suchen).
2. Ziehen Sie das/die Element(e) in das Protokoll-Zustandsdiagramm.

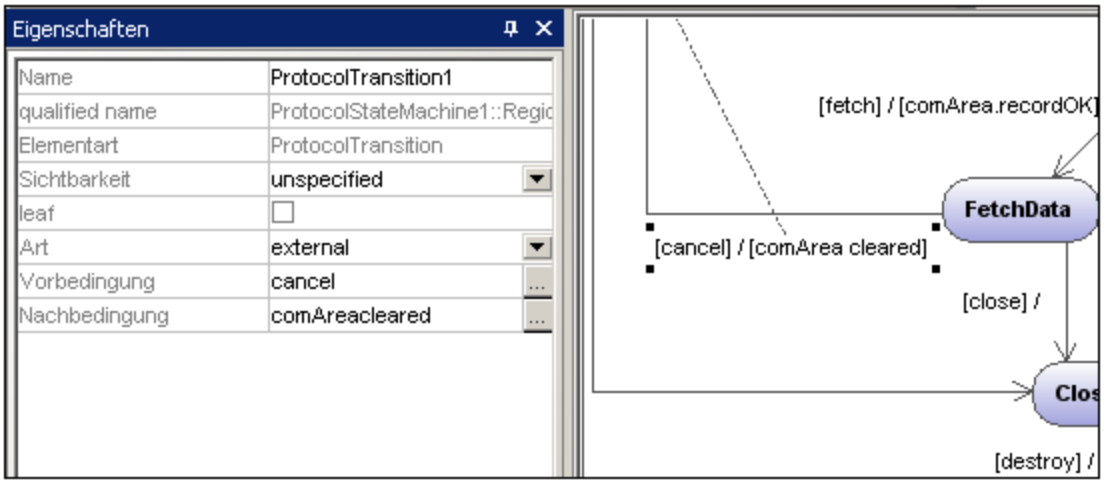
So fügen Sie einen einfachen Zustand ein:

1. Klicken Sie in der Symbolleiste auf das Zustandssymbol  und anschließend in das Protokoll-Zustandsdiagramm, um es einzufügen.
2. Geben Sie den Namen des Zustands ein und drücken Sie zur Bestätigung die Eingabetaste. Einfache Zustände haben keine Regionen oder andere Arten von Substrukturen.

So erstellen Sie eine Protokoll-Transition zwischen zwei Zuständen:

1. Klicken Sie auf den Transition-Ziehpunkt des Ausgangszustands (auf der rechten Seite des Elements) oder klicken Sie in der Symbolleiste auf die Schaltfläche "Protocol Transition".
2. Ziehen Sie den Transition-Pfeil mit der Maus auf den Ziel-Zustand.
   Der Textcursor ist automatisch so eingestellt, dass Sie die Vor- und/oder Nachbedingung eingeben können. Verwenden Sie unbedingt die eckigen Klammern [] und den Schrägstrich, wenn Sie die Bedingungen direkt eingeben.

   Wenn Sie die Vor-/Nachbedingung im Fenster "Eigenschaften" eingeben, werden die eckigen Klammern und der Schrägstrich automatisch in das Diagramm eingefügt.



So erstellen Sie Elemente zusammengesetzter Zustände und Unterautomatenzustände bzw. fügen diese ein:

- Siehe Zusammengesetzte Zustände[383]

9.1.3.2 Protokoll-Zustandsdiagramm-Elemente



Zustand

Ein einfaches Zustandselement mit nur einem Bereich.



Zusammengesetzter Zustand

Diese Art von Zustand enthält einen zweiten Bereich bestehend aus einer einzigen Region. Innerhalb dieser Region können beliebig viele Zustände platziert werden.



Orthogonaler Zustand

Diese Art von Zustand enthält einen zweiten Bereich bestehend aus zwei oder mehreren Regionen, wobei separate Regionen Gleichzeitigkeit kennzeichnen.

Wenn Sie mit der rechten Maustaste auf einen Zustand klicken und den Befehl **Neu | Region** auswählen, können Sie neue Regionen hinzufügen.



Unterautomatenzustand

Mit Hilfe dieses Zustands können Sie die Einzelheiten eines Zustandsautomaten ausblenden. Dieser Zustand hat keine Regionen, ist aber mit einem separaten Zustandsautomaten verknüpft.



Startzustand (Pseudozustand)

Der Beginn eines Prozesses.



Endzustand

Das Ende der Abfolge von Prozessen.



Eintrittspunkt (Pseudozustand)

Der Eintrittspunkt eines Zustandsautomaten oder eines zusammengesetzten Zustands.



Austrittspunkt (Pseudozustand)

Der Austrittspunkt eines Zustandsautomaten oder eines zusammengesetzten Zustands.



Entscheidung

Dieses Element stellt eine dynamische bedingte Verzweigung dar, wobei einander gegenseitig ausschließende Guard Trigger ausgewertet werden (OR Operation).

© 2016-2022 Altova GmbH

 Kreuzung (Pseudozustand)

Dieses Element stellt das Ende der OR-Operation dar, die durch das Element "Entscheidung" definiert ist.

 Beendung (Pseudozustand)

Das Anhalten der Ausführung des Zustandsautomaten.

 Gabelung (Pseudozustand)

Fügt eine vertikale Gabelungsleiste ein.
Dient zum Aufteilen von Sequenzen in nebenläufige Untersequenzen.

 Gabelung horizontal (Pseudozustand)

Fügt eine horizontale Gabelungsleiste ein.
Dient zum Aufteilen von Sequenzen in nebenläufige Untersequenzen

 Vereinigung (Pseudozustand)

Vereinigt/führt zuvor definierte Untersequenzen zusammen. Alle Aktivitäten müssen abgeschlossen sein, bevor Sie fortfahren können.

 Vereinigung, horizontal (Pseudozustand)

Vereinigt/führt zuvor definierte Untersequenzen zusammen. Alle Aktivitäten müssen abgeschlossen sein, bevor Sie fortfahren können.

 Verbindungspunktreferenz

Eine Verbindungspunktreferenz stellt eine Verwendung (als Teil eines Unterautomatenzustands) eines Eintritts-/Austrittspunkts dar, der in der Zustandsautomatenreferenz durch den Unterautomatenzustand definiert ist.

So fügen Sie Eintritts- oder Austrittspunkte zu einer Verbindungspunktreferenz hinzu:

- Der Zustand, mit dem der Punkt verbunden ist, muss selbst einen Unterautomatenzustand referenzieren. (sichtbar auf dem Register "Eigenschaften").
- Dieser Unterautomat muss einen oder mehrere Eintritts- oder Austrittspunkte enthalten.

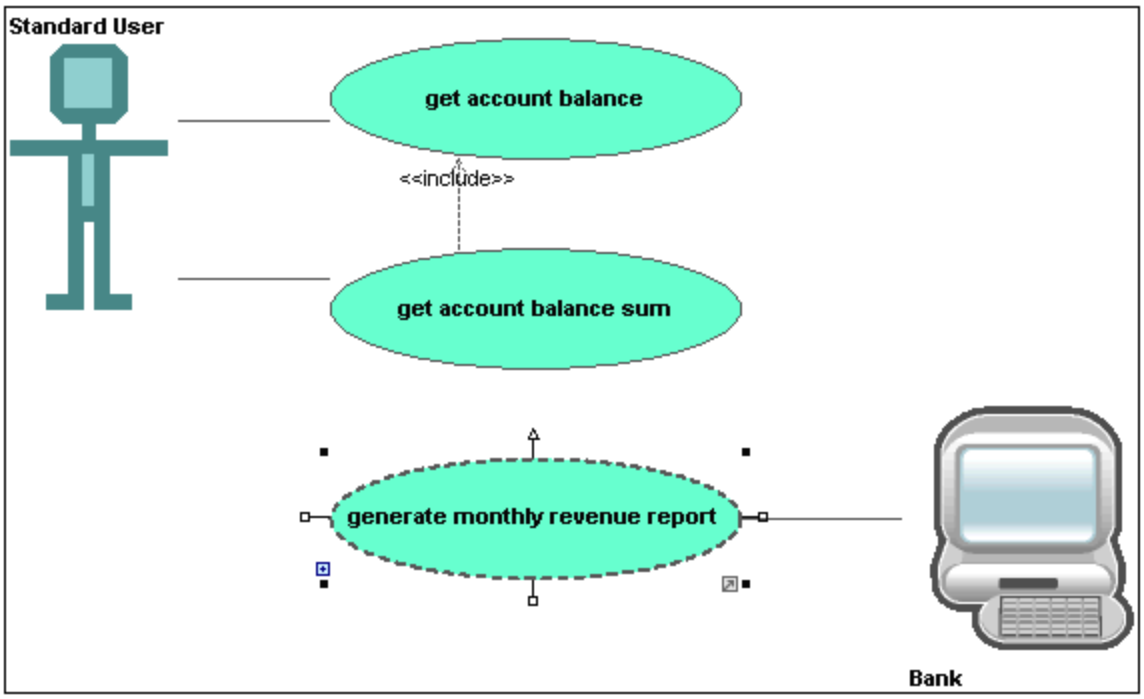
 Protocol Transition

Eine direkte Beziehung zwischen zwei Zuständen. Ein Objekt im ersten Zustand führt eine oder mehrere Aktionen durch und tritt anschließend abhängig von einem Ereignis und der Erfüllung etwaiger Vor- oder Nachbedingungen in den zweiten Zustand ein.

Nähere Informationen dazu finden Sie unter Einfügen von Protokoll-Zustandselementen 399.

9.1.4 Use Case-Diagramm

Eine Anleitung zum Hinzufügen von Klassen zum Diagramm finden Sie im Tutorial im Abschnitt Use Cases 22.



9.1.5 Kommunikationsdiagramm

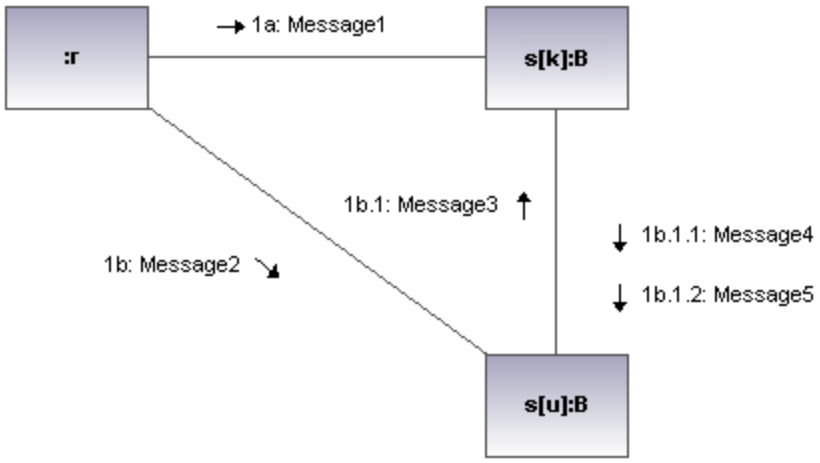
Altova Website: UML-Kommunikationsdiagramme

In Kommunikationsdiagrammen werden Interaktionen d.h. Nachrichtenflüsse zwischen Objekten zur Laufzeit und die Beziehungen zwischen den miteinander in Wechselbeziehung stehenden Objekten dargestellt. Im Grunde dienen diese Diagramme zum Modellieren des dynamischen Verhaltens von Anwendungsfällen (Use Cases).

Kommunikationsdiagramme weisen denselben Aufbau wie Sequenzdiagramme auf, mit Ausnahme dessen, dass die Notation ein anderes Layout aufweist. Zur Kennzeichnung der Reihenfolge der Nachrichten und der Schachtelung sind die Nachrichten nummeriert.

Sie können in UModel mit einer einzigen einfachen Aktion Kommunikationsdiagramme anhand von Sequenzdiagrammen erstellen und umgekehrt. Nähere Informationen dazu finden Sie unter "Generieren von Sequenzdiagrammen 406".

 © 2016-2022 Altova GmbH



9.1.5.1 Einfügen von Kommunikationsdiagrammelementen

Verwendung der Symbolleisten-Schaltflächen:

1. Klicken Sie in der Kommunikationsdiagramm-Symbolleiste auf das entsprechende Kommunikationssymbol.



2. Klicken Sie in das Kommunikationsdiagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen vorhandener Elemente in das Kommunikationsdiagramm

Elemente, die in anderen Diagrammen vorkommen, z.B. Klassen, können in ein Kommunikationsdiagramm eingefügt werden.

1. Suchen Sie das gewünschte Element auf dem Register Modell-Struktur (Sie können dazu das Suchfunktionstextfeld verwenden oder Strg + F drücken, um nach einem beliebigen Element zu suchen).
2. Ziehen Sie das/die Element(e) in das Kommunikationsdiagramm.

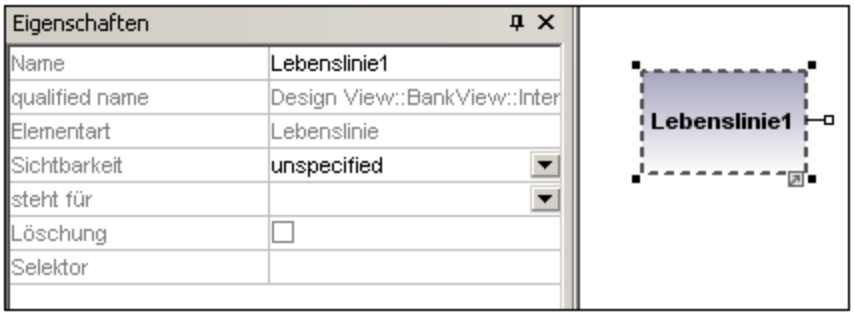
Lebenslinie

Das Element "Lebenslinie" ist Teil einer Interaktion mehrerer Elemente. Sie haben in UModel die Möglichkeit auch andere Elemente in das Kommunikationsdiagramm einzufügen, z.B. Klassen. Jedes dieser Elemente wird als neue Lebenslinie angezeigt. Sie können die Farben/Schattierung der Lebenslinie über die "Farbverlauf Titel"-Auswahllisten auf dem Register "Stile" neu definieren.

Zur Erstellung einer Lebenslinie mit **mehreren Zeilen**, drücken Sie **Strg + Eingabetaste**, um eine neue Zeile zu erstellen.

So fügen Sie eine Kommunikationslebenslinie ein:

1. Klicken Sie in der Titelleiste auf das Symbol "Lebenslinie" und anschließend auf das Kommunikationsdiagramm, um sie einzufügen.



2. Geben Sie einen Namen für die Lebenslinie ein, um den Standardnamen "Lebenslinie1" bei Bedarf zu ändern.

Nachrichten

Eine Nachricht ist ein Modellierungselement, das eine bestimmte Art von Kommunikation in einer Interaktion definiert. Bei einer Kommunikation kann es sich z.B. um das Auslösen eines Signals, den Aufruf einer Operation, das Erstellen oder Löschen einer Instanz handeln. Die Nachricht definiert den Absender und Empfänger und um welche Art von Kommunikation es sich handelt.

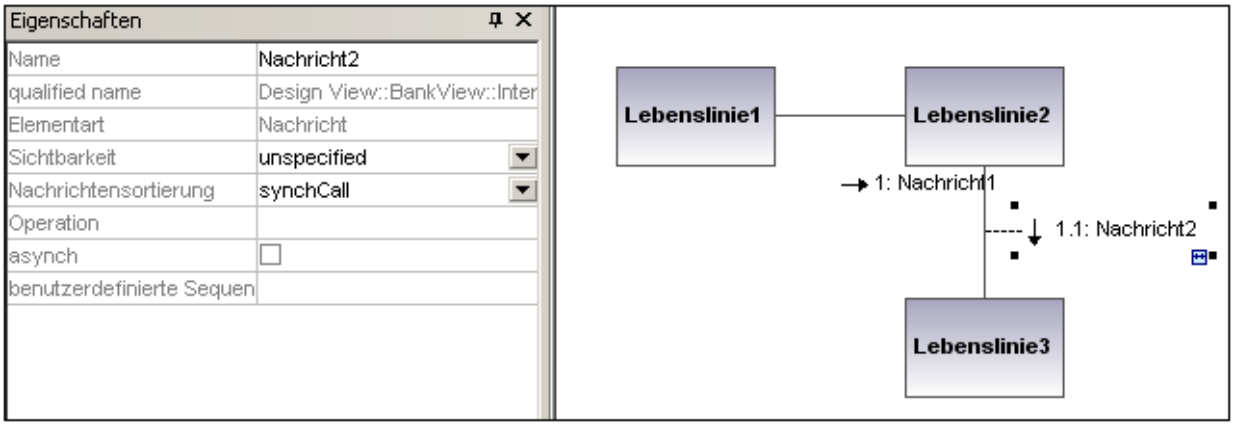
 Nachricht (Aufruf)  Nachricht (Antwort)  Nachricht (Erstellung)  Nachricht (Löschung)

So fügen Sie eine Nachricht ein:

1. Klicken Sie auf das entsprechende Nachrichtensymbol in der Symbolleiste.
2. Ziehen Sie die Nachrichtenlinie auf die empfangenden Objekte.

   Die Lebenslinie erscheint markiert, wenn die Nachricht an diese bestimmte Stelle gezogen werden kann.

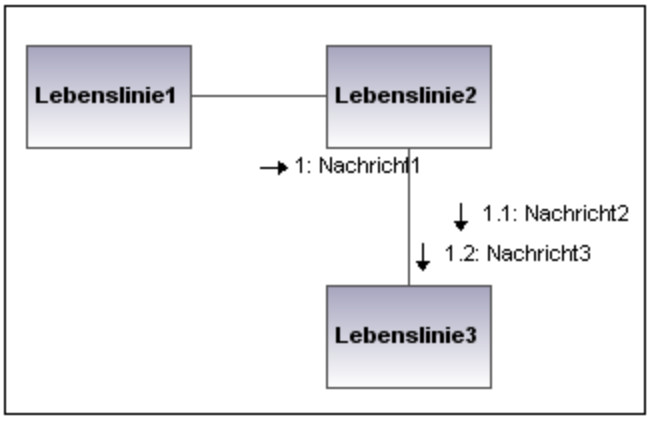
© 2016-2022 Altova GmbH



Anmerkung: Wenn Sie die Strg-Taste gedrückt halten, können Sie mit jedem Klick eine Nachricht einfügen.

So fügen Sie weitere Nachrichten ein:

1. Rechtsklicken Sie auf die vorhandene Kommunikationsverbindung und wählen Sie **Neu | Nachricht**.



- Die Richtung, in die Sie den Pfeil ziehen, bestimmt die Richtung der Nachricht. Antwortnachrichten können in jede der beiden Richtungen weisen.
- Wenn Sie auf ein Nachrichtensymbol klicken und dabei die Strg-Taste gedrückt halten, können Sie mehrere Nachrichten einfügen, indem Sie mehrmals ins Diagramm klicken und die Maus ziehen.

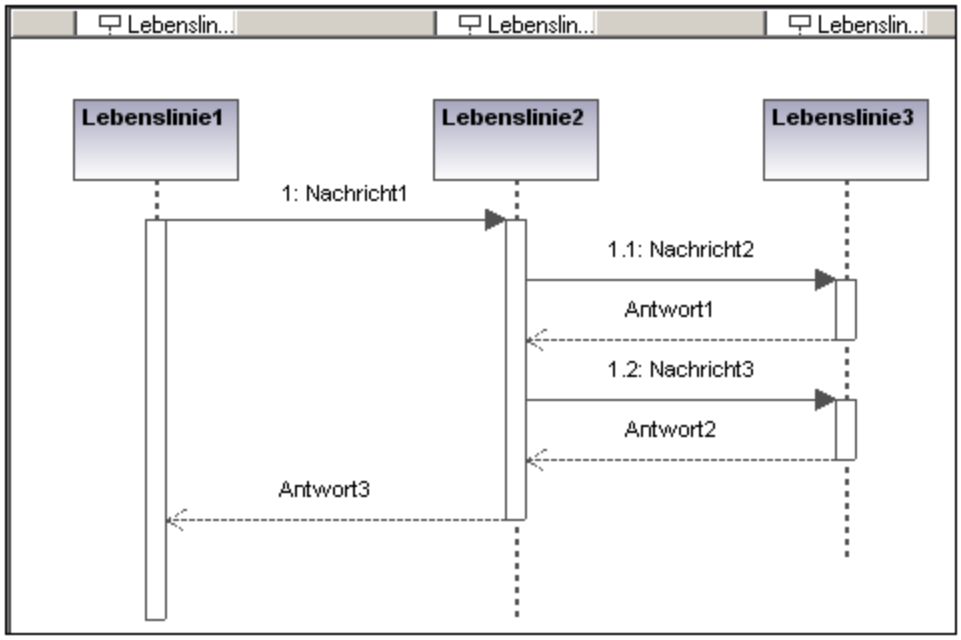
Nachrichtennummerierung

In Kommunikationsdiagrammen wird die Nummerierung in Dezimalschreibweise verwendet, wodurch die hierarchische Struktur der Nachrichten im Diagramm klarer ersichtlich ist. Bei der Reihenfolge handelt es sich um eine durch Punkte getrennte Liste von aufeinander folgenden Zahlen, gefolgt von einem Doppelpunkt und dem Namen der Nachricht.

Generieren von Sequenzdiagrammen anhand von Kommunikationsdiagrammen

Sie können in UModel in einer einzigen Aktion Kommunikationsdiagramme anhand von Sequenzdiagrammen generieren und umgekehrt,

- Rechtsklicken Sie an eine beliebige Stelle in einem Kommunikationsdiagramm und wählen Sie im Kontextmenü den Befehl **Sequenzdiagramm generieren.**

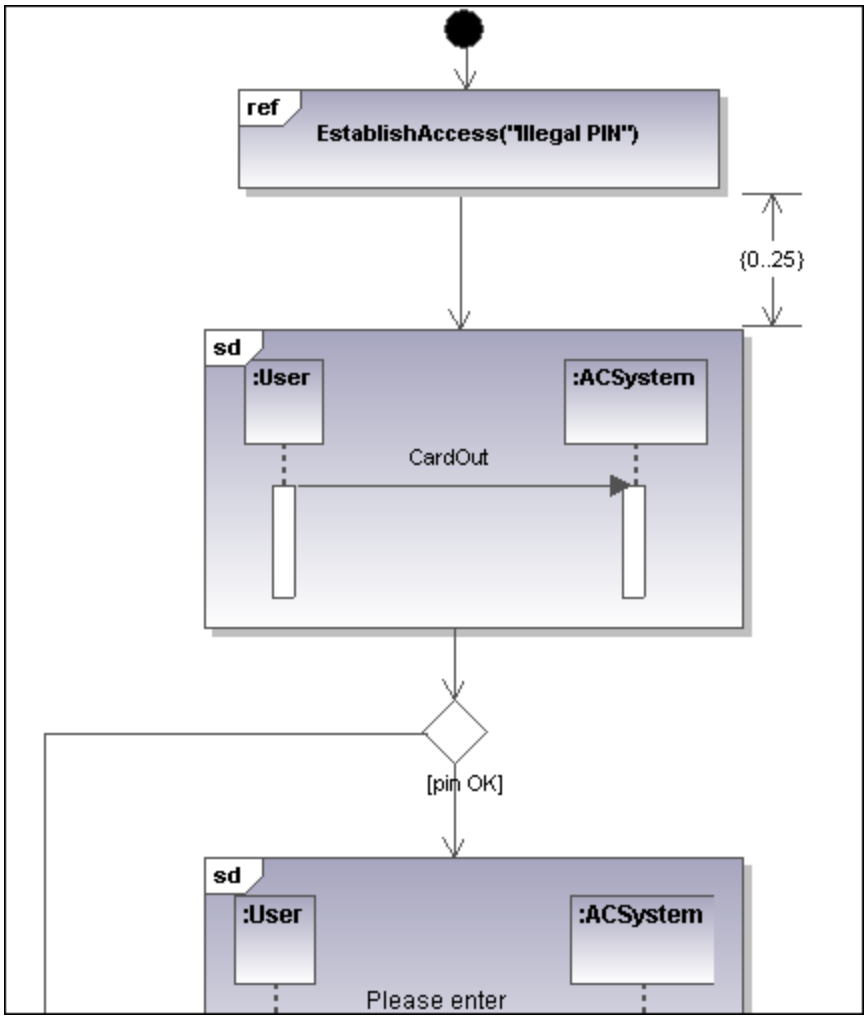


9.1.6 Interaktionsübersichtsdiagramm

Altova Website: UML-Interaktionsübersichtsdiagramme

Interaktionsübersichtsdiagramme sind eine Variante von Aktivitätsdiagrammen und geben Ihnen einen Überblick über die Interaktion zwischen anderen Interaktionsdiagrammen wie z.B. Sequenz-, Aktivitäts, Kommunikations- oder Zeitverlaufsdiagrammen. Interaktionsübersichtsdiagramme werden auf ähnliche Art wie Aktivitätsdiagramme und unter Verwendung derselben Modellierungselemente (Start-/Endpunkte, Gabelungen, Vereinigungen usw.) erstellt.

© 2016-2022 Altova GmbH



Anstelle von Aktivitätselementen werden zwei Arten von Interaktionselementen verwendet: Die Elemente "Interaktion" und "Interaktionsverwendung".

Sequenz-, Kommunikations-, Zeitverlaufs- oder Interaktionsübersichtsdiagramme werden als Elemente in Form grafischer Symbole innerhalb eines Rahmens dargestellt, in dessen linker oberer Ecke das Schlüsselwort "SD" angezeigt wird.

Elemente für Interaktionsinstanzen sind Referenzen, die auf vorhandene Interaktionsdiagramme verweisen, wobei im Titelbereich des Rahmens das Schlüsselwort "Ref" und innerhalb des Rahmens der Name der Interaktionsinstanz angezeigt wird.

9.1.6.1 Einfügen von Interaktionsübersichtselementen

Verwendung der Symbolleisten-Schaltflächen:

1. Klicken Sie in der Interaktionsübersichtsdiagramm-Symbolleiste auf das entsprechende Symbol



2. Klicken Sie in das Diagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

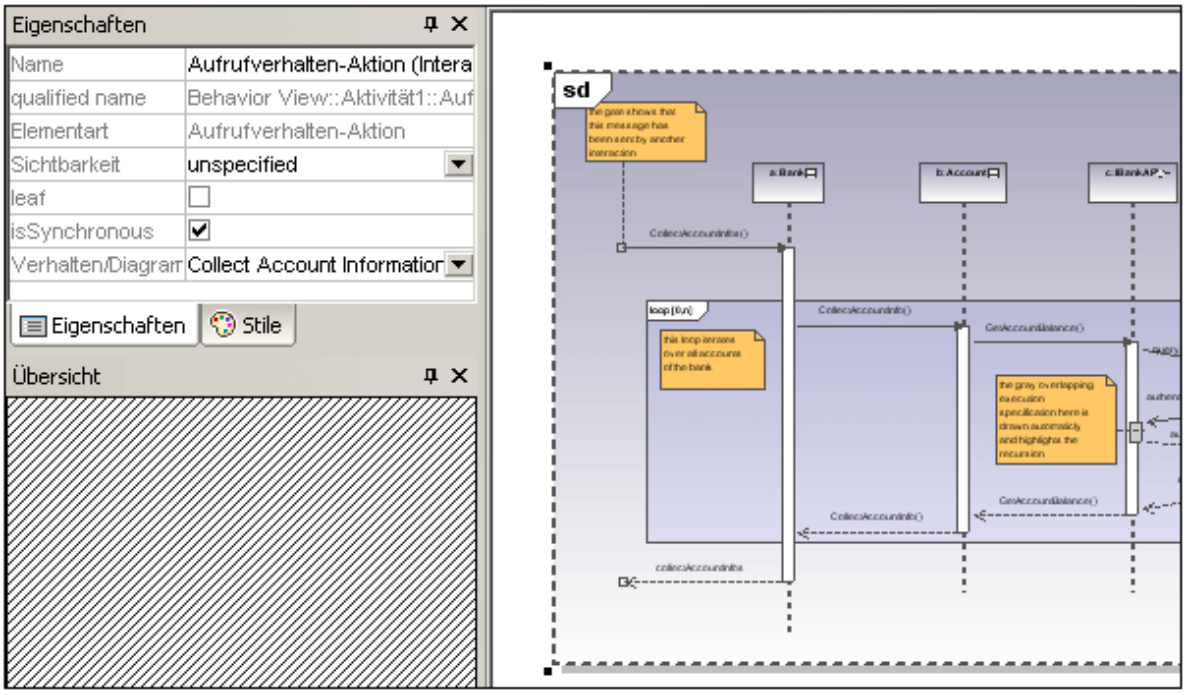
Ziehen vorhandener Elemente in das Interaktionsübersichtsdiagramm

Elemente, die in anderen Diagrammen vorkommen, z.B. Sequenz-, Aktivitäts-, Kommunikations- oder Zeitverlaufsdiagramme, können in ein Interaktionsübersichtsdiagramm eingefügt werden.

1. Suchen Sie das gewünschte Element auf dem Register Modell-Struktur (Sie können dazu das Suchfunktionstextfeld verwenden oder Strg + F drücken, um nach einem beliebigen Element zu suchen).
2. Ziehen Sie das/die Element(e) in das Diagramm.

Einfügen eines Interaktionselements

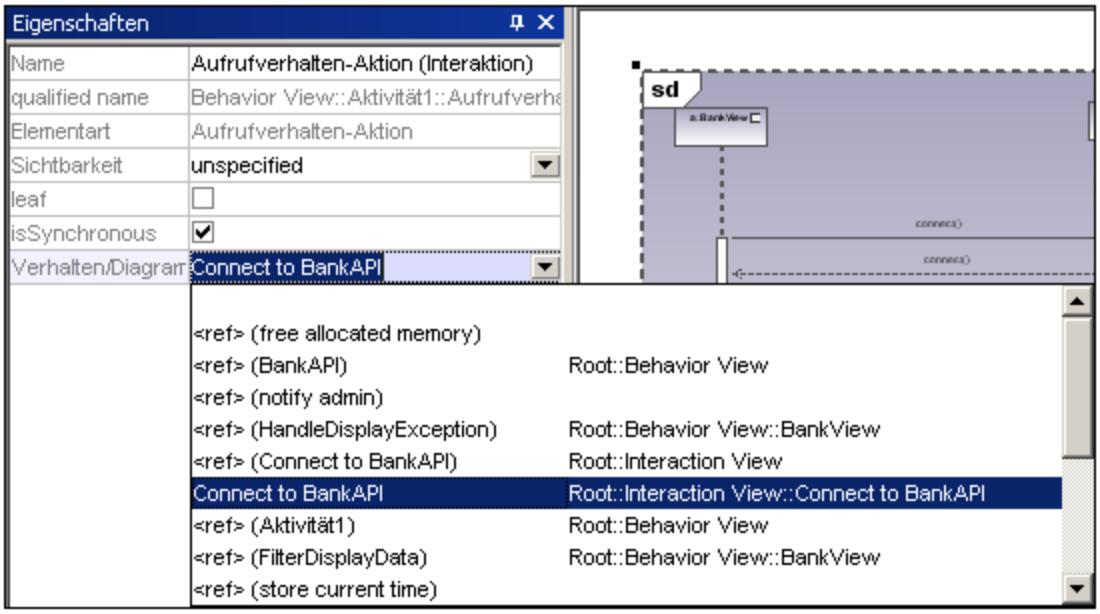
1. Klicken Sie in der Symbolleiste auf das Symbol "Aufrufverhalten-Aktion (Interaktion)"  und anschließend in das Interaktionsübersichtsdiagramm, um es einzufügen.



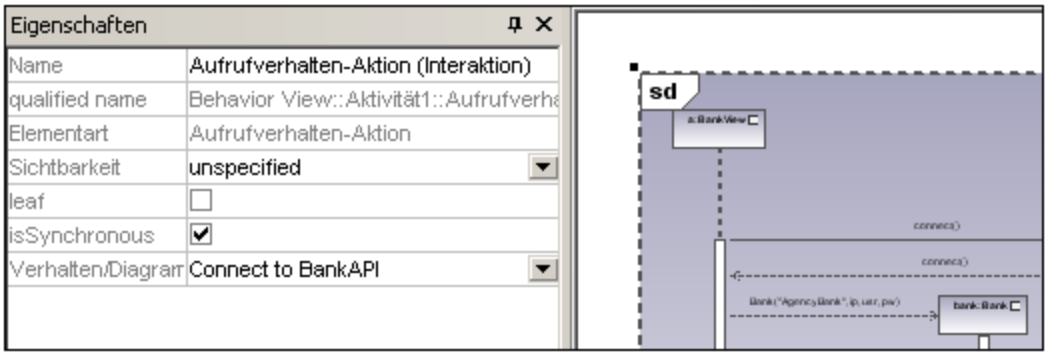
Wenn Sie die Beispieldatei Bank_Multilanguage.ump aus dem Ordner ...\Examples verwenden, wird das "Collect Account Information" Sequenzdiagramm automatisch eingefügt. Standardmäßig wird das erste in der Modell-Struktur gefundene Sequenzdiagramm ausgewählt.

2. Um ein anderes Interaktionselement auszuwählen, klicken Sie auf dem Register "Eigenschaften" auf die Auswahlliste **Verhalten/Diagramm**.
Daraufhin wird eine Liste aller Elemente angezeigt, die eingefügt werden können.

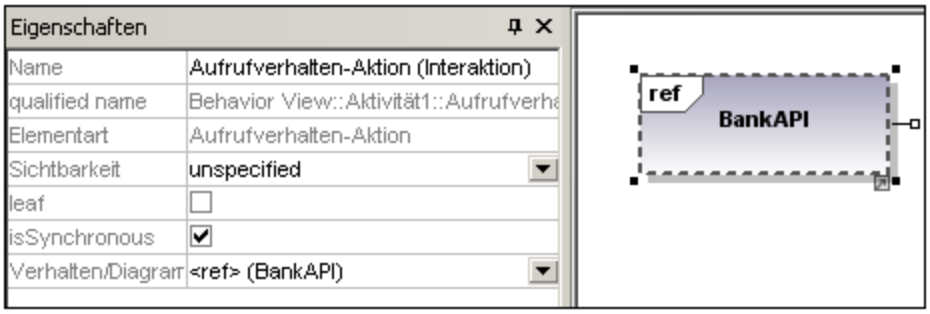
© 2016-2022 Altova GmbH



3. Klicken Sie auf das gewünschte Element, z.B. auf Connect to BankAPI.



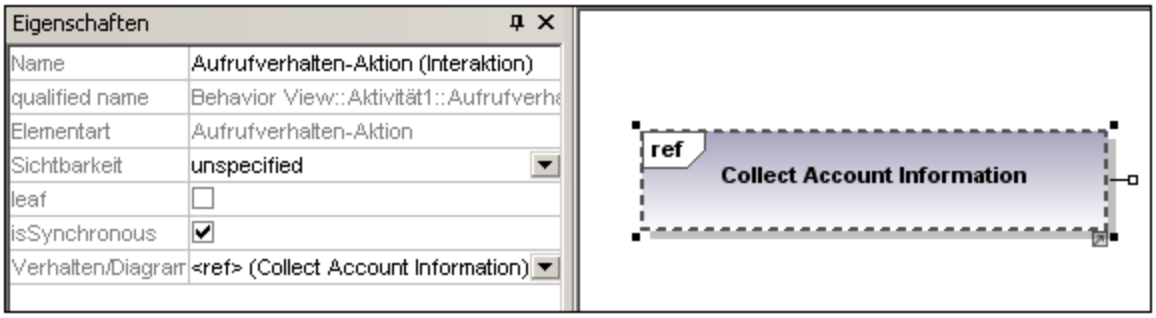
Da es sich dabei auch um ein Sequenzdiagramm handelt, wird das Interaktionselement als Symbol eines Sequenzdiagramms angezeigt.
Bei Auswahl von **<ref> BankAPI** wird die Instanz des Interaktionselements angezeigt.



Einfügen einer Interaktionselementinstanz

1. Klicken Sie in der Symbolleiste auf das Symbol "Aufrufverhalten-Aktion (Interaktionsverwendung)  und anschließend in das Interaktionsübersichtsdiagramm, um das Element einzufügen.

   Wenn Sie die Beispieldatei Bank_Multilanguage.ump aus dem Ordner ...**UModelExamples** verwenden, wird das "Collect Account Information" Sequenzdiagramm automatisch als Interaktionselement eingefügt. Standardmäßig wird das erste in der Modell-Struktur gefundene Sequenzdiagramm ausgewählt



2. Um das Interaktionselement zu ändern, doppelklicken Sie auf dem Register "Eigenschaften" auf die Auswahlliste **Verhalten**.
   Daraufhin wird eine Liste aller Elemente angezeigt, die eingefügt werden können.
3. Wählen Sie das gewünschte Element aus.
   Beachten Sie, dass alle mit dieser Methode eingefügten Elemente wie in der Abbildung oben angezeigt werden, d.h. mit "**ref**" im Titelbereich des Rahmens.

 Verzweigungsknoten

Fügt einen Verzweigungsknoten mit einer einzigen eingehenden Transition und mehreren mit Guards versehenen ausgehenden Transitionen ein. Nähere Informationen dazu finden Sie unter "Erstellen einer Verzweigung 361".

 Verbindungsknoten

Fügt einen Verbindungsknoten ein, der mehrere alternative durch den Verzweigungsknoten definierte Transitionen zusammenführt. Gleichzeitige Prozesse werden dabei von diesem Knoten nicht synchronisiert, sondern es wird einer der Prozesse ausgewählt.

 Startknoten

Der Anfang eines Aktivitätsprozesses. Eine Interaktion kann mehrere Startknoten haben.

 Aktivitätsendknoten

Das Ende des Interaktionsprozesses. Eine Interaktion kann mehrere Endknoten haben. Alle Flüsse werden gestoppt, wenn der "erste" Endknoten erreicht wird.

© 2016-2022 Altova GmbH



Parallelisierungsknoten

Fügt einen vertikalen Parallelisierungsknoten ein.
Dient zum Teilen von Flüssen in mehrere parallele Flüsse.



Parallelisierungsknoten (horizontal)

Fügt einen horizontalen Parallelisierungsknoten ein.
Dient zum Teilen von Flüssen in mehrere parallele Flüsse.



Synchronisationsknoten

Fügt einen vertikalen Synchronisationsknoten ein.
Ein Synchronisationsknoten synchronisiert mehrere durch den Parallelisierungsknoten definierte Flüsse.



Synchronisationsknoten (horizontal)

Fügt einen horizontalen Synchronisationsknoten ein.
Ein Synchronisationsknoten synchronisiert mehrere durch den Parallelisierungsknoten definierte Flüsse.



Zeitdauerbedingung hinzufügen

Eine Zeitdauer definiert eine Wertespezifikation, die eine Zeitdauer zwischen einem Start- und einem Endpunkt angibt. Eine Zeitdauer ist oft ein Ausdruck, der die Anzahl der Uhrticks darstellt, die während dieser Dauer verstreichen.



Kontrollfluss

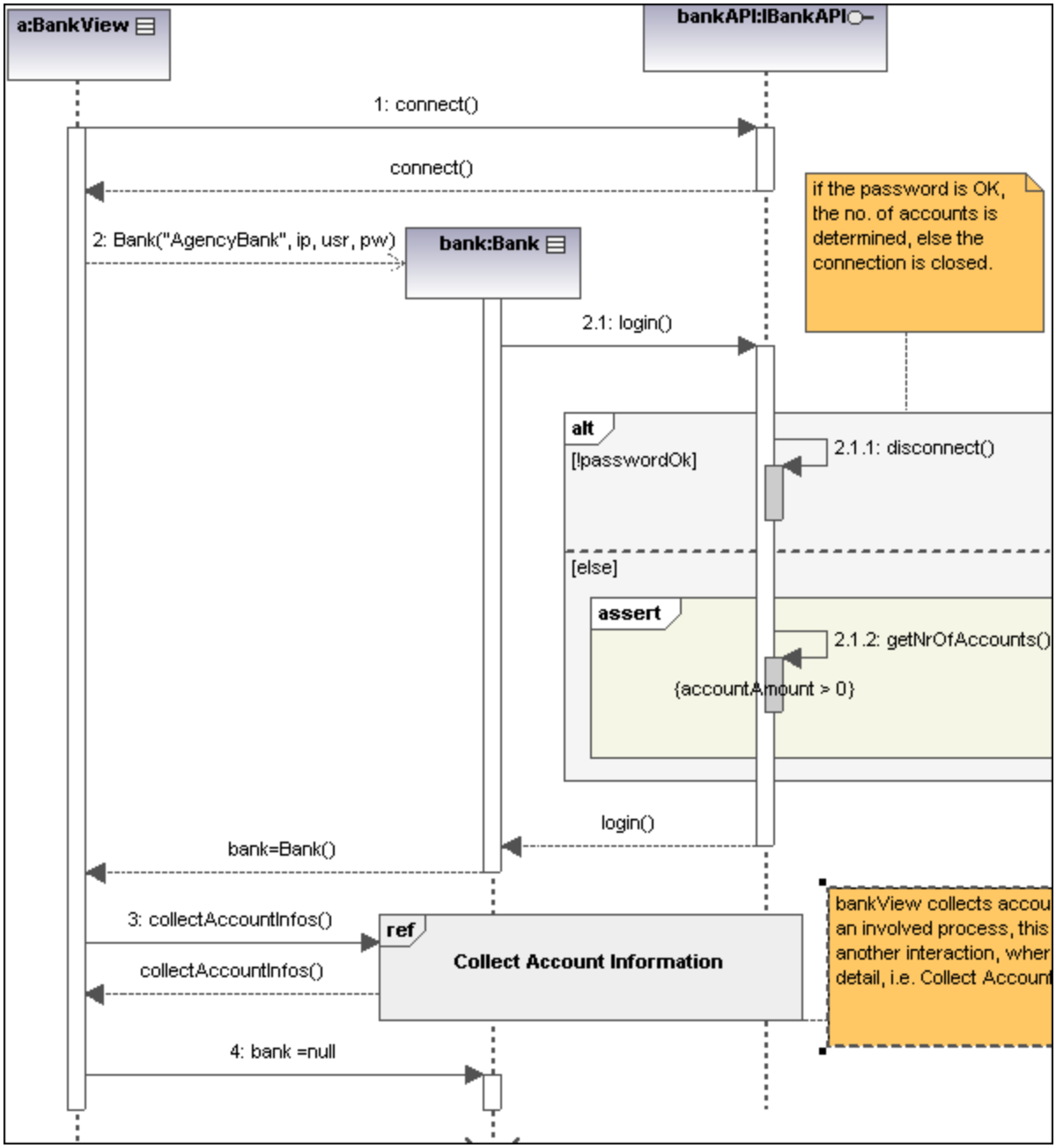
Ein Kontrollfluss ist eine Kante, d.h. eine mit einem Pfeil versehene Linie, die zwei Verhalten verbindet und eine Interaktion startet, nachdem die vorherige zu Ende geführt wurde.

9.1.7 Sequenzdiagramm

Altova Website: UML-Sequenzdiagramme

UModel unterstützt das in UML definierte Standard-Sequenzdiagramm und ermöglicht die einfache Manipulation von Objekten und Nachrichten zur Modellierung von Fallszenarien. Das im folgenden Abschnitt gezeigte Aktivitätsdiagramm steht im UModel-Ordner **...\UModelExamples** in den Beispielen **Bank_Java.ump**, **Bank_CSharp.ump** and **Bank_MultiLanguage.ump** zur Verfügung.

Sie können Sequenzdiagramme manuell modellieren oder diese alternativ dazu anhand von mit Reverse-Engineering erstelltem Quellcode generieren, wie unter Generieren von Sequenzdiagrammen anhand von Quellcode 426 beschrieben. Über die UModel API kann ein Sequenzdiagramm programmatisch generiert oder modelliert werden, siehe Anleitung zum Erstellen von Sequenzdiagrammen 864.



9.1.7.1 Einfügen von Sequenzdiagrammelementen

In einem Sequenzdiagramm werden dynamische Laufzeit-Objektbeziehungen mit Hilfe von Nachrichten modelliert. Sequenzdiagramme dienen im Allgemeinen zur Verdeutlichung einzelner Use Case-Szenarios.

- **Lebenslinien** sind die horizontal angeordneten Kästchen am oberen Rand des Diagramms. Zusammen mit einer strichlierten vertikalen Linie stellen sie das Objekt und seine Interaktionen in

© 2016-2022 Altova GmbH

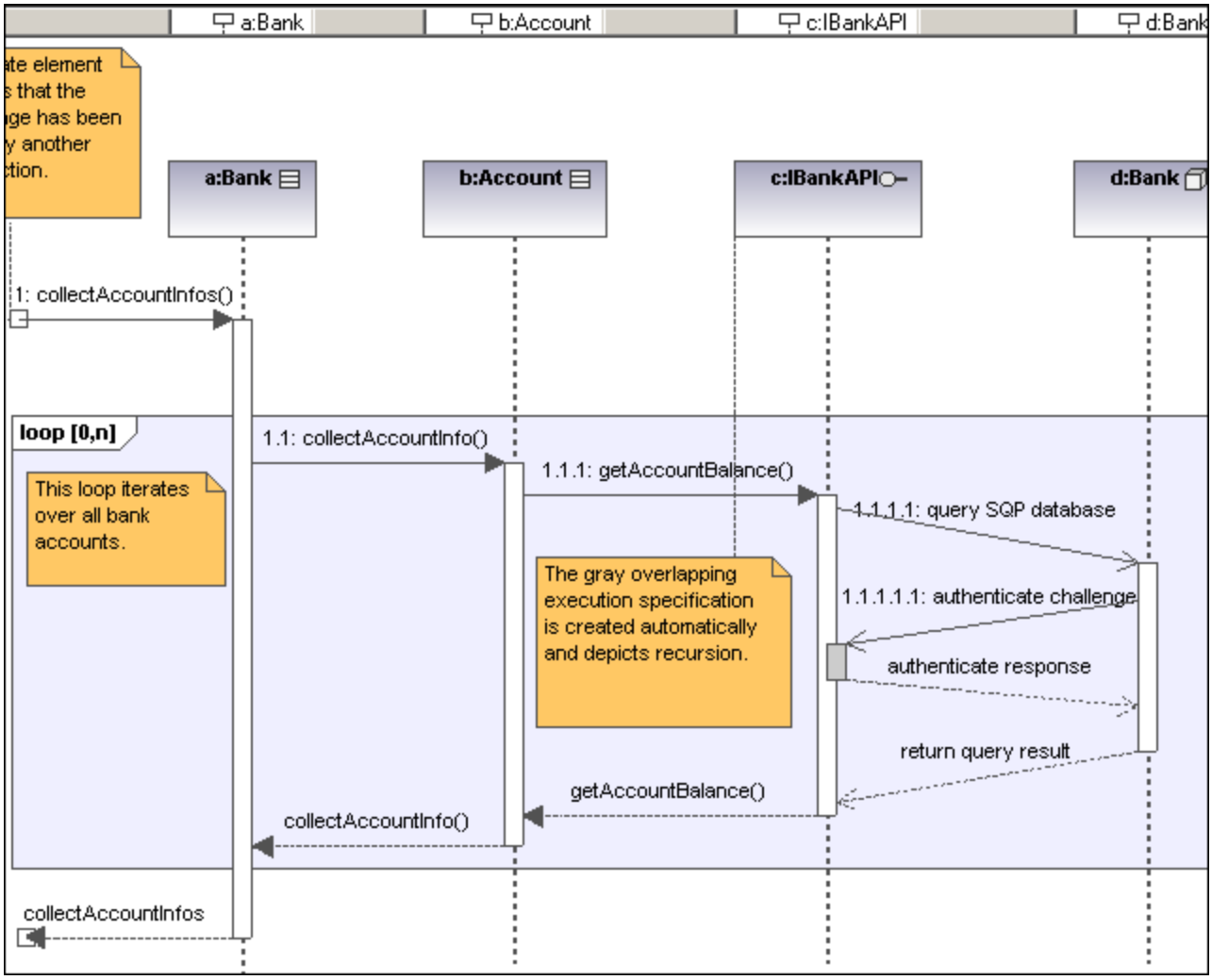
einem zeitlichen Ablauf dar. Nachrichten werden in Form von Pfeilen zwischen den Lebenslinien von zwei oder mehr Objekten dargestellt.

- **Nachrichten** werden zwischen sendenden und empfangenden Objekten gesendet und als Pfeile mit einer Bezeichnung dargestellt. Nachrichten können eine Sequenznummer und verschiedene weitere optionale Attribute haben: argument list usw. Es werden bedingte, optionale und alternative Nachrichten unterstützt. Nähere Informationen dazu finden Sie unter Combined Fragment [416].

Nähere Informationen finden Sie unter den folgenden Themen:

Lebenslinie [414]
Combined Fragment [416]
Interaction Use [419]
Gate [419]
Zustandsinvariante [420]
Nachrichten [420]

Sequenzdiagramm- und andere UModel-Elemente können auf verschiedene Arten in ein Sequenzdiagramm eingefügt werden.



Über die Symbolleistensymbole

1. Klicken Sie in der Sequenzdiagrammsymbolleiste auf das jeweilige Sequenzdiagrammsymbol.
2. Klicken Sie in das Sequenzdiagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen bestehender Elemente in das Sequenzdiagramm

Die meisten Classifier-Arten sowie Elemente, die in anderen Sequenzdiagrammen vorkommen, können in ein bestehendes Sequenzdiagramm eingefügt werden.

1. Suchen Sie das gewünschte Element auf dem Register "Modell-Struktur" (Sie können dazu das Suchfunktionstextfeld verwenden oder Strg + F drücken).
2. Ziehen Sie das Element/die Elemente in das Sequenzdiagramm.

9.1.7.1.1 Lebenslinie

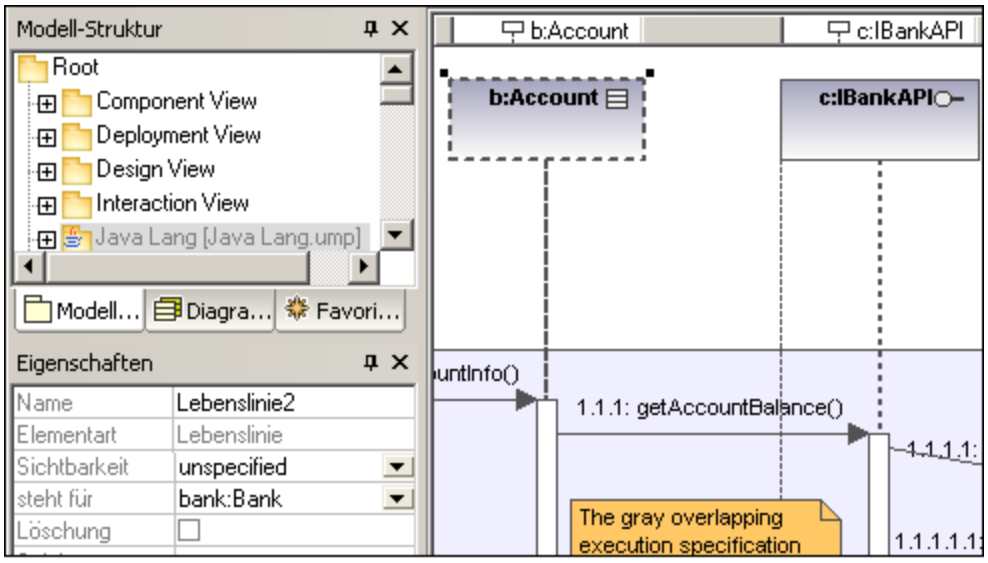
Das Element **Lebenslinie**  ist als einzelnes Element an einer Interaktion beteiligt. Sie haben in UModel die Möglichkeit auch andere Elemente in das Sequenzdiagramm einzufügen, z.B. Klassen und Akteure. Jedes dieser Elemente wird als neue Lebenslinie angezeigt, sobald es vom Register "Modell-Struktur" in das Diagrammfenster gezogen wurde.

Die Bezeichnung der Lebenslinie wird in einer Leiste am oberen Rand des Sequenzdiagramms angezeigt. Die Beschriftungen können in der Leiste neu positioniert und in der Größe angepasst werden, wobei die Änderungen sofort im Diagrammfenster zu sehen sind. Auch die Farben/Schattierung der Beschriftung lassen sich über die Farbverlaufsauswahllisten für den Titel auf dem Register "Stile" anpassen.

Zur Erstellung einer Lebenslinie mit **mehreren Zeilen**, drücken Sie **Strg + Eingabetaste**, um eine neue Zeile zu erstellen.

Die meisten Arten von Classifiern können in ein Sequenzdiagramm eingefügt werden. Im Feld "steht für" auf dem Register "Eigenschaften" wird der Elementtyp angezeigt, der als Lebenslinie fungiert. Wenn Sie Eigenschaften, deren Typ definiert ist, in ein Sequenzdiagramm ziehen, wird ebenfalls eine Lebenslinie erstellt.

© 2016-2022 Altova GmbH



Ausführungsspezifikation (Objektaktivierung)

Eine Ausführungsspezifikation (Aktivierung) wird in Form eines Kästchens (Rechteck) auf der Objektlebenslinie dargestellt. Eine Aktivierung ist die Ausführung einer Prozedur und die Zeit, die benötigt wird, um etwaige geschachtelte Prozeduren auszuführen. Aktivierungskästchen werden automatisch erstellt, wenn zwischen zwei Lebenslinien eine Nachricht erstellt wird.

Eine rekursive Nachricht oder Selbstnachricht (self message), also eine Nachricht, die eine andere Methode in derselben Klasse aufruft), erstellt gestapelte Aktivierungskästchen.

Ein-/Ausblenden von Aktivierungskästchen:

1. Klicken Sie auf das Register **Stile** und scrollen Sie zum unteren Ende der Liste.
   Über das Auswahlfeld "**Ausführungsspezifikationen anzeigen**" können Sie die Aktivierungskästchen im Sequenzdiagramm ein- und ausblenden.

Lebenslinienattribute

Über das Kontrollkästchen **Löschung** können Sie einen Lösch-Marker oder Stopp zur Lebenslinie hinzufügen, ohne eine Löschnachricht verwenden zu müssen.

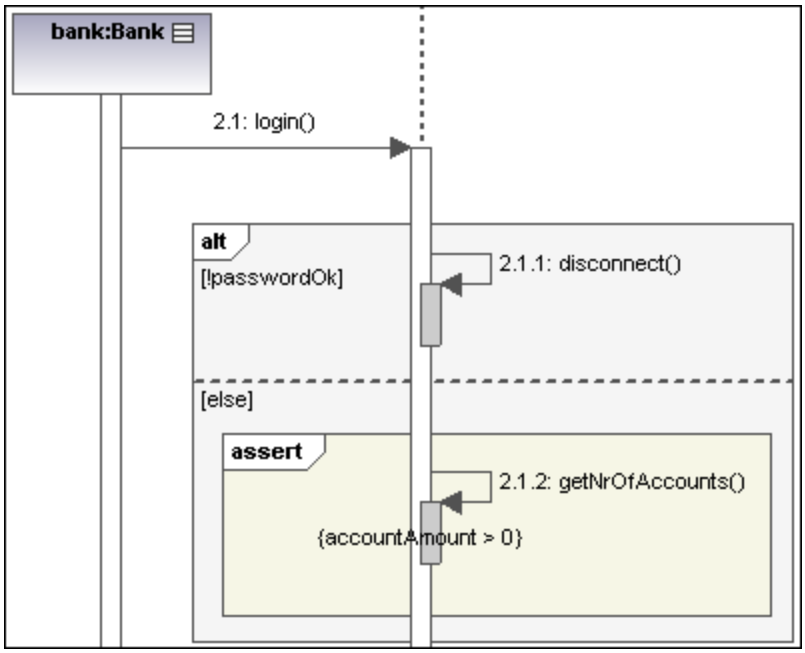
Über das Feld **Selektor** können Sie einen Ausdruck eingeben, der den durch die Lebenslinie dargestellten Teil spezifiziert, wenn das Element, das verbunden werden kann, mehrere Werte hat, d.h. eine Multiplizität größer als eins aufweist.

Gehe zu Lebenslinienelement

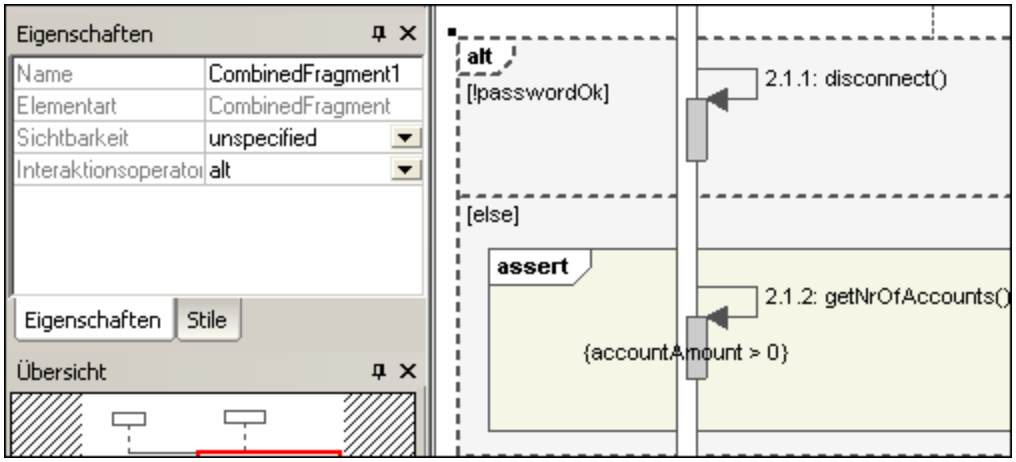
Wenn Sie mit der rechten Maustaste auf eine Lebenslinie klicken, können Sie die Option "Gehe zu XXX" auswählen, wobei XXX für den Typ der speziellen Lebenslinie steht, auf die Sie geklickt haben. Das Element wird daraufhin im Fenster "Model-Struktur" angezeigt.

9.1.7.1.2 Combined Fragment

Combined Fragments  sind Untereinheiten oder Abschnitte einer Interaktion. Der Interaktionsoperator, der im Fünfeck in der linken oberen Ecke angezeigt wird, definiert, um welche spezifische Art von Combined Fragment es sich handelt. Die Einschränkung definiert daher das jeweilige Fragment z.B. loop fragment, alternative fragment usw., welches in der Interaktion verwendet wird.

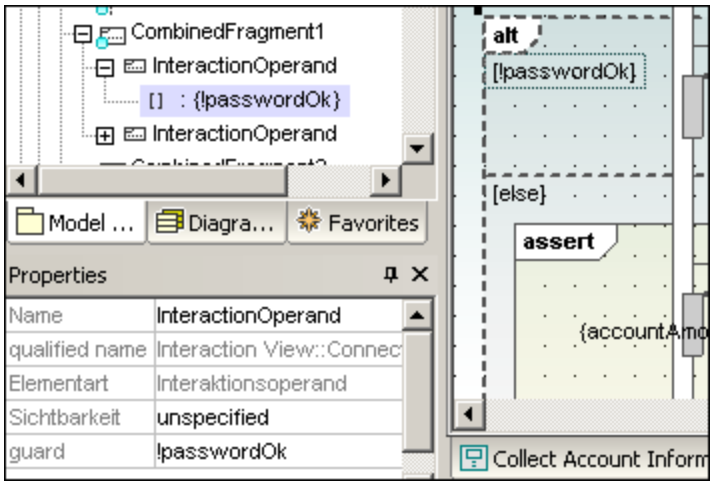


Über die Symbole für Combined Fragments in der Symbolleiste können Sie bestimmte Combined Fragments einfügen: seq, alt oder loop. Auch durch Klicken auf die **Interaktionsoperator-**Auswahlliste können Sie das spezifische Interaktionsfragment definieren.



© 2016-2022 Altova GmbH

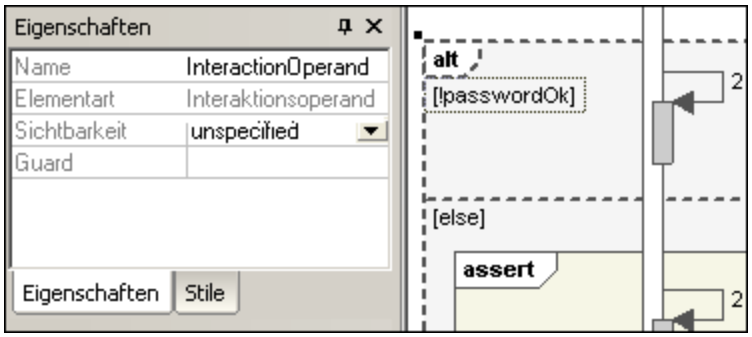
Interaktionsoperatoren

Lose Ordnung	**seq** 	 Das Combined Fragment stellt eine lose Ordnung zwischen dem Verhalten der Operanden dar.
Alternativen	**alt** 	Nur einer der definierten Operanden wird ausgewählt. Der Operand muss eine "Guard-Expression" haben, deren Auswertung "true" ergibt.  Wenn in einem der Operanden der Guard-Ausdruck "else" verwendet wird, wird dieser Operand ausgeführt, wenn alle anderen Guards den Wert "false" zurückgeben. Die Guard-Expression kann unmittelbar beim Einfügen eingegeben werden und wird innerhalb der beiden eckigen Klammern angezeigt.  Bei der **Interaktionsbedingung** handelt es sich eigentlich um den Guard-Ausdruck zwischen den eckigen Klammern.
Option	**opt**	Option steht für eine Auswahl, wobei entweder nur der Operand ausgeführt wird oder nichts passiert.
Abbruchfragment	**break**	Der Break-Operator wird ausgewählt, wenn Guard "true" ist. Der Rest des umschließenden Fragments wird ignoriert.
Parallel	**par**	Zeigt an, dass das Combined Fragment eine parallele Zusammenführung von Operanden darstellt.

Strenge Ordnung	**strict**	Das Combined Fragment stellt eine strenge Ordnung zwischen dem Verhalten der Operanden dar.
Schleife	**loop** 	Der Schleifen-Operand wird so oft wiederholt, wie in der Guard-Expression definiert.  Nachdem Sie diesen Operand ausgewählt haben, können Sie den Ausdruck (im Schleifen-Fünfeck) direkt bearbeiten, indem Sie darauf doppelklicken.
Kritischer Bereich	**critical**	Das Combined Fragment stellt einen kritischen Bereich dar. Die Sequenz(en) darf/dürfen nicht durch andere Prozesse unterbrochen werden.
Negativ	**neg**	Definiert, dass das Fragment ungültig ist und dass alle anderen als gültig gelten.
Sicherstellung	**assert**	Kennzeichnet das gültige Combined Fragment und seine Sequenzen. Wird oft in Kombination mit consider- oder ignore-Operanden verwendet.
Ignore	**ignore**	Definiert, welche Nachricht in der Interaktion ignoriert werden soll. Wird oft im Zusammenhang mit "Sicherstellung" oder "Consider"-Operanden verwendet.
Consider	**consider**	Definiert, welche Nachricht in der Interaktion berücksichtigt werden soll.

Hinzufügen von Interaktionsoperanden zu einem Combined Fragment

1. Rechtsklicken Sie auf das Combined Fragment und wählen Sie **Neu | Interaktionsoperand**. Der Textcursor wird automatisch so gesetzt, dass Sie die Guard Condition eingeben können.
2. Geben Sie die Guard Condition für den Interaktionsoperanden ein, z.B. **!passwordOK** und bestätigen Sie mit der Eingabetaste. Mit Strg + Eingabetaste können Sie einen **mehrzeiligen** Interaktionsoperanden erstellen.



3. Fügen Sie nun auf dieselbe Art den zweiten Interaktionsoperanden mit der Guard Condition "else" ein. Die einzelnen Operanden im Fragment werden durch strichlierte Linien getrennt.

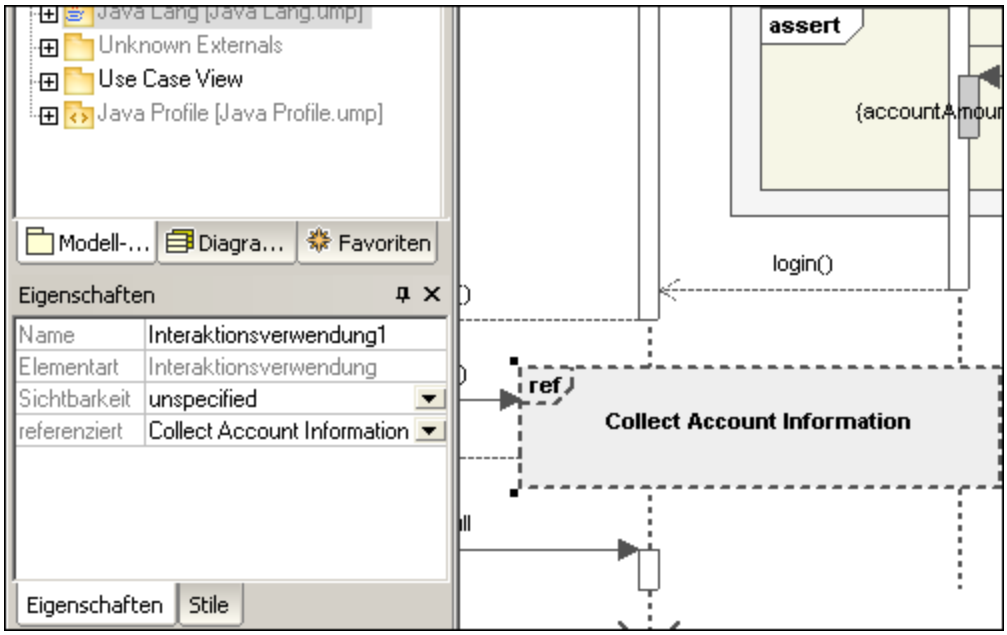
© 2016-2022 Altova GmbH

Löschen von Interaktionsoperanden

1. Doppelklicken Sie im Diagramm (nicht auf dem Register "Eigenschaften") auf die Guard-Expression im Combined Fragment.
2. Löschen Sie die Guard-Expression zur Gänze und drücken Sie zur Bestätigung die Eingabetaste. Die Guard Expression/der Interaktionsoperand wird entfernt und die Größe des Combined Fragment wird automatisch angepasst.

9.1.7.1.3 Interaction Use

Das Element "Interaction Use"  ist eine Referenz auf ein Interaktionselement. Mit Hilfe dieses Elements können Sie Abschnitte einer Interaktion mit verschiedenen anderen Interaktionen gemeinsam verwenden.



Wenn Sie auf die Auswahlliste "referenziert" klicken, können Sie die Interaktion auswählen, auf die Sie referenzieren möchten. Der Name der ausgewählten Interaktion wird im Element angezeigt.

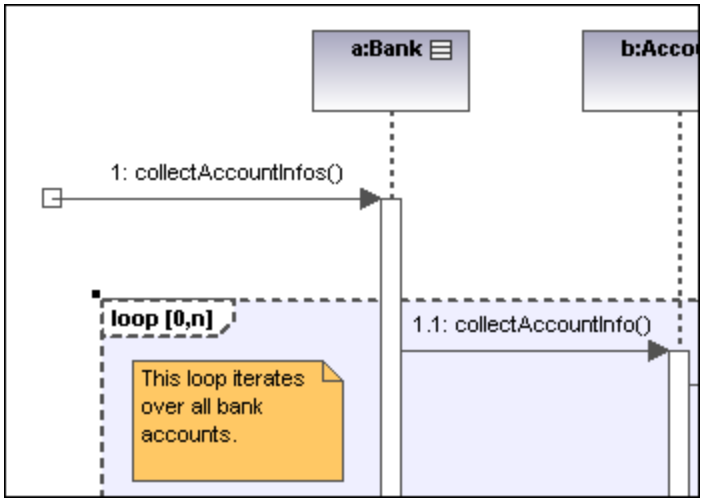
Anmerkung:
Sie können auch ein vorhandenes Interaction Use-Element aus der Modell-Struktur ins Diagrammfenster ziehen.

9.1.7.1.4 Gate

Ein **Gate**  ist ein Verbindungspunkt, über den Nachrichten in und aus Interaktionsfragmente/n übertragen werden können. Gates werden mittels Nachrichten miteinander verbunden.

1. Fügen Sie das Gate-Element in das Diagramm ein.

2. Erstellen Sie eine neue Nachricht und ziehen Sie den Cursor vom Gate zur Lebenslinie oder von einer Lebenslinie auf ein Gate.
   Dadurch werden die beiden Elemente miteinander verbunden. Das Quadrat, das das Gate darstellt, wird nun kleiner angezeigt.



9.1.7.1.5 Zustandsinvariante

Eine **Zustandsinvariante** {s} ist eine Bedingung oder eine Einschränkung, die auf eine Lebenslinie angewendet wird. Die Bedingung muss zutreffen, damit die Lebenslinie vorhanden sein kann.

So definieren Sie eine Zustandsinvariante:

1. Klicken Sie auf das Symbol "Zustandsinvariante" und anschließend auf die Lebenslinie oder auf eine Objektaktivierung, um diese einzufügen.
2. Geben Sie die gewünschte Bedingung/Einschränkung ein, z.B. accountAmount > 0 und drücken Sie zur Bestätigung die Eingabetaste.



9.1.7.1.6 Nachrichten

Nachrichten werden von sendenden an empfangende Lebenslinien gesendet und in Form beschrifteter Pfeile angezeigt. Nachrichten können eine Sequenznummer und verschiedene andere optionale Attribute aufweisen: argument list usw. Nachrichten werden von oben nach unten angezeigt, d.h. die vertikale Linie stellt den zeitlichen Ablauf im Sequenzdiagramm dar.

© 2016-2022 Altova GmbH

- Ein **Aufruf** ist eine synchrone oder asynchrone Kommunikation, die eine Operation aufruft, über die die Kontrolle zum sendenden Objekt zurückkehren kann. Ein Aufrufpfeil zeigt zum **Anfang** der Aktivierung, die der Aufruf initiiert.
- Rekursionen oder Aufrufe einer anderen Operation desselben Objekts werden durch Übereinanderstapeln von Aktivierungskästchen angezeigt (Ausführungsspezifikationen).

So fügen Sie eine Nachricht ein:

1. Klicken Sie in der Sequenzdiagramm-Symbolleiste auf das jeweilige Nachrichtensymbol.
2. Klicken Sie auf die Lebenslinie oder das Aktivierungskästchen des Sender-Objekts.
3. Ziehen Sie die Nachrichtenlinie auf die Lebenslinie oder das Aktivierungskästchen des Empfängerobjekts. Objektlebenslinien erscheinen markiert, wenn Sie die Nachricht dorthin ziehen können.

- Die Richtung, in die Sie die Pfeile ziehen, bestimmt die Richtung der Nachricht. Antwortnachrichten können in jede der beiden Richtungen weisen.

- Ein oder mehrere Aktivierungskästchen wird bzw. werden automatisch auf den Sender-/Empfänger-Objekten erstellt oder in der Größe angepasst. Sie können deren Größe auch manuell durch Ziehen der entsprechenden Ziehpunkte anpassen.

- Abhängig davon, welche Einstellungen Sie für die Nummerierung der Nachrichten aktiviert haben, wird die Nummerierungsreihenfolge aktualisiert.

- Wenn Sie auf ein Nachrichtensymbol klicken und dabei die Strg-Taste gedrückt halten, können Sie mehrere Nachrichten einfügen. Ziehen Sie dazu einfach mehrere Nachrichten in das Diagrammfenster.

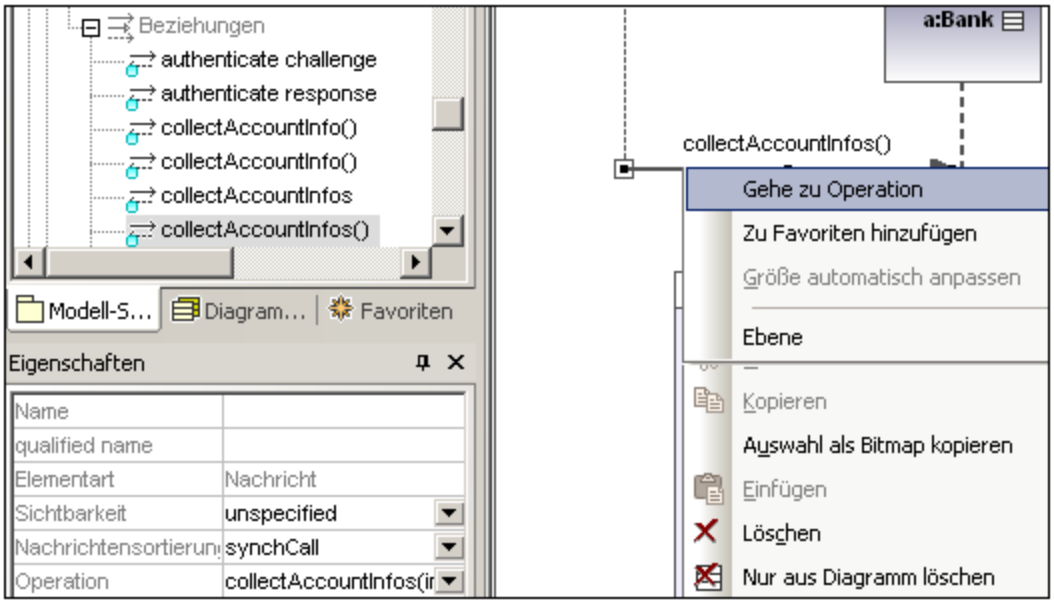
So löschen Sie eine Nachricht:

1. Klicken Sie auf die jeweilige Nachricht, um sie auszuwählen.
2. Drücken Sie die Entf-Taste, um die Nachricht aus dem Modell zu löschen oder rechtsklicken Sie auf die Nachricht und wählen Sie den Befehl "Aus Diagramm löschen".
   Die Nummerierung der Nachrichten und die Aktivierungskästchen der verbleibenden Objekte werden aktualisiert.

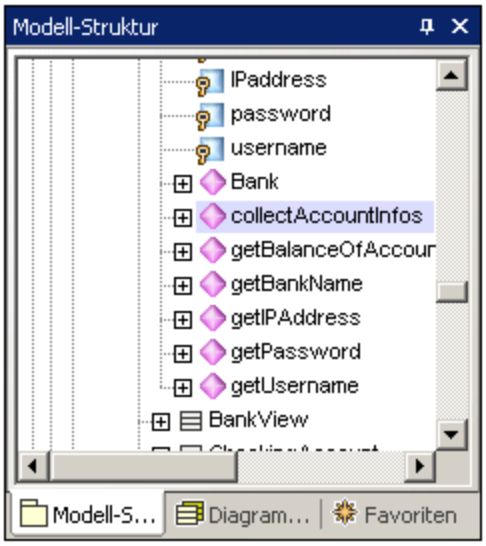
"Gehe zu-Operation" für Call-Nachrichten

Die durch Call-Nachrichten referenzierten Operationen werden in Sequenz- und Kommunikationsdiagrammen verwendet.

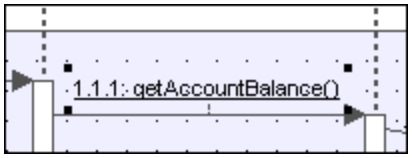
1. Rechtsklicken Sie auf eine Call-Nachricht und wählen Sie den Option "Gehe zu Operation".



Die Anzeige ändert sich und die die Verbindungsoperation wird auf dem Register "Modell-Struktur" angezeigt.



Anmerkung:
Statische Operationsnamen werden in Sequenzdiagrammen unterstrichen angezeigt.



© 2016-2022 Altova GmbH

So positionieren Sie zusammengehörige Nachrichten:

1. Klicken Sie auf die entsprechende Nachricht und ziehen Sie sie in vertikaler Richtung, um sie neu zu positionieren. Standardmäßig werden beim Neupositionieren einer Nachricht alle mit der aktiven in Zusammenhang stehenden Nachrichten mitverschoben.

   Mit Strg + Klick können Sie mehrere Nachrichten auswählen.

So positionieren Sie Nachrichten einzeln:

1. Klicken Sie auf das Symbol "**Verschieben zusammengehöriger Nachrichten ein/aus**" , um diese Funktion zu deaktivieren.
2. Klicken Sie auf die gewünschte Nachricht und ziehen Sie sie, um sie zu verschieben.
   Nur die ausgewählte Nachricht wird beim Ziehen verschoben. Sie können die Nachricht an einer beliebigen Stelle auf der vertikalen Achse zwischen den Objektlebenslinien positionieren.

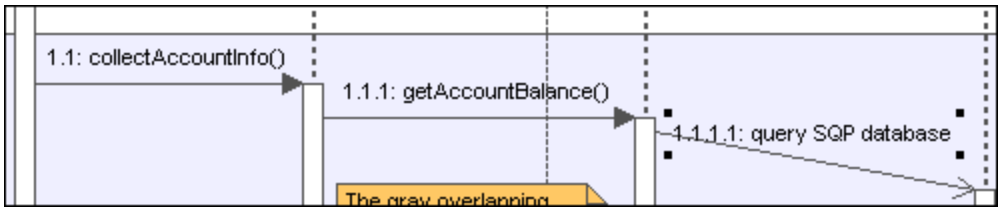
So erstellen Sie automatisch Antwortnachrichten:

1. Klicken Sie auf die Schaltfläche "**Automatische Erstellung von Antworten auf Nachrichten (Call) ein/aus**" .
2. Erstellen Sie eine neue Nachricht zwischen den beiden Lebenslinien. Daraufhin wird automatisch eine Antwortnachricht eingefügt.

Nachrichtennummerierung

UModel unterstützt verschiedene Methoden der Nachrichtennummerierung: hierarchisch, einfach und keine.

- **Keine**  entfernt alle Nachrichtennummern.
- **Einfach**  nummeriert alle Nachrichten von oben nach unten, also in der Reihenfolge, in der sie auf der Zeitachse vorkommen, durch.
- **Hierarchisch**  verwendet eine Dezimaldarstellung, sodass die hierarchische Struktur der Nachrichten im Diagramm deutlich wird. Die Hierarchie wird in Form einer durch Punkte getrennten Nummerierung, gefolgt von einem Doppelpunkt und dem Namen der Nachricht dargestellt.



Es gibt zwei Methoden, um das Nummerierungssystem auszuwählen:

- Klicken Sie auf das jeweilige Symbol in der Symbolleiste.
- Wählen Sie das Nummerierungssystem über das Register **Stile** aus.

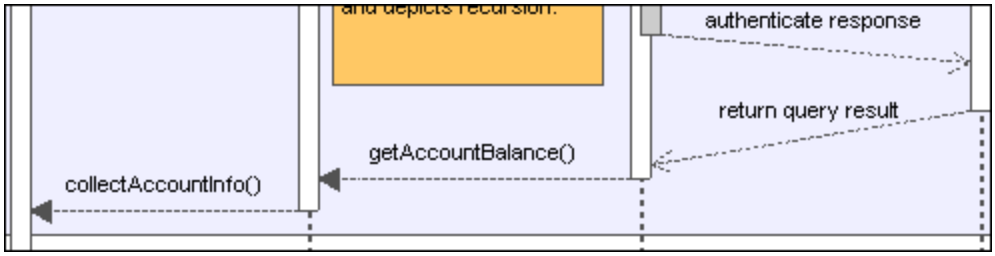
Um das Nummerierungssystem über das Register "Stile" auszuwählen:

1. Klicken Sie auf das Register **Stile** und scrollen Sie hinunter zum Feld **Nachrichtennummern anzeigen**.
2. Klicken Sie auf das Auswahllistenfeld und wählen Sie die gewünschte Nummerierungsoption aus. Die gewählte Nummerierungsoption wird sofort im Sequenzdiagramm angezeigt.

Anmerkung:
Bei Vorhandensein nicht eindeutiger Spuren werden mit dem gewählten Nummerierungssystem nicht immer alle Nachrichten richtig nummeriert. Fügen Sie in diesem Fall Antwortnachrichten hinzu, um Unklarheiten zu beseitigen.

Antwortnachrichten

Zum Erstellen von Antwortnachrichten stehen Nachrichtenantwortsymbole zur Verfügung. Diese werden als strichlierte Pfeile dargestellt.

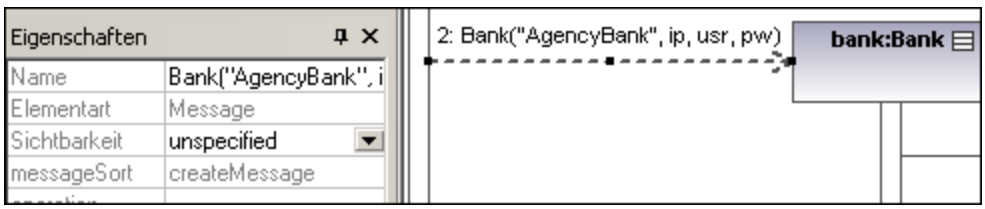


Im Allgemeinen geht es außerdem aus dem unteren Bereich der Aktivierungskästchen - falls diese aktiviert sind - hervor, ob Antwortnachrichten erwartet werden. Wenn Aktivierungskästchen deaktiviert wurden (Register "Stile" | Show Execution Specifics=false), sollten aus Gründen der Übersichtlichkeit Pfeile verwendet werden.

Wenn Sie die Schaltfläche "Automatische Erstellung von Antworten auf Nachrichten (Call) ein/aus" aktivieren, wird automatisch eine syntaktisch korrekte Antwortnachricht erstellt, wenn Sie eine Call-Nachricht zwischen Lebenslinien/Aktivierungskästen erstellen.

Erstellen von Objekten mit Hilfe von Nachrichten

1. Nachrichten können neue Objekte erstellen. Dies erfolgt über das Symbol "Nachricht (Erstellung)" .
2. Ziehen Sie den Nachrichtenpfeil zur Lebenslinie eines bestehenden Objekts, um das Objekt zu erstellen. Diese Nachrichtenart endet in der Mitte eines Objektrechtecks und positioniert das Objektkästchen oft in vertikaler Richtung neu.

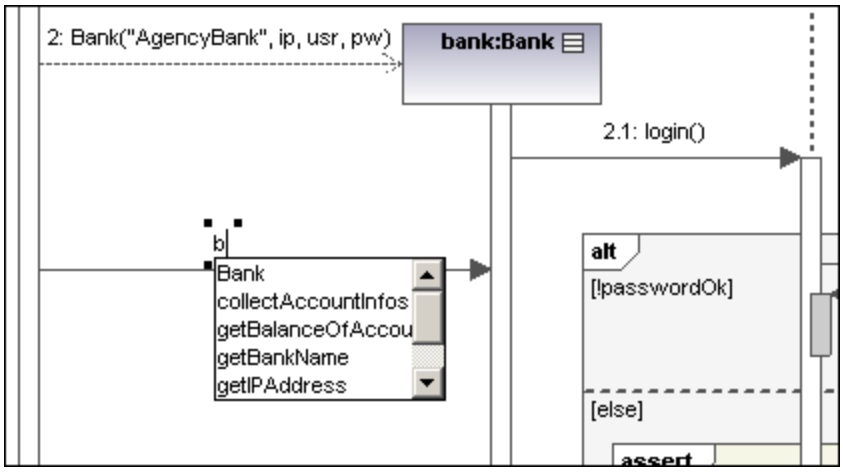


 © 2016-2022 Altova GmbH

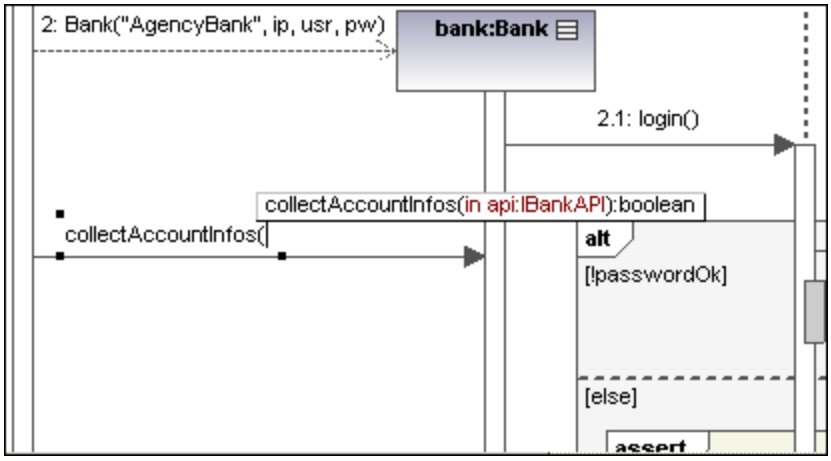
Senden von Nachrichten an bestimmte Klassenmethoden/-operationen in Sequenzdiagrammen

Nachdem Sie eine Klasse aus der Modellstruktur in ein Sequenzdiagramm eingefügt haben, können Sie anschließend mit Hilfe der UModel Syntaxhilfe und der Autokomplettierungsfunktionen eine Nachricht von einer Lebenslinie zu einer bestimmten Methode der Empfängerklasse (Lebenslinie) erstellen.

1. Erstellen Sie eine Nachricht zwischen zwei Lebenslinien, wobei es sich beim Empfängerobjekt um eine Klassenlebenslinie (Bank) handeln muss.Nachdem Sie den Nachrichtenpfeil gezogen haben, erscheint der Name der Nachricht automatisch markiert.
2. Geben Sie über die Tastatur ein Zeichen ein, z.B. "b". Es erscheint ein Popup-Fenster mit einer Liste der verfügbaren Klassenmethoden.



3. Wählen Sie eine Operation aus der Liste aus, z.B. collectAccountinfos und drücken Sie die Eingabetaste.
4. Drücken Sie die Leertaste und anschließend die Eingabetaste, um das automatisch vorgegebene Klammerzeichen auszuwählen.
   Es erscheint ein Syntaxhilfefenster, über das Sie den Parameter korrekt eingeben können.



Erstellen von Operationen in referenzierten Klassen

Wenn Sie die Schaltfläche "Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten"  aktivieren, wird die entsprechende Operation in der referenzierten Klasse automatisch erstellt, wenn Sie eine Transition erstellen und einen Namen myOperation() eingeben.

Anmerkung: Operationen können nur dann automatisch erstellt werden, wenn sich der Zustandsautomat innerhalb einer Klasse oder Schnittstelle befindet.

Nachrichtensymbole

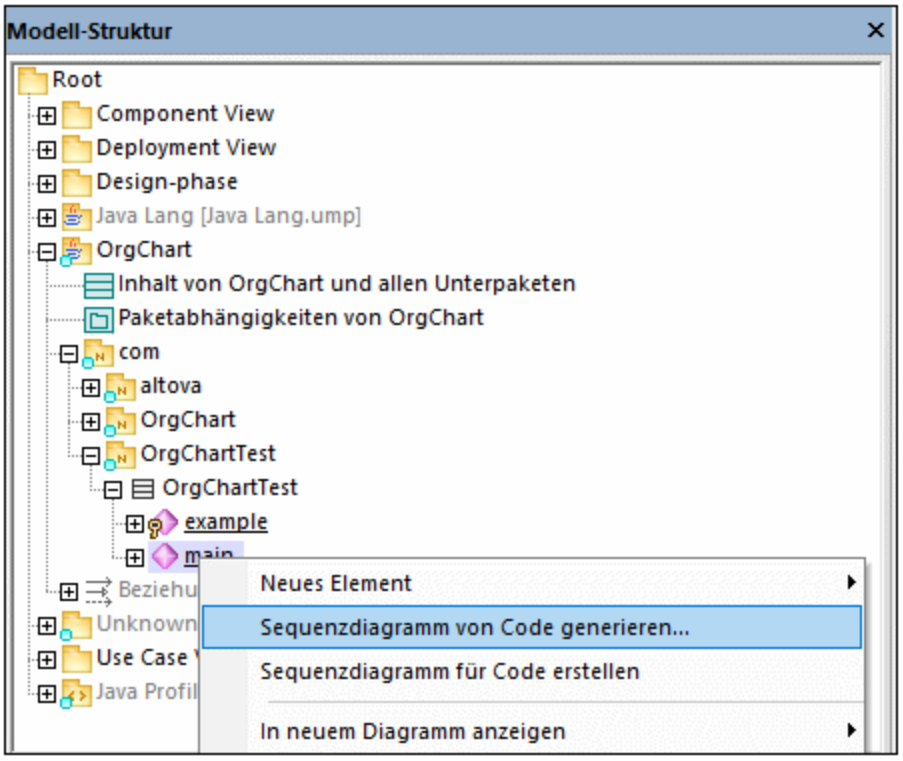
 Nachricht (Aufruf)
 Nachricht (Antwort)
 Nachricht (Erstellung)
 Nachricht (Löschung)
 Asynchrone Nachricht (Aufruf)
 Asynchrone Nachricht (Antwort)
 Asynchrone Nachricht (Löschung)
 Verschieben zusammengehöriger Nachrichten ein/aus
 Automatische Erstellung von Antworten auf Nachrichten (Call) ein/aus
 Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten

9.1.7.2 Generieren von Sequenzdiagrammen anhand von Quellcode

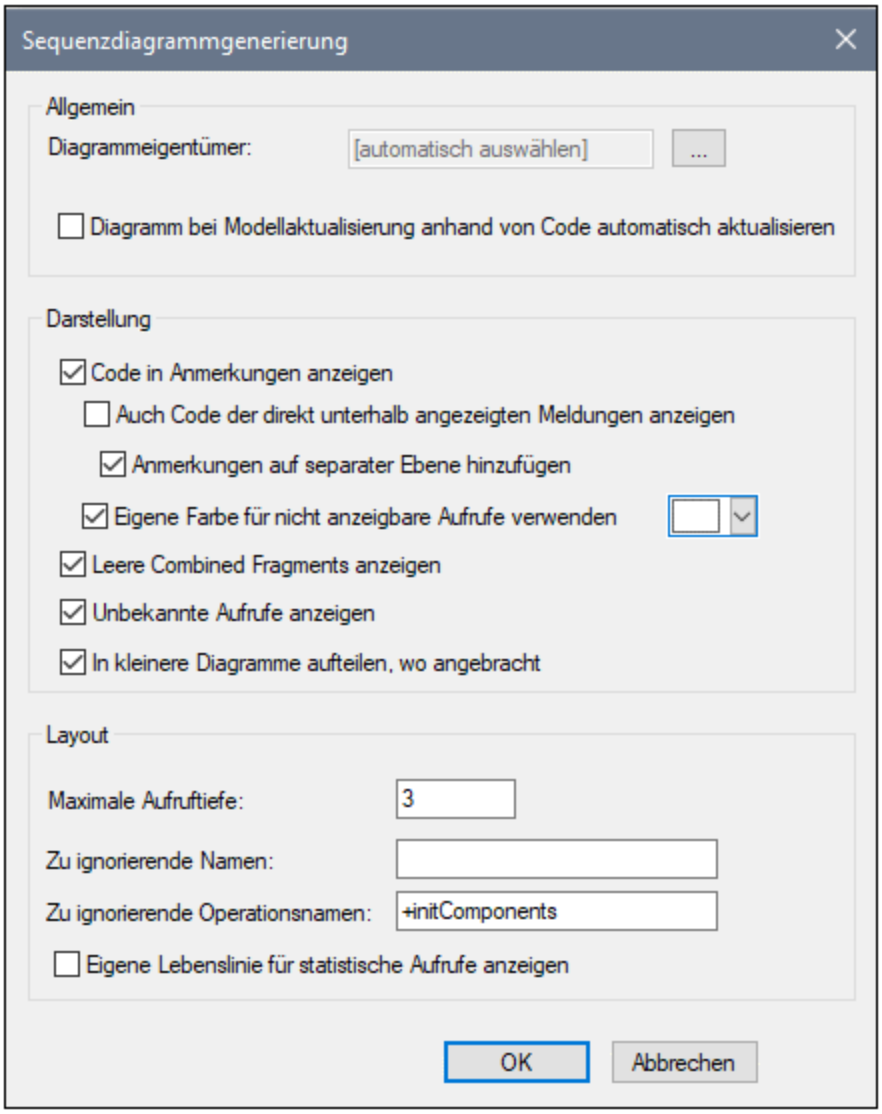
In diesem Beispiel wird gezeigt, wie Sie ein Sequenzdiagramm anhand von einer Methode generieren. Das Projekt, das diese Methode enthält, wird mittels Reverse Engineering anhand von Java-Quellcode erstellt. Sie finden den Java-Quellcode unter dem folgenden Pfad: **C:\Benutzer\<Benutzer>\Dokumente\Altova\UModel2023\UModelExamples\OrgChart.zip**. Entpacken Sie das **OrgChart.zip**-Archiv zuerst in denselben Ordner (Klicken Sie z.B. mit der rechten Maustaste in Windows Explorer auf das Archiv und wählen Sie **Extract All**).

1. Klicken Sie im Menü **Projekt** auf **Quellverzeichnis importieren** und wählen Sie das zuvor entpackte Verzeichnis aus.
2. Befolgen Sie die Anweisungen des Assistenten, um den Quellcode als Java-Projekt zu importieren. Nähere Informationen zu diesem Schritt finden Sie unter Reverse Engineering (Code zu Modell) 75.
3. Nachdem Sie den Code importiert haben, klicken Sie mit der rechten Maustaste in der Modellstruktur auf die Methode `main` der Klasse `OrgChartText` und wählen Sie im Kontextmenü den Befehl **Sequenzdiagramm von Code generieren**.

© 2016-2022 Altova GmbH

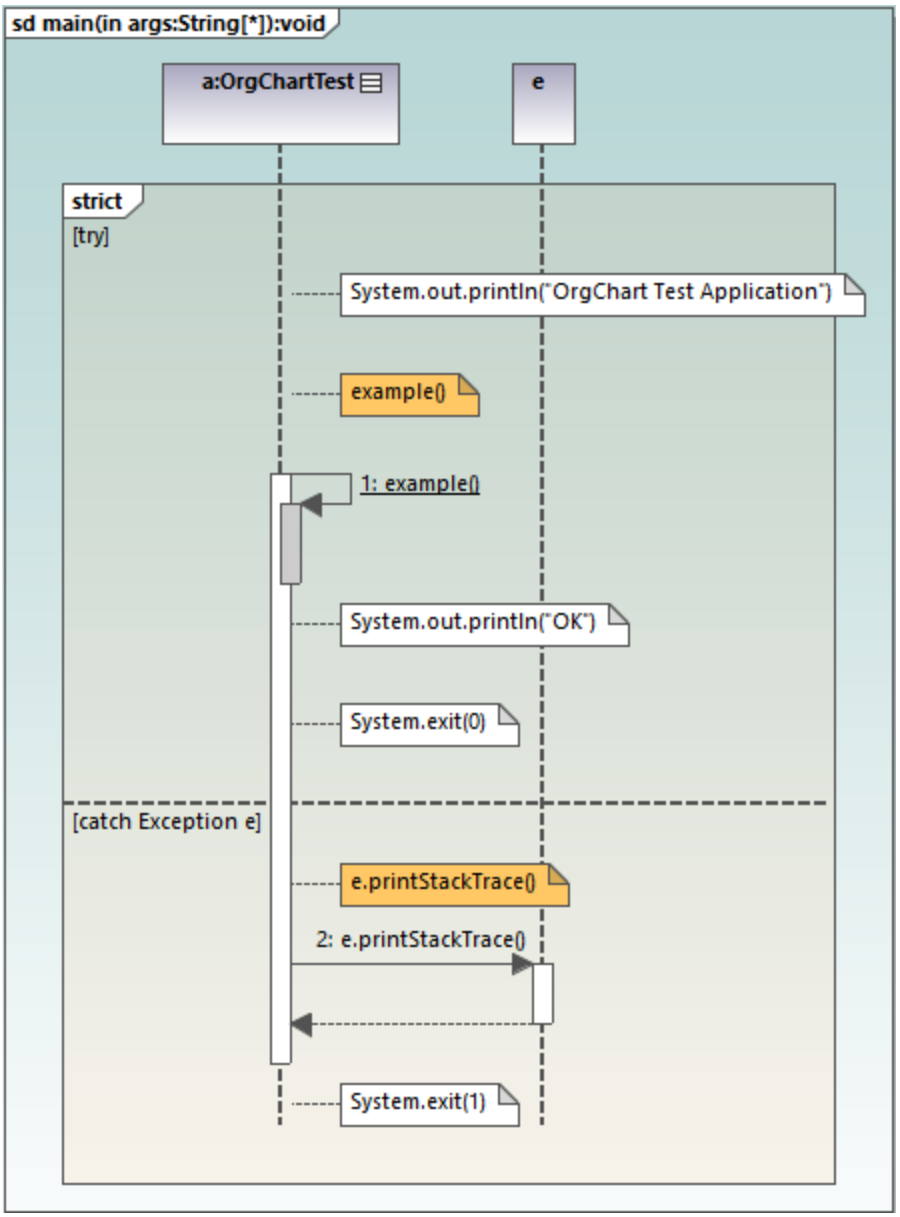


Daraufhin wird das Dialogfeld "Sequenzdiagrammgenerierung" geöffnet, in dem Sie die Einstellungen für die Generierung definieren können.



4. Wählen Sie die Darstellungs- und Layout-Optionen aus und klicken Sie anschließend auf **OK**, um das Diagramm zu generieren. Mit den oben gewählten Einstellungen wird das unten gezeigte Sequenzdiagramm erzeugt.

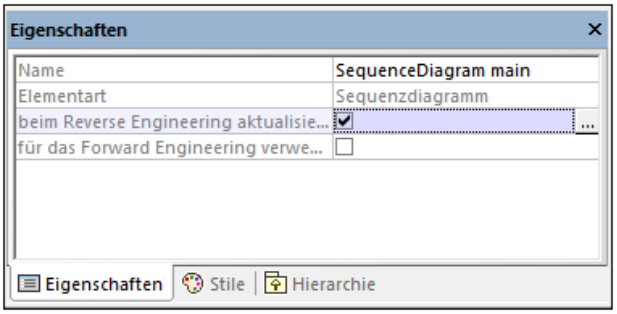
© 2016-2022 Altova GmbH



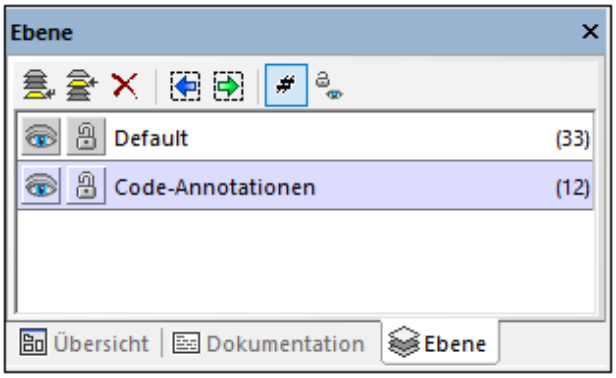
Optionen für die Generierung von Sequenzdiagrammen

In der unten stehenden Tabelle werden die Optionen für die Generierung von Sequenzdiagrammen aufgelistet.

Option	Aufgabe
Diagrammeigentümer	Diese Option kann definiert werden, wenn das Diagramm zum ersten Mal generiert wird. Bei vorhandenen Diagrammen ist diese Information schreibgeschützt.

Option	Aufgabe
	Klicken Sie auf die **Auslassungszeichen**, um das Owner-Paket des Diagramms auszuwählen. Andernfalls platziert die Option **[automatisch auswählen]** das Diagramm in das Standardpaket.
Diagramm bei Modellaktualisierung anhand von Code automatisch aktualisieren.	Beim Reverse Engineering (von Code zu Modell) werden Sequenzdiagramme im Modell automatisch neu generiert, vorausgesetzt, Sie haben die Option **Diagramm bei Modellaktualisierung anhand von Code automatisch aktualisieren** aktiviert, als Sie das Diagramm zum ersten Mal generiert haben. Bei bestehenden Diagrammen können Sie diese Option folgendermaßen ändern: 1. Wählen Sie das Sequenzdiagramm in der Modellstruktur oder der Diagrammstruktur aus. 2. Aktivieren Sie im Fenster "Eigenschaften" die Option **beim Rerverse Engineering aktualisieren**.  Wenn Sie das Kontrollkästchen **für das Forward Engineering verwenden** aktivieren, wird bei der Durchführung eines Forward Engineering (von Modell zu Code) bei der Synchronisierung von Modell zu Code Code anhand des Sequenzdiagramms generiert, siehe auch Generieren von Code anhand von Sequenzdiagrammen 434. Wenn die beiden "Engineering"-Kontrollkästchen fehlen, ist das Diagramm wahrscheinlich nur ein Fragment eines größeren Diagramms oder wurde eventuell anhand einer nicht mit Reverse Engineering erstellten Operation generiert.
Code in Anmerkungen anzeigen	Aktivieren Sie dieses Kontrollkästchen, um das Diagramm mit Anmerkungen (Beschriftungen), die Programmcode enthalten, zu generieren.

© 2016-2022 Altova GmbH

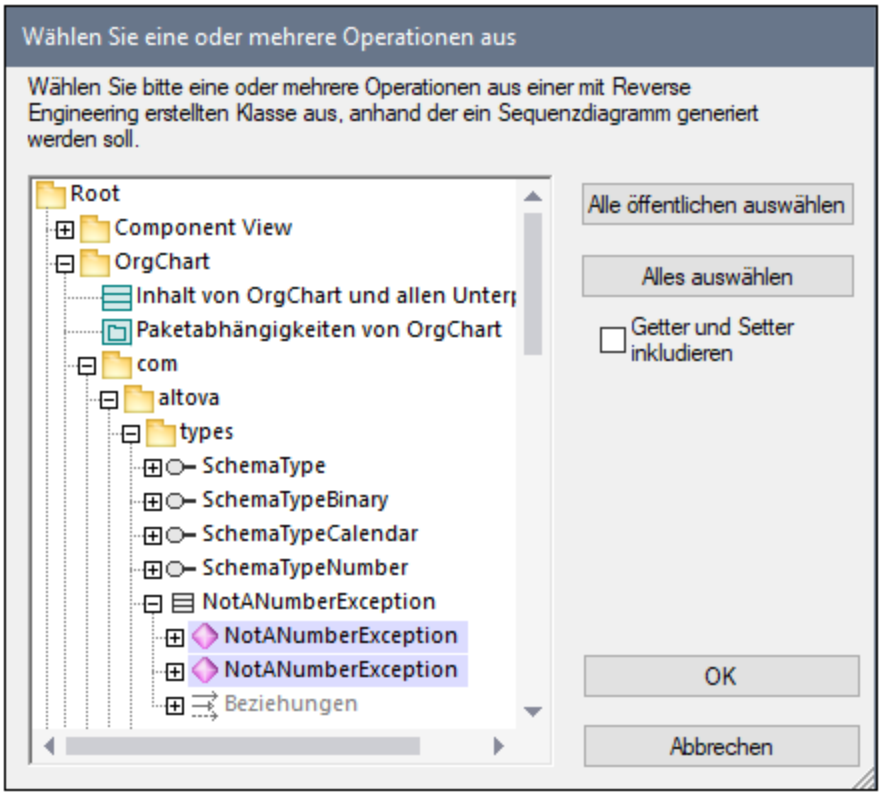
Option	Aufgabe
Auch Code der unterhalb angezeigten Meldungen anzeigen	Selbst in Fällen, in denen ein Codefragment im Diagramm als UML-Meldung angezeigt werden kann, wird der Code dieser Meldung mit dieser Option dennoch als Anmerkung angezeigt.
Anmerkungen auf separater Ebene hinzufügen	Weist einer Code-Annotationsebene Code-Anmerkungen zu. 
Eigene Farbe für nicht anzeigbare Aufrufe verwenden	Weist nicht anzeigbaren Aufrufen eine Farbe Ihrer Wahl zu.
Leere Combined Fragments anzeigen	Die Combined Fragment[416]-Blöcke werden auch dann im Diagramm beibehalten, wenn sie keinen Inhalt haben.
Unbekannte Aufrufe anzeigen	Wenn diese Option aktiviert ist, werden auch Meldungen für Operationen oder Konstruktoren, die nicht aufgelöst werden konnten (d.h. die im Modell nicht gefunden wurden) angezeigt.
In kleinere Diagramme aufteilen, wo angebracht	Damit werden Sequenzdiagramme in kleinere Subdiagramme aufgeteilt, zwischen denen zur einfacheren Navigation automatisch Hyperlinks angelegt werden.
Maximale Aufruftiefe	Definiert die Aufruftiefe, die im Diagramm verwendet werden soll. Wenn z.B. `method1()` `method2()` aufruft, die wiederum `method3()` aufruft, und als Aufruftiefe **2** definiert ist, wird nur `method2` angezeigt. `method3` wird nicht mehr angezeigt.
Zu ignorierende Typnamen	Damit können Sie eine kommagetrennte Liste von Typen definieren, die im Sequenzdiagramm nicht aufscheinen sollen, wenn es generiert wird.
Zu ignorierende Operationsnamen	Damit können Sie eine kommagetrennte Liste von Operationen definieren, die im generierten Sequenzdiagramm nicht aufscheinen sollen. Wenn Sie die Operationsnamen zur Liste hinzufügen, wird

Option	Aufgabe
	die komplette Operation ignoriert. Wenn Sie der Operation in der Liste ein + (Pluszeichen) voranstellen, z.B. **+InitComponent**, werden die Operationsaufrufe im Diagramm angezeigt, jedoch ohne Inhalt.
Eigene Lebenslinie für statische Aufrufe anzeigen	Wenn statische Methodenaufrufe vorhanden sind und es im Diagramm bereits eine Instanz dieses Objekts gibt, werden die Meldungen normalerweise mit dieser vorhanden Lebenslinie verbunden. Wenn diese Option aktiviert ist, wird bei der Diagrammgenerierung eine eigene neue Lebenslinie nur für statische Methodenaufrufe für diesen Classifier verwendet.

9.1.7.2.1 Generieren von Sequenzdiagrammen anhand von Eigenschaften

Sie können auch mehrere Sequenzdiagrammmodelle anhand mehrerer Operationen generieren. Gehen Sie dazu folgendermaßen vor:

1. Wählen Sie die Menüoption **Projekt | Sequenzdiagramme von Code generieren**.



© 2016-2022 Altova GmbH

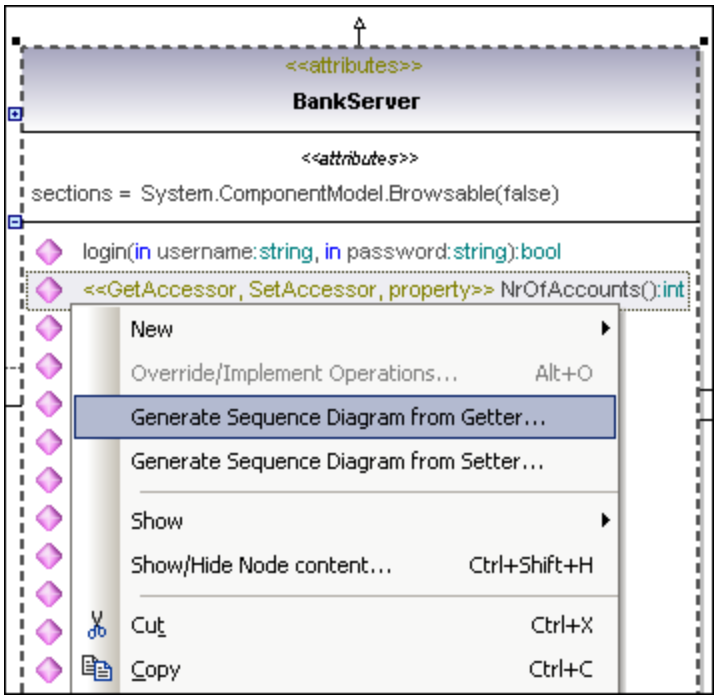
2. Aktivieren Sie die Operationen, für die ein Sequenzdiagramm generiert werden soll, aus und klicken Sie auf **OK**. (Klicken Sie, wo nötig, auf die Schaltflächen **Alle öffentlichen auswählen** und **Alle auswählen**.
3. Aktivieren Sie optional das Kontrollkästchen **Getter und Setter inkludieren**, um Sequenzdiagramme für C#/VB.NET Getter und Setter zu generieren.
4. Klicken Sie auf **OK.** Daraufhin wird ein Dialogfeld geöffnet, in dem Sie die Optionen für die Generierung von Sequenzdiagrammen 429 definieren können.
5. Klicken Sie auf **OK.** Für jede der ausgewählten Operationen wird ein Sequenzdiagramm generiert und automatisch in UModel geöffnet.

Wenn Ihr Projekt groß ist, dauert die Erstellung mehrerer Sequenzdiagramme wahrscheinlich etwas länger. Beachten Sie, dass die ersten 10 Diagramme von UModel automatisch geöffnet werden. Der Rest wird generiert, ohne geöffnet zu werden.

9.1.7.2.2 Generieren von Sequenzdiagrammen anhand von Gettern/Settern

Sie können ein Sequenzdiagramm auch anhand von Getter/Setter-Eigenschaften (in C#, VB .NET) generieren. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste auf eine Operation mit einem `GetAccessor/SetAccessor` Stereotyp.



2. Wählen Sie im Kontextmenü den Befehl **Sequenzdiagramm von Code (Getter / Setter) generieren** aus. Daraufhin wird ein Dialogfeld geöffnet, in dem Sie die Optionen für die Generierung von Sequenzdiagrammen 429 definieren können.
3. Klicken Sie auf **OK**, um das Sequenzdiagramm zu generieren.

9.1.7.3 Generieren von Code anhand eines Sequenzdiagramms

UModel kann Code anhand eines mit mindestens einer Operation verknüpften Sequenzdiagramms generieren.

Die Generierung von Code anhand von Sequenzdiagrammen steht zur Verfügung für:
- VB.NET, C# und Java
- UModel Standalone Edition, Eclipse und Visual Studio Edition
- alle drei UModel Editions

Code anhand von Sequenzdiagrammen kann auf zwei Arten erstellt werden:

- durch einen Reverse Engineering Vorgang. Siehe dazu Generieren von Sequenzdiagrammen anhand von Quellcode 426,
- indem ein mit einer Operation verknüpftes Sequenzdiagramm von Grund auf **neu** erstellt wird. Mit einem Rechtsklick auf die Operation in der Modell-Struktur können Sie über das Kontextmenü den Befehl Sequenzdiagramm für Code erstellen 436 aufrufen.

Wenn Sie ein mit Reverse Engineering erstelltes Sequenzdiagramm als Basis verwenden, vergewissern Sie sich, dass die Option "Code in Anmerkungen anzeigen" beim Reverse Engineering aktiviert ist, damit kein Code verloren geht, wenn Sie wieder mit dem Forward Engineering beginnen. Dies ist darauf zurückzuführen, dass in einem UML-Sequenzdiagramm nicht alle Funktionen von VB.NET, Java und C# angezeigt werden können. Daher werden diese Codeabschnitte in Form von Code-Anmerkungen angezeigt.

So fügen Sie bei der Erstellung eines Sequenzdiagramms einfachen Text als Code hinzu:

1. Hängen Sie eine Anmerkung an eine Lebenslinie in einem Sequenzdiagramm an.
2. Geben Sie den Code ein, der in den endgültigen Quellcode geschrieben werden soll. Aktivieren Sie im Bereich "Eigenschaften" das Kontrollkästchen "Ist Code", damit diese Anmerkung zur Verfügung steht.

Ein Beispiel dazu finden Sie unter Hinzufügen von Code zu Sequenzdiagrammen 437.

Wenn bei jedem Start eines Code Engineering automatisch ein Sequenzdiagramm für das Code Engineering verwendet werden soll:

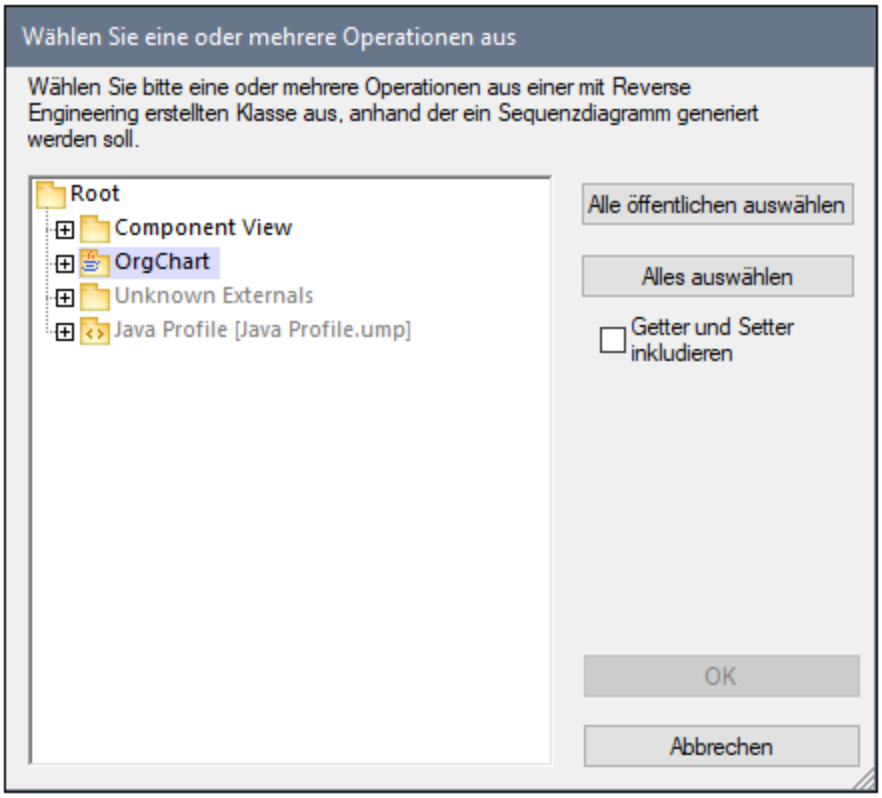
1. Wählen Sie das Diagramm im Fenster "Modell-Struktur" oder "Diagramm-Struktur" aus.
2. Aktivieren Sie im Bereich **Eigenschaften** das Kontrollkästchen **Für das Forward Engineering verwenden**.

Der alte Code geht beim Forward Engineering von Code anhand eines Sequenzdiagramms immer verloren, da er durch den neuen Code überschrieben wird.

So generieren Sie über das Menü "Projekt" Code:

1. Wählen Sie die Menüoption **Projekt | Code von Sequenzdiagrammen generieren**. Daraufhin werden Sie aufgefordert, das/die gewünschten Sequenzdiagramm(e) auszuwählen.

© 2016-2022 Altova GmbH

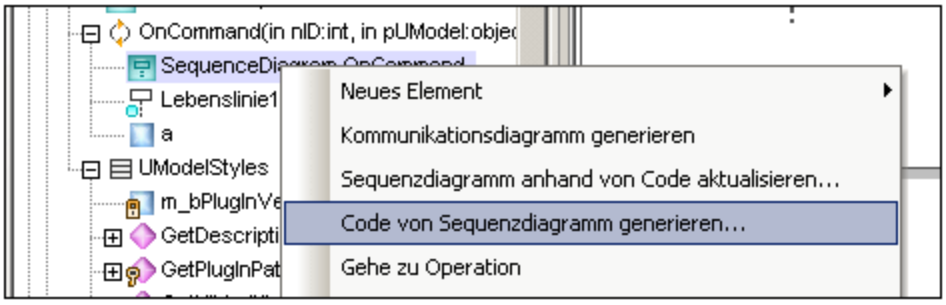


Wenn Sie auf die Schaltfläche "Alles auswählen" klicken, werden alle Sequenzdiagramme im UModel-Projekt ausgewählt.

2. Klicken Sie auf OK, um den Code zu generieren.
   Im Fenster "Meldungen" wird der Status der Codegenerierung angezeigt.

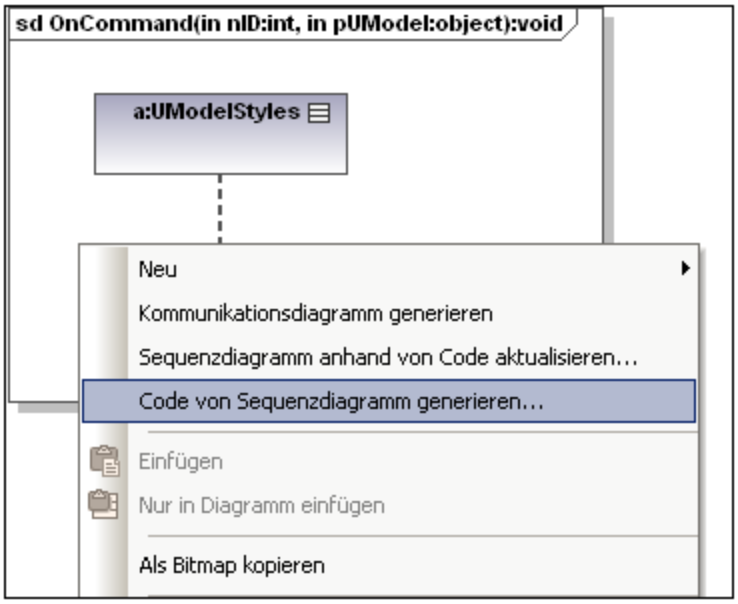
So generieren Sie über die Modell-Struktur Code:

- Klicken Sie mit der rechten Maustaste auf ein Sequenzdiagramm und wählen Sie den Befehl "Code von Sequenzdiagramm generieren".



So generieren Sie ein Sequenzdiagramm, das Code einer Operation enthält:

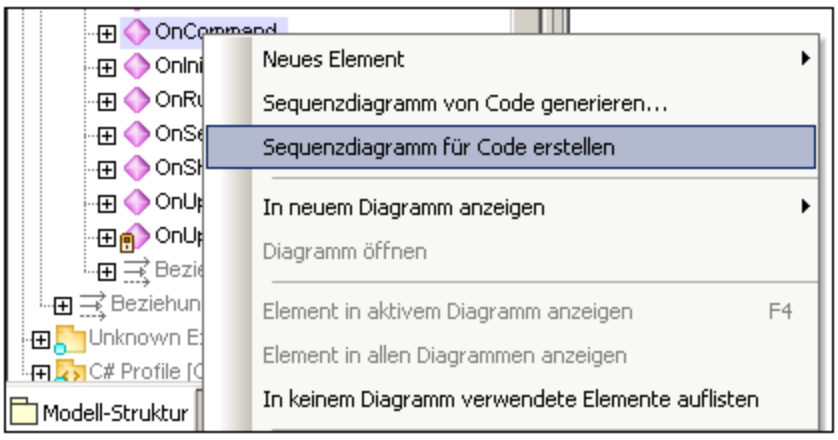
1. Klicken Sie in den leeren Bereich des Sequenzdiagramms, das Code einer Operation enthält.
2. Wählen Sie den Befehl "Code von Sequenzdiagramm generieren".



Mit diesem Befehl wird das Forward Engineering gestartet.

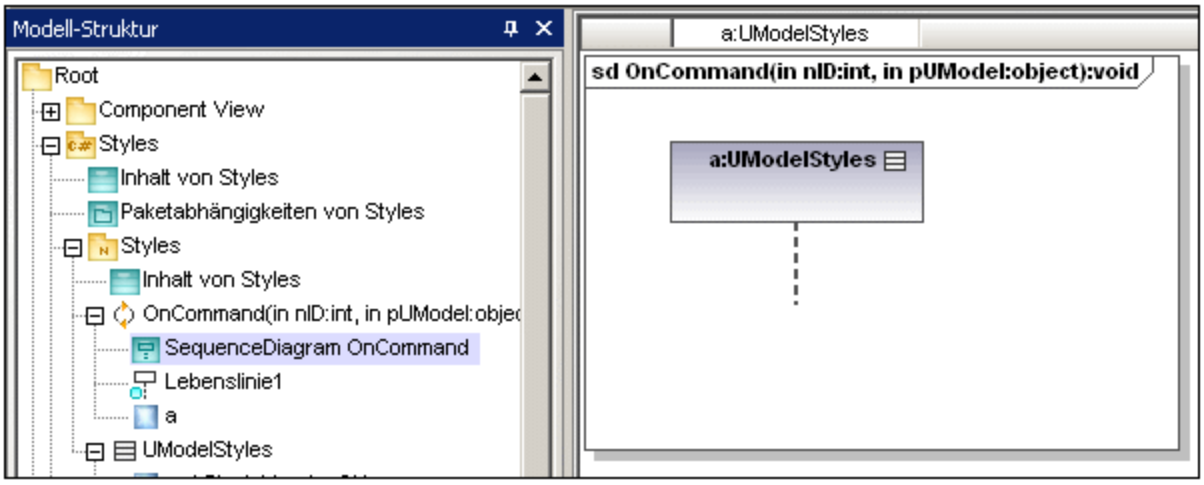
So erstellen Sie ein Sequenzdiagramm für Code (Engineering):

1. Klicken Sie in der Model-Struktur mit der rechten Maustaste auf eine Operation und wählen Sie den Befehl "Sequenzdiagramm für Code erstellen".



Sie werden daraufhin gefragt, ob Sie das neue Diagramm für das Forward Engineering verwenden möchten.

© 2016-2022 Altova GmbH



Das Ergebnis ist ein neues Sequenzdiagramm, das die Lebenslinie dieser Klasse enthält.

9.1.7.3.1 Hinzufügen von Code zu einem Sequenzdiagramm

Programmcode kann anhand eines neuen und eines mit Reverse Engineering erstellten Sequenzdiagramms generiert werden, allerdings nur für ein Sequenzdiagramm, das mit der "Hauptoperation" verknüpft ist.

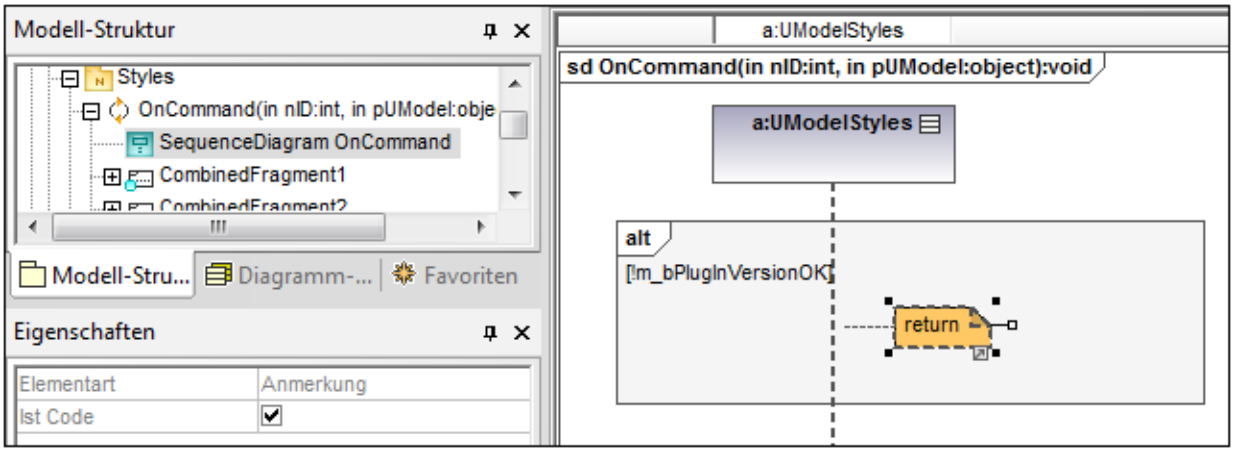
Beim Reverse Engineering werden Standardelemente des Sequenzdiagramms, wie z.B. CombinedFragments Codierungselementen (z.B. "if" Anweisungen, Schleifen usw.) "zugewiesen" bzw. auf diese "gemappt".

Bei Programmanweisungen, für die es keine entsprechenden Sequenzdiagrammelemente gibt, wie z.B. "i = i+1" werden in UModel "Code- Anmerkungen" verwendet, um Code zu Diagrammen hinzuzufügen. Diese Anweisungen müssen anschließend mit der Lebenslinie verknüpft werden.

Beachten Sie, dass UModel diese Codefragmente nicht überprüft oder parst. Sie müssen vorher sicherstellen, dass die Codefragmente korrekt und kompilierbar sind.

So fügen Sie Code zu einem Sequenzdiagramm hinzu:

1. Klicken Sie auf die Schaltfläche "Anmerkung"  und klicken Sie anschließend auf das Modellelement, an dem diese eingefügt werden soll, z.B. CombinedFragment.
2 Geben Sie das Codefragment ein, z.B. return.
3. Klicken Sie auf den Ziehpunkt der eingefügten Anmerkung und ziehen Sie den Cursor auf die Lebenslinie.
4. Aktivieren Sie auf dem Register "Eigenschaften" das Kontrollkästchen "Ist Code", damit dieses Codefragment bei der Codegenerierung inkludiert wird.



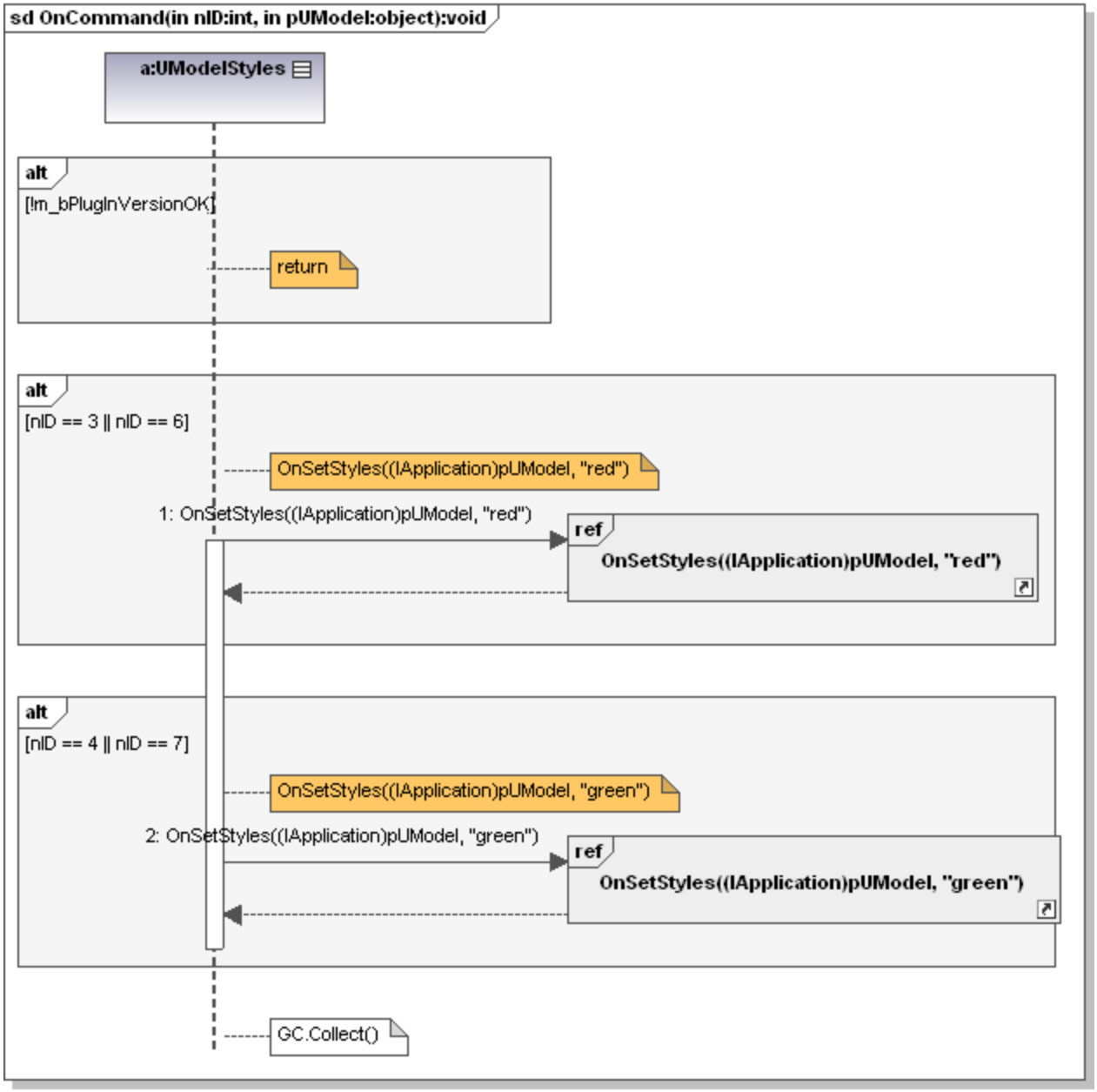
Wenn Sie eine Anmerkung in einem Sequenzdiagramm auswählen, die zur Codegenerierung verwendet werden kann, so steht die Eigenschaft "Ist Code" im Fenster "Eigenschaften" zur Verfügung. Durch Aktivieren/Deaktivieren des Kontrollkästchens können Sie zwischen "normalen" Anmerkungen und Codegenerierungsanmerkungen wechseln.

Normale Anmerkungen:	return
Codegenerierungsanmerkungen	return - werden mit einem schattierten Eck angezeigt

Wenn das Kontrollkästchen "Für das Forward Engineering verwenden" aktiviert ist, wird der Code automatisch bei jedem Forward Engineering aktualisiert. Wenn Änderungen am Sequenzdiagramm vorgenommen wurden, wird der Code der Operation immer überschrieben.

Das unten gezeigte Sequenzdiagramm wurde durch Rechtsklick auf die OnCommand-Operation und Auswahl des Befehls "Sequenzdiagramm von Code generieren" generiert. Der C# Code dieses Beispiels steht im OrdnerC:\Benutzer\<benutzer>\Dokumente\Altova\UModel2023\UModelExamples\IDEPlugIn\Styles\ zur Verfügung. Über die Option "Projekt | Quellprojekt importieren können Sie das Projekt importieren.

© 2016-2022 Altova GmbH



Anhand dieses Sequenzdiagramms wird der unten gezeigte Code generiert.

```
Public void OnCommand(int nID, object pUModel)
{
   //Generated by UModel. This code will be overwritten when you re-run code generation.

   if (!m_bPlugINVersionOK)
   {
   return;
   }

   if (nID == 3 || nID == 6)
```

```
    {
    OnSetStyles((IApplication)pUModel, "red");
    }

    if (nID == 4 || nID == 7)
    {
    OnSetStyles((IApplication)pUModel, "green");
    }
    GC.Collect();

}
```

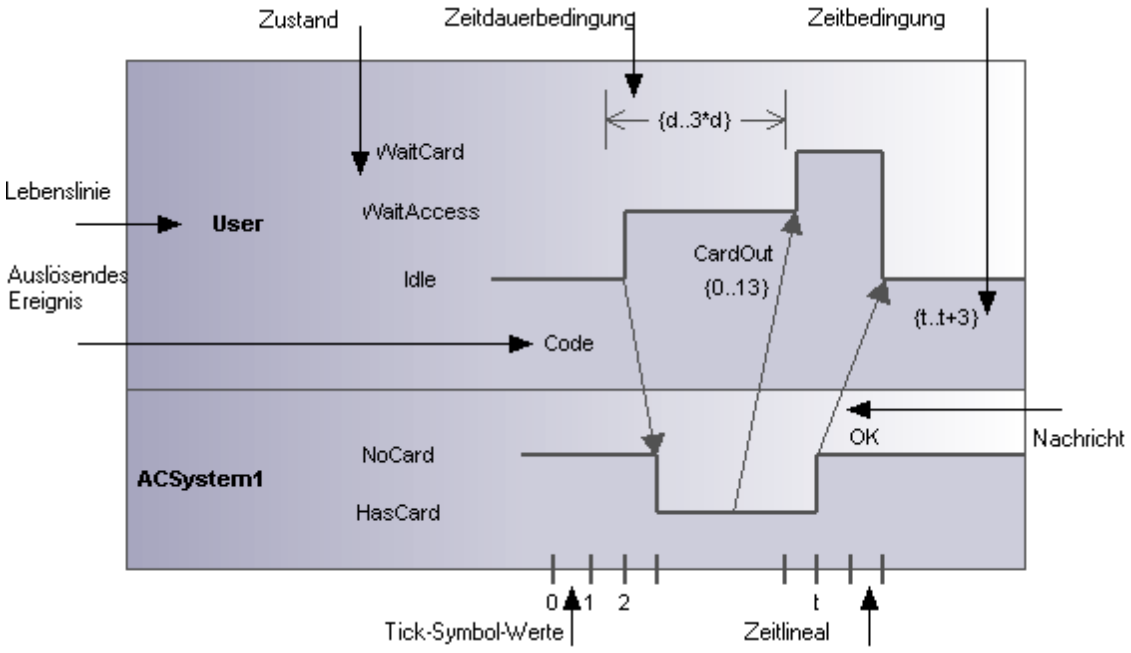
9.1.8 Zeitverlaufsdiagramm

Altova Website: UML-Zeitverlaufsdiagramme

Zeitverlaufsdiagramme dienen zum Darstellen von Änderungen am Status oder Zustand von einem oder mehreren miteinander in Wechselbeziehung stehenden Objekten über einen bestimmten Zeitraum. Zustände werden in Form von Zeitlinien dargestellt, die auf Message Events reagieren, wobei eine Lebenslinie eine Classifier Instance oder Classifier Role darstellt.

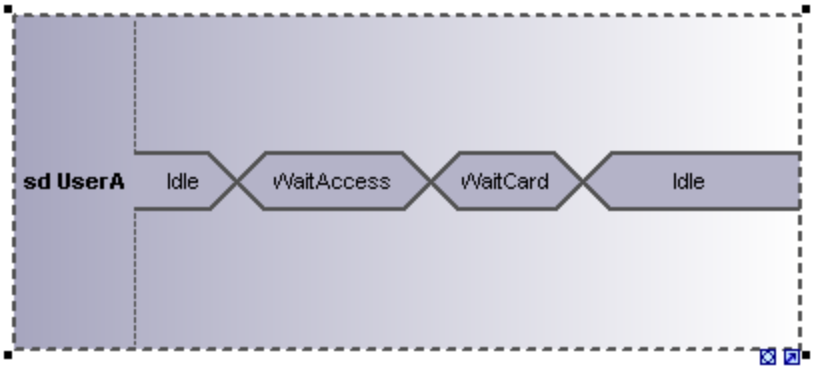
Bei Zeitverlaufsdiagrammen handelt es sich um eine Sonderform eines Sequenzdiagramms. Im Unterschied zum Sequenzdiagramm sind beim Zeitverlaufsdiagramm die Achsen vertauscht, d.h. der Zeitverlauf wird aufsteigend von links nach rechts dargestellt und Lebenslinien werden in separaten vertikal angeordneten Bereichen angezeigt.

Zeitverlaufsdiagramme werden im Allgemeinen zum Entwerfen von eingebetteter Software oder Echtzeitsystemen verwendet.



© 2016-2022 Altova GmbH

Es gibt zwei verschiedene Arten von Zeitverlaufsdiagrammen: Die eine Art enthält, wie oben gezeigt, eine Zustands-Zeitlinie, die andere allgemeinere Art enthält, wie unten gezeigt, eine Lebenslinie.



9.1.8.1 Einfügen von Elementen des Zeitverlaufsdiagramms

Verwendung der Symbolleisten-Schaltflächen

1. Klicken Sie in der Zeitverlaufsdiagramm-Symbolleiste auf das entsprechende Zeitverlaufselement.



2. Klicken Sie zum Einfügen des Elements in das Zeitverlaufsdiagramm. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen vorhandener Elemente in das Zeitverlaufsdiagramm

Sie können Elemente aus anderen Diagrammen, z.B. Klassen, in ein Zeitverlaufsdiagramm einfügen.

1. Suchen Sie das gewünschte Element auf dem Register "Modell-Struktur" (über das Suchfunktionstextfeld oder durch Drücken der Tasten Strg + F).
2. Ziehen Sie das Element/die Elemente in das Diagramm.

9.1.8.2 Lebenslinie

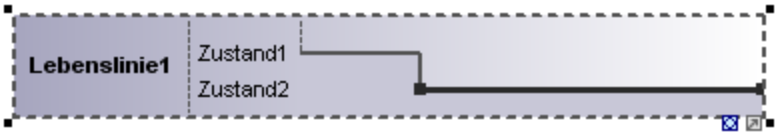
Das Element "Lebenslinie" bildet einen Teil einer Interaktion und steht in zwei Formen zur Verfügung:

1. als Zustands-Lebenslinie 
2. als allgemeine Wertverlaufslinie 

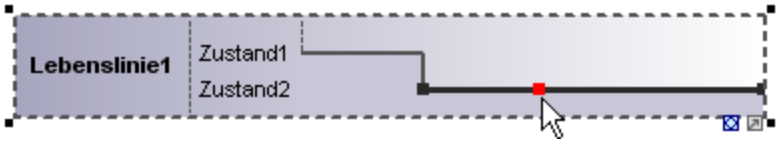
Um eine Lebenslinie **mit mehreren** Zeilen zu erstellen, drücken Sie Strg + Eingabetaste, um eine neue Zeile anzulegen.

So fügen Sie eine Zustands-Lebenslinie (Zustandsinvariante) ein und definieren Zustandsänderungen:

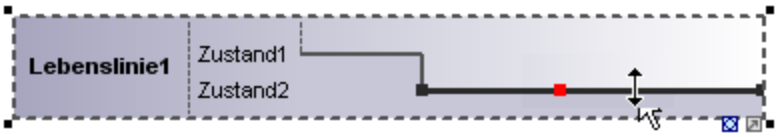
1. Klicken Sie auf das Symbol "Lebenslinie (Zustand)"  in der Titelleiste und anschließend in das Zeitverlaufsdiagramm, um die Lebenslinie einzufügen.



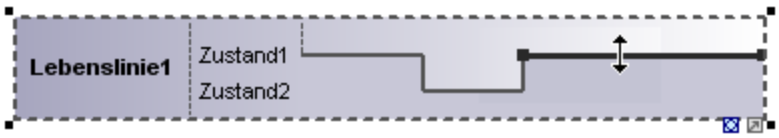
2. Geben Sie einen Namen für die Lebenslinie ein, um den Standardnamen (Lebenslinie1) wenn nötig zu ändern.
3. Platzieren Sie den Mauszeigen über einen Abschnitt einer der Zeitlinien und klicken Sie auf die linke Maustaste. Daraufhin wird die Linie ausgewählt.
4. Verschieben Sie den Mauszeiger an die Stelle, an der die Zustandsänderung eintreten soll, und klicken Sie nochmals auf die linke Maustaste. Beachten Sie: Während dieses Vorgangs wird ein Pfeil mit zwei Spitzen angezeigt. An der Klickposition erscheint ein rotes Kästchen, das die Zeile an diesem Punkt teilt.



5. Ziehen Sie den Cursor auf der Linie nach rechts und ziehen Sie die Linie nach oben.



Beachten Sie, dass die Linien nur zwischen bestehenden Zuständen der aktuellen Lebenslinie verschoben werden kann.

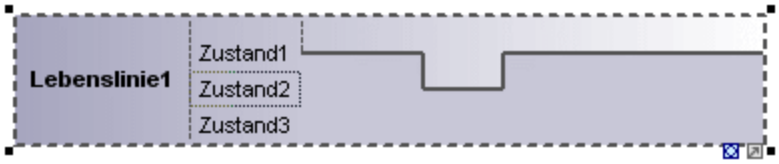


Sie können beliebig viele Zustandsänderungen pro Lebenslinie definieren. Wenn auf einer Linie ein rotes Kästchen angezeigt wird, können Sie an eine beliebige Stelle im Diagramm klicken, um es zu löschen.

So fügen Sie einen neuen Zustand zur Lebenslinie hinzu:

1. Rechtsklicken Sie auf die Lebenslinie und wählen Sie **Neu | Zustand (Zustandsinvariante)**. Daraufhin wird ein neuer Zustand, z.B. Zustand2 zur Lebenslinie hinzugefügt.

© 2016-2022 Altova GmbH



So verschieben Sie einen Zustand innerhalb einer Lebenslinie:

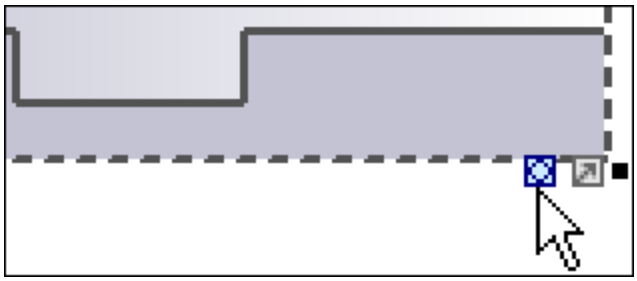
1. Klicken Sie auf die Beschriftung des Zustands, den Sie verschieben möchten.
2. Ziehen Sie den Zustand an eine andere Stelle innerhalb der Lebenslinie.

So löschen Sie einen Zustand aus einer Lebenslinie:

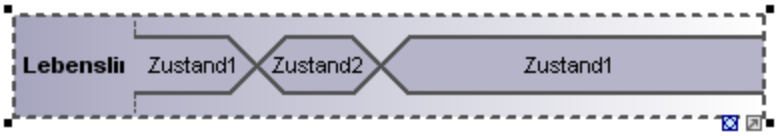
1. Klicken Sie auf den Zustand und drücken Sie die Entf-Taste oder klicken Sie auf die rechte Maustaste und wählen Sie den Befehl "Löschen".

So wechseln Sie zwischen Zeitverlaufsdiagrammtypen:

1. Klicken Sie auf das Symbol "Darstellung wechseln" rechts unterhalb der Lebenslinie.



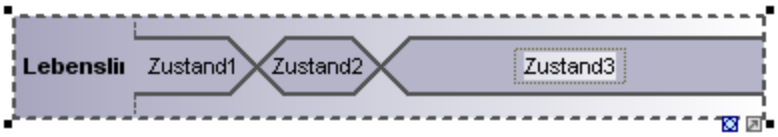
Daraufhin wird stattdessen eine Wertverlaufslebenslinie angezeigt, der Kreuzungspunkt stellt eine Änderung eines Zustands/Werts dar.



Anmerkung: Wenn Sie auf das Symbol "**Lebenslinie (allgemeiner Wert)**  klicken, wird die oben gezeigte Lebenslinie eingefügt. Sie können jederzeit zwischen den beiden Darstellungen wechseln.

So fügen Sie einen neuen Zustand zur Wertverlaufslebenslinie hinzu:

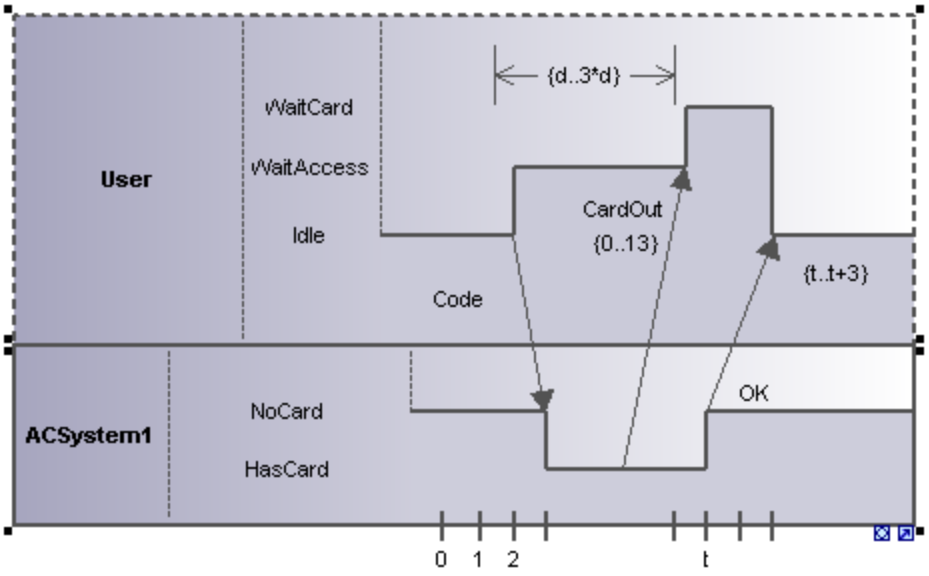
1. Rechtsklicken Sie auf die Lebenslinie und wählen Sie **Neu | Zustand (Zustandsinvariante)**.
2. Bearbeiten Sie den neuen Namen, z.B. Zustand3 und bestätigen Sie die Änderung mit der Eingabetaste.



Daraufhin wird ein neuer Zustand zur Lebenslinie hinzugefügt.

Gruppieren von Lebenslinien

Beim Platzieren oder Stapeln von Lebenslinien werden diese automatisch korrekt angeordnet, wobei auch hinzugefügte Häkchen erhalten bleiben. Durch Ziehen des entsprechenden Nachrichtenobjekts können auch Nachrichten zwischen separaten Lebenslinien erstellt werden.

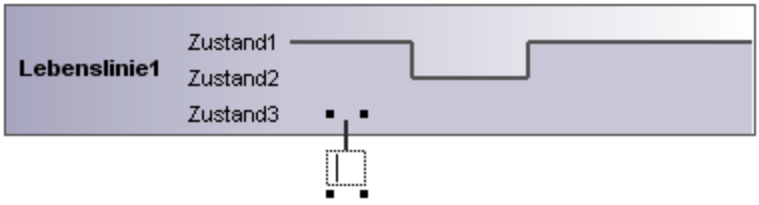


9.1.8.3 Tick-Symbol

Das **Tick-Symbol**  dient zum Einfügen der Tick-Symbole eines Zeitverlaufslineals auf einer Lebenslinie.

So fügen Sie ein Tick-Symbol ein:

1. Klicken Sie auf das Tick-Symbol in der Symbolleiste und anschließend auf die Lebenslinie, um es einzufügen.

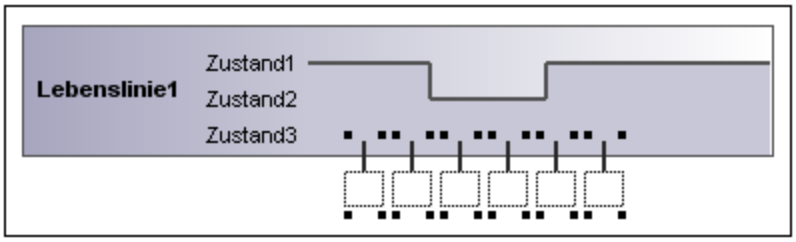


© 2016-2022 Altova GmbH

2. Um mehrere Tick-Symbole einzufügen, halten Sie die Strg-Taste gedrückt, während Sie auf verschiedene Positionen am Rand der Lebenslinie klicken.
3. Geben Sie in das dafür vorgesehene Feld eine Beschriftung für das Tick-Symbol ein.
Zum Neupositionieren der Tick-Symbole auf der Lebenslinie ziehen Sie die Symbole mit der Maus an die gewünschte Stelle.

So ordnen Sie die Tick-Symbole in gleichmäßigen Abständen auf einer Lebenslinie an:

1. Ziehen Sie im Hauptfenster ein Rechteck auf, um die einzelnen Tick-Symbole zu markieren.
2. Klicken Sie in der Symbolleiste auf das Symbol **Waagrecht anordnen** .

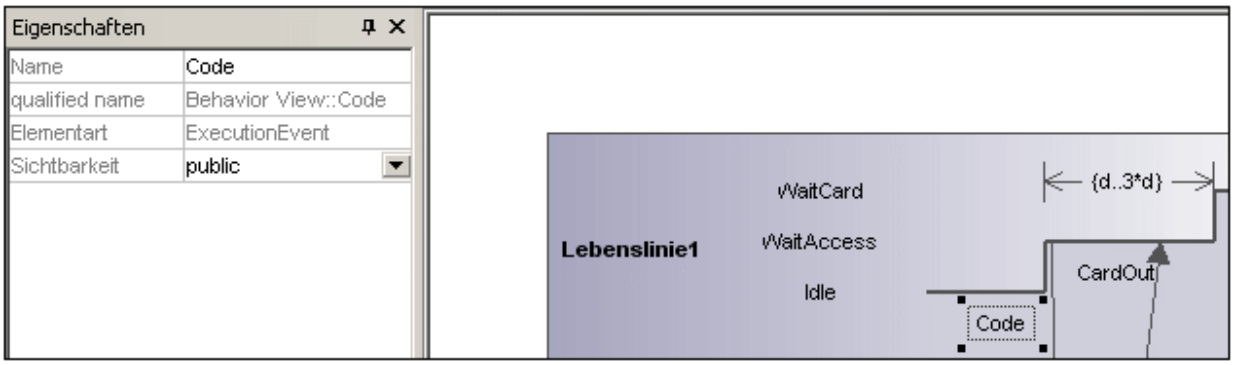


9.1.8.4 Auslösendes Ereignis

Das ExecutionEvent "Auslösendes Ereignis"  dient dazu, eine Änderung im Zustand eines Objekts anzuzeigen, die durch das entsprechende auslösende Ereignis verursacht wurde. Die eingehenden Ereignisse werden mit einer Beschriftung versehen, um das Ereignis zu kennzeichnen, das die Zustandsänderung verursacht.

So fügen Sie ein auslösendes Ereignis ein:

1. Klicken Sie auf das Symbol "Auslösendes Ereignis" und anschließend auf die entsprechende Stelle auf der Zeitlinie, an der die Veränderung stattfindet.



2. Geben Sie einen Namen für das Ereignis ein. In diesem Beispiel heißt das Ereignis "Code".

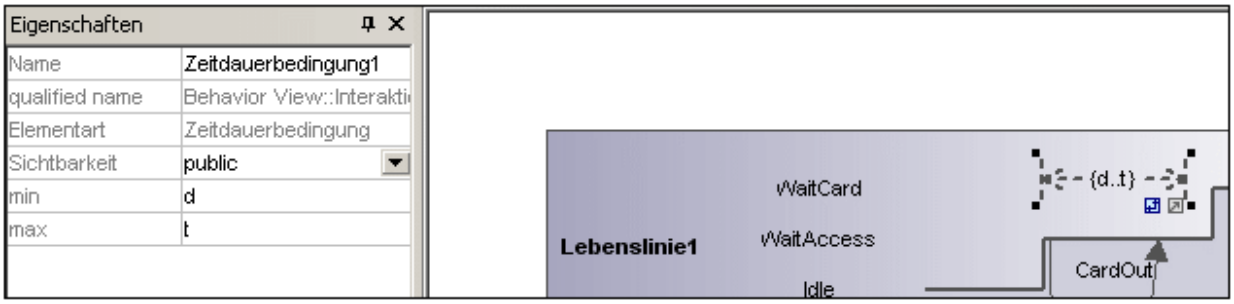
Beachten Sie, dass die Eigenschaften des Ereignisses auf dem Register "Eigenschaften" angezeigt werden.

9.1.8.5 Zeitdauerbedingung

Eine **Zeitdauerbedingung**  definiert eine Wertespezifikation zur Angabe einer Zeitdauer zwischen einem Start- und einem Endpunkt. Eine Zeitdauer ist oft ein Ausdruck zur Darstellung der Ticks der Uhr, die während dieser Zeitdauer verstreichen.

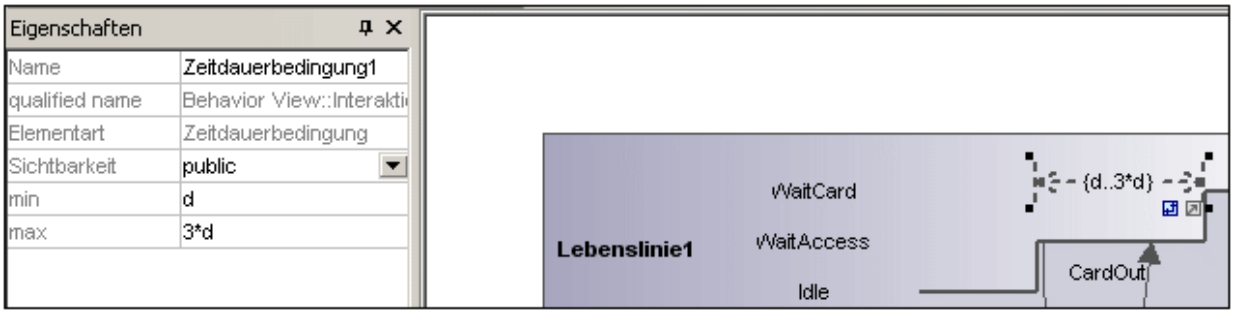
So fügen Sie eine Zeitdauerbedingung ein:

1. Klicken Sie auf das Symbol "Zeitdauerbedingung" und anschließend auf die Stelle auf der Lebenslinie, an der die Bedingung angezeigt werden soll.



Die Standard-Minimum- und Maximum-Werte "d..t" werden automatisch vorgegeben. Durch Doppelklick auf die Zeitdauerbedingung oder durch Bearbeitung der Werte im Fenster "Eigenschaften" können Sie diese Werte bearbeiten.

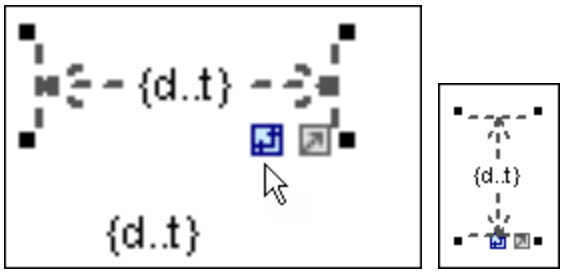
2. Mit Hilfe der Ziehpunkte können Sie die Größe des Objekts gegebenenfalls anpassen.



Ändern der Ausrichtung der Zeitdauerbedingung:

1. Klicken Sie auf das "Umdrehen"-Symbol, zum die Bedingung vertikal auszurichten.

© 2016-2022 Altova GmbH

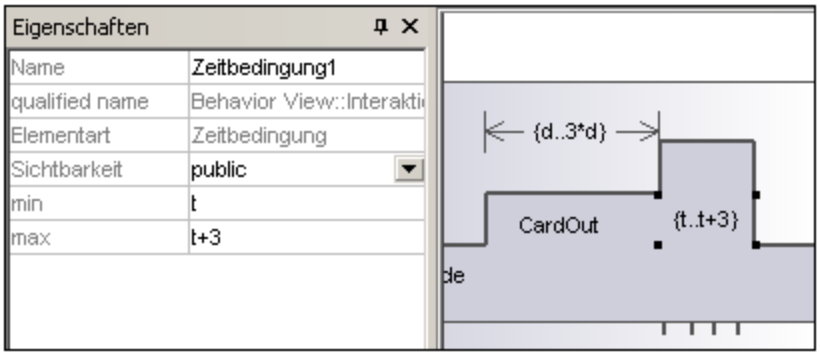


9.1.8.6 Zeitbedingung

Eine **Zeitbedingung** wird im Allgemeinen als grafische Assoziation zwischen einem Zeitintervall und dem Konstrukt, das es einschränkt, dargestellt. Normalerweise handelt es sich um eine grafischen Assoziation zwischen dem Auftreten eines Ereignisses und einem Zeitintervall.

So fügen Sie eine Zeitbedingung ein:

1. Klicken Sie auf das Symbol "Zeitbedingung" und anschließend auf die entsprechende Position auf der Lebenslinie, an der die Bedingung angezeigt werden soll.



Die Standard-Minimum- und Maximum-Werte "d..t" werden automatisch vorgegeben. Durch Doppelklick auf die Zeitbedingung oder durch Bearbeitung der Werte im Fenster "Eigenschaften" können Sie diese Werte bearbeiten.

9.1.8.7 Nachricht

Eine Nachricht ist ein Modellierungselement, das eine bestimmte Art von Kommunikation in einer Interaktion definiert. Dabei kann es sich z.B. um das Auslösen eines Signals, den Aufruf einer Operation, das Erstellen oder Löschen einer Instanz handeln. Die Nachricht definiert die Art der durch die absendende Ausführungsspezifikation definierten Kommunikation sowie den Absender und den Empfänger.

Verwenden Sie die folgenden Symbolleisten-Schaltflächen, um bestimmte Nachrichtentypen hinzuzufügen:

Message (Aufruf)

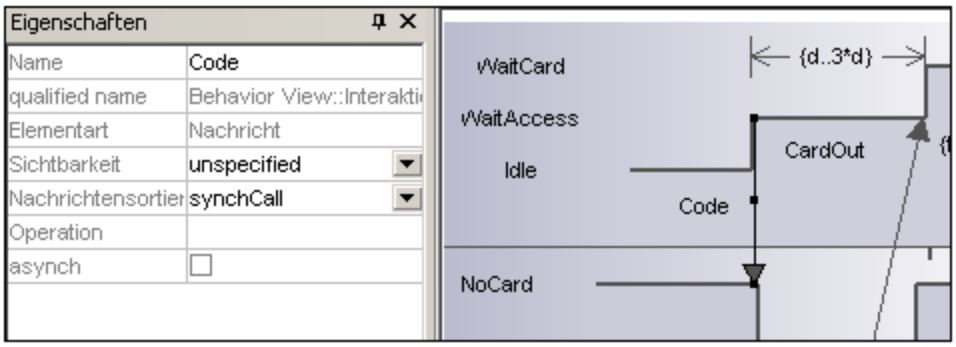
 Nachricht (Antwort)

 Asynchrone Nachricht (Aufruf)

Nachrichten werden zwischen sendenden und empfangenden Zeitlinien gesendet und werden als beschriftete Pfeile angezeigt.

So fügen Sie eine Nachricht ein:

1. Klicken Sie auf das entsprechende Nachrichtensymbol in der Symbolleiste.
2. Klicken Sie an eine beliebige Stelle auf der Zeitlinie für das sendende Objekt, z.B. auf Idle.
3. Ziehen Sie die Nachrichtenlinie auf die Zeitlinie der empfangenden Objekte, z.B. NoCard. Die Lebenslinie erscheint markiert, wenn die Nachricht an diese bestimmte Stelle gezogen werden kann.



- Die Richtung, in die Sie den Pfeil ziehen, bestimmt die Richtung der Nachricht. Antwortnachrichten können in jede der beiden Richtungen weisen.
- Wenn Sie auf ein Nachrichtensymbol klicken und dabei die Strg-Taste gedrückt halten, können Sie mehrere Nachrichten einfügen, indem Sie mehrmals ins Diagramm klicken und die Maus ziehen.

So löschen Sie eine Nachricht:

1. Klicken Sie auf die gewünschte Nachricht.
2. Drücken Sie die Entf-Taste, um die Nachricht aus dem Modell zu löschen oder rechtsklicken Sie darauf und wählen Sie den Befehl "Aus Diagramm löschen".

© 2016-2022 Altova GmbH

9.2 Strukturdiagramme

Diese Diagramme dienen zur Darstellung der Strukturelemente, aus denen ein System oder eine Funktion besteht. Es werden sowohl die statischen, z.B. Klassendiagramm, als auch die dynamischen Beziehungen, z.B. Objektdiagramm, dargestellt.

Klassendiagramm
Komponentendiagramm
Kompositionsstrukturdiagramm
Deployment-Diagramm
Objektdiagramm
Paketdiagramm
Profildiagramm 476

9.2.1 Klassendiagramm

Dieser Abschnitt enthält die folgenden Kapitel zu Aufgaben und Begriffen im Zusammenhang mit Klassendiagrammen:

- Anpassen von Klassendiagrammen 449
- Außerkraftsetzen von Basisklassenoperationen und Implementieren von Schnittstellenoperationen 456
- Erstellen von Getter / Setter-Methoden 457
- Ball-and-socket Notation 459
- Hinzufügen von Ausnahmeereignissen zu Methoden einer Klasse 460
- Hinzufügen von Signalempfängern zu einer Klasse 461
- Generieren von Klassendiagrammen 462

Eine grundlegende Einführung in Klassendiagramme finden Sie unter Klassendiagramme 31 im Abschnitt "Tutorial" diese Dokumentation.

9.2.1.1 Anpassen von Klassendiagrammen

Erweitern / Ausblenden von Klassenbereichen in einem UML-Diagramm

Es gibt verschiedene Methoden, um die verschiedenen Bereiche von Klassendiagrammen zu erweitern.

- Klicken Sie auf das **+** bzw. das **-** Symbol der gerade aktiven Klasse, um den jeweiligen Bereich zu erweitern/auszublenden.
- Ziehen Sie ein Rechteck (über dem Diagrammhintergrund) auf, um **mehrere** Klassen zu markieren und klicken Sie anschließend die Schaltfläche "Erweitern/Ausblenden". Sie können mehrere Klassen auch mit Hilfe von **Strg+Klick** auswählen.
- Drücken Sie **Strg + A**, um **alle Klassen** auszuwählen. Klicken Sie anschließend in einer der Klassen auf die Schaltfläche "Erweitern/Reduzieren", um die entsprechenden Bereiche zu erweitern bzw. zu reduzieren.

Erweitern/Reduzieren von Klassenbereichen in der Modellstruktur

In der Modellstruktur sind Klassen Unterelemente von Paketen und Sie können entweder die Pakete oder die Klassen erweitern bzw. reduzieren.

- Klicken Sie auf das Paket / die Klasse um es/sie zu **erweitern** und:
  - Drücken Sie die ***-Taste, um das aktuelle Paket/die aktuelle Klasse und alle Subelemente zu erweitern
  - Drücken Sie die **+**-Taste um das aktuelle Paket/die aktuelle Klasse zu öffnen.

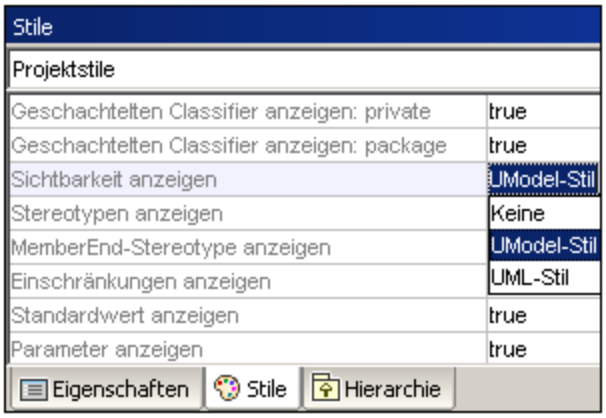
Um das Paket/die Klassen zu **reduzieren**, drücken Sie die **-** Taste.

Sie können dazu die Standardtasten der Tastatur oder die Tasten des Ziffernblocks verwenden.

Ändern der Symbole für die Sichtbarkeit

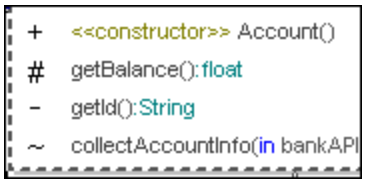
Wenn Sie auf das **Sichtbarkeitssymbol** links von einer Operation oder Eigenschaft klicken, wird eine Dropdown-Liste geöffnet, in der Sie den Sichtbarkeitsstatus ändern können. Sie können auch ändern, welche Art von Sichtbarkeitssymbol angezeigt werden soll.

- Klicken Sie im Diagrammfenster auf eine Klasse, anschließend auf das Register **Stile** und scrollen Sie in der Liste hinunter bis zum Eintrag **Sichtbarkeit anzeigen**.



  Sie können wählen zwischen dem oben gezeigten UModel Typ und den unten gezeigten UML-konformen Symbolen.

© 2016-2022 Altova GmbH

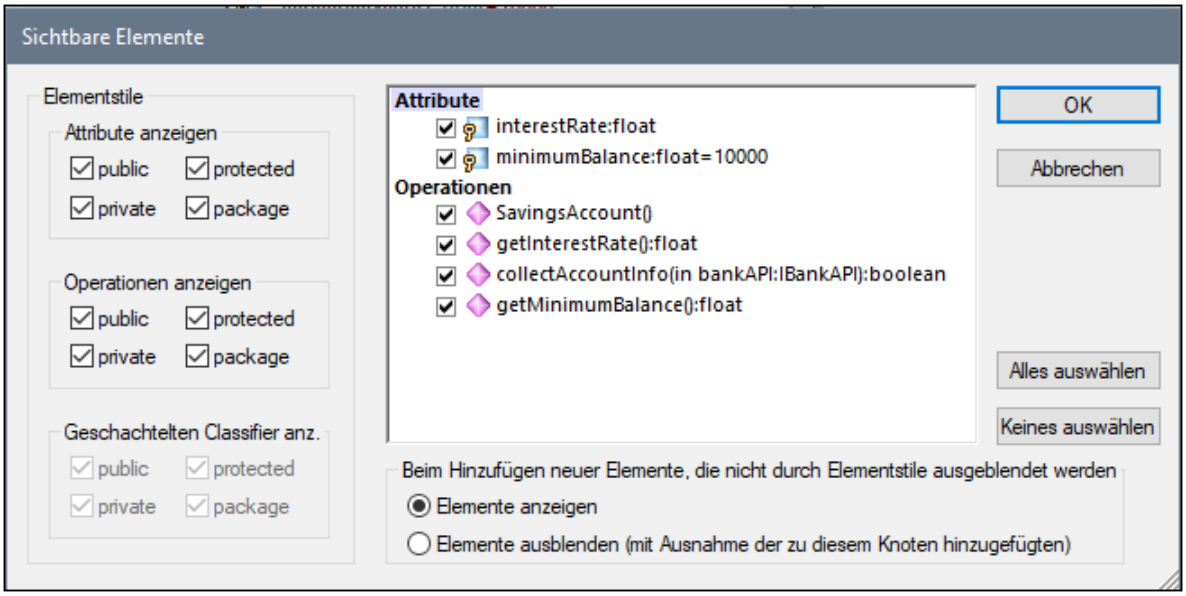


Ein-/Ausblenden von Knoteninhalt (Klassenattribute, Operationen, Slots)

Sie können bestimmte Mitglieder einer Klasse wie Attribute oder Operationen in Klassendiagrammen ein- oder ausblenden. Sie können nicht nur einzelne Mitglieder, sondern auch mehrere Mitglieder desselben Typs nach ihrer Sichtbarkeit ein- oder ausblenden. So können Sie z.B. nur diejenigen Klassenattribute ausblenden, die eine private Sichtbarkeit haben. Das Ein- oder Ausblenden wird auch für Objekt-Slots (`Instanzspezifikationen`) in Objektdiagrammen unterstützt.

So blenden Sie Klassenmitglieder oder Objekt-Slots ein oder aus:

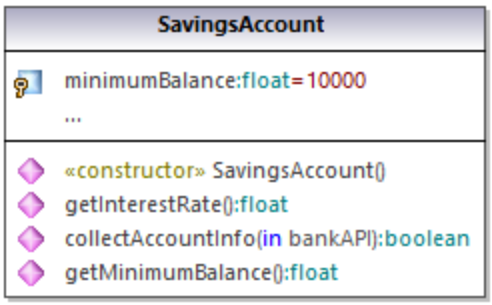
1. Klicken Sie mit der rechten Maustaste auf eine Klasse (z.B. `SavingsAccount` aus dem Beispielprojekt **Bank_MultiLanguage.ump**) und wählen Sie im Kontextmenü den Befehl **Knoteninhalt ein-/ausblenden**.
2. Aktivieren oder deaktivieren Sie das Kontrollkästchen neben den Mitgliedern, die ein- bzw. ausgeblendet werden sollen.



Um mehrere Mitglieder auf Basis ihrer Sichtbarkeit anzuzeigen, verwenden Sie die Kontrollkästchen in der Gruppe **Elementstile**. Wenn Sie z.B. in der Gruppe **Attribute anzeigen** das Kontrollkästchen **protected** deaktivieren, werden alle geschützten Attribute der Klasse ausgeblendet.

Anmerkung: Eigenschaftswerte von ausgeblendeten Elementen werden ebenfalls ausgeblendet, wenn Sie die Option "Ausblenden" auswählen.

Nachdem Sie die Einstellungen mit **OK** bestätigt und das Dialogfeld geschlossen haben, werden alle ausgeblendeten Mitglieder im Diagramm durch das Auslassungssymbol `...` ersetzt. Um das Dialogfeld wieder zu öffnen, doppelklicken Sie auf das Auslassungszeichen.



Mit Hilfe der Option **Beim Hinzufügen neuer Elemente, die nicht durch Elementstile ausgeblendet werden** können Sie festlegen, was sichtbar gemacht werden soll, wenn neue Elemente zur Klasse hinzugefügt werden. Dies gilt nicht nur für Elemente, die im Diagramm oder der Modell-Struktur manuell hinzugefügt wurden, sondern auch für solche, die während des Code Engineering automatisch hinzugefügt wurden. Für diese Option gibt es die folgenden gültigen Werte:

Elemente anzeigen	Wenn ein neues Mitglied zur Klasse hinzugefügt wird, wird dieses im Diagramm angezeigt. Wenn das Element jedoch aufgrund einer der Optionen unter "Elementstile" ausgeblendet werden muss, wird es ausgeblendet.
Elemente ausblenden (mit Ausnahme der zu diesem Knoten hinzugefügten)	Hier bezieht sich "Knoten" auf die aktuelle Instanz der Klasse im Diagramm. (Bedenken Sie, dass dieselbe Klasse mehrmals zum selben Diagramm hinzugefügt werden kann, siehe Umbenennen, Verschieben und Kopieren von Elementen 116.) Wenn im Diagramm zwei oder mehr Instanzen derselben Klasse vorhanden sind und ein neues Mitglied zu *dieser Instanz* der Klasse hinzugefügt wird, wird das Mitglied in allen Instanzen der Klasse ausgeblendet und nur für die aktuelle Instanz angezeigt.

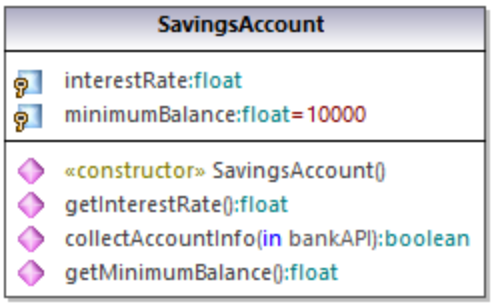
Ein Beispiel für die Verwendung der obigen Optionen finden Sie im Beispielprojekt **Bank_MultiLanguage.ump**. Suchen Sie darin nach dem Klassendiagramm "Hierarchy of Account".

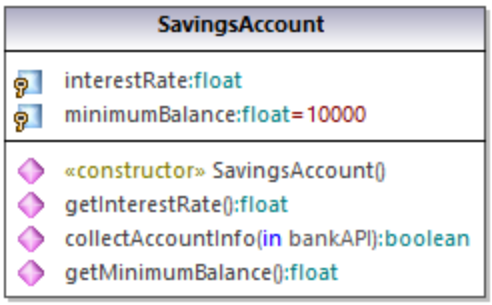
Erstellen Sie als nächstes folgendermaßen eine neue Instanz der Klasse `SavingsAccount`:

1. Rechtsklicken Sie im Diagramm auf die Klasse `SavingsAccount` und wählen Sie **Kopieren**.
2. Klicken Sie mit der rechten Maustaste auf einen leeren Bereich im selben Diagramm und wählen Sie im Kontextmenü den Befehl **Nur in Diagramm einfügen**.

Es gibt nun im Diagramm zwei Instanzen der Klasse `SavingsAccount`.

 © 2016-2022 Altova GmbH

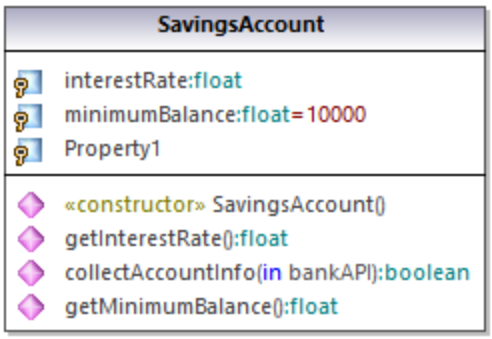


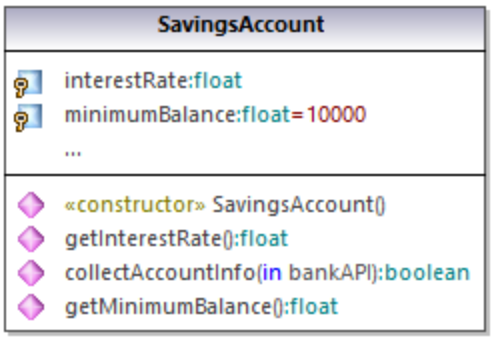


Definieren Sie als nächstes in jeder der Instanzen verschiedene Sichtbarkeitsoptionen:

1. Klicken Sie mit der rechten Maustaste auf die linke Instanz der Klasse, wählen Sie **Knoteninhalt ein-/ausblenden** und aktivieren Sie anschließend die Option **Elemente anzeigen**.
2. Klicken Sie mit der rechten Maustaste auf die rechte Instanz der Klasse, wählen Sie **Knoteninhalt ein-/ausblenden** und aktivieren Sie anschließend die Option **Elemente ausblenden (mit Ausnahme der zu diesem Knoten hinzugefügten)**.

Fügen Sie als nächstes eine neue Eigenschaft zur linken Instanz hinzu (wählen Sie die Klasse aus und drücken Sie **F7**). Wie unten gezeigt, ist die neue Eigenschaft (`Eigenschaft1`) in der linken Instanz sichtbar, nicht aber in der rechten Instanz. Der Grund dafür ist, dass für die rechte Instanz der Klasse die Option **Elemente ausblenden (mit Ausnahme der zu diesem Knoten hinzugefügten)** aktiviert wurde.



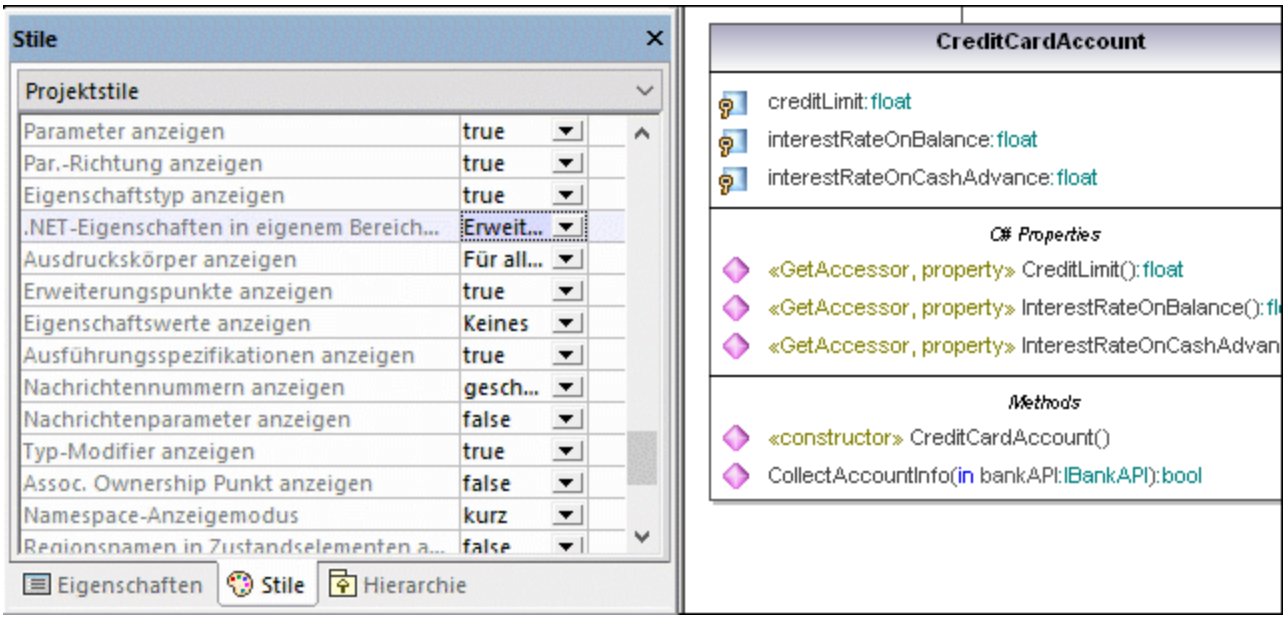


Fügen Sie abschließend eine neue Eigenschaft zur rechten Instanz der Klasse hinzu. Wie unten gezeigt, ist die neue Eigenschaft (`Eigenschaft2`) in beiden Instanzen sichtbar. Der Grund dafür ist, dass neue Elemente laut Konfiguration für die linke Instanz angezeigt werden sollen, während die rechte Instanz die *aktuelle Instanz* ist, zu der die Eigenschaft hinzugefügt wird, weswegen die neue Eigenschaft auf jeden Fall angezeigt wird.



Ein- /Ausblenden von .NET-Bereichen

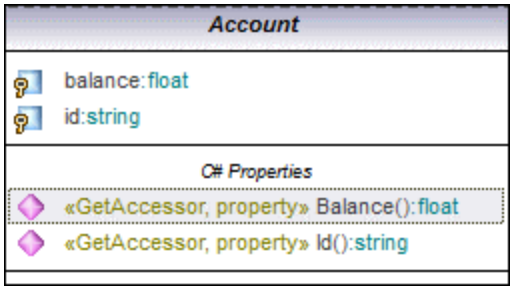
Um .NET-Eigenschaften in ihrem eigenen Bereich anzuzeigen, aktivieren Sie auf dem Register **Stile** die Option ".NET-Eigenschaften in eigenem Bereich anzeigen".



Anzeige von .NET-Eigenschaften als Assoziationen

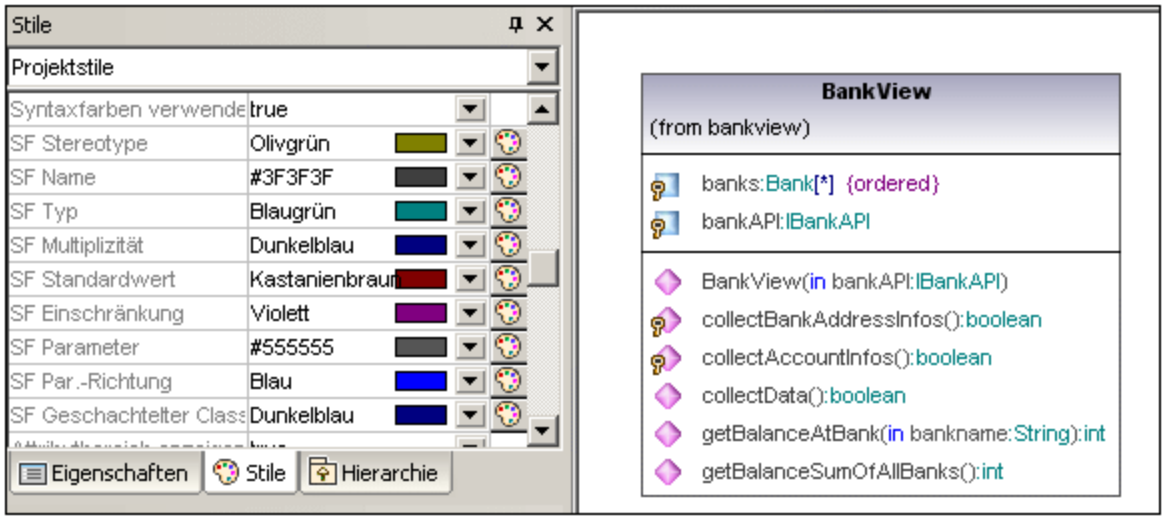
Um .NET-Eigenschaften als Assoziationen anzuzeigen, klicken Sie mit der rechten Maustaste, wie unten gezeigt, auf eine C#-Eigenschaft und wählen Sie den Befehl **Anzeigen | Alle .NET-Eigenschaften als Assoziationen**.

© 2016-2022 Altova GmbH



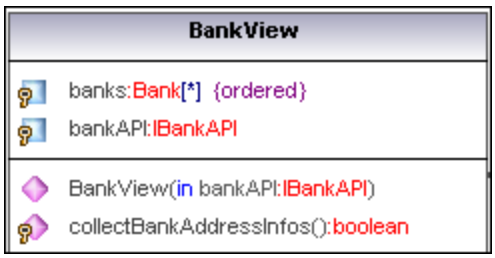
Ändern der Syntaxfarbe von Operationen/Eigenschaften

Die Syntaxfärbung ist in UModel automatisch aktiviert. Sie können die Syntaxfarben allerdings nach Ihren eigenen Wünschen anpassen. In der Abbildung unten sehen Sie die Standardeinstellungen.



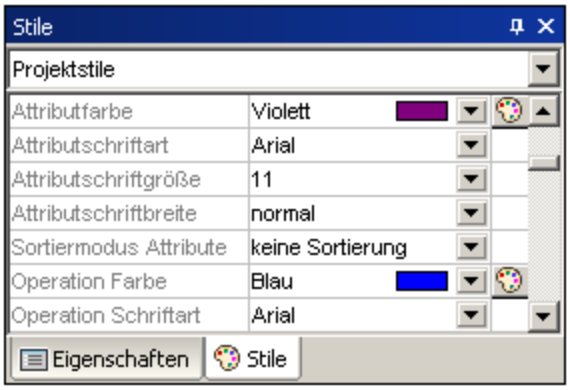
So ändern Sie die Standardeinstellung für die Syntaxfarben (siehe unten):

1. Wechseln Sie zum Register **Stile** und scrollen Sie zu den vordefinierten **SF**-Einträgen.
2. Ändern Sie einen der SF-Farbeinträge, z.B. "SF-Typ" zu "rot".



So deaktivieren Sie die Syntaxfärbung:

1. Wechseln Sie zum Register **Stile** und ändern Sie den Eintrag **Syntaxfarben verwenden** in **false**.
2. Über die Einträge **Attributfarbe** oder **Operation Farbe** auf dem Register **Stile** können Sie diese Einstellungen in der Klasse ändern.

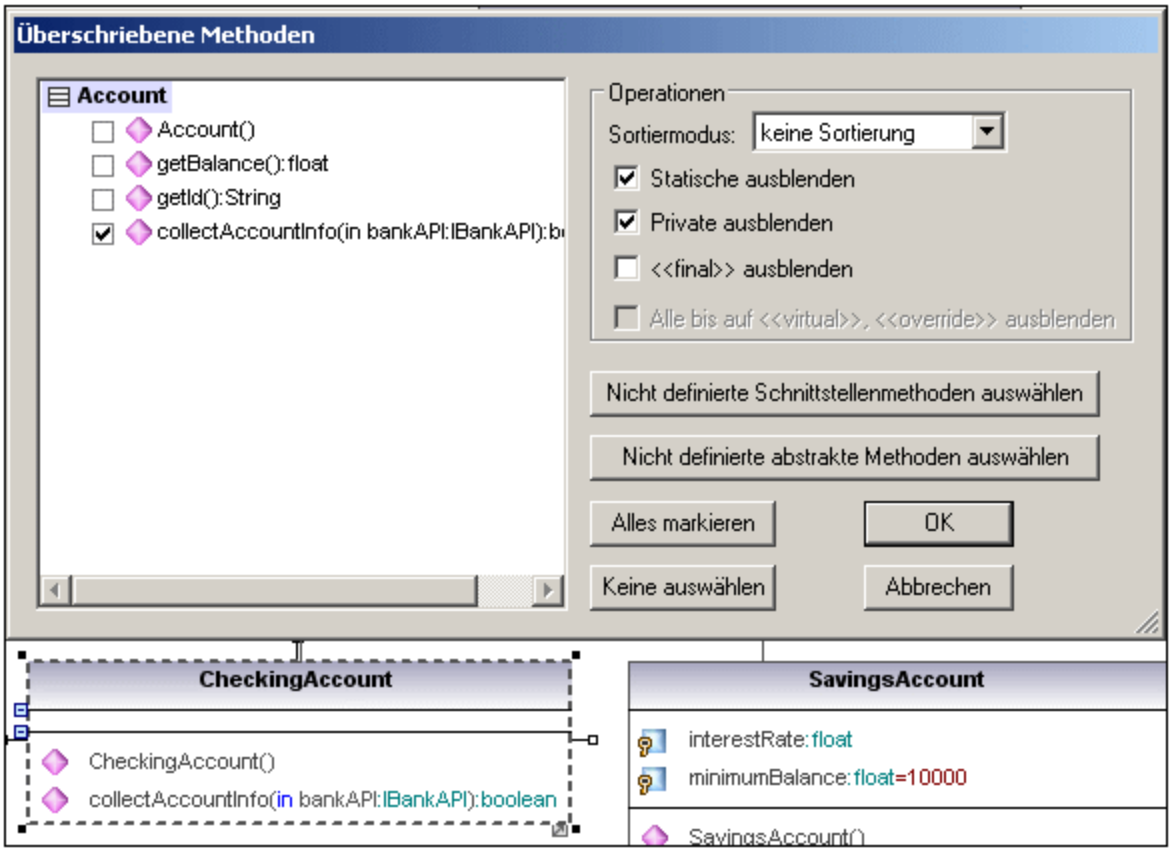


9.2.1.2 Außerkraftsetzen von Basisklassenoperationen und Implementieren von Schnittstellenoperationen

Sie haben in UModel die Möglichkeit, die Basisklassenoperationen/Schnittstellen vor der Code Engineering-Phase außer Kraft zu setzen oder Schnittstellenoperationen einer Klasse zu implementieren. Die kann über die Modell-Struktur, das Register "Favoriten" oder in Klassendiagrammen durchgeführt werden.

1. Rechtsklicken Sie auf eine der abgeleiteten Klassen im Klassendiagramm, z.B. auf CheckingAccount und wählen Sie den Befehl **Operationen überschreiben/implementieren**.
   Daraufhin wird das unten gezeigte Dialogfeld geöffnet.

© 2016-2022 Altova GmbH



2. Wählen sie die Operationen aus, die Sie außer Kraft setzen möchten und bestätigen Sie mit OK. Mit Hilfe der Schaltflächen "Nicht definierte...auswählen" werden die jeweiligen Methodenarten im Fenster auf der linken Seite ausgewählt.

Anmerkung:
Beim Öffnen des Dialogfelds werden Operationen von Basisklassen und implementierten Schnittstellen, die dieselbe Signatur wie bestehende Operationen haben, automatisch ausgewählt, d.h. aktiviert.

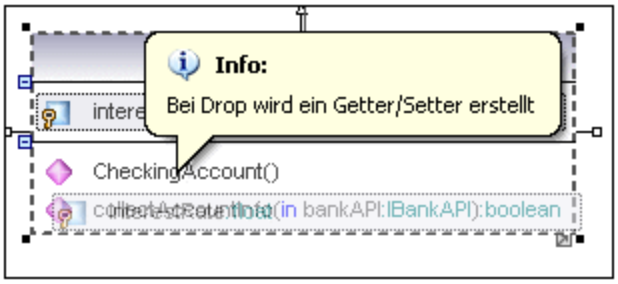
9.2.1.3 Erstellen von Getter / Setter-Methoden

Während des Modellierens müssen oft get/set-Methoden für bestehende Attribute erstellt werden. UModel bietet für diesen Zweck zwei separate Methoden:

- Verschieben eines Attributs mit Hilfe von Drag and Drop in den Operation-Bereich
- Verwendung des Kontextmenüs zum Öffnen eines Dialogfelds, in dem Sie die get/set-Methoden verwalten können

So erstellen Sie Getter/Setter-Methoden mittels Drag and Drop:

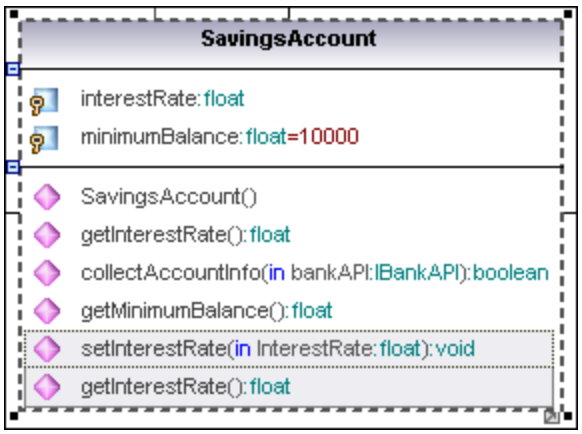
1. Ziehen Sie ein Attribut aus dem Attribut-Bereich in den Operations-Bereich.



Daraufhin wird ein Popup-Fenster angezeigt, in dem Sie auswählen können, welche Art von get/set-Methode erstellt werden soll.



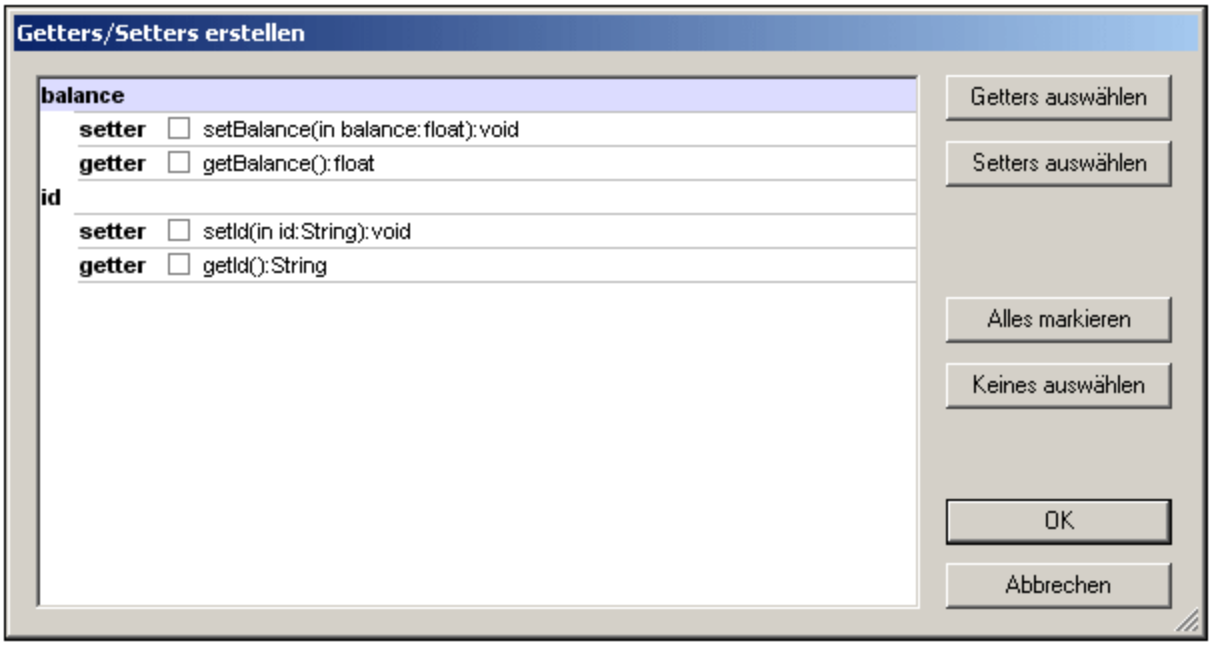
Wenn Sie den ersten Eintrag auswählen, wird eine get- und set-Methode für interestRate:float erstellt.



So erstellen Sie Getter/Setter-Methoden über das Kontextmenü:

1. Rechtsklicken Sie auf den Namen einer Klasse, z.B. SavingsAccount und wählen Sie im Kontextmenü die Option **Getter/Setter-Operationen erstellen**.

© 2016-2022 Altova GmbH



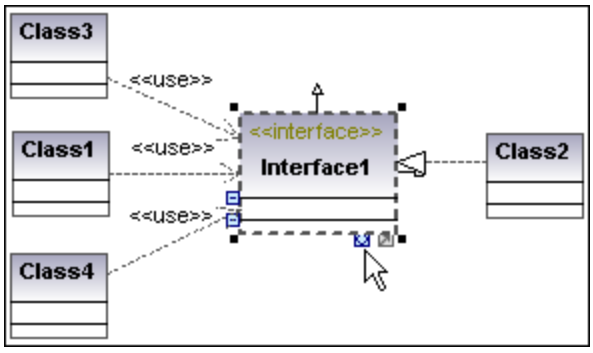
Daraufhin erscheint das Dialogfeld "Getters/Setters erstellen", in dem alle in der derzeit aktiven Klasse vorhandenen Attribute angezeigt werden.

2. Verwenden Sie die Schaltflächen, um die Einträge als Gruppe auszuwählen oder klicken Sie auf die Kontrollkästchen der einzelnen Methoden.

Anmerkung:
Sie können auch auf ein einzelnes Attribut rechtsklicken und auf dieselbe Art eine Operation dafür erstellen.

9.2.1.4 Ball-and-socket Notation

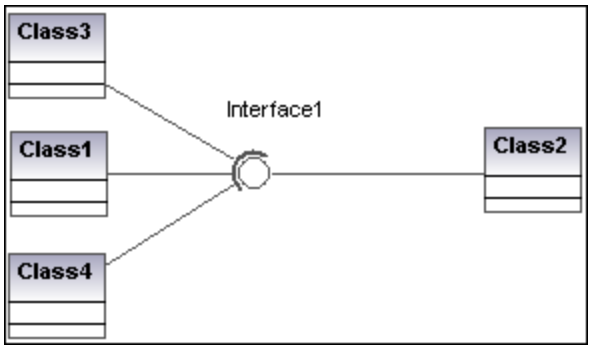
UModel unterstützt die Ball-and-socket Notation von UML-Klassen, die eine Schnittstelle benötigen, ein "Socket" (Buchse) und den Schnittstellennamen anzeigen, während Klassen, die eine Schnittstelle implementieren den "ball" (Stecker) anzeigen.



In den Abbildungen oben realisiert Class2 Interface1, welche von den Klassen 1, 3 und 4 verwendet wird. Die Verwendungssymbole dienten zum Erstellen der Verwendungsbeziehung zwischen den Klassen und der Schnittstelle.

So wechseln Sie zwischen der Standard- und der Ball-and-socket-Ansicht:

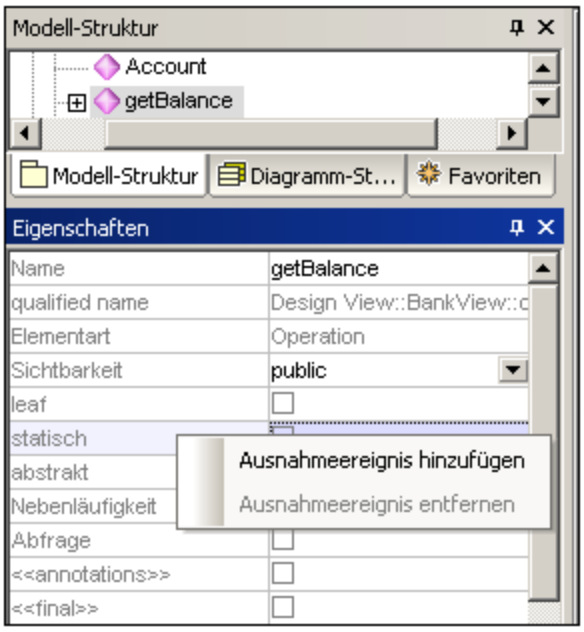
- Klicken Sie auf das Symbol "Darstellung wechseln" am unteren Rand des Schnittstellenelements.



9.2.1.5 Hinzufügen von Ausnahmeereignissen zu Methoden einer Klasse

So fügen Sie Ausnahmeereignisse zu Methoden einer Klasse hinzu:

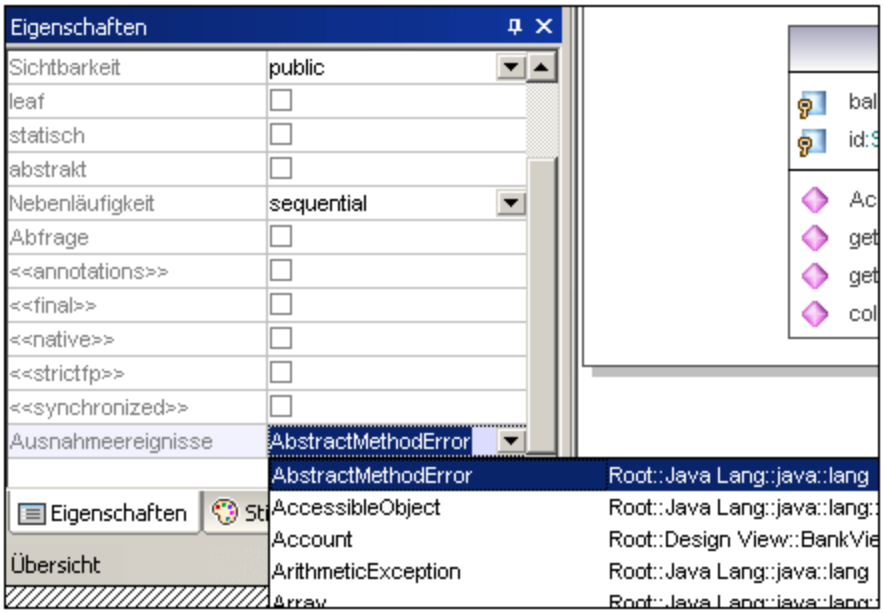
1. Klicken Sie im Fenster "Modellstruktur" auf die Methode der Klasse, zu der Sie das Ausnahmeereignis hinzufügen möchten, z.B. in der Klasse "Account" auf getBalance.
2. Rechtsklicken Sie in das Fenster "Eigenschaften" und wählen Sie im Popup-Menü den Befehl **Ausnahmeereignis hinzufügen**.



© 2016-2022 Altova GmbH

Daraufhin wird das Feld "Ausnahmeereignis" zum Fenster "Eigenschaften" hinzugefügt und der erste Eintrag im Popup-Menü wird automatisch markiert.

3. Wählen Sie einen Eintrag im Popup-Menü aus oder geben Sie Ihren eigenen in das Feld ein.



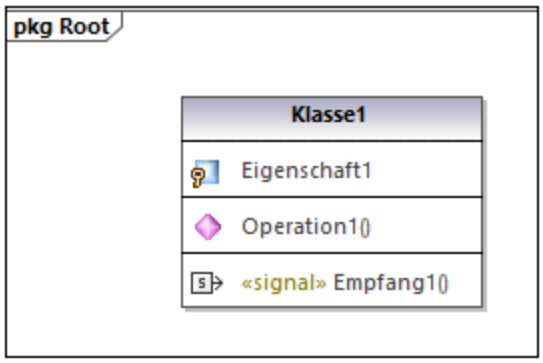
9.2.1.6 Hinzufügen von Signalempfängern zu einer Klasse

Neben Operationen und Eigenschaften können Sie auch Signalempfängerelemente zu einer Klasse hinzufügen.

So fügen Sie einen Signalempfänger zu einer Klasse hinzu:

- Klicken Sie mit der rechten Maustaste in das Diagramm und wählen Sie im Kontextmenü den Befehl **Neu | Empfang**.

Signalempfänger werden ähnlich Eigenschaften und Operationen in einem eigenen Bereich im Klassendiagramm angezeigt, z.B:



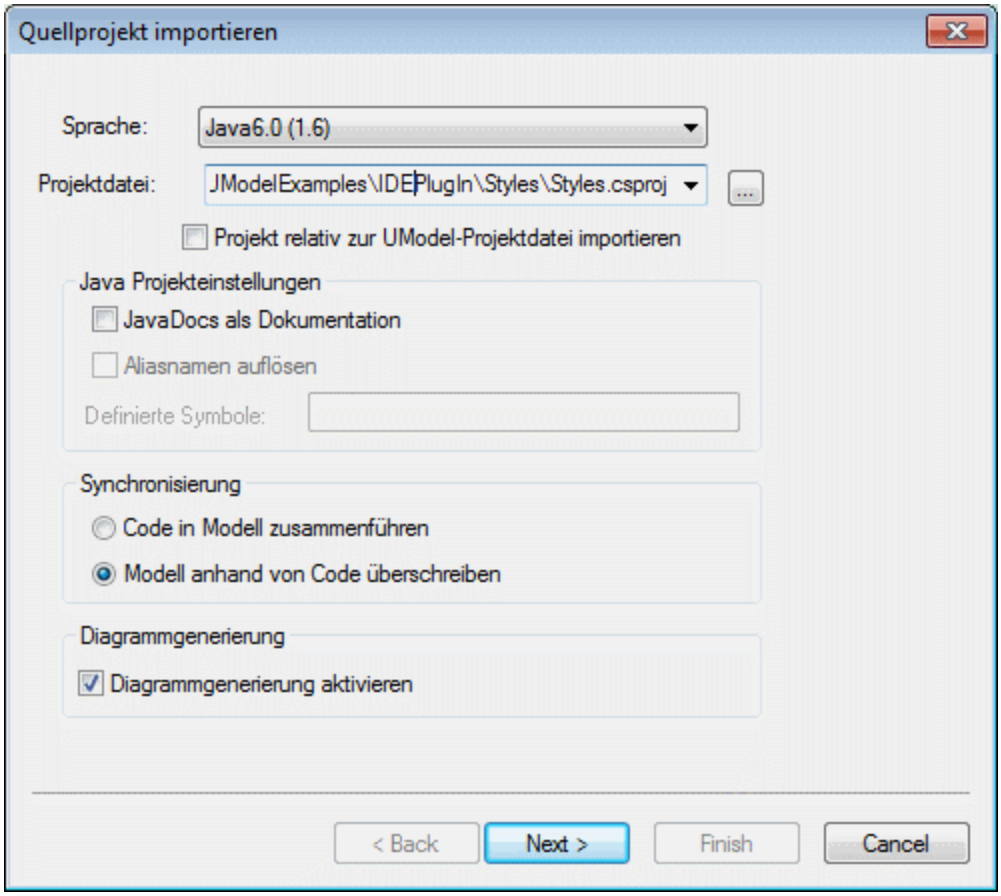
Signalempfänger haben dieselben Stile wie Operationen. D.h., immer wenn Sie den Stil von Operationen ändern, wirken sich diese Änderungen auch auf Signalempfänger aus. Nähere Informationen dazu finden Sie unter Ändern des Stils von Elementen 126.

9.2.1.7 Generieren von Klassendiagrammen

Anstatt Klassendiagramme direkt in UModel zu erstellen, können Sie diese beim Import von Quellcode oder Binärdateien in UModel-Projekte automatisch generieren (siehe Importieren von Quellcode in Projekte 209 und Importieren von Java-, C#- und VB.NET-Binärdateien 225). Wenn Sie den Assistenten verwenden, achten Sie auf folgende Dinge:

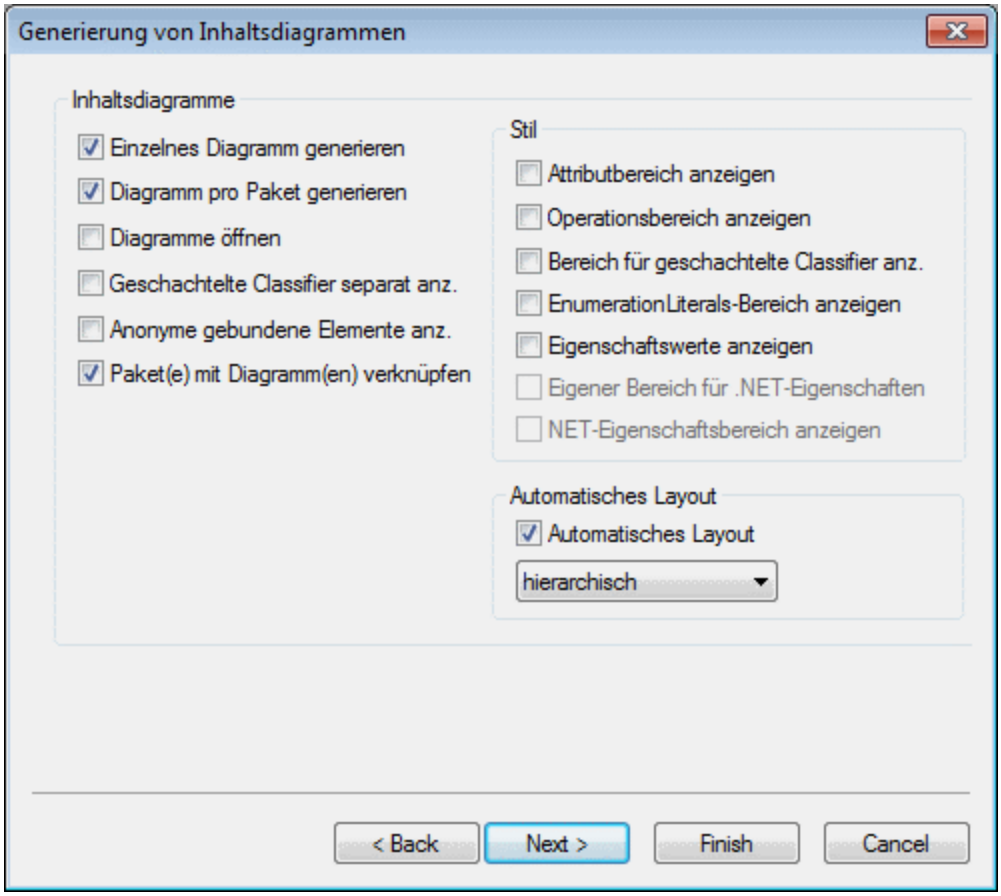
1) Im Dialogfeld "Quellprojekt importieren", "Binärtypen importieren" bzw. "Quellverzeichnis importieren" muss das Kontrollkästchen **Diagrammgenerierung aktivieren** aktiviert sein.

 © 2016-2022 Altova GmbH



Dialogfeld "Quellprojekt importieren"

2) Im Dialogfeld "Generierung von Inhaltsdiagrammen" müssen die Optionen **Einzelnes Diagramm generieren** und/oder **Diagramm pro Paket generieren** aktiviert sein.



Dialogfeld "Generierung von Inhaltsdiagrammen"

Nach Fertigstellung des Imports stehen die generierten Klassendiagramme in der Diagramm-Struktur unter "Klassendiagramme" zur Verfügung.

© 2016-2022 Altova GmbH

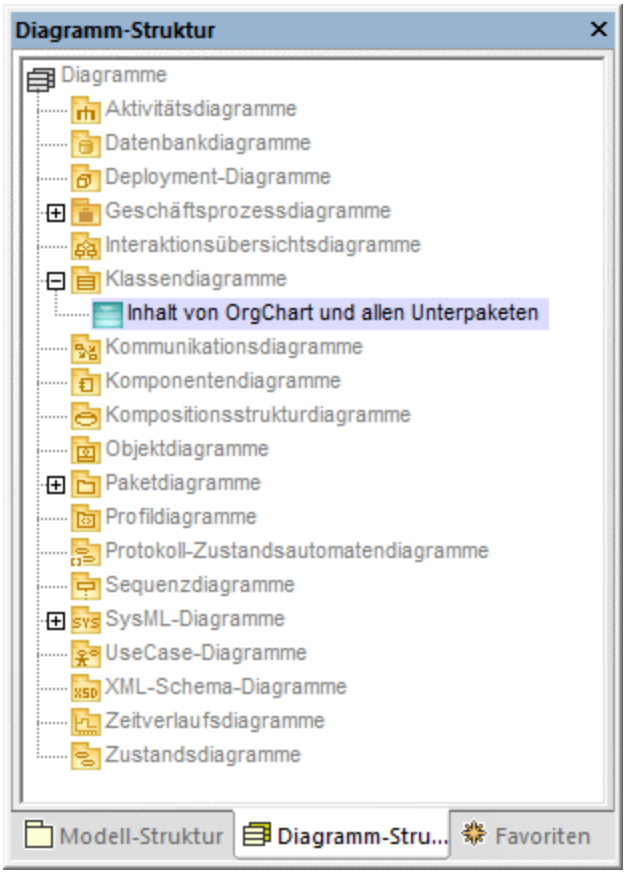
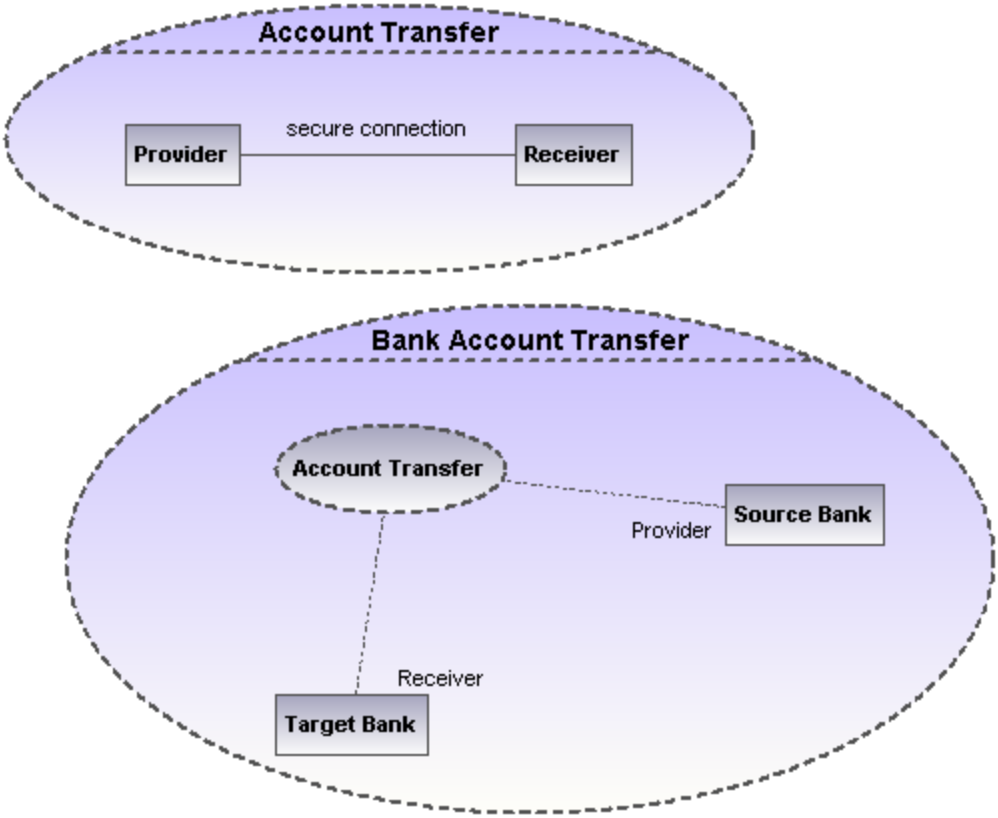


Diagramm-Struktur

9.2.2 Kompositionsstrukturdiagramm

Altova Website: UML-Kompositionsstrukturdiagramme

In UML 2.0 wurde das Kompositionsstrukturdiagramm hinzugefügt. Es dient dazu, die interne Struktur einschließlich Bereichen, Ports und Konnektoren eines strukturierten Classifier oder einer Kollaboration anzuzeigen.



9.2.2.1 Einfügen von Elementen eines Kompositionsstrukturdiagramms

Über die Schaltflächen der Symbolleiste:

1. Klicken Sie in der Symbolleiste auf das jeweilige Kompositionsstrukturdiagramm-Symbol.



2. Klicken Sie in das Kompositionsstrukturdiagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen bestehender Elemente in das Kompositionsstrukturdiagramm

Die meisten Elemente, die in anderen Kompositionsstrukturdiagrammen vorkommen, können in ein bestehendes Kompositionsstrukturdiagramm eingefügt werden.

1. Suchen Sie das gewünschte Element auf dem Register "Modell-Struktur" (Sie können das Element über das Suchfunktionstextfeld oder mit Hilfe von Strg + F suchen).
2. Ziehen Sie das/die Element(e) in das Kompositionsstrukturdiagramm.

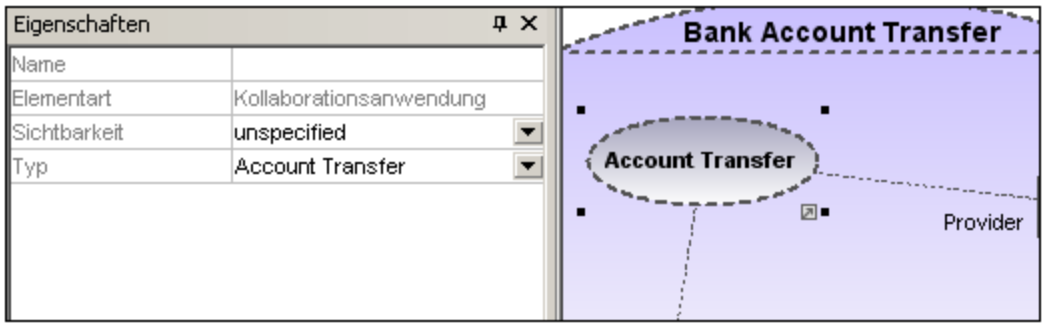
© 2016-2022 Altova GmbH

Kollaboration

Fügt ein Kollaborationselement ein. Dabei handelt es sich um eine Art von Classifier/Instanz, der/die mit anderen Instanzen kommuniziert, um das Verhalten des Systems zu erzeugen.

Kollaborationsanwendung

Fügt ein Kollaborationsanwendungselement ein. Dabei handelt es sich um eine bestimmte Kollaborationsanwendung, wobei bestimmte Klassen oder Instanzen die Rolle der Kollaboration spielen. Eine Kollaborationsanwendung wird als strichlierte Ellipse angezeigt, die den Namen der Instanz, einen Doppelpunkt und den Namen des Kollaborationstyps enthält.



Beim Erstellen von Abhängigkeiten zwischen Kollaborationsanwendungselementen muss das Feld "Typ" ausgefüllt sein, damit eine Rollenbindung erstellt werden kann und die Zielkollaboration muss mindestens einen Part/eine Rolle haben.

Part (Eigenschaft)

Fügt ein Part-Element ein, das eine Gruppe von einer oder mehreren Instanzen darstellt, die ein Classifier besitzt, der diese enthält. Ein Part kann zu Kollaborationen und Klassen hinzugefügt werden.

Port

Fügt ein Port-Element ein, das den Interaktionspunkt zwischen einem Classifier und seiner Umgebung definiert. Ein Port-Element kann zu einem Part-Elementen mit einem definierte Typ hinzugefügt werden.

Klasse

Fügt ein Klassenelement ein. Dabei handelt es sich um den eigentlichen Classifier, der in dieser bestimmten Verwendung der Kollaboration vorkommt.

Konnektor

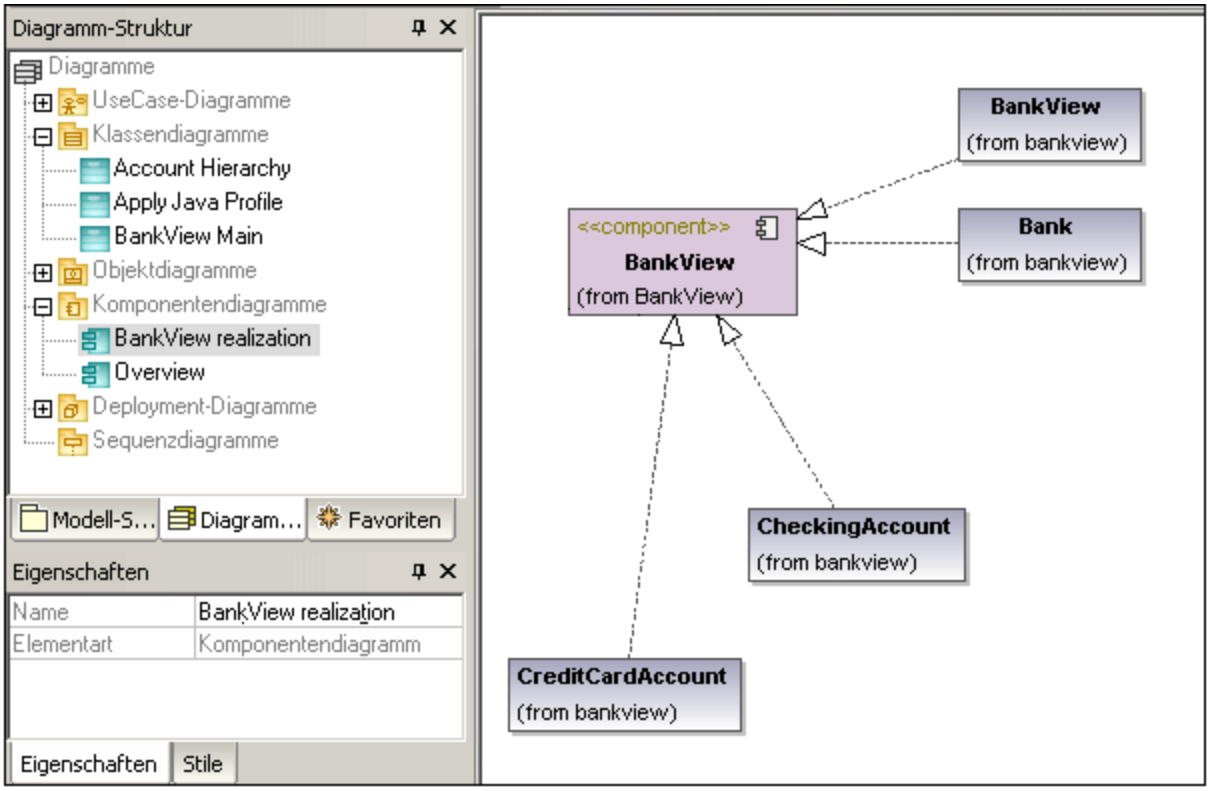
Fügt ein Konnektorelement hinzu, das verwendet werden kann, um zwei oder mehr Instanzen eines Teils oder Ports zu verbinden. Der Konnektor definiert die Beziehung zwischen den Objekten und identifiziert die Kommunikation zwischen den Rollen.

Abhängigkeit (Rollenbindung)

Fügt das Abhängigkeitselement ein. Dieses Element zeigt an, welches verbindbare Element des Classifiers oder der Operation welche Rolle in der Kollaboration spielt.

9.2.3 Komponentendiagramm

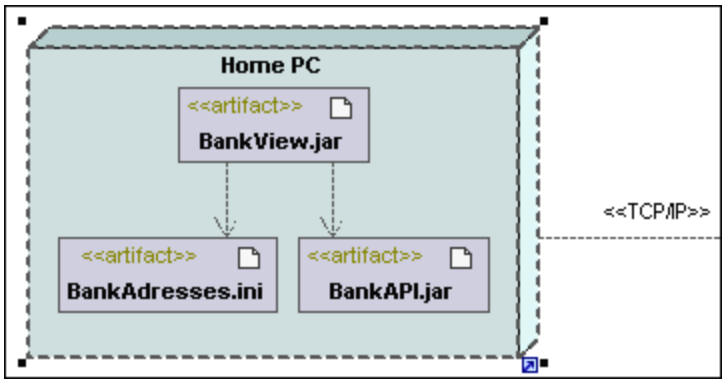
Eine Anleitung zum Hinzufügen von Komponentenelementen zum Diagramm finden Sie im Tutorial im Abschnitt Komponentendiagramme[54].



9.2.4 Deployment-Diagramm

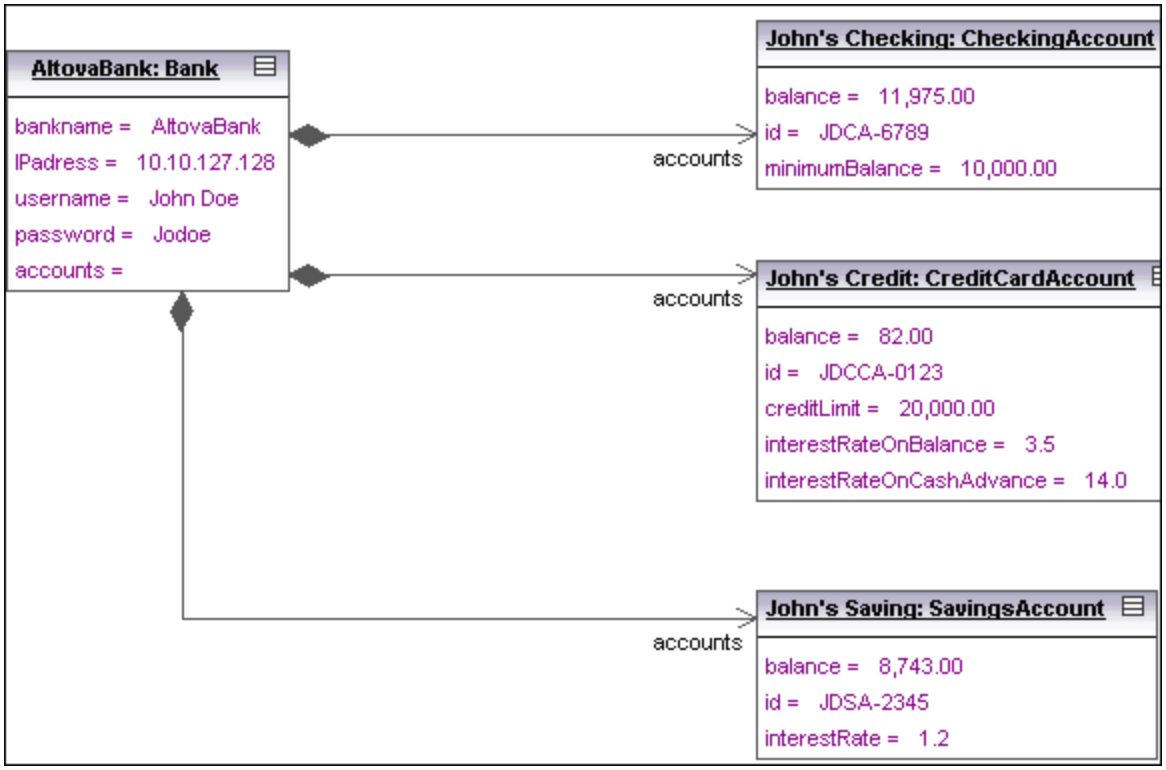
Eine Anleitung zum Hinzufügen von Knoten und Artefakten zum Diagramm finden Sie im Tutorial im Abschnitt Deployment-Diagramme[60].

© 2016-2022 Altova GmbH



9.2.5 Objektdiagramm

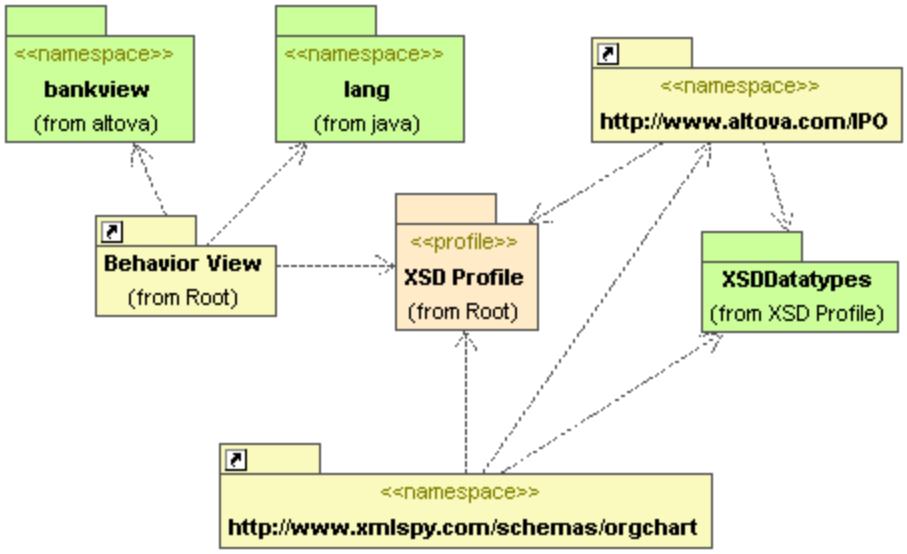
Eine Anleitung zum Hinzufügen neuer Objekte/Instanzen zum Diagramm finden Sie im Tutorial im Abschnitt Objektdiagramme[46].



9.2.6 Paketdiagramm

In Paketdiagrammen werden die Struktur von Paketen und ihren Elementen sowie die entsprechenden Namespaces angezeigt. Zusätzlich dazu können Sie in UModel einen Hyperlink erstellen und zum jeweiligen Inhalt des Pakets navigieren.

Pakete werden als Ordner dargestellt und können in jedem der UML-Diagramme verwendet werden. Sie werden jedoch hauptsächlich in Use Case- und Klassendiagrammen verwendet.



Automatische Generierung von Paketabhängigkeiten

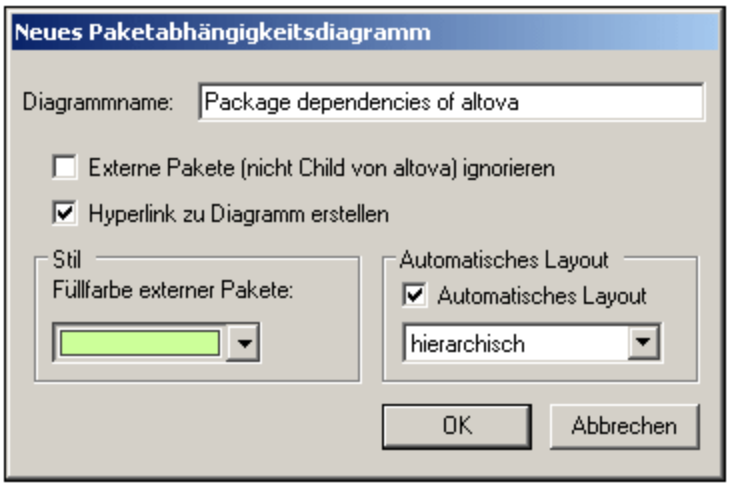
UModel bietet die Möglichkeit, ein Paketabhängigkeitsdiagramm für jedes beliebige in der Modell-Struktur bereits existierende Paket zu generieren.

Abhängigkeitslinks zwischen Paketen werden erstellt, wenn Referenzen zwischen den Modellierungselementen dieser Pakete vorhanden sind, d.h. wenn es Abhängigkeiten zwischen Klassen, abgeleitete Klassen oder Attributen gibt, die Typen haben, die in einem anderen Paket definiert sind.

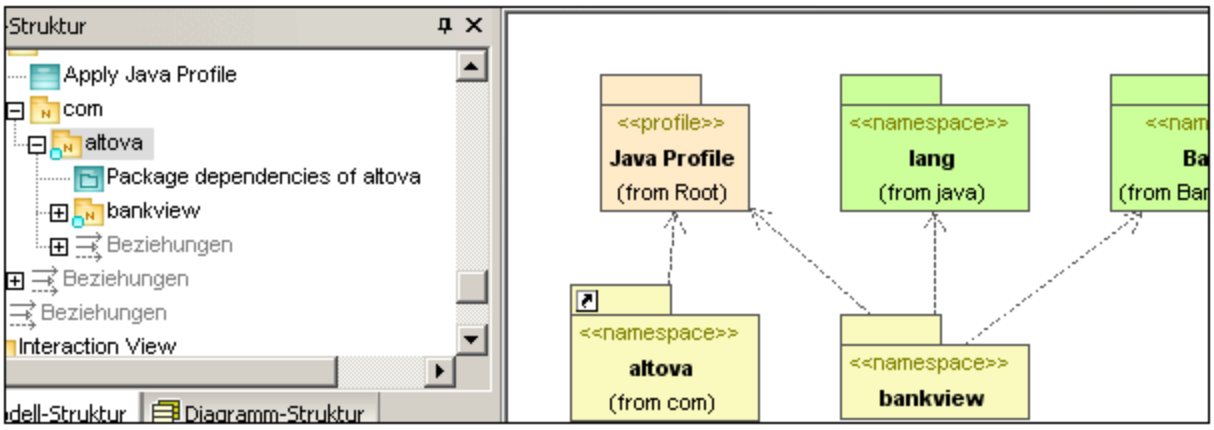
So generieren Sie ein Paketabhängigkeitsdiagramm:

1. Rechtsklicken Sie auf ein Paket in der Modell-Struktur, z.B. altova und wählen Sie den Befehl **In neuem Diagramm anzeigen | Paketabhängigkeiten...** .
   Daraufhin wird das Dialogfeld "Neues Paketabhängigkeitsdiagramm" geöffnet.

© 2016-2022 Altova GmbH



2. Wählen Sie die gewünschten Optionen aus und klicken Sie zur Bestätigung auf OK.



Es wird ein neues Diagramm generiert, in dem die Paketabhängigkeiten des Pakets "altova" angezeigt werden.

9.2.6.1 Einfügen von Paketdiagrammelementen

Verwendung der Symbolleisten-Schaltflächen

1. Klicken Sie in der Paketdiagramm-Symbolleiste auf das entsprechende Symbol.



2. Klicken Sie in das Diagramm, um das Element einzufügen. Um mehrere Elemente des ausgewählten Typs einzufügen, halten Sie die **Strg-Taste** gedrückt und klicken Sie in das Diagrammfenster.

Ziehen vorhandener Elemente in das Paketdiagramm

Sie können Elemente aus anderen Diagrammen, z.B. Pakete, in ein Paketdiagramm einfügen.

1. Suchen Sie das gewünschte Element auf dem Register "Modell-Struktur" (über das Suchfunktionstextfeld oder durch Drücken der Tasten Strg + F).
2. Ziehen Sie das Element/die Elemente in das Diagramm.

 Paket

Fügt das Paketelement in das Diagramm ein. Pakete dienen zum Gruppieren von Elementen und zur Bereitstellung eines Namespace für die gruppierten Elemente. Da es sich bei einem Paket um einen Namespace handelt, kann ein Paket einzelne Elemente oder alle Elemente anderer Pakete importieren. Pakete können auch mit anderen Paketen zusammengeführt werden.

 Profil

Fügt das Profil-Element ein, einen bestimmten Pakettyp, der auf andere Pakete angewendet werden kann.

Das Profile-Paket dient zum Erweitern des UML-Metamodells. Das primäre Erweiterungskonstrukt ist das Stereotyp, welches selbst Teil des Profils ist. Profile müssen immer in Beziehung zu einem Referenz-Metamodell wie z.B. UML stehen und können nicht alleine stehen.

 Abhängigkeit

Fügt das Abhängigkeits-Element ein, welches eine Bereitsteller/Client-Beziehung zwischen Modellierungselementen - in diesem Fall Paketen oder Profilen anzeigt.

 Paketimport

Fügt eine <<import>> Beziehung ein, welche anzeigt, dass die Elemente des inkludierten Pakets in das inkludierende Paket importiert werden. Der Namespace des inkludierenden Pakets erhält Zugriff auf den inkludierten Namespace; der Namespace des inkludierten Pakets ist nicht betroffen.

Anmerkung: Elemente, die in einem Paket als "privat" definiert sind, können nicht zusammengeführt oder importiert werden.

 Paketmerge

Fügt eine <<merge>> Beziehung ein, welche anzeigt, dass die Elemente des zusammengeführten (Quell)-Pakets einschließlich aller in das zusammengeführte (Quell)-Paket importierten Inhalte in das zusammenführende (Ziel)-Paket importiert werden.

Wenn dasselbe Element im Zielpaket bereits vorhanden ist, werden die Definitionen dieser Elemente um die Definitionen aus dem Zielpaket erweitert. Aktualisierte oder hinzugefügte Elemente werden durch eine Generalisierungsbeziehung zurück zum Quellpaket gekennzeichnet.
Anmerkung: Elemente, die in einem Paket als "privat" definiert sind, können nicht zusammengeführt oder importiert werden.

© 2016-2022 Altova GmbH



Profilzuweisung

Fügt eine Profilzuweisung ein, die anzeigt, welche Profile einem Paket zugewiesen wurden. Hierbei handelt es sich um eine Art des Paketimports, der festlegt, dass ein Profil auf ein Paket angewendet wird.

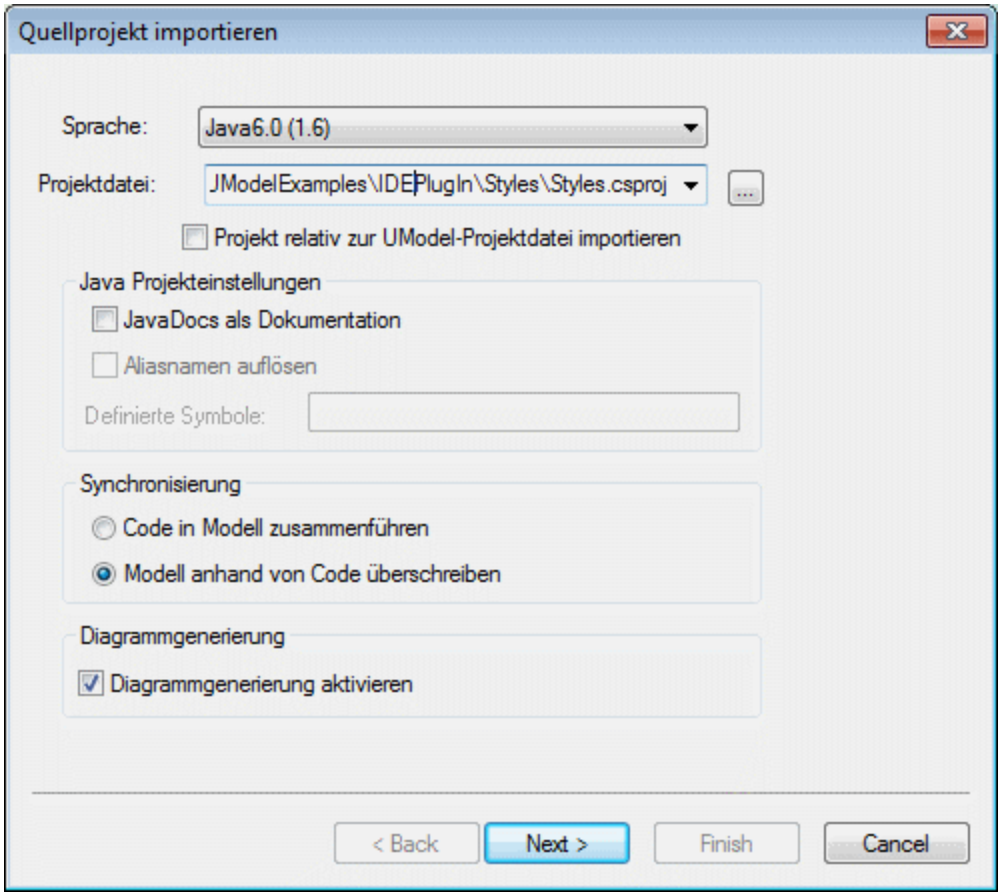
Das Profil erweitert das Paket, dem es zugewiesen wurde. Bei Zuweisung eines Profils mit Hilfe des Symbols "Profilzuweisung" stehen alle Stereotype, die Teil dieses Profils sind, auch dem Paket zur Verfügung.

Profilnamen werden als strichlierte Pfeile vom Paket zum zugewiesenen Profil mit dem Schlüsselwort <<apply>> angezeigt.

9.2.6.2 Generieren von Paketdiagrammen

Sie können bei Import von Quellcode oder Binärdateien in ein UModel-Projekt (siehe Importieren von Quellcode in Projekte 209 und Importieren von Java-, C#- und VB-Binärdateien 225) Paketdiagramme generieren. Nehmen Sie im Importassistenten folgende Einstellungen vor:

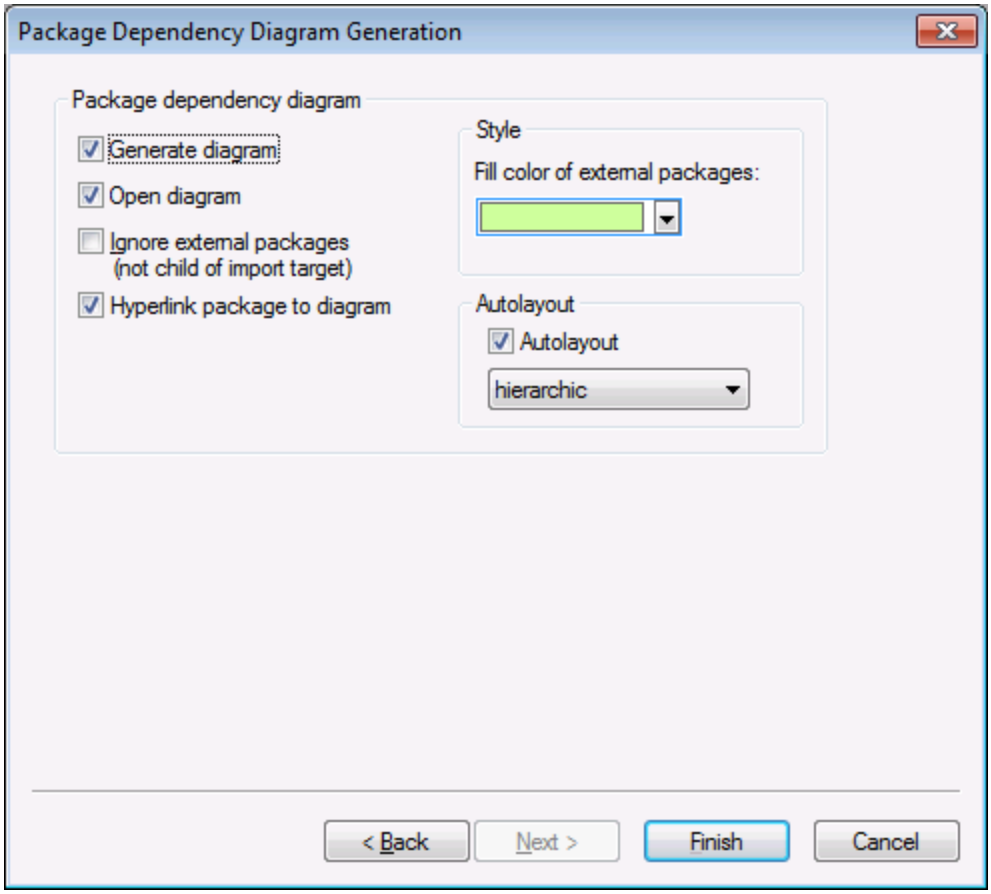
1) Im Dialogfeld "Quellprojekt importieren", "Binärtypen importieren" bzw. "Quellverzeichnis importieren" muss das Kontrollkästchen **Diagrammgenerierung aktivieren** aktiviert sein.



Dialogfeld "Quellprojekt importieren"

2) Im Dialogfeld "Generierung von Paketabhängigkeitsdiagrammen" muss die Option **Diagramm generieren** aktiviert sein.

© 2016-2022 Altova GmbH



Dialogfeld "Generierung von Paketabhängigkeitsdiagrammen"

Nach Fertigstellung des Imports stehen die generierten Klassendiagramme in der Diagramm-Struktur unter "Klassendiagramme" zur Verfügung.

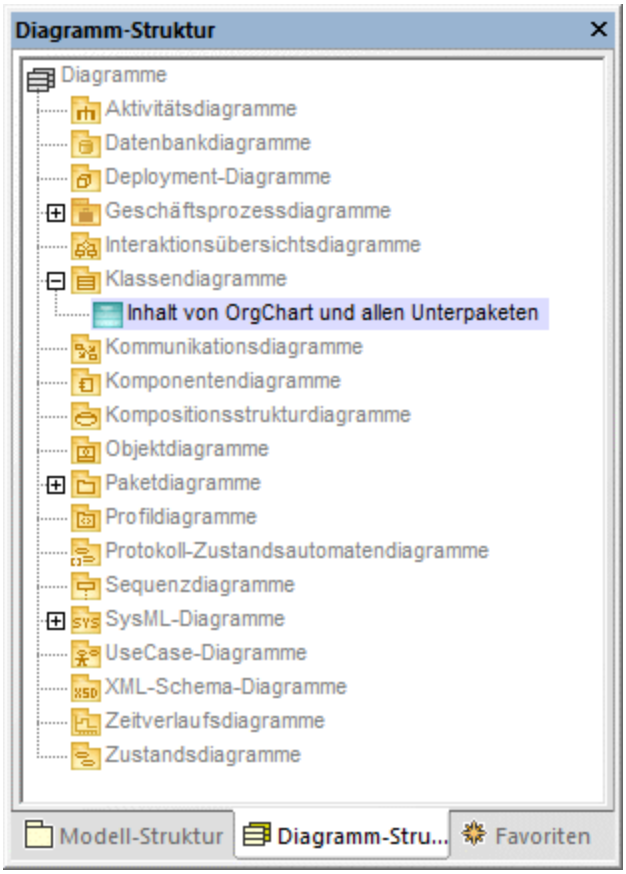


Diagramm-Struktur

9.2.7 Profildiagramm

Altova Website: UML-Profildiagramme

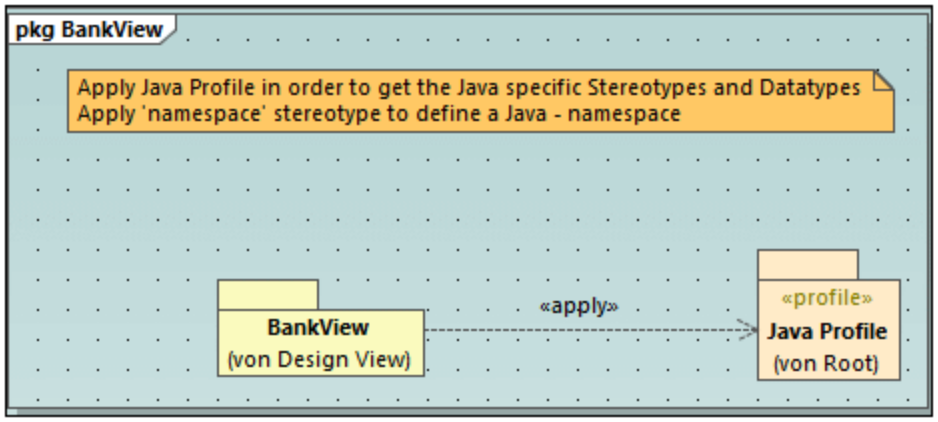
Mit Hilfe von Profildiagrammen kann UML für eine bestimmte Plattform oder Domain erweitert werden. Im Gegensatz zu einem Paket befindet sich ein Profil im Metamodell und besteht aus Metabausteinen, die etwas erweitern oder einschränken. Dies lässt sich mit Hilfe der folgenden in ein Profil inkludierten Erweiterungsmechanismen bewerkstelligen: Stereotype, Eigenschaftswerte und Constraints.

Im Profildiagramm können Sie in UModel Ihre eigenen in einem benutzerdefinierten Profil gebündelten Stereotype, Eigenschaftswerte und Constraints erstellen. Mit Hilfe von Profilen können Sie UML erweitern oder an Ihre spezifische Domain adaptieren oder das Aussehen von Elementen in Ihren Modellierungsprojekten anpassen. So können Sie z.B. benutzerdefinierte Stile definieren oder benutzerdefinierte Symbole für UML-Elemente wie Klassen, Schnittstellen, usw. hinzufügen.

Über das Profildiagramm können Sie ein *Profil auf ein Paket anwenden*. Im unten gezeigten Profildiagramm sehen Sie z.B. eine **Profilzuweisungsbeziehung** zwischen dem Paket **BankView** und dem vordefinierten Java-Profil in UModel. Sie finden dieses Diagramm im folgenden Beispielprojekt: **C:**

© 2016-2022 Altova GmbH

\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\BankView_Java.ump unter dem Namen "Apply Java Profile".



Profildiagramm

Gemäß dem angewendeten Java-Profil muss jede Klasse oder Schnittstelle, die Teil des **BankView**-Pakets ist (oder in Zukunft zu diesem Paket hinzugefügt wird) wie eine Java-Klasse oder -Schnittstelle aussehen und alle ihre Mitglieder müssen das sprachspezifische Verhalten aufweisen. Beispiel:

- Alle im Profil vorhandenen Java-Datentypen stehen bei der Erstellung einer Klasse in einem Klassendiagramm über eine Dropdown-Liste zur Auswahl zu Verfügung, siehe auch Klassendiagramme 31.
- Alle im Profil definierten Java-spezifischen Stereotype wie z.B. `«annotations»`, `«final»`, `«static»`, `«strictfp»`, usw., werden als Eigenschaften im Fenster "Eigenschaften" angezeigt, wenn Sie ein Element auswählen.

In diesem Kapitel wird beschrieben, wie Sie UModel-Projekte mit Hilfe von benutzerdefinierten Profilen und Stereotypen erweitern können. Informationen zur Verwendung der vordefinierten UModel-Profile finden Sie unter Anwenden von UModel-Profilen 170 und Stereotype und Eigenschaftswerte 155.

9.2.7.1 Erstellen und Anwenden von benutzerdefinierten Profilen

In der Anleitung unten wird beschrieben, wie Sie ein benutzerdefiniertes UModel-Profil erstellen und auf ein Paket anwenden. Dies ist normalerweise erforderlich, wenn Sie Stereotype erstellen und anwenden müssen, die nicht wie die UModel-Standardprofile bereits vordefiniert sind. Informationen zur Anwendung der UModel-Standardprofile finden Sie unter Anwenden von UModel-Profilen 170.

So erstellen Sie ein benutzerdefiniertes Profil:

1. Klicken Sie mit der rechten Maustaste auf das Paket, für das Sie das neue Profil erstellen möchten (z.B. "Root") und wählen Sie im Kontextmenü den Befehl **Neues Element | Profil**.
2. Erstellen Sie alle Elemente, die Teil dieses Profils sein sollen, wie z.B. Stereotype, Datentypen, usw. Sie können dies entweder im Fenster "Modell-Struktur" oder über ein Profildiagramm tun. Um z.B. ein neues Stereotyp im Modell zu erstellen, klicken Sie mit der rechten Maustaste auf das Profil und wählen Sie im Kontextmenü den Befehl **Neues Element | Stereotyp**. Siehe auch Erstellen von Stereotypen 479.

3. Erstellen Sie optional ein Profildiagramm (Klicken Sie mit der rechten Maustaste auf das Profil und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Profildiagramm**). Um die gewünschten Elemente zum Diagramm hinzuzufügen, verwenden Sie die UModel-Standardmenübefehle und Symbolleisten, siehe Anleitung zur Modellierung von... 111.

> Wenn Sie das Profil anhand eines Profildiagramms erstellen möchten, stellen Sie sicher, dass das Diagramm im Eigentum eines Profils (unter einem Profil erstellt wurde) oder eines Pakets innerhalb eines Profils ist.

Wenn Sie das Profil in mehreren UModel-Projekten verwenden möchten, gehen Sie folgendermaßen vor:

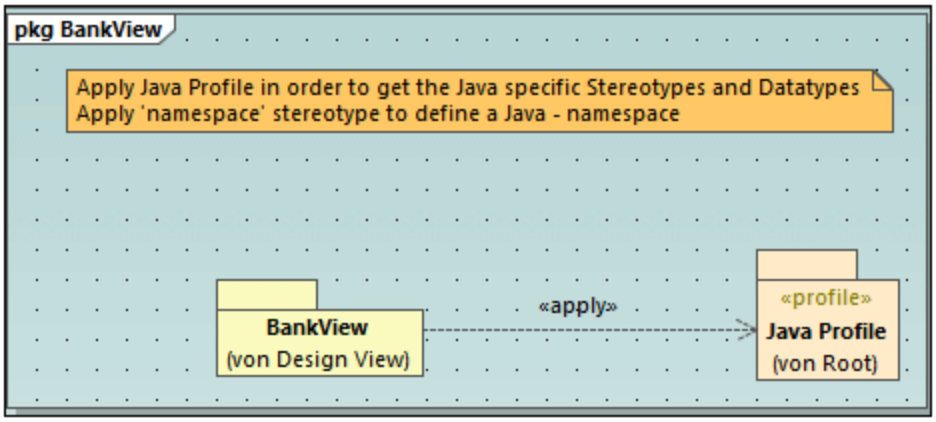
1. Geben Sie alle Pakete, die Sie wiederverwenden möchten, frei. (Rechtsklick auf das Paket oder Profil selbst und Auswahl des Kontextmenübefehls **Unterprojekt | Paket freigeben**.)
2. Speichern Sie das Projekt in einem Verzeichnis, über das Sie es später als Unterprojekt inkludieren können, siehe Inkludieren von Unterprojekten 174.

Sie haben bisher ein Profil erstellt, es aber noch nicht zu einem Paket hinzugefügt (oder darauf angewendet). Durch Anwendung eines Profils auf ein Paket stellen Sie alle Erweiterungsmechanismen dieses Profils (wie z.B. Stereotype, Datentypen, usw.) für Elemente des Pakets zur Verfügung.

So wenden Sie ein benutzerdefiniertes Profil auf ein Paket an:

1. Erstellen Sie ein neues UModel-Projekt oder öffnen Sie ein bestehendes.
2. Wählen Sie eine der folgenden Methoden:
   a. Erstellen Sie Ihr benutzerdefiniertes Profil im vorhandenen Projekt, wie oben gezeigt.
   b. Inkludieren Sie mit dem Menübefehl **Projekt | Unterprojekt inkludieren** ein benutzerdefiniertes Profil aus einem vorhanden Projekt. Beachten Sie, dass entweder das gesamte Profil oder seine Pakete darunter freigegeben werden müssen, damit sie wiederverwendet werden können, siehe Freigeben von Paketen und Diagrammen 176 .
3. Klicken Sie mit der rechten Maustaste auf das Profil und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Profildiagramm**.
4. Fügen Sie ein oder mehrere Pakete und das benutzerdefinierte Profil zum Diagramm hinzu.
5. Ziehen Sie eine **Profilzuweisungsbeziehung**  vom Paket zum Profil. So sehen Sie etwa im Profildiagramm unten eine **Profilzuweisungsbeziehung** zwischen dem Paket **BankView** und dem vordefinierten Java-Profil in UModel. Profilzuweisungen werden, wie unten gezeigt, zusammen mit dem Schlüsselwort `<<apply>>` als gestrichelte Pfeile vom Paket zum angewendeten Profil dargestellt.

 © 2016-2022 Altova GmbH



9.2.7.2 Erstellen von Stereotypen

Wenn Sie Projekte mit Hilfe eines der vordefinierten UModel-Profile (wie z.B. C#, Java, VB.NET, XML Schema, Datenbank, usw.) modellieren, ist es normalerweise nicht notwendig, benutzerdefinierte Stereotype zu erstellen. Sie können stattdessen einfach die vorhandenen Stereotype auf die Elemente Ihres Modells anwenden, wie unter Anwenden von Stereotypen[157] beschrieben.

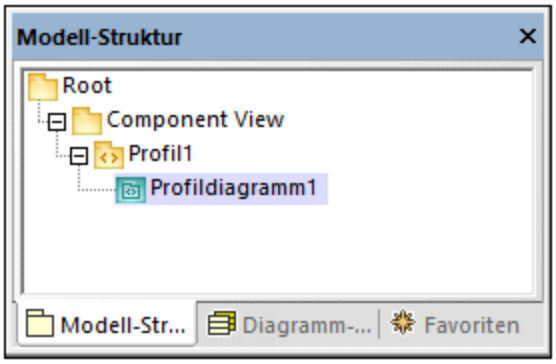
Wenn Sie jedoch benutzerdefinierte Symbole zu Elementen hinzufügen möchten oder deren Aussehen auf Basis des angewendeten Stereotyps anpassen möchten, können Sie dies mit Hilfe von benutzerdefinierten Stereotypen tun. Beachten Sie dabei die folgenden Voraussetzungen:

- Stereotype müssen als Owner ein Profil oder ein Paket innerhalb eines Profils haben. Um daher ein Stereotyp erstellen zu können, müssen Sie ein Profil (oder ein Paket innerhalb eines vorhandenen Profils) erstellen.
- Nachdem Sie das Profil erstellt haben, müssen Sie es auf das Paket, in dem Sie die benutzerdefinierten Stereotype verwenden möchten, anwenden, wie unter Erstellen und Anwenden von benutzerdefinierten Profilen[477] beschrieben.

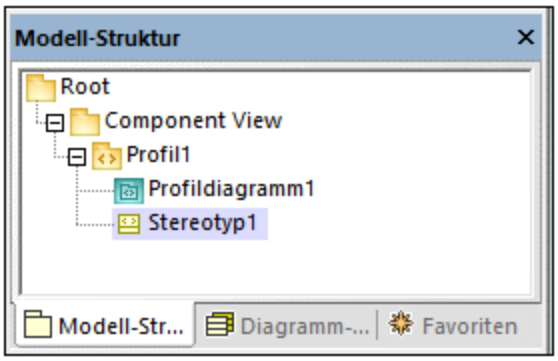
Nachdem Sie ein Profil erstellt haben, können Sie Stereotype dazu hinzufügen. Sie können dies entweder direkt im Fenster "Modell-Struktur" oder über ein Profildiagramm tun. Wenn Sie Stereotype über ein Profildiagramm erstellen möchten, stellen Sie sicher, dass das Diagramm als Owner ein Profil oder ein Paket innerhalb eines Profils hat, wie unten gezeigt.

So erstellen Sie ein Stereotyp:

1. Erstellen Sie zuerst ein Profil, falls Sie dies noch nicht getan haben, siehe Erstellen und Anwenden von benutzerdefinierten Profilen[477].
2. Klicken Sie optional mit der rechten Maustaste auf das Profil und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Profildiagramm**. Dadurch wird unter dem aktuellen Profil ein neues Profil erstellt - so sehen sie alle Stereotype, Datentypen und anderen Elemente, die sie später zu diesem Profil hinzufügen, an einer Stelle.



3. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf das Profil und wählen Sie im Kontextmenü den Befehl **Neues Element | Stereotyp**.



4. Definieren Sie optional im Fenster "Eigenschaften" die Eigenschaften des Stereotyps. Wenn Sie z.B. die **Metaklasse** des Stereotyps auf "Klasse" setzen, gilt das Stereotyp nur für Klassen. Ebenso können Sie ein benutzerdefiniertes Symbol für das Stereotyp definieren, indem Sie neben **Symboldateiname** auf die **Auslassungspunkte** ... klicken.



Anmerkungen

- Wenn der Pfad des Bilds relativ ist, muss er relativ zum UModel-Projektordner sein.

© 2016-2022 Altova GmbH

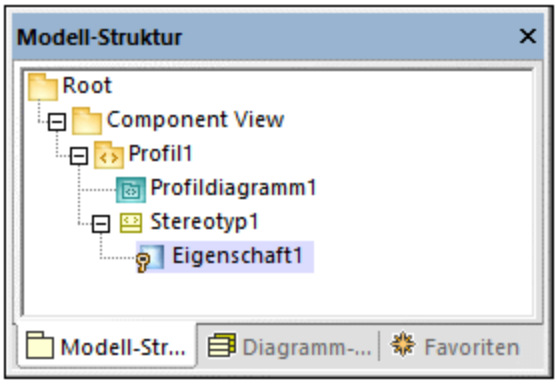
- Um benutzerdefinierte Symbole mit transparentem Hintergrund zu verwenden, setzen Sie die Hintergrundfarbe auf den RGB-Wert 82,82,82.
- Um Stereotype für Assoziationsbeziehungen anzuzeigen, setzen Sie im Fenster "Stile" die Eigenschaft **MemberEnd-Stereotype anzeigen** auf "true".

Hinzufügen von Stereotyp-Attributen (Eigenschaften)

Das oben erstellte Stereotyp ist sehr einfach gehalten und es sind damit keine Attribute (Eigenschaften) verknüpft. Es können jedoch auch Eigenschaften zu einem Stereotyp hinzugefügt werden. Solche Eigenschaften werden zu Eigenschaftswerten, wenn dieses Stereotyp in Zukunft auf ein Element angewendet wird.

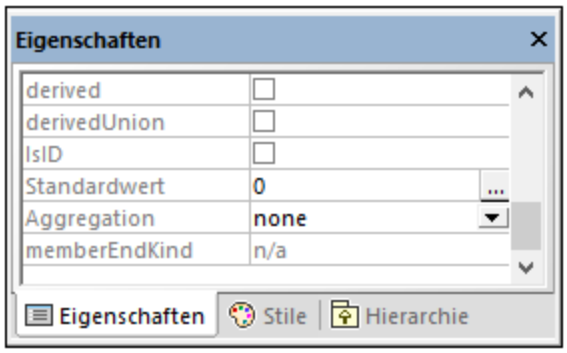
So fügen Sie Attribute (Eigenschaften) zu einem Stereotyp hinzu:

1. Klicken Sie im Fenster "Modell-Struktur" oder im Diagramm auf das Stereotyp.
2. Wählen Sie eine der folgenden Methoden
   a. Klicken Sie mit der rechten Maustaste auf das Stereotyp und wählen Sie im Kontextmenü den Befehl **Neu | Eigenschaft**.
   b. Drücken Sie **F7**.



Sie können über das Fenster "Eigenschaften" den Datentyp jeder Eigenschaft definieren, indem Sie einen Wert aus der Liste **Typ** auswählen. Zur Auswahl steht jeder Datentyp, der zuvor im selben Profil wie das Stereotyp definiert wurde. Wenn das Profil noch keine Datentypen enthält, können Sie einen definieren, indem Sie mit der rechten Maustaste auf das Profildiagramm klicken und im Kontextmenü den Befehl **Neu | Datentyp** auswählen.

Um den Standardwert einer Eigenschaft zu definieren, geben Sie diesen Wert im Fenster "Eigenschaften" in das Feld **Standardwert** ein. Die unten gezeigte Stereotypeigenschaft hat z.B. den Standardwert "0".



Der Datentyp eines Stereotypattributs (Eigenschaft) kann auch eine Enumeration sein, siehe Beispiel: Erstellen und Anwenden von Stereotypen 482.

9.2.7.3 Beispiel: Erstellen und Anwenden von Stereotypen

In diesem Beispiel wird Schritt für Schritt beschrieben, wie ein Stereotyp erstellt wird. Es wird gezeigt, wie Sie die folgenden Zielsetzungen erreichen:

- Erstellung eines Stereotyps
- Erstellung von Stereotypattributen (Eigenschaften), die bei Anwendung auf ein Element zu Eigenschaftswerten werden
- Definition eines Stereotypattributs als Enumeration
- Definition eines Standardwerts für ein Stereotypattribut
- Anwendung des Stereotyps auf Elemente im Modell.

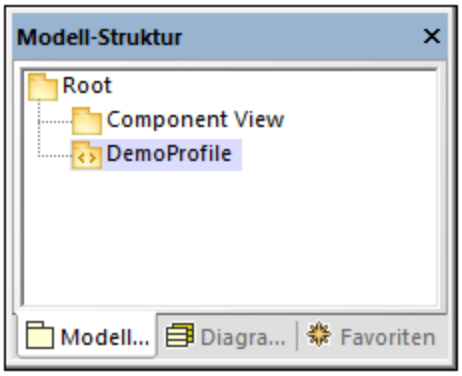
Die Beispielprojektdatei dazu finden Sie unter dem Namen **StereotypesDemo.ump** unter dem folgenden Pfad: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial**. Mit der nachstehenden Anleitung können Sie ein ähnliches Projekt erstellen.

Erstellung eines neuen Profils

Wie oben erwähnt, muss ein Stereotyp als Owner ein Profil haben. Wir wollen daher zuerst ein Profil erstellen.

1. Erstellen Sie ein neues UModel-Projekt.
2. Klicken Sie mit der rechten Maustaste auf das Root-Paket und fügen Sie durch Auswahl des Kontextmenüs **Neues Element | Profil** ein neues Profil hinzu.
3. Benennen Sie das neue Profil in "DemoProfile" um.

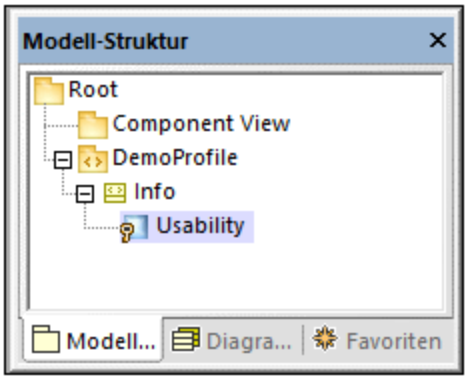
© 2016-2022 Altova GmbH



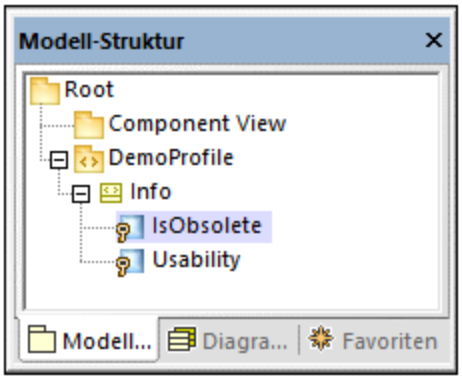
Erstellen eines Stereotyps

Wir wollen in diesem Tutorial ein Stereotyp mit zwei Attributen erstellen: "Usability" und "IsObsolete". Das Attribut "IsObsolete" wird als Enumeration definiert. Die Enumeration besteht aus den zwei Werten "Yes" und "No", wobei "No" der Standardwert ist.

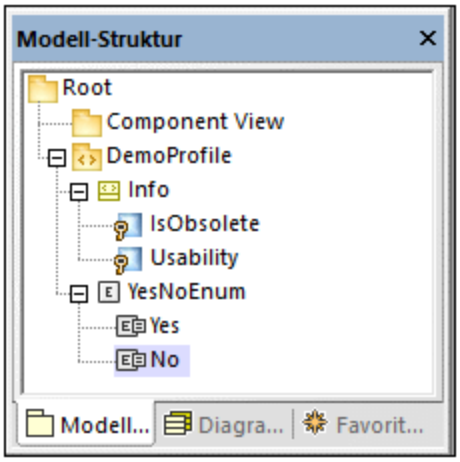
1. Klicken Sie mit der rechten Maustaste auf das Profil und wählen Sie im Kontextmenü den Befehl **Neues Element | Stereotyp** aus. Daraufhin wird ein neues Stereotyp zum Profil hinzugefügt.
2. Benennen Sie das neue Stereotyp in "Info" um.
3. Klicken Sie mit der rechten Maustaste auf das Stereotyp und wählen Sie im Kontextmenü den Befehl **Neues Element | Eigenschaft**. Daraufhin wird eine neue Eigenschaft hinzugefügt.
4. Benennen Sie die neue Eigenschaft in "Usability" um.



5. Wiederholen Sie die obigen Schritte, um eine neue Eigenschaft namens "IsObsolete" zu erstellen.

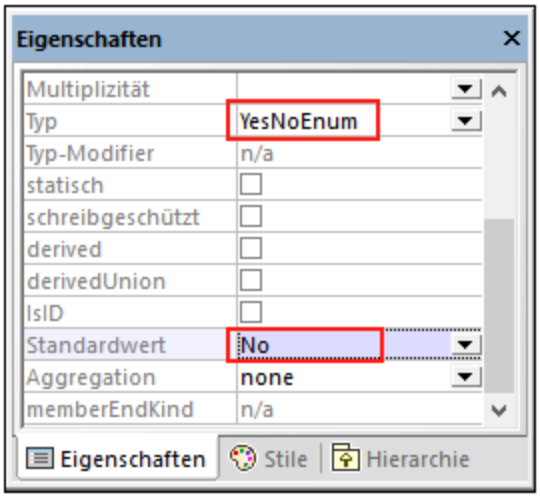


6. Klicken Sie mit der rechten Maustaste auf das "DemoProfile" und wählen Sie im Kontextmenü **Neues Element | Enumeration.** Benennen Sie die Eumeration in "YesNoEnum" um.
7. Klicken Sie mit der rechten Maustaste auf die Enumeration und wählen Sie im Kontextmenü **Neues Element | EnumerationLiteral.** Benennen Sie die Eumeration in "Yes" um.
8. Wiederholen Sie den obigen Schritt und erstellen Sie ein Enumerationsliteral namens "No".



9. Klicken Sie auf die Eigenschaft "IsObsolete" und ändern Sie ihren Typ in `YesNoEnum`. Setzen Sie außerdem die Eigenschaft **Standardwert** auf "No"

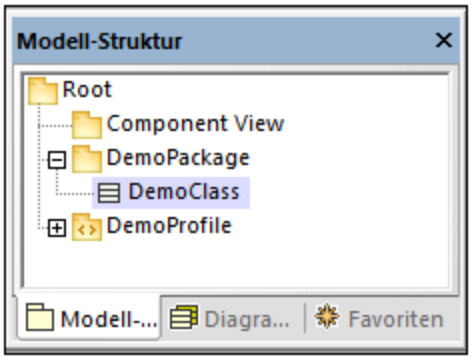
© 2016-2022 Altova GmbH



Erstellen eines neues Pakets

Um zu zeigen, wie das benutzerdefinierte Stereotyp verwendet werden kann, wollen wir ein einfaches Paket, das nur eine Klasse enthält, erstellen.

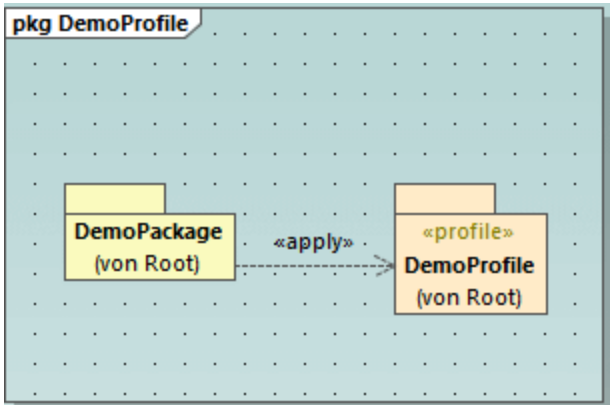
1. Klicken Sie mit der rechten Maustaste auf das Root-Paket und fügen Sie durch Auswahl des Kontextmenübefehls **Neues Element | Paket** ein neues Paket hinzu.
2. Benennen Sie das neue Paket in "DemoPackage" um.
3. Fügen Sie eine Klasse zum Paket hinzu (in diesem Beispiel "DemoClass").



Anwenden des Profils auf ein Paket

Wie in Schritt 1 erwähnt, wurde das Stereotyp innerhalb eines Profils erstellt. In diesem Schritt wenden wir das Profil auf ein Paket an, damit das Stereotyp für das Paket "sichtbar" wird.

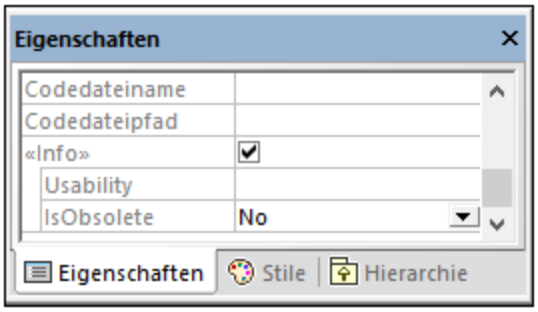
1. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf das "DemoProfile" und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Profildiagramm**.
2. Ziehen Sie sowohl das Paket "DemoPackage" als auch das Profil "DemoProfile" aus dem Fenster "Modell-Struktur" in das Diagramm.
3. Klicken Sie auf die Symbolleistenschaltfläche **Profilzuweisung** und ziehen Sie eine **Profilzuweisungsbeziehung** vom Paket auf das Profil.



Anwenden des Stereotyps auf Klassen

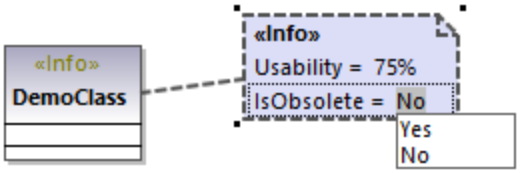
Sie können das Stereotyp nun auf eine Klasse anwenden.

1. Klicken Sie mit der rechten Maustaste auf das "DemoPackage" und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Klassendiagramm**.
2. Ziehen Sie die Klasse "DemoClass" in das Diagramm.
3. Klicken Sie auf die Klasse und aktivieren Sie im Fenster "Eigenschaften" das Stereotyp `«Info»`. Beachten Sie, dass für die Eigenschaft "IsObsolete" der Standardwert ausgefüllt wurde.



4. Geben Sie einen Wert für die Eigenschaft "Usability" ein (in diesem Beispiel "75%").

Die Klasse im Diagramm hat nun einen Eigenschaftswerte-Abschnitt, in dem die Stereotypattribute und deren Werte angezeigt werden. Sie können diese Werte entweder über das Fenster "Eigenschaften" oder direkt über das Diagramm ändern.



© 2016-2022 Altova GmbH

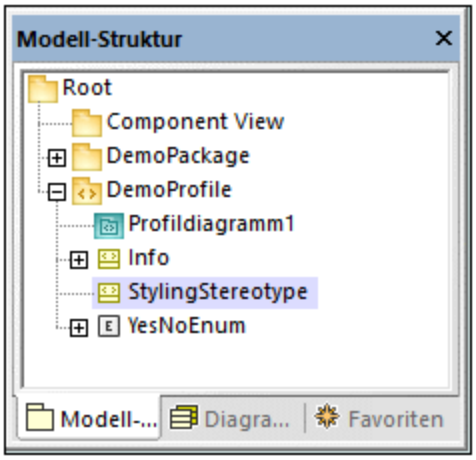
9.2.7.4 Beispiel: Anpassen von Symbolen und Stilen

In diesem Beispiel wird gezeigt, wie Sie das Aussehen einer Klasse in UModel mit Hilfe von Stereotypen anpassen. Nachdem Sie dieses Beispiel fertig gestellt haben, erfahren Sie, wie Sie benutzerdefinierte Elemente zu Symbolen hinzufügen und den Stil aller Elemente, für die dasselbe Stereotyp verwendet wird, ändern.

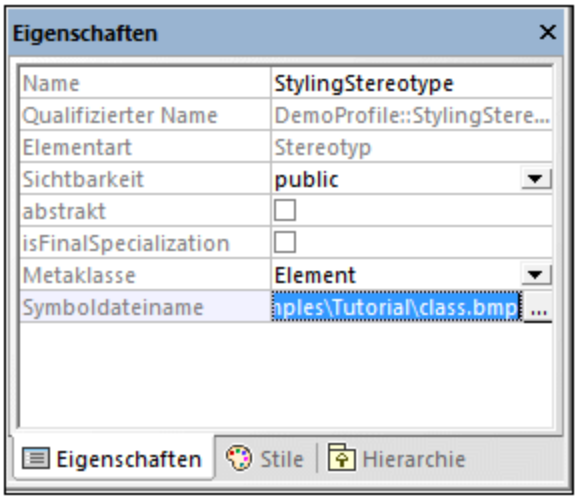
Die Klasse, die in diesem Beispiel angepasst wird, befindet sich im Projekt **StereotypesDemo.ump,** das unter dem folgenden Pfad zur Verfügung steht: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial**. Dies ist ein einfaches Demo-Projekt, das ein benutzerdefiniertes Profil enthält, unter dem wir das Stereotyp erstellen werden. Ein Beispiel dafür, wie Sie Profile und Stereotype von Grund auf neu erstellen, finden Sie unter Beispiel: Erstellen und Anwenden von Stereotypen 482.

Erstellen wir zuerst das Stereotyp, das wir für die Stile verwenden:

1. Öffnen Sie das Projekt **StereotypesDemo.ump**.
2. Klicken Sie in der Modell-Struktur mit der rechten Maustaste auf das Profil "DemoProfile" und wählen Sie im Kontextmenü den Befehl **Neues Element | Stereotyp**.
3. Benennen Sie das Stereotyp in "StylingStereotype" um.



Um ein benutzerdefiniertes Bild zum Stereotyp hinzuzufügen, klicken Sie auf das Stereotyp und anschließend im Fenster Eigenschaften" auf die **Auslassungspunkte** ... neben der Eigenschaft **Symboldateiname**. Wählen Sie das folgende Beispielbild aus: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial\class.bmp**.

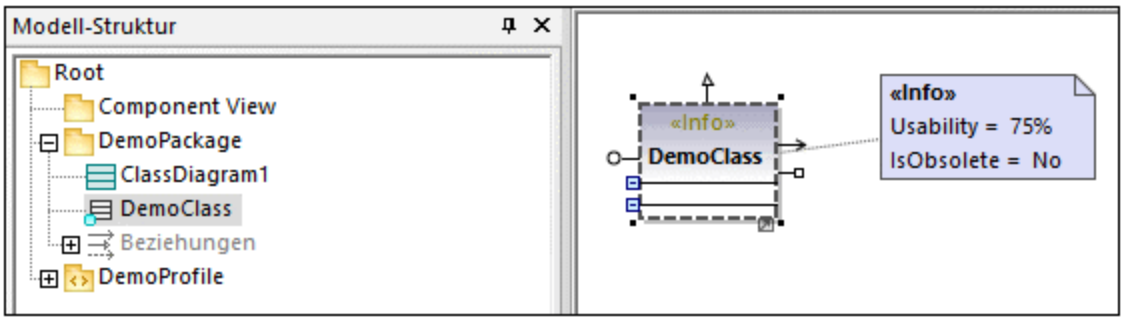


Klicken Sie als nächstes auf das Register **Stile** des Fensters **Eigenschaften**. Wählen Sie in der Liste ganz oben den Eintrag **Stile von Elementen mit diesem Stereotyp** und ändern Sie die Eigenschaft **Schriftgröße Titel** in "16".



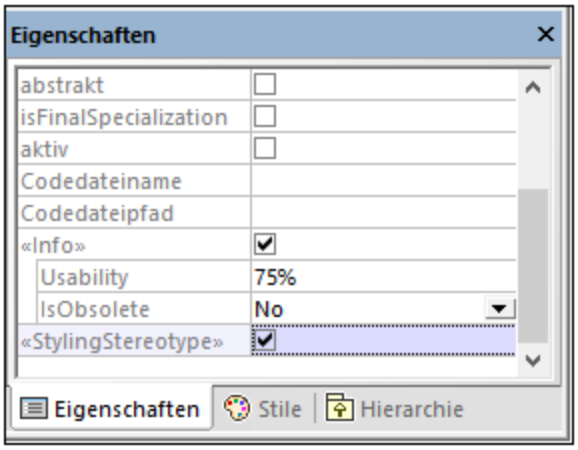
Wenden Sie das Stereotyp schließlich auf eine Klasse an.

1. Öffnen Sie das Klassendiagramm "ClassDiagram1". Sie finden dieses Diagramm im Fenster "Modell-Struktur" unter dem "DemoPackage".

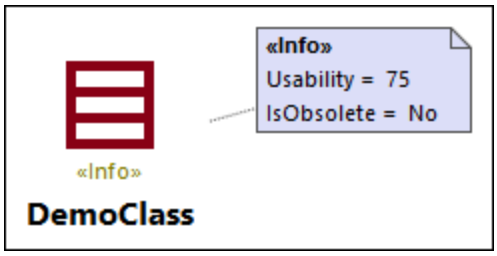


2. Klicken Sie auf die Klasse "DemoClass" und aktivieren Sie im Fenster "Eigenschaften" das Kontrollkästchen **«StylingStereotype»**.

© 2016-2022 Altova GmbH

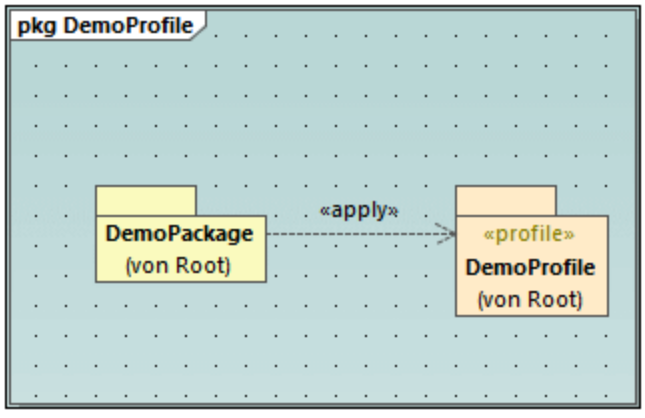


Das Aussehen der Klasse im Diagramm ändert sich nun gemäß dem angewendeten Stereotyp:



Anmerkungen

Das Demo-Projekt enthält das Profildiagramm "ProfileDiagram1". Beachten, Sie dass das "DemoProfile" mit einer Profilzuweisungsbeziehung auf das "DemoPackage" angewendet wurde. Dadurch steht das Stereotyp für das Paket zur Verfügung, siehe auch Erstellen und Anwenden von benutzerdefinierten Profilen 477.



Sie wissen nun, wie man das Aussehen von Elementen mit Hilfe von Stereotypen ändert. Auf dieselbe Art können Sie auch in anderen Projekten arbeiten. Denken Sie allerdings daran, dass das Profil, in dem Sie das Stereotyp erstellen, auf das Zielpaket angewendet werden muss, wie oben gezeigt.

9.3 Zusätzliche Diagramme

Die **UModel Enterprise Edition** unterstützt die folgenden zusätzlichen Diagramme:

- XML-Schema-Diagramme 490
- Business Process Modeling Notation
- SysML-Diagramme
- Datenbankdiagramme 555

9.3.1 XML-Schema-Diagramme

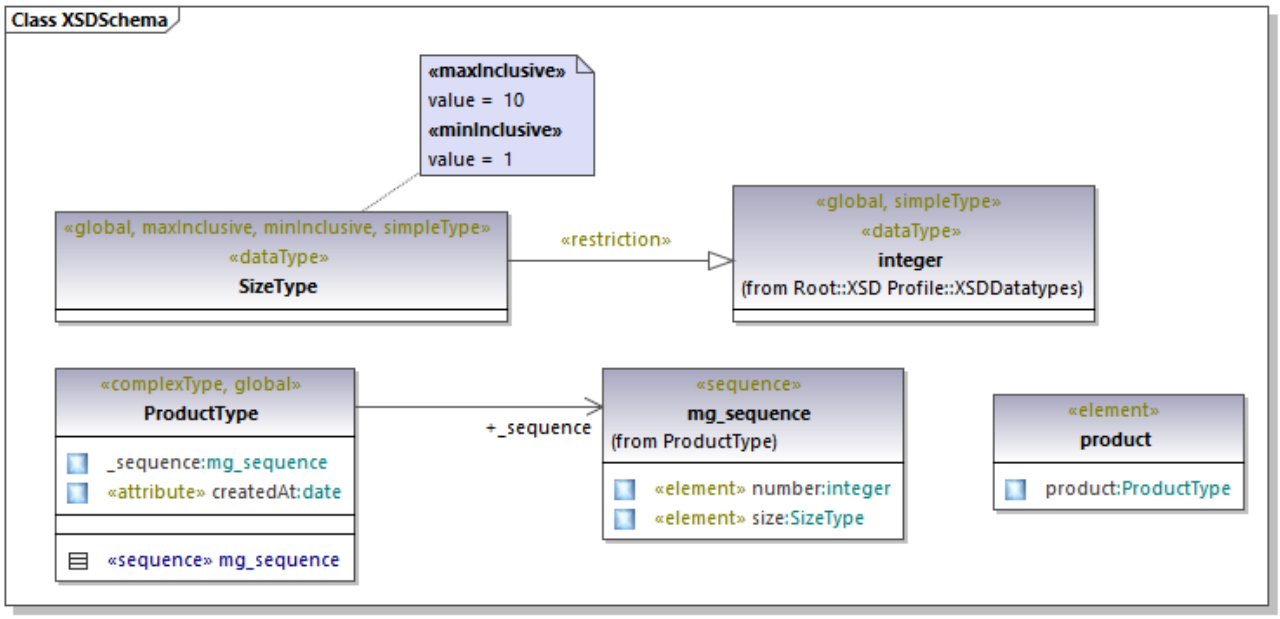
Altova Website: XML-Schemas in UML

UModel unterstützt den Import und die Generierung von W3C XML Schemas sowie deren Forward und Reverse Engineering. Im Fall von XML-Schemas bedeutet Forward und Reverse Engineering, dass Sie ein Schema (oder mehrere Schemas aus einem Verzeichnis) in UModel importieren, das Modell anzeigen oder ändern und die Änderungen wieder in die Schema-Datei schreiben können. Wenn Sie Daten aus dem Modell in einer Schema-Datei synchronisieren, wird die Schema-Datei immer durch das Modell überschrieben.

Anmerkung: Das XML-Schema muss gültig sein, bevor es in UModel importiert werden kann. XML-Schemas werden nicht validiert, wenn Sie diese in UModel erstellen oder importieren oder wenn Sie eine Syntaxüberprüfung am Projekt durchführen. UModel überprüft jedoch beim Import, ob das XML-Schema wohlgeformt ist.

In XML-Schema-Diagrammen werden Schemakomponenten in UML-Notation dargestellt. So werden etwa simpleTypes in UModel als Datentypen mit dem Stereotyp «simpleType» angezeigt. ComplexTypes werden als Klassen mit dem Stereotyp «complexType» angezeigt. Verschiedene Schemainformationen werden als Eigenschaftswerte 156 ,dargestellt, während Schema-Annotationen als Kommentare ausgewiesen werden. Unter XML-Schema-Entsprechungen 294 finden Sie eine Tabelle, in der aufgelistet ist, wie alle XML-Schema-Komponenten UModel-Elementen zugeordnet werden.

© 2016-2022 Altova GmbH



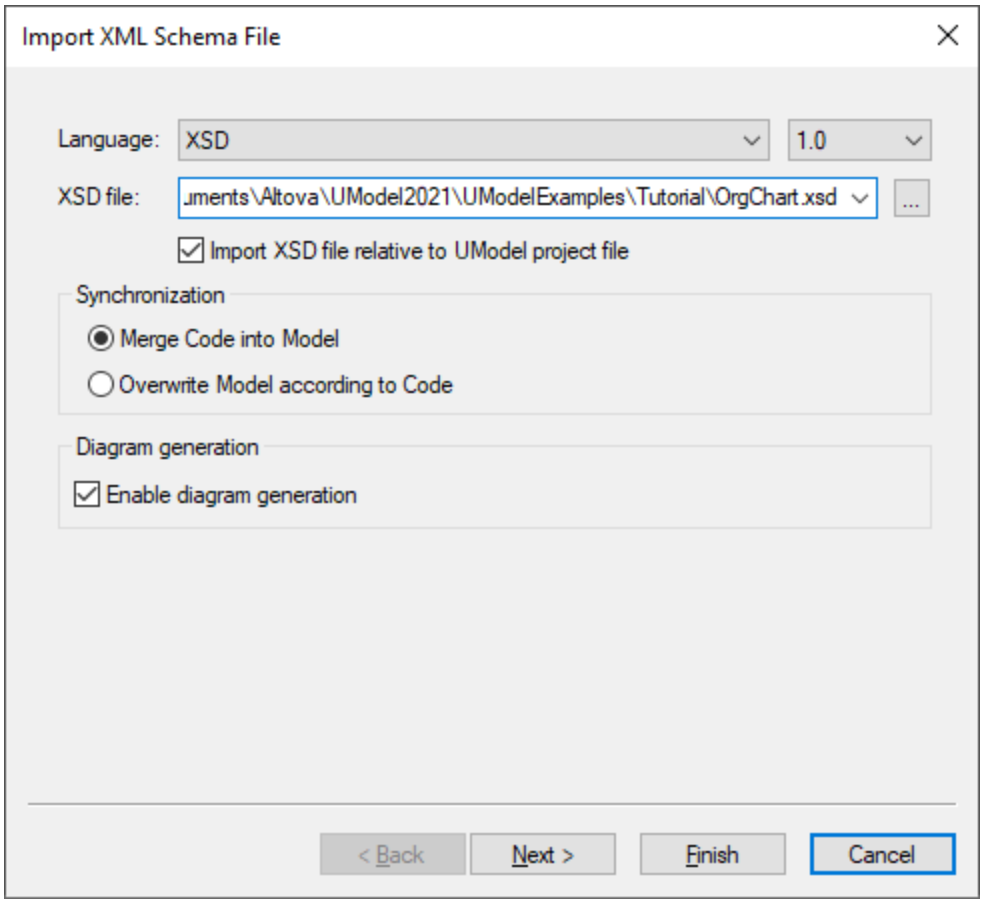
Beispiel für ein XML-Schema-Diagramm

9.3.1.1 Importieren von XML-Schemas

Sie können entweder eine einzelne Schema-Datei oder alle Schemas aus einem Verzeichnis in UModel importieren. Wenn in ein Schema andere Schemas inkludiert oder importiert wurden, so werden diese auch in das Modell importiert.

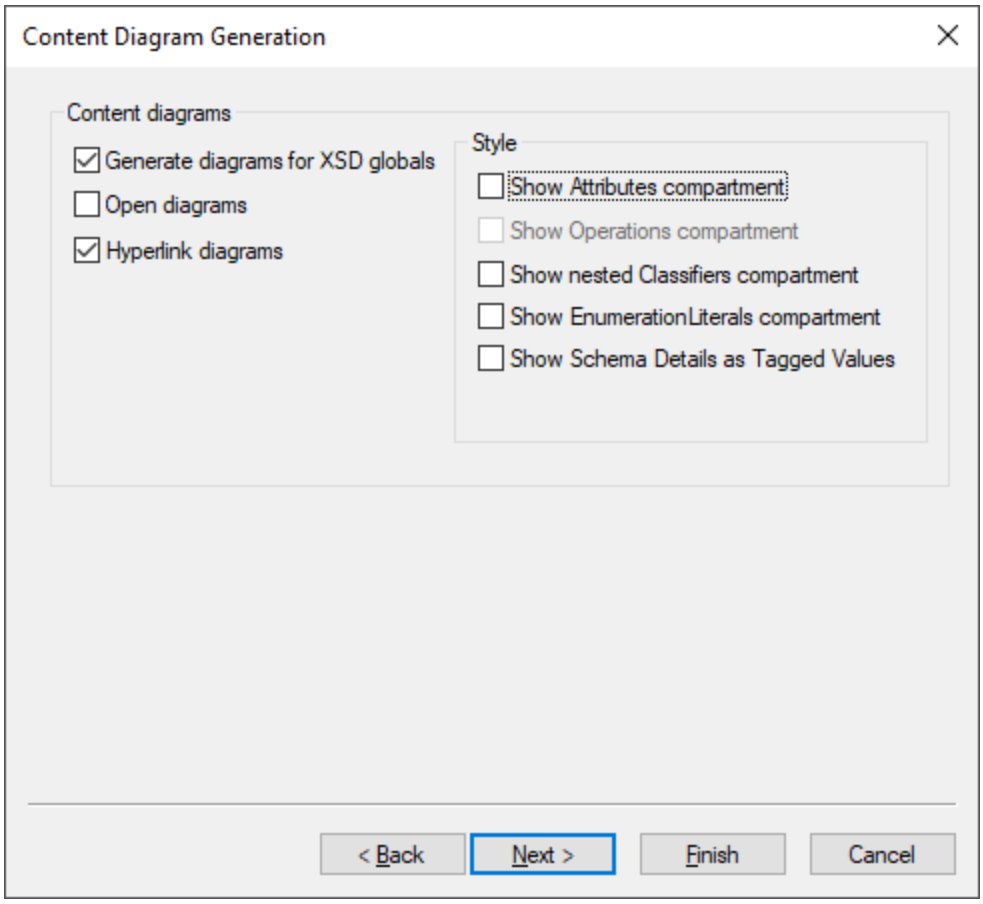
So importieren Sie ein einzelnes XML-Schema:

1. Wählen Sie den Menübefehl **Projekt | XML-Schema-Datei importieren**.
2. Klicken Sie auf **Durchsuchen** und wählen Sie das Quellschema aus. In diesem Beispiel können Sie das folgende Schema verwenden: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial\OrgChart.xsd**.

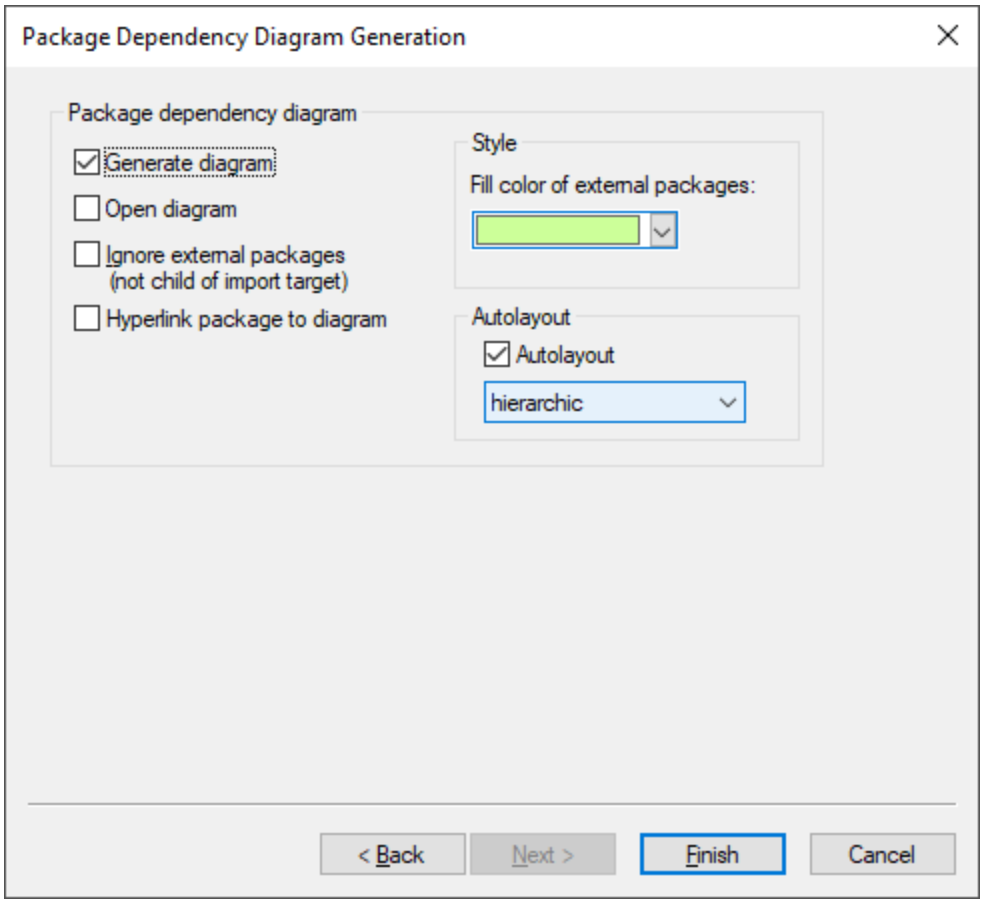


3. Damit anhand des Schemas Diagramme generiert werden, stellen Sie sicher, dass das Kontrollkästchen **Diagrammgenerierung aktivieren** aktiviert ist und klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH



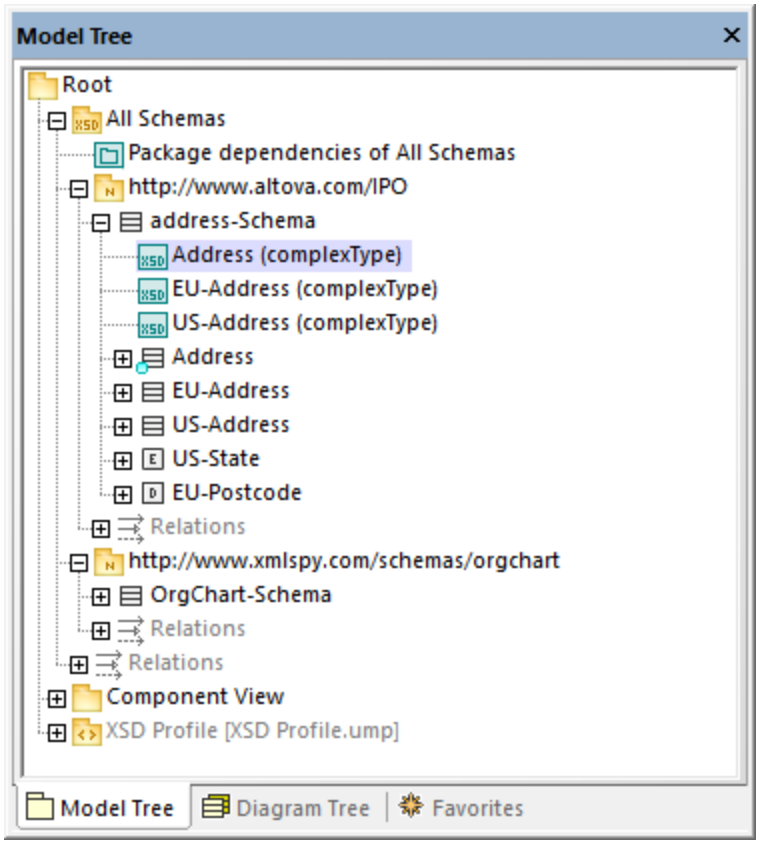
4. Um, wie in diesem Beispiel gezeigt, für jede globale Komponente im Schema ein separates Diagramm zu erstellen, aktivieren Sie die Option **Diagramme für globale XSD-Elemente generieren**. Um alle generierten Diagramme nach dem Import zu öffnen, wählen Sie die Option **Diagramme öffnen**. Über die Optionen aus der Gruppe "Stil" können Sie definieren, welche Bereiche standardmäßig für die einzelnen Schemakomponenten in Diagrammen angezeigt werden sollen. Bei Auswahl der Option **Schemainformationen als Eigenschaftswerte anzeigen** werden die Schemainformationen als Eigenschaftswerte[156] angezeigt.
5. Klicken Sie auf **Weiter**. Um ein Paketabhängigkeitsdiagramm wie dasjenige in diesem Beispiel zu generieren, aktivieren Sie das Kontrollkästchen **Diagramm generieren**.



6. Klicken Sie auf **Fertig stellen**.

Nachdem das Schema fertig in UModel importiert wurde, wird ein neues Paket namens **Alle Schemas** erstellt und automatisch als die XSD Namespace Root definiert. Im Schema **OrgChart.xsd** aus diesem Beispiel werden Typen aus einem anderen Namespace, nämlich aus dem Schema **ipo.xsd** importiert. Folglich werden beide Schemas nach dem Import im Fenster "Modell-Struktur" unter ihrem jeweiligen Namespace angezeigt:

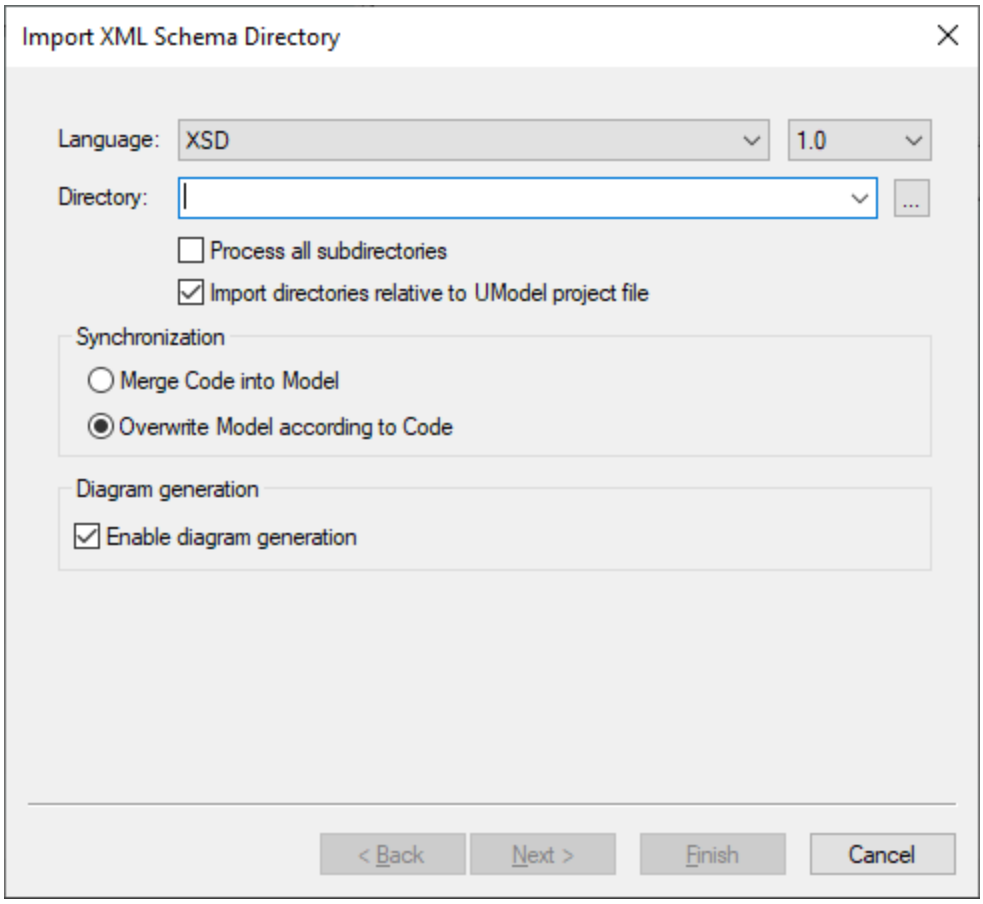
© 2016-2022 Altova GmbH



Wenn Sie das Kontrollkästchen **Diagramme für globale XSD-Elemente generieren** aktiviert haben, wird anhand aller globalen XSD-Komponenten ein XML-Schema-Diagramm generiert, wobei die Diagramme, wie etwa das Diagramm "Address (complexType)" in der Abbildung oben, unter den jeweiligen Namespace-Paketen angelegt werden.

So importieren Sie mehrere XML-Schemas:

1. Wählen Sie den Menübefehl **Projekt | XML-Schemaverzeichnis importieren**.

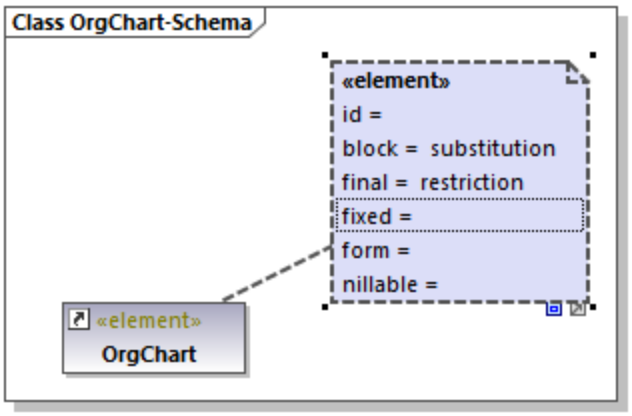


2. Um Schemas aus allen Unterverzeichnissen des ausgewählten Verzeichnisses zu importieren, aktivieren Sie das Kontrollkästchen **Alle Unterverzeichnisse verarbeiten**. Der restliche Importvorgang läuft ab, wie oben für ein einzelnes XML-Schema beschrieben.

Änderung der Anzeige von Eigenschaftswerten

Nach dem Import eines XML-Schemas werden bestimmte Schemainformationen im Diagramm eventuell als Eigenschaftswerte angezeigt, wenn Sie beim Import die Option **Schemainformationen als Eigenschaftswerte anzeigen** aktiviert haben.

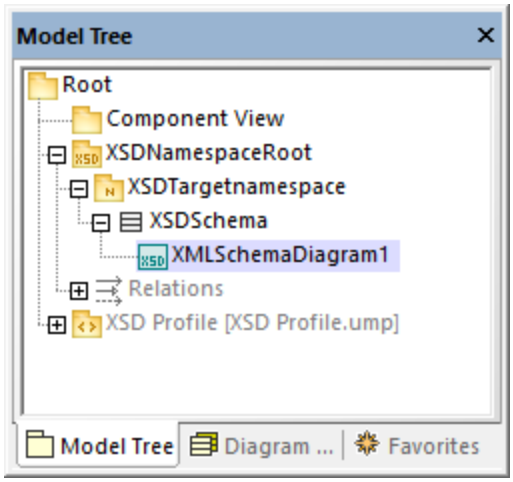
© 2016-2022 Altova GmbH



Sie können konfigurieren, ob diese Informationen im Diagramm ein- oder ausgeblendet werden sollen. Klicken Sie dazu mit der rechten Maustaste auf das Element und wählen Sie im Kontextmenü den Befehl **Eigenschaftswerte | <Option>**. Sie können die Anzeige von Eigenschaftswerten nicht nur für einzelne Elemente, sondern auch global auf Projektebene konfigurieren. Nähere Informationen dazu finden Sie unter Anzeigen oder Ausblenden von Eigenschaftswerten 159.

9.3.1.2 Modellieren von XML-Schemas

Neue XML-Schema-Projekte haben in UModel die unten gezeigte Struktur. Diese Struktur wird beim ersten Mal, wenn Sie ein XML-Schema-Diagramm zu einem neuen UModel-Projekt hinzufügen, automatisch erstellt.



Die Pakete "Root" und "Komponentenansicht" kommen in allen UModel-Projekten vor und können nicht gelöscht werden. "Root" ist die oberste Ebene, unterhalb welcher weitere Pakete hinzugefügt werden und "Component View" wird für das Code Engineering (in diesem Fall für den Import und die Generierung von Schema-Dateien) verwendet.

Das Paket "XSDNamespaceRoot" enthält alle in Ihrem bzw. Ihren Schemas verwendeten Namespaces. Um ein Paket in eine XSD Namespace Root umzuwandeln, klicken Sie mit der rechten Maustaste darauf und wählen Sie im Kontextmenü den Befehl **Code Engineering | Als XSD Namespace Root definieren**. Wenn Sie ein

vorhandenes XML-Schema in das Projekt importieren, erhält das Paket standardmäßig den Namen "Alle Schemas".

Das Paket "XSDTargetNamespace" ist ein XML-Schema-Namespace. Unter ein und derselben XSD Namespace Root können sich mehrere solcher Namespaces befinden. Um ein Paket in einen Namespace umzuwandeln, wählen Sie es zuerst aus und aktivieren Sie anschließend im Fenster "Eigenschaften" die Eigenschaft (Stereotyp) «namespace».

"XSDSchema" ist ein Schema oder in UML eine Klasse, für die im Fenster "Eigenschaften" die Eigenschaft (Stereotyp) «schema» ausgewählt ist.

XMLSchemaDiagram1 ist das eigentliche Diagramm, in dem das Modell des Schemas beschrieben ist. Sie können XML-Schema-Diagramme unter einer XSD Namespace Root, einem XML Schema Namespace oder einem XML-Schema erstellen. Im oben gezeigten Beispielprojekt wurde das Diagramm unter dem XML-Schema erstellt.

Mit Hilfe des **XSD-Profils** werden alle Typen und Strukturen, die für die Arbeit mit XML-Schemas benötigt werden, im Projekt aktiviert. Wenn Ihr Projekt dieses Profil nicht hat, werden Sie bei der Erstellung eines neuen XML-Schemas aufgefordert, es zu inkludieren. Sie können das XSD-Profil auch explizit zu einem Projekt hinzufügen, siehe Anwenden von UModel-Profilen 170.

Erstellen von XML-Schema-Diagrammen

So erstellen Sie ein neues XML-Schema-Diagramm:

1. Wählen Sie eine der folgenden Methoden:
   a. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" 86 auf ein Paket und wählen Sie im Kontextmenü den Befehl **XML-Schema-Diagramm**.
   b. Klicken Sie mit der rechten Maustaste im Fenster "Diagramm-Struktur" 90 auf "Diagramme" oder "XML-Schema-Diagramme" und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | XML-Schema-Diagramm**. Darauf wird ein Dialogfeld angezeigt, in dem Sie aufgefordert werden, den Owner (Eigentümer) des Diagramms auszuwählen. Wählen Sie ein Paket für das Diagramm aus und klicken Sie auf **OK**.
2. Wenn das aktuelle UModel-Projekt das XSD-Profil nicht enthält, wird ein Dialogfeld geöffnet, in dem Sie aufgefordert werden, es zu inkludieren. Klicken Sie auf **OK**, um das XSD-Profil in das aktuelle Projekt zu inkludieren, siehe auch Anwenden von UModel-Profilen 170.

Hinzufügen von neuen XML-Schema-Elementen

So fügen Sie XML-Schema-Elemente zu einem Diagramm hinzu:

- Klicken Sie auf eine bestimmte Symbolleisten-Schaltfläche und anschließend in das XML-Schema-Diagramm.



Um mehrere Elemente desselben Typs einzufügen, halten Sie die **Strg**-Taste gedrückt und klicken Sie mehrmals in das Diagramm.

© 2016-2022 Altova GmbH

XML-Schema-Diagramme können, wie oben erwähnt, auf verschiedenen Ebenen der Projektstruktur erstellt werden. Wenn sich das Diagramm auf einer Ebene befindet, die die Platzierung bestimmter Elemente nicht zulässt, werden bestimmte Symbolleisten-Schaltflächen nicht benötigt. Für diese wird ein Tooltipp mit Informationen angezeigt, anstatt dass das Element hinzugefügt wird.

In der nachstehenden Tabelle finden Sie eine Liste aller Symbolleisten-Schaltflächen und ihrer Bedeutung.

	**XSD Targetnamespace**	Fügt einen XSD Target Namespace hinzu. Klicken auf diese Schaltfläche ist sinnvoll, wenn das Diagramm direkt unter einer XSD Namespace Root erstellt wurde.
	**XSD Schema**	Fügt eine XML-Schema-Definition (XSD) hinzu. Klicken auf diese Schaltfläche ist sinnvoll, wenn das Diagramm unter einem XSD Target Namespace erstellt wurde.
	**Element (global)**	Fügt ein globales Element zum Diagramm hinzu. Wenn Sie ein Element hinzufügen, wird im Attributbereich automatisch eine Eigenschaft mit demselben Namen wie das Element generiert. Definieren Sie, dass der Typ der Eigenschaft den Typ des Elements definiert.
	**Group**	Fügt eine benannte Modellgruppe zum Diagramm hinzu.
	**Complex Type**	Fügt einen globalen ComplexType zum Diagramm hinzu. In der UML-Terminologie ist dies eine Klasse, auf die die Stereotype «global» und «complexType» angewendet wurden.
	**Complex Type mit Simple Content**	Fügt einen globalen ComplexType mit einfachem Inhalt zum Diagramm hinzu. In der UML-Terminologie ist dies ein Datentyp, auf den die Stereotype «global», «complexType» und «simpleContent» angewendet wurden.
	**Simple Type**	Fügt einen globalen SimpleType hinzu.
	**List**	Fügt einen list-Typ hinzu.
	**Union**	Fügt einen union-Typ hinzu.
	**Enumeration**	Fügt eine Enumeration hinzu.
	**Attribute**	Fügt ein Attribut hinzu.
	**Attribute Group**	Fügt eine Attributgruppe hinzu.
	**Notation**	Fügt einen notation-Typ hinzu.
	**Import**	Fügt eine Import-Beziehung hinzu.
	**Include**	Fügt eine Include-Beziehung hinzu.
	**Redefine**	Fügt eine Redefine-Beziehung hinzu.
	**Restriction**	Fügt eine Restriction-Beziehung hinzu.

	Extension	Fügt eine Extension-Beziehung hinzu.
	Substitution	Fügt eine Substitution -Beziehung hinzu.
	Kommentar	Fügt einen Kommentar hinzu. Kommentare werden in Annotationen konvertiert, wenn Sie anhand des Modells die Schema-Datei generieren. Durch Auswahl des gewünschten Stereotyps aus dem Fenster "Eigenschaften" können Sie den Annotationstyp definieren.
	Anmerkung	Fügt eine erklärende Anmerkung hinzu.
	Anmerkung verknüpfen	Verknüpft eine Anmerkung mit einem anderen Element im Diagramm.

Eine schrittweise Anleitung zur Modellieung eines Schemas finden Sie unter Beispiel: Erstellen und Generieren eines XML-Schemas 500.

9.3.1.3 Beispiel: Erstellen und Generieren eines XML-Schemas

In diesem Beispiel wird beschrieben, wie Sie mit UModel Schritt für Schritt ein neues XML-Schema modellieren. Nachdem Sie das Schema grafisch mittels UML modelliert haben, werden Sie die Schema-Datei generieren. Dabei lernen Sie, wie Sie das im Codefragment unten gezeigte Schema **product.xsd** erstellen und generieren.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.altova.com/umodel"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:prod="http://www.altova.com/umodel">
   <xs:simpleType name="SizeType">
      <xs:restriction base="xs:integer">
         <xs:maxInclusive value="10"/>
         <xs:minInclusive value="1"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="ProductType">
      <xs:sequence>
         <xs:element name="number" type="xs:integer">
         </xs:element>
         <xs:element name="size" type="prod:SizeType">
         </xs:element>
      </xs:sequence>
      <xs:attribute name="createdAt" type="xs:date">
      </xs:attribute>
   </xs:complexType>
   <xs:element name="product" type="prod:ProductType">
   </xs:element>
</xs:schema>
```

product.xsd

Wie oben gezeigt, hat das Schema **product.xsd** zwei Namespace-Deklarationen:

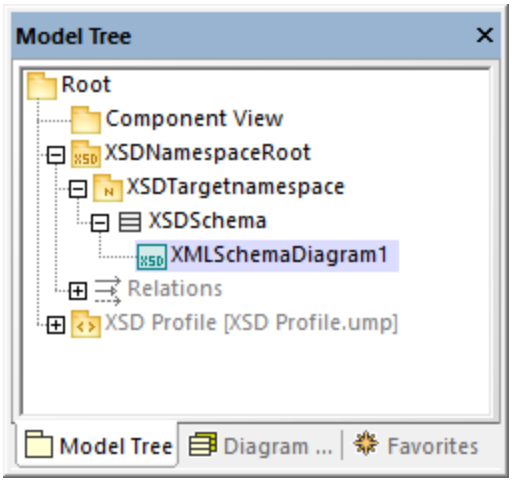
© 2016-2022 Altova GmbH

1. den XML-Schema-Standard-Namespace `http://www.w3.org/2001/XMLSchema`, der auf das Präfix "xs" gemappt ist.
2. den sekundären Namespace `http://www.altova.com/umodel`, der auf das Präfix "prod" gemappt ist, welcher auch der Target Namespace ist.

Des Weiteren hat das XML-Schema ein globales `product`-Element, einen complexType `ProductType` und einen simpleType `SizeType`.

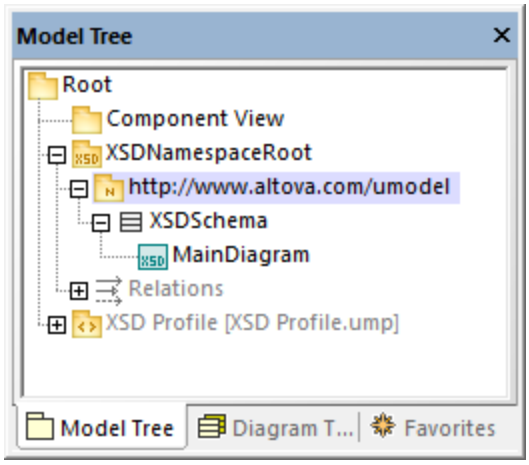
Deklarieren von Namespaces und Dateikodierung

Um fortzufahren, erstellen Sie ein neues UModel-Projekt. Klicken Sie mit der rechten Maustaste auf das **Root**-Paket und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | XML-Schema-Diagramm**. Wenn Sie aufgefordert werden, das UModel XSD-Profil zu inkludieren, klicken Sie auf **OK**.



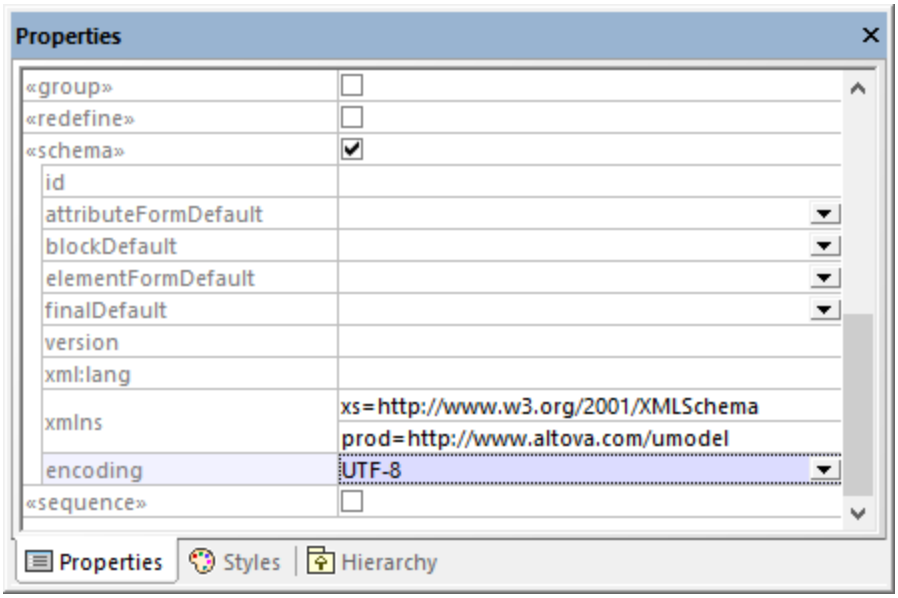
Benennen Sie "XMLSchemaDiagram1" im Fenster Modell-Struktur[86] in "MainDiagram" um. Dies ist das Diagramm, in dem mit Ausnahme der Namespace-Deklarationen die meisten Schema-Komponenten erstellt werden.

Benennen Sie als nächstes "XSDTargetNamespace" in "http://www.altova.com/umodel" um (da dies der erforderliche Target Namespace ist). Damit wird der Target Namespace des neuen Schemas deklariert.



Die beiden "xmlns" Namespaces und die UTF-8-Kodierung können folgendermaßen definiert werden:

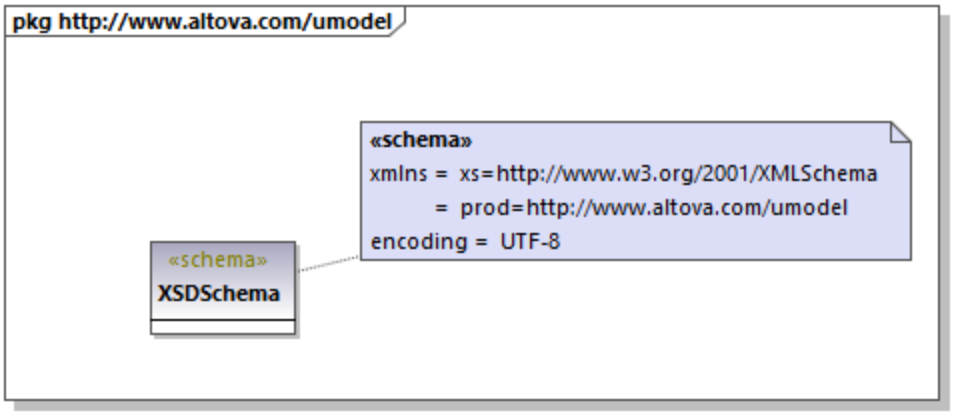
1. Wählen Sie in der Modell-Struktur das **XSDSchema** aus.
2. Klicken Sie im Fenster "Eigenschaften" mit der rechten Maustaste auf die Eigenschaft **xmlns** und wählen Sie den Befehl **Eigenschaftswert hinzufügen | xlmns**.
3. Bearbeiten Sie die Eigenschaften **xmlns** und **Kodierung** wie unten gezeigt.



Optional können Sie schnell auf Namespace-Ebene ein neues XML-Schema-Diagramm erstellen, das dieselben Informationen folgendermaßen visuell darstellt:

1. Klicken Sie in der Modell-Struktur mit der rechten Maustaste auf den Namespace "http://www.altova.com/umodel" und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | XML-Schema-Diagramm**.
2. Wenn ein Meldungsfeld mit dem folgenden Text erscheint: *"Möchten Sie das 'XML-Schema-Diagramm' zu einem neuen 'XSD-Schema' hinzufügen?"*, klicken Sie auf **Nein**.
3. Ziehen Sie das XML-Schema aus der Modell-Struktur in das Diagramm.

© 2016-2022 Altova GmbH

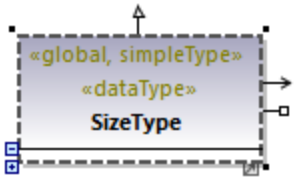


Der Namespace und die Kodierung werden, wie oben gezeigt, als Eigenschaftswerte[156] gespeichert und können auch über das Diagrammfenster bearbeitet werden.

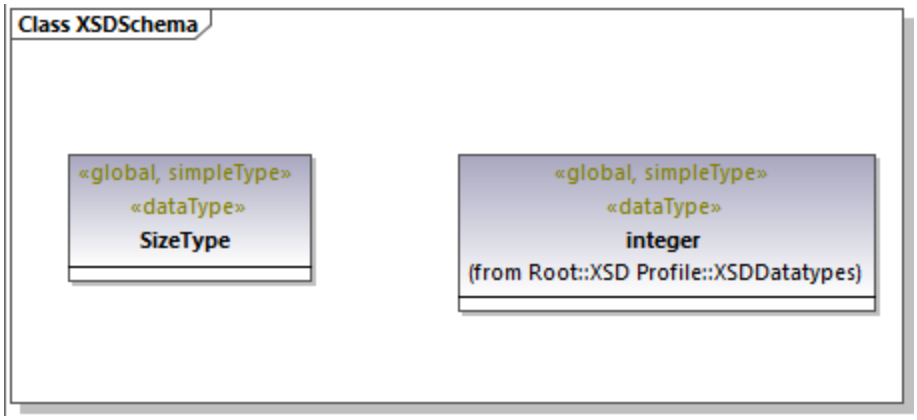
Hinzufügen eines simpleType

In den folgenden Schritten wird der simpleType `SizeType` zum XML-Schema hinzugefügt. Dies ist ein Typ, der den Basistyp `xs:integer` einschränkt, daher fügen wir auch den Basistyp zum Diagramm hinzu und erstellen eine Restriction-Beziehung.

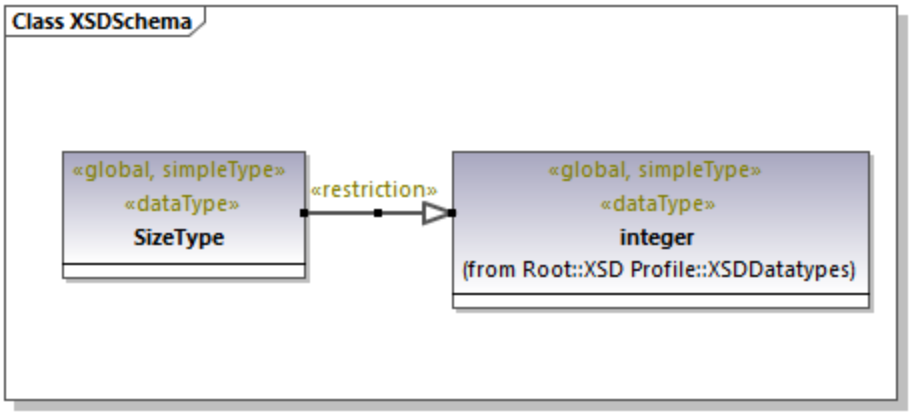
1. Doppelklicken Sie in der Modell-Struktur auf **MainDiagram**, um es zu öffnen.
2. Klicken Sie auf die Symbolleisten-Schaltfläche **XSD Simple Type** und anschließend in das Diagramm.
3. Benennen Sie den neu hinzugefügten SimpleType in `SizeType` um.



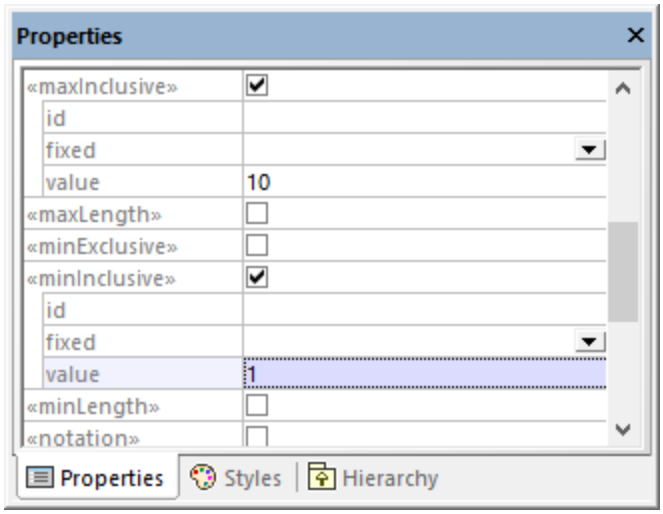
4. Klicken Sie in die Modell-Struktur und drücken Sie **Strg + F.**. Daraufhin wird das Dialogfeld "Suchen" aufgerufen. Geben Sie die ersten Buchstaben von "integer" ein und wählen Sie den Typ `integer` aus dem "XSDDataTypes"-Paket des XSD-Profils aus.
5. Ziehen Sie den Typ `integer` in das Diagramm.



6. Klicken Sie auf die Symbolleisten-Schaltfläche **Restriction** und ziehen Sie den Cursor von `SizeType` auf `integer`. Dadurch wird die Restriction-Beziehung erstellt, siehe auch Erstellen von Beziehungen 143.



7. Um die Werte `minInclusive` und `maxInclusive` zu definieren, wählen Sie den SimpleType aus und bearbeiten Sie die Eigenschaften desselben Namens im Fenster "Eigenschaften".

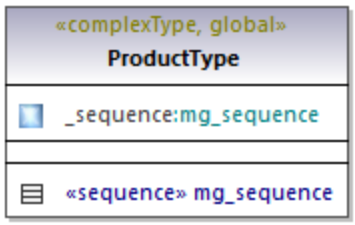


© 2016-2022 Altova GmbH

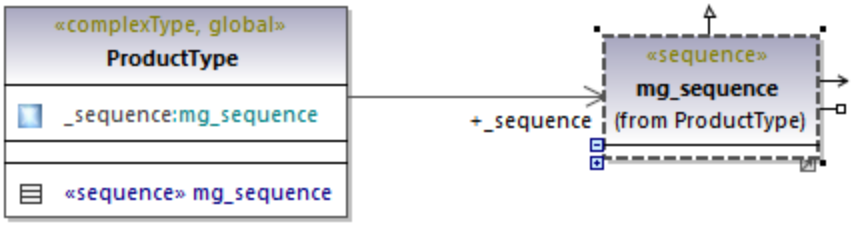
Hinzufügen eines complexType

In den folgenden Schritten wird der complexType `ProductType` zum XML-Schema hinzugefügt. Alle diese Schritte werden ebenfalls im **MainDiagram** durchgeführt.

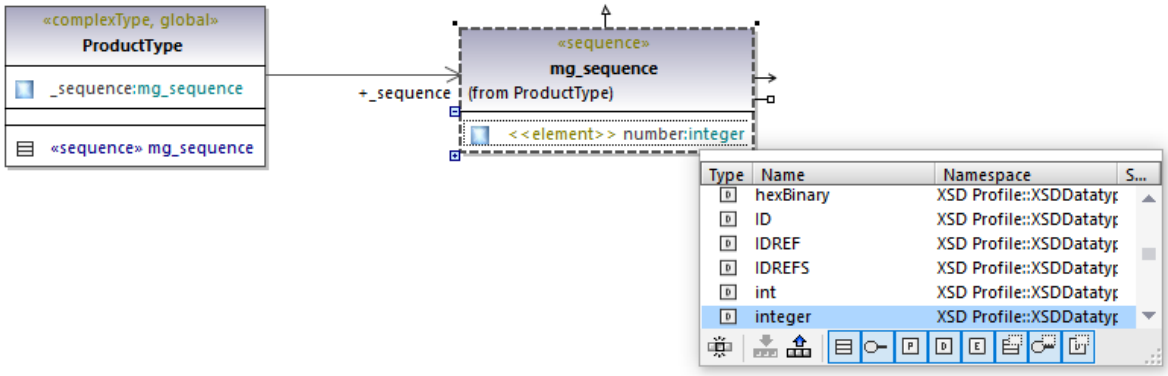
1. Klicken Sie auf die Symbolleisten-Schaltfläche **XSD Complex Type** und anschließend in das Diagramm.
2. Benennen Sie den ComplexType in `ProductType` um.
3. Klicken Sie mit der rechten Maustaste auf den ComplexType und wählen Sie im Kontextmenü den Befehl **Neu | XSD Sequence**.



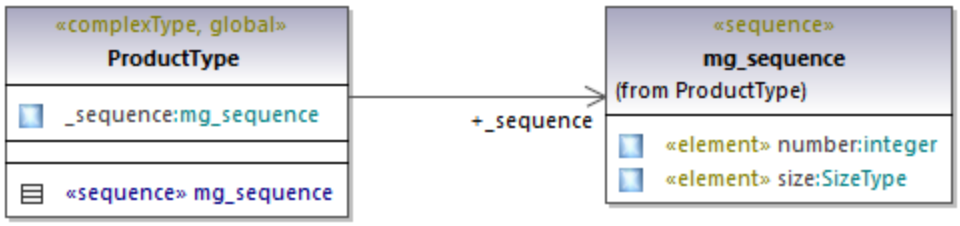
4. Ziehen Sie die Klasse «sequence» aus dem ComplexType in das Diagramm.



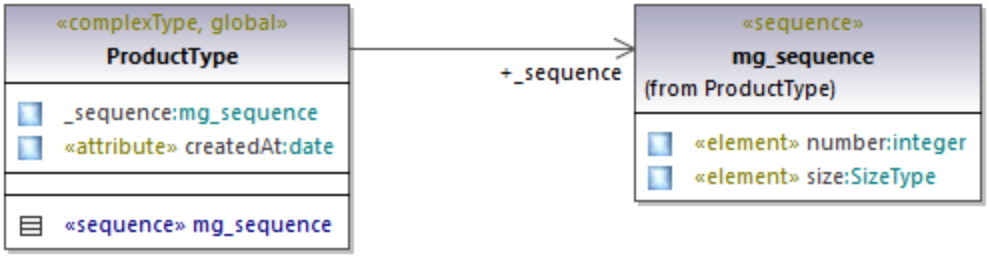
5. Rechtsklicken Sie auf die Sequenz und wählen Sie **Neu | XSD Element (local)**.
6. Ändern Sie den Elementnamen in **number** und definieren Sie als Typ `integer`. Der Typ `integer` ist ein XML-Schema-Basistyp aus dem XSD-Profil. Eine Anleitung zum Definieren des Typs eines Elements finden Sie unter Typ-Autokomplettierung in Klassen 140.



7. Erstellen Sie auf dieselbe Art, wie oben beschrieben, das Element **size** vom Typ `SizeType`. Beachten Sie, dass `SizeType` der zuvor erstellte SimpleType ist.



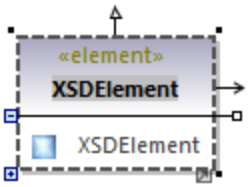
8. Klicken Sie mit der rechten Maustaste im Diagramm auf den ComplexType und wählen Sie im Kontextmenü den Befehl Neu | XSD Attribute (local).
9. Ändern Sie den Attributnamen in **createdAt** und definieren Sie als Typ `date`.



Hinzufügen eines Elements

Nachdem wir nun alle erforderlichen Typen des Schemas definiert haben, können Sie folgendermaßen ein Produktelement vom Typ `ProductType` hinzufügen:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **XSD Element (global)** und anschließend in das Diagramm. Beachten Sie, dass eine Klasse mit dem Stereotyp `«element»` und einer einzigen Eigenschaft hinzugefügt wird.



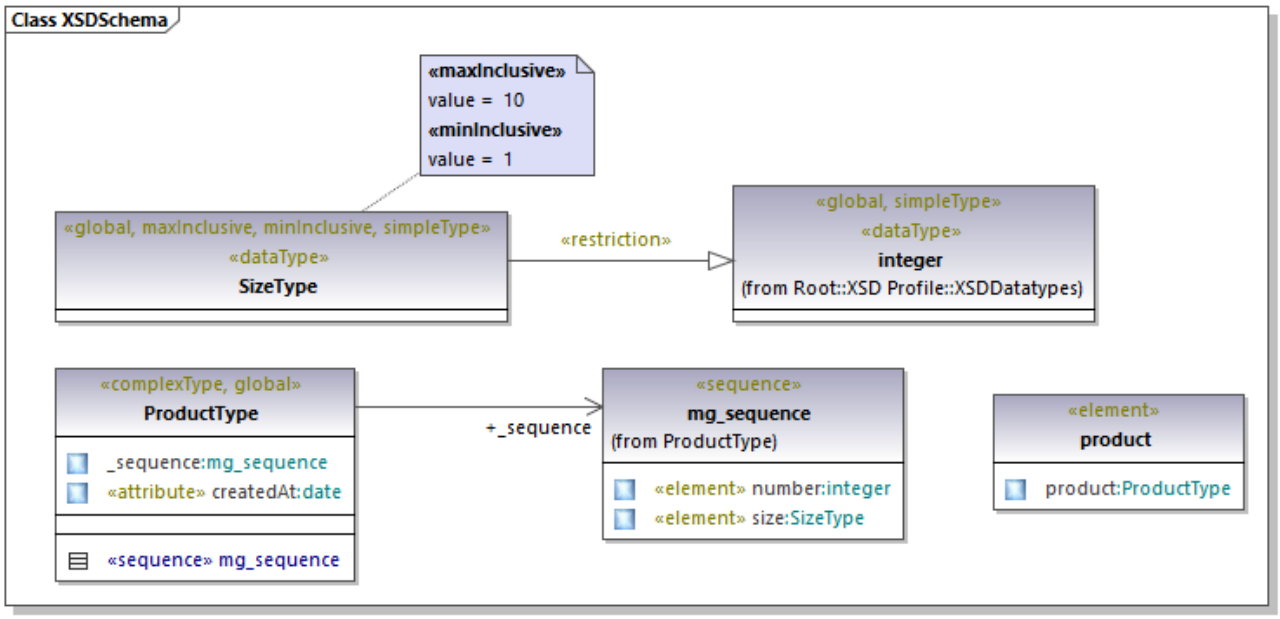
2. Benennen Sie die Eigenschaft in **product** um und ändern Sie ihren Typ in `ProductType`.



Fertiges Design

Mit den obigen Schritten haben wir den Design-Teil des Schemas abgeschlossen. Ihr komplettes Schema-Design sollte nun folgendermaßen aussehen:

© 2016-2022 Altova GmbH

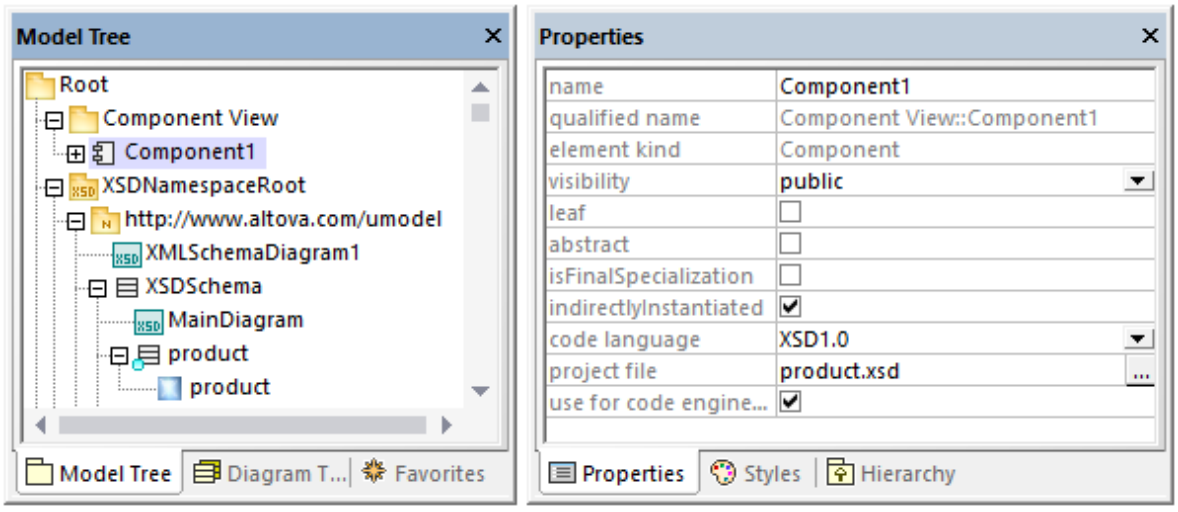


Vorbereitung für das Code Engineering

Damit anhand des Modells eine Schema-Datei generiert werden kann, wollen wir nun eine Code Engineering-Komponente hinzufügen, die die Informationen für die Schemagenerierung liefert. Die Code Engineering-Komponente ähnelt anderen UModel-Projektarten, siehe auch Hinzufügen einer Code Engineering-Komponente 181.

Klicken Sie in der Modell-Struktur mit der rechten Maustaste auf das Paket "Component View" und fügen Sie ein neues Element vom Typ **Komponente** hinzu. Die Eigenschaften der Komponente müssen nun wie unten gezeigt geändert werden:

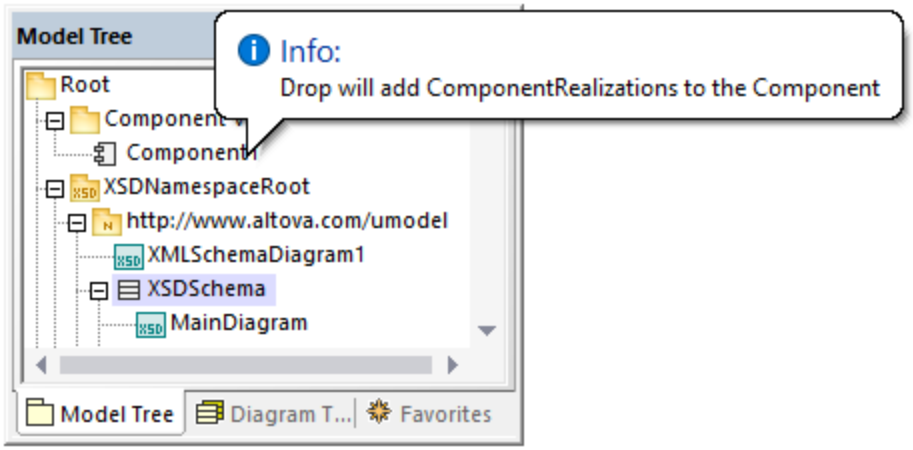
1. Die Eigenschaft **für Code Engineering verwenden** muss aktiviert werden.
2. Die Eigenschaft **Codesprache** der Code Engineering-Komponente muss auf "XSD 1.0" gesetzt werden.
3. Die Eigenschaft **Projektdatei** der Code Engineering-Komponente muss auf die zu generierende Schema-Datei (in diesem Beispiel **product.xsd**) verweisen.



Anmerkung: Wenn die Eigenschaft **Projektdatei** fehlt, geben Sie in die Eigenschaft **Verzeichnis product.xsd**ein und drücken Sie die **Eingabetaste**. Daraufhin wird ein Meldungsfeld angezeigt, in dem Sie aufgefordert werden, statt dessen eine Projektdatei zu referenzieren. Klicken Sie zur Bestätigung auf **Ja**.

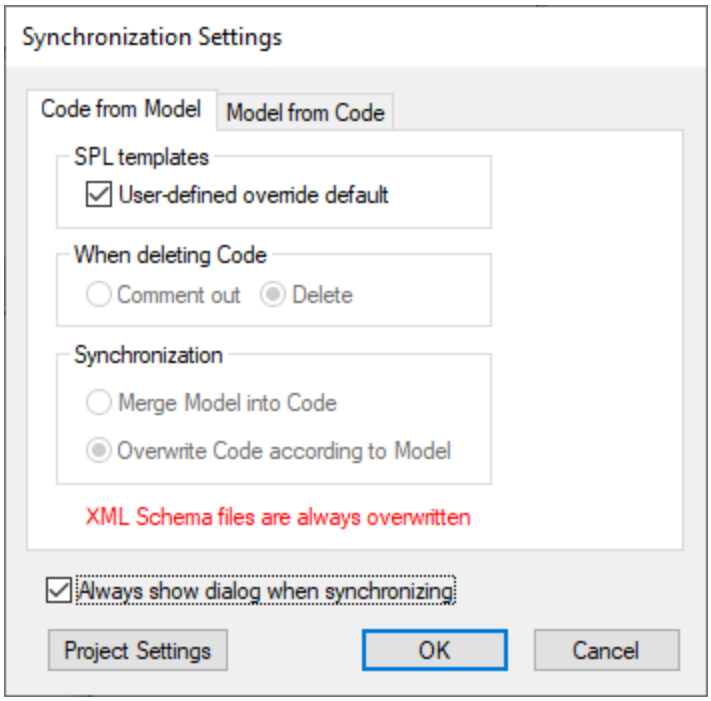
Schließlich muss das XML-Schema von der Code Engineering-Komponente wie unter Hinzufügen einer Code Engineering-Komponente[181] beschrieben, realisiert werden. Die schnellste Art, um die **Komponentenrealisierungsbeziehung** in diesem Beispiel zu erstellen, ist die folgende:

- Ziehen Sie das **XSDSchema**-Schema in der Modell-Struktur über die Code Engineering-Komponente (**Component1**) und lassen Sie die Maustaste los, wenn ein Tooltipp wie der folgende angezeigt wird:



Sie können nun die Schema-Datei generieren. Drücken Sie dazu entweder **F12** oder wählen Sie den Menübefehl **Projekt | Überschreibe Programmcode von UModel Projekt**. Beachten Sie, dass eine Zusammenführung im Fall von XML-Schemas nicht unterstützt wird, daher wird im Dialogfeld eine Meldung in Rot mit diesem Inhalt angezeigt.

© 2016-2022 Altova GmbH



Das neue XML-Schema wird im selben Ordner wie Ihr UModel-Projekt generiert.

9.3.2 Business Process Modeling Notation 1.0 / 2.0

Altova Website: Business Process Modeling in UModel

BPMN ist eine standardisierte Notation für Flussdiagramme, in der Geschäftsprozesse als Arbeitsablauf dargestellt werden und die für alle am Geschäftsprozess Beteiligten einfach zu verstehen ist. UModel unterstützt die BPMN-Versionen 1.0 und 2.0. BPMN 1.0- und BPMN 2.0-Diagramme können gemeinsam im selben UModel-Projekt verwendet werden. BPMN 1.0-Diagramme können jederzeit in BPMN 2.0-Diagramme konvertiert werden.

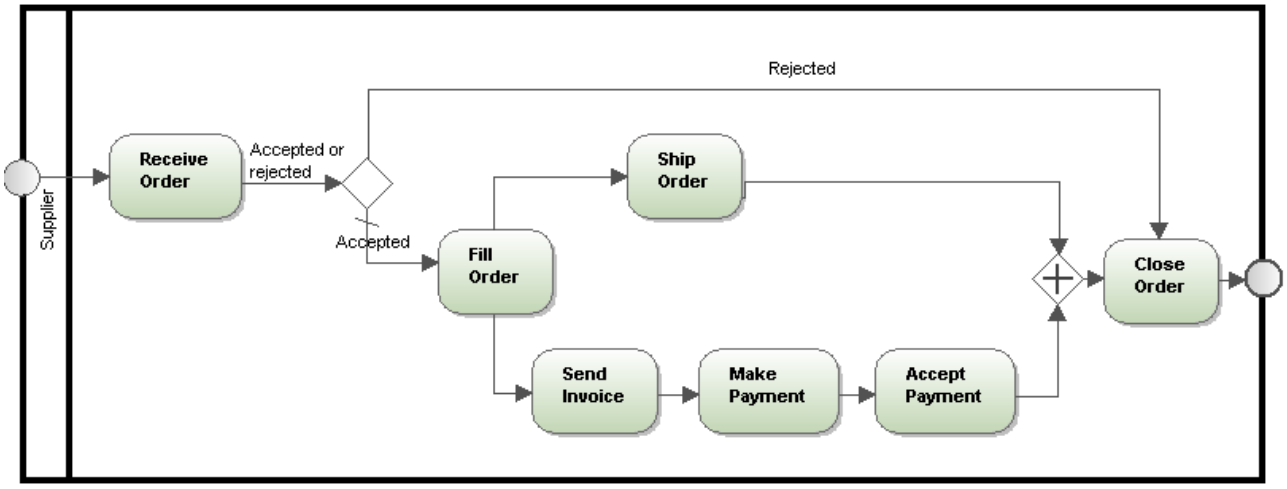
Es gibt vier grundlegende BPMN-Elementkategorien:

Flussobjekte 510	Ereignisse, Aktivitäten (Tasks oder Unterprozesse), Gateways
Verbindungsobjekte 520	Sequenzfluss, Nachrichtenfluss, Assoziation
Swimlanes 522 **(Schwimmbahnen)**	Pool, Lane (Bahn)
Artefakte 523	Datenobjekte, Gruppe, Textannotation

Das Einfügen von BPMN-Diagrammen (Geschäftsprozessdiagrammen) und BPMN-Objekten in UModel erfolgt auf genau dieselbe Art wie das Einfügen von Modellierungselementen.

Objekte können über die Symbolleiste eingefügt werden; Assoziationen zu anderen Objekten können durch Klicken auf die Objektziehpunkte und Ziehen des Konnektors auf das Zielobjekt hergestellt werden. Eigenschaften können über das Fenster "Eigenschaften" [92] angezeigt und definiert werden.

Beachten Sie, dass Sie mehrere Ebenen pro BPMN-Diagramm erstellen können. Nähere Informationen dazu siehe Hinzufügen von Ebenen zu Diagrammen [138].



So konvertieren Sie BPMN 1.0-Diagramme in BPMN 2.0-Diagramme:

1. Klicken Sie mit der rechten Maustaste in ein BPMN 1.0-Diagramm und wählen Sie die Option **In BPMN 2.0-Diagramm konvertieren.**

   Wenn mehrere BPMN 1.0-Diagramme im selben Paket vorhanden sind, werden Sie gefragt, ob alle Diagramme in diesem Paket konvertiert werden sollen.

   Es erscheint eine zweite Meldung, in der Sie gefragt werden, ob das BPMN 2-Profil zum Projekt hinzugefügt werden soll.

   Wenn Sie auf OK klicken, werden die Diagramme konvertiert.

9.3.2.1 Flussobjekte

Flussobjekte sind die grafischen Elemente, die das Verhalten eines Geschäftsprozesses definieren. Es gibt drei Flussobjekte: Ereignisse, Aktivitäten und Gateways.

Ereignisse

Ein Ereignis tritt während eines Geschäftsprozesses auf und wird durch einen Kreis dargestellt. Ereignisse wirken Sie auf den Ablauf des Prozesses aus und haben im Allgemeinen ein Ursache (Auslöser) und ein Ergebnis. Es gibt drei verschiedene Arten von Ereignissen: Startereignisse, Zwischenereignisse und Endereignisse. Jede Gruppe hat ihre eigene Dropdown-Auswahlliste.

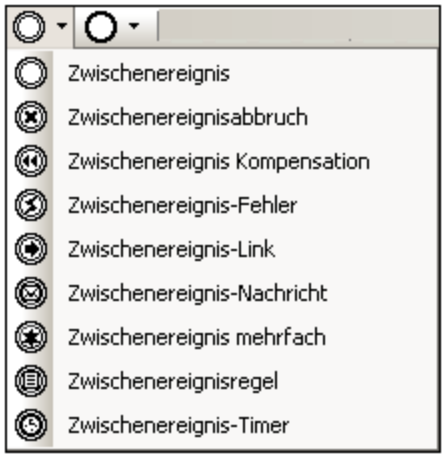
© 2016-2022 Altova GmbH

So fügen Sie ein Ereignis ein:

1. Klicken Sie auf die Auswahlliste, um die Dropdown-Liste des gewünschten Ereignistyps zu öffnen.
2. Wählen Sie das gewünschte Ereignis aus und klicken Sie in das Diagrammregister, um es einzufügen.



Startereignis



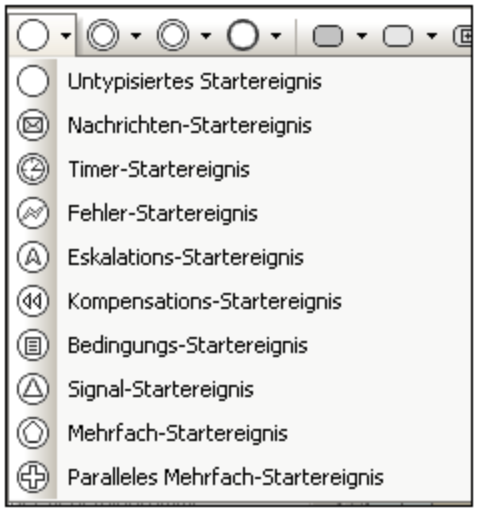
Zwischenereignis

Zwischenereignisse können an den Rand eines Task oder eines Unterprozesses angehängt werden und zeigen an, dass die Aktivität unterbrochen werden soll, wenn das Ereignis ausgelöst wird.



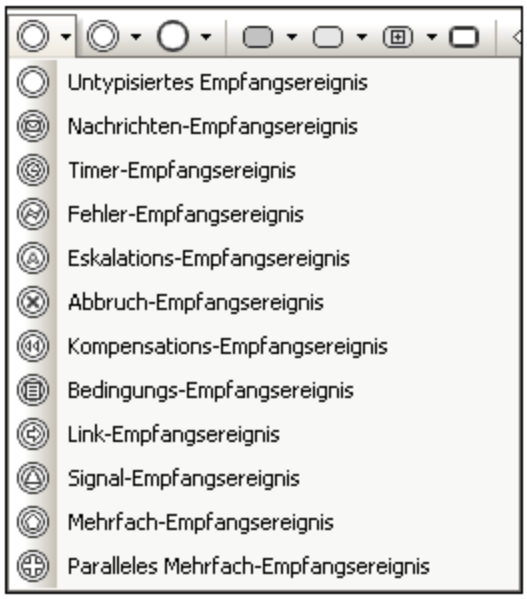
Endereignis

BMPN 2.0-Ereignisse

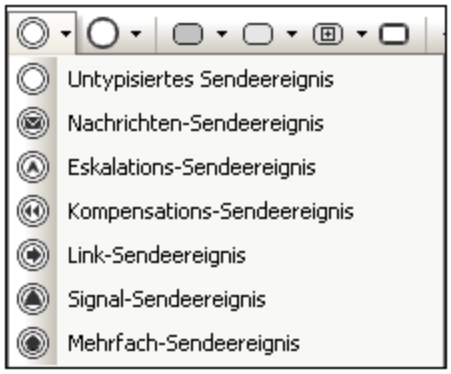


Startereignisse (Start Events)

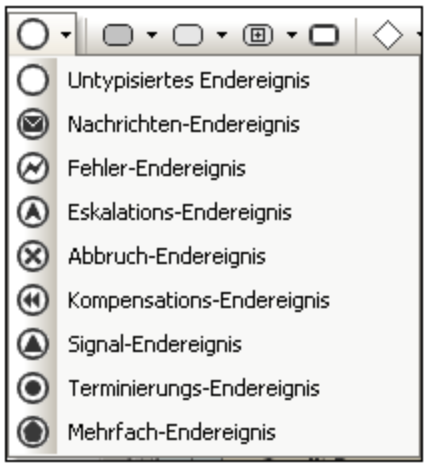
© 2016-2022 Altova GmbH



Empfangsereignisse (Catching Events)



Sendeereignisse (Throwing Events)



Endereignisse (End Events)

Aktivität

Aktivitäten sind Aktionen, die während eines Geschäftsprozesses durchgeführt werden. Aktivitäten werden durch abgerundete Rechtecke dargestellt. Prozessmodelle können die folgenden Aktivitätstypen enthalten: Prozess, Unterprozess und Task. Aktivitäten können einmal erfolgen oder mehrmals in einer Schleife.



So fügen Sie eine Aktivität ein:

1. Klicken Sie in der Symbolleiste auf die Schaltfläche für den jeweiligen Task bzw. den Unterprozess.
2. Klicken Sie in das Diagrammregister.

Aktivität - Task

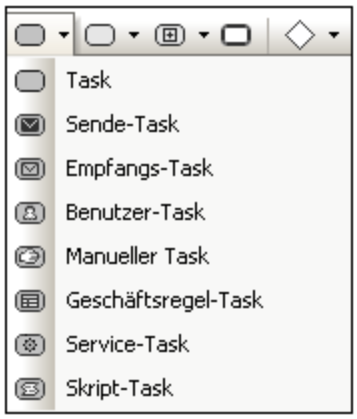
Tasks sind Aktivitäten, die in einem Prozess enthalten sind. Aufgaben können nicht weiter in Sub-Tasks unterteilt werden. Sie sind unteilbar.

Schleifen-Task	
Mehrfachinstanz-Task	

 © 2016-2022 Altova GmbH

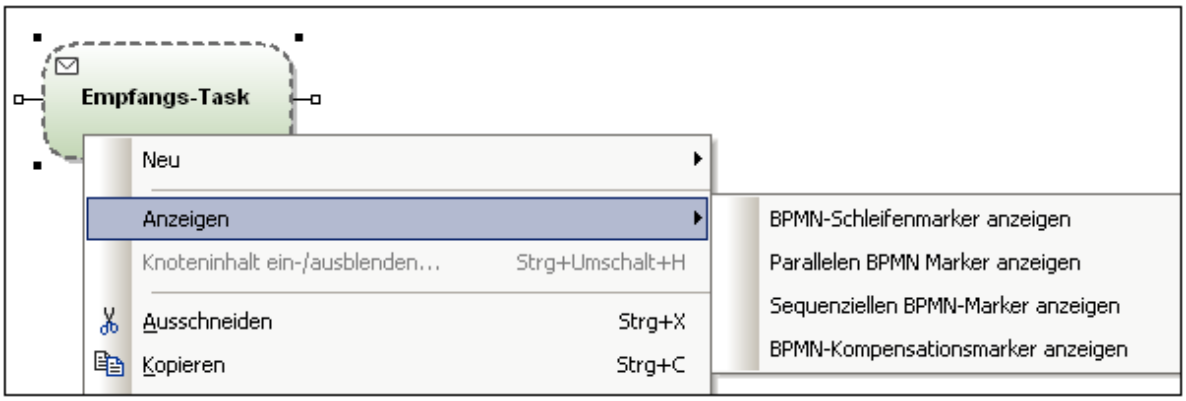
Kompensations-Task	

BPMN 2.0 Tasks



So definieren Sie einen Schleifenmarker, einen parallelen Marker, einen sequenziellen Marker oder einen Kompensations-Marker:

- Klicken Sie mit der rechten Maustaste auf den eingefügten Task und wählen Sie den entsprechenden Marker aus, z.B. **Anzeigen | Parallelen BPMN-Marker anzeigen**.



Anmerkung: Sie können den Marker auch auf dem Register "Eigenschaften" unter dem Eintrag "MultiInstanceLoopCharacteristics" definieren.

Aktivität - Unterprozess

Ein Unterprozess ist eine aus mehreren Aktivitäten zusammengesetzte Aktivität, die in einem Prozess enthalten ist und die Entwicklung eines hierarchischen Geschäftsprozessmodells gestattet. Ein Unterprozess kann in Form verschiedener Sub-Aktivitäten in weitere Unterabschnitte aufgeteilt werden.

Ein eingeklappter Unterprozess[518] wird als Element auf oberster Ebene angezeigt, wobei die Details des Unterprozesses nicht sichtbar sind. Ein Plus-Symbol im Element zeigt an, dass eine weitere komplexere Ebene vorhanden ist.

Bei einem aufgeklappten Unterprozess[517] werden die Einzelheiten des Unterprozesses innerhalb des Rahmens angezeigt. Beachten Sie, dass ein Sequenzfluss die Umrandung eines Unterprozesses nicht passieren kann.

Gateway

Gateways dienen als Punkt, an dem Sequenzflüsse sich in einem Prozess teilen bzw. zusammenlaufen. Gateways werden immer als Raute angezeigt (*siehe Tabelle unten*).

Inklusiver Gateway (OR)	
Paralleler Gateway (AND)	
Datenbasierter exklusiver Gateway (XOR)	
Ereignisbasierter exklusiver Gateway (XOR)	
Komplexer Gateway (Entscheidung/Zusammenfluss)	

BPMN 2.0 Gateways

In der Abbildung unten sehen Sie unterstützte BPMN 2.0 Gateways. Sie können einen exklusiven Gateway in UModel mit oder ohne an X anzeigen. Um das Symbol mit X zu sehen, setzen Sie den Wert `showXIcon` eines exklusiven Gateways auf `true`.

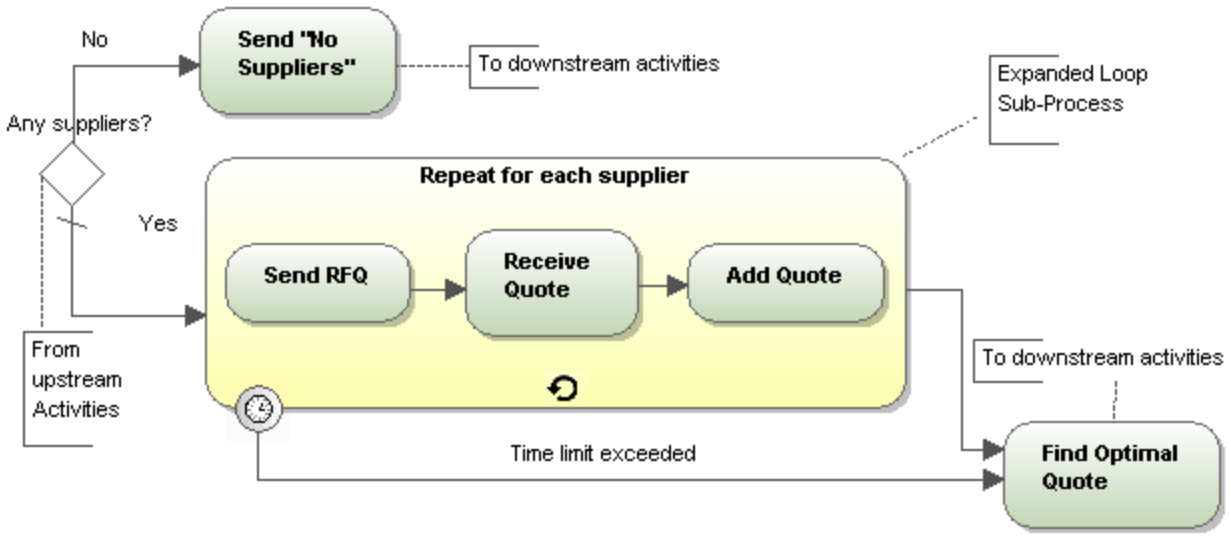


© 2016-2022 Altova GmbH

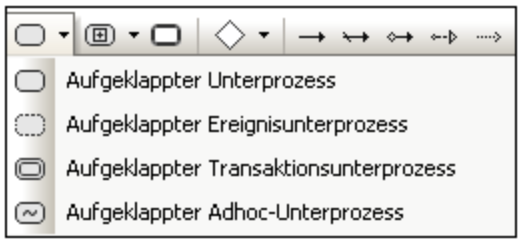
9.3.2.1.1 Aufgeklappte Unterprozesse

Bei aufgeklappten Versionen von Unterprozessen werden die Prozessdetails innerhalb der Elementumrandung angezeigt.

Aufgeklappter Unterprozess	
Aufgeklappter **Schleifen**unterprozess	
Aufgeklappter **Mehrfachinstanz**-Unterprozess	
Aufgeklappter **Ad Hoc**-Unterprozess	
Aufgeklappter **Kompensations**-Unterprozess	

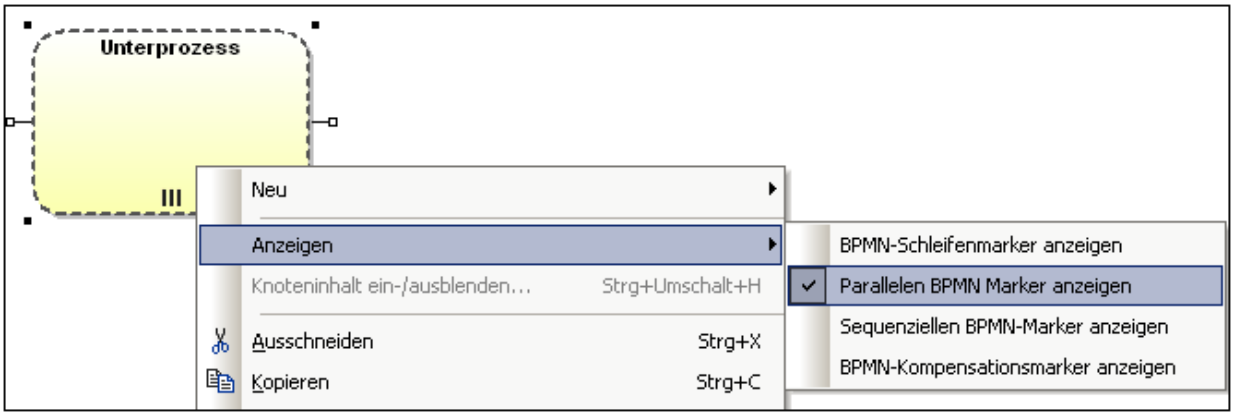


Aufgeklappte BPMN 2.0-Unterprozesse



So definieren Sie einen Schleifenmarker, einen parallelen Marker, einen sequenziellen Marker oder einen Kompensations-Marker:

1. Klicken Sie mit der rechten Maustaste auf den eingefügten Task und wählen Sie den entsprechenden Marker aus, z.B. **Anzeigen | Parallelen BPMN-Marker anzeigen**.



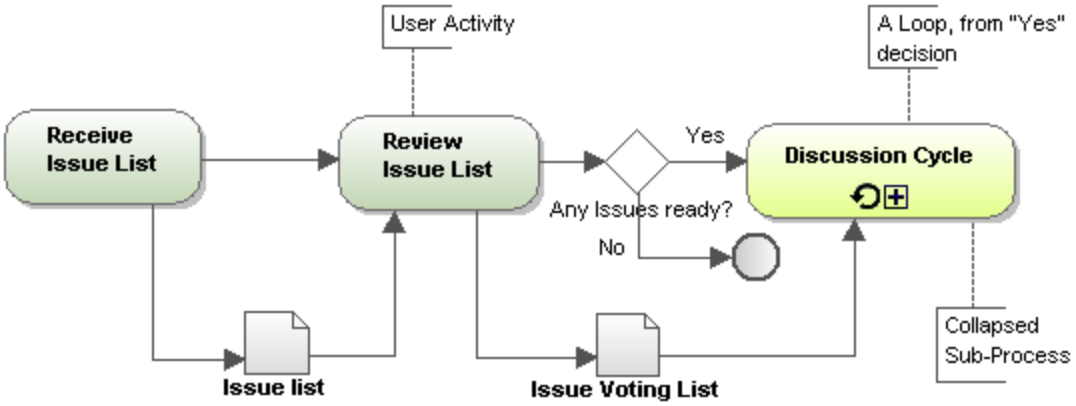
9.3.2.1.2 Eingeklappte Unterprozesse

Bei eingeklappten Versionen von Unterprozessen werden die Prozessdetails ausgeblendet. Die jeweilige Unterprozessart wird durch das Symbol im Unterprozesselement angezeigt.

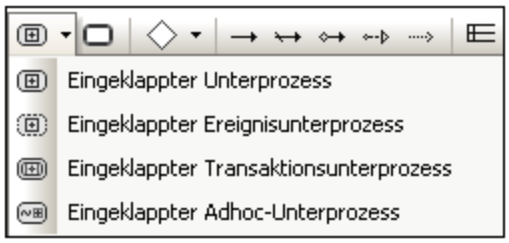
Eingeklappter Unterprozess	
Eingeklappter **Schleifen**unterprozess	

© 2016-2022 Altova GmbH

Eingeklappter **Mehrfachinstanz-** Unterprozess	
Eingeklappter **Ad Hoc** Unterprozess	
Eingeklappter **Kompensations-** Unterprozess	

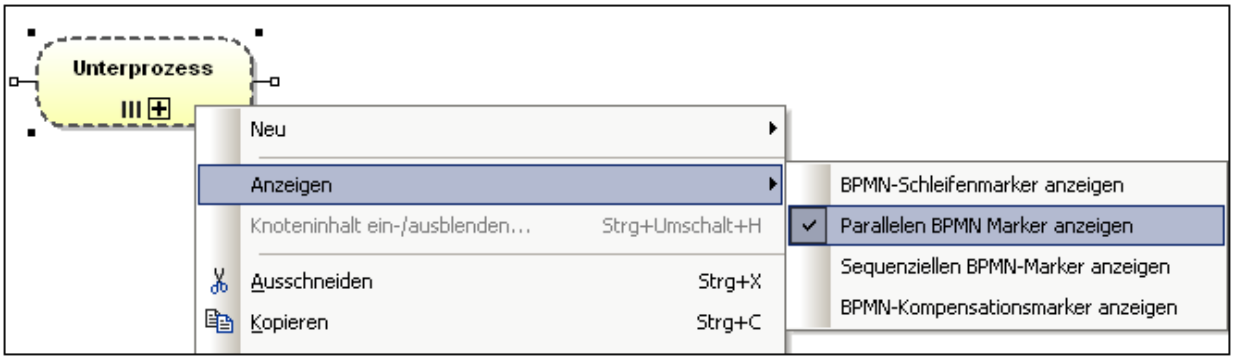


Eingeklappte BPMN 2.0-Unterprozesse



So definieren Sie einen Schleifenmarker, einen parallelen Marker, einen sequenziellen Marker oder einen Kompensations-Marker:

1. Klicken Sie mit der rechten Maustaste auf den eingefügten Task und wählen Sie den entsprechenden Marker aus, z.B. **Anzeigen | Parallelen BPMN-Marker anzeigen**.

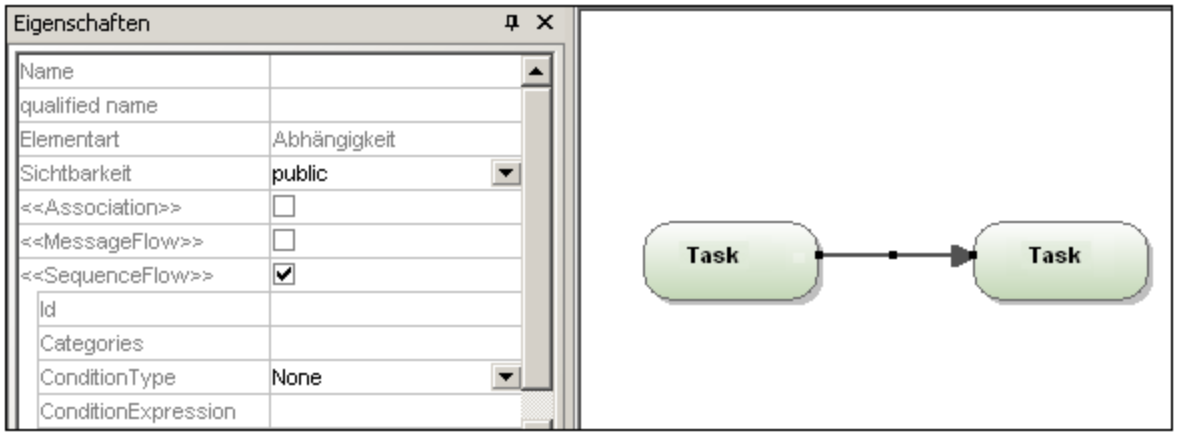


9.3.2.2 Verbinden von Objekten

Objekte können auf zwei Arten miteinander verbunden werden: durch einen Fluss (mit einer Sequenz oder Nachricht) und eine Assoziation.

Sequenzfluss

In einem Sequenzfluss werden die bei einem Prozess durchgeführten Aktivitäten in der Reihenfolge gezeigt, in der sie durchgeführt werden.



Bedingter Fluss

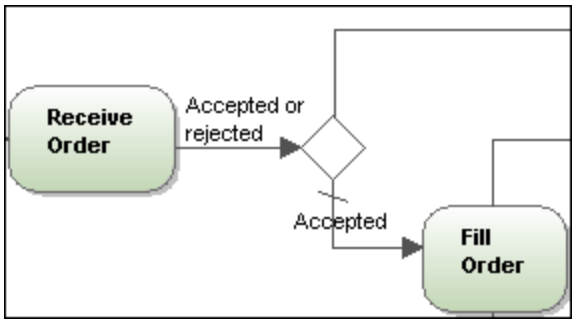
Diese Art von Sequenzfluss kann einen Bedingungsausdruck haben, der darüber entscheidet, ob der Fluss verwendet wird oder nicht. Wenn der Ausgangspunkt des bedingten Flusses eine Aktivität ist, wird am Pfeilanfang eine kleine Raute angezeigt.



© 2016-2022 Altova GmbH

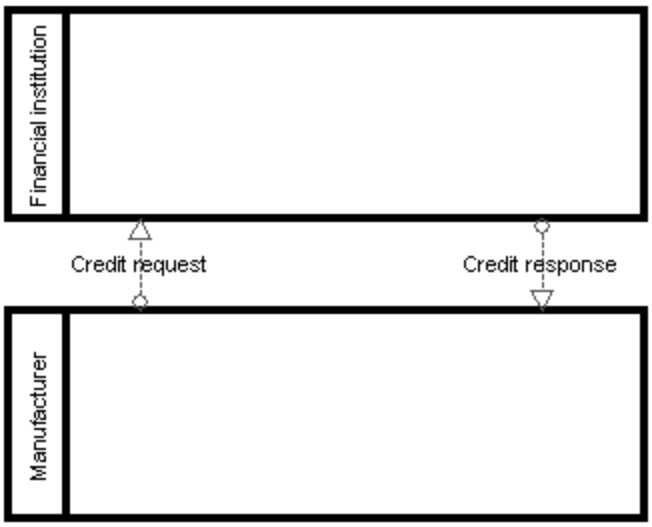
Standardfluss

Diese Art von Fluss wird verwendet, wenn alle anderen bedingten Flüsse in datenbasierten exklusiven oder inklusiven Entscheidungen "false" sind. Dies wird durch einen waagrechten Schrägstrich am Anfang des Pfeils angezeigt, z.B. beim Standardfluss "Accepted".



Nachrichtenfluss

Mit einem Nachrichtenfluss wird der Nachrichtenfluss zwischen zwei Teilnehmern (Einheiten oder Rollen), die diese senden und empfangen können, angezeigt. Teilnehmer werden im Diagramm als separate Pools dargestellt.



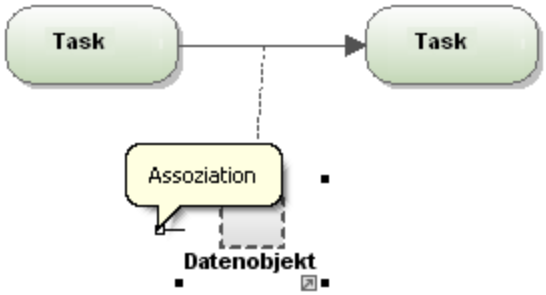
Assoziation

Assoziationen dienen dazu, Text und keinem Fluss angehörende Objektdaten mit Flussobjekten zu verknüpfen und zu zeigen, wie Daten von Aktivitäten ein- und ausgegeben werden. Im Diagramm unten sehen Sie eine Textannotation, die die zusätzliche Information "User Activity" für den Task "Review Issue List" enthält.

So erstellen Sie eine Assoziation zwischen einem Datenobjekt und einer Flusskontrolle:

1. Klicken Sie auf den Assoziationsziehpunkt des Datenobjekts (auf der linken Seite des Objekts).

2. Ziehen Sie den Konnektor auf den Flusskontrollpfeil, der markiert erscheint, wenn Sie die Maustaste loslassen können.



Alternativ dazu können Sie auch auf die Schaltfläche "Assoziation" klicken und das Symbol vom Datenobjekt auf die Flusskontrolle ziehen.

9.3.2.3 Pools / Swimlanes

Pool

Pools dienen zum Aufteilen und Strukturieren von Aktivitäten. In einem Geschäftsprozess kann die Interaktion zwischen verschiedenen Prozessen oder Teilnehmern dargestellt werden. Jeder Teilnehmer wird durch ein rechteckiges Feld, einen so genannten Pool dargestellt. Bei einem Teilnehmer kann es sich um eine Geschäftsrolle oder eine Einheit handeln.



- BPMN-Objekte, die in einen Pool platziert werden, werden Teil des Pools, wenn die Poolumrandung markiert erscheint.
- Objekte in einem Pool können mit Hilfe von Strg + Klick oder durch Aufziehen eines Rechtecks im Pool einzeln ausgewählt werden.
- Klicken Sie auf die Umrandung oder den Titel des Pools, um ihn an eine andere Stelle zu ziehen.

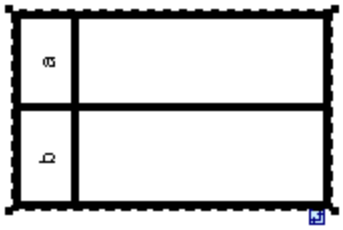
Bahn

Pools können in Bahnen (Lanes) unterteilt werden, die die Aktivitäten in einem Pool kategorisieren. Beachten Sie: Es können sowohl horizontale als auch vertikale Bahnen definiert werden.

So fügen Sie zu einem Pool eine neue Bahn hinzu:

1. Klicken Sie mit der rechten Maustaste auf die Überschrift eines bestehendes Pool-Objekts und wählen Sie den Befehl **Neu | Bahn**.
   Daraufhin wird eine neue Bahn zum Pool hinzugefügt. Jede Bahn kann einen eigenen Namen erhalten. Doppelklicken Sie dazu in das Namensfeld.

© 2016-2022 Altova GmbH



Anmerkung:

Wenn Sie mit der rechten Maustaste in eine der **Bahnen** klicken, können Sie jedes Element hinzufügen, das mit der neuen Option in einen Pool platziert werden kann.

9.3.2.4 Artefakte

Artefakte dienen dazu, zusätzliche Informationen über einen Prozess anzuzeigen, z.B. wie Daten, Dokumente und andere Objekte im Rahmen eines Geschäftsprozesses verwendet und aktualisiert werden. Artefakte stehen in keinem direkten Zusammenhang zum Sequenz- oder Nachrichtenfluss des Prozesses.

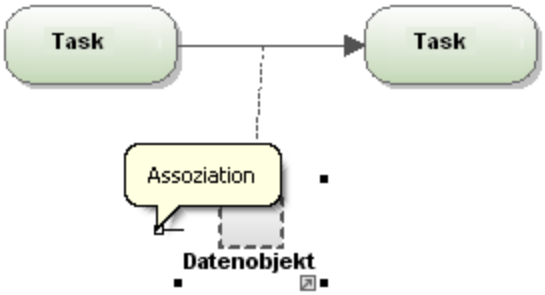
Datenobjekt

Datenobjekte sind Dokumente oder andere Arten von Daten, die zeigen, wie Daten bei einem Geschäftsprozess verwendet werden. Datenobjekte können dazu verwendet werden, um die Ein- und Ausgabe von Daten in/aus Aktivitäten zu definieren.



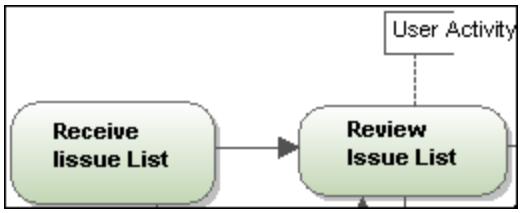
So erstellen Sie eine Assoziation zwischen einem Datenobjekt und einer Flusskontrolle:

1. Klicken Sie auf den Assoziationsziehpunkt des Datenobjekts (auf der linken Seite des Objekts).
2. Ziehen Sie den Konnektor auf das Flusskontrollobjekt, welches markiert erscheint, wenn Sie die Maustaste loslassen können.



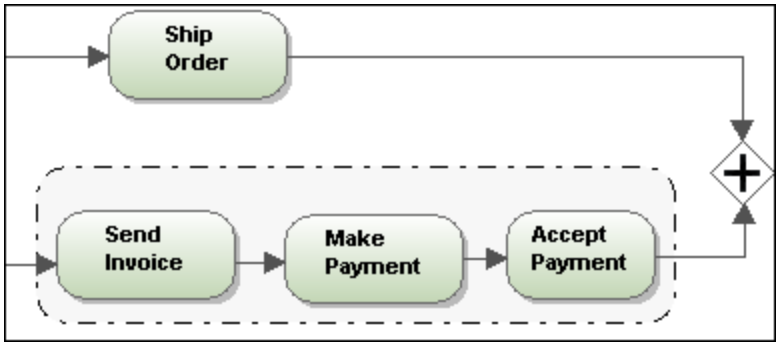
Textannotation

Mit Hilfe von Textannotationen können Sie Anmerkungen zu verschiedenen Abschnitten des Geschäftsprozesses machen. Textannotationen sind mit dem jeweiligen Objekt über eine Assoziation verbunden.



Gruppe

Gruppen werden oft verwendet, um bestimmte Abschnitte eines Diagramms selbst über verschiedene Pools hinweg zu markieren. Gruppen können nicht mit einem Sequenz- oder Nachrichtenfluss verbunden werden. Gruppenobjekte werden im Allgemeinen hinter Task- oder Prozessobjekte im Diagramm platziert.



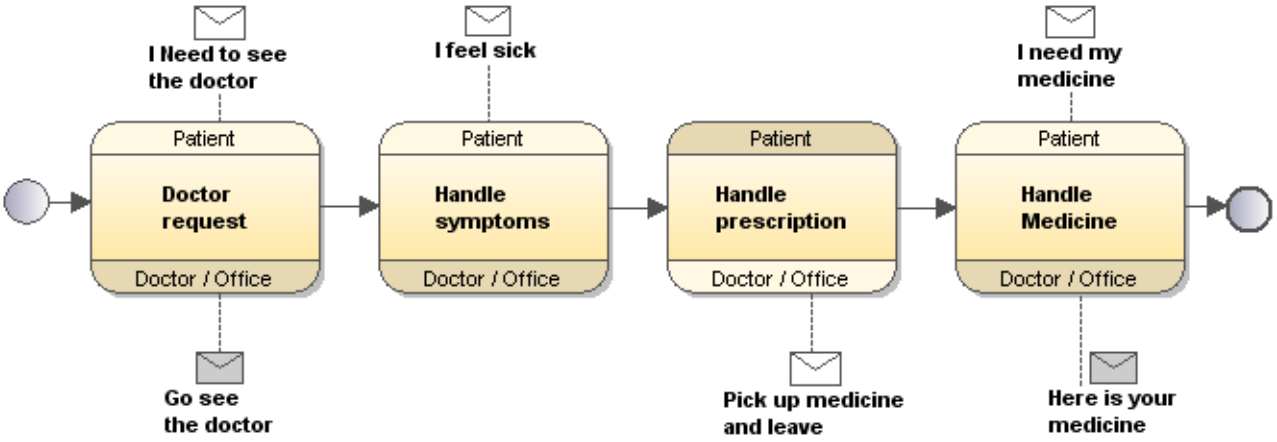
9.3.2.5 Choreographiediagramm

In Choreographiediagrammen wird definiert, wie Prozessbeteiligte ihre Interaktionen **koordinieren**. Man könnte Choreographiediagramme auch als einen geschäftlichen Vertrag zwischen Prozessbeteiligten sehen, dessen Schwerpunkt auf dem Austausch von Informationen (Nachrichten) zwischen den Beteiligten liegt.

Geschäftliche Verträge treten oft in folgender Form auf: Eine Bestellung wird an den Lieferanten gesendet, der Lieferant bestätigt den Eingang der Bestellung und schließlich wird die Bestellung ausgeführt. In Choreograpien werden Aktivitäten auch nach Sequenzflüssen geordnet.

Aktivitäten bestehen aus einer oder mehreren Interaktionen zwischen den verschiedenen Beteiligten. Interaktionen werden oft als Message Exchange Patterns (MEP) bezeichnet. Ein MEP ist die "Aktivität" einer Choreographie und kann auch als Choreographie-Task bezeichnet werden.

© 2016-2022 Altova GmbH



9.3.2.5.1 Choreographie-Tasks

Es gibt vier Arten von Choreographie-Tasks, die in das Diagramm eingefügt werden können:

- Choreographie-Task
- Subchoreographie-Task (eingeklappt)
- Subchoreographie-Task (aufgeklappt)
- Aufruf-Choreographie-Task

So fügen Sie einen Choreographie-Task ein:

1. Klicken Sie auf das Task-Symbol für den gewünschten Task, z.B. Choreographie-Task und klicken Sie anschließend in das Choreographiediagramm.



2. In der Abbildung sehen Sie die Standardansicht, wenn der Task eingefügt wird; der Text "Choreographie-Task" erscheint automatisch markiert.
3. Geben Sie Text ein, um den Choreographie-Task umzubenennen.
4. Klicken Sie in den oberen Bereich, um die Namen des Beteiligten A einzugeben und in den unteren Bereich, um den Namen des Beteiligten B einzugeben.
   Die Beteiligtenbereiche werden schattiert/nicht schattiert angezeigt. Der **Initiator** der Aktivität ist der nicht schattierte Beteiligte, in diesem Fall ist dies beim erstmaligen Einfügen des Task der Beteiligte A.

So fügen Sie Nachrichten zu einem Choreographie-Task hinzu bzw. verknüpfen diese:

1. Klicken Sie in der Symbolleiste auf die Schaltfläche "Nachricht"  und anschließend in das Diagramm, um die Nachricht einzufügen.

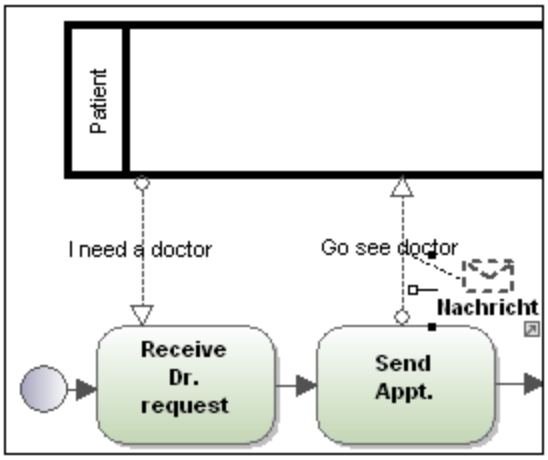


2. Geben Sie den Namen der Nachricht ein, z.B. "Ich muss zum Arzt".
3. Klicken Sie auf den Assoziationsziehpunkt (auf der linken Seite) und ziehen Sie ihn auf den Choreographie-Task, mit dem Sie ihn verknüpfen möchten.



So fügen Sie eine Nachricht zu einer Linie, z.B. einer Assoziation, hinzu:

1. Klicken Sie auf die **Linie**, zu der Sie eine Nachricht hinzufügen möchten.
2. Klicken Sie in der Symbolleiste auf die Schaltfläche "Nachricht".
3. Klicken Sie auf dieselbe Linie, um die Nachricht anzuhängen.



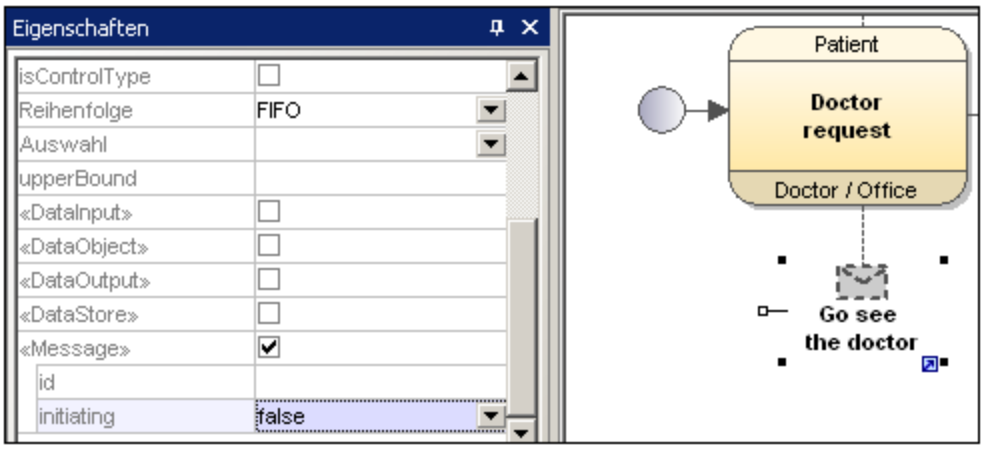
Die Nachricht wird oberhalb der Linie platziert und automatisch angehängt.

© 2016-2022 Altova GmbH

So ändern Sie die Ausgangsnachricht / den initiierenden Beteiligten:

- Wenn Sie eine **Nachricht** hinzufügen, wird sie automatisch als Ausgangsnachricht definiert, d.h. sie ist nicht schattiert.

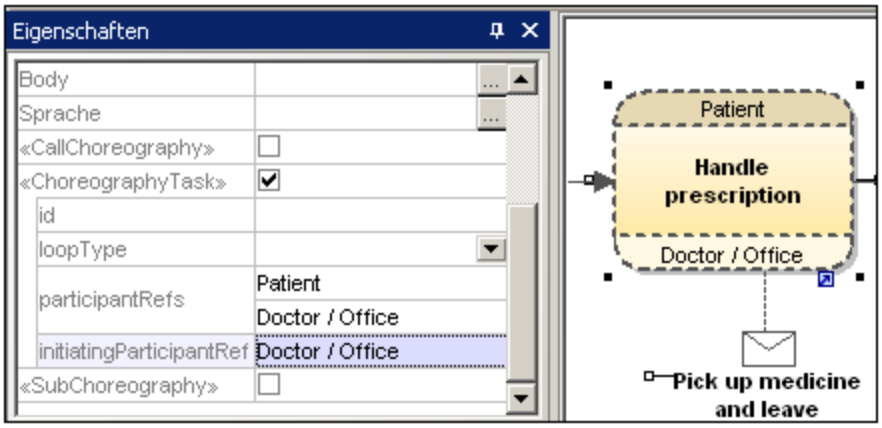
1. Klicken Sie auf die Nachricht und wählen Sie auf dem Register "Eigenschaften" in der Auswahlliste "initiating" den Eintrag **false.**



Das Nachrichtenelement erscheint nun schattiert.

- Wenn ein **Choreographie-Task** eingefügt wird, wird der Beteiligte A automatisch als initiierender Beteiligter definiert.

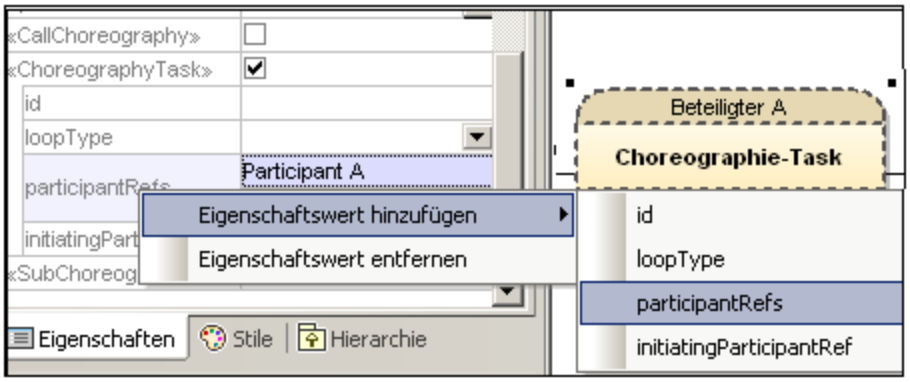
1. Klicken Sie auf den Choreographie-Task, der den Beteiligten enthält, den Sie als initiierenden Beteiligten definieren möchten.
2. Geben Sie in der Auswahlliste "InitiatingParticipantRef" den Beteiligten ein, den Sie als Initiator definieren möchten, z.B. "Arzt / Büro".



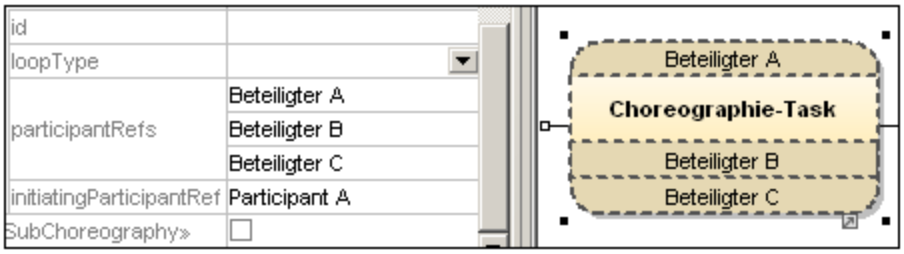
Der Bereich Arzt / Büro wird nun unschattiert angezeigt, d.h. es handelt sich hierbei um den initiierenden Beteiligten. Der Bereich "Patient" erscheint nun schattiert.

So fügen Sie neue Beteiligte zu einem Choreographie-Task hinzu:

1. Klicken Sie auf den Task, zu dem Sie den Beteiligten im Diagrammfenster hinzufügen möchten.

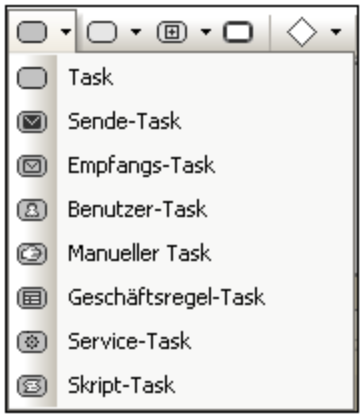


2. Klicken Sie mit der rechten Maustaste auf dem Register "Eigenschaften" in das Feld **participantsRefs** und wählen Sie den Befehl **Eigenschaftswert hinzufügen | participantRefs**.
3. Geben Sie den Namen des neuen Beteiligten ein, z.B. Beteiligter C.



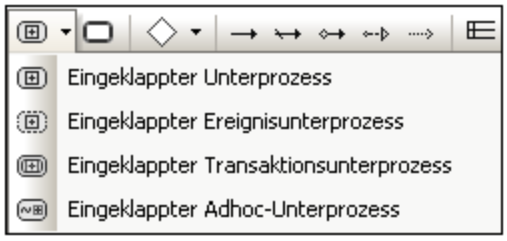
9.3.2.5.2 Tasks und Unterprozesse

Klicken Sie auf das Dropdown-Symbol der Schaltfläche "Task", um den entsprechenden Task einzufügen.

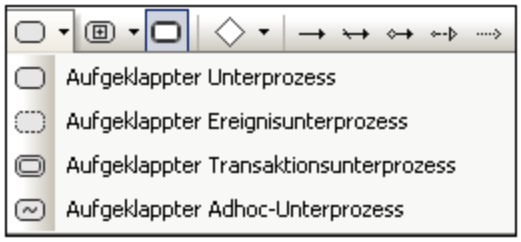


© 2016-2022 Altova GmbH

Klicken Sie auf das Dropdown-Symbol der Schaltfläche "Eingeklappter Unterprozess", um den entsprechenden eingeklappten Unterprozess einzufügen.



Über das Dropdown-Symbol der Schaltfläche "Aufgeklappter Unterprozess" können Sie den entsprechenden aufgeklappten Unterprozess einfügen.



9.3.2.5.3 Datenobjekte

Daten werden durch fünf Modellierungselemente dargestellt, die durch Klicken auf eine der folgenden Schaltflächen eingefügt werden:



Datenobjekt

Repräsentiert Informationen, die durch den Prozess hindurchfließen, z.B. E-Mails, Geschäftsdokumente, usw. Datenobjekte liefern Informationen darüber, welche Aktivitäten durchgeführt werden müssen und/oder was sie erzeugen.



Datenoutput

Repräsentiert das Ergebnis des Prozesses.



Dateninput

Ist ein externer Input für den gesamten Prozess. Kann als Aktivität gelesen werden.



Datenliste

Repräsentiert eine Liste von Informationen, z.B. eine Liste von Bestellartikeln.

Datenspeicher

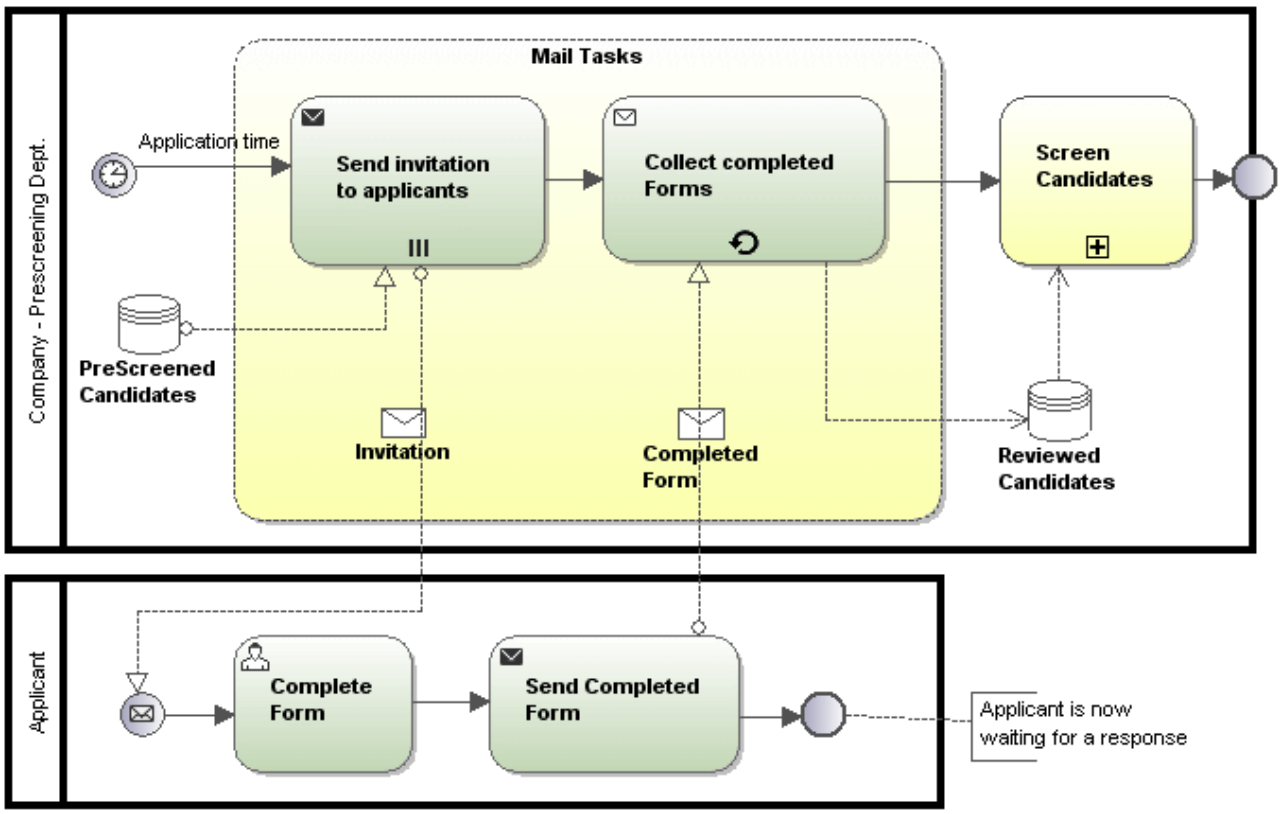
Ein Ort, von wo der Prozess Daten auslesen kann, bzw. wo Daten hingeschrieben werden können, z.B. eine Datenbank.

9.3.2.6 Kollaborationsdiagramm

In Kollaborationsdiagrammen werden die **Interaktionen** zwischen zwei oder mehreren Prozessen definiert.

Eine Kollaboration besteht im Allgemeinen aus zwei oder mehr Pools, die die Beteiligten in der Kollaboration repräsentieren. Nachrichten, die zwischen den Beteiligten ausgetauscht werden, werden in Form von Nachrichtenflüssen dargestellt, die die beiden Pools oder die Objekte innerhalb dieser Pools verbinden. Pools können auch leer sein. In diesem Fall werden sie als schwarze Kästen (Black Box) dargestellt.

In einem Kollaborationsdiagramm können alle Kombinationen von Pools, Prozessen und einer Choreographie verwendet werden.



9.3.2.6.1 Konversationen

Eine Konversation ist eine vereinfachte Version einer Kollaboration. Es stehen dieselben Modellierungselemente wie in einer Kollaboration zur Verfügung. In einer Konversation wird eine Gruppe logisch

© 2016-2022 Altova GmbH

miteinander in Zusammenhang stehender Nachrichtenaustauschszenarien definiert, wobei die ausgetauschten Nachrichten im Zusammenhang mit einem speziellen Geschäftsszenario stehen, z.B. eine Anfrage (Request) gefolgt von einer Antwort (Response).

In einer Konversation stehen zwei weitere grafische Elemente zur Verfügung, die in keinem anderen BPMN-Diagramm verfügbar sind:

- Konversations-Node-Elemente (Konversation, Unterkonversation und Aufrufkonversation)
- Konversationsbeziehungen

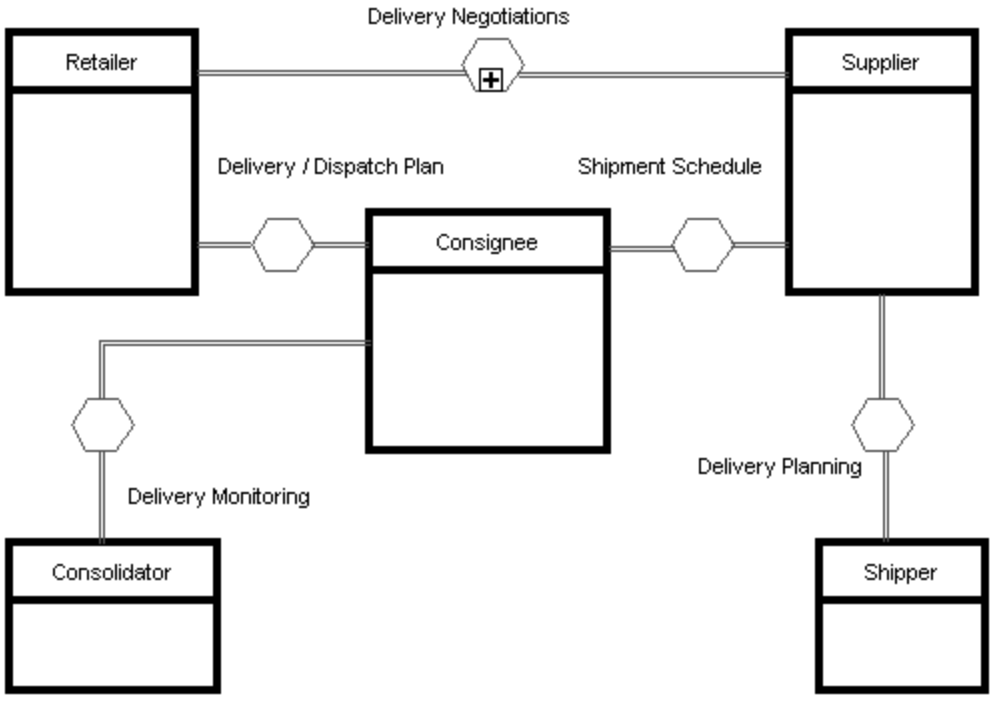
 Konversation

 Unterkonversation

 Aufrufkonversation

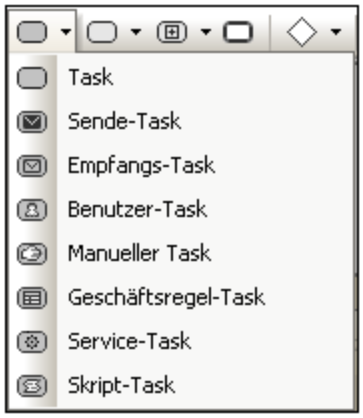
 Beteiligter / Pool

 Konversationsbeziehung

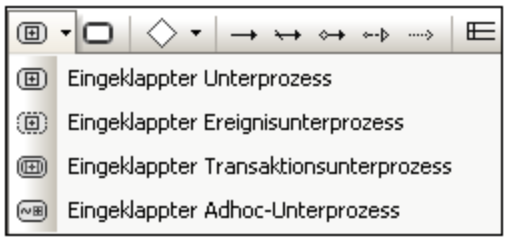


9.3.2.6.2 Tasks und Unterprozesse

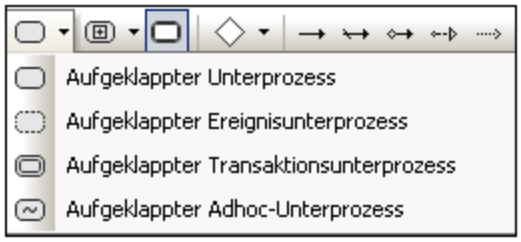
Klicken Sie auf das Dropdown-Symbol der Schaltfläche "Task", um den entsprechenden Task einzufügen.



Klicken Sie auf das Dropdown-Symbol der Schaltfläche "Eingeklappter Unterprozess", um den entsprechenden eingeklappten Unterprozess einzufügen.



Über das Dropdown-Symbol der Schaltfläche "Aufgeklappter Unterprozess" können Sie den entsprechenden aufgeklappten Unterprozess einfügen.



9.3.2.6.3 Datenobjekte

Daten werden durch fünf Modellierungselemente dargestellt, die durch Klicken auf eine der folgenden Schaltflächen eingefügt werden:

Datenobjekt

Repräsentiert Informationen, die durch den Prozess hindurchfließen, z.B. E-Mails, Geschäftsdokumente, usw. Datenobjekte liefern Informationen darüber, welche Aktivitäten durchgeführt werden müssen und/oder was sie erzeugen.

© 2016-2022 Altova GmbH

 Datenoutput

Repräsentiert das Ergebnis des Prozesses.

 Dateninput

Ist ein externer Input für den gesamten Prozess. Kann als Aktivität gelesen werden.

 Datenliste

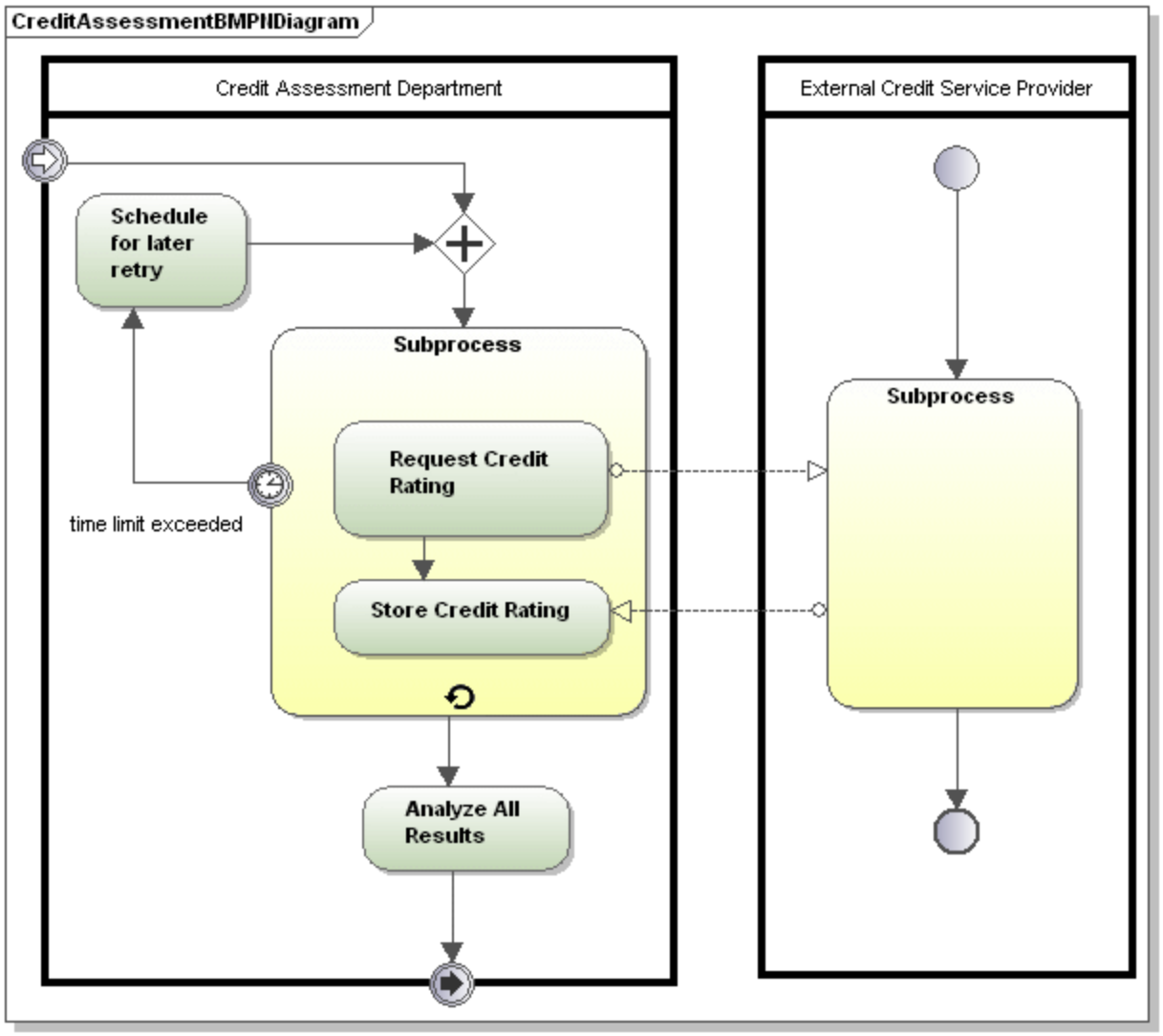
Repräsentiert eine Liste von Informationen, z.B. eine Liste von Bestellartikeln.

 Datenspeicher

Ein Ort, von wo der Prozess Daten auslesen kann, bzw. wo Daten hingeschrieben werden können, z.B. eine Datenbank.

9.3.2.7 Standard-Geschäftsprozessdiagramm BPMN 2.0

Mit Geschäftsprozessdiagrammen wird eine große Bandbreite an Informationen abgedeckt. Zur Erstellung von Geschäftsprozessen wird eine ganze Reihe verschiedener Modellierungsarten verwendet.



Es gibt drei Arten von Geschäftsprozessen:

Private nicht ausführbare (interne) Geschäftsprozesse:

- Nicht ausführbare Prozesse sind Prozesse, zu denen es nicht genug Informationen gibt, um sie ausführen zu können; im Allgemeinen ist dies während der Entwicklungsphase der Fall.

Private ausführbare (interne) Geschäftsprozesse:

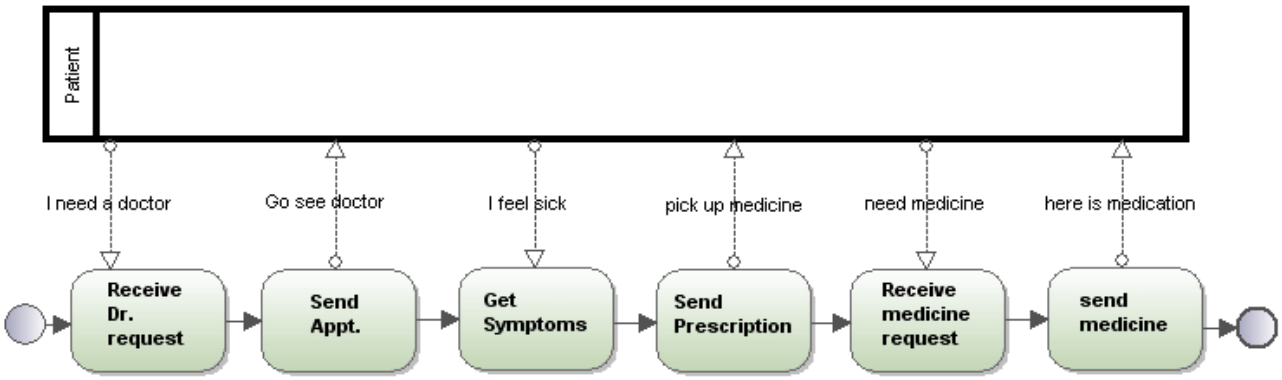
- Ausführbare Prozesse sind Prozesse, die aufgrund dessen, dass sie zur Gänze der BPMN 2.0-Semantik gemäß modelliert wurden, ausgeführt werden können.



© 2016-2022 Altova GmbH

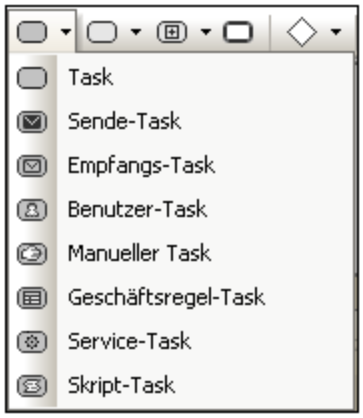
Öffentliche Prozesse:

- Definieren die Interaktion zwischen einem privaten Prozess und einem separaten Prozess oder Beteiligten, z.B. Arzt-Patient-Interaktionen.

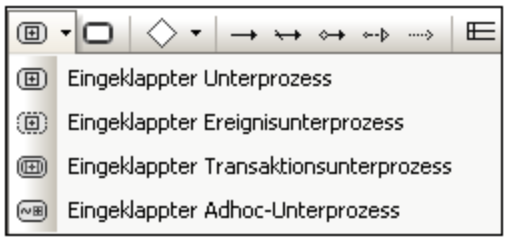


9.3.2.7.1 Tasks und Unterprozesse

Klicken Sie auf das Dropdown-Symbol der Schaltfläche "Task", um den entsprechenden Task einzufügen.

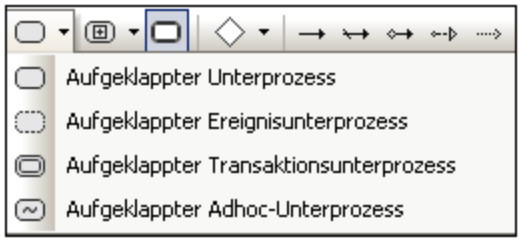


Klicken Sie auf das Dropdown-Symbol der Schaltfläche "Eingeklappter Unterprozess", um den entsprechenden eingeklappten Unterprozess einzufügen.



Über das Dropdown-Symbol der Schaltfläche "Aufgeklappter Unterprozess" können Sie den entsprechenden

aufgeklappten Unterprozess einfügen.



9.3.2.7.2 Datenobjekte

Daten werden durch fünf Modellierungselemente dargestellt, die durch Klicken auf eine der folgenden Schaltflächen eingefügt werden:



Datenobjekt

Repräsentiert Informationen, die durch den Prozess hindurchfließen, z.B. E-Mails, Geschäftsdokumente, usw. Datenobjekte liefern Informationen darüber, welche Aktivitäten durchgeführt werden müssen und/oder was sie erzeugen.



Datenoutput

Repräsentiert das Ergebnis des Prozesses.



Dateninput

Ist ein externer Input für den gesamten Prozess. Kann als Aktivität gelesen werden.



Datenliste

Repräsentiert eine Liste von Informationen, z.B. eine Liste von Bestellartikeln.



Datenspeicher

Ein Ort, von wo der Prozess Daten auslesen kann, bzw. wo Daten hingeschrieben werden können, z.B. eine Datenbank.

9.3.3 SysML-Diagramme

Altova Website: Modellieren von SysML-Diagrammen in UModel

SysML ist eine grafische Modellierungssprache, die die Analyse, Spezifizierung, die Erstellung, Überprüfung und Validierung von Systemen wie unter anderem Hardware, Software, Daten, Prozeduren unterstützt. Sie

© 2016-2022 Altova GmbH

können SysML-Diagramme in UModel von Grund auf neu erstellen oder bestehende SysML-Modell über XMI importieren oder exportieren, siehe Austausch von Metadaten zwischen XMI und XML 663.

Die nachstehende Tabelle enthält eine Liste der in SysML verfügbaren Diagramme.

Art	Diagramm	Anmerkungen	Abkürzung
Strukturdiagramme	Blockdefinitionsdiagramm 538	Wird über UML geändert	bdd
	Internes Blockdiagramm 541	Wird über UML geändert	ibd
	Paketdiagramm 547	Wird über UML wiederverwendet	pkg
	Zusicherungsdiagramm 546	SysML-spezifisch	par
Anforderungsdiagramme	Anforderungsdiagramm 549	SysML-spezifisch	req
Verhaltensdiagramme	Aktivitätsdiagramm 550	Wird über UML geändert	act
	Sequenzdiagramm 551	Wird über UML wiederverwendet	sd
	Zustandsdiagramm 552	Wird über UML wiederverwendet	stm
	Use Case-Diagramm 553	Wird über UML wiederverwendet	uc

Wie oben gezeigt, können SysML-Diagramme grob in Struktur-, Anforderungs- und Verhaltensdiagramme eingeteilt werden. Außerdem werden einige der SysML-Diagramme über UML wiederverwendet, einige über UML geändert, während einige rein SysML-spezifisch sind. Die Abkürzung für die einzelnen Diagramme wird standardmäßig in der linken oberen Ecke des Diagrammfensters 101 angezeigt, wenn Sie die Diagrammüberschrift nicht ausblenden.

Abgesehen von den Besonderheiten der einzelnen Diagramme unterscheidet sich die Erstellung von SysML-Projekten mit UModel nicht von der Erstellung von Standard-UModel-Projekten, siehe Erstellen, Öffnen und Speichern von Projekten 163. Unter dem folgenden Pfad finden Sie ein UModel-Beispielprojekt, das verschiedene SysML-Diagramme enthält: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_SysML.ump**.

Erstellen von SysML-Diagrammen

Um SysML-Diagramme zu erstellen, muss Ihr UModel-Projekt das *SysML-Profil*, ein vordefiniertes UModel-Profil, enthalten. Wenn Sie das erste SysML-Diagramm, wie unten beschrieben, zu Ihrem Projekt hinzufügen, werden Sie aufgefordert, dieses Profil zu inkludieren. Sie können das SysML-Profil auch explizit zu Ihrem Projekt hinzufügen, siehe Anwenden von UModel-Profilen 170.

So erstellen Sie ein SysML-Diagramm:

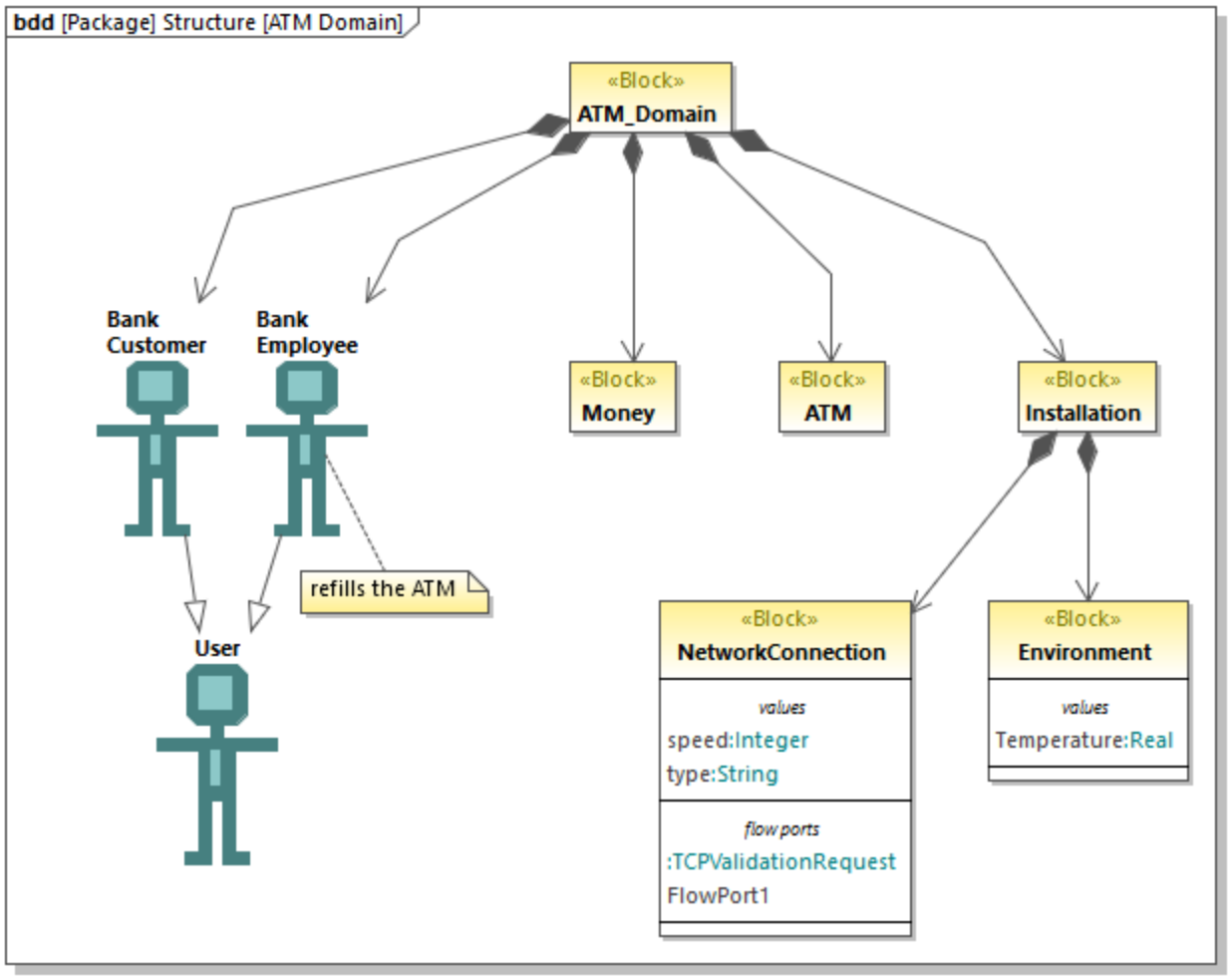
1. Wählen Sie eine der folgenden Methoden:
   a. Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" 86 auf ein Paket und wählen Sie im Kontextmenü **Neues Diagramm | SysML-Diagramme | <Diagrammart>**, wobei "Diagrammart" einer der SysML-Diagrammtypen ist.
   b. Klicken Sie mit der rechten Maustaste im Fenster "Diagramm-Struktur" 90 auf "Diagramme" oder "SysML-Diagramme" und wählen Sie im Kontextmenü **Neues Diagramm | <Diagrammart>**, wobei "Diagrammart" einer der SysML-Diagrammtypen ist. Darauf wird ein Dialogfeld angezeigt, in dem Sie aufgefordert werden, den Owner (Eigentümer) des Diagramms auszuwählen. Wählen Sie ein Paket für das Diagramm aus und klicken Sie auf **OK**.
2. Wenn das aktuelle UModel-Projekt das SysML-Profil nicht enthält, wird ein Dialogfeld geöffnet, in dem Sie aufgefordert werden, es zu inkludieren. Klicken Sie auf **OK**, um das SysML-Diagramm in das aktuelle Projekt zu inkludieren, siehe auch Anwenden von UModel-Profilen 170.

Anmerkung: Wenn Sie das "Root"-Paket in Schritt 1 ausgewählt haben, werden SysML-Diagramme in ihrem eigenen "SysML"-Paket erstellt.

9.3.3.1 Blockdefinitionsdiagramm

Blockdefinitionsdiagramme basieren auf UML-Klassendiagrammen 449 und haben zusätzlich in SysML definierte Einschränkungen und Erweiterungen. Blockdefinitionsdiagramme enthalten Strukturelemente - sogenannte "Blöcke" und deren Beziehungen, wie z.B. Assoziationen, Generalisierungen und Abhängigkeiten.

© 2016-2022 Altova GmbH



Blockdefinitionsdiagramm

Blöcke sind grundlegende Einheiten zur Beschreibung von Strukturen in SysML; sie ähneln Klassen in UML-Klassendiagrammen. Blöcke können Komponenten wie Teile (parts), Operationen (operations), Eigenschaften (properties) und Ports (ports) enthalten. Bei Eigenschaften kann es sich um spezielle Eigenschaften wie z.B. eine **PartProperty**, eine **ReferenceProperty** oder eine **ValueProperty** handeln.

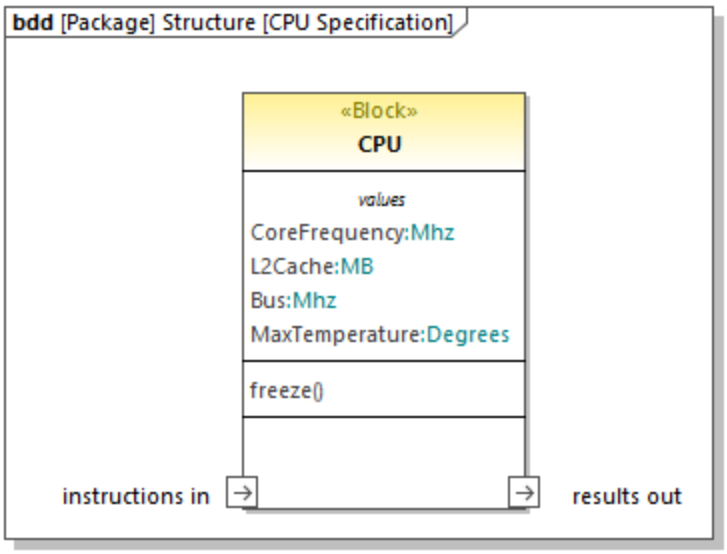
So erstellen Sie einen Block:

1. Erstellen Sie ein neues Blockdefinitionsdiagramm, siehe Erstellen von SysML-Diagrammen 537.
2. Wählen Sie eine der folgenden Methoden:
   - Klicken Sie mit der rechten Maustaste auf einen leeren Bereich im Diagramm und wählen Sie im Kontextmenü den Befehl **Neu | Block**.
   - Klicken Sie auf die Symbolleisten-Schaltfläche **Block** und anschließend in das Diagramm.

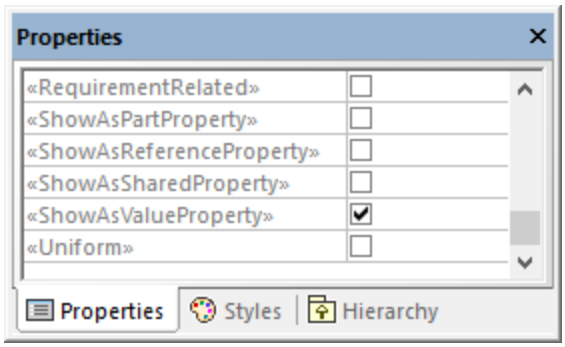
So fügen Sie eine Eigenschaft zu einem Block hinzu:

- Klicken Sie mit der rechten Maustaste auf einen bestehenden Block und wählen Sie im Kontextmenü den Befehl **Neu | Eigenschaft** (oder **PartProperty**, **ReferenceProperty**, **ValueProperty**).

  Daraufhin wird ein neuer Bereich zum Block hinzugefügt, z.B. "parts" für eine **PartProperty** oder "values" für eine **ValueProperty**.



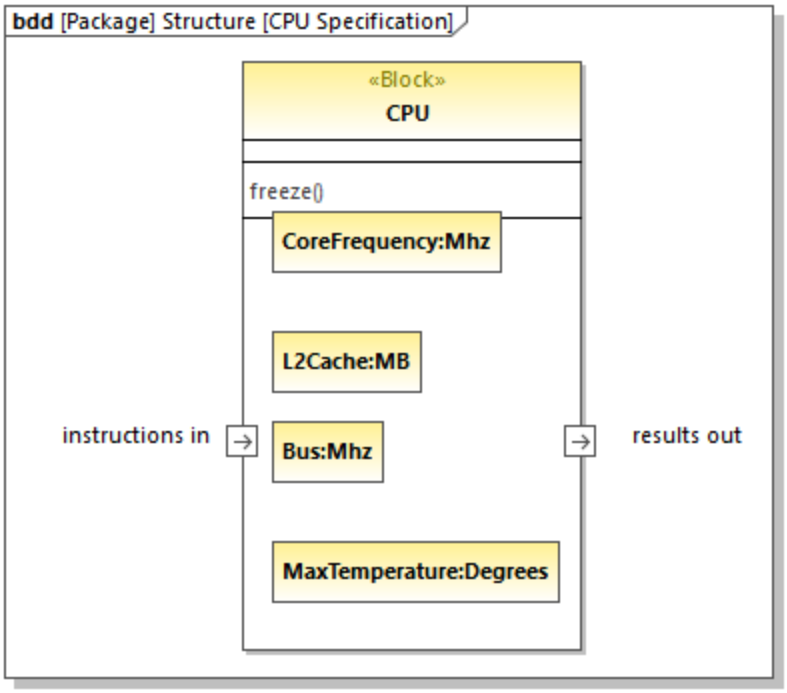
Sie können die Spezialisierung einer bestehenden Eigenschaft jederzeit ändern (So können Sie z.B. eine **PartProperty** in eine **ValueProperty** konvertieren). Wählen Sie dazu die Eigenschaft zuerst im Diagramm oder in der Modell-Struktur aus und aktivieren Sie anschließend im Fenster "Eigenschaften" das Kontrollkästchen mit dem entsprechenden Stereotyp, z.B:



So zeigen Sie Blockeigenschaften als Nodes (Knoten) an:

- Klicken Sie mit der rechten Maustaste auf einen Block und wählen Sie den Befehl **Anzeigen | Eigenschaften als Nodes in einem Node** anzeigen.

© 2016-2022 Altova GmbH



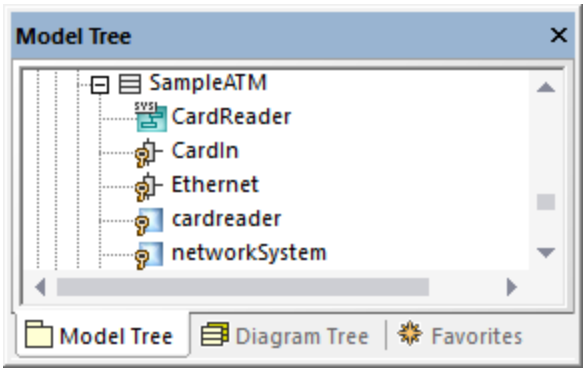
Um die obige Aktion rückgängig zu machen, klicken Sie mit der rechten Maustaste auf eine Eigenschaft (z.B. `Bus:Mhz` in der Abbildung oben) und wählen Sie im Kontextmenü den Befehl **Nur aus Diagramm löschen**.

9.3.3.2 Internes Blockdiagramm

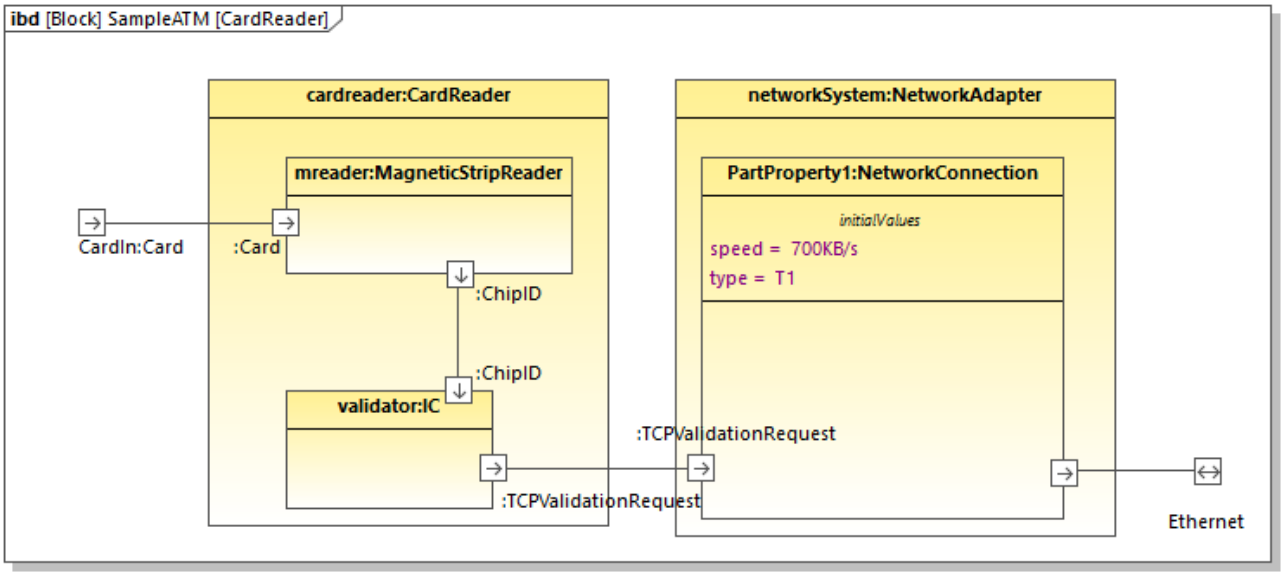
Interne Blockdiagramme basieren auf UML-Kompositionsstrukturdiagrammen[465] und weisen in SysML definierte Einschränkungen und Erweiterungen auf. Mit einem internen Blockdiagramm werden die interne Struktur eines Blocks und die Verbindungen zwischen seinen Bestandteilen mittels Ports, Konnektoren und Flüssen beschrieben. Normalerweise wird ein neues internes Blockdiagramm folgendermaßen erstellt:

- Klicken Sie mit der rechten Maustaste im Fenster "Modell-Struktur" auf einen vorhandenen Block und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Internes SysML Blockdiagramm**.

Wenn Sie ein neues internes Blockdiagramm erstellen, ohne zuerst mit der rechten Maustaste auf einen vorhandenen Block zu klicken, wird auch in der Modell-Struktur ein neuer Block erstellt und das neue Diagramm wird unterhalb dieses Blocks erstellt, da angenommen wird, dass es diesen beschreibt. So beschreibt etwas das Diagramm "CardReader" im Modell unten den Block "SampleATM".



Das Diagramm "CardReader" steht unter dem folgenden Demo-Projekt zur Verfügung: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Bank_SysML.ump**.



CardReader-Diagramm

Die oben gezeigten Eigenschaften **`cardreader`** und **`networksystem`** haben den Typ `CardReader` bzw. `NetworkAdapter`. Diese Typen sind im selben Modell vorhanden. Es handelt sich dabei um Blöcke, wovon das Aussehen der Eigenschaften im Diagramm abhängt. Beachten Sie, dass Sie den Typ einer Eigenschaft im Fenster "Eigenschaften" 92 über die Dropdown-Liste **Typ** definieren bzw. ändern können.

Anfangswerte

Wenn eine Eigenschaft einen Typ hat, der, wie in diesem Beispiel, ein "Block" ist, kann sie mit Anfangswerten erstellt werden. So hat etwa die Eigenschaft **`PartProperty1`** im Diagramm **CardReader** den Anfangswert `speed = 700KB/s`.

So fügen Sie Anfangswerte zu einer Eigenschaft hinzu:

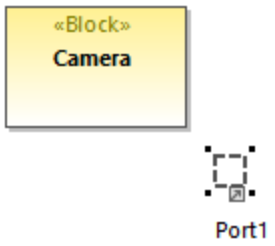
1. Klicken Sie mit der rechten Maustaste auf die Eigenschaft und wählen Sie **Neu | Anfangswerte**.

© 2016-2022 Altova GmbH

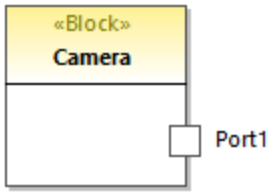
2. Doppelklicken Sie auf den Platzhalter und geben Sie die Werte ein (z.B. `speed = 700KB/s`).

Standard-Ports

Um einen Standard-Port hinzuzufügen, klicken Sie auf die Symbolleisten-Schaltfläche **Port** und anschließend auf das Diagramm. Daraufhin wird der Port zum Diagramm hinzugefügt.



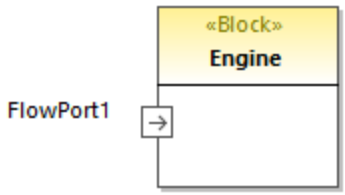
Um den Port an einen Block anzuhängen, ziehen Sie ihn über die Umrandung des Blocks (in diesem Beispiel "Camera") bis diese markiert erscheint und lassen Sie die Maustaste dann los. Der Port wird nun an die Umrandung des Blocks angehängt.



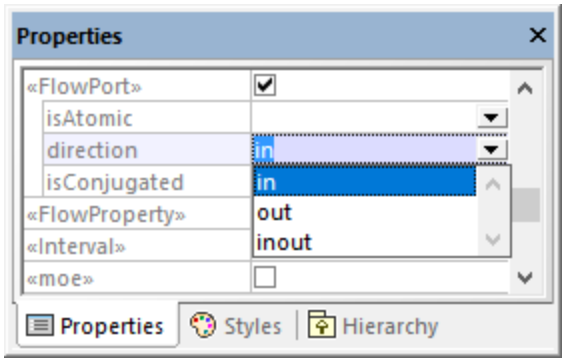
Um den Namen und Typ des Blocks zu ändern, wählen Sie den Port zuerst im Diagramm aus und ändern Sie anschließend im Fenster "Eigenschaften" 92 die Eigenschaften **Name** und **Typ**.

Flow Ports

Um einen Flow Port zu erstellen, klicken Sie auf die Symbolleisten-Schaltfläche **FlowPort** und anschließend auf die Umrandung eines Blocks. Der Flow Port wird nun an die Umrandung des Blocks angehängt. Sie können Flow Ports auch, wie oben für Standard-Ports gezeigt, in zwei separaten Schritten erstellen und anhängen.

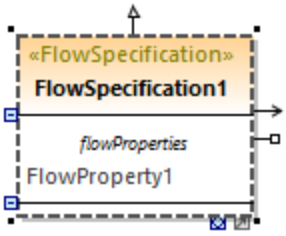


Durch Bearbeitung der entsprechenden Eigenschaften im Fenster "Eigenschaften" können Sie den Namen und Typ des Ports ändern. Beachten Sie, dass Flow Ports im Fenster "Eigenschaften" zusätzliche Eigenschaften haben, über die Sie die Richtung (direction) definieren können (z.B. in, out, inout).



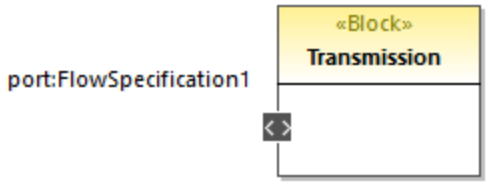
So erstellen Sie einen Atomic Conjugated Flow Port:

1. Erstellen Sie in einem Blockdefinitionsdiagramm (BDD) eine FlowSpecification (Interface).



2. Klicken Sie im internen Blockdiagramm (BDD) auf den gewünschten Flow Port.
3. Setzen Sie im Fenster "Eigenschaften" die Eigenschaft **Typ** auf die zuvor erstellte FlowSpecification.
4. Setzen Sie im Fenster "Eigenschaften" die Eigenschaft **isConjugated** auf `true`.

Ein Atomic Conjugated Port wird mit einem dunklen Hintergrund angezeigt.



Verbinden von Ports

So können Sie zwei Ports miteinander verbinden:

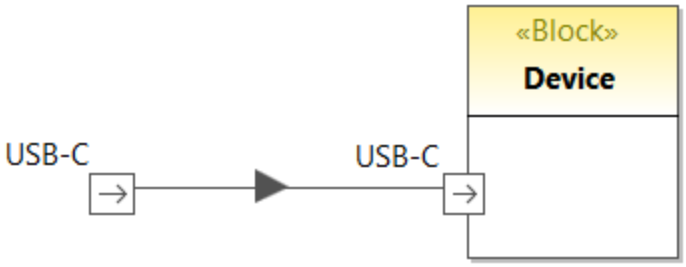
1. Klicken Sie auf die Symbolleisten-Schaltfläche **Konnektor** .
2. Ziehen Sie den Konnektor vom ersten zum zweiten Port.
3. Lassen Sie die Maustaste los, wenn das Port-Objekt im Diagramm markiert angezeigt wird.

© 2016-2022 Altova GmbH

Objektflüsse

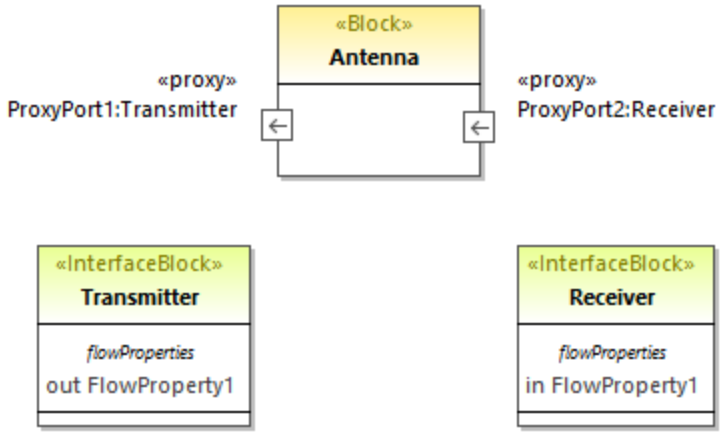
Objektflüsse können zwischen Blockassoziationen oder an anderen Konnektoren zwischen Bestandteilen von SysML-Diagrammen erstellt werden.

Um Objektflüsse zu erstellen, klicken Sie mit der rechten Maustaste auf einen bestehenden Konnektor und wählen Sie im Kontextmenü den Befehl **Neu | Objektfluss (L nach R, oder R nach L)**. Daraufhin wird eine Pfeilspitze zum Konnektor hinzugefügt, um die Richtung des Objektflusses anzuzeigen.



Proxy-Ports und Richtung

In neueren Versionen von SysML können Proxy-Ports ähnlich Flow-Ports älterer SysML-Versionen die Richtung anzeigen. So besteht etwa das unten gezeigte Diagramm aus einem Block ("Antenna") mit zwei Proxy-Ports, die die Richtung anzeigen.



Hier finden Sie ein Beispiel dazu, wie Sie eine Richtung zu Proxy-Ports hinzufügen:

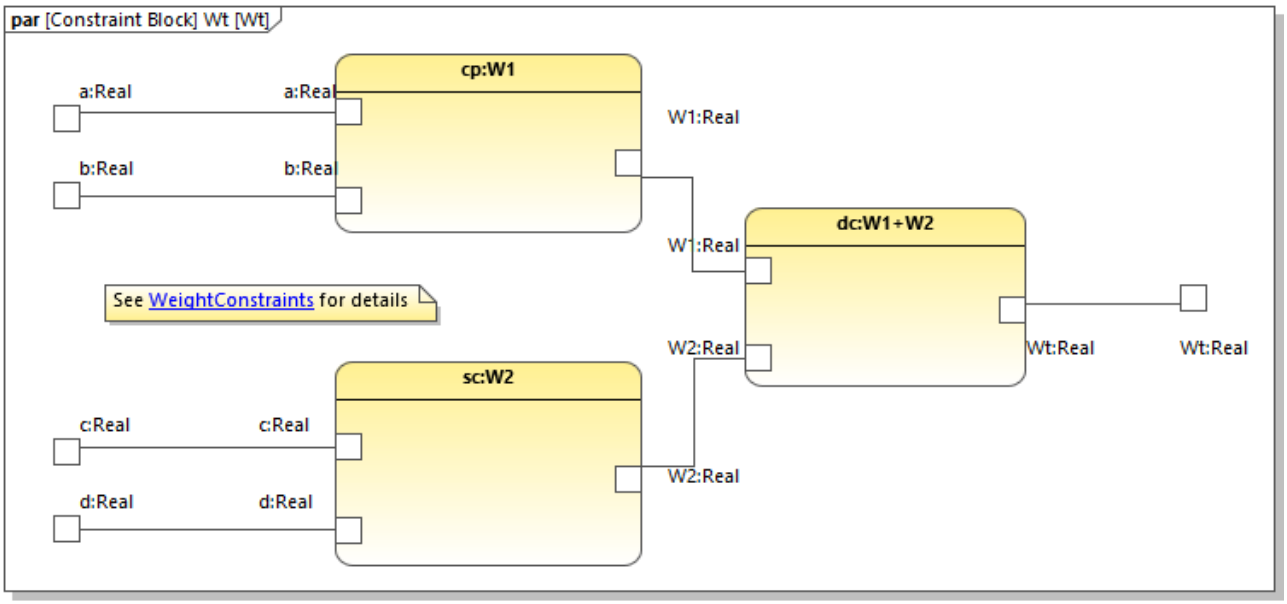
1. Fügen Sie einen Block und zwei Schnittstellenblöcke zum Diagramm hinzu. Der Block ist im Diagramm oben "Antenna", die beiden Schnittstellenblöcke sind "Transmitter" und "Receiver".
2. Wählen Sie den "Transmitter" aus und drücken Sie **F7**, um eine neue Fluss-Eigenschaft dazu hinzuzufügen.
3. Fügen Sie den Proxy-Port zum Block hinzu und ändern Sie seinen Typ über das Fenster "Eigenschaften" in "Transmitter".

4. Wählen Sie die Fluss-Eigenschaft im Diagramm aus und ändern Sie die Eigenschaft "Richtung" über das Fenster "Eigenschaften" in **out**. Wie Sie sehen, ändert sich die Richtung des Proxy-Port entsprechend.
5. Wählen Sie den "Receiver" aus und drücken Sie **F7**, um eine neue Fluss-Eigenschaft dazu hinzuzufügen.
6. Fügen Sie einen zweiten Proxy-Port zum Block hinzu und ändern Sie seinen Typ über das Fenster "Eigenschaften" in "Receiver".
7. Wählen Sie die Fluss-Eigenschaft im Diagramm aus und ändern Sie die Eigenschaft "Richtung" über das Fenster "Eigenschaften" in **in**. Wieder ändert sich die Richtung des Proxy-Port entsprechend.

9.3.3.3 Zusicherungsdiagramm

Das Zusicherungsdiagramm ist ein SysML-spezifischer Diagrammtyp, der die Programmanalyse mit der Design-Modellierung integriert. Ein Zusicherungsdiagramm ähnelt einem internen Blockdiagramm[541], mit der Ausnahme, dass nur die Konnektortypen angezeigt werden dürfen, die an mindestens einem Ende mit Constraint-Parametern verbunden sind.

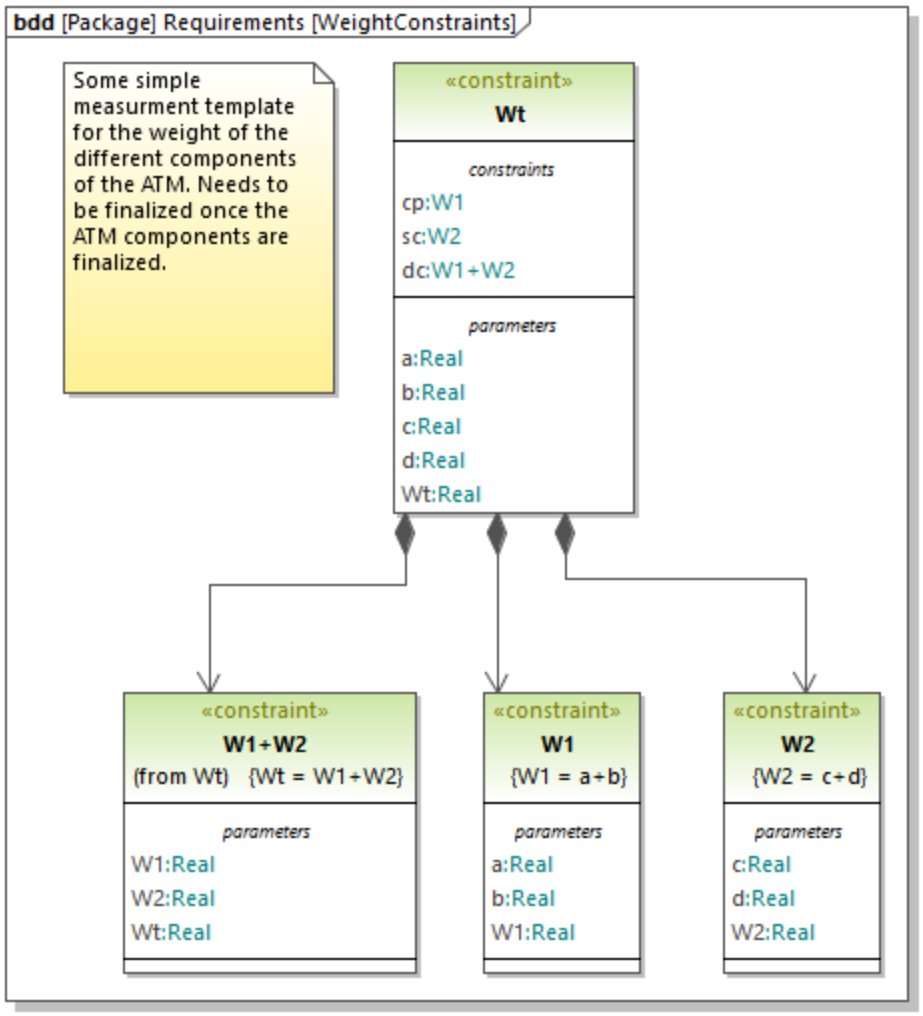
Im Zusicherungsdiagramm werden die Eigenschaften anderer Blöcke im Zusicherungsdiagramm mit Hilfe von in einem Blockdefinitionsdiagramm definierten Constraint-Blöcken eingeschränkt. Constraint-Blöcke werden zur Unterscheidung von normalen, eckig umrandeten Teilen mit abgerundeten Ecken angezeigt.



Zusicherungsdiagramm

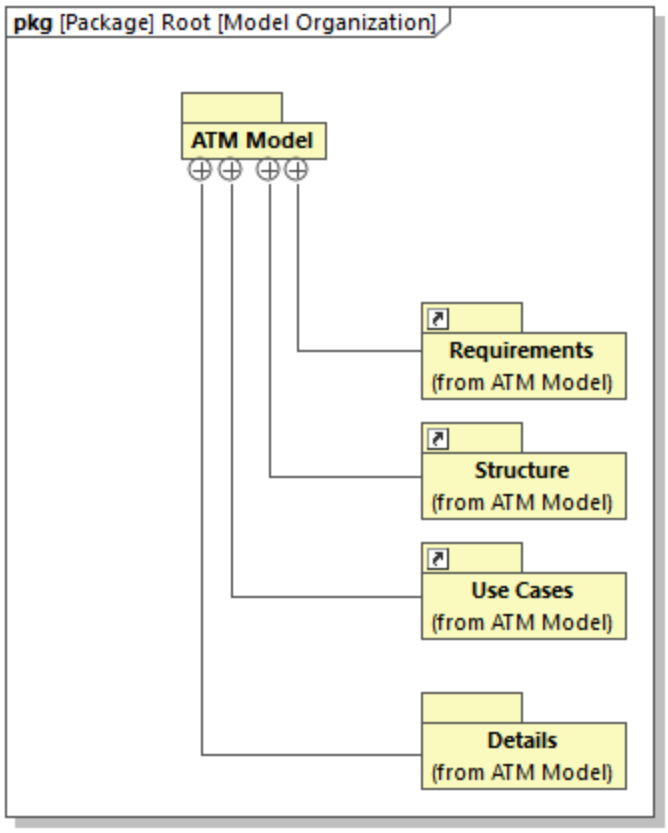
Das Stereotyp «constraint» in einem Block gibt an, dass es sich beim Block um einen Constraint-Block handelt. Die Parameter des Constraint werden in Blockdefinitionsdiagrammen in einem Bereich "parameters" angezeigt.

© 2016-2022 Altova GmbH



9.3.3.4 Paketdiagramm

Paketdiagramme dienen dazu, Modellelemente in verpackbare Elemente zu gliedern. Außerdem können Sie in solchen Diagrammen Abhängigkeiten zwischen Paketen und Modellelementen innerhalb des Pakets definieren. So wird etwa im Diagramm unten die grundlegende Struktur des Modells aus dem Demo-Projekt **Bank_SysML.ump** aus dem Verzeichnis **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples** dargestellt. Die Links für **Requirements**, **Structure** und **Use Cases** verweisen auf die entsprechenden Pakete im selben Modell, siehe auch Erstellen von Hyperlinks zu Elementen 122.



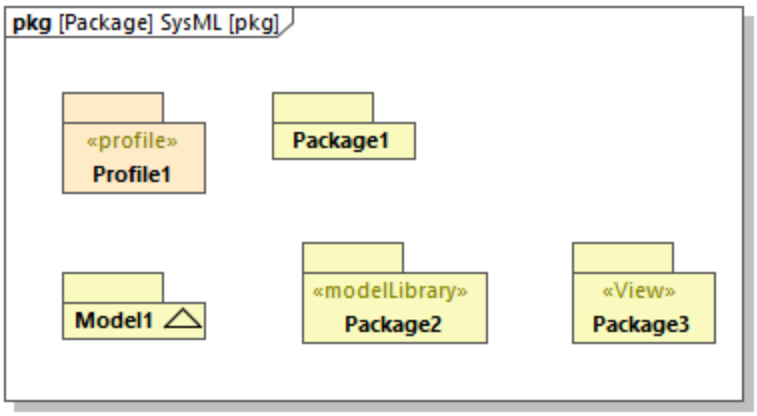
Paketdiagramm

Das oben gezeigte Paketdiagramm ist nur eine der Möglichkeiten, ein Modell zu strukturieren; Sie können ein Modell natürlich auch nach anderen Aspekten, z.B. nach Systemhierarchie oder Diagrammtyp gliedern.

In einem Paketdiagramm können Sie verschiedene Elemente wie gewohnt durch Klick auf die entsprechende Symbolleisten-Schaltfläche (wie z.B. **Paket**, **Profil** oder **Ansicht**) und anschließenden Klick ins Diagramm zum Diagramm hinzufügen. Beachten Sie jedoch, dass für einige Paketspezialisierungen eventuell keine Symbolleisten-Schaltflächen zur Verfügung stehen. In diesem Fall können Sie diese folgendermaßen hinzufügen:

1. Klicken Sie auf die Symbolleisten-Schaltfläche **Paket** und klicken Sie anschließend in das Diagramm, um das neue Paket hinzuzufügen.
2. Aktivieren Sie im Fenster "Eigenschaften" das Kontrollkästchen mit dem gewünschten Stereotyp (z.B. «ModelLibrary»).

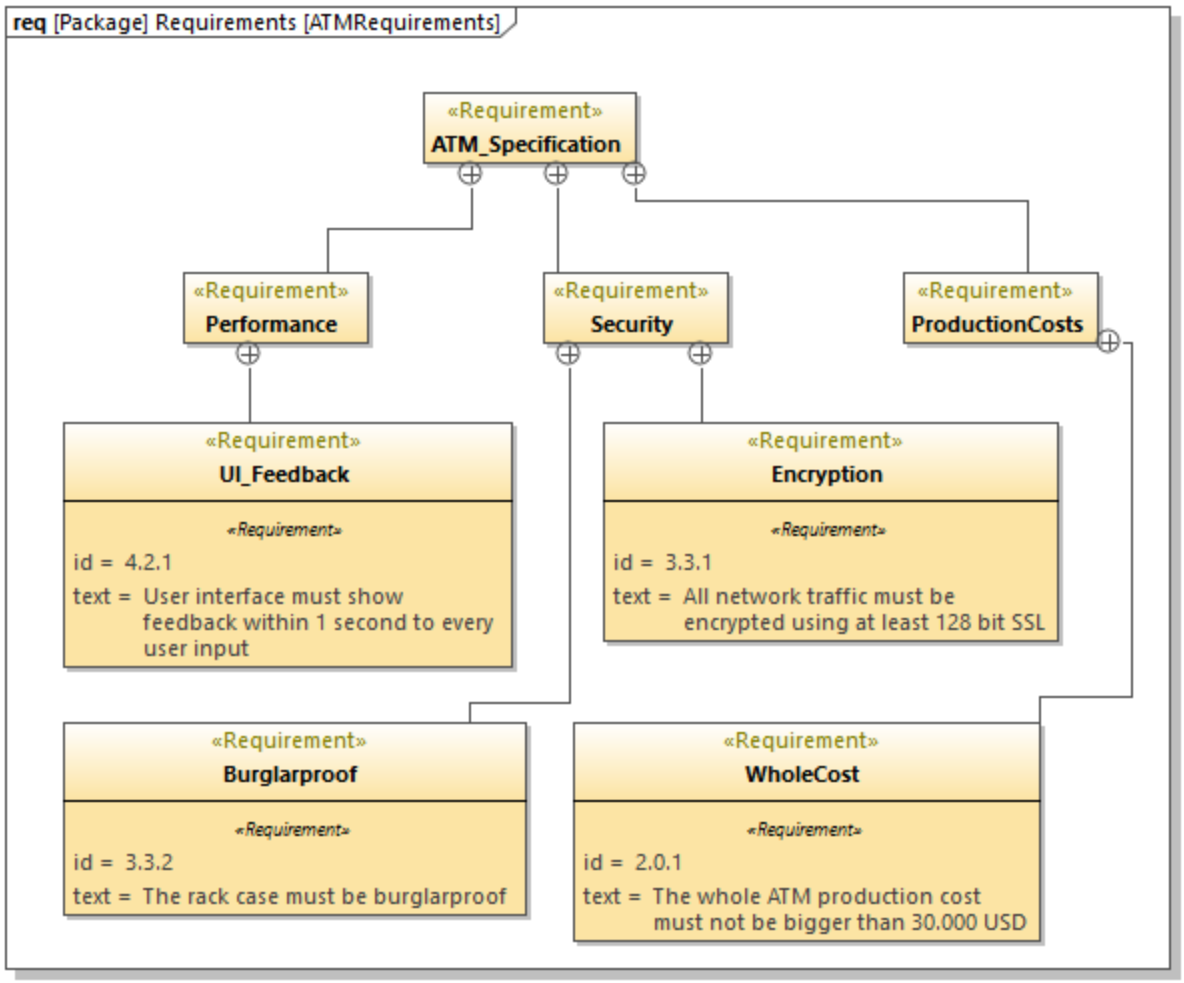
© 2016-2022 Altova GmbH



Im obigen Paketdiagramm hat **Package2** das Stereotyp `«ModelLibrary»` und **Package3** das Stereotyp `«View»`. Siehe auch Anwenden von Stereotypen 157.

9.3.3.5 Anforderungsdiagramm

Das Anforderungsdiagramm ist ein speziell für SysML konzipierter Diagrammtyp. Darin werden das Verhalten und die Strukturmodelle von SysML mit Softwareanalysemodellen wie z.B. Performanz- oder Zuverlässigkeitsmodellen integriert. Es ist ein Modell für textbasierte Anforderungen und die Beziehung zwischen Anforderungen und anderen Modellelementen, die diese erfüllen oder überprüfen.



Anforderungsdiagramm

Bei Anforderungsdiagrammen müssen Sie oft mehrere Textzeilen erstellen, damit die Anforderungsblöcke nicht zu groß werden.

So erstellen Sie mehrere Textzeilen:

1. Doppelklicken Sie auf den Text.
2. Halten Sie die **Strg**-Taste gedrückte, während Sie die **Eingabetaste** drücken.

9.3.3.6 Aktivitätsdiagramm

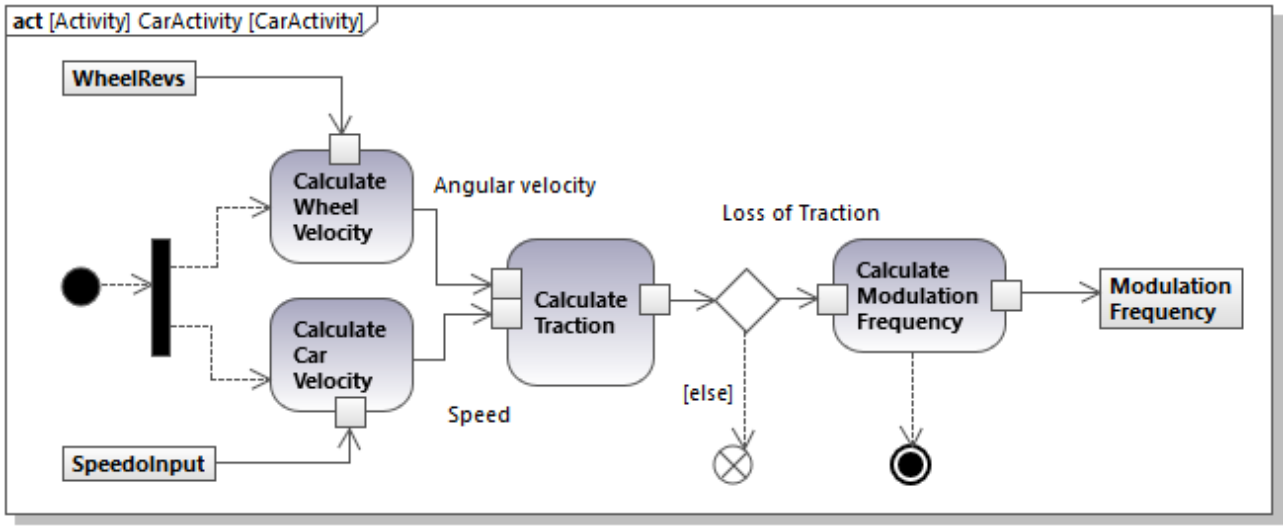
SysML-Aktivitätsdiagramme drücken Informationen über das dynamische Verhalten eines Systems wie z.B. Objektflüsse während des Systembetriebs aus. Solche Diagramme stellen die Reihenfolge, in der Aktionen durchgeführt werden, dar und beschreiben, welche der Strukturen eine bestimmte Aktion durchführt. Bei den Flüssen selbst kann es sich um Kontroll- oder Objektflüsse handeln. Beide Flussarten können mit Hilfe der entsprechenden Symbolleisten-Schaltfläche hinzugefügt werden:

© 2016-2022 Altova GmbH

→ **Kontrollfluss**

⤑ **Objektfluss**

Im unten gezeigten Aktivitätsdiagramm werden beide Arten von Flüssen verwendet. Kontrollflüsse werden als gestrichelte Linien, Objektflüsse als durchgezogene Linien dargestellt.

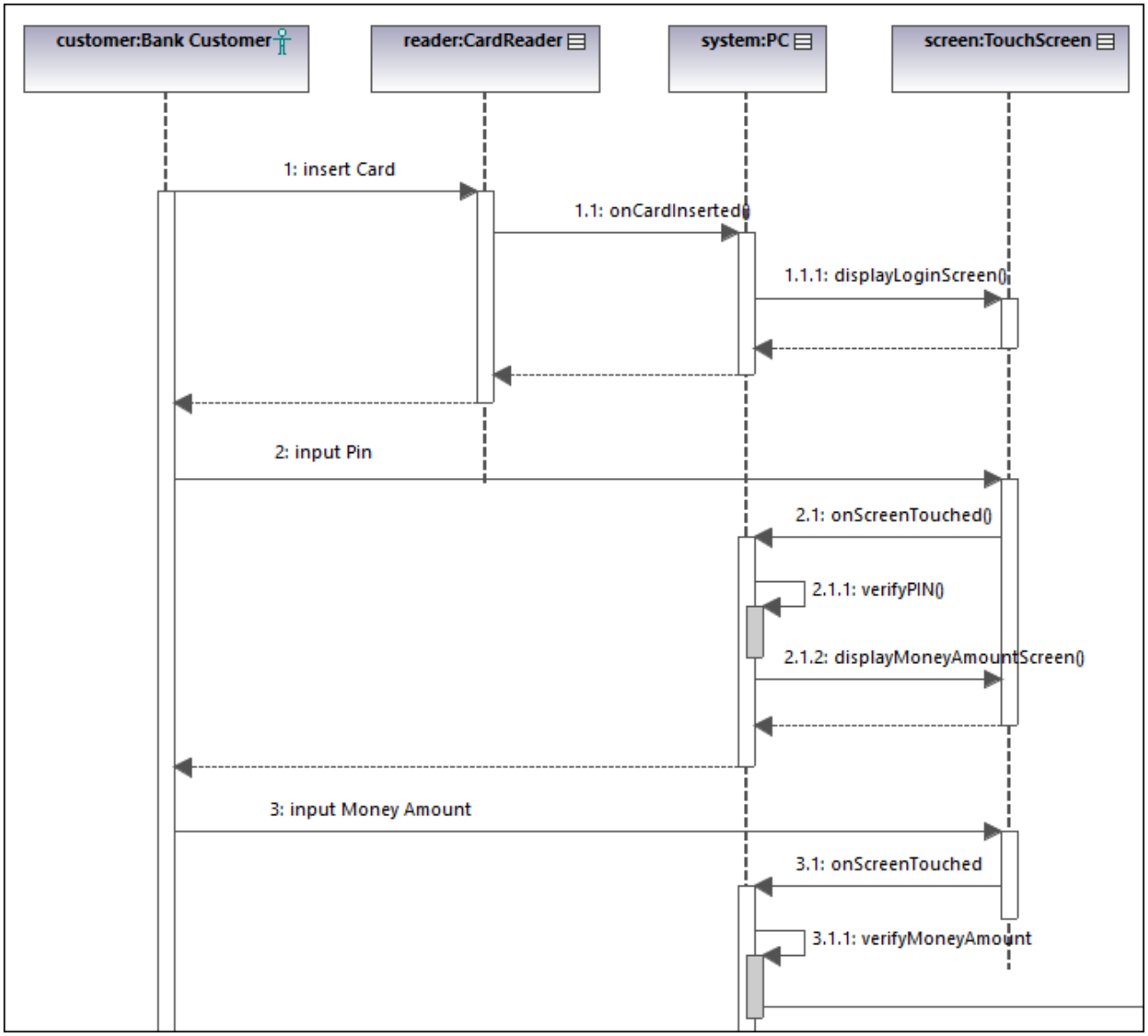


SysML-Aktivitätsdiagramm

Das SysML-Aktivitätsdiagramm ist ein verändertes UML-Diagramm mit SysML-Erweiterungen. Allgemeine Informationen zum Erstellen von UML-Aktivitätsdiagrammen mit UModel finden Sie unter Aktivitätsdiagramm 357
.

9.3.3.7 Sequenzdiagramm

Wie das Aktivitätsdiagramm beschreibt auch das SysML-Sequenzdiagramm das dynamische Verhalten eines Systems, jedoch mit größerer Genauigkeit. Es gibt nicht nur Aufschluss über die *Reihenfolge* von Aktionen und von welchen Strukturen die Aktionen ausgeführt werden, sondern auch über die Strukturen, die eine bestimmte Aktion *aufrufen*. Daher tendieren Sequenzdiagramme oft dazu, sehr komplex zu werden, außer sie konzentrieren sich auf ein ganz bestimmtes Szenario. In der Abbildung unten sehen Sie ein Fragment eines SysML-Sequenzdiagramms aus dem Beispielprojekt **Bank_SysML.ump**.



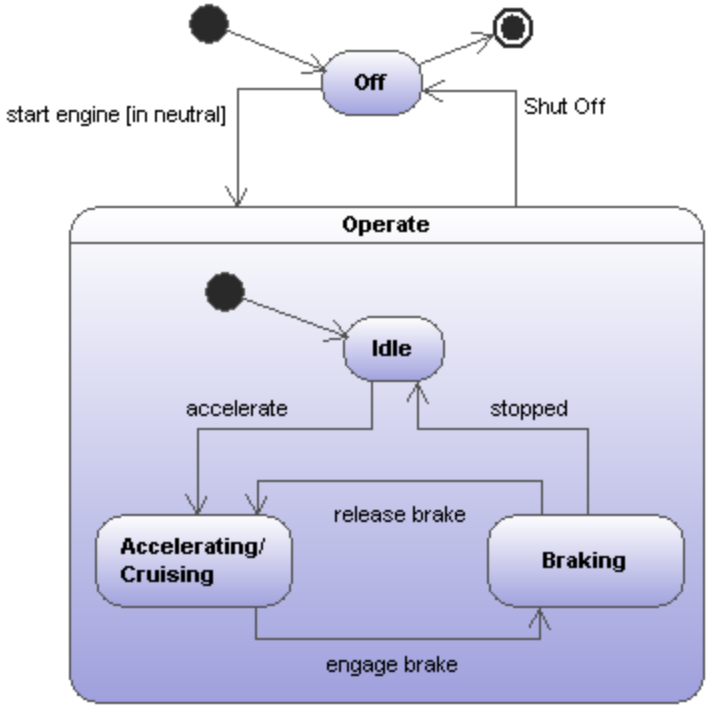
SysML-Sequenzdiagramm

Das SysML-Sequenzdiagramm entspricht der UML-Spezifikation. Um dieses Diagramm in UModel zu erstellen, benötigen Sie keine speziellen Kenntnisse im Vergleich zum Standard-UML-Sequenzdiagramm. Allgemeine Informationen zu UML-Sequenzdiagrammen finden Sie unter Sequenzdiagramm 411.

9.3.3.8 Zustandsdiagramm

Mit SysML-Zustandsdiagrammen werden Übergänge zwischen den Zuständen eines laufenden Systems beschrieben. Wie SysML- Sequenz- und Aktivitätsdiagramme dienen SysML-Zustandsdiagramme zur Darstellung von Systemverhalten.

© 2016-2022 Altova GmbH



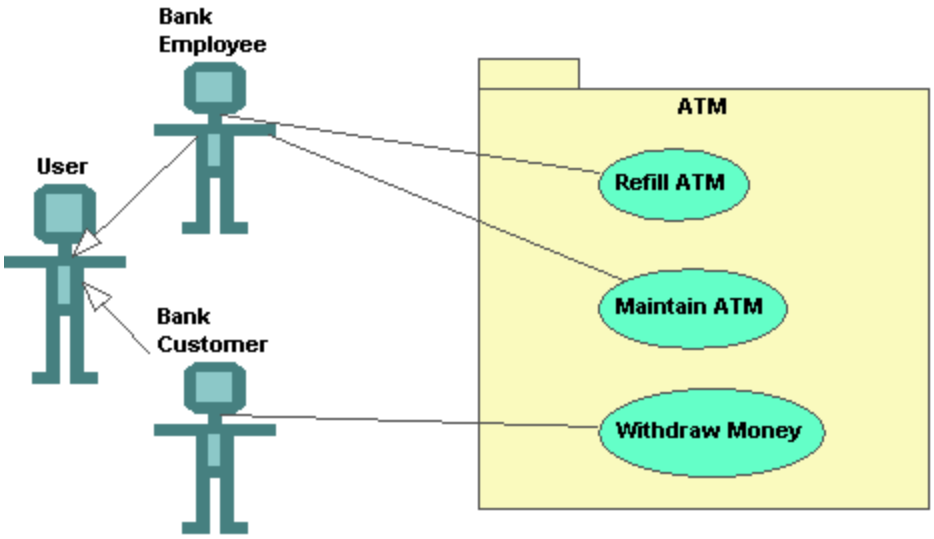
SysML-Zustandsdiagramm

Das SysML-Zustandsdiagramm entspricht der UML-Spezifikation. Um dieses Diagramm in UModel zu erstellen, benötigen Sie keine speziellen Kenntnisse im Vergleich zum Standard-UML-Zustandsdiagramm. Allgemeine Informationen zu UML-Zustandsdiagrammen finden Sie unter Zustandsdiagramm 374.

9.3.3.9 Use Case-Diagramm

Im SysML Use Case-Diagramm werden Elemente und Beziehungen, die von einem System bereitgestellte Dienste beschreiben, angezeigt. Außerdem werden darin verschiedene Beteiligte, die diese Dienste in Anspruch nehmen (wie z.B. Benutzer oder Systemoperatoren), dargestellt. Im Beispielprojekt **Bank_SysML.ump** aus dem Ordner **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples** finden Sie folgende Beispiele für Dienste:

- Ein Bankkunde, der von einem Bankomaten Geld abhebt
- Ein Bankangestellter, der den Bankomaten wartet
- Ein Bankangestellter, der den Bankomaten mit Geld befüllt



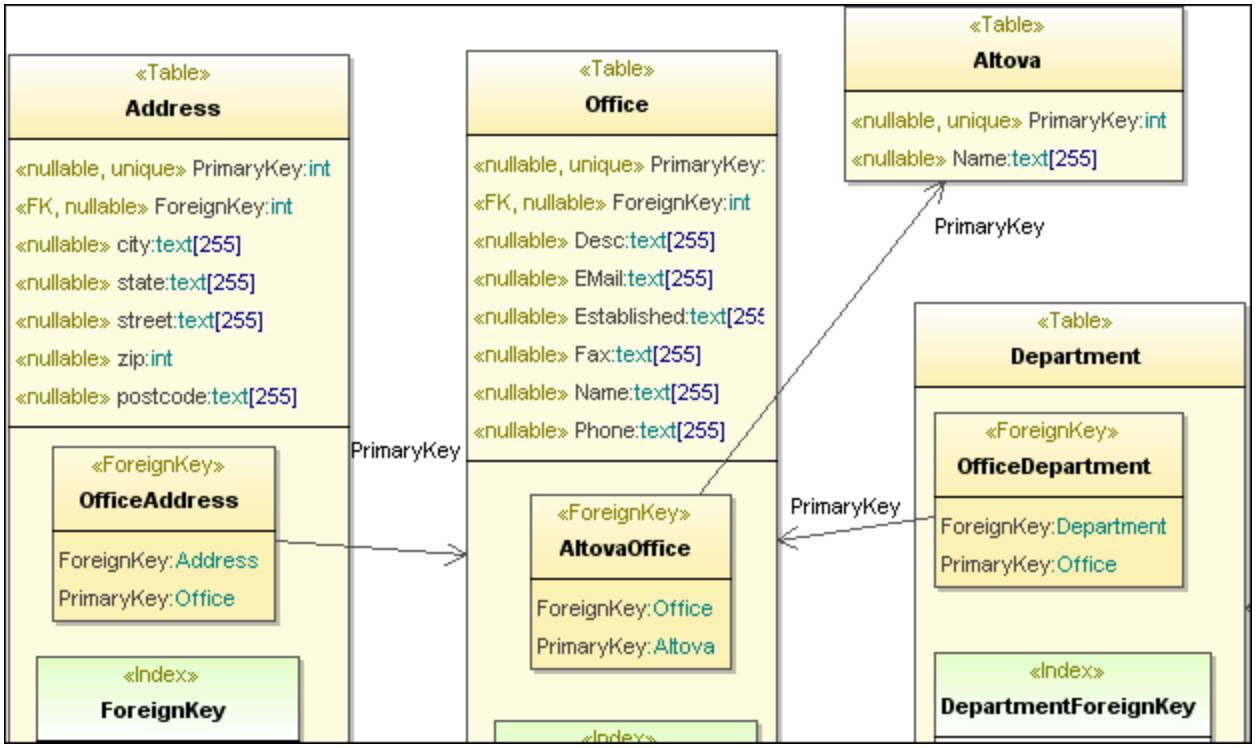
SysML Use Case-Diagramm

Das SysML Use Case-Diagramm entspricht der UML-Spezifikation und lässt sich genauso leicht erstellen wie ein Standard-UML-Use Case-Diagramm. Nähere Informationen finden Sie im Abschnitt Use Cases[22] des Tutorials.

© 2016-2022 Altova GmbH

10 UModel und Datenbanken

Sie können in UModel SQL-Datenbanken importieren, um deren Struktur anzuzeigen oder mittels UML zu ändern (Eine Liste der unterstützten Datenbanken finden Sie unter Datenbankunterstützung[17]). Die Datenbankstruktur kann in übersichtlichen UML-Datenbankdiagrammen, ähnlich dem unten gezeigten, angezeigt werden.



Die folgenden Datenbankelemente können in das Datenbankmodell importiert werden:

- Tabellen
- Check Constraints
- Primärschlüssel / Sekundärschlüssel / Eindeutige Schlüssel
- Indizes
- Ansichten
- Trigger
- Stored Procedures
- Funktionen

Anmerkung: Ansichten, Trigger, Stored Procedures und Funktionen können in UModel nur importiert, nicht aber dort hinzugefügt werden.

Nach Import der Datenbankstruktur in UModel kann diese geändert werden und die Änderungen können mit Hilfe des Menübefehls "**Merge Programmcode aus UModel-Projekt**" auf die tatsächliche Datenbank angewendet werden. Dadurch wird eine Datenbank Change Script-Datei erstellt, die ausgeführt oder für die spätere Ausführung gespeichert werden kann. Wenn alternativ dazu seit der letzten Synchronisierung Änderungen an der Datenbank vorgenommen wurden, können Sie diese im Modell übernehmen (oder das Modell mit den Änderungen überschreiben).

Nähere Informationen zum Mappen von Datenbankelementen auf UModel-Elemente finden Sie unter Datenbank-Entsprechungen[304].

© 2016-2022 Altova GmbH

10.1 Modellieren von Datenbanken in UModel

Sie können Datenbanken in UModel auf eine der folgenden Arten modellieren:

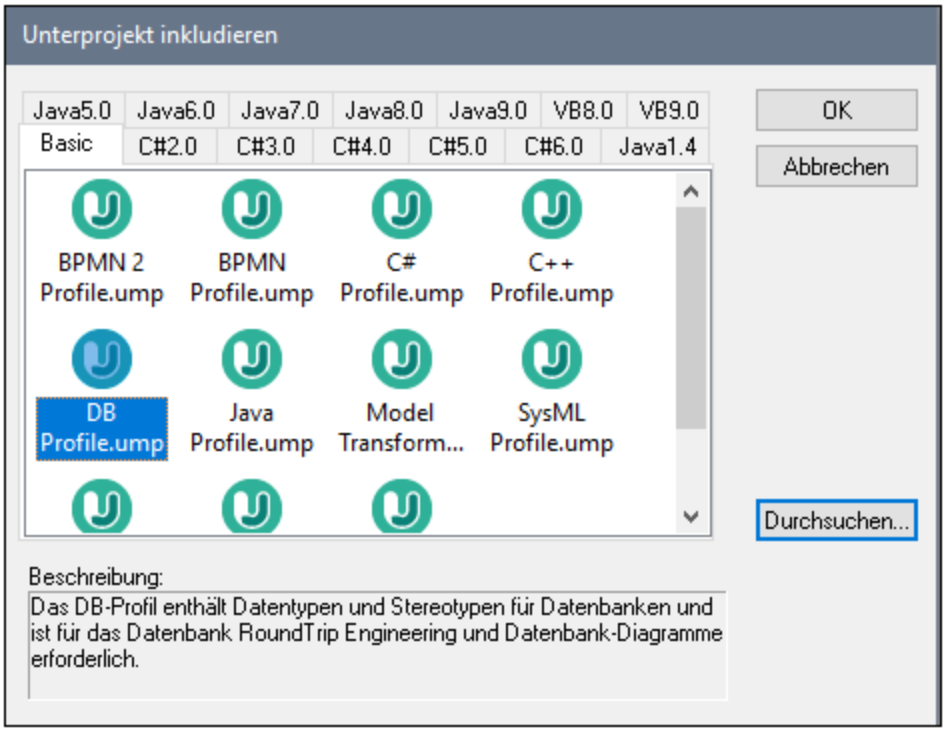
1. Ohne Code Engineering-Unterstützung modellieren Sie die Datenbankobjekte, ohne dass Sie eine Verbindung zu einer tatsächlichen Datenbank herstellen (z.B., wenn Sie nur ein Diagramm erstellen möchten, das die potenzielle Struktur einer Datenbank darstellt).
2. Mit Code Engineering-Unterstützung. In diesem Szenario stellen Sie eine Verbindung zu einer Datenbank her, importieren ihre Struktur in das Modell und zeigen die Datenbankobjektdefinitionen anschließend direkt in UModel an. UModel kann nach dem Auslesen der Datenbankstruktur automatisch Datenbankdiagramme generieren. Optional können Sie die Datenbankobjekte im Modell ändern (z.B. indem Sie eine neue Tabelle hinzufügen oder eine vorhandene löschen) und anschließend die tatsächliche Datenbank mit Hilfe von mit UModel generierten Skripts aktualisieren. Die Synchronisierung zwischen Ihrer Datenbank und dem UModel-Projekt funktioniert ähnlich wie bei Programmiersprachen in beide Richtungen. Sie haben auch die Option, nur die Änderungen (mittels Merge) zu synchronisieren oder alle vorhandenen Daten (entweder die Datenbank anhand des Modells oder das Modell anhand der Datenbank) zu überschreiben.

In jedem der oben beschriebenen Fälle muss Ihr Projekt das mit UModel verfügbare Datenbankprofil enthalten. Dieses Profil enthält alle erforderlichen Metadaten (wie z.B. UML-Stereotypen), mit Hilfe derer Sie Datenbankobjekte in UModel anzeigen oder erstellen können.

Wenn Sie die Code Engineering-Methode verwenden, werden das Datenbankprofil und alle erforderlichen Code Engineering-Konfigurationen automatisch beim ersten Import einer Datenbank in das Modell zum Projekt hinzugefügt. Andernfalls müssen Sie das Datenbankprofil manuell inkludieren.

So fügen Sie manuell ein Datenbankprofil zu einem UModel-Projekt hinzu:

1. Erstellen Sie ein neues UModel-Projekt oder öffnen Sie ein vorhandenes, siehe Erstellen, Öffnen und Speichern von Projekten 163.
2. Klicken Sie im Menü **Projekt** auf **Unterprojekt inkludieren**.



3. Wählen Sie auf dem Register **Basic DB Profile.ump** aus und klicken Sie zur Bestätigung auf **OK**.

Gehen Sie alternativ dazu folgendermaßen vor:

1. Klicken Sie im Fenster "Diagramm-Struktur" 90 mit der rechten Maustaste auf **Diagramme** und wählen Sie den Befehl **Neues Diagramm | Datenbankdiagramm**.
2. Wenn Sie dazu aufgefordert werden, wählen Sie ein Paket aus, zu dem das neue Diagramm gehören soll.
3. Wenn Sie UModel fragt, ob das Datenbankprofil zu Ihrem Projekt hinzugefügt werden soll, klicken Sie zur Bestätigung auf **OK**.

Nachdem das UModel DB-Profil nun hinzugefügt wurde, können Sie mit der Modellierung Ihrer Datenbankobjekte beginnen. Wenn Sie z.B. mit der rechten Maustaste in ein Datenbankdiagramm klicken, steht Ihnen über das Kontextmenü eine Option zur Erstellung einer neuen Tabelle zur Verfügung. Wenn Sie mit der rechten Maustaste auf eine Tabelle klicken, können Sie über das Kontextmenü Spalten, Schlüssel, Indizes usw. erstellen. Nähere Informationen dazu finden Sie unter Erstellen von Datenbankobjekten 565.

Um eine Verbindung zu einer Datenbank zu erstellen, verwenden Sie die Code Engineering-Methode, siehe Importieren von SQL-Datenbanken in UModel 558.

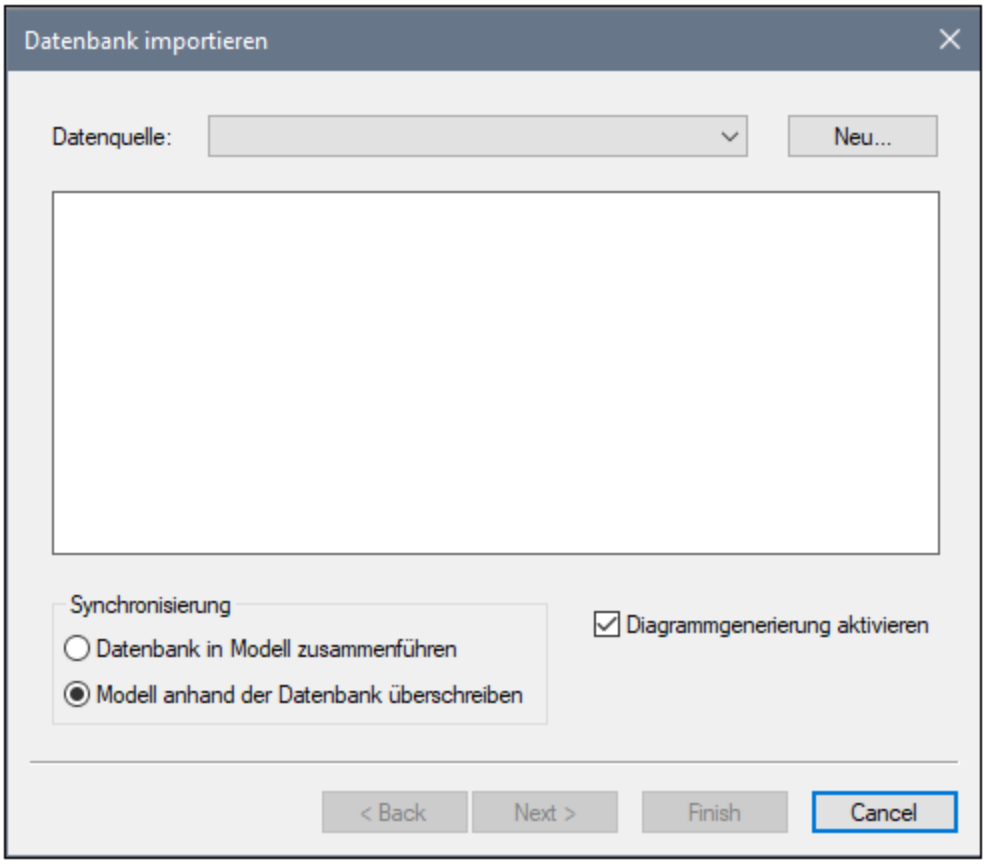
10.1.1 Importieren von SQL-Datenbanken in UModel

In der Anleitung unten wird beschrieben, wie Sie die Struktur einer Datenbank in UModel importieren. Außerdem erfahren Sie, wie Sie ein UML-Diagramm, das die Datenbankstruktur darstellt, generieren. Die in diesem Tutorial verwendete Datenbank ist eine Microsoft Access-Beispieldatenbank, doch sind die Schritte für andere von UModel unterstützte Datenbanken sehr ähnlich.

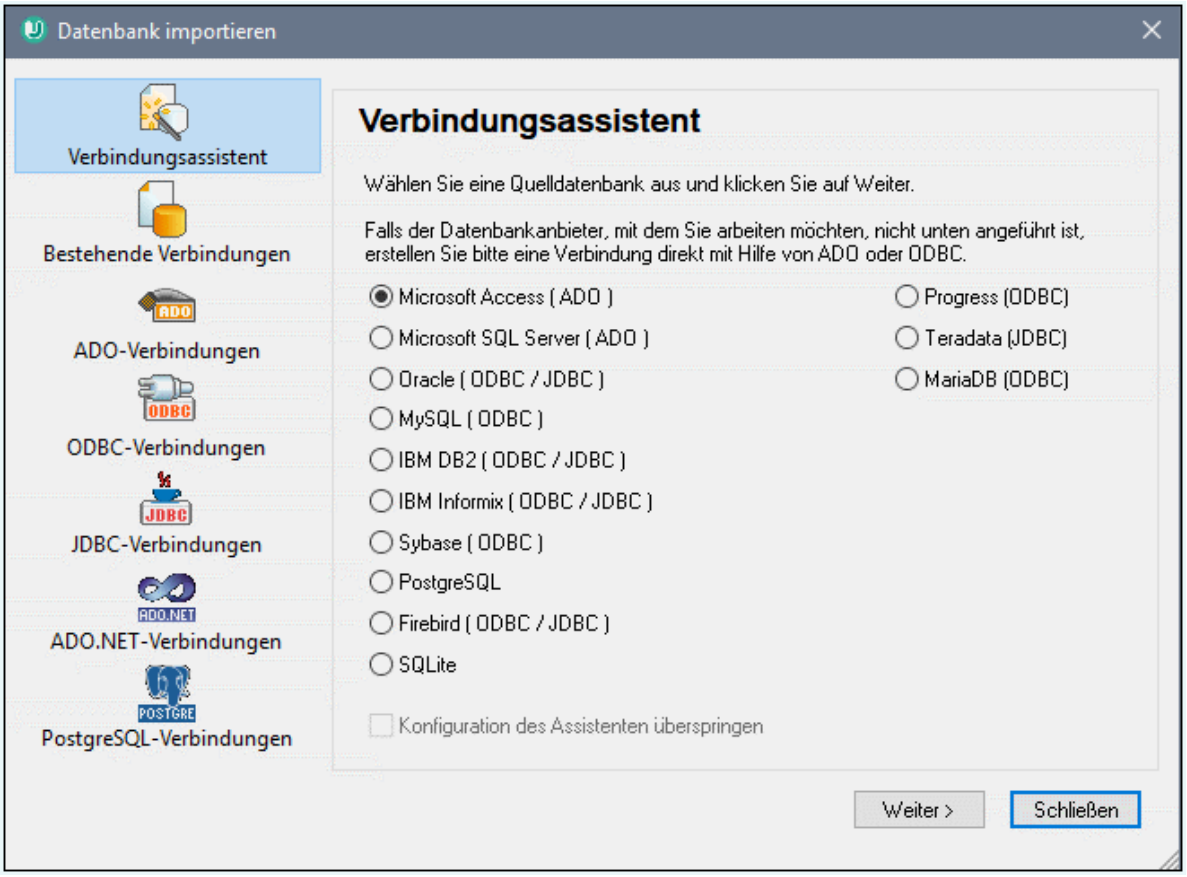
© 2016-2022 Altova GmbH

So importieren Sie eine Datenbank in UModel:

1. Klicken Sie im Menü **Projekt** auf **SQL-Datenbank importieren**.
2. Falls dies das erste Mal ist, dass Sie eine Datenbank in UModel importieren, klicken Sie auf **Neu**. Andernfalls können Sie eine vorhandene Datenbankverbindung aus der Liste **Datenquelle** auswählen.



3. Wir stellen in diesem Beispiel eine Verbindung zu einer lokalen Microsoft Access-Datenbank her. Wählen Sie daher als Datenbankart **Microsoft Access (ADO)** aus und klicken Sie auf **Weiter**. Befolgen Sie andernfalls die Anweisungen des Assistenten, um eine Verbindung zur Datenbank Ihrer Wahl herzustellen. Eventuell müssen Sie, abhängig von der Art der Datenbank, einen Datenbanktreiber installieren, bevor Sie die Verbindung herstellen können. Ein konkretes Beispiel dafür finden Sie unter Beispiele für Datenbankverbindungen 607.

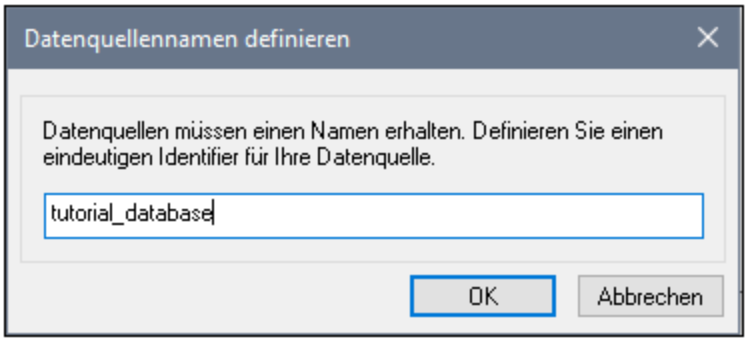


4. Navigieren Sie zur folgenden Datenbankdatei: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial\altova.mdb** und klicken Sie auf **Verbinden**.

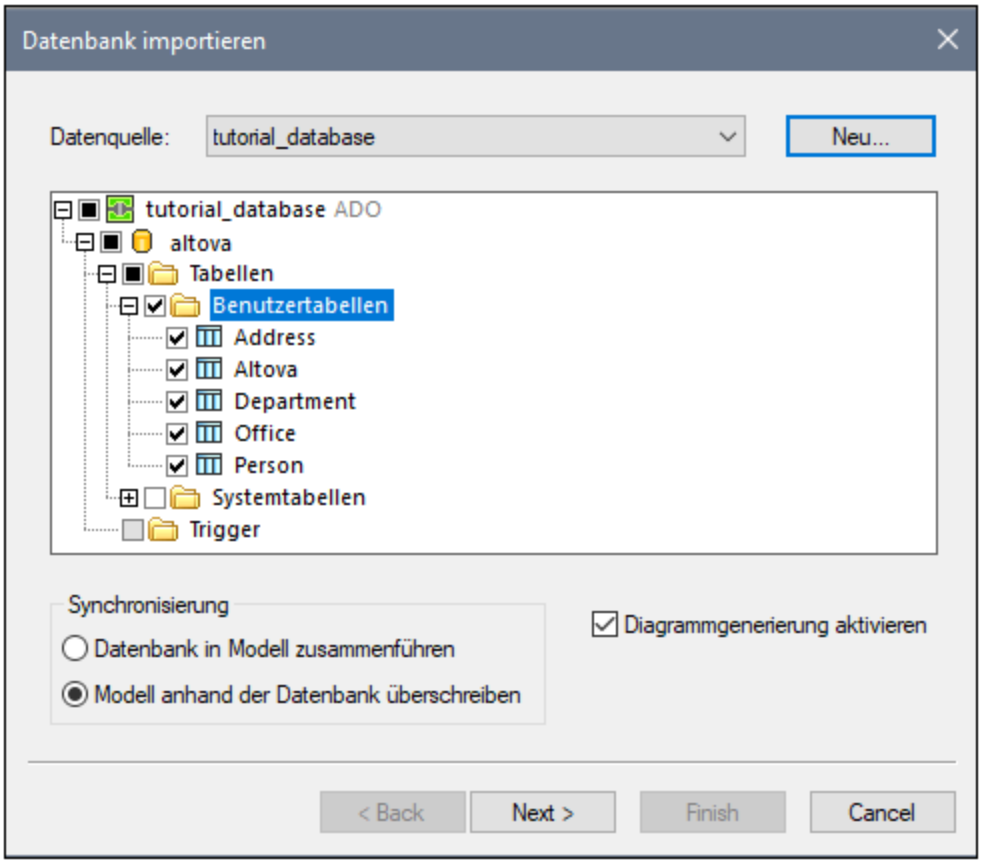
© 2016-2022 Altova GmbH



5. Geben Sie einen beschreibenden Namen für Ihre Datenquelle ein. Der hier definierte Datenquellenname steht später zur Auswahl, wenn Sie wieder eine Verbindung zur selben Datenbank herstellen möchten.

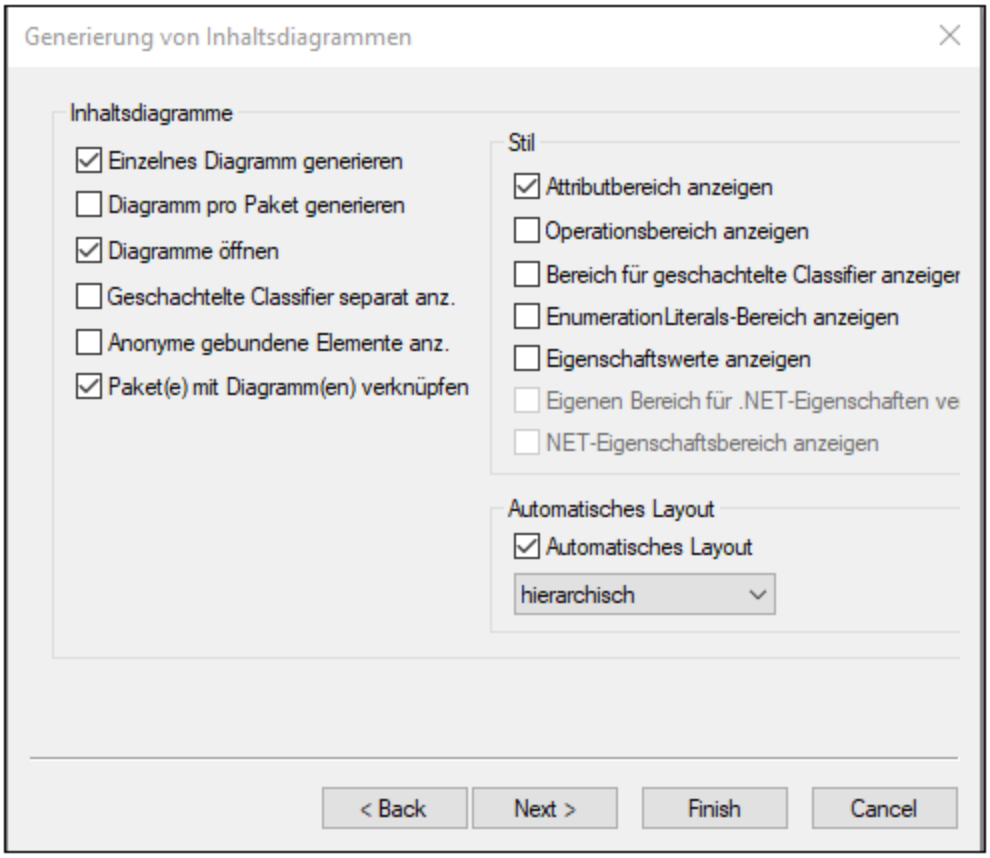


6. Wählen Sie die Datenbankobjekte, die Sie in das Modell importieren möchten, aus. In diesem Fall werden alle Benutzertabellen importiert. Beachten Sie außerdem, dass die Option **Modell anhand der Datenbank überschreiben** ausgewählt ist (d.h. alle Elemente im Projekt werden durch die aus der Datenbank importierten ersetzt). Ändern Sie diese Option bei bestehenden Projekten in **Datenbank in Modell zusammenführen**.



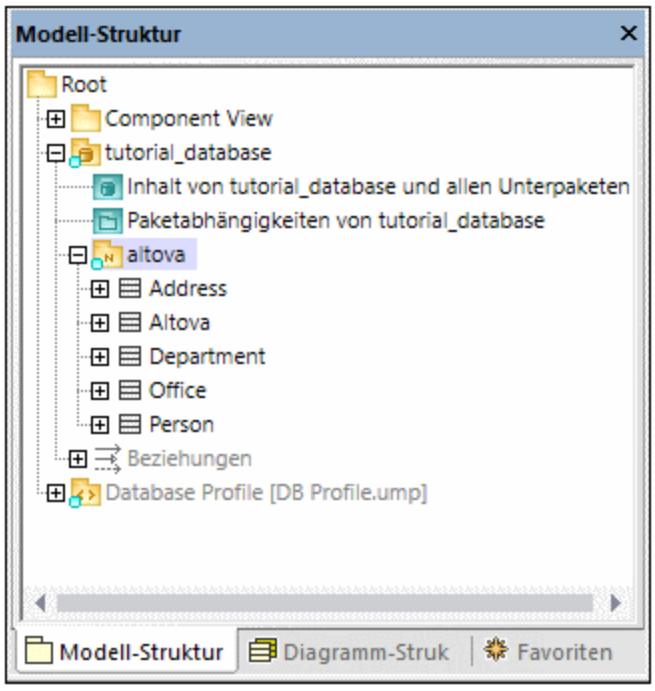
7. Klicken Sie auf **Weiter**. Wählen Sie die unten gezeigten Optionen aus:

© 2016-2022 Altova GmbH

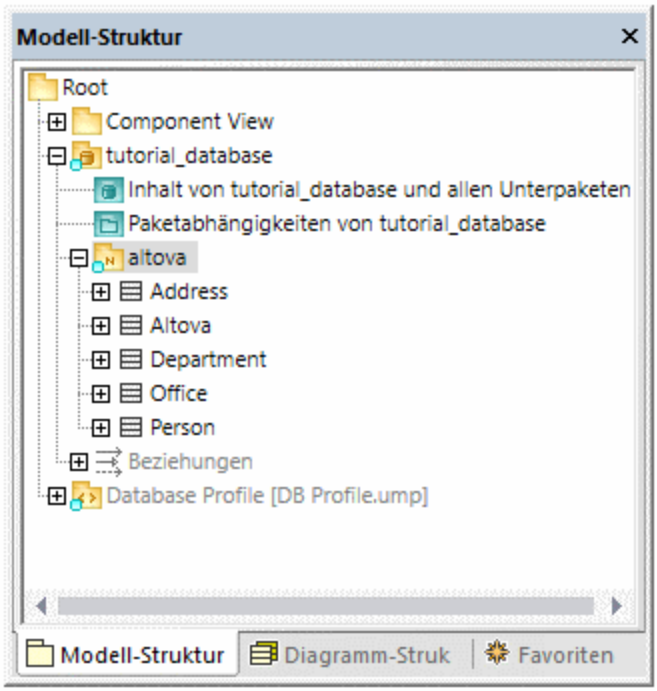


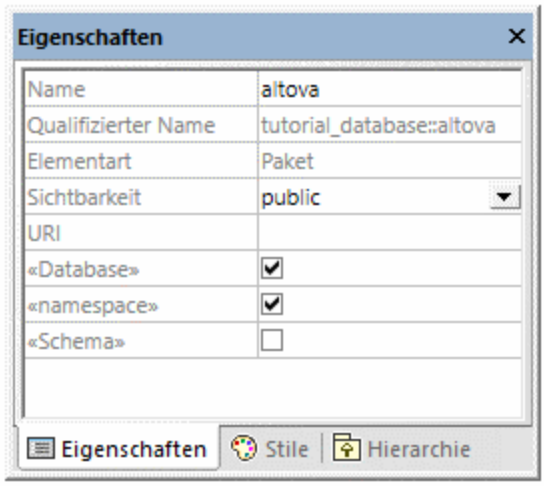
8. Klicken Sie auf **Fertig stellen**.

Nach dem Import enthält das Projekt alle aus der Datenbank importierten Objekte (Tabellen und deren Struktur). Außerdem werden zwei Diagramme erstellt, ein Datenbankdiagramm, das die Datenbankobjekte darstellt, und ein Paketabhängigkeitsdiagramm.



Wie Sie oben sehen, wurde die Datenquelle (in diesem Beispiel "tutorial_database") zu einem Paket im Modell. Die Datenbank selbst ("altova") wurde ebenfalls zu einem Paket, das sowohl das «Database» Stereotyp aus auch das «namespace» Stereotyp hat. Um die Eigenschaften eines Pakets anzuzeigen, klicken Sie auf das Paket und sehen Sie sich das Fenster "Eigenschaften" an, z.B.:





© 2016-2022 Altova GmbH

Anmerkung: Nach Import einer Datenbank erstellt UModel Pakete und wendet je nach Datenbankart Stereotype an. Das oben gezeigte Modell stellt Access-Datenbanken dar.

Alle Datenbanktabellen werden im Modell zu Klassen und erhalten das Stereotyp «Table». Beachten Sie außerdem, dass das Datenbankprofil (**DB Profile.ump**) nach Import der Datenbank automatisch zum Projekt hinzugefügt wurde.

Das Projekt ist zu diesem Zeitpunkt für das Code Engineering anhand der Datenbank zu einem Modell konfiguriert, d.h. immer, wenn Sie das UModel-Projekt mit den neuesten Datenbankänderungen aktualisieren möchten, starten Sie den folgenden Befehl:

- Klicken Sie im Menü **Projekt** entweder auf **Merge UModel-Projekt aus Programmcode** oder **Überschreibe UModel-Projekt aus Programmcode**.

Wenn Sie vorhaben, die Datenbank anhand Ihres Modells zu synchronisieren, lesen Sie nach unter Konfigurieren des Round-Trip Engineering für Datenbanken 570.

10.1.2 Erstellen von Datenbanobjekten

Sie können in UModel Datenbankobjekte (wie Tabellen, Spalten, Sekundärschlüssel, usw.) entweder über ein Datenbankdiagramm oder das Fenster "Modell-Struktur" erstellen, bearbeiten oder löschen.

Beachten Sie beim Anzeigen oder Erstellen von Datenbankobjekten in UModel die folgenden Grundregeln:

- Tabellen sind Klassen mit dem Stereotyp «Table».
- Spalten sind Klasseneigenschaften.
- Primär-, Sekundärschlüssel und eindeutige Schlüssel sind Klassen mit dem Stereotyp «PrimaryKey», «ForeignKey» bzw. «UniqueKey».
- Check Constraints sind Klassen mit dem Stereotyp «CheckConstraint».
- Indizes sind Klassen mit dem Stereotyp «Index».

Eine umfassende Tabelle, in der die Entsprechungen der einzelnen Datenbankobjekte mit UModel-Elementen dargestellt sind, finden Sie unter Datenbank-Entsprechungen 304 .

Hinzufügen von Tabellen

Um eine Tabelle zum Modell hinzuzufügen, wählen Sie eine der folgenden Methoden:

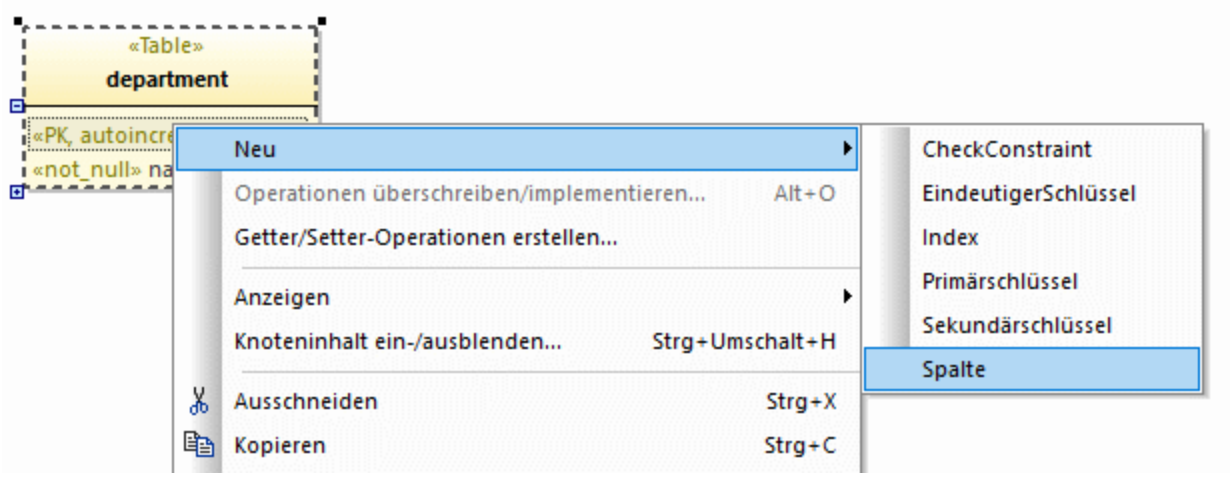
1. Erstellen Sie ein Datenbankdiagramm oder öffnen Sie ein vorhandenes. Um ein neues Datenbankdiagramm zu erstellen, klicken Sie mit der rechten Maustaste im Fenster Modell-Struktur 86 auf ein Paket und wählen Sie im Kontextmenü den Befehl **Neues Diagramm | Datenbankdiagramm**.
2. Wählen Sie eine der folgenden Methoden:
   a. Klicken Sie mit der rechten Maustaste in das Diagramm und wählen Sie im Kontextmenü den Befehl **Neu | Tabelle**.
   b. Klicken Sie auf die Symbolleisten-Schaltfläche **Neue Tabelle** und klicken Sie anschließend in das Diagramm, um die Tabelle hinzuzufügen.

Anmerkung: Sie können überall im Modell eine Tabellenklasse hinzufügen. Es hat sich jedoch bewährt, v.a., wenn Sie vorhaben, Code Engineering zu verwenden, alle Tabellenklassen unter einem Paket, das das Stereotyp «Database» hat, anzulegen. Ein solches Paket wird jedes Mal, wenn Sie eine

vorhandene Datenbank in das Modell importieren, automatisch erstellt, siehe Importieren von SQL-Datenbanken in UModel 558.

Hinzufügen von anderen Datenbankobjekten

Um eine Spalte, einen Index, Sekundärschlüssel, usw. zu einer Tabelle hinzuzufügen, klicken Sie mit der rechten Maustaste im Diagramm auf die Tabelle und wählen Sie den entsprechenden Befehl aus dem Kontextmenü aus, z.B.:

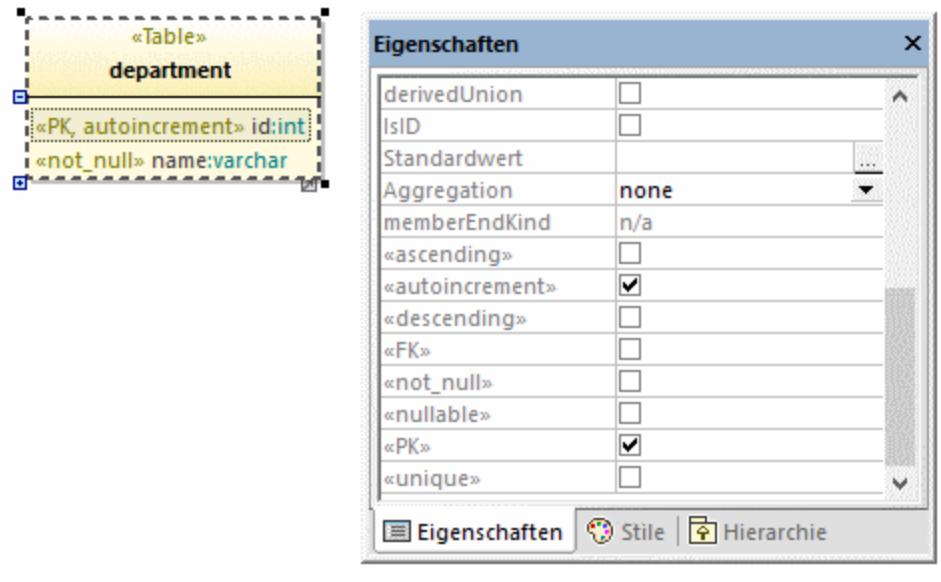


Klicken Sie alternativ dazu in der Symbolleiste des Diagramms auf eine Symbolleisten-Schaltfläche und anschließend in die Zieltabelle.



Um Spaltenattribute wie "autoincrement", "nullable", "primary key" zu definieren, klicken sie zuerst auf die Spalte und aktivieren Sie im Fenster "Eigenschaften" anschließend das gewünschte Kontrollkästchen (Stereotyp):

© 2016-2022 Altova GmbH



Sie können auch die Spalte erstellen und während der Eingabe alle erforderlichen Attribute direkt definieren. Um z.B. eine Primärschlüsselspalte mit dem Namen "id" und dem Typ "int", die automatisch inkrementiert wird, zu erstellen, gehen Sie folgendermaßen vor:

1. Wählen Sie im Diagramm eine Tabelle aus und drücken Sie F7.
2. Beginnen Sie mit der Eingabe von <<PK, autoincrement>> id:int. Während Sie den Text eingeben, zeigt UModel automatisch eine Liste an, aus der Sie die gewünschten Werte auswählen können.

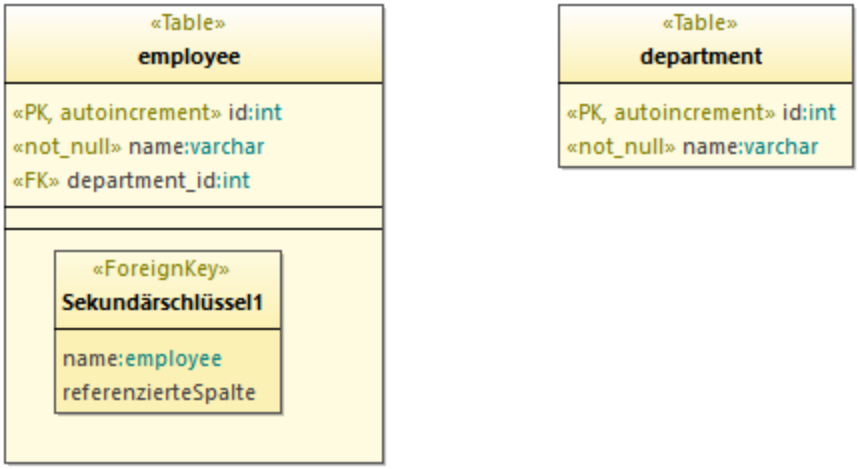
Hinzufügen von Datenbankbeziehungen

Normalerweise fügt man Beziehungen hinzu, um Sekundärschlüsselabhängigkeiten zwischen Spalten verschiedener Tabellen darzustellen. Angenommen, Sie haben die folgenden Klassen:

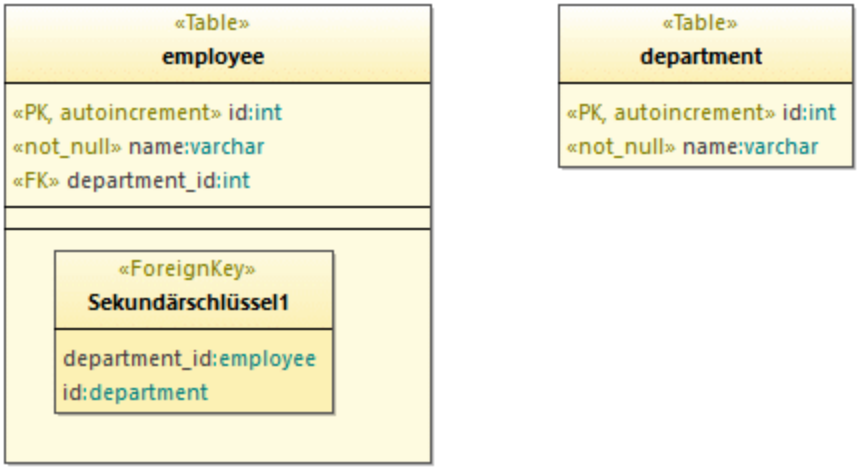


Um eine Sekundärschlüsselbeziehung zwischen der Spalte **department_id** in der Tabelle "employee" und der Spalte **id** in der Tabelle "department" zu erstellen, gehen Sie folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste auf die Tabelle "employee" und wählen Sie im Kontextmenü den Befehl **Neu | Sekundärschlüssel**. Daraufhin wird innerhalb der Klasse "employee" eine neue Klasse namens "Sekundärschlüssel1" hinzugefügt.

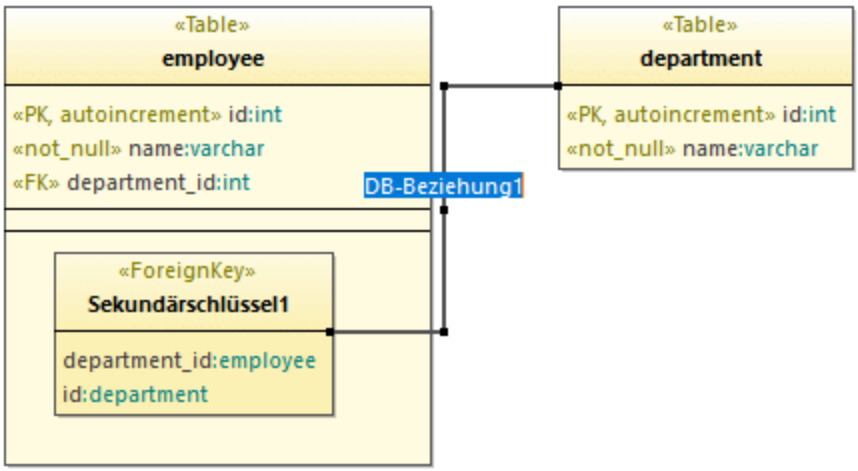


2. Ändern Sie den ersten Spalteneintrag in der Klasse "Sekundärschlüssel1" so, dass er der Besitzerspalte und -tabelle entspricht (in diesem Beispiel department_id:employee). Ändern Sie anschließend den zweiten Spalteneintrag so, dass er der referenzierten Spalte und Tabelle entspricht (in diesem Beispiel id:department).

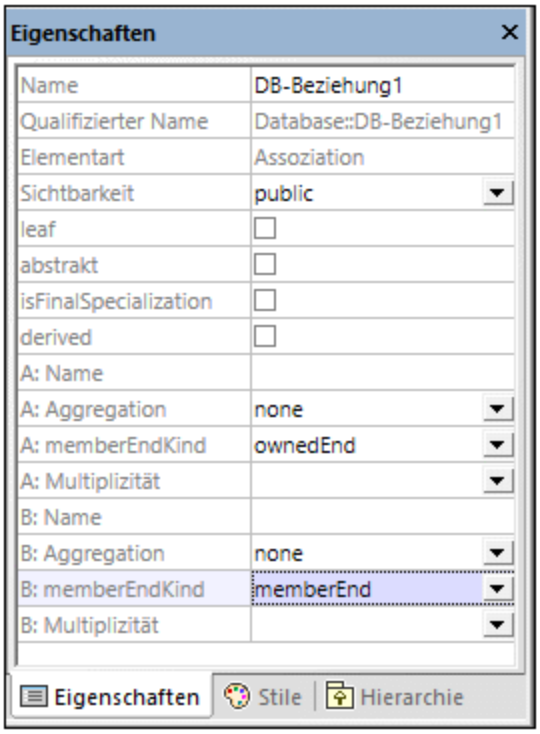


3. Klicken Sie auf die Symbolleisten-Schaltfläche **Datenbankbeziehungs-Assoziation** und ziehen Sie diese anschließend von der Klasse "Sekundärschlüssel1" auf die Klasse "department".

© 2016-2022 Altova GmbH



4. Wählen Sie die Beziehungslinie im Fenster "Eigenschaften" aus und ändern Sie die Eigenschaft **A :memberEndKind** in **memberEnd**.



5. Drücken Sie **F11,** um die Projektsyntax auf Fehler zu überprüfen (nähere Informationen siehe unten).

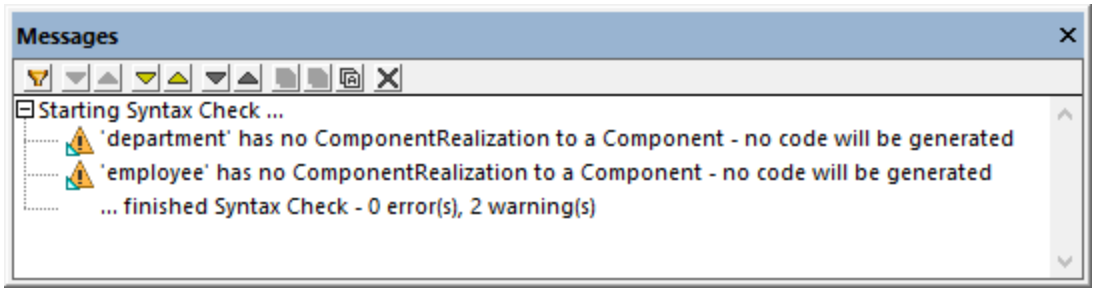
Anmerkung: Falls nötig, können Sie mehrere Spalteneinträge pro "Sekundärschlüssel"-Klasse hinzufügen. Sie können auch mehrere Indizes für dieselbe Tabelle hinzufügen.

Überprüfen der Projektsyntax

Es empfiehlt sich, während der Erstellung von oder Änderungen an Datenbankobjekten in UModel regelmäßig die Syntax ihres Projekts auf Designprobleme zu überprüfen (z.B. auf Tabellen, die nicht mindestens eine Spalte haben, fehlende Sekundärschlüsselreferenzen, usw.). So überprüfen Sie die Projektsyntax:

- Klicken Sie im Menü **Projekt** auf **Projektsyntax überprüfen**.
- Drücken Sie **F11**.

UModel validiert das Projekt und zeigen alle gefundenen Probleme im Fenster "Meldungen" an, z.B.:



Die beiden Warnungen in der Abbildung oben, bedeuten, dass für die Tabellen "department" und "employee" kein Code generiert wird. Wenn Sie in Ihrem UModel-Projekt keine Code Engineering-Unterstützung benötigen, können Sie solche Warnungen ignorieren. Lesen Sie andernfalls nach unter Konfigurieren des Round-Trip Engineering für Datenbanken 570.

10.1.3 Konfigurieren des Round-Trip Engineering für Datenbanken

Immer, wenn Sie eine Datenbank, wie in Importieren von SQL-Datenbanken in UModel 558 beschrieben, in UModel importieren, wird Ihr Projekt an die Datenbank gebunden und Sie können entweder Elemente anhand der Datenbank mit Ihrem Modell synchronisieren oder umgekehrt.

Wenn Sie nur eine Synchronisierung des Modells anhand der Datenbank vornehmen möchten, sind keine zusätzlichen Konfigurationsmaßnahmen erforderlich - UModel kümmert sich im Hintergrund um die erforderlichen Zuordnungen. So werden z.B. neue Datenbanktabellen nach jeder Synchronisierung im Modell zu neuen Klassen, geänderte Datenbankspaltendefinitionen werden im Modell aktualisiert, usw. Auch alle Ihre Datenbankdiagramme werden automatisch entsprechend aktualisiert.

Wenn Sie jedoch am Modell Änderungen vornehmen und Sie diese mit der Datenbank synchronisieren und in dieser übernehmen möchten, müssen Sie in Ihrem UModel-Projekt eventuell einige zusätzliche Konfigurationsschritte vornehmen. Diese Konfigurationsmaßnahmen sind eventuell auch erforderlich, wenn Sie verhindern möchten, dass das Projekt (oder bestimmte Tabellen) mit der Datenbank synchronisiert wird.

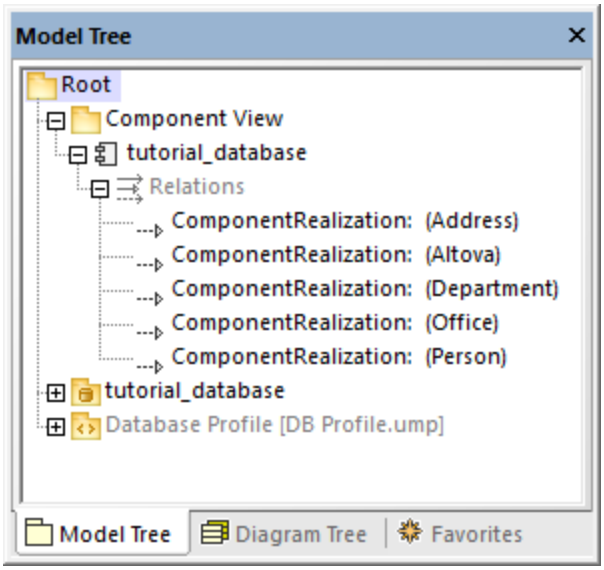
Bei einer Synchronisierung können Änderungen entweder zusammengeführt oder überschrieben werden. Sie können dies mit dem Menübefehl **Projekt | Synchronisierungseinstellungen** jederzeit konfigurieren.

Anmerkung: Bei einigen Datenbankarten sind Änderungen der Datenbankstruktur aufgrund ihres Designs nicht möglich. So wird etwa das Umbenennen von Tabellen und Spalten von Microsoft Access-

 © 2016-2022 Altova GmbH

Datenbanken nicht unterstützt. Ebenso wird das Umbenennen von Spalten in SQLite nicht unterstützt. Aus diesem Grund wird durch solche Änderungen im Modell keine Datenbankaktualisierung ausgelöst und unter Umständen wird eine entsprechende Warnmeldung im UModel-Fenster "Meldungen" angezeigt.

Das Round-Trip Engineering ist bei Datenbanken ähnlich wie das Round-Trip Engineering bei Programmcode - es erfolgt für eine Komponente im Paket "Component View", das Ihr Projekt mit der tatsächlichen Datenbank verknüpft. Immer, wenn Sie die Datenbank das erste Mal importieren, wird unter dem Paket "Component View" automatisch eine Code Engineering-Komponente generiert. Wenn Sie z.B. die gesamte Anleitung im Kapitel Importieren von SQL-Datenbanken in UModel 558 befolgt haben, wurde eine Komponente namens **tutorial_database** generiert:



Wie bereits erwähnt, entspricht jede Klasse im Modell einer Datenbanktabelle. Damit ein Code Engineering durchgeführt werden kann, muss die Code Engineering-Komponente alle Klassen (Tabellen) aus dem Modell realisieren. Beachten Sie alle **Komponentenrealisierungsbeziehung** in der Abbildung oben. Klassen, die von dieser Komponente nicht realisiert werden, bilden nicht Teil des Code Engineering. Wenn Sie nicht vorhaben, die Datenbank jemals anhand des Modells zu aktualisieren, müssen Sie nichts tun - UModel erstellt jedes Mal, wenn Sie das Modell anhand der Datenbank synchronisieren, automatisch alle Realisierungen.

Wenn Sie jedoch beabsichtigen, die Datenbank anhand des Modells zu synchronisieren, muss jede neue Klasse (Tabelle), die Sie hinzufügen, eine **Komponentenrealisierungsbeziehung** zur Code Engineering-Komponente haben. Andernfalls zeigt UModel beim Versuch die Datenbank anhand des Modells zu aktualisieren, eine Warnung ähnlich der folgenden an: `Tabelle1 hat keine Komponentenrealisierung zu einer Komponente - es wird kein Code generiert.`

Die einfachste Methode, um eine **Komponentenrealisierung** von einer Klasse zu einer Komponente zu erstellen, ist, die Klasse auf die Code Engineering-Komponente zu ziehen. Wenn Sie z.B. eine neue Klasse (Tabelle) erstellen, ziehen Sie die Klasse (im Fenster "Modell-Struktur") auf die Komponente **tutorial_database,** um die Beziehung zu erstellen. Sie können solchen Beziehungen auch über ein Komponentendiagramm hinzufügen oder entfernen (siehe Komponentendiagramme 54).

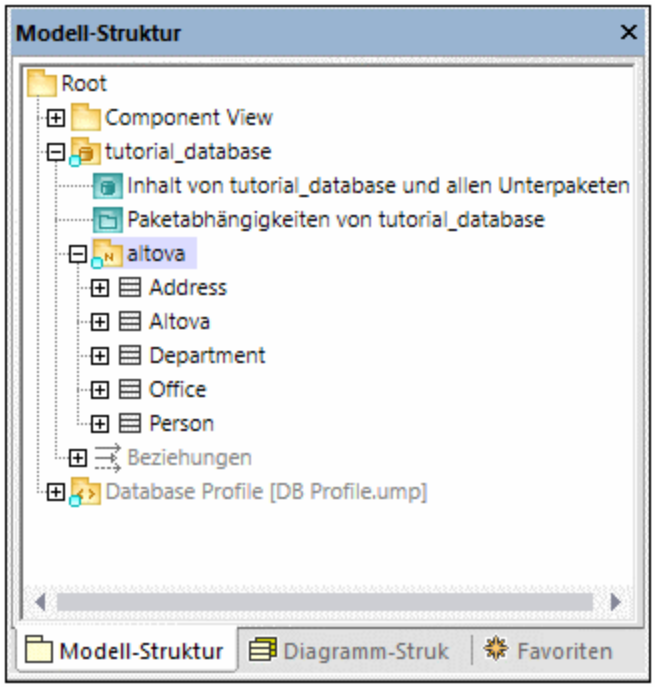
Ein Arbeitsbeispiel dazu finden Sie unter Beispiel: Aktualisieren einer Datenbank anhand des Modells 572.

10.1.4 Beispiel: Aktualisieren einer Datenbank anhand des Modells

In diesem Beispiel wird beschrieben, wie Sie die Struktur einer Datenbank mit Hilfe von mit UModel generierten Skripts aktualisieren. In diesem Beispiel wird eine lokale Access-Datenbank verwendet, die unter dem folgenden Pfad zur Verfügung steht: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\Tutorial\altova.mdb**. Wir fügen in diesem Beispiel eine neue Tabelle zur Datenbank in UModel hinzu, generieren dann ein SQL-Skript, das die Struktur der zugrunde liegenden Access-Datenbank aktualisiert.

Um mit diesem Beispiel fortzufahren, importieren Sie zuerst die Datenbank in das Modell, wie im Kapitel Importieren von SQL-Datenbanken in UModel 558 beschrieben. Wie unten aufgelistet, enthält Ihr Projekt nach dem Import die folgenden Komponenten:

- eine Code Engineering-Komponente für die Codegenerierung in beide Richtungen (anhand des Modells in die Datenbank und umgekehrt). Um die Code Engineering-Komponente zu sehen, erweitern Sie "Component View".
- ein Paket, das die Struktur der importierten Datenbank repräsentiert (so ist z.B. jede Datenbanktabelle eine Klasse).
- das Datenbankprofil für die Arbeit mit Datenbankmodellierungsprojekten.



Hinzufügen einer Tabelle

Fügen wir nun eine neue Tabelle zur Datenbank im Modell hinzu.

1. Doppelklicken Sie auf das Diagramm "Inhalt von tutorial_database...".

 © 2016-2022 Altova GmbH

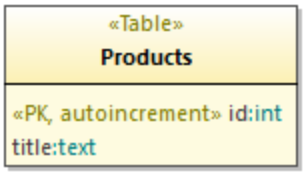
2. Klicken Sie mit der rechten Maustaste in das Diagramm und wählen Sie im Kontextmenü den Befehl **Neu | Tabelle**.
3. Geben Sie einen Tabellennamen ein, z.B., "Products".



4. Klicken Sie auf die Tabelle und drücken Sie **F7,** um eine neue Eigenschaft hinzuzufügen (diese wird in der Datenbank zu einer Tabellenspalte).
5. Geben Sie im Hauptteil der Eigenschaft `<<PK, autoincrement>> id:int` ein.

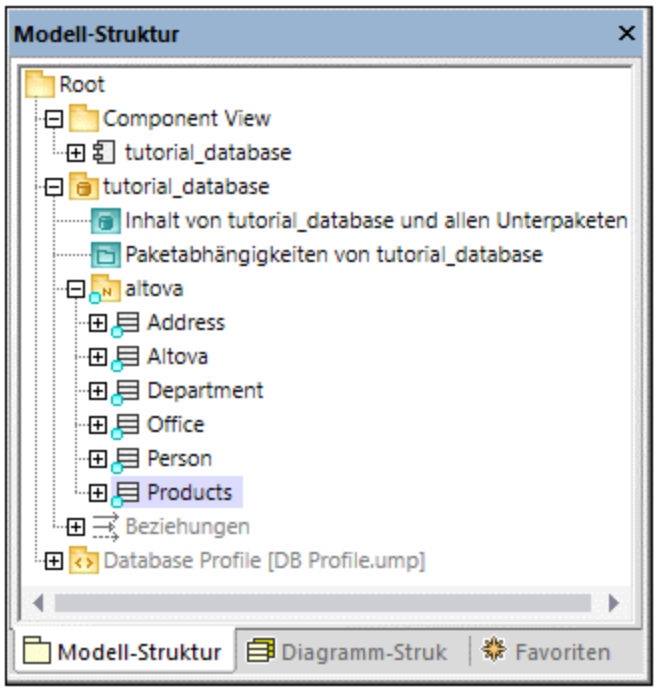


6. Fügen Sie auf dieselbe, oben beschriebene Art, eine neue Spalte "title" vom Typ "text" hinzu.



Vorbereiten des Modells für das Forward Engineering

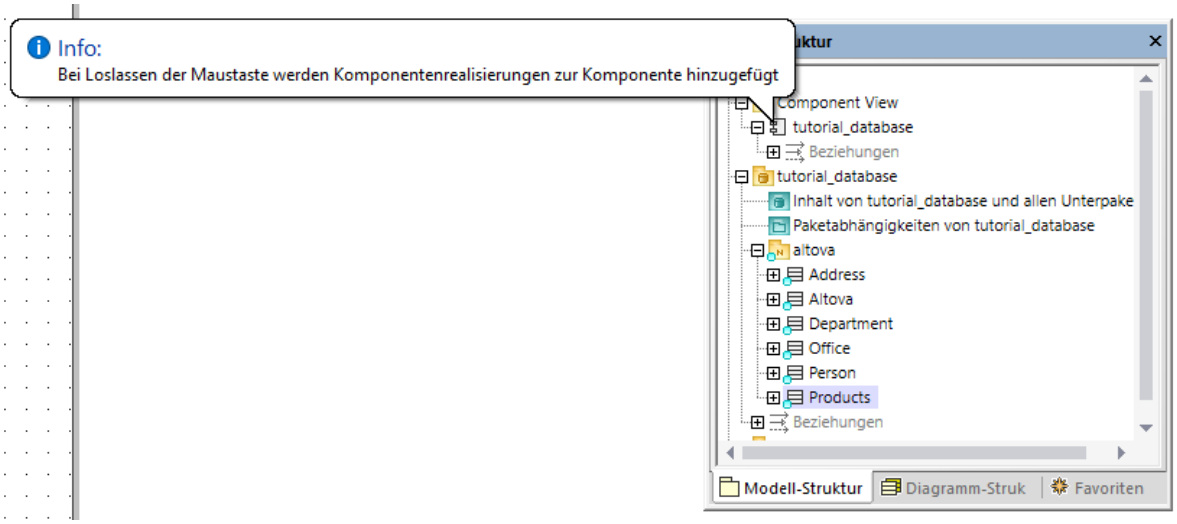
Bevor eine Tabelle mittels Forward Engineering vom Modell aus in der Datenbank übernommen werden kann, muss sie zum richtigen Namespace gehören. Stellen Sie dazu im Fenster "Modell-Struktur" sicher, dass sich die Klasse "Products" unter dem Namespace "tutorial_database" befindet. Falls dies nicht der Fall ist, ziehen Sie sie einfach mit der Maus auf den Namespace "tutorial_database". Ihr Modell sollte nun folgendermaßen aussehen:



Wie unter Konfigurieren des Round-Trip Engineering für Datenbanken 570 erläutert, empfiehlt es sich, die Projektsyntax zu validieren, bevor Sie versuchen, die Datenbank zu aktualisieren. Wenn Sie zu diesem Zeitpunkt **F11** drücken, um die Projektsyntax zu überprüfen, wird im Fenster "Meldungen" eine Warnung angezeigt, dass die Tabelle "Products" keine Realisierung zu einer Komponente aufweist.

Folgendermaßen können Sie schnell eine Realisierung zu einer Komponente erstellen:

- Ziehen Sie die Klasse "Products" im Fenster "Modell-Struktur" auf die Komponente "tutorial_database".

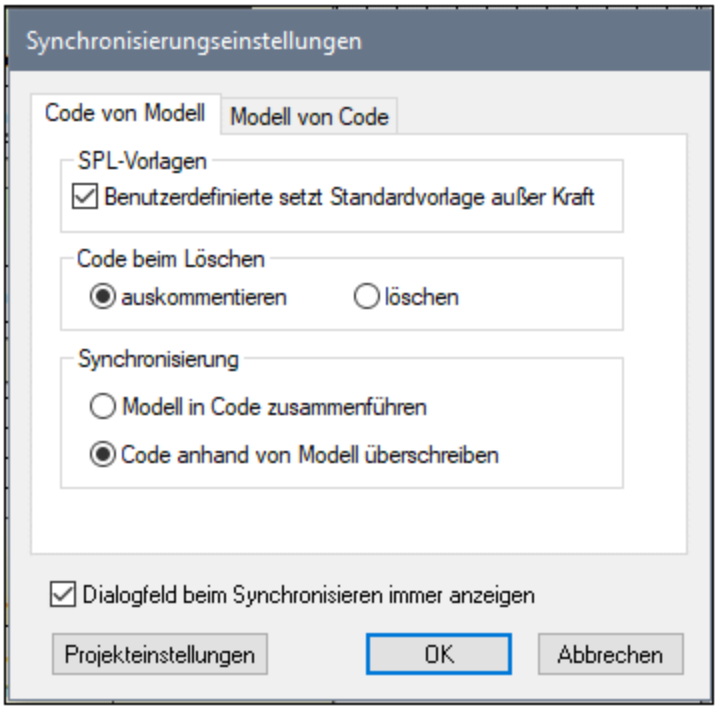


 © 2016-2022 Altova GmbH

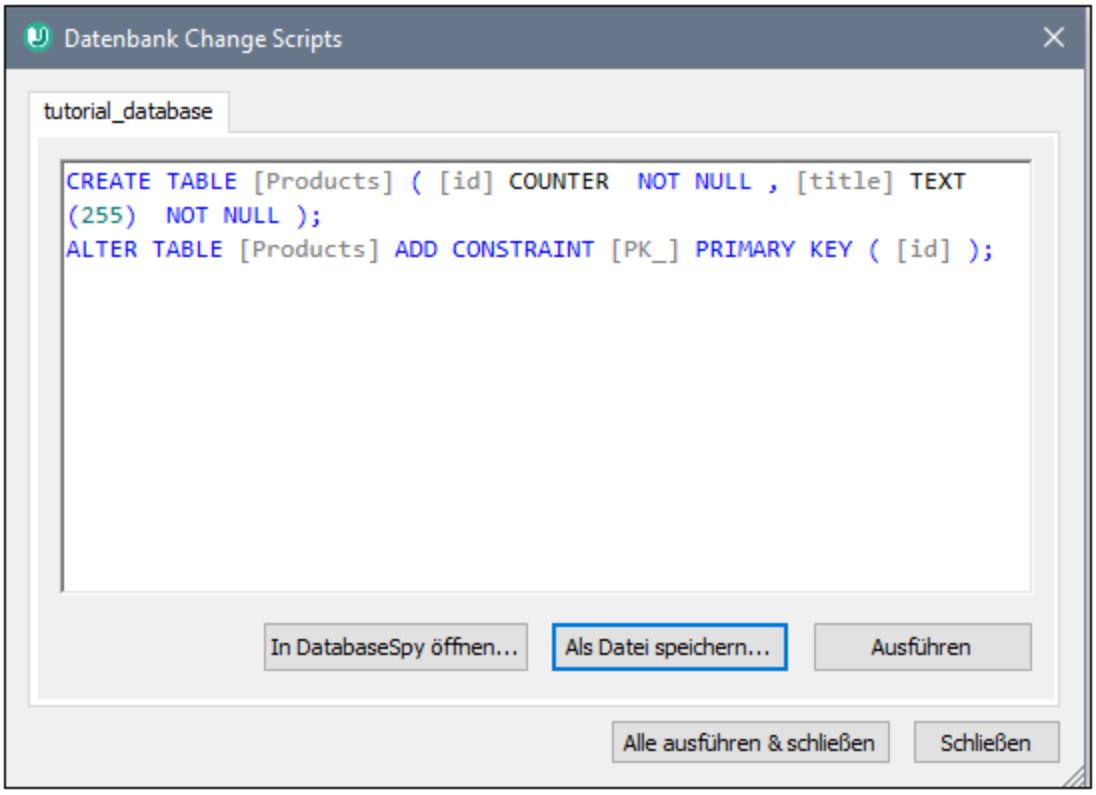
Generieren des SQL-Skripts

Wenn das Projekt bei Drücken von **F11** keine weiteren Fehler oder Warnungen aufweist, können Sie fortfahren, um das Datenbank-Skript zu generieren:

1. Klicken Sie im Menü **Projekt** auf **Überschreibe Programmcode aus UModel-Projekt**. (Mit "Programmcode" ist im Zusammenhang mit Datenbanken die Datenbank selbst gemeint)
2. Im Dialogfeld unten können Sie wählen, ob Sie die Änderungen in der Datenbank zusammenführen möchten oder die Datenbanken durch die Änderungen überschreiben möchten. In diesem Beispiel wählen wir die Option **Code anhand von Modell überschreiben**. Sie können aber je nach Fall auch die Option **Modell in Code zusammenführen** auswählen. Nähere Informationen dazu finden Sie unter Codesynchronisierungseinstellungen 244.



3. Klicken Sie auf **OK**. Daraufhin wird ein Datenbank-Skript mit den am Modell vorgenommenen Änderungen generiert.



Sie haben zu diesem Zeitpunkt die folgenden Möglichkeiten:

- Öffnen Sie das Skript in Altova DatabaseSpy, um es zu überprüfen oder auszuführen. Nähere Informationen zu DatabaseSpy finden Sie unter https://www.altova.com/de/databasespy.
- Speichern Sie das Skript in einer Datei, um es später zu verwenden.
- Klicken Sie auf **Ausführen** und führen Sie das Skript an der Datenbank aus. Führen Sie diesen Schritt nur dann durch, wenn Sie genau wissen, welche Folgen dies haben wird (die Datenbank wird dadurch sofort aktualisiert).

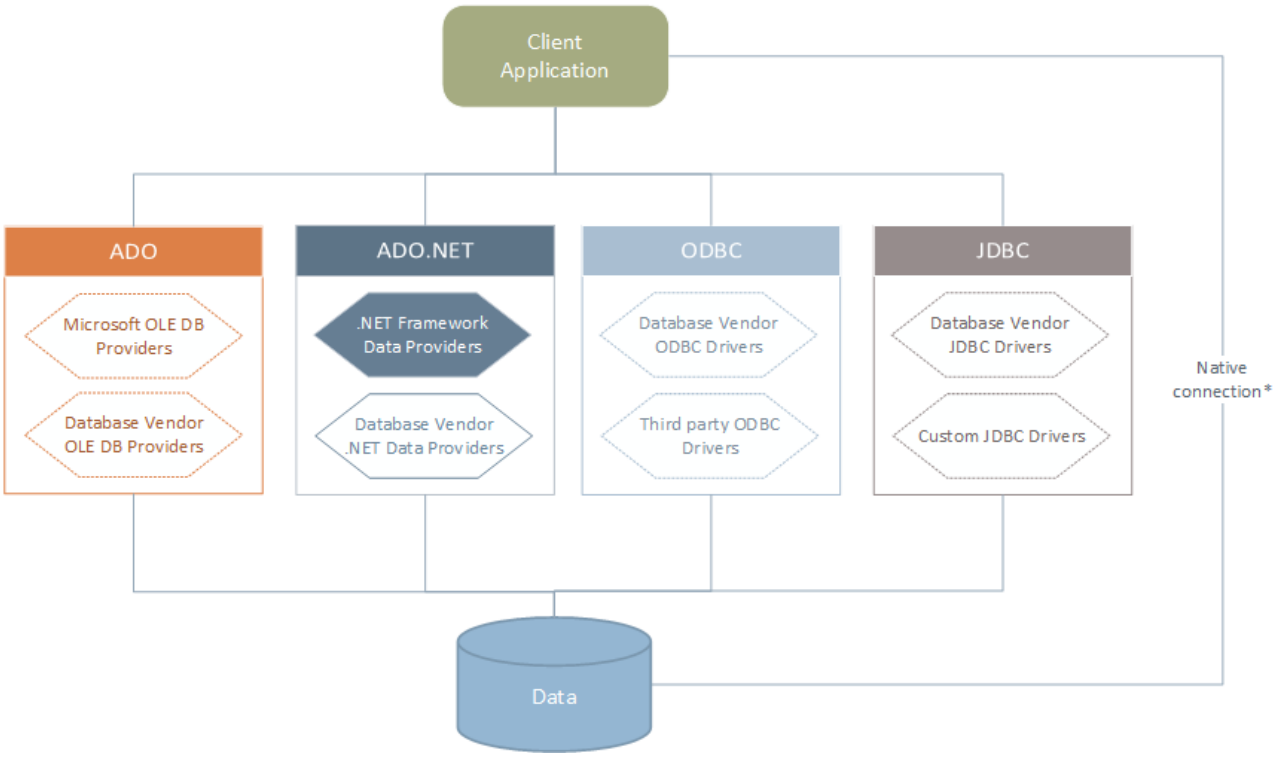
© 2016-2022 Altova GmbH

10.2 Herstellen einer Verbindung zu einer Datenquelle

Im einfachsten Fall kann es sich bei einer Datenbank um eine lokale Datei wie z.B. eine Microsoft Access- oder SQLite-Datenbankdatei handeln. In komplexeren Szenarien befindet sich die Datenbank manchmal auf einem entfernten Server oder eine Datenbank-Netzwerk-Server, auf dem nicht notwendigerweise dasselbe Betriebssystem wie das der damit verbundenen Applikation verwendet wird. Während z.B. UModel auf einem Windows-System läuft, könnte die Datenbank, über die Sie die Daten aufrufen möchten (z.B. MySQL), auf einem Linux-Rechner installiert sein.

Für die Verbindung mit verschiedenen Datenbanktypen - sowohl entfernten und lokalen, werden in UModel Datenverbindungsschnittstellen und Datenbanktreiber verwendet, die auf Ihrem Betriebssystem bereits vorhanden sind oder von denen regelmäßig aktualisierte Versionen von Anbietern gebräuchlicher Datenbanken bereitgestellt werden. Aufgrund sich ständig weiterentwickelnder Datenbanktechnologien bietet diese Methode bessere plattformübergreifende Flexibilität und Interoperabilität.

Im folgenden Diagramm werden Datenbankverbindungsoptionen zwischen UModel (als allgemeine Client-Applikation dargestellt) und einem Datenspeicher (einem Datenbank-Server oder einer Datenbankdatei) dargestellt.



** Für SQLite-, PostgreSQL-, CouchDB- und MongoDB-Datenbanken werden direkte native Verbindungen unterstützt. Für die Verbindung mit solchen Datenbanken müssen keine zusätzlichen Treiber auf Ihrem System installiert werden.*

Wie im Diagramm oben gezeigt, kann UModel zu jeder der gebräuchlichen Datenbankarten über die folgenden Technologien eine Verbindung herstellen:

- ADO (Microsoft® ActiveX® Data Objects), wofür wiederum ein zugrunde liegender OLE DB (Object Linking and Embedding, Database) Provider verwendet wird
- ADO.NET (eine Gruppe von im Microsoft .NET Framework verfügbaren Bibliotheken, die die Interaktion mit Daten ermöglichen)
- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Anmerkung: Einige ADO.NET-Anbieter werden nicht oder nur eingeschränkt unterstützt. Siehe Anmerkungen zur Unterstützung von ADO.NET 595.

Datenzugriffstechnologien

Welche Datenverbindungsschnittstelle Sie verwenden sollten, hängt größtenteils von der vorhandenen Software-Infrastruktur ab. Normalerweise werden Sie die Datenzugriffstechnolgie und den Datenbanktreiber verwenden, die enger mit dem gewünschten Datenbanksystem integriert sind. Um z.B. eine Verbindung zu einer Microsoft Access 2013-Datenbank herzustellen, würden Sie einen ADO Connection String erstellen, der einen nativen Provider wie z.B. den **Microsoft Office Access Database Engine OLE DB Provider** verwendet. Um eine Verbindung zu Oracle herzustellen, sollten Sie eventuell die neueste JDBC-, ODBC- oder ADO.NET-Schnittstelle von der Oracle Webseite herunterladen.

Während die Treiber für Windows-Produkte (wie z.B. Microsoft Access oder SQL Server) wahrscheinlich bereits auf Ihrem Windows Betriebssystem vorhanden sind, ist dies bei anderen Datenbanktypen möglicherweise nicht der Fall. Die wichtigsten Datenbankanbieter bringen regelmäßig öffentlich verfügbare Datenbank Client-Software und Treiber heraus, die durch beliebige Kombinationen von OLE DB, ODBC oder JDBC plattformübergreifenden Zugriff auf die jeweilige Datenbank ermöglichen. Zusätzlich dazu steht für jede der oben angeführten Technologien eine Reihe von Treibern von Drittanbietern zur Verfügung. In den meisten Fällen gibt es mehrere Möglichkeiten, um von Ihrem Betriebssystem und somit von UModel aus, eine Verbindung zur gewünschten Datenbank herzustellen. Welche Funktionalitäten und Performance-Parameter zur Verfügung stehen und welche bekannten Einschränkungen es gibt, hängt normalerweise von der Datenzugriffstechnologie oder den Treibern, die Sie verwenden, ab.

10.2.1 Starten des Verbindungsassistenten

UModel bietet einen Datenbankverbindungsassistenten, der Sie Schritt für Schritt durch das Herstellen einer Verbindung zu einer Datenquelle führt. Bevor Sie den Assistenten aufrufen, denken Sie daran, dass bei einige Datenbanktypen vorher einige Dinge wie z.B. ein Datenbanktreiber oder Datenbank Client-Software separat installiert und konfiguriert werden müssen. Diese erhalten Sie normalerweise vom jeweiligen Datenbankanbieter. Darin enthalten ist die Dokumentation zu Ihrer jeweiligen Windows-Version. Unter Übersicht über Datenbanktreiber 580 finden Sie eine Liste von Datenbanktreibern, gruppiert nach Datenbanktyp.

Um den Datenbankverbindungsassistenten (*siehe Abbildung unten*) zu starten, gehen Sie folgendermaßen vor:

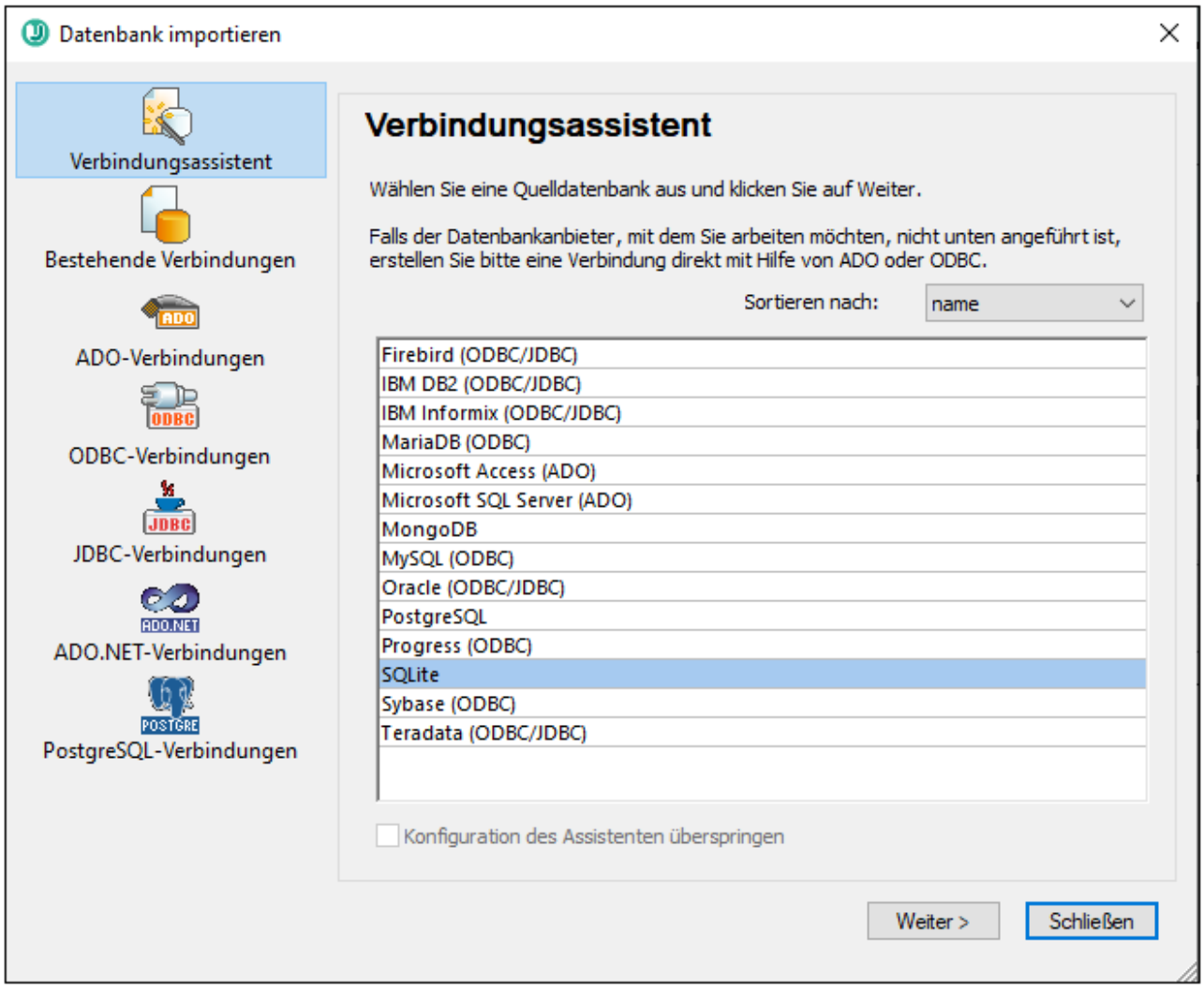
1. Klicken Sie im Menü **Projekt** auf **SQL-Datenbank importieren**.
2. Klicken Sie auf **Neu**.

Daraufhin wird der Datenbankverbindungsassistent gestartet (*Abbildung unten*). Auf der linken Seite des Fensters können Sie aus den folgenden Verbindungsarten die am besten geeignete auswählen:

 © 2016-2022 Altova GmbH

- Verbindungsassistent: Hier werden Sie aufgefordert, Ihren Datenbanktyp auszuwählen. Anschließend werden Sie Schritt für Schritt durch den Vorgang zur Herstellung einer Verbindung mit einer Datenbank dieses Typs geführt.
- Auswahl einer bestehenden Verbindung
- Auswahl einer Datenzugriffstechnologie: ADO, ADO.NET, ODBC oder JDBC
- Native PostgreSQL-Verbindung

Die Datenbanken können im Verbindungsassistent-Fenster (*siehe Abbildung unten*) alphabetisch nach dem Namen des Datenbanktyps oder dem zuletzt verwendeten Datenbanktyp sortiert werden. Wählen Sie die gewünschte Option in der Auswahlliste *Sortieren nach* aus. Klicken Sie nach Auswahl des gewünschten Datenbanktyps auf **Weiter**.



Je nach gewähltem Datenbanktyp, nach gewählter Verbindungstechnologie (ADO, ADO.NET, ODBC, JDBC) und Treiber werden Sie vom Assistenten durch den Verbindungsvorgang geführt. Beispiele zum jeweiligen Datenbanktyp finden Sie im Abschnitt Beispiele für Datenbankverbindungen 607.

Anstelle des Verbindungsassistenten können Sie eine der folgenden Datenbankzugriffstechnologien verwenden:

- Einrichten einer ADO-Verbindung 583
- Einrichten einer ADO.NET-Verbindung 589
- Einrichten einer ODBC-Verbindung 596
- Einrichten einer JDBC-Verbindung 599

10.2.2 Übersicht über Datenbanktreiber

Die folgende Tabelle enthält eine Liste gebräuchlicher Datenbanktreiber, über die Sie mit Hilfe einer bestimmten Datenzugriffstechnologie eine Verbindung zu einer bestimmten Datenbank herstellen können. Bitte beachten Sie, dass diese Liste keinen Anspruch auf Vollständigkeit erhebt und auch nicht zwingend befolgt werden muss; Sie können neben den unten angeführten Treibern auch andere native Treiber oder Produkte von Drittanbietern verwenden.

Standardmäßig stehen auf Windows Betriebssystemen zwar einige Datenbanktreiber bereits zur Verfügung, Sie müssen eventuell aber dennoch auch andere Treiber herunterladen und verwenden. Bei einigen Datenbanken empfiehlt es sich, anstelle des mit dem Betriebssystem mitgelieferten Treibers den neuesten Treiber des Datenbankanbieters zu verwenden.

Die meisten Datenbankanbieter bieten Treiber entweder als separat herunterladbare Pakete oder mit Datenbank-Client-Software gebündelt an. In letzterem Fall enthält die Datenbank Client-Software normalerweise alle erforderlichen Datenbanktreiber oder gibt Ihnen bei der Installation die Möglichkeit, die gewünschten Treiber und Komponenten auszuwählen. Datenbank Client-Software besteht normalerweise aus Verwaltungs- und Konfigurationstools zur einfacheren Verwaltung und Datenbankanbindung sowie der Dokumentation zum Installieren und Konfigurieren des Datenbank Client und seiner Komponenten.

Damit eine funktionierende Verbindung zur Datenbank hergestellt werden kann, muss der Datenbank Client unbedingt richtig konfiguriert werden. Bevor Sie die Datenbank Client-Software installieren, empfiehlt es sich, vorher die Installations- und Konfigurationsanleitungen zum Datenbank Client sorgfältig zu lesen, da diese normalerweise je nach Datenbank- und Windows-Version unterschiedlich ist.

Um die Möglichkeiten und Einschränkungen der einzelnen Datenzugriffstechnologien im Zusammenhang mit einzelnen Datenbanktypen zu verstehen, lesen Sie die Dokumentation zum jeweiligen Datenbankprodukt und testen Sie die Verbindung in Ihrer jeweiligen Rechnerumgebung. Beachten Sie die folgenden Hinweise und Empfehlungen, um häufige Verbindungsprobleme zu vermeiden:

- Einige ADO.NET-Anbieter werden nicht oder nur eingeschränkt unterstützt. Nähere Informationen dazu finden Sie unter Anmerkungen zur Unterstützung von ADO.NET 595.
- Bei Installation eines Datenbanktreibers wird empfohlen, dass dieser dieselbe Plattform wie die Altova-Applikation hat (32-Bit oder 64-Bit). Wenn Sie z.B. eine 32-Bit-Altova-Applikation auf einem 64-Bit-Betriebssystem verwenden, richten Sie ihre Datenbankverbindung mit der 32-Bit-Treiberversion ein, siehe auch Anzeigen der verfügbaren ODBC-Treiber 598.
- Es empfiehlt sich beim Einrichten einer ODBC-Datenquelle, im Allgemeinen den DSN (Data Source Name) als System-DSN und nicht als Benutzer-DSN zu erstellen. Nähere Informationen dazu finden Sie unter Einrichten einer ODBC-Verbindung 596.
- Stellen Sie beim Einrichten einer JDBC-Datenquelle sicher, dass JRE (Java Runtime Environment) installiert ist und dass die CLASSPATH-Umgebungsvariable des Betriebssystems konfiguriert ist. Nähere Informationen dazu finden Sie unter Einrichten einer JDBC-Verbindung 599.
- Nähere Informationen zur Installation und zu den Treibern oder der Datenbank Client-Software eines Datenbankanbieters finden Sie in der Dokumentation zum jeweiligen Installationspaket.

© 2016-2022 Altova GmbH

Datenbank	Schnittstelle	Treiber
Firebird	ADO.NET	Firebird ADO.NET-Datenanbieter (https://www.firebirdsql.org/en/additional-downloads/)
	JDBC	Firebird JDBC-Treiber (https://www.firebirdsql.org/en/jdbc-driver/)
	ODBC	Firebird ODBC-Treiber (https://www.firebirdsql.org/en/odbc-driver/)
IBM DB2	ADO	IBM OLE DB-Anbieter für DB2
	ADO.NET	IBM Datenserver-Anbieter für .NET
	JDBC	IBM Datenserver-Treiber für JDBC und SQLJ
	ODBC	IBM DB2 ODBC-Treiber
IBM DB2 for i	ADO	• IBM DB2 für i5/OS IBMDA400 OLE DB-Anbieter • IBM DB2 für i5/OS IBMDARLA OLE DB-Anbieter • IBM DB2 für i5/OS IBMDASQL OLE DB-Anbieter
	ADO.NET	.NET Framework Data Provider für IBM i
	JDBC	IBM Toolbox für Java JDBC-Treiber
	ODBC	iSeries Access ODBC-Treiber
IBM Informix	ADO	IBM Informix OLE DB-Treiber
	JDBC	IBM Informix JDBC-Treiber
	ODBC	IBM Informix ODBC-Treiber
Microsoft Access	ADO	• Microsoft Jet OLE DB-Anbieter • Microsoft Access Database Engine OLE DB-Anbieter
	ADO.NET	.NET Framework-Datenanbieter für OLE DB
	ODBC	• Microsoft Access-Treiber
MariaDB	ADO.NET	Falls kein eigener .NET-Konnektor für MariaDB vorhanden ist, verwenden Sie **Connector/NET** für MySQL (https://dev.mysql.com/downloads/connector/net/).
	JDBC	MariaDB Connector/J (https://downloads.mariadb.org/)
	ODBC	MariaDB Connector/ODBC (https://downloads.mariadb.org/)
Microsoft SQL Server	ADO	• Microsoft OLE DB-Treiber für SQL Server (MSOLEDBSQL) • Microsoft OLE DB-Anbieter für SQL Server (SQLOLEDB) • SQL Server Native Client (SQLNCLI)
	ADO.NET	• .NET Framework-Datenanbieter für SQL Server • .NET Framework-Datenanbieter für OLE DB

Datenbank	Schnittstelle	Treiber
	JDBC	• Microsoft JDBC-Treiber für SQL Server (https://docs.microsoft.com/en-us/sql/connect/jdbc/microsoft-jdbc-driver-for-sql-server)
	ODBC	• ODBC-Treiber für Microsoft SQL Server (https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server)
MySQL	ADO.NET	• Connector/NET (https://dev.mysql.com/downloads/connector/net/)
	JDBC	Connector/J (https://dev.mysql.com/downloads/connector/j/)
	ODBC	Connector/ODBC (https://dev.mysql.com/downloads/connector/odbc/)
Oracle	ADO	• Oracle-Anbieter für OLE DB • Microsoft OLE DB-Anbieter für Oracle
	ADO.NET	Oracle-Datenanbieter für .NET (http://www.oracle.com/technetwork/topics/dotnet/index-085163.html)
	JDBC	• JDBC Thin-Treiber • JDBC Oracle Call Interface (OCI)-Treiber Diese Treiber werden normalerweise während der Installation Ihres Oracle-Datenbank-Client installiert. Stellen Sie die Verbindung über den OCI-Treiber (und nicht den Thin-Treiber) her, wenn Sie die Oracle XML DB-Komponente verwenden.
	ODBC	• Microsoft ODBC für Oracle • Oracle ODBC-Teiber (wird normalerweise während der Installation Ihres Oracle-Datenbank-Client installiert)
PostgreSQL	JDBC	PostgreSQL JDBC-Treiber (https://jdbc.postgresql.org/download.html)
	ODBC	psqlODBC (https://odbc.postgresql.org/)
	Native Verbindung	Verfügbar. Bei Verwendung der nativen Verbindung müssen keine Treiber installiert werden.
Progress OpenEdge	JDBC	JDBC Connector (https://www.progress.com/jdbc/openedge)
	ODBC	ODBC Connector (https://www.progress.com/odbc/openedge)
SQLite	Native Verbindung	Verfügbar. Bei Verwendung der nativen Verbindung müssen keine Treiber installiert werden.
Sybase	ADO	Sybase ASE OLE DB-Anbieter
	JDBC	jConnect™ für JDBC
	ODBC	Sybase ASE ODBC-Treiber
Teradata	ADO.NET	.NET-Datenanbieter für Teradata (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata)

© 2016-2022 Altova GmbH

Datenbank	Schnittstelle	Treiber
	JDBC	Teradata JDBC-Treiber (https://downloads.teradata.com/download/connectivity/jdbc-driver)
	ODBC	Teradata ODBC-Treiber für Windows (https://downloads.teradata.com/download/connectivity/odbc-driver/windows)

10.2.3 ADO-Verbindung

Microsoft ActiveX Data Objects (ADO) ist eine Datenzugriffstechnologie, mit der Sie über OLE DB eine Verbindung zu einer ganzen Reihe von Datenquellen herstellen können. OLE DB ist eine Alternativschnittstelle zu ODBC oder JDBC und ermöglicht einen einheitlichen Zugriff auf Daten in einer COM (Component Object Model)-Umgebung. ADO ist ein Vorläufer des neueren ADO.NET[589] und ist weiterhin eine der möglichen Methoden, um eine Verbindung zu nativen Microsoft-Datenbanken wie Microsoft Access oder SQL Server herzustellen, kann aber auch für andere Datenquellen eingesetzt werden.

Beachten Sie, dass Sie zwischen mehreren ADO-Anbietern wählen können. Einige davon müssen zuerst heruntergeladen und auf Ihrem Rechner installiert werden, bevor Sie sie verwenden können. Für die Verbindung mit SQL Server stehen z.B. die folgenden ADO-Anbieter zur Verfügung:

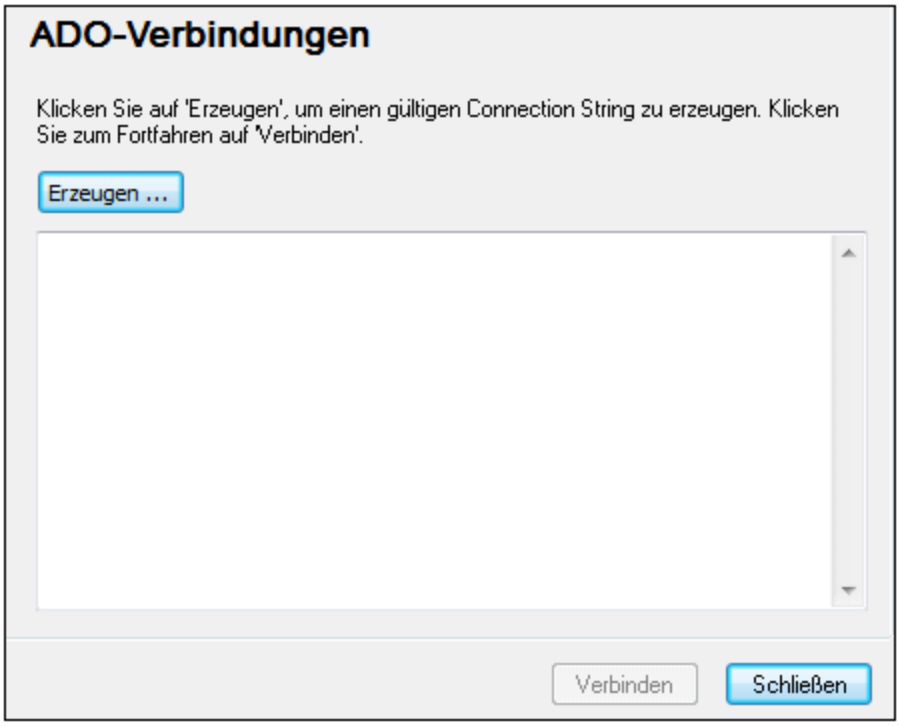
- Microsoft OLE DB-*Treiber* für SQL Server (MSOLEDBSQL)
- Microsoft OLE DB-*Anbieter* für SQL Server (SQLOLEDB)
- SQL Server Native Client (SQLNCLI)

Von den oben aufgelisteten Anbietern wird MSOLEDBSQL empfohlen; Sie können diesen von https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15 herunterladen. Beachten Sie, dass er mit der Plattform von UModel (32-Bit oder 64-Bit) übereinstimmen muss. Die Anbieter SQLOLEDB und SQLNCLI gelten als veraltet und werden daher nicht empfohlen.

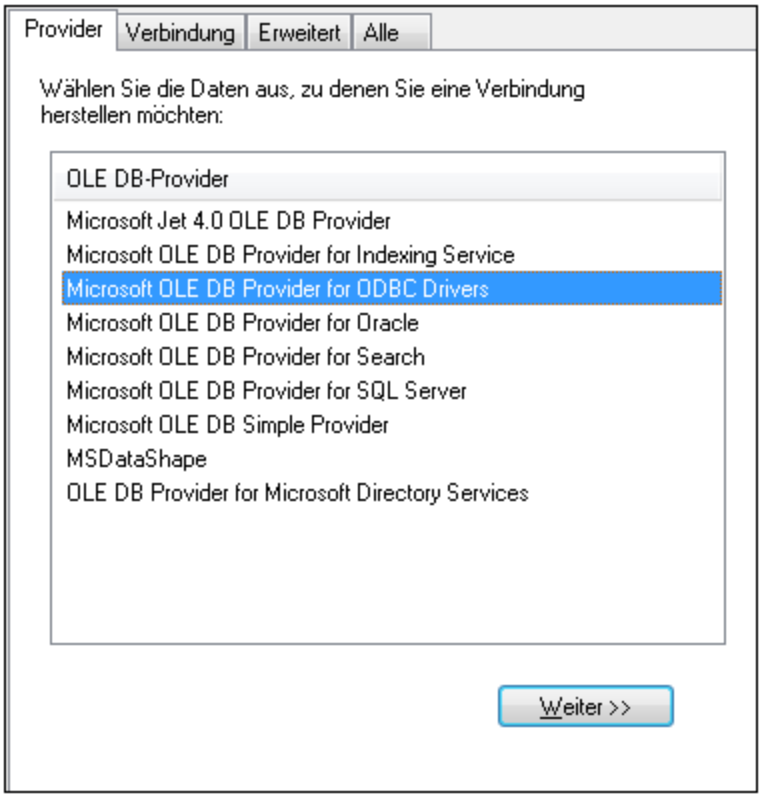
> Es ist bekannt, dass es beim **Microsoft OLE DB-Anbieter für SQL Server (SQLOLEDB)** zu Problemen mit der Parameterbindung komplexer Abfragen wie Common Table Expressions (CTE) und verschachtelten SELECT-Anweisungen kommt.

So richten Sie eine ADO-Verbindung ein:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **ADO-Verbindungen**.



3. Klicken Sie auf **Erzeugen**.



© 2016-2022 Altova GmbH

4. Wählen Sie den Daten-Provider, über den Sie die Verbindung erstellen möchten. In der unten stehenden Tabelle sind einige häufige Szenarien aufgelistet.

Zum Verbinden mit dieser Datenbank...	Verwenden Sie diesen Anbieter...
Microsoft Access	• **Microsoft Office Access Database Engine OLE DB-Anbieter** (empfohlen) • **Microsoft Jet OLE DB-Anbieter** Wenn der **Microsoft Office Access Database Engine OLE DB-Anbieter** in der Liste nicht vorhanden ist, überprüfen Sie, ob Sie entweder Microsoft Access oder die Microsoft Access Database Engine Redistributable (https://www.microsoft.com/en-us/download/details.aspx?id=54920) auf Ihrem Rechner installiert haben.
SQL Server	• **Microsoft OLE DB-Treiber für SQL Server (MSOLEDBSQL)** - dies ist der empfohlene OLE DB-Anbieter. Damit dieser Anbieter in der Liste angezeigt wird, muss er von https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15 heruntergeladen und installiert werden. • **Microsoft OLE DB-Anbieter für SQL Server (OLEDBSQL)** • **SQL Server Native Client (SQLNCLI)**
Andere Datenbank	Wählen Sie den für Ihre Datenbank benötigten Anbieter aus. Wenn für Ihre Datenbank kein OLE DB-Anbieter zur Verfügung steht, installieren Sie den erforderlichen Treiber des Datenbankanbieters (siehe Übersicht über Datenbanktreiber 580). Alternativ dazu können Sie eine ADO.NET, ODBC- oder JDBC-Verbindung einrichten. Wenn das Betriebssystem über einen ODBC-Treiber für die gewünschte Datenbank verfügt, können Sie auch den **Microsoft OLE DB-Anbieter für ODBC-Treiber** verwenden oder sich vorzugsweise für eine ODBC-Verbindung 596 entscheiden.

5. Klicken Sie nach Auswahl des gewünschten Anbieters auf **Weiter** und stellen Sie den Assistenten fertig.

Die nächsten Schritte im Assistenten hängen vom verwendeten Anbieter ab. Bei SQL Server müssen Sie den Host-Namen des Datenbankservers angeben oder auswählen sowie die Authentifizierungsmethode, den Datenbanknamen und den Datenbank-Benutzernamen und das Passwort dafür. Ein Beispiel dafür finden Sie unter Verbinden mit Microsoft SQL Server (ADO) 629. Bei Microsoft Access müssen Sie zur Datenbankdatei navigieren bzw. den Pfad dafür angeben. Ein Beispiel dafür finden Sie unter Verbinden mit Microsoft Access (ADO) 627.

Auf dem Register **Alle** des Verbindungsdialogfelds finden Sie die vollständige Liste der Initialisierungseigenschaften (Verbindungsparameter). Diese Eigenschaften sind je nach gewähltem Anbieter

unterschiedlich und müssen eventuell explizit konfiguriert werden, um die Verbindung herstellen zu können. In den folgenden Abschnitten finden Sie eine Anleitung, wie Sie die grundlegenden Initialisierungseigenschaften für Microsoft Access- und SQL Server-Datenbanken konfigurieren:

- Einrichten der SQL Server-Datenverknüpfungseigenschaften 586
- Einrichten der Microsoft Access-Datenverknüpfungseigenschaften 587

10.2.3.1 Herstellen einer Verbindung zu einer vorhandenen Microsoft Access-Datenbank

Gehen Sie auf diese Art und Weise vor, wenn Sie eine Verbindung zu einer nicht durch ein Passwort geschützten Microsoft Access-Datenbank herstellen möchten. Wenn die Datenbank passwortgeschützt ist, richten Sie das Datenbankpasswort, wie unter Verbinden mit Microsoft Access (ADO) 627 beschrieben, ein.

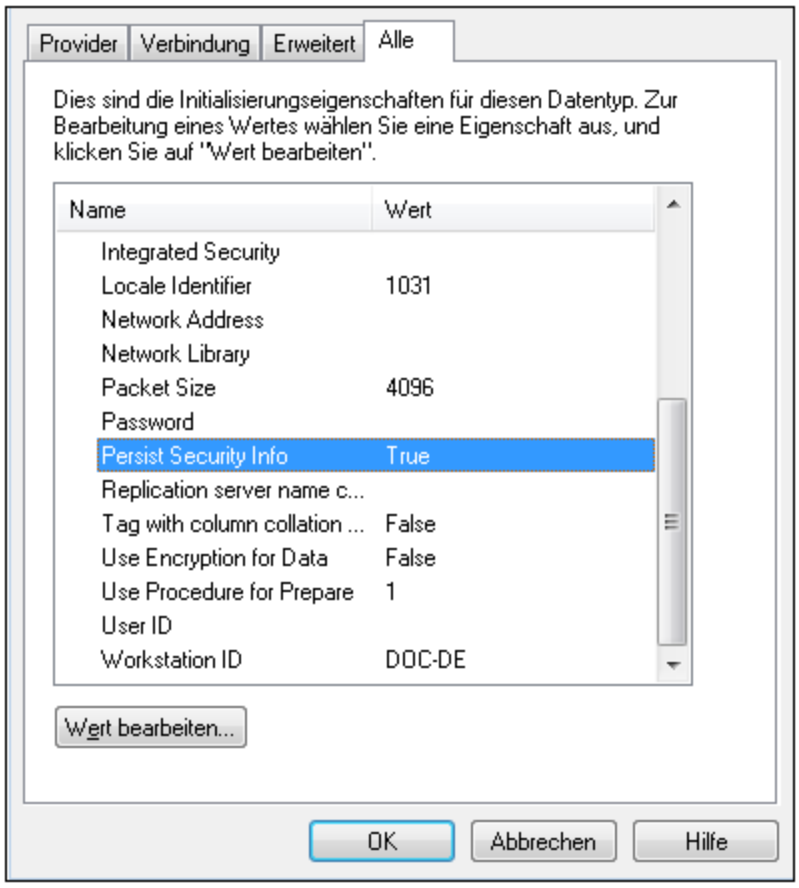
So stellen Sie eine Verbindung zu einer vorhandenen Microsoft Access-Datenbank her:

1. Starten Sie den Datenbankverbindungsassistenten (siehe Starten des Datenbankverbindungsassistenten 578).
2. Wählen Sie **Microsoft Access (ADO)** aus und klicken Sie auf **Weiter**.
3. Navigieren Sie zur Datenbankdatei oder geben Sie (entweder den relativen oder den absoluten) Pfad ein.
1. Klicken Sie auf **Verbinden**.

10.2.3.2 Einrichten der SQL Server-Datenverknüpfungseigenschaften

Wenn Sie über ADO 583 eine Verbindung zu einer Microsoft SQL Server-Datenbank herstellen, müssen Sie eventuell die folgenden Datenverknüpfungseigenschaften auf dem Register **Alle** des Dialogfelds "Datenverknüpfungseigenschaften" konfigurieren.

© 2016-2022 Altova GmbH

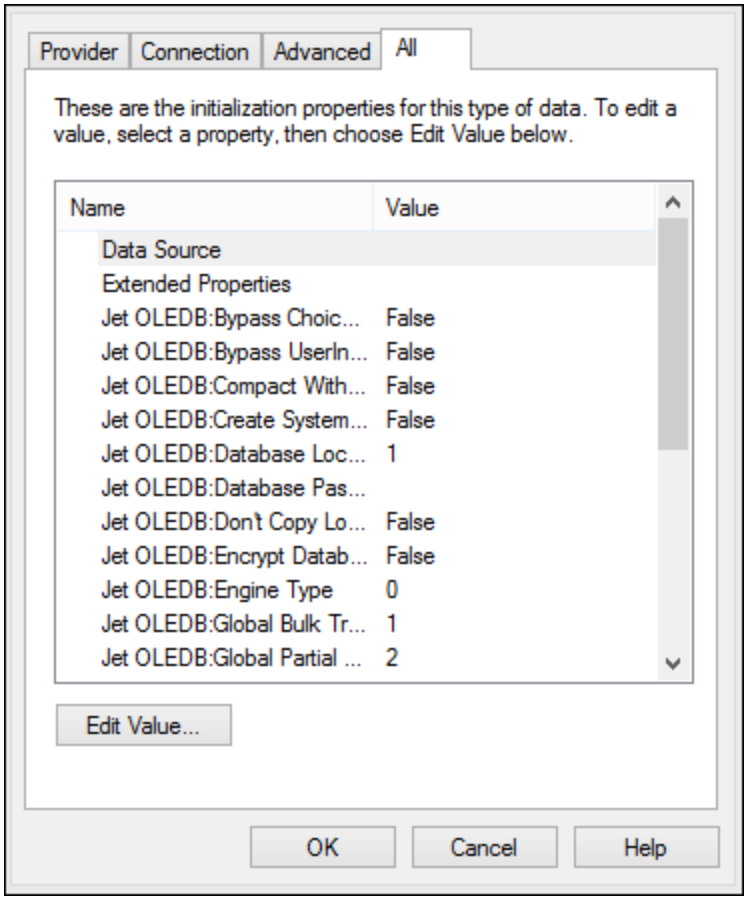


Dialogfeld "Datenverknüpfungseigenschaften"

Eigenschaft	Anmerkungen
Integrated Security	Wenn Sie den Daten-Provider **SQL Server Native Client** auf dem Register **Provider** auswählen, definieren Sie für diese Eigenschaft ein Leerzeichen.
Persist Security Info	Setzen Sie diese Eigenschaft auf **True**.

10.2.3.3 Einrichten der Microsoft Access-Datenverknüpfungseigenschaften

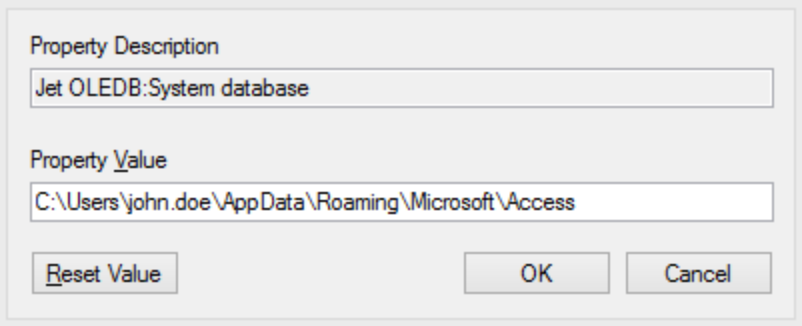
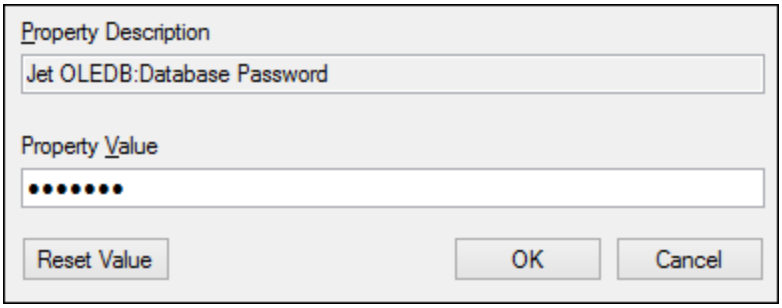
Wenn Sie über ADO[583] eine Verbindung zu einer Microsoft Access-Datenbank herstellen, müssen Sie eventuell die folgenden Datenverknüpfungseigenschaften auf dem Register **Alle** des Dialogfelds "Datenverknüpfungseigenschaften" konfigurieren.



Dialogfeld "Datenverknüpfungseigenschaften"

Eigenschaft	Anmerkungen
Datenquelle	In dieser Eigenschaft ist der Pfad zur Microsoft Access-Datenbankdatei gespeichert. Um Verbindungsprobleme zu vermeiden, wird empfohlen, das UNC (Universal Naming Convention)-Pfadformat zu verwenden, z.B.: `\\anyserver\share$\filepath`
Jet OLEDB:Systemdatenbank	In dieser Eigenschaft ist der Pfad zur Arbeitsgruppen-Informationsdatei gespeichert. Eventuell muss der Wert dieser Eigenschaft explizit definiert werden, bevor Sie eine Verbindung zu einer Microsoft Access-Datenbank herstellen können. Wenn die Verbindung aufgrund eines "Arbeitsgruppen-Informationsdatei"-Fehlers fehlschlägt, suchen Sie die Arbeitsgruppen-Informationsdatei (**System.MDW**) für Ihr Benutzerprofil und setzen Sie den Eigenschaftswert auf den Pfad der Datei **System.MDW**.

© 2016-2022 Altova GmbH

	
Jet OLEDB:Datenbankkennwort	Wenn die Datenbank durch ein Passwort geschützt ist, definieren Sie als Wert dieser Eigenschaft das Datenbank-Passwort. 

10.2.4 ADO.NET-Verbindung

ADO.NET ist eine Gruppe von Microsoft .NET Framework-Bibliotheken für die Interaktion mit Daten, darunter auch mit Daten aus Datenbanken. Für die Verbindung zu einer Datenbank von UModel aus über ADO.NET wird Microsoft .NET Framework 4 oder höher benötigt. Wie unten gezeigt, erfolgt die Verbindung zu einer Datenbank über ADO.NET durch Auswahl eines .NET-Anbieters und Bereitstellung eines Connection String.

Ein .NET-Datenanbieter ist eine Sammlung von Klassen, mit Hilfe derer Sie eine Verbindung zu einem bestimmten Datenquellentyp (z.B. einem SQL Server oder einer Oracle-Datenbank) herstellen können, Befehle daran ausführen und Daten aus dieser Quelle abrufen können, d.h. mit Hilfe von ADO.NET kann eine Applikation wie UModel über einen Datenanbieter mit einer Datenbank kommunizieren. Jeder Datenanbieter ist für den spezifischen Datenquellentyp, für den er entwickelt wurde, optimiert. Es gibt zwei Arten von .NET-Anbietern:

1. Solche, die standardmäßig mit Microsoft .NET Framework bereitgestellt werden.
2. Solche, die von Anbietern gebräuchlicher Datenbanken als Erweiterung zum .NET Framework bereitgestellt werden. ADO.NET-Anbieter dieser Art müssen separat installiert werden und können normalerweise von der Website des entsprechenden Datenbankanbieters heruntergeladen werden.

Anmerkung: Einige ADO.NET-Anbieter werden nicht oder nur eingeschränkt unterstützt. Siehe Anmerkungen zur Unterstützung von ADO.NET 595.

So richten Sie eine ADO.NET-Verbindung ein:

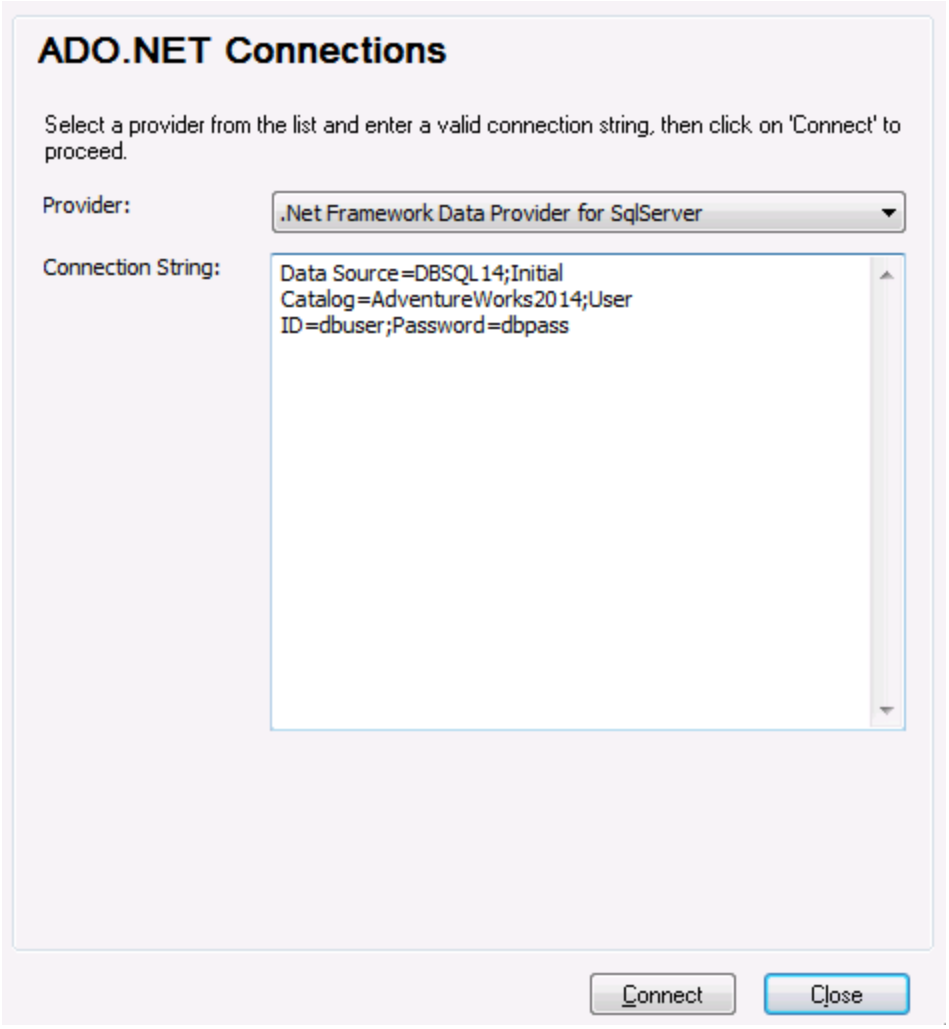
1. Starten Sie den Datenbankverbindungsassistenten[578].
2. Klicken Sie auf **ADO.NET-Verbindungen**.
3. Wählen Sie einen .NET-Datenanbieter aus der Liste aus.

   Die Liste der Anbieter, die standardmäßig mit dem .NET Framework zur Verfügung stehen, wird in der Liste der Anbieter angezeigt. Anbieterspezifische .NET-Datenanbieter stehen in der Liste nur zur Verfügung, wenn sie bereits auf Ihrem System installiert sind. Damit anbieterspezifische .NET-Anbieter zur Verfügung stehen, müssen diese durch Ausführung der vom Datenbankanbieter bereitgestellten .msi- oder .exe-Datei im GAC (Global Assembly Cache) installiert werden.

4. Geben Sie einen Datenbank-Connection String ein. Mit einem Connection String werden die Datenbankverbindungsinformationen in Form von durch Semikola getrennte Schlüssel/Wert-Paare von Verbindungsparametern definiert. Mit einem Connection String wie `Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass` wird z.B. eine Verbindung zur SQL Server-Datenbank `ProductsDB` auf dem Server `DBSQLSERV` unter dem Benutzernamen `dbuser` und mit dem Passwort `dbpass` hergestellt. Sie können einen Connection String erstellen, indem Sie die Schlüssel/Wert-Paare direkt in das Dialogfeld "Connection String" eingeben. Eine weitere Methode ist die Erstellung über Visual Studio (siehe Erstellen eines Connection String in Visual Studio[591]).

   Die Syntax des Connection String ist von dem in der Liste "Anbieter" ausgewählten Anbieter abhängig. Beispiele dazu finden Sie unter Beispiele für ADO.NET Connection Strings[594].

© 2016-2022 Altova GmbH



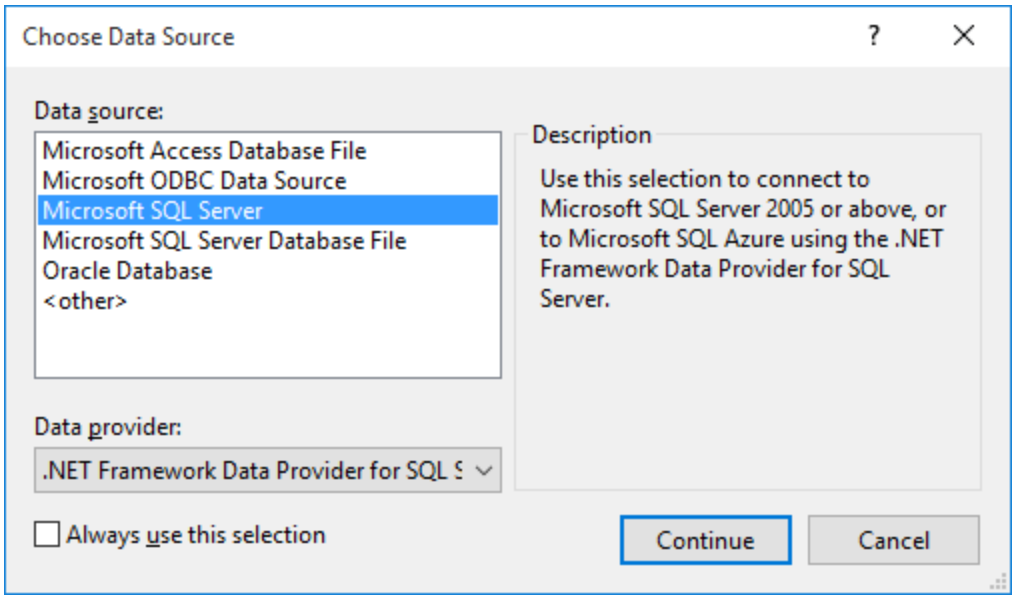
5. Klicken Sie auf **Verbinden**.

10.2.4.1 Erstellen eines Connection String in Visual Studio

Um über ADO.NET eine Verbindung zu einer Datenquelle herstellen zu können, wird ein gültiger Datenbank Connection String benötigt. Im Folgenden wird beschrieben, wie Sie über Visual Studio einen Connection String erstellen.

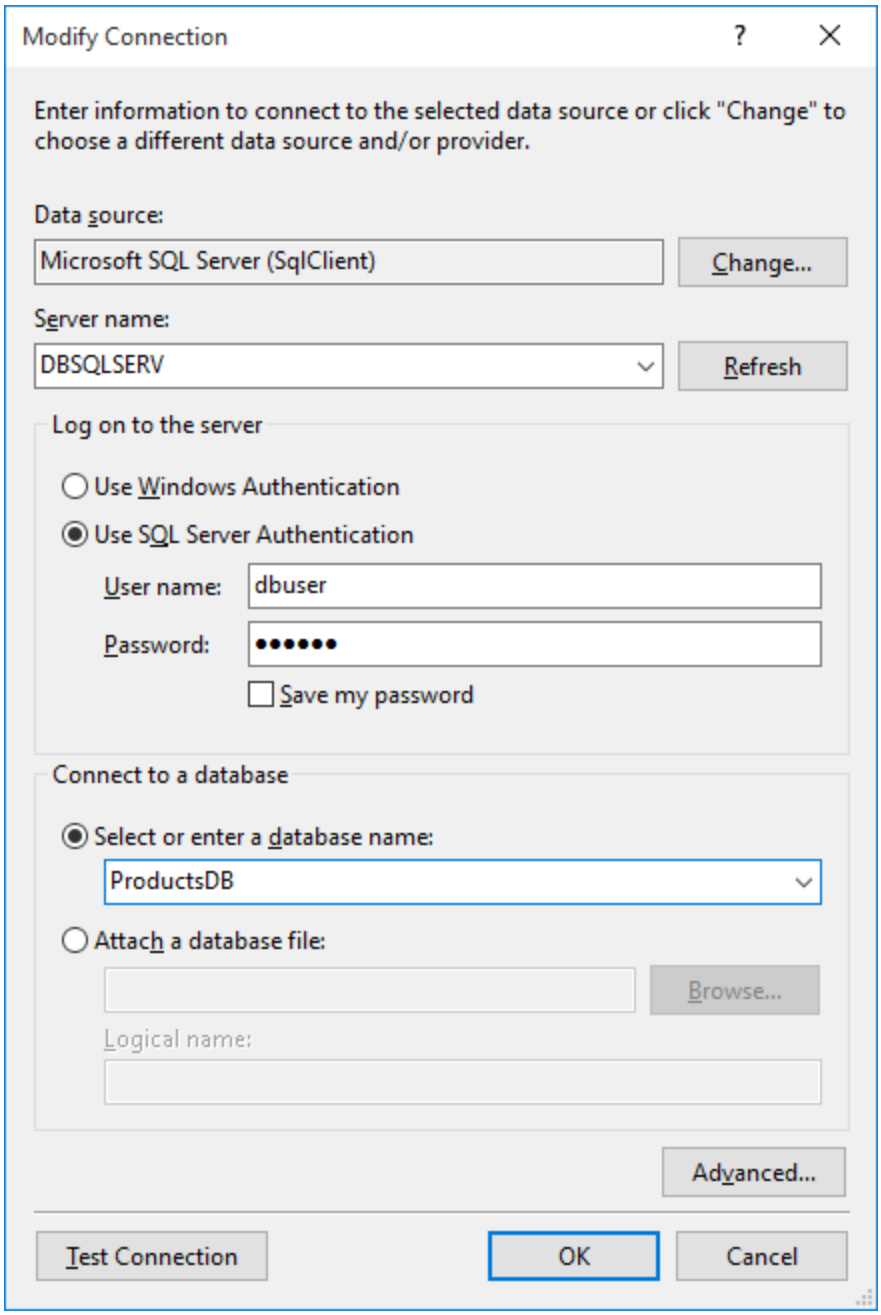
So erstellen Sie einen Connection String in Visual Studio:

1. Klicken Sie im Menü **Extras** auf **Mit Datenbank verbinden**.
2. Wählen Sie eine Datenquelle aus der Liste aus (in diesem Beispiel Microsoft SQL Server). Der Datenanbieter wird auf Basis Ihrer Auswahl automatisch ausgefüllt.



3. Klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH



4. Geben Sie den Server Host-Namen und den Benutzernamen und das Passwort für die Datenbank ein. In diesem Beispiel stellen wir unter Verwendung der SQL Server-Authentifizierung eine Verbindung zur Datenbank `ProductsDB` auf dem Server `DBSQLSERV` her.
5. Klicken Sie auf **OK**.

Wenn die Datenbankverbindung erfolgreich hergestellt wurde, wird sie im Server Explorer-Fenster angezeigt. Sie können das Server-Explorer-Fenster mit dem Menübefehl **Ansicht | Server-Explorer** aufrufen. Um den Datenbank Connection String anzuzeigen, klicken Sie im Server-Explorer-Fenster mit der rechten Maustaste auf die Verbindung und wählen Sie **Eigenschaften**. Der Connection String wird nun im Fenster "Eigenschaften"

von Visual Studio angezeigt. Beachten Sie, dass Sie das Sternchen (*) vor dem Einfügen des String in das Feld "Connection String" von UModel durch das tatsächliche Passwort ersetzen müssen.

10.2.4.2 Beispiele für ADO.NET Connection Strings

Um eine ADO.NET-Verbindung einzurichten, müssen Sie einen ADO.NET-Anbieter aus dem Datenbankverbindungsdialogfeld auswählen und einen Connection String eingeben (siehe auch Einrichten einer ADO.NET-Verbindung 589). Beispiele für ADO.NET Connection Strings für verschiedene Datenbanken sind unter dem jeweiligen .NET-Anbieter aufgelistet.

.NET Data Provider für Teradata

Dieser Anbieter kann von der Teradata-Website (https://downloads.teradata.com/download/connectivity/net-data-provider-for-teradata) heruntergeladen werden. Ein Beispiel-Connection String sieht folgendermaßen aus:

```
Data Source=ServerAddress;User Id=user;Password=password;
```

.NET Framework Data Provider für IBM i

Dieser Anbieter wird im Rahmen von *IBM i Access Client Solutions - Windows Application Package*. installiert. Ein Connection String sieht folgendermaßen aus:

```
DataSource=Serveradresse;UserID=Benutzer;Password=Passwort;DataCompression=True;
```

Nähere Informationen dazu finden Sie in der im obigen Installationspaket inkludierten Hilfedatei ".NET Provider Technical Reference".

.NET Framework Data Provider für MySQL

Dieser Anbieter kann von der MySQL Website (https://dev.mysql.com/downloads/connector/net/) heruntergeladen werden. Ein Beispiel-Connection String sieht folgendermaßen aus:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

Siehe auch: https://dev.mysql.com/doc/connector-net/en/connector-net-programming-connecting-connection-string.html

.NET Framework Data Provider für SQL Server

Ein Beispiel-Connection String sieht folgendermaßen aus:

```
Data Source=DBSQLSERV;Initial Catalog=ProductsDB;User ID=dbuser;Password=dbpass
```

Siehe auch: https://msdn.microsoft.com/en-us/library/ms254500(v=vs.110).aspx

© 2016-2022 Altova GmbH

IBM DB2 Data Provider 10.1.2 für .NET Framework 4.0

```
Database=PRODUCTS;UID=user;Password=password;Server=localhost:50000;
```

Anmerkung: Dieser Anbieter wird normalerweise mit dem IBM DB2 Data Server Client-Paket installiert. Wenn der Anbieter nach Installation des IBM DB2 Date Server Client-Pakets nicht in der Liste der ADO.NET-Anbieter aufgelistet wird, lesen Sie nach unter: https://www-01.ibm.com/support/docview.wss?uid=swg21429586.

Siehe auch:
https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.swg.im.dbclient.adonet.ref.doc/doc/DB2ConnectionClassConnectionStringProperty.html

Oracle Data Provider für .NET (ODP.NET)

Das Installationspaket, das den ODP.NET-Anbieter enthält, kann von der Oracle Website heruntergeladen werden (siehe http://www.oracle.com/technetwork/topics/dotnet/downloads/index.html). Ein Beispiel-Connection String sieht folgendermaßen aus:

```
Data Source=DSORCL;User Id=user;Password=password;
```

`DSORCL` ist hierbei der Name der Datenquelle, der auf einen in der Datei **tnsnames.ora** definierten Oracle-Dienstnamen verweist. Eine Beschreibung dazu finden Sie unter Verbinden mit Oracle (ODBC) 642.

Um eine Verbindung herzustellen, ohne einen Dienstnamen in der Datei **tnsnames.ora** zu konfigurieren, verwenden Sie einen String wie den folgenden:

```
Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=host)(PORT=port)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=MyOracleSID)));User
Id=user;Password=password;
```

Siehe auch: https://docs.oracle.com/cd/B28359_01/win.111/b28375/featConnecting.htm

10.2.4.3 Anmerkungen zur Unterstützung von ADO.NET

In der folgenden Tabelle sind bekannte ADO.NET-Datenbanktreiber aufgelistet, die derzeit in UModel nicht oder nur eingeschränkt unterstützt werden.

Datenbank	Treiber	Anmerkungen zur Unterstützung
Alle Datenbanken	**.Net Framework Data Provider for ODBC**	Eingeschränkte Unterstützung. Bekannte Probleme bei Microsoft Access-Verbindungen. Es wird empfohlen stattdessen direkte ODBC-Verbindungen zu verwenden.

Datenbank	Treiber	Anmerkungen zur Unterstützung
	.Net Framework Data Provider for OleDb	Eingeschränkte Unterstützung. Bekannte Probleme bei Microsoft Access-Verbindungen. Es wird empfohlen stattdessen direkte ADO-Verbindungen zu verwenden.
Firebird	**Firebird ADO.NET Data Provider**	Eingeschränkte Unterstützung. Es wird empfohlen stattdessen ODBC oder JDBC zu verwenden.
Informix	**IBM Informix Data Provider for .NET Framework 4.0**	Wird nicht unterstützt. Verwenden Sie stattdessen **DB2 Data Server Provider**.
IBM DB2 for i (iSeries)	**.Net Framework Data Provider for i5/OS**	Wird nicht unterstützt. Verwenden Sie stattdessen den im Rahmen des *IBM i Access Client Solutions - Windows Application-Pakets* bereitgestellten **.Net Framework Data Provider for IBM i**-Treiber.
Oracle	**.Net Framework Data Provider for Oracle**	Eingeschränkte Unterstützung. Der Treiber wird zwar mit dem .NET Framework zur Verfügung gestellt, doch wird von Microsoft von der Verwendung abgeraten, da er veraltet ist.
PostgreSQL	**-**	Es werden keine ADO.NET-Treiber für diesen Anbieter unterstützt. Verwenden Sie stattdessen eine native Verbindung.
Sybase	**-**	Es werden keine ADO.NET-Treiber für diesen Anbieter unterstützt.

10.2.5 ODBC-Verbindung

ODBC (Open Database Connectivity) ist eine häufig verwendete Datenzugriffstechnologie, mit der Sie von UModel aus eine Verbindung zu einer Datenbank herstellen können. ODBC kann entweder als primäre Verbindungsmethode oder als Alternative zu nativen Verbindungen oder Verbindungen über OLE DB oder JDBC verwendet werden.

Um über ODBC eine Datenbankverbindung herzustellen, müssen Sie zuerst einen ODBC-Datenquellennamen (DSN = Data Source Name) auf dem Betriebssystem erstellen. Wenn bereits ein DSN erstellt wurde - möglicherweise von einem anderen Benutzer auf dem Betriebssystem - entfällt dieser Schritt. Der DSN bietet eine einheitliche Methode, um die Datenbankverbindung für jede ODBC-fähige Client-Applikation auf dem Betriebssystem einschließlich UModel zu beschreiben. Es gibt folgende Arten von DSN:

- System-DSN
- Benutzer-DSN
- Datei-DSN

© 2016-2022 Altova GmbH

Eine *Systemdatenquelle* kann von allen Benutzern mit Rechten auf dem Betriebssystem aufgerufen werden. Eine *Benutzerdatenquelle* steht nur dem Benutzer, der sie erstellt hat, zur Verfügung. Wenn Sie einen *Datei-DSN* erstellen, wird die Datenquelle als Datei mit der Erweiterung .dsn erstellt, die Sie gemeinsam mit anderen Benutzern verwenden können, vorausgesetzt diese haben die für die Datenquelle erforderlichen Treiber installiert.

Alle auf Ihrem Rechner bereits verfügbaren DSNs werden im Dialogfeld "Datenbankverbindung" aufgelistet, wenn Sie im Dialogfeld "ODBC-Verbindungen" auf **ODBC-Verbindungen** klicken.



Dialogfeld "ODBC-Verbindungen"

Wenn zur gewünschten Datenbank kein DSN vorhanden ist, hilft Ihnen der UModel Datenbank-Verbindungsassistent dabei, einen zu erstellen; Sie können den DSN aber auch direkt in Ihrem Windows-Betriebssystem erstellen. Stellen Sie in jedem Fall, bevor Sie fortfahren, sicher, dass der für die Datenbank erforderliche ODBC-Treiber in der Liste der ODBC-Treiber zur Verfügung steht (siehe Anzeigen der verfügbaren ODBC-Treiber 598).

So stellen Sie mit Hilfe eines neuen DSN eine Verbindung her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie im Dialogfeld "Datenbankverbindung" auf **ODBC-Verbindungen**.
3. Wählen Sie einen Datenquellentyp aus (Benutzer-DSN, System-DSN, Datei-DSN).

> Zur Erstellung eines System-DSN benötigen Sie Administratorrechte auf Ihrem Betriebssystem und UModel muss von Ihnen als Adminstrator ausgeführt werden.

4. Klicken Sie auf **Hinzufügen** .
5. Wählen Sie einen Treiber aus und klicken Sie anschließend auf **Benutzer-DSN** oder **System-DSN** (je nachdem, welche Art von DSN Sie erstellen möchten). Wenn der Treiber für Ihre Datenbank nicht aufgelistet ist, laden Sie ihn vom Datenbankanbieter herunter und installieren Sie ihn (siehe Übersicht über Datenbanktreiber 580).
6. Füllen Sie im Dialogfeld, das daraufhin angezeigt wird, alle treiberspezifischen Informationen aus, um die Verbindung fertig zu konfigurieren.

Damit eine Verbindung hergestellt werden kann, müssen Sie den Host-Namen (oder die IP-Adresse) des Datenbankservers sowie den Datenbank-Benutzernamen und das Passwort dafür angeben. Eventuell gibt es weitere optionale je nach Anbieter unterschiedliche Verbindungsparameter. Nähere Informationen zu den Parametern für die einzelnen Verbindungsmethoden finden Sie in der Dokumentation des Treiberanbieters. Sobald der DSN erstellt wurde, steht er in der Liste der Datenquellennamen zur Verfügung. Auf diese Art können Sie die Datenbankverbindungsinformationen jedes Mal, wenn Sie eine Verbindung zur Datenbank herstellen, wiederverwenden. Beachten Sie, dass Benutzer-DSNs zur Liste der Benutzer-DSNs hinzugefügt werden, während System-DSNs zur Liste der System-DSNs hinzugefügt werden.

So stellen Sie über einen vorhandenen DSN eine Verbindung her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Wählen Sie einen Datenquellentyp aus (Benutzer-DSN, System-DSN, Datei-DSN).
4. Klicken Sie auf den vorhandenen DSN-Eintrag und anschließend auf **Verbinden.**

So erzeugen Sie auf Basis einer vorhandenen .dsn-Datei einen Connection String:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Wählen Sie **Connection String erstellen** und klicken Sie anschließend auf **Erzeugen**.
4. Wenn Sie den Connection String mit Hilfe eines Datei-DSN erstellen möchten, klicken Sie auf das Register **Dateidatenquelle**. Klicken Sie andernfalls auf das Register **Computerdatenquelle**. (System-DSNs und Benutzer-DSNs werden als "Computerdatenquelle" bezeichnet.)
5. Wählen sie die benötigte .dsn-Datei aus und klicken Sie auf **OK.**

So stellen Sie die Verbindung mittels eines bereits definierten Connection String her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Wählen Sie **Connection String erstellen-**
4. Kopieren Sie den Connection String in das entsprechende Feld ein und klicken Sie auf **Verbinden**.

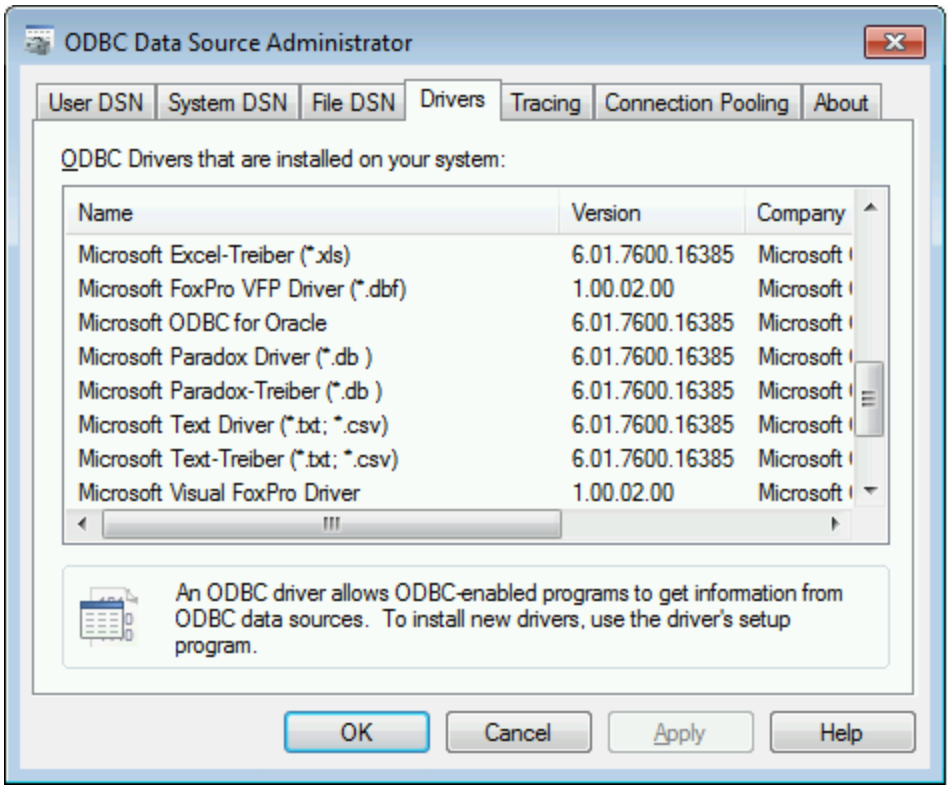
10.2.5.1 Verfügbare ODBC-Treiber

Im ODBC-Datenquellen-Administrator können Sie die auf Ihrem Betriebssystem verfügbaren ODBC-Treiber anzeigen. Sie können den ODBC-Datenquellen-Administrator (**Odbcad32.exe**) über die Windows-Systemsteuerung unter **Verwaltung** aufrufen. Auf 64-Bit-Betriebssystemen gibt es zwei Versionen dieser ausführbaren Datei:

© 2016-2022 Altova GmbH

- Die 32-Bit-Version der Datei **Odbcad32.exe** befindet sich im Verzeichnis **C:\Windows\SysWoW64** (wenn **C:** Ihr Systemlaufwerk ist).
- Die 64-Bit-Version der Datei **Odbcad32.exe** befindet sich im Verzeichnis **C:\Windows\System32**.

Die installierten 32-Bit-Datenbanktreiber sind in der 32-Bit-Version des ODBC-Datenquellen-Administrators zu sehen, während die 64-Bit-Treiber in der 64-Bit-Version angezeigt werden. Vergewissern Sie sich daher, dass Sie die richtige Version des ODBC-Datenquellen-Administrators geöffnet haben, wenn Sie die Datenbanktreiber überprüfen.



ODBC-Datenquellen-Administrator

Wenn der Treiber für die gewünschte Datenbank in der Liste nicht vorhanden ist oder wenn Sie eine anderen Treiber hinzufügen möchten, müssen Sie diesen von der Webseite des Datenbankanbieters herunterladen (siehe Übersicht über Datenbanktreiber 580). Sobald der ODBC-Treiber auf Ihrem System verfügbar ist, können Sie damit ODBC-Verbindungen herstellen (siehe Einrichten der ODBC-Verbindung 596).

10.2.6 JDBC-Verbindung

JDBC (Java Database Connectivity) ist eine Datenbankzugriffsschnittstelle, die Teil der Java-Software-Plattform von Oracle ist. JDBC-Verbindungen beanspruchen im Allgemeinen mehr Ressourcen als ODBC-Verbindungen, bieten aber Funktionen, die über ODBC nicht zur Verfügung stehen.

Voraussetzungen

- JRE (Java Runtime Environment) oder Java Development Kit (JDK) muss installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel

ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.

- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- Die JDBC-Treiber des Datenbankanbieters müssen installiert sein. Dabei kann es sich um JDBC-Treiber, die im Rahmen der Datenbankclient-Installation installiert wurden oder um separat heruntergeladene JDBC-Bibliotheken (.jar-Dateien) (falls verfügbar und von der Datenbank unterstützt) handeln, siehe auch Beispiele für Datenbankverbindungen 607.
- Die `CLASSPATH`-Umgebungsvariable muss den Pfad zum JDBC-Treiber auf Ihrem Windows-Betriebssystem enthalten. Diese Variable wird unter Umständen bei der Installation einiger Datenbank Clients automatisch konfiguriert, siehe auch Konfigurieren des CLASSPATH 602.

Verbinden mit SQL Server über JDBC mit Windows-Anmeldeinformationen

Wenn Sie über JDBC mit Windows-Anmeldeinformationen (integrierte Sicherheit) eine Verbindung zu SQL Server herstellen, beachten Sie die folgenden Punkte:

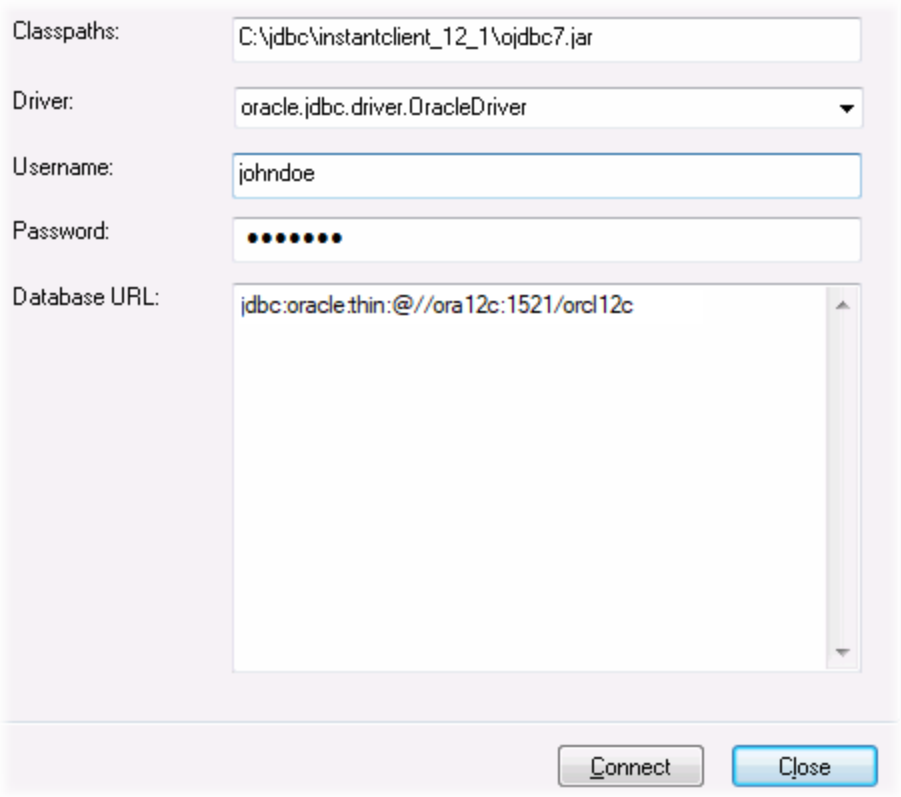
- Die im JDBC-Treiberpaket enthaltene Datei **sqljdbc_auth.dll** muss in ein Verzeichnis kopiert werden, das sich in der System PATH-Umgebungsvariablen befindet. Es gibt zwei solche Dateien, eine für die x86- und eine für die x64-Plattform. Vergewissern Sie sich, dass Sie die Ihrer JDK-Plattform entsprechende zum PATH hinzufügen.
- Der JDBC Connection String muss die Eigenschaft **`integratedSecurity=true`** enthalten.

Nähere Informationen dazu finden Sie in der *Microsoft-Dokumentation zu JDBC-Treibern für SQL Server* unter https://docs.microsoft.com/de-de/sql/connect/jdbc/building-the-connection-url?view=sql-server-ver15.

Einrichten einer JDBC-Verbindung

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie optional in das Textfeld "Classpaths" eine durch Semikola getrennte Liste von .jar-Dateipfaden ein. Die hier eingegebenen .jar-Bibliotheken werden zusätzlich zu den bereits in der Umgebungsvariablen `CLASSPATH` definierten in die Umgebung geladen. Nachdem Sie Ihre Eingaben ins Textfeld "Classpaths" beendet haben, werden alle in den .jar-Quellbibliotheken gefundenen JDBC-Treiber automatisch zur Liste "Treiber" (siehe nächster Schritt) hinzugefügt.

© 2016-2022 Altova GmbH



4. Wählen Sie neben "Treiber" einen JDBC-Treiber aus der Liste aus oder geben Sie einen Java-Klassennamen ein. Beachten Sie, dass diese Liste alle über die Umgebungsvariable CLASSPATH konfigurierten JDBC-Treiber (siehe Konfigurieren des CLASSPATH 602) sowie die im Textfeld "Classpaths" gefundenen JDBC-Treiber enthält.

> Die in der CLASSPATH-Variablen definierten JDBC-Treiberpfade sowie alle direkt in das Datenbankverbindungsdialogfeld eingegebenen Pfade zu .jar-Dateien werden alle der Java Virtual Machine (JVM) zur Verfügung gestellt. Die JVM entscheidet anschließend, welche Treiber zur Herstellung einer Verbindung verwendet werden sollen. Es wird empfohlen, die in die JVM geladenen Java-Klassen im Auge zu behalten, damit es zu keinen potenziellen JDBC-Treiberkonflikten und unerwarteten Ergebnissen bei der Herstellung der Datenbankverbindung kommt.

5. Geben Sie den Benutzernamen und das Passwort für die Datenbank in die entsprechenden Felder ein.
6. Geben Sie im Textfeld "Datenbank-URL" die JDBC Connection-URL (String) im datenbanktypspezifischen Format ein. In der folgenden Tabelle sehen Sie die Syntaxvorgaben für die JDBC Connection-URLs (Strings) für gebräuchliche Datenbanktypen.

Datenbank	JDBC-Verbindungs-URL
Firebird	`jdbc:firebirdsql://<host>[:<port>]/<database path or alias>`
IBM DB2	`jdbc:db2://hostName:port/databaseName`

IBM DB2 for i	jdbc:as400://**[host]**
IBM Informix	jdbc:informix-sqli://**hostName**:**port**/**databaseName**:INFORMIXSERVER=**myserver**
MariaDB	jdbc:mariadb://**hostName**:**port**/**databaseName**
Microsoft SQL Server	jdbc:sqlserver://**hostName**:**port**;**databaseName**=name
MySQL	jdbc:mysql://**hostName**:**port**/**databaseName**
Oracle	jdbc:oracle:thin:@**hostName**:**port**:**SID** jdbc:oracle:thin:@//**hostName**:**port**/**service**
Oracle XML DB	jdbc:oracle:oci:@//**hostName**:**port**:**service**
PostgreSQL	jdbc:datadirect:openedge://**host**:**port**;databaseName=**db_name**
Progress OpenEdge	jdbc:datadirect:openedge://**host**:**port**;databaseName=**db_name**
Sybase	jdbc:sybase:Tds:**hostName**:**port**/**databaseName**
Teradata	jdbc:teradata://**databaseServerName**

Anmerkung: Bei den oben aufgelisteten Formaten sind auch Syntaxvarianten möglich (die Datenbank-URL kann eventuell ohne Port oder einschließlich Benutzernamen und Datenbank-Passwort angegeben werden). Nähere Informationen dazu finden Sie in der Dokumentation des jeweiligen Datenbankanbieters.

7. Klicken Sie auf **Verbinden**.

10.2.6.1 Konfigurieren des CLASSPATH

Mit Hilfe der CLASSPATH-Umgebungsvariablen findet das Java Runtime Environment (JRE) bzw. der Java Development Kit (JDK) Java-Klassen und andere Ressourcendateien auf Ihrem Betriebssystem. Bei Herstellung einer Datenbankverbindung über JDBC muss diese Variable so konfiguriert werden, dass sie den Pfad zum JDBC-Treiber auf Ihrem Betriebssystem und in einigen Fällen den Pfad zur zusätzlichen datenbanktypspezifischen Bibliotheken enthält.

In der folgenden Tabelle sind typische Beispieldateipfade aufgelistet, die in der CLASSPATH-Variablen enthalten sein müssen. Sie müssen diese Informationen eventuell anhand des JDBC-Treibers auf Ihrem System, des JDBC-Treibernamens sowie der JRE/JDK-Version auf Ihrem Betriebssystem anpassen. Um Verbindungsprobleme zu vermeiden, lesen Sie die Installationsanleitung zu dem auf Ihrem Betriebssystem installierten JDBC-Treiber und führen Sie alle vor oder nach der Installation erforderlichen Schritte durch.

Datenbank	**CLASSPATH-Beispieleinträge**
Firebird	C:\Programme\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar

© 2016-2022 Altova GmbH

IBM DB2	`C:\Programme (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Programme (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;`
IBM DB2 for i	`C:\jt400\jt400.jar;`
IBM Informix	`C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;`
Microsoft SQL Server	`C:\Programme\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar`
MariaDB	**<installation directory>**\mariadb-java-client-2.2.0.jar
MySQL	mysql-connector-java-*version*-bin.jar;
Oracle	**ORACLE_HOME**\jdbc\lib\ojdbc6.jar;
Oracle (mit XML DB)	**ORACLE_HOME**\jdbc\lib\ojdbc6.jar;**ORACLE_HOME**\LIB\xmlparserv2.jar;**ORACLE_HOME**\RDBMS\jlib\xdb.jar;
PostgreSQL	**<installation directory>**\postgresql.jar
Progress OpenEdge	%DLC%\java\openedge.jar;%DLC%\java\pool.jar; Anmerkung: Angenommen, Progress OpenEdge SDK ist auf dem Rechner installiert, so ist %DLC% das Verzeichnis, in dem OpenEdge installiert ist.
Sybase	`C:\sybase\jConnect-7_0\classes\jconn4.jar`
Teradata	**<installation directory>**\tdgssconfig.jar;**<installation directory>**\terajdbc4.jar

- Wenn Sie die CLASSPATH-Variable ändern, kann sich dies auf das Verhalten von Java-Applikationen auf Ihrem Rechner auswirken. Lesen Sie dazu die Java-Dokumentation.
- Umgebungsvariablen können benutzer- oder systemspezifisch sein. Um System-Umgebungsvariablen zu ändern, benötigen Sie Administratorrechte auf Ihrem Betriebssystem.
- Nachdem Sie die Umgebungsvariable geändert haben, starten Sie alle laufenden Programme neu, damit die Änderungen wirksam werden. Alternativ dazu können Sie sich am Betriebssystem auch ab- und wieder anmelden oder dieses neu starten.

So konfigurieren Sie den CLASSPATH unter Windows 7:

1. Öffnen Sie das **Startmenü** und klicken Sie mit der rechten Maustaste auf **Computer**.
2. Klicken Sie auf **Eigenschaften**.
3. Klicken Sie auf **Erweiterte Systemeinstellungen**.
4. Klicken Sie auf dem Register **Erweitert** auf **Umgebungsvariablen.**
5. Gehen Sie unter Benutzer- oder System-Umgebungsvariablen zur CLASSPATH-Variablen und klicken Sie anschließend auf **Bearbeiten**. Wenn die CLASSPATH-Variable nicht vorhanden ist, klicken Sie auf **Neu**, um sie zu erstellen.
6. Bearbeiten Sie den Wert der Variablen, damit sie den Pfad enthält, auf dem sich auf Ihrem Betriebssystem der JDBC-Treiber befindet. Verwenden Sie das Semikolon (;), um den JDBC-Treiberpfad von anderen in der CLASSPATH-Variablen bereits vorhandenen Pfaden zu trennen.

So konfigurieren Sie den CLASSPATH unter Windows 10:

1. Drücken Sie die Windows-Taste und beginnen Sie mit der Eingabe von "Umgebungsvariablen".
2. Klicken Sie auf den Vorschlag **Systemumgebungsvariablen bearbeiten**.
3. Klicken Sie auf **Umgebungsvariablen.**
4. Gehen Sie unter Benutzer- oder System-Umgebungsvariablen zur CLASSPATH-Variablen und klicken Sie anschließend auf **Bearbeiten**. Wenn die CLASSPATH-Variable nicht vorhanden ist, klicken Sie auf **Neu**, um sie zu erstellen.
5. Bearbeiten Sie den Wert der Variablen, damit sie den Pfad enthält, auf dem sich auf Ihrem Betriebssystem der JDBC-Treiber befindet. Verwenden Sie das Semikolon (;), um den JDBC-Treiberpfad von anderen in der CLASSPATH-Variablen bereits vorhandenen Pfaden zu trennen.

10.2.7 PostgreSQL-Verbindung

Verbindungen zu PostgreSQL-Datenbanken können entweder als native Verbindung oder über ODBC, JDBC und andere Treiber hergestellt werden. Der Vorteil einer nativen Verbindung ist, dass keine Treiber auf Ihrem System installiert werden müssen.

Wenn Sie die Verbindung lieber über einen nicht nativen Treiber herstellen möchten, lesen Sie in den folgenden Kapiteln nach:

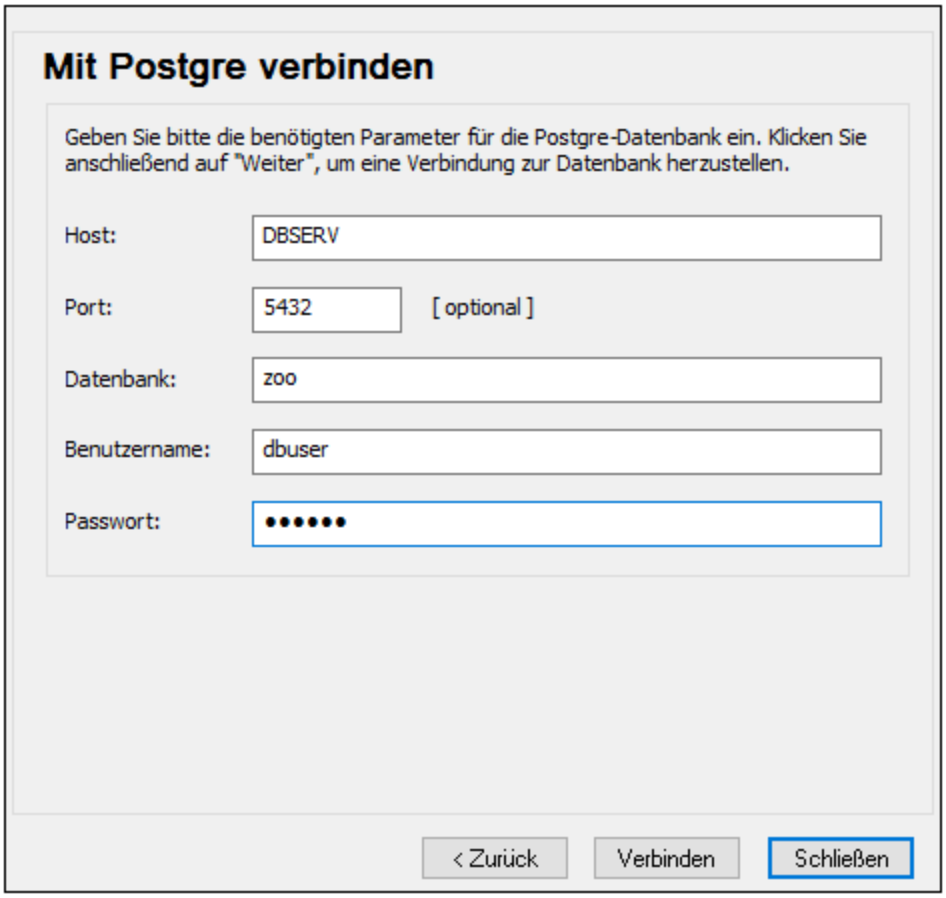
- Einrichten einer JDBC-Verbindung 599
- Verbinden mit PostgreSQL (ODBC) 647

Wenn Sie eine native Verbindung zu PostgreSQL herstellen möchten, befolgen Sie die unten stehenden Anleitung. Um fortzufahren, benötigen Sie die folgenden Informationen: Host-Name, Port, Datenbankname, Benutzername und Passwort.

So richten Sie eine native PostgreSQL-Verbindung ein:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **PostgreSQL-Verbindungen**.
3. Geben Sie den Host (*localhost*, wenn PostgreSQL auf demselben Rechner ausgeführt wird), Port (normalerweise 5432, dies ist optional), den Datenbanknamen, den Benutzernamen und das Passwort in die entsprechenden Felder ein.

© 2016-2022 Altova GmbH



4. Klicken Sie auf **Verbinden**.

Wenn sich der PostgreSQL-Datenbankserver auf einem anderen Rechner befindet, beachten Sie die folgenden Punkte:

- Der PostgreSQL-Datenbankserver muss so konfiguriert sein, dass er Verbindungen von Clients zulässt. Insbesondere muss die Datei **pg_hba.conf** so konfiguriert werden, dass sie nicht lokale Verbindungen zulässt. Außerdem muss die Datei **postgresql.conf** so konfiguriert werden, dass sie eine listen-Verbindung zu bestimmten ID-Adressen und einem bestimmten Port hat. Nähere Informationen dazu finden Sie in der Dokumentation zu PostgreSQL (https://www.postgresql.org/docs/9.5/static/client-authentication-problems.html).
- Der Server-Rechner muss so konfiguriert sein, dass er Verbindungen am angegebenen Port (normalerweise 5432) durch die Firewall zulässt. So müssen Sie z.B. eventuell auf einem Windows-Datenbankserver eine Regel erstellen, damit Verbindungen über Port 5432 durch die Firewall zugelassen werden. Wählen Sie dazu **Systemsteuerung > Windows Firewall > Erweiterte Einstellungen > Eingehende Regeln**.

10.2.8 SQLite-Verbindung

SQLite ist ein dateibasierter, eigenständiger Datenbanktyp, der dadurch ideal für Szenarien geeignet ist, in denen Portabilität und einfache Konfiguration wichtig sind. Da SQLite-Datenbanken von UModel nativ

unterstützt werden, müssen zur Herstellung einer Verbindung mit SQLite-Datenbanken keine Treiber installiert werden.

Anmerkungen zur Unterstützung von SQLite-Datenbanken

- Für SQLite-Datenbanken kann auf Linux-Systemen kein Timeout für eine Anweisungsausführung definiert werden.
- Die Volltextsuche in Tabellen wird nicht unterstützt.
- Bei SQLite können in jeder Zeile einer Tabelle Werte unterschiedliche Datentypen verwendet werden. Alle verarbeiteten Werte müssen mit dem deklarierten Spaltentyp kompatibel sein; daher können unerwartete Werte abgerufen werden und Laufzeitfehler auftreten, wenn der Wert in der Zeile einer SQLite-Datenbank nicht mit dem deklarierten Spaltentyp übereinstimmt.

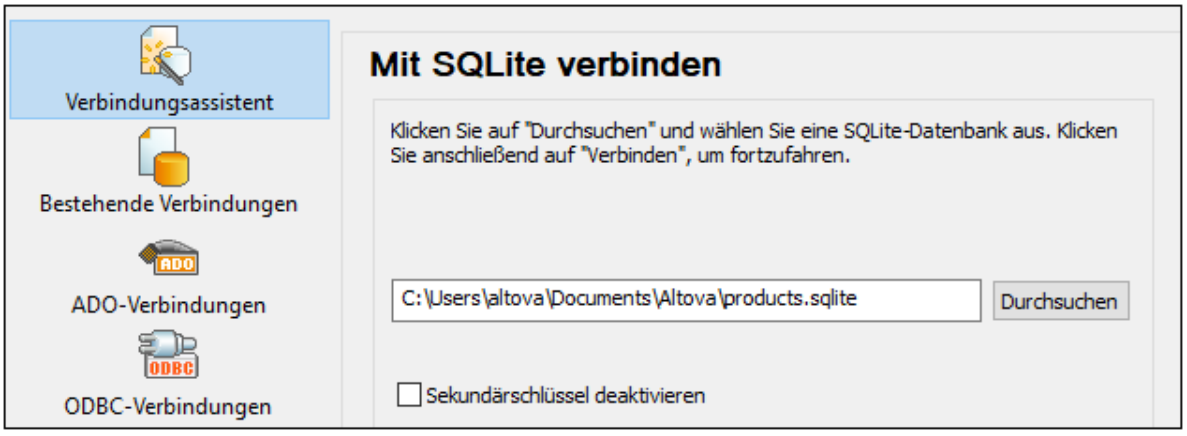
Achtung

Es wird empfohlen, Tabellen mit dem Schlüsselwort `STRICT` zu erstellen, um ein besser vorhersehbares Verhalten Ihrer Daten sicherzustellen. Andernfalls können Daten eventuell nicht korrekt gelesen oder geschrieben werden, wenn eine Spalte Werte unterschiedlichen Typs enthält. Nähere Informationen zu STRICT-Tabellen finden Sie in der SQLite-Dokumentation.

10.2.8.1 Herstellen einer Verbindung zu einer bestehenden SQLite-Datenbank

So stellen Sie eine Verbindung zu einer vorhandenen SQLite-Datenbank her:

1. Starten Sie den Datenbankverbindungsassistenten (siehe Starten des Datenbankverbindungsassistenten 578).
2. Wählen Sie **SQLite** aus und klicken Sie auf **Weiter**.
3. Navigieren Sie zur SQLite-Datenbankdatei oder geben Sie (entweder den relativen oder den absoluten) Pfad zur Datenbank ein. Die Schaltfläche **Verbinden** wird aktiv, sobald Sie den Pfad zur SQLite-Datenbankdatei eingegeben haben.



 © 2016-2022 Altova GmbH

4. Aktivieren Sie optional das Kontrollkästchen **Sekundärschlüssel deaktivieren**, siehe Sekundärschlüssel-Constraints[605].
5. Klicken Sie auf **Verbinden**.

10.2.9 Beispiele für Datenbankverbindungen

In diesem Abschnitt sind Beispiele dafür beschrieben, wie Sie von UModel aus über ADO, ODBC oder JDBC eine Verbindung zu einer Datenbank herstellen. Die Beispiele für ADO.NET-Verbindungen sind separat aufgelistet, siehe Beispiele für ADO.NET Connection Strings[594]. Anleitungen dazu, wie Sie eine native Verbindung zu PostgreSQL und SQLite herstellen, finden Sie unter Einrichten einer PostgreSQL-Verbindung[604] bzw. Einrichten einer SQLite-Verbindung[605].

Beachten Sie die folgenden Punkte:

- Wenn Ihre Windows-Konfiguration, Ihre Netzwerkumgebung und Ihre Datenbank Client- oder Server-Software nicht genau der in den Beispielen beschriebenen Konfiguration entsprechen, weicht die Vorgangsweise eventuell etwas von der in den Beispielen beschriebenen ab.
- Bei den meisten Datenbanktypen kann die Verbindung über unterschiedliche Datenzugriffstechnologien (ADO, ADO.NET, ODBC, JDBC) oder Treiber hergestellt werden. Das Verhalten der Datenbankverbindung, die verfügbaren Funktionalitäten und Einschränkungen hängen vom ausgewählten Treiber, (gegebenenfalls) der Datenbank Client-Software und eventuellen zusätzlich außerhalb von UModel konfigurierten Verbindungsparametern ab.

10.2.9.1 Firebird (JDBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einem Firebird Datenbankserver mittels JDBC.

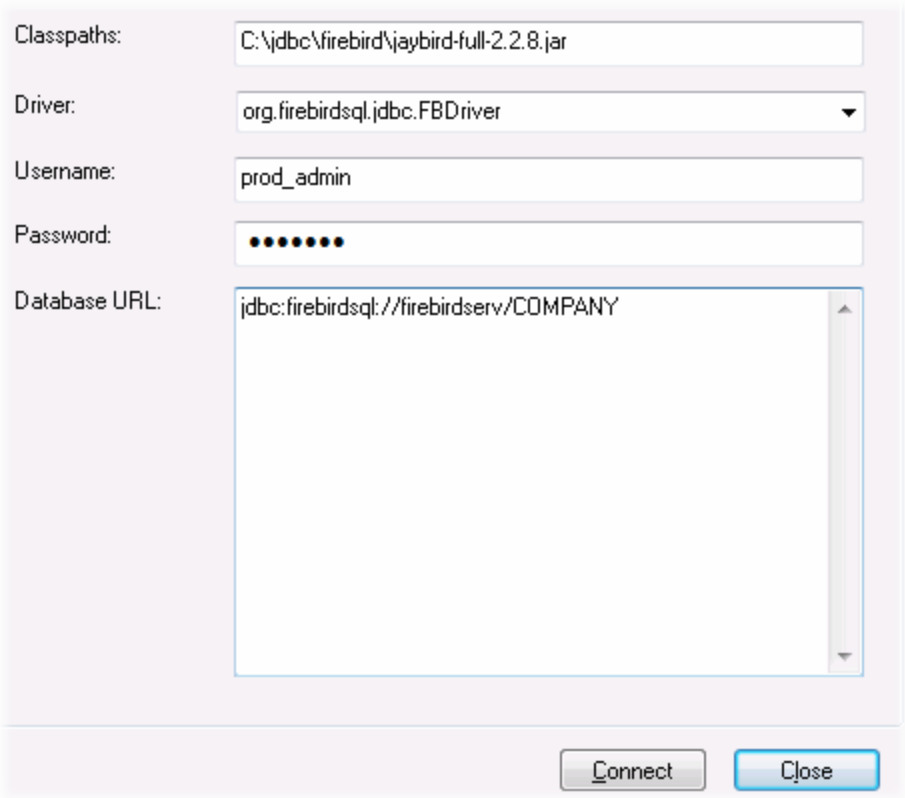
Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen[788]; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- Der Firebird JDBC-Treiber muss auf Ihrem Betriebssystem verfügbar sein (Er hat die Form einer .jar-Datei, die eine Verbindung zur Datenbank herstellt). Der Treiber kann von der Firebird Website (https://www.firebirdsql.org/) heruntergeladen werden. In diesem Beispiel wird *Jaybird 2.2.8* verwendet.
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Host, Datenbankpfad oder Alias, Benutzername und Passwort.

So stellen Sie über JDBC eine Verbindung zu Firebird her:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **JDBC-Verbindungen**.

3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. Falls nötig, können Sie auch eine durch Semikola getrennte Liste von .jar-Dateipfaden eingeben. Die benötigte .jar-Datei in diesem Beispiel befindet sich unter dem folgenden Pfad: **C:\jdbc\firebird\jaybird-full-2.2.8.jar**. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH 602).
4. Wählen Sie im Feld "Treiber" **org.firebirdsql.jdbc.FBDriver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpaths" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).



5. Geben Sie den Benutzernamen und das Passwort in die entsprechenden Textfelder ein.
6. Geben Sie in das Textfeld "Datenbank-URL" den Connection String zum Datenbankserver ein, indem Sie die hervorgehobenen Werte durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:firebirdsql://<host>[:<port>]/<database path or alias>
```

7. Klicken Sie auf **Verbinden**.

10.2.9.2 Firebird (ODBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einer Firebird 2.5.4-Datenbank auf einem Linux Server.

© 2016-2022 Altova GmbH

Voraussetzungen:

- Der Firebird-Datenbankserver akzeptiert aufgrund seiner Konfiguration TCP/IP-Verbindungen von Clients.
- Der Firebird ODBC-Treiber muss auf Ihrem Betriebssystem installiert sein. In diesem Beispiel wird der Firebirg ODBC-Treiber Version 2.0.3.154 verwendet, der von der Firebird Website (https://www.firebirdsql.org/) heruntergeladen wurde.
- Der Firebird Client muss auf Ihrem Betriebssystem installiert sein. Beachten Sie, dass für den Firebird 2.5.4 Client kein eigenständiger Installer verfügbar ist; der Client ist Teil des Firebird Server-Installationspakets, welches Sie unter "Windows executable installer for full Superclassic/Classic or Superserver" von der Firebird Website (https://www.firebirdsql.org/) herunterladen können. Um nur die Client-Dateien zu installieren, wählen Sie im Zuge der Installation die Option "**Minimum client install - no server, no tools**" aus.

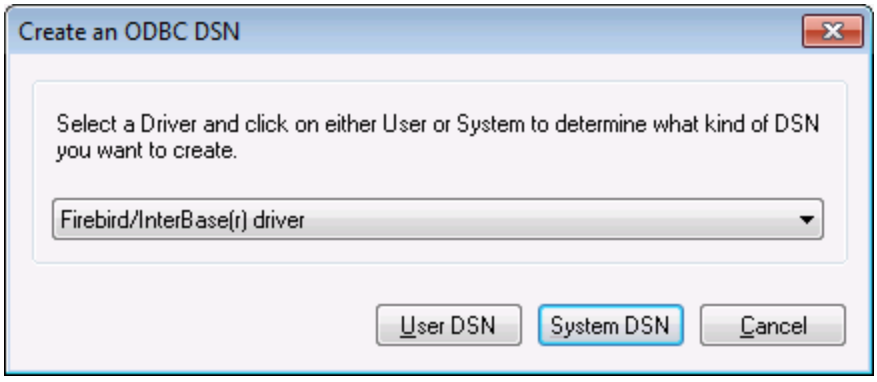
  Wichtig:

  - Die Plattform des Firebird ODBC-Treibers und Client (32-Bit oder 64-Bit) muss mit der von UModel übereinstimmen.
  - Die Version des Firebird Client muss mit der des Firebird Servers, zu dem Sie die Verbindung herstellen, übereinstimmen.

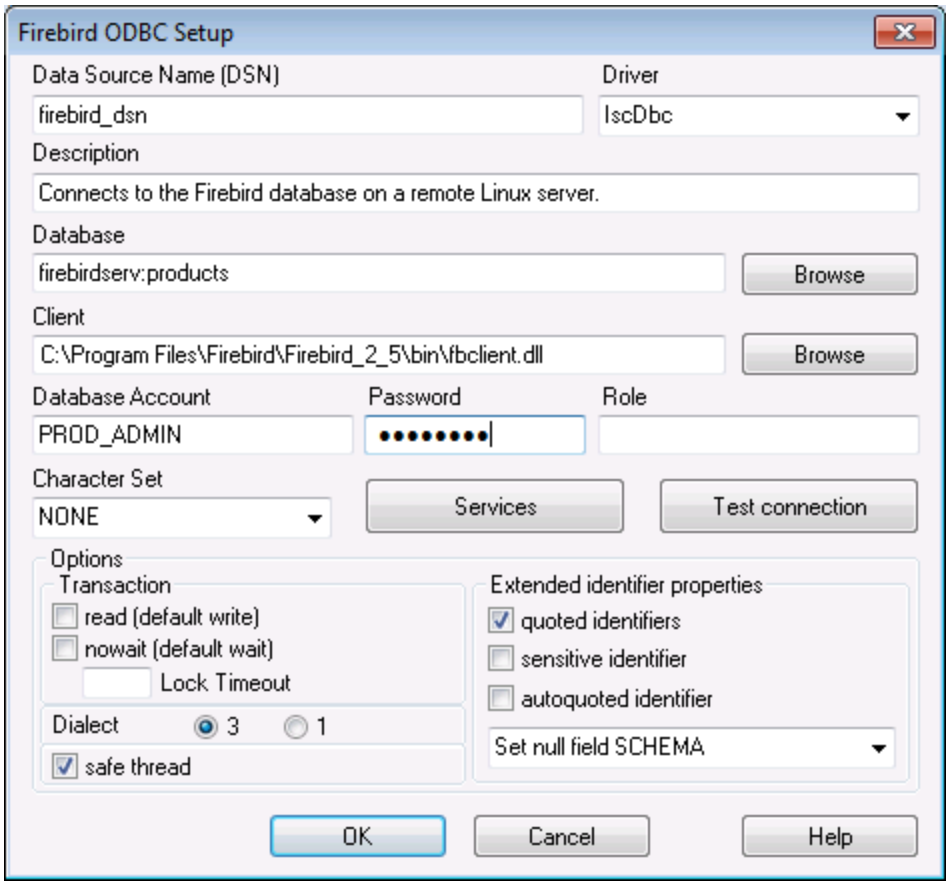
- Sie verfügen über die folgenden Datenbankverbindungsinformationen: Name oder IP-Adresse des Server Host, Datenbankpfad (oder Alias) auf dem Server, Benutzername und Passwort.

So stellen Sie über ODBC eine Verbindung zu Firebird her:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Wählen Sie Benutzer-DSN (oder **System-DSN**, wenn Sie Administratorrechte haben) und klicken Sie anschließend auf **Hinzufügen** .



4. Wählen Sie den Firebird-Treiber aus und klicken Sie anschließend, je nachdem, was Sie im vorherigen Schritt ausgewählt haben, auf **Benutzer-DSN** oder **System-DSN**. Wenn der Firebird-Treiber in der Liste nicht zur Verfügung steht, stellen Sie sicher, dass er auf Ihrem Betriebssystem installiert ist (siehe auch Anzeigen der verfügbaren ODBC-Treiber[598]).



5. Geben Sie die folgenden Datenbankverbindungsinformationen ein:

Data Source Name (DSN)	Geben Sie einen beschreibenden Namen für die zu erstellende Datenquelle ein.
Database	Geben Sie den Namen oder die IP-Adresse des Server Host, gefolgt von einem Doppelpunkt, gefolgt vom Datenbank-Alias (oder Pfad) ein. In diesem Beispiel lautet der Host-Name `firebirdserv` und der Datenbank-Alias `products`. Geben Sie daher den folgenden String ein: `firebirdserv:products` Wenn Sie einen Datenbank-Alias verwenden, wird davon ausgegangen, dass der Datenbankadministrator den Alias *products* auf dem Server so konfiguriert hat, dass er auf die tatsächliche Firebird (.fdb) Datenbankdatei verweist (nähere Informationen dazu siehe Firebird-Dokumentation). Sie können anstelle des Host-Namens auch die Server IP-Adresse und anstelle eines Pfads einen Alias verwenden; daher ist jeder der folgenden Beispiel Connection-Strings gültig:

 © 2016-2022 Altova GmbH

	`firebirdserver:/var/Firebird/databases/butterflies.fdb`
	`127.0.0.1:D:\Misc\Lenders.fdb`
	
	Wenn sich die Datenbank auf dem lokalen Windows-Rechner befindet, klicken Sie auf **Durchsuchen** und wählen Sie die Firebird (.fdb) Datenbank direkt aus.

Client	Geben Sie den Pfad zur Datei **fbclient.dll** ein. Standardmäßig ist dies das Unterverzeichnis `bin` des Firebird-Installationsverzeichnisses.
Database Account	Geben Sie den vom Datenbankadministrator bereitgestellten Datenbank-Benutzernamen ein (in diesem Beispiel `PROD_ADMIN`).
Password	Geben Sie das vom Datenbankadministrator bereitgestellte Datenbank-Passwort ein

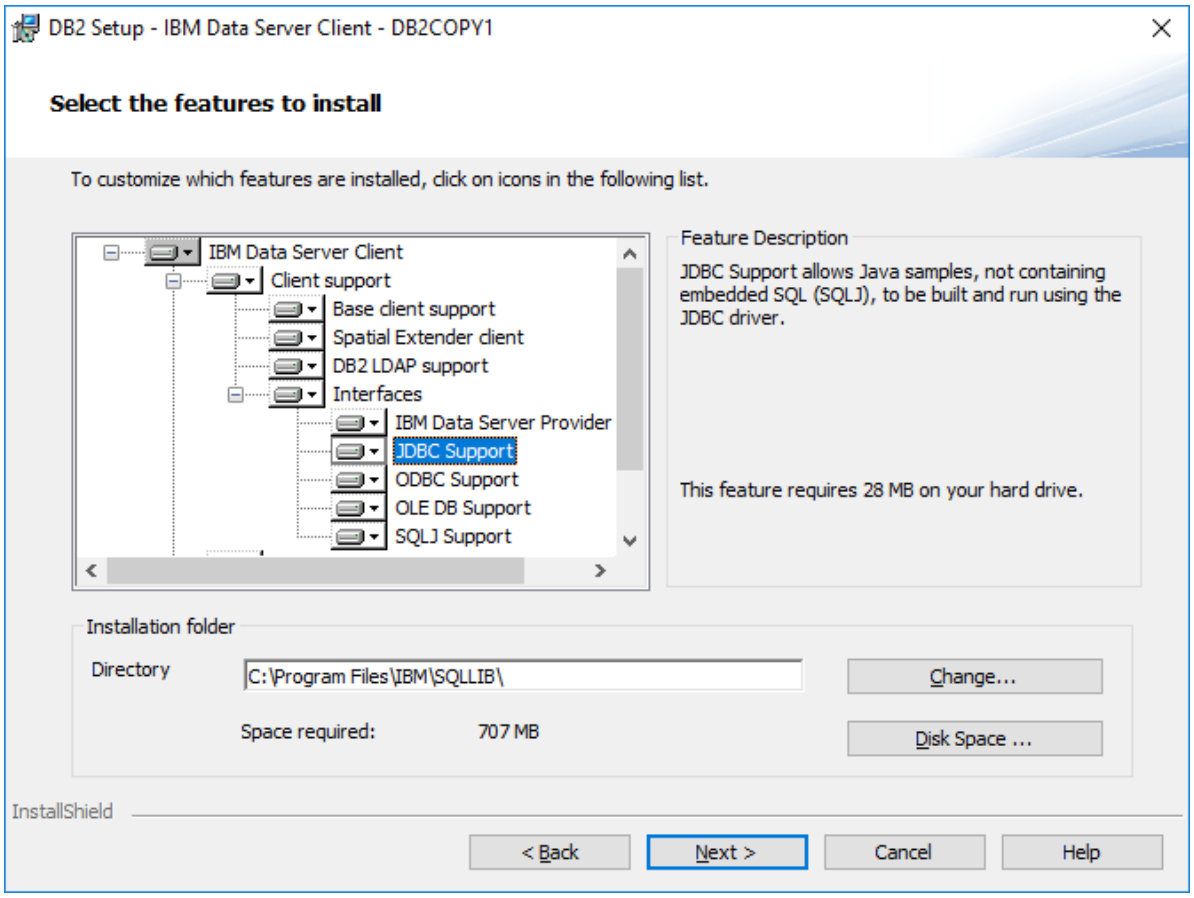
1. Klicken Sie auf **OK**.

10.2.9.3 IBM DB2 (JDBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einem **IBM DB2** Datenbankserver über JDBC.

Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt. In diesem Beispiel wird Oracle OpenJDK 11.0 64-Bit verwendet, folglich wird auch die 64-Bit-Version von UModel verwendet.
- Der Firebird JDBC-Treiber (eine oder mehrere .jar-Dateien, die die Verbindung zur Datenbank herstellen) muss auf Ihrem Betriebssystem verfügbar sein. In diesem Beispiel wird der JDBC-Treiber verwendet, der zur Verfügung steht, nachdem Sie die **IBM Data Server Client** Version 10.1 (64-Bit) installiert haben. Um die JDBC-Treiber zu installieren, wählen Sie eine **Standardinstallation** oder wählen Sie diese Option explizit im Installationsassistenten aus.



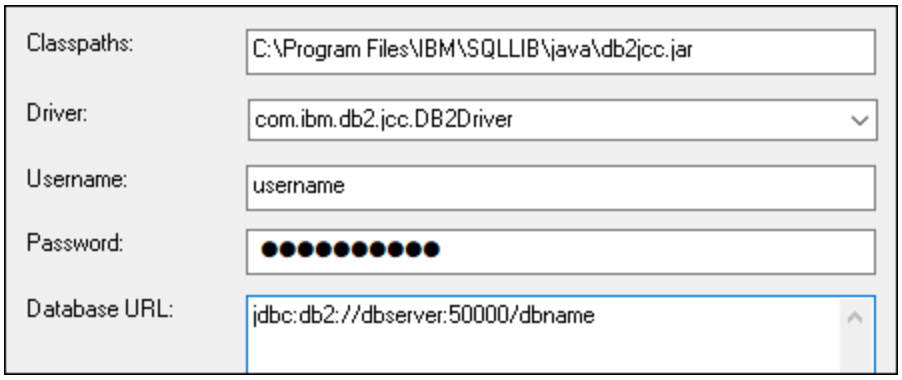
Wenn Sie den Standardinstallationspfad nicht geändert haben, befinden sich die erforderlichen .jar-Dateien nach der Installation im Verzeichnis **C:\Programme\IBM\SQLLIB\java**.

- Sie benötigen die folgenden Datenbankverbindungsinformationen: Host, Port, Datenbankname, Benutzername und Passwort.

So stellen Sie über JDBC eine Verbindung zu IBM DB2 her:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. In diesem Beispiel wird der Pfad **C:\Programme\IBM\SQLLIB\java\db2jcc.jar** referenziert. Je nach Datenbankversion müssen Sie eventuell den Treiber **db2jcc4.jar** referenzieren. Informationen zur Treiberkompatibilität finden Sie in der IBM-Dokumentation unter (http://www-01.ibm.com/support/docview.wss?uid=swg21363866). Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH[602]).
4. Wählen Sie im Feld "Treiber" **com.ibm.db2.jcc.DB2Driver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpath" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).

© 2016-2022 Altova GmbH



5. Geben Sie den Benutzernamen und das Passwort des Datenbankbenutzers in die entsprechenden Textfelder ein.
6. Geben Sie in das Textfeld **Datenbank-URL** den Connection String zum Datenbankserver ein. Dabei müssen Sie die Verbindungsinformationen durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:db2://hostName:port/databaseName
```

7. Klicken Sie auf **Verbinden**.

10.2.9.4 IBM DB2 (ODBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einer IBM DB2-Datenbank über ODBC.

Voraussetzungen:

- Auf Ihrem Betriebssystem muss IBM Data Server Client installiert und konfiguriert sein (in diesem Beispiel wird IBM Data Server Client 9.7 verwendet). Eine Installationsanleitung dazu finden Sie in der Dokumentation zu Ihrer IBM DB2 Software. Nachdem Sie IBM Data Server Client installiert haben, überprüfen Sie, ob die ODBC-Treiber auf Ihrem Rechner verfügbar sind (siehe Anzeigen der verfügbaren ODBC-Treiber 598).
- Erstellen Sie einen Datenbank-Alias. Es gibt mehrere Methoden, dies zu tun:
  - über den IBM DB2-Konfigurationsassistenten
  - über den IBM DB2-Befehlszeilenprozessor
  - über den ODBC-Datenquellenassistenten (die Anleitung dazu finden Sie weiter unten)
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Host, Datenbank, Port, Benutzername und Passwort.

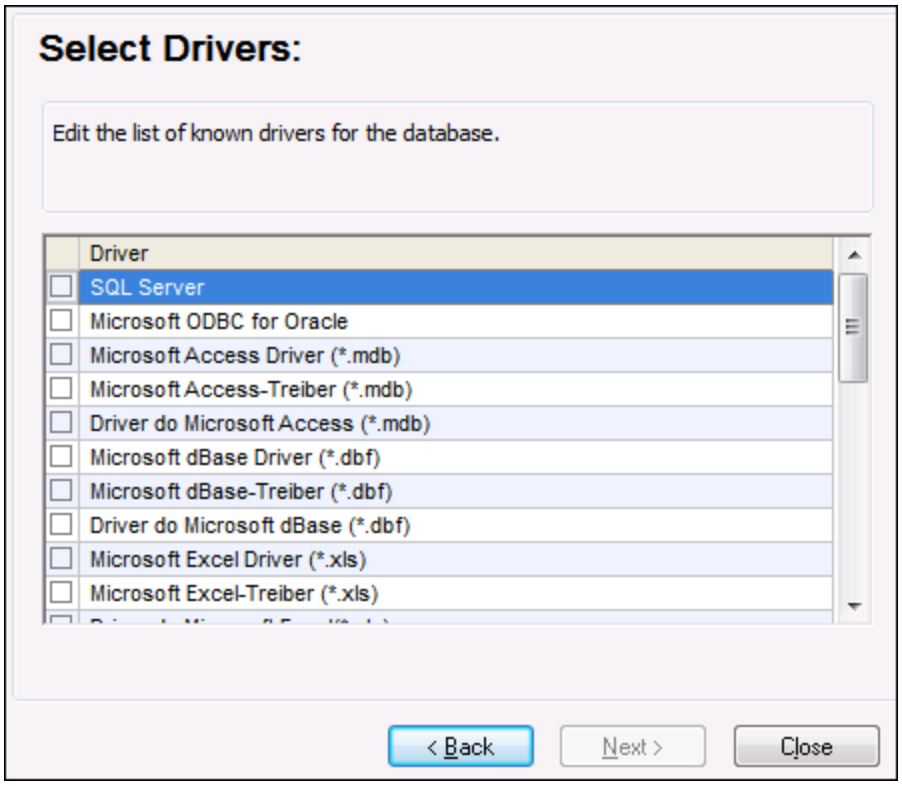
So stellen Sie eine Verbindung zu IBM DB2 her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578 und wählen Sie **IBM DB2 (ODBC/JDBC)** aus.
2. Klicken Sie auf **Weiter**.



3. Wählen Sie **ODBC** aus und klicken Sie auf **Weiter.** Wenn Sie aufgefordert werden, die Liste der bekannten Treiber für die Datenbank zu bearbeiten, wählen Sie die Datenbanktreiber für IBM DB2 aus (siehe Voraussetzungen[613]) und klicken Sie auf **Weiter**.

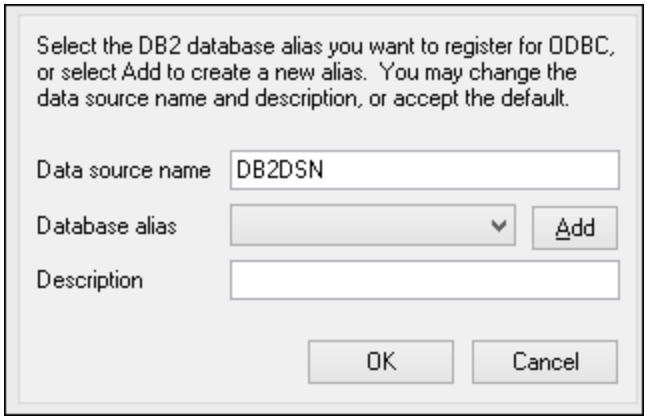
© 2016-2022 Altova GmbH



4. Wählen Sie den IBM DB2-Treiber aus der Liste aus und klicken Sie anschließend auf **Verbinden**. (Um die Liste der verfügbare Treiber zu bearbeiten, klicken Sie auf **Treiber bearbeiten** und aktivieren bzw. deaktivieren Sie anschließend die IBM DB2-Treiber, die Sie hinzufügen bzw. entfernen möchten.)

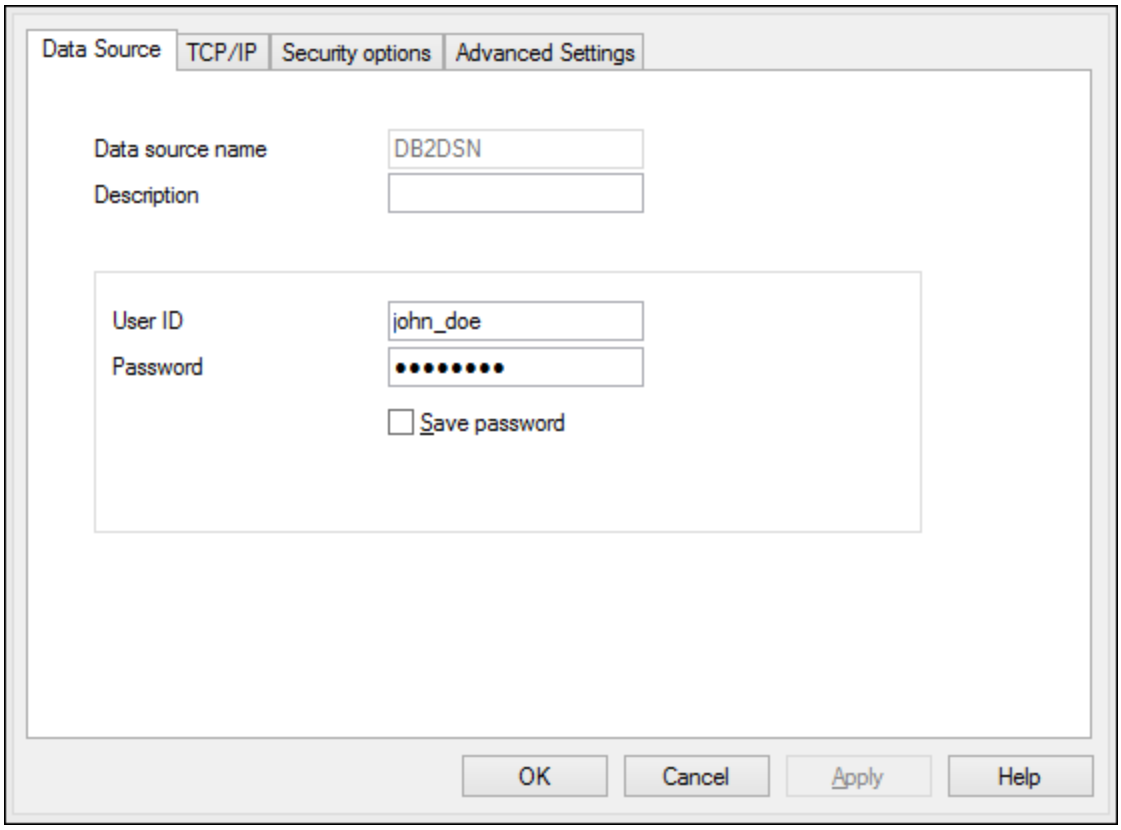


5. Geben Sie einen Datenquellennamen ein (in diesem Beispiel **DB2DSN**) und klicken Sie auf **Hinzufügen**.

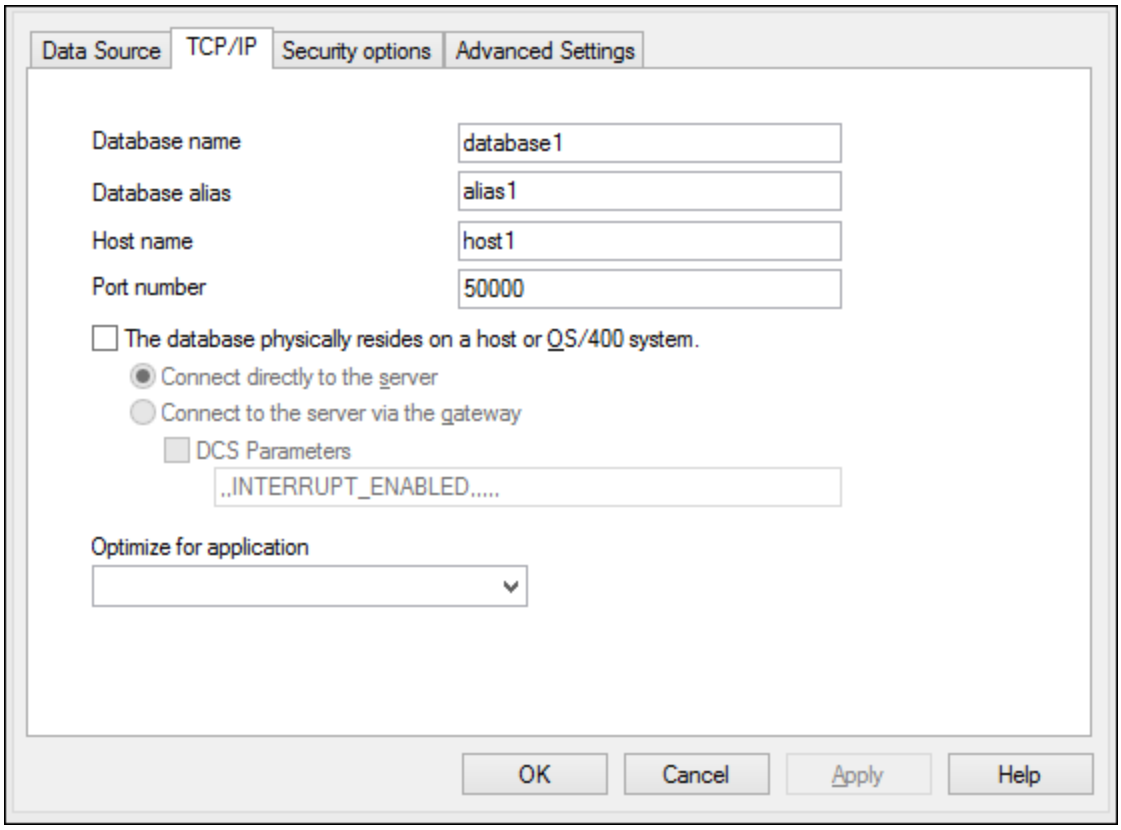


6. Geben Sie auf dem Register **Datenquelle** den Benutzernamen und das Passwort für die Datenbank ein.

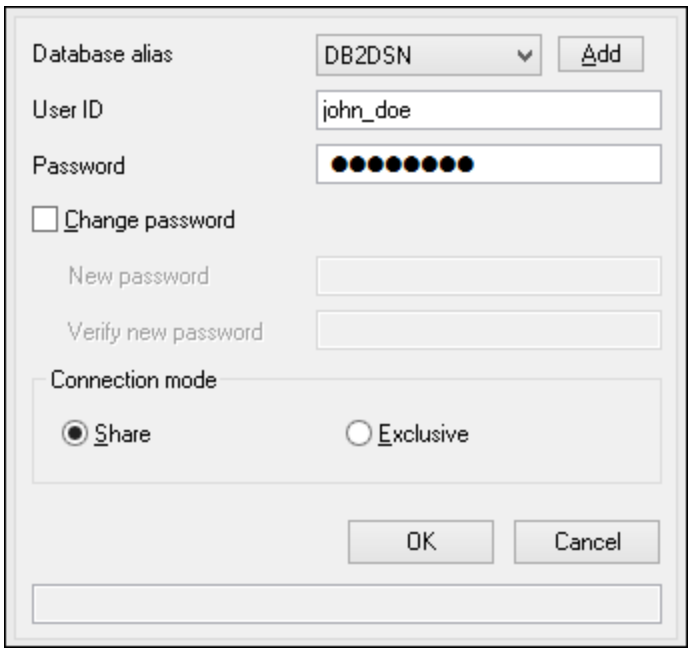
© 2016-2022 Altova GmbH



7. Geben Sie auf dem Register **TCP/IP** den Datenbanknamen, einen Namen für den Alias, den Host-Namen und die Port-Nummer ein und klicken Sie auf OK.



8. Geben Sie den Benutzernamen und das Passwort erneut ein und klicken Sie auf **OK**.



© 2016-2022 Altova GmbH

10.2.9.5 IBM DB2 für i (JDBC)

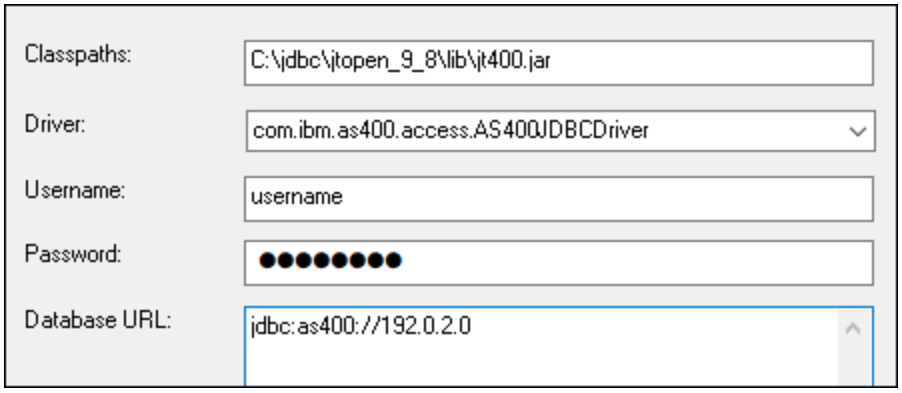
Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einem **IBM DB2 für i**-Datenbankserver über JDBC.

Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt. In diesem Beispiel wird Oracle OpenJDK 11.0 64-Bit verwendet, folglich wird auch die 64-Bit-Version von UModel verwendet.
- Der Firebird JDBC-Treiber (eine oder mehrere .jar-Dateien, die die Verbindung zur Datenbank herstellen) muss auf Ihrem Betriebssystem verfügbar sein. In diesem Beispiel wird die Open Source **Toolbox for Java/JTOpen** Version 9.8 (http://jt400.sourceforge.net/) verwendet. Nachdem Sie das Paket heruntergeladen und in ein lokales Verzeichnis entpackt haben, stehen die erforderlichen .jar-Dateien im Unterverzeichnis **lib** zur Verfügung.
- Sie benötigen die folgenden Datenbankverbindungsinformationen: Host, Benutzername und Passwort.

So stellen Sie über JDBC eine Verbindung zu IBM DB2 für i her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. In diesem Beispiel wird der Pfad **C:\jdbc\jtopen_9_8\jt400.jar** referenziert. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH 602).
4. Wählen Sie im Feld "Treiber" **com.ibm.as400.access.AS400JDBCDriver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpath" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).



5. Geben Sie den Benutzernamen und das Passwort des Datenbankbenutzers in die entsprechenden Textfelder ein.
6. Geben Sie in das Textfeld **Datenbank-URL** den JDBC-Connection String zum Datenbankserver ein. Dabei müssen Sie `host` durch den Hostnamen oder die IP-Adresse Ihres Datenbankservers ersetzen.

```
jdbc:as400://host
```

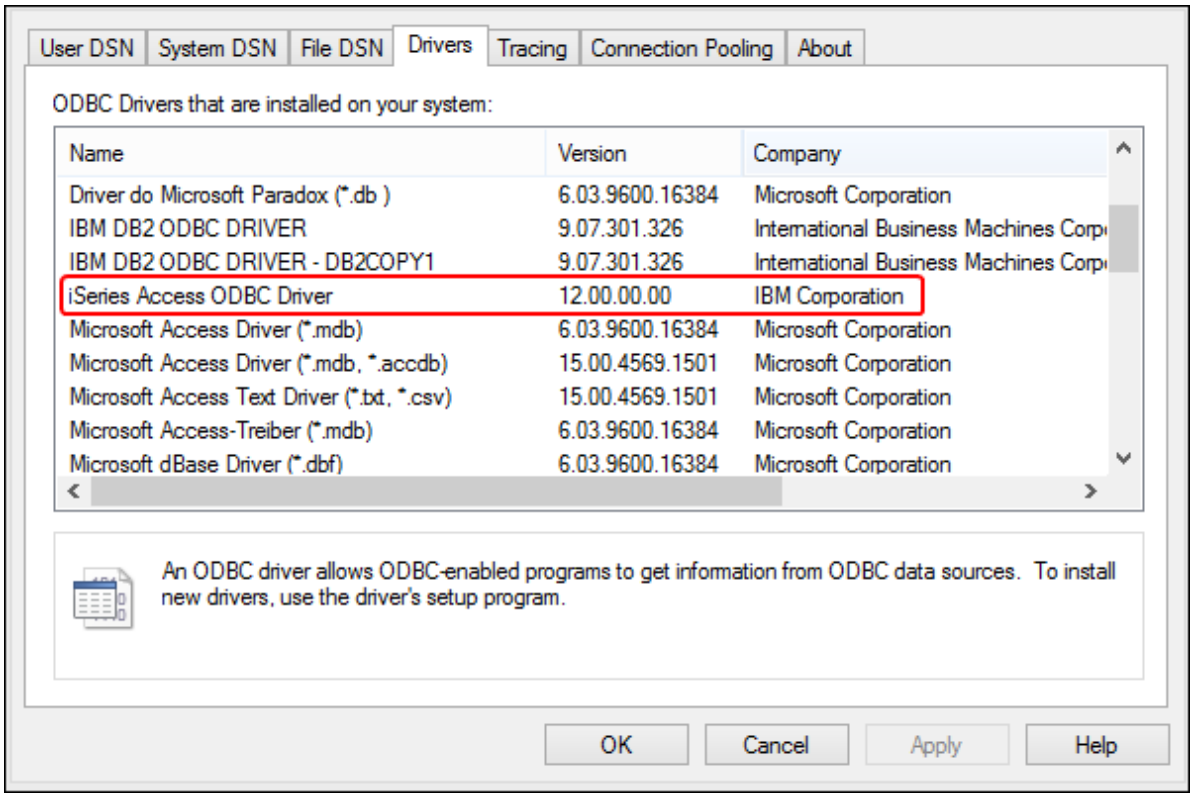
7. Klicken Sie auf **Verbinden**.

10.2.9.6 IBM DB2 für i (ODBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einer *IBM DB2 für i*-Datenbank über ODBC.

Voraussetzungen:

- Auf Ihrem Betriebssystem muss *IBM System i Access für Windows* installiert sein (in diesem Beispiel wird *IBM System i Access für Windows V6R1M0 verwendet*). Eine Installationsanleitung finden Sie in der Dokumentation zu Ihrer *IBM DB2 für i*-Software. Überprüfen Sie nach der Installation, ob der ODBC-Treiber auf Ihrem Rechner verfügbar ist (siehe Anzeigen der verfügbaren OBBC-Treiber 598).



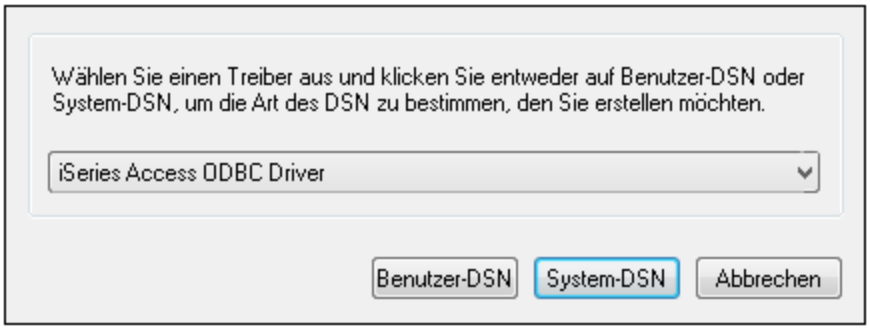
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: die IP-Adresse des Datenbankservers, den Datenbank-Benutzernamen und das Passwort.
- Führen Sie den *System i Navigator* aus und befolgen Sie die Anweisungen des Assistenten, um eine neue Verbindung zu erstellen. Wenn Sie nach einem System gefragt werden, geben Sie die IP-

© 2016-2022 Altova GmbH

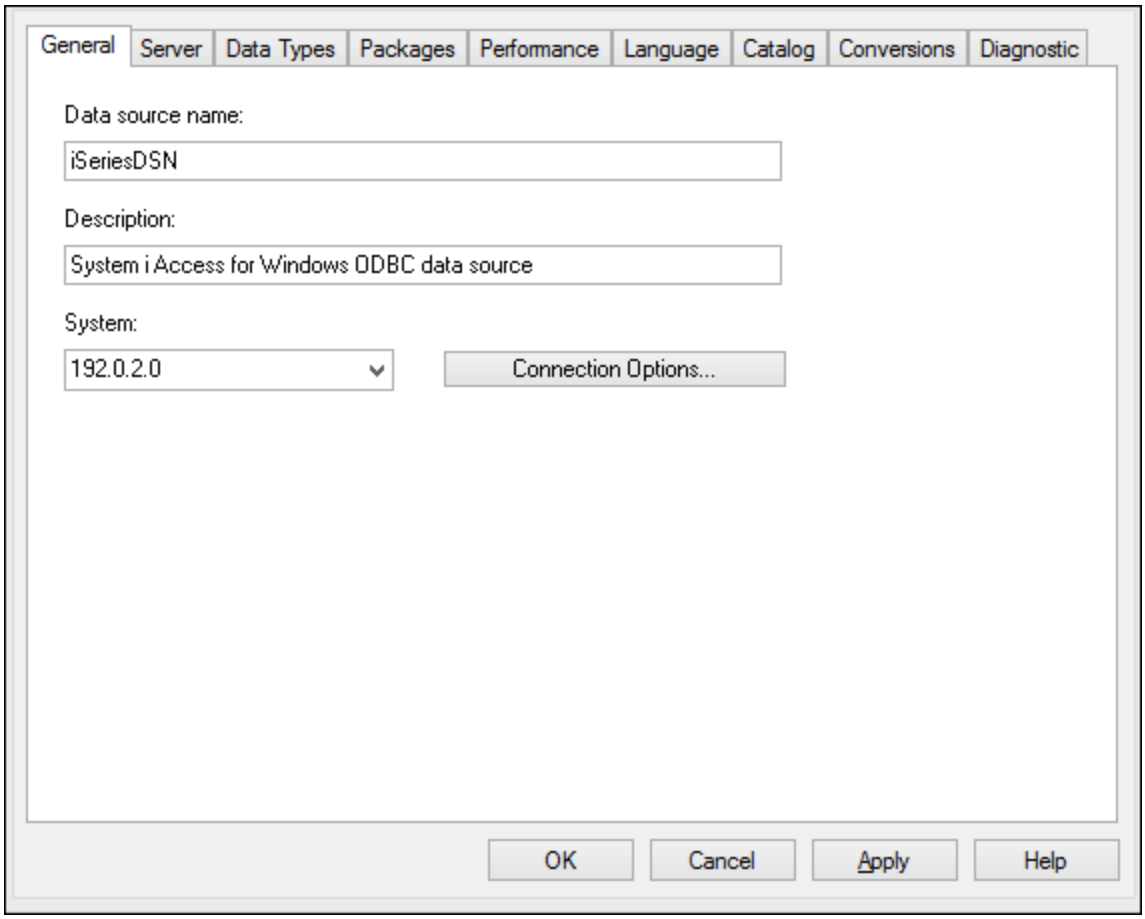
Adresse des Datenbankservers ein. Nachdem Sie die Verbindung hergestellt haben, sollten Sie diese überprüfen (Klicken Sie auf den Verbindung und wählen Sie **Datei** > **Diagnose** > **Verbindung überprüfen**). Wenden Sie sich an den Datenbankserver-Administrator, wenn Sie Verbindungsfehler erhalten.

So stellen Sie eine Verbindung zu IBM DB2 für i ein:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Klicken Sie auf **Benutzer-DSN** (Klicken Sie alternativ dazu auf **System-DSN** oder **Datei-DSN** - die darauf folgende Anleitung ist ähnlich).
4. Klicken Sie auf **Hinzufügen** .
5. Wählen Sie aus der Liste **iSeries Access ODBC Driver** aus und klicken Sie auf **Benutzer-DSN** (bzw. gegebenenfalls auf **System- DSN**).

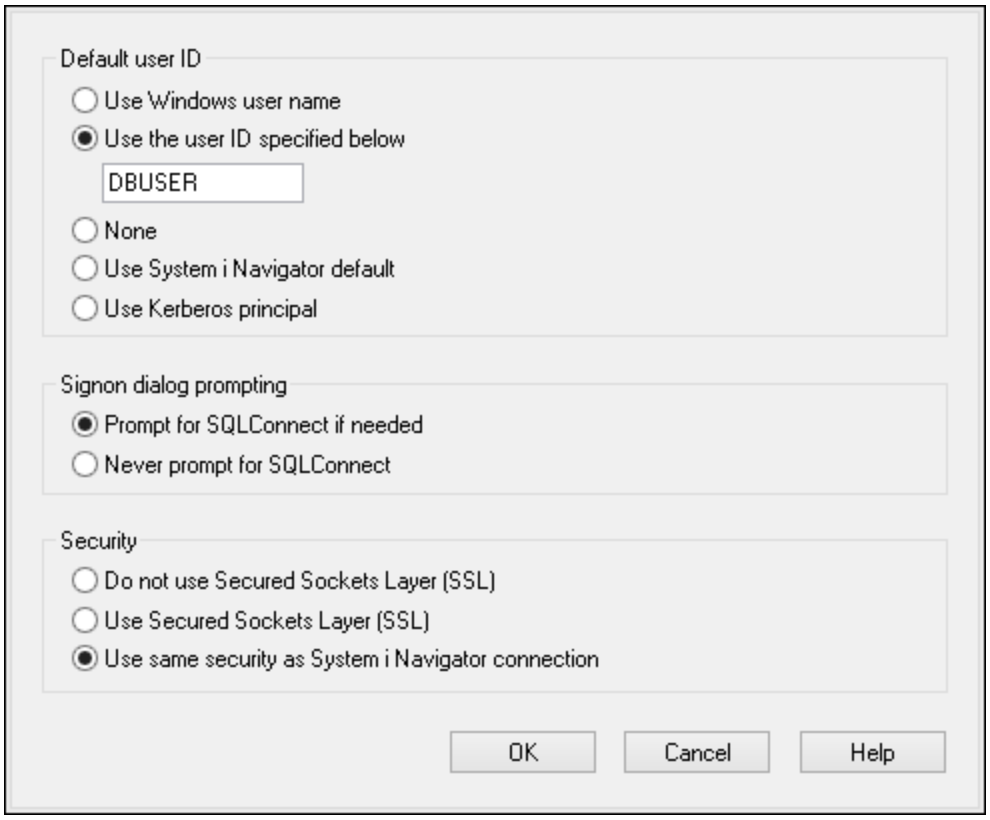


6. Geben Sie den Datenquellennamen ein und wählen Sie die Verbindung aus der System-Auswahlliste aus. In diesem Beispiel lautet der Datenquellenname **iSeriesDSN** und das System ist **192.0.2.0**.



7. Klicken Sie auf **Verbindungsoptionen** und wählen Sie **unten angeführte Benutzer-ID verwenden** und geben Sie den Namen des Datenbankbenutzers ein (in diesem Beispiel **DBUSER**).

© 2016-2022 Altova GmbH



8. Klicken Sie auf **OK**. Die neue Datenquelle steht nun in der Liste der DSNs zur Verfügung.
9. Klicken Sie auf **Verbinden**.
10. Geben Sie den Benutzernamen und das Passwort für die Datenbank ein, wenn Sie dazu aufgefordert werden und klicken Sie auf **OK**.

10.2.9.7 IBM Informix (JDBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einer IBM Informix-Datenbank über JDBC.

Voraussetzungen:

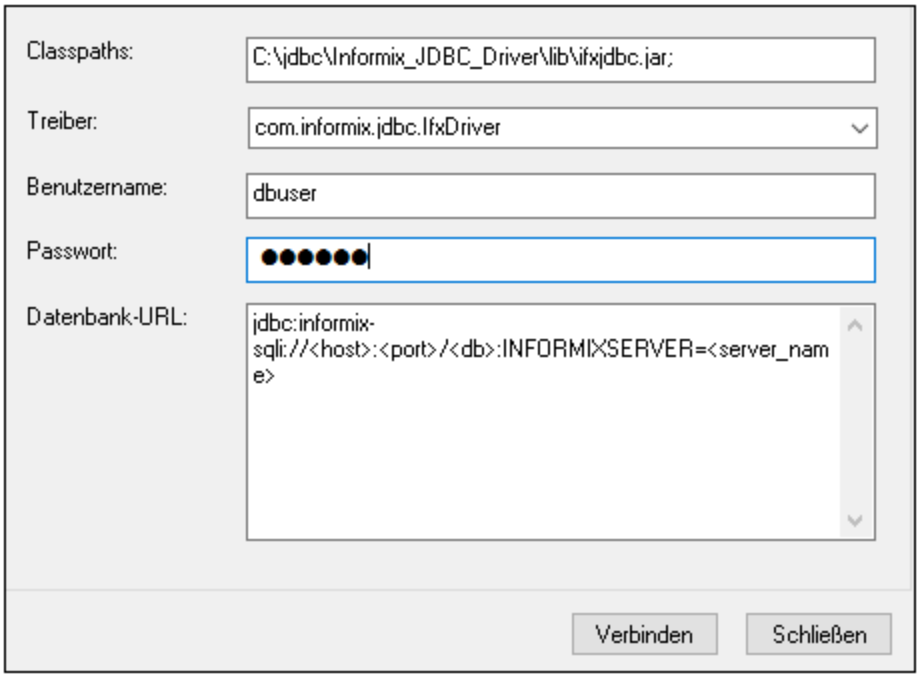
- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- Der JDBC-Treiber (eine oder mehrere .jar-Dateien, die die Verbindung zur Datenbank herstellen) muss auf Ihrem Betriebssystem installiert sein. In diesem Beispiel wird die IBM Informix JDBC-Treiberversion 3.70 verwendet. Die Installationsanleitung zum Treiber finden Sie in der dazugehörigen Dokumentation bzw. im "IBM Informix JDBC Driver Programmer's Guide").

- Sie haben die folgenden Datenbankinformationen zur Verfügung: Host, Name des Informix-Servers, Datenbank, Port. Benutzername und Passwort.

So stellen Sie über JDBC eine Verbindung zu IBM Informix her:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. Falls nötig, können Sie auch eine durch Semikola getrennte Liste von .jar-Dateipfaden eingeben. Die benötigte .jar-Datei in diesem Beispiel befindet sich unter dem folgenden Pfad: **C:\Informix_JDBC_Driver\lib\ifxjdbc.jar**. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH[602]).

4. Wählen Sie im Feld "Treiber" **com.informix.jdbc.IfxDriver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpaths" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).



5. Geben Sie den Benutzernamen und das Passwort für die Datenbank in die entsprechenden Textfelder ein.
6. Geben Sie den Connection String zum Datenbankserver in das Textfeld "Datenbank-URL" ein, indem Sie die hervorgehobenen Werte durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:informix-sqli://hostName:port/databaseName:INFORMIXSERVER=myserver;
```

7. Klicken Sie auf **Verbinden.**

© 2016-2022 Altova GmbH

10.2.9.8 MariaDB (ODBC)

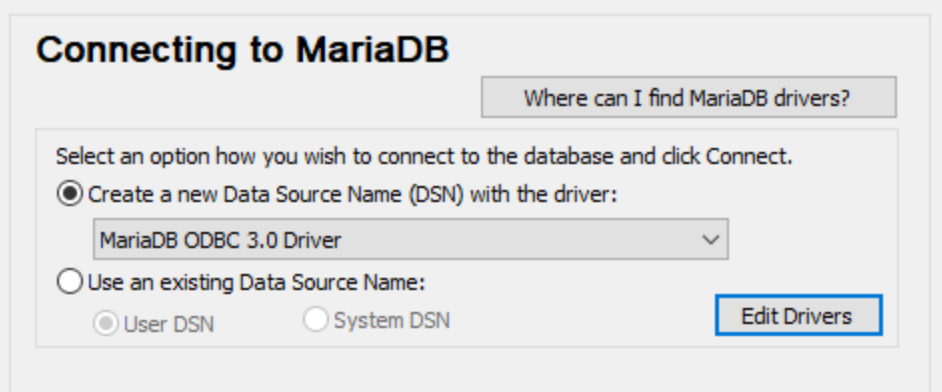
In diesem Beispiel wird gezeigt, wie Sie über ODBC eine Verbindung zu einem MariaDB-Datenbankserver herstellen.

Voraussetzungen:

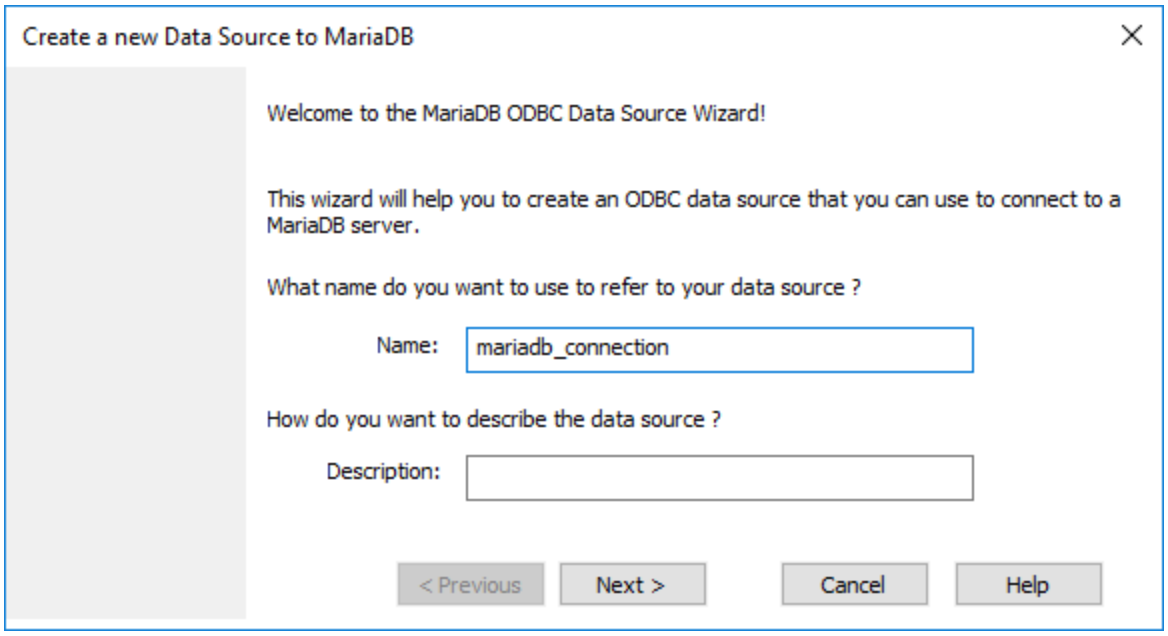
- Der MariaDB Connector/ODBC (https://downloads.mariadb.org/connector-odbc/) muss installiert sein.
- Sie haben die folgenden Datenbankinformationen zur Verfügung: Host, Datenbank, Port. Benutzername und Passwort.

So stellen Sie über ODBC eine Verbindung zu MariaDB her:

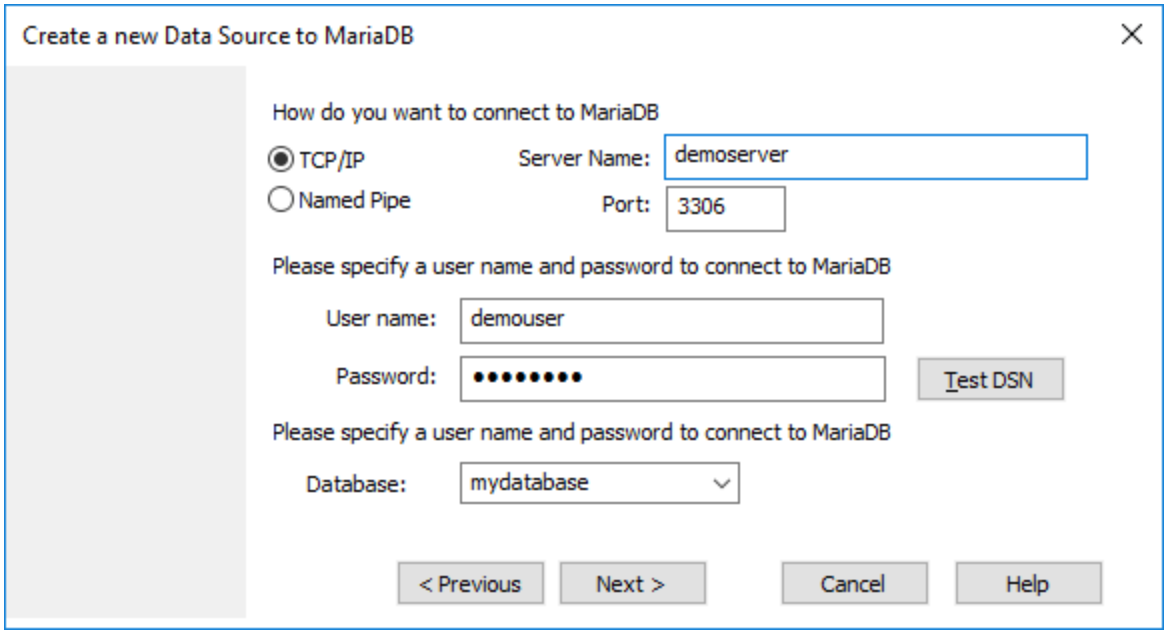
1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Wählen Sie **MariaDB (ODBC)** aus und klicken Sie auf **Weiter**.



3. Wählen Sie **Erstelle neuen Data Source Name (DSN) mit dem Treiber** und wählen Sie **MariaDB ODBC 3.0 Driver** aus. Wenn in der Liste kein MySQL-Treiber verfügbar ist, klicken Sie auf **Treiber bearbeiten** und wählen Sie einen beliebigen verfügbaren MariaDB-Treiber aus (die Liste enthält alle auf Ihrem Betriebssystem installierten ODBC-Treiber).
4. Klicken Sie auf **Verbinden**.

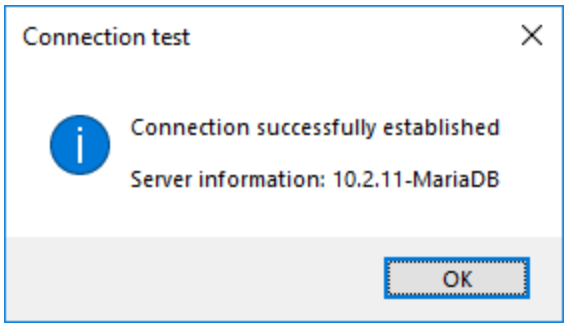


5. Geben Sie einen Namen und optional eine Beschreibung für diese ODBC-Datenquelle ein.



6. Füllen Sie die Anmeldeinformationen für die Datenbankverbindung aus (TCP/IP Server, Benutzer, Passwort), wählen Sie eine Datenbank aus und klicken Sie auf **DSN testen**. Wenn die Verbindung erfolgreich hergestellt werden konnte, wird eine Meldung angezeigt:

© 2016-2022 Altova GmbH



7. Klicken Sie auf **Weiter** und schließen Sie den Vorgang ab. Von Fall zu Fall werden eventuell auch andere Parameter benötigt (z.B. SSL-Zertifikate, wenn Sie eine sichere Verbindung zu MariaDB herstellen möchten).

Anmerkung: Wenn es sich um einen entfernten Datenbankserver handelt, muss er vom Server-Administrator so konfiguriert werden, dass er entfernte Verbindungen von der IP-Adresse Ihres Rechners aus zulässt.

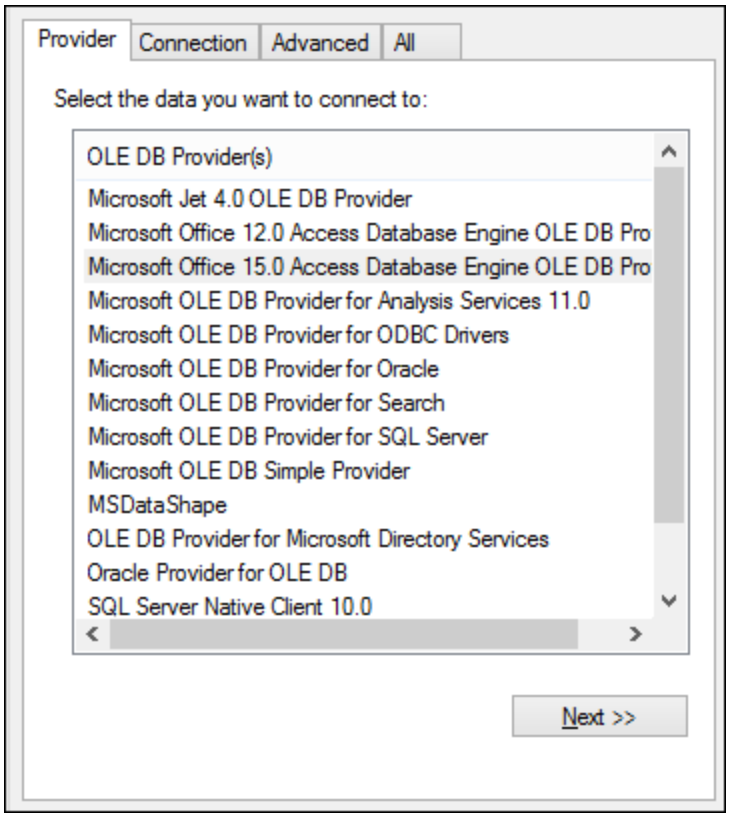
10.2.9.9 Microsoft Access (ADO)

Eine einfache Methode, eine Verbindung zu einer Microsoft Access-Datenbank herzustellen, ist, den Anweisungen des Assistenten zu folgen und zur Datenbankdatei zu navigieren, wie unter Herstellen einer Verbindung zu einer vorhandenen Microsoft Access-Datenbank 586 beschrieben. Alternativ dazu können Sie explizit eine ADO-Verbindung definieren, wie im Folgenden gezeigt. Diese Methode empfiehlt sich, wenn Ihre Datenbank durch ein Passwort geschützt ist.

Sie können zwar auch über ODBC eine Verbindung zu Microsoft Access herstellen, doch sollte diese Methode vermieden werden, da sich dadurch einige Einschränkungen ergeben.

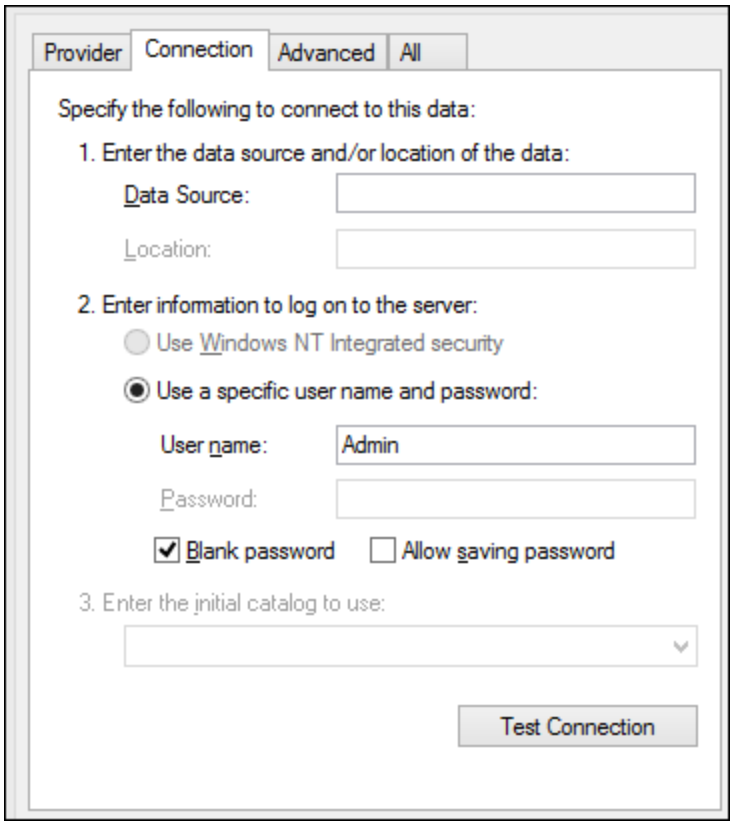
Herstellen einer Verbindung zu einer durch ein Passwort geschützten Microsoft Access-Datenbank:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ADO-Verbindungen**.
3. Klicken Sie auf **Erzeugen**.

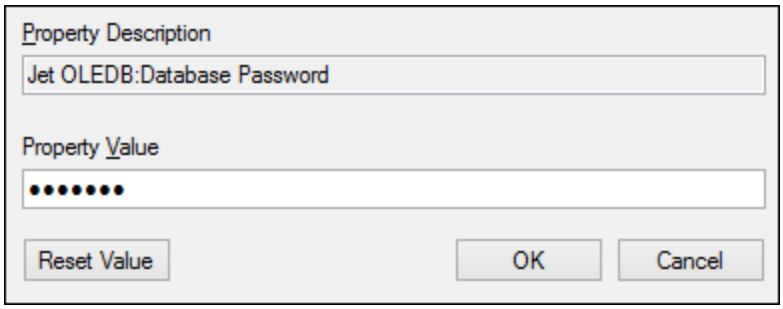


4. Wählen Sie den **Microsoft Office 15.0 Access Database Engine OLE DB Provider** aus und klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH



5. Geben Sie in das Feld "Datenquelle" den Pfad zur Microsoft Access-Datei im UNC-Format ein, z.B. \ **\myserver\\mynetworkshare\Reports\Revenue.accdb**, wobei **myserver** der Name des Servers und **mynetworkshare** der Name des gemeinsamen Netzwerklaufwerks ist.
6. Doppelklicken Sie auf dem Register **Alle** auf die Eigenschaft **Jet OLEDB:Database Password** und geben Sie das Datenbank-Passwort als Eigenschaftswert ein.



Anmerkung: Wenn die Verbindung immer noch nicht hergestellt werden kann, suchen Sie die Arbeitsgruppen-Informationsdatei (**System.MDW**) für Ihr Benutzerprofil und setzen Sie den Wert der -Eigenschaft **Jet OLEDB: System** auf den Pfad der Datei **System.MDW**.

10.2.9.10 Microsoft SQL Server (ADO)

In diesem Beispiel wird gezeigt, wie Sie über ADO eine Verbindung zu einer SQL Server-Datenbank herstellen. Diese Anleitung gilt für die Verwendung des empfohlenen **Microsoft OLE DB-Treibers für SQL Server**

(MSOLEDBSQL), der von https://docs.microsoft.com/en-us/sql/connect/oledb/download-oledb-driver-for-sql-server?view=sql-server-ver15 heruntergeladen werden kann.

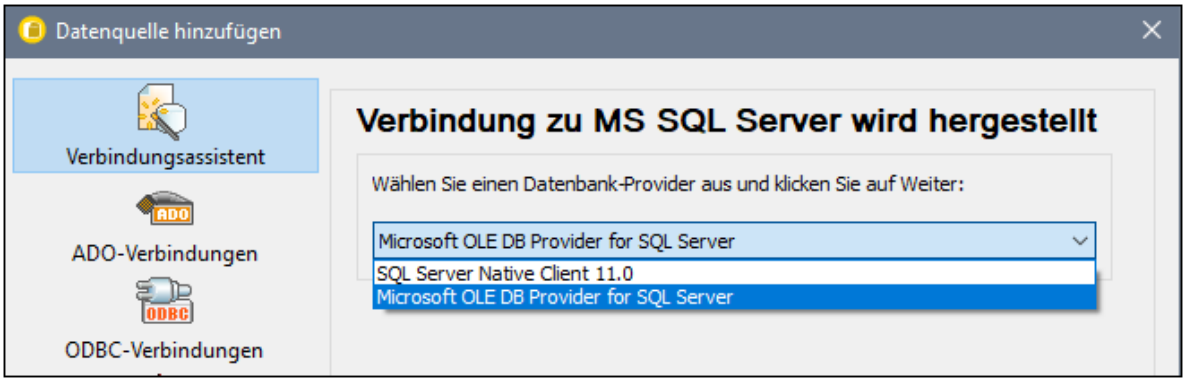
> Bevor Sie die Schritte aus dieser Anleitung durchführen, stellen Sie sicher, dass Sie den oben genannten Anbieter heruntergeladen und auf Ihrem Rechner installiert haben. Der ADO-Anbieter muss mit der Plattform von UModel (32-Bit oder 64-Bit) übereinstimmen.

Wenn Sie andere ADO-Anbieter wie **SQL Server Native Client (SQLNCLI)** oder **Microsoft OLE DB-Anbieter für SQL Server (SQLOLEDB)** verwenden möchten, ist die Vorgangsweise ähnlich, doch sind diese Anbieter veraltet und werden daher nicht empfohlen. Damit die Verbindung zu einem veralteten Anbieter hergestellt werden kann, müssen Sie außerdem eventuell zusätzliche Verbindungseigenschaften konfigurieren, wie unter Einrichten der SQL Server-Datenverknüpfungseigenschaften 586 beschrieben.

> Es ist bekannt, dass es beim **Microsoft OLE DB-Anbieter für SQL Server (SQLOLEDB)** zu Problemen mit der Parameterbindung komplexer Abfragen wie Common Table Expressions (CTE) und verschachtelten SELECT-Anweisungen kommt.

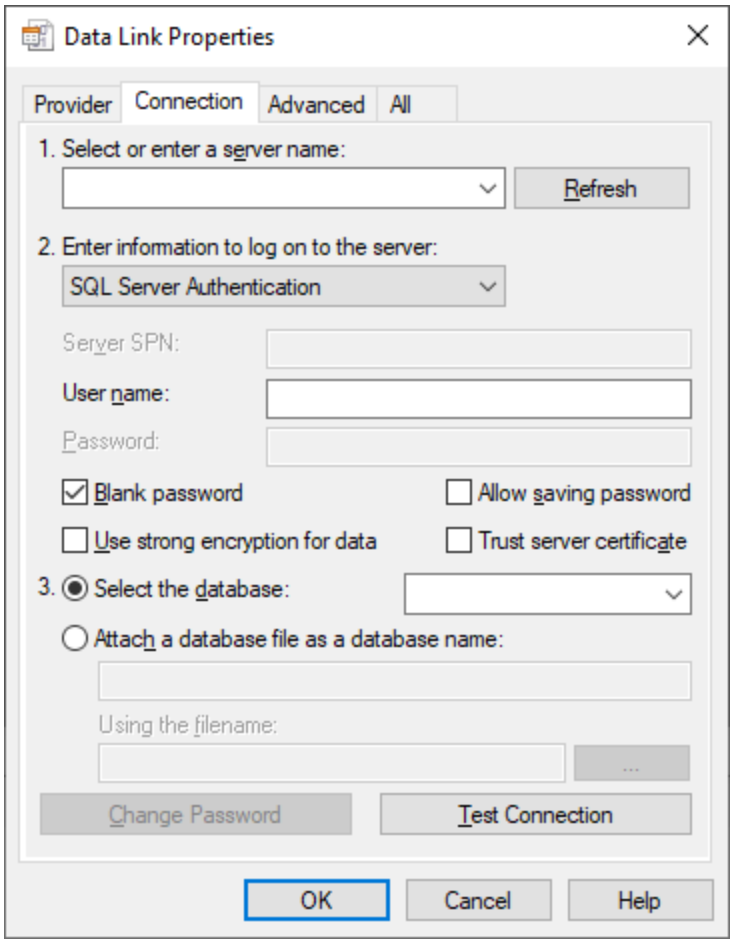
So stellen Sie eine Verbindung zu SQL Server her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Wählen Sie **Microsoft SQL Server (ADO)** und klicken Sie auf **Weiter**. Daraufhin wird die Liste der verfügbaren ADO-Anbieter angezeigt. In diesem Beispiel wird der **Microsoft OLE DB-Treiber für SQL Server** verwendet. Falls er in der Liste nicht enthalten ist, überprüfen Sie, ob er, wie oben erwähnt, auf Ihrem Rechner installiert ist.

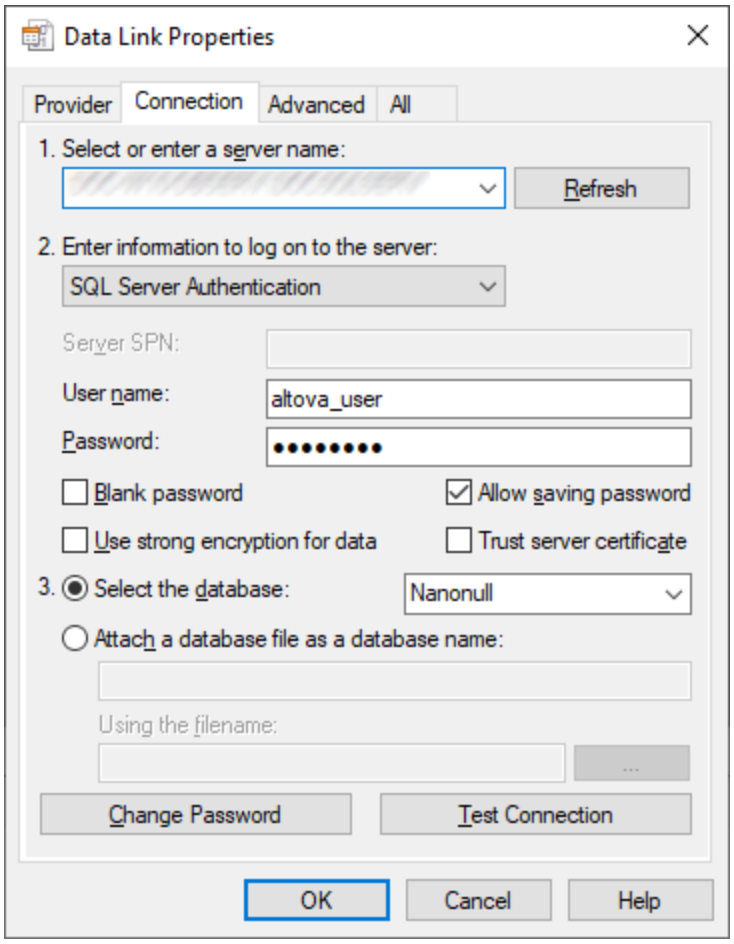


3. Klicken Sie auf **Weiter**. Daraufhin wird das Dialogfeld "Datenverknüpfungseigenschaften" angezeigt.

 © 2016-2022 Altova GmbH



4. Wählen Sie den Namen des Datenbankservers aus oder geben Sie ihn ein (z.B. **SQLSERV01**). Wenn Sie eine Verbindung zu einer benannten SQL-Server-Instanz herstellen, sieht der Servername folgendermaßen aus: **SQLSERV01\INSTANZ**.
5. Wenn der Datenbankserver so konfiguriert ist, dass er Verbindungen von bei der Windows Domain angemeldeten Benutzern gestattet, wählen Sie **Windows-Authentifizierung**. Wählen Sie andernfalls **SQL Server-Authentifizierung**, deaktivieren Sie das Kontrollkästchen **Leeres Kennwort** und geben Sie die Anmeldeinformationen für die Datenbank in die entsprechenden Felder ein.
6. Aktivieren Sie das Kontrollkästchen **Passwort speichern zulassen** und wählen Sie die gewünschte Datenbank aus (in diesem Beispiel "Nanonull").



7. Um die Verbindung zu diesem Zeitpunkt zu überprüfen, klicken Sie auf **Verbindung testen**. Dieser Schritt ist optional, wird aber empfohlen.
8. Klicken Sie auf **OK.**

10.2.9.11 Microsoft SQL Server (ODBC)

In diesem Beispiel wird gezeigt, wie Sie über ODBC eine Verbindung zu einer SQL Server-Datenbank herstellen.

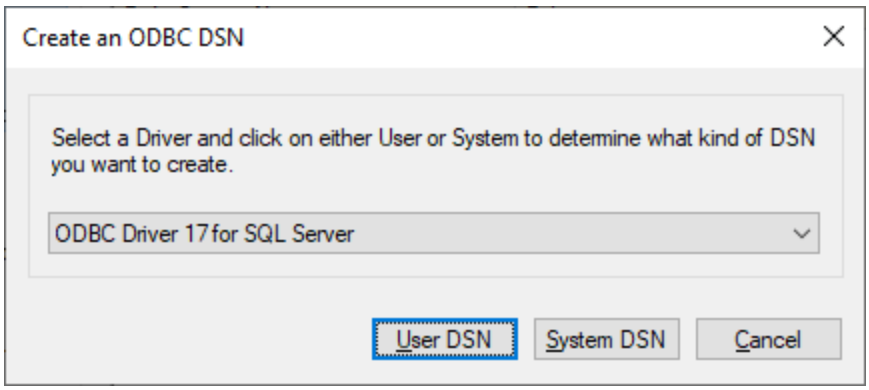
Voraussetzungen:

- Laden Sie den **Microsoft ODBC-Treiber für SQL Server** von der Microsoft-Website herunter und installieren Sie ihn, siehe https://docs.microsoft.com/en-us/SQL/connect/odbc/download-odbc-driver-for-sql-server. In diesem Beispiel wird für die Verbindung mit einer **SQL Server 2016**-Datenbank der **Microsoft ODBC-Treiber 17 für SQL Server** verwendet. Je nach gewünschter SQL-Server-Version müssen Sie eventuell eine andere ODBC-Treiberversion herunterladen. Informationen über von Ihrer SQL Server-Datenbank unterstützte ODBC-Treiberversionen schlagen Sie bitte unter den Systemanforderungen des Treibers nach.

© 2016-2022 Altova GmbH

So stellen Sie über ODBC eine Verbindung zu SQL Server her:

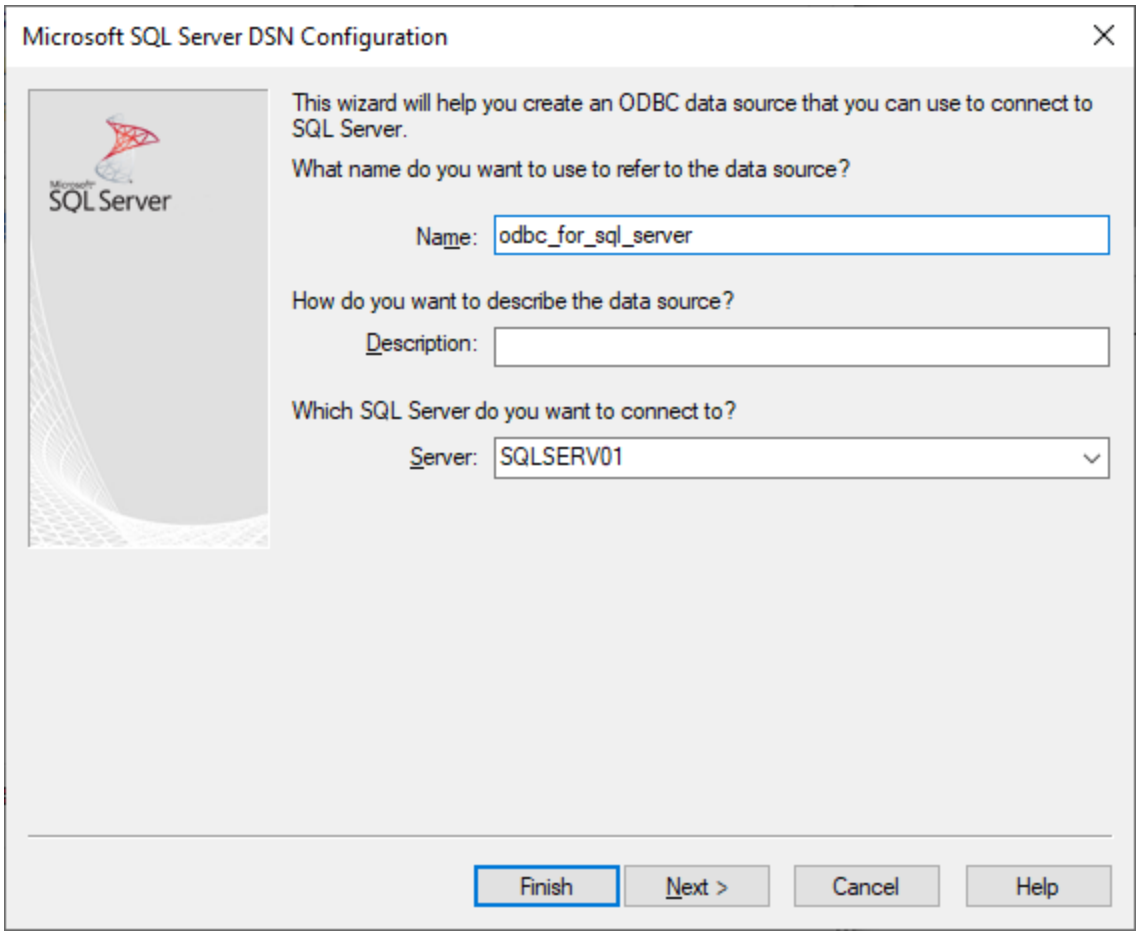
1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Wählen Sie **Benutzer-DSN** (oder **System-DSN**, wenn Sie Administratorrechte haben) und klicken Sie auf **Hinzufügen** .
4. Wählen Sie den Treiber aus der Liste aus. Beachten Sie, dass der Treiber erst nach Installation auf der Liste angezeigt wird.



5. Wählen Sie **Benutzer-DSN** (oder **System-DSN**, wenn Sie einen System-DSN erstellen).

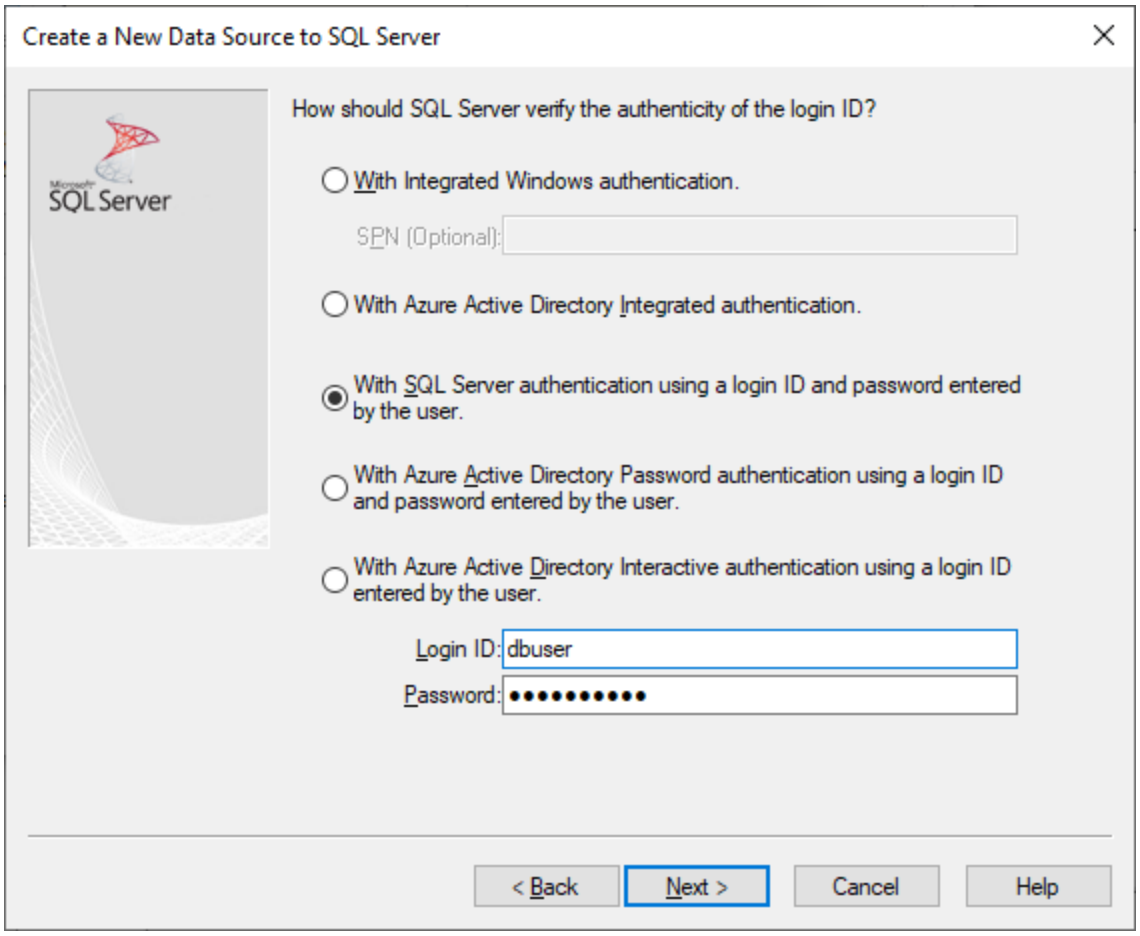
   Um einen **System-DSN** zu erstellen, müssen Sie UModel als Administrator ausführen. Um daher einen **System-DSN** zu erstellen, brechen Sie den Assistenten ab, stellen Sie sicher, dass Sie UModel als Administrator ausführen und führen Sie die obigen Schritte erneut durch.

6. Geben Sie einen Namen und optional eine Beschreibung für diese Verbindung ein und wählen Sie anschließend aus der Liste den gewünschten SQL Server aus (in diesem Beispiel **SQLSERV01**).

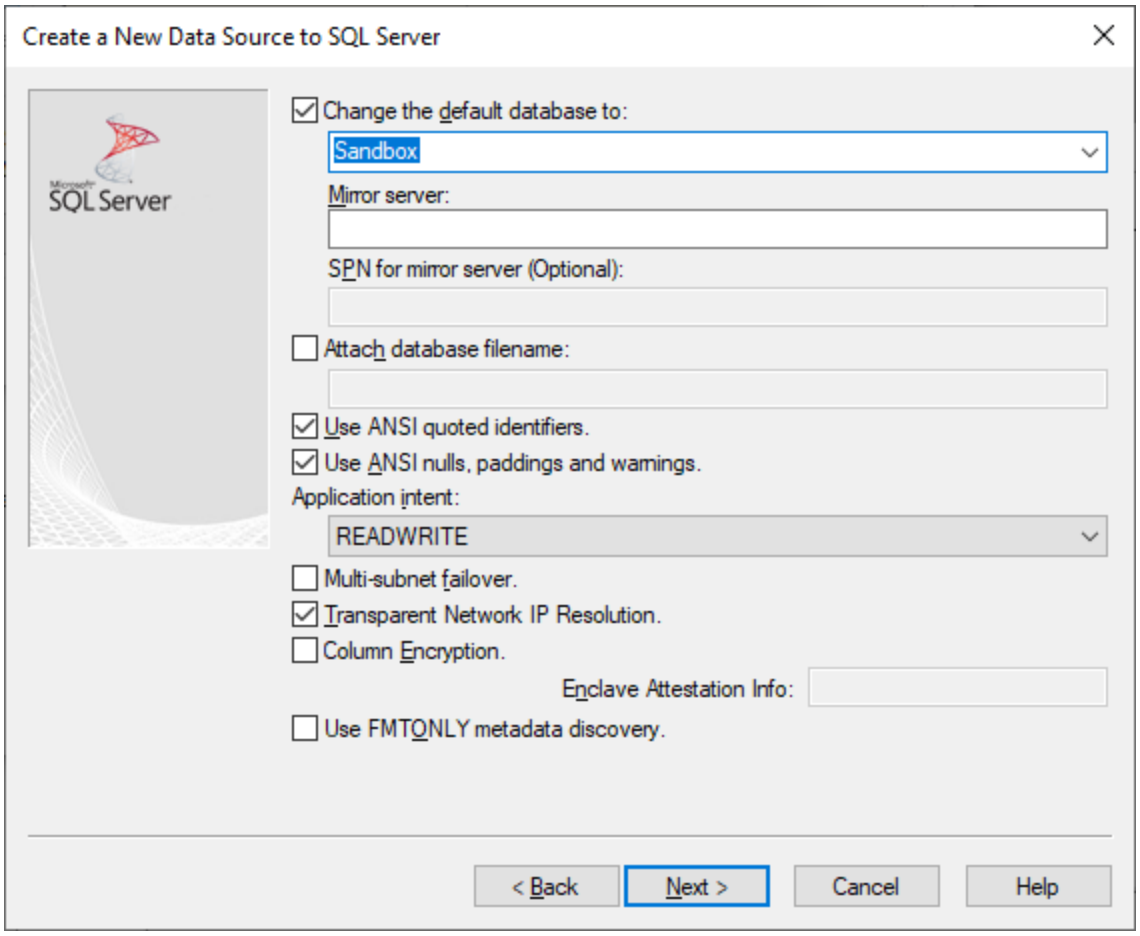


7. Wählen Sie die Option **Mit integrierter Windows NT-Authentifizierung**, wenn der Datenbankserver so konfiguriert ist, dass er Verbindungen von bei der Windows Domain angemeldeten Benutzern gestattet. Wählen Sie andernfalls je nach Bedarf eine der anderen Optionen aus. In diesem Beispiel wird **Mit SQL Server-Authentifizierung...** verwendet. In diesem Fall müssen Benutzername und Passwort in die entsprechenden Felder eingegeben werden.

© 2016-2022 Altova GmbH

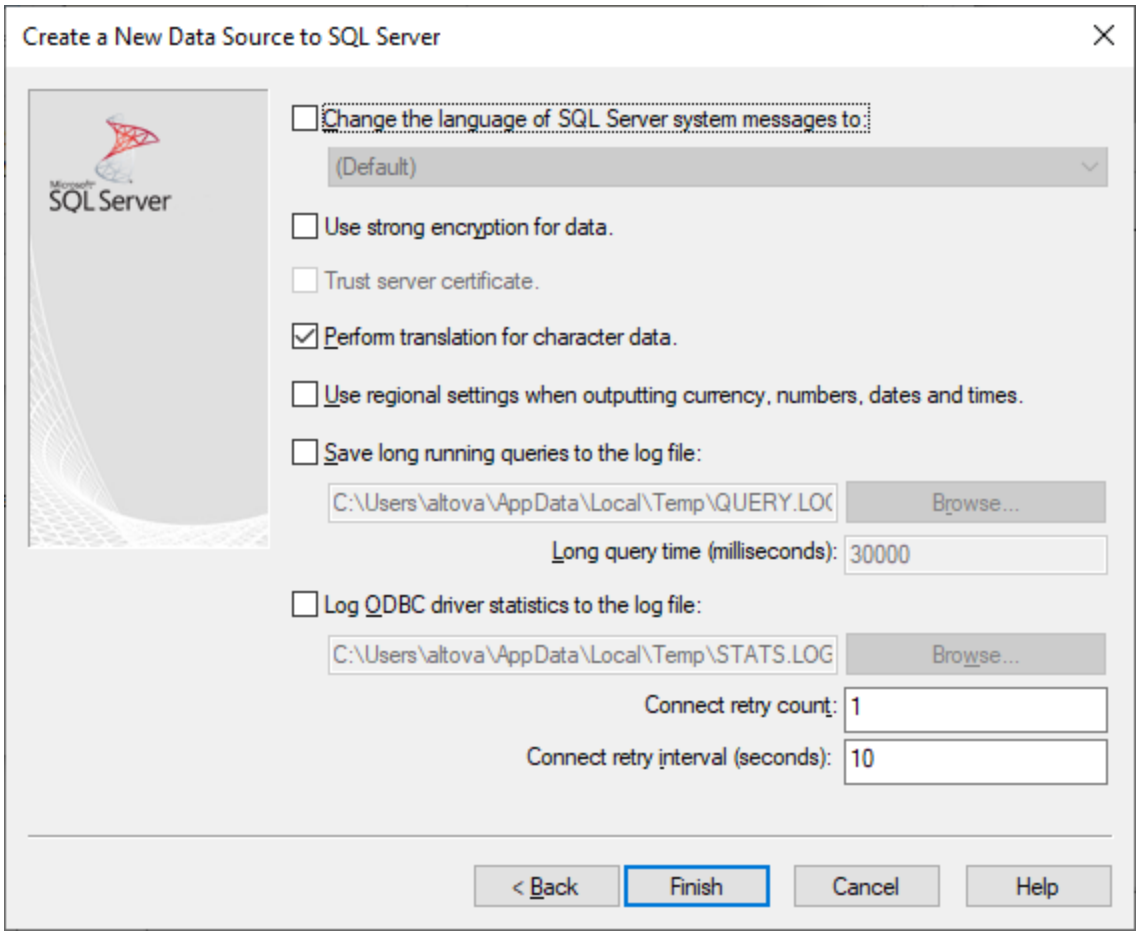


8. Aktivieren Sie optional das Kontrollkästchen **Standarddatenbank ändern in** und geben Sie den Namen der Datenbank, zu der Sie eine Verbindung herstellen (in diesem Beispiel **Sandbox**) ein.

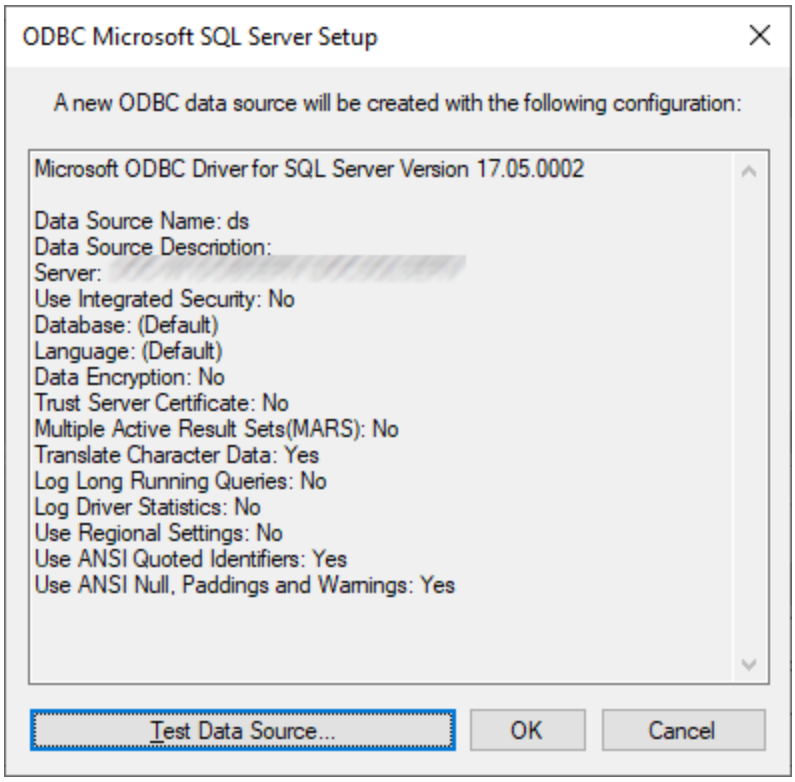


9. Klicken Sie auf **Weiter** und konfigurieren Sie optional weitere Parameter für diese Verbindung.

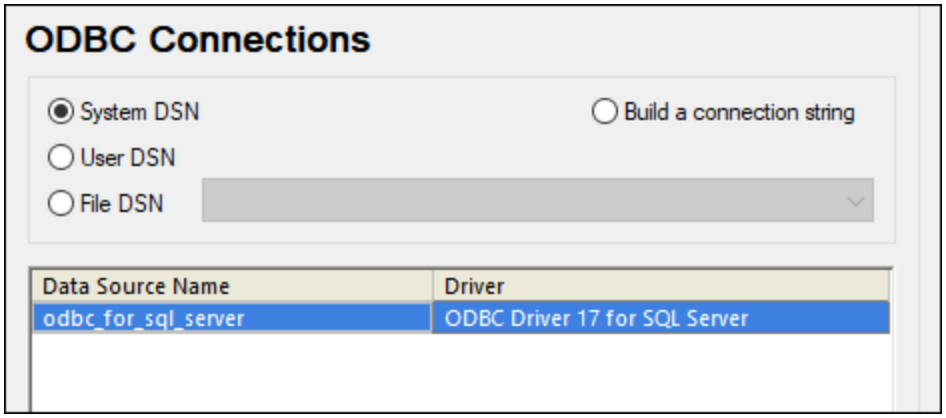
© 2016-2022 Altova GmbH



10. Klicken Sie auf **Fertig stellen**. Daraufhin wird ein Bestätigungsdialogfeld mit den Verbindungsinformationen angezeigt.



11. Klicken Sie auf **OK.** Daraufhin wird die Datenquelle je nach Konfiguration in der Liste der **Benutzer**- oder **System**-Datenquellen angezeigt, z.B:



10.2.9.12 MySQL (ODBC)

In diesem Kapitel wird gezeigt, wie Sie von einem Windows-Rechner aus über den ODBC-Treiber eine Verbindung zu einem MySQL-Datenbankserver herstellen. Der MySQL ODBC-Treiber steht auf Windows nicht zur Verfügung, daher müssen Sie ihn separat herunterladen und installieren. In diesem Beispiel wird die MySQL Connector/ODBC-Version 8.0 verwendet.

© 2016-2022 Altova GmbH

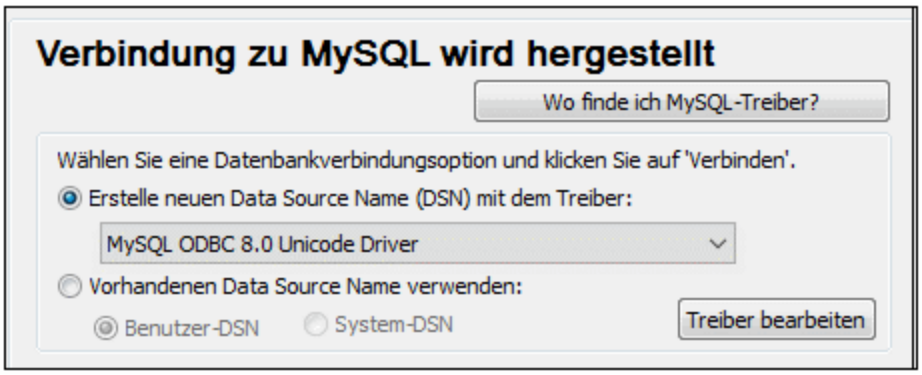
Voraussetzungen:

- Der MySQL ODBC-Treiber muss auf Ihrem Betriebssystem installiert sein. Lesen Sie nach in der Dokumentation zu MySQL, um zu ermittlen, welche Treiberversion für Ihre Datenbankserverversion empfohlen wird (siehe https://dev.mysql.com/downloads/connector/odbc/).
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Host, Datenbank, Port, Benutzername und Passwort.

Wenn Sie MySQL Connector/ODBC für die 64-Bit-Plattform installiert haben, stellen Sie sicher, dass auch UModel für die 64-Bit-Plattform installiert ist.

So stellen Sie über ODBC eine Verbindung zu MySQL her:

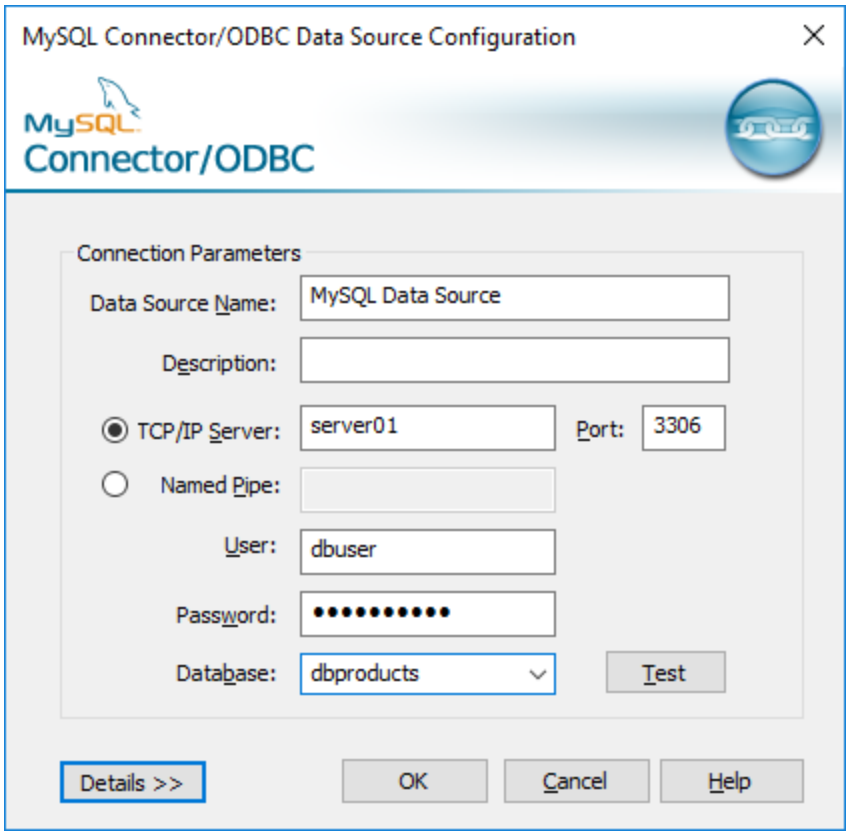
1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Wählen Sie **MySQL (ODBC)** und klicken Sie auf **Weiter**.



3. Wählen Sie **Erstelle neuen Data Source Name (DSN) mit dem Treiber** und wählen Sie einen MySQL-Treiber aus. Wenn in der Liste kein MySQL-Treiber verfügbar ist, klicken Sie auf **Treiber bearbeiten** und wählen Sie einen beliebigen verfügbaren MySQL-Treiber aus (die Liste enthält alle auf Ihrem Betriebssystem installierten ODBC-Treiber).

   Wenn Sie UModel 64-Bit installiert haben, werden in der Liste die 64-Bit-ODBC-Treiber angezeigt. Andernfalls werden die 32-Bit-ODBC-Treiber angezeigt. Siehe auch Anzeigen der verfügbaren ODBC-Treiber 598.

4. Klicken Sie auf **Verbinden**.



5. Geben Sie in das Feld "Datenquellenname" einen Namen ein, anhand dessen Sie diese ODBC-Datenquelle in Zukunft identifizieren können.
6. Füllen Sie die Anmeldeinformationen für die Datenbankverbindung aus (TCP/IP Server, Benutzer, Passwort), wählen Sie eine Datenbank aus und klicken Sie auf **OK**.

Anmerkung: Wenn es sich um einen Remote-Datenbankserver handelt, muss er vom Server-Administrator so konfiguriert sein, dass er remote-Verbindungen von der IP-Adresse Ihres Rechners zulässt. Wenn Sie außerdem auf **Details>>** klicken, können Sie eine Reihe zusätzlicher Parameter konfigurieren. Lesen Sie die Dokumentation zum Treiber, bevor Sie die Standardwerte ändern.

10.2.9.13 Oracle (JDBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung von einem Client-Rechner zu einem Oracle Datenbankserver mittels JDBC. Die Verbindung wird mit Hilfe des auf der Oracle Website verfügbaren **Oracle Instant Client Package (Basic)** als reine Java-Verbindung hergestellt. Der Vorteil dieser Verbindungsart ist, dass nur die Java-Umgebung und die vom Oracle Instant Client Package bereitgestellten .jar-Bibliotheken benötigt werden, sodass Sie keinen komplexeren Datenbank-Client installieren und konfigurieren müssen.

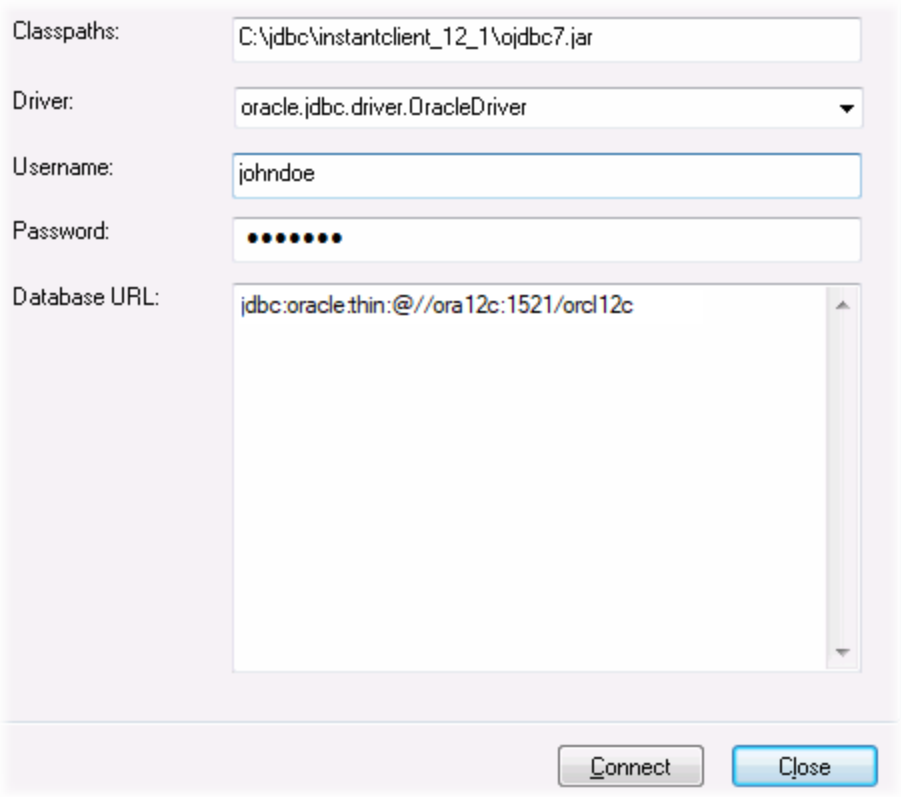
© 2016-2022 Altova GmbH

Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- **Oracle Instant Client Package (Basic)** muss auf Ihrem Betriebssystem verfügbar sein. Das Paket kann von der offiziellen Oracle Website heruntergeladen werden. In diesem Beispiel wird Oracle Instant Client Package Version 12.1.0.2.0 für Windows 32-Bit und folglich Oracle JDK 32-Bit verwendet.
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Host, Port, Servicename, Benutzername und Passwort.

So stellen Sie über das Instant Client Package eine Verbindung zu Oracle her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. Falls nötig, können Sie auch eine durch Semikola getrennte Liste von .jar-Dateipfaden eingeben. Die benötigte .jar-Datei in diesem Beispiel befindet sich unter dem folgenden Pfad: **C:\jdbc\instantclient_12_1\ojdbc7.jar**. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH 602).
4. Wählen Sie im Feld "Treiber" **oracle.jdbc.driver.OracleDriver** oder **oracle.jdbc.driver.OracleDriver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpath" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).
5. Geben Sie den Benutzernamen und das Passwort in die entsprechenden Textfelder ein.



6. Geben Sie in das Textfeld "Datenbank-URL" den Connection String zum Datenbankserver ein, indem Sie die hervorgehobenen Werte durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:oracle:thin:@//host:port:service
```

7. Klicken Sie auf **Verbinden**.

10.2.9.14 Oracle (ODBC)

In diesem Beispiel wird ein häufig vorkommendes Szenario beschrieben: Sie stellen von UModel aus über einen auf dem lokalen Betriebssystem installierten Oracle Datenbank Client eine Verbindung zu einer Oracle-Datenbank im Netzwerk her.

Das Beispiel enthält eine Anleitung, wie man mit Hilfe des Datenbankverbindungsassistenten in UModel eine ODBC-Datenquelle (DSN) konfiguriert. Wenn Sie bereits einen DSN erstellt haben oder wenn Sie diesen lieber direkt über den ODBC-Datenquellen-Administrator in Windows erstellen, können Sie dies tun und den DSN dann auswählen, sobald Sie vom Assistenten dazu aufgefordert werden. Nähere Informationen zu ODBC-Datenquelle finden Sie unter Einrichten einer ODBC-Verbindung 596.

 © 2016-2022 Altova GmbH

Voraussetzungen:

- Der Oracle Datenbank Client (der den Oracle-ODBC-Treiber enthält) muss auf Ihrem Betriebssystem installiert und konfiguriert sein. Eine Anleitung zum Installieren und Konfigurieren eines Oracle Datenbank Client finden Sie in der Dokumentation zur Oracle-Software.
- Die Datei **tnsnames.ora** im Oracle-Startverzeichnis enthält einen Eintrag, der die Datenbankverbindungsparameter in etwa wie folgt beschreibt:

```
ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = server01)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = orcl)
      (SERVER = DEDICATED)
    )
  )
```

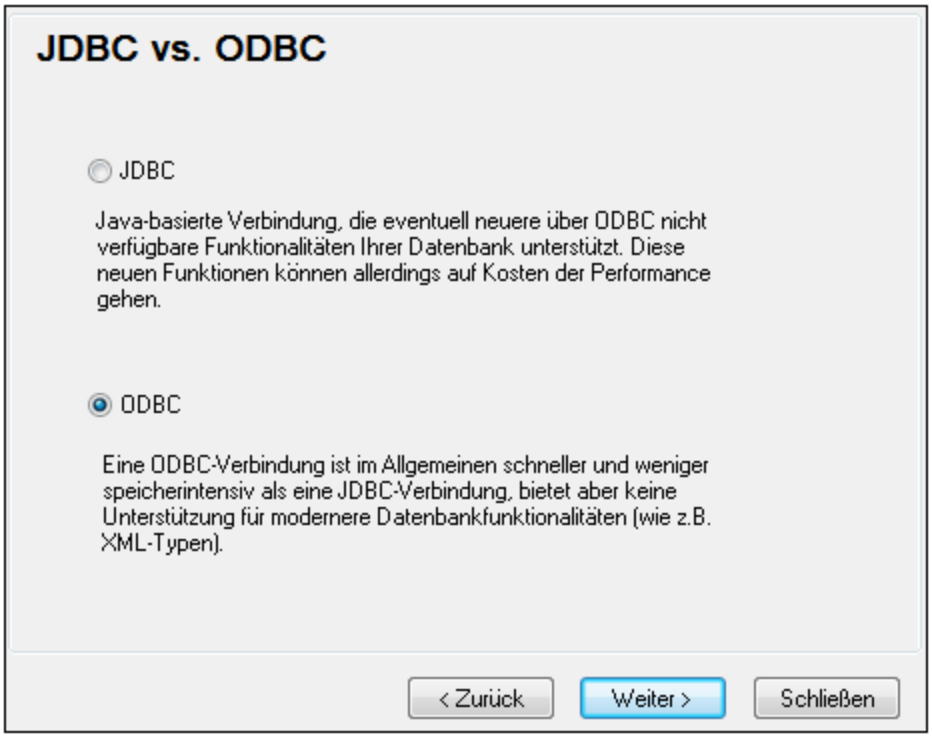
Der Pfad zur Datei **tnsnames.ora** hängt davon ab, wo das Oracle-Startverzeichnis installiert wurde. Beim Oracle-Datenbank-Client 11.2.0 könnte der Standardpfad folgendermaßen lauten:

```
C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
```

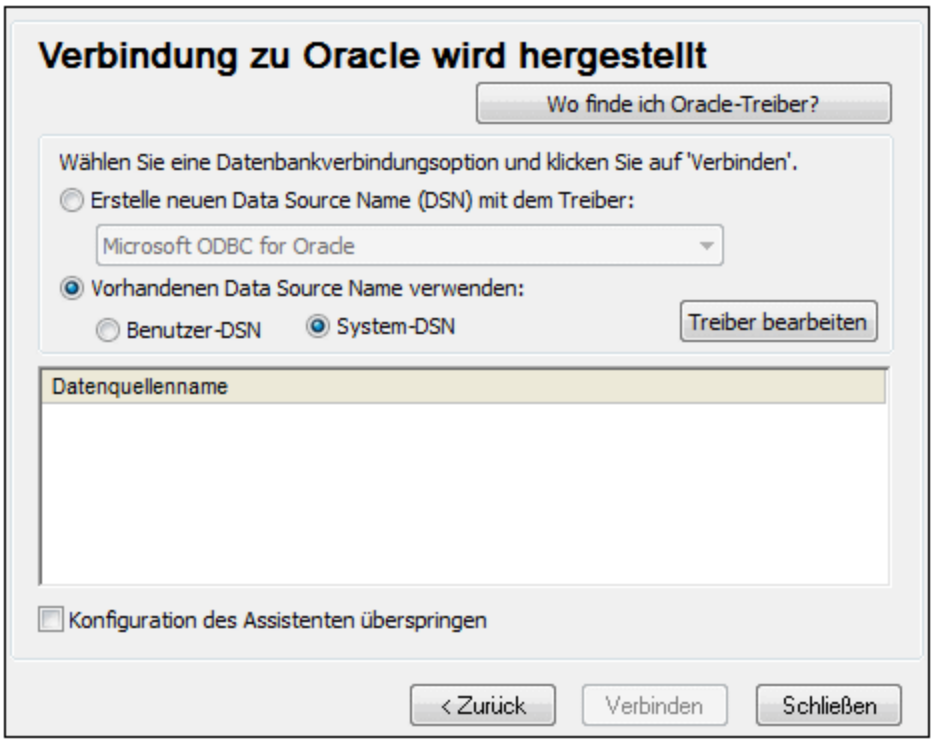
Sie können neue Einträge zur Datei **tnsnames.ora** hinzufügen, indem Sie die Verbindungsdetails entweder hineinkopieren und die Datei speichern oder indem Sie den Oracle *Net-Konfigurationsassistenten* ausführen (falls vorhanden). Wenn diese Werte bei der Konfiguration in Dropdown-Listen aufscheinen sollen, müssen Sie den Pfad zum admin-Ordner eventuell als **TNS_ADMIN**-Umgebungsvariable hinzufügen.

So stellen Sie über ODBC eine Verbindung zu Oracle her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Wählen Sie **Oracle (ODBC / JDBC)** und klicken Sie auf **Weiter**.

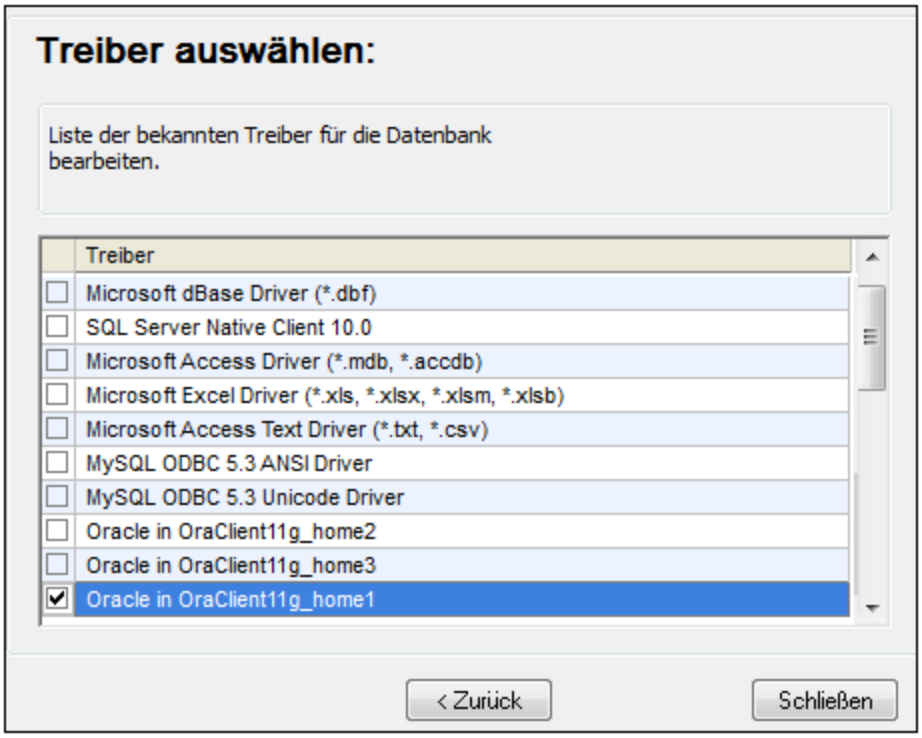


3. Wählen Sie **ODBC**.

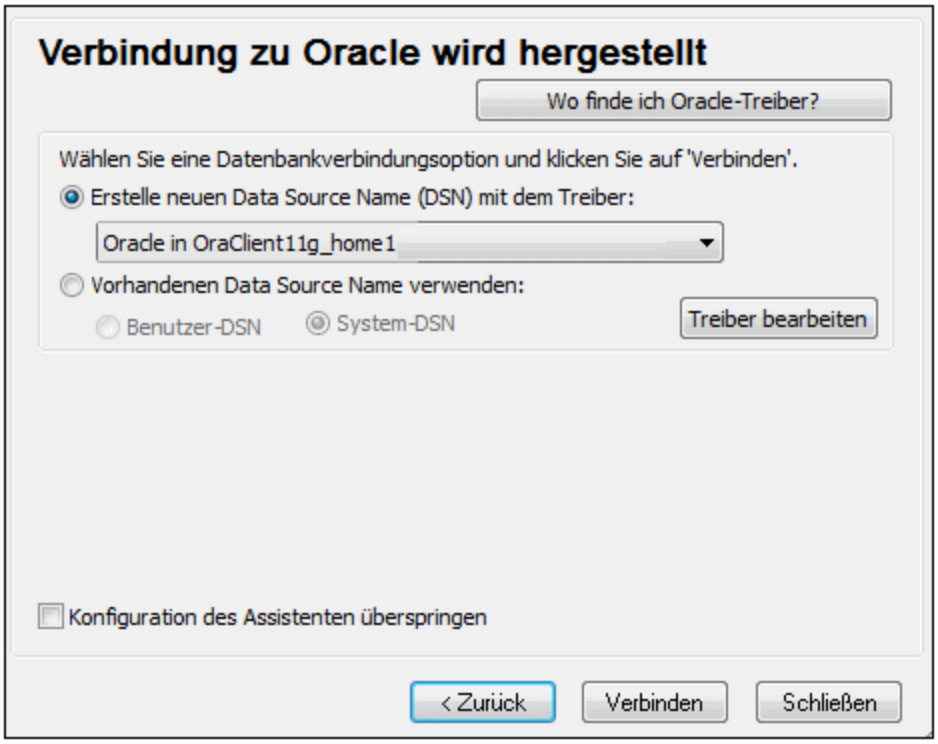


4. Klicken Sie auf **Treiber bearbeiten**.

© 2016-2022 Altova GmbH



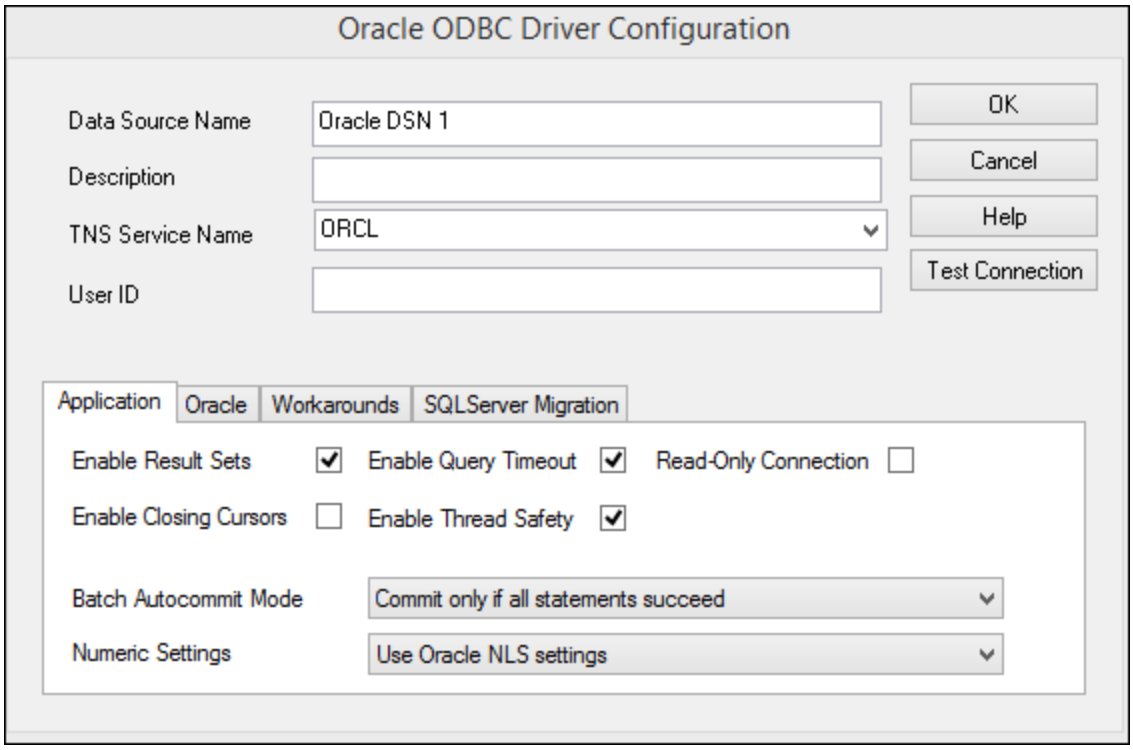
5. Wählen Sie die gewünschten Oracle-Treiber aus (in diesem Beispiel **Oracle in OraClient11g_home1**). In der Liste werden die nach der Installation des Oracle Client auf Ihrem System verfügbaren Oracle-Treiber angezeigt.
6. Klicken Sie auf **Zurück.**
7. Wählen Sie **Erstelle neuen Data Source Name (DSN) mit dem Treiber** und wählen Sie den in Schritt 4 ausgewählten Oracle-Treiber aus.



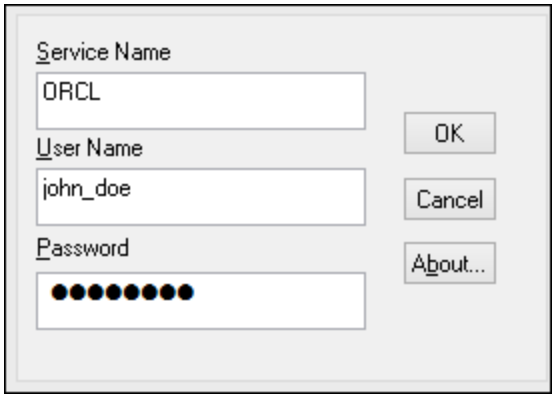
Verwenden Sie den von Microsoft bereitgestellten Treiber **Microsoft ODBC for Oracle** möglichst nicht. Microsoft empfiehlt, den von Oracle bereitgestellten ODBC-Treiber zu verwenden (siehe http://msdn.microsoft.com/en-us/library/ms714756%28v=vs.85%29.aspx)

8. Klicken Sie auf **Verbinden**.

© 2016-2022 Altova GmbH



9. Geben Sie im Textfeld "Datenquellenname" einen Namen für die Datenquelle ein (in diesem Beispiel **Oracle DSN 1**).
10. Geben Sie im Feld "TNS-Dienstname" den in der Datei **tnsnames.ora** definierten Verbindungsnamen ein (siehe Voraussetzungen[643]). In diesem Beispiel lautet der Verbindungsname **ORCL**. *Anmerkung:* Wenn die Dropdown-Liste der Auswahlliste mit den Werten der **tnsnames.ora**-Datei befüllt werden soll, müssen Sie den Pfad zum admin-Ordner als **TNS_ADMIN**-Umgebungsvariable hinzufügen.
11. Klicken Sie auf **OK**.



12. Geben sie den Benutzernamen und das Passwort für die Datenbank ein und klicken Sie auf OK.

10.2.9.15 PostgreSQL (ODBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung über den ODBC-Treiber von einem Windows-Rechner zu einem PostgreSQL Datenbankserver. Der PostgreSQL ODBC-Treiber ist auf

Windows nicht verfügbar und muss separat heruntergeladen und installiert werden. In diesem Beispiel wird der von der offiziellen Website heruntergeladene psqlODBC-Treiber (Version 11.0) verwendet (siehe auch Übersicht über Datenbanktreiber[580]).

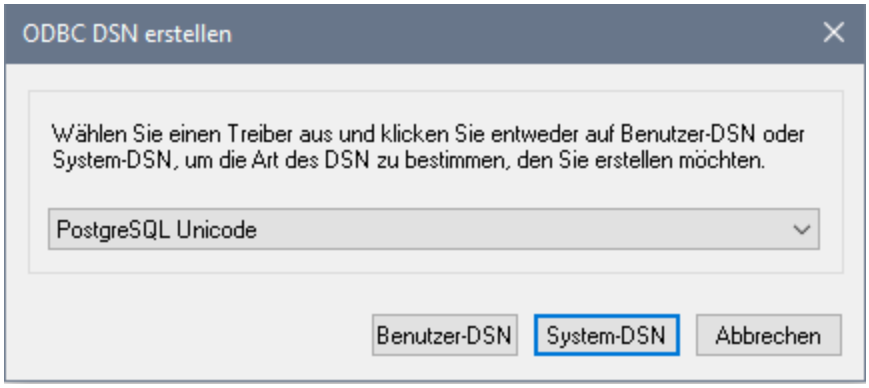
Anmerkung: Sie können die Verbindung zu einer PostgreSQL-Datenbank auch direkt (ohne ODBC-Treiber) herstellen, siehe Einrichten einer PostgreSQL-Verbindung[604].

Voraussetzungen:

- Der *psqlODBC*-Treiber muss auf Ihrem Betriebssystem installiert sein.
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Server, Port, Datenbank, Benutzername und Passwort.

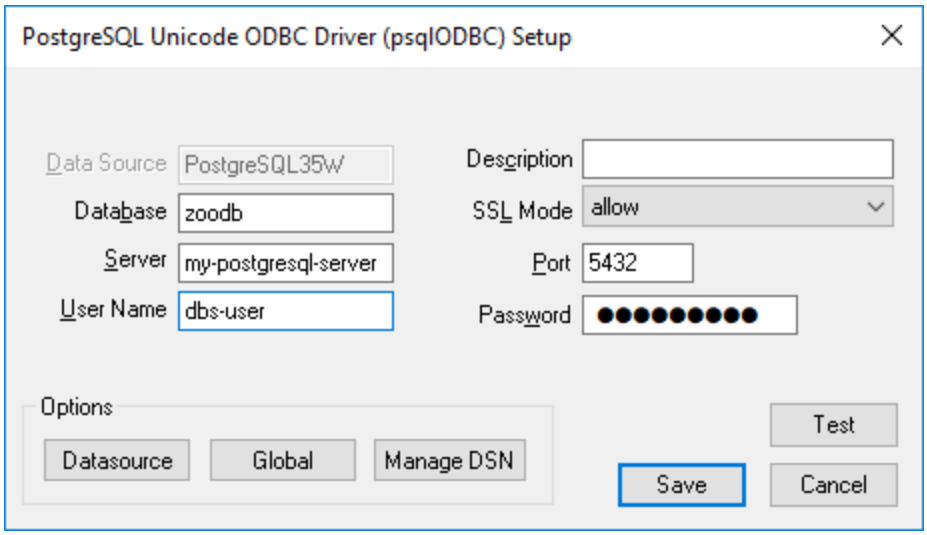
So richten Sie über ODBC eine Verbindung zu PostgreSQL ein:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Aktivieren Sie die Option **Benutzer-DSN**.
4. Klicken Sie auf **Neuen DSN erstellen** und wählen Sie den Treiber aus der Dropdown-Liste aus. Wenn in der Liste kein PostgreSQL-Treiber zur Verfügung steht, stellen Sie sicher, dass der PostgreSQL ODBC-Treiber auf Ihrem Betriebssystem installiert ist, wie in den Voraussetzungen oben erwähnt.



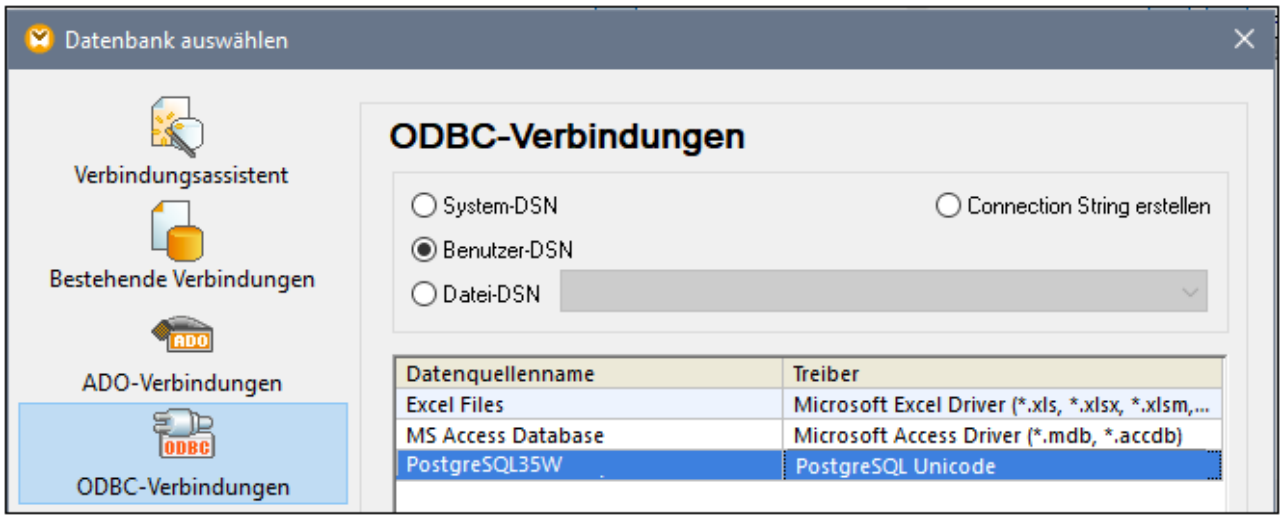
5. Klicken Sie auf **Benutzer-DSN**.

© 2016-2022 Altova GmbH



6. Füllen Sie die Anmeldeinformationen für die Datenbankverbindung aus (diese müssen vom Inhaber der Datenbank bereitgestellt werden) und klicken Sie anschließend auf **Speichern**.

Die Verbindung steht nun in der Liste der ODBC-Verbindungen zur Verfügung. Um eine Verbindung zur Datenbank herzustellen, können Sie entweder auf die Verbindung doppelklicken oder die Verbindung auswählen und auf **Verbinden** klicken.



10.2.9.16 Progress OpenEdge (JDBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einem Progress OpenEdge 11.6-Datenbankserver mittels JDBC.

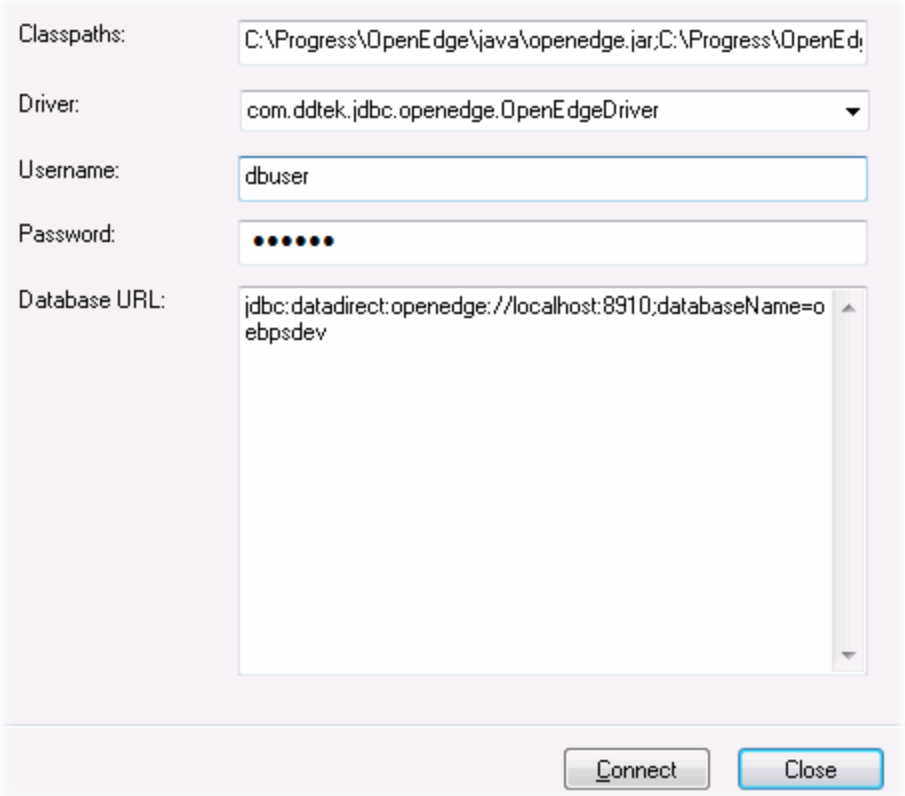
Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen[788]; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- Die `PATH`-Umgebungsvariable des Betriebssystems muss den Pfad zum `bin`-Verzeichnis des JRE- bzw. JDK-Installationsverzeichnisses enthalten, z.B. `C:\Programe (x86)\Java\jre1.8.0_51\bin`.
- Der Progress OpenEdge JDBC-Treiber muss auf Ihrem Betriebssystem installiert sein. In diesem Beispiel erfolgt die JDBC-Verbindung über die Treiberkomponentendateien **openedge.jar** und **pool.jar,** die als Teil der OpenEdge SDK-Installation unter **C:\Progress\OpenEdge\java** zur Verfügung stehen.
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Host, Port, Datenbankname, Benutzername und Passwort.

So stellen Sie über JDBC eine Verbindung zu OpenEdge her:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. Falls nötig, können Sie auch eine durch Semikola getrennte Liste von .jar-Dateipfaden eingeben. Die benötigte .jar-Datei in diesem Beispiel befindet sich unter dem folgenden Pfad: **`C:\Progress\OpenEdge\java\openedge.jar;C:\Progress\OpenEdge\java\pool.jar;`**. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH[602]).
4. Wählen Sie im Feld "Treiber" **com.ddtek.jdbc.openedge.OpenEdgeDriver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpaths" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).

© 2016-2022 Altova GmbH



5. Geben Sie den Benutzernamen und das Passwort für die Datenbank in die entsprechenden Textfelder ein.
6. Geben Sie in das Textfeld "Datenbank-URL" den Connection String zum Datenbankserver ein, indem Sie die hervorgehobenen Werte durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:datadirect:openedge://host:port;databaseName=db_name
```

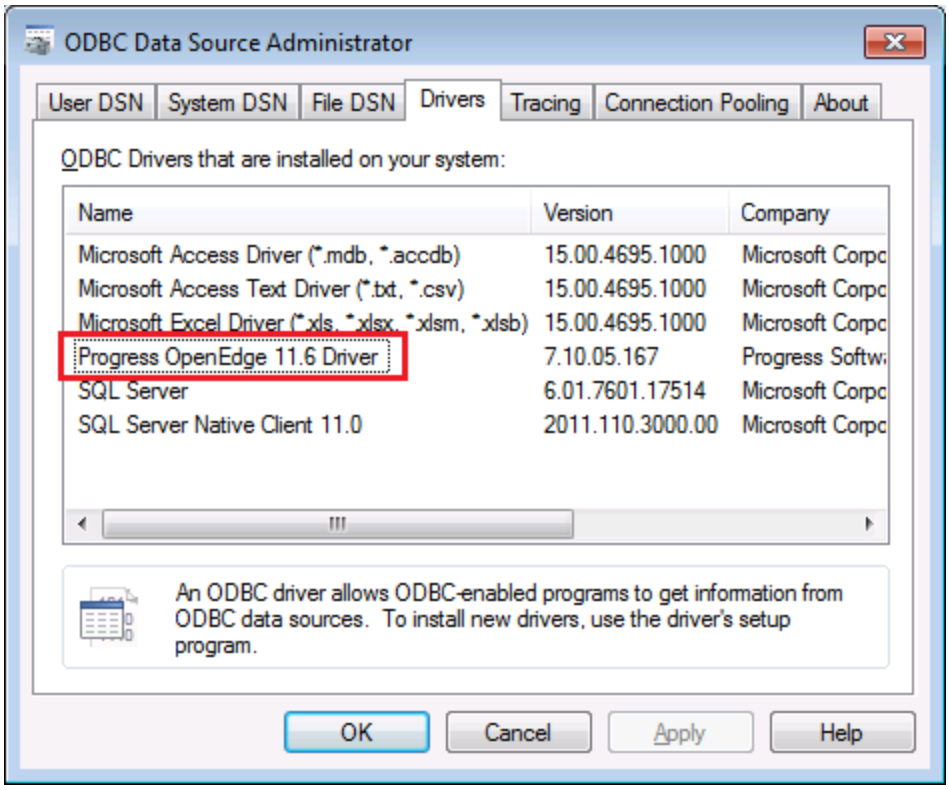
7. Klicken Sie auf **Verbinden**.

10.2.9.17 Progress OpenEdge (ODBC)

Dieses Kapitel enthält eine Beispielanleitung für das Herstellen einer Verbindung zu einer Progress OpenEdge-Datenbank über den Progress OpenEdge 11.6 ODBC-Treiber.

Voraussetzungen:

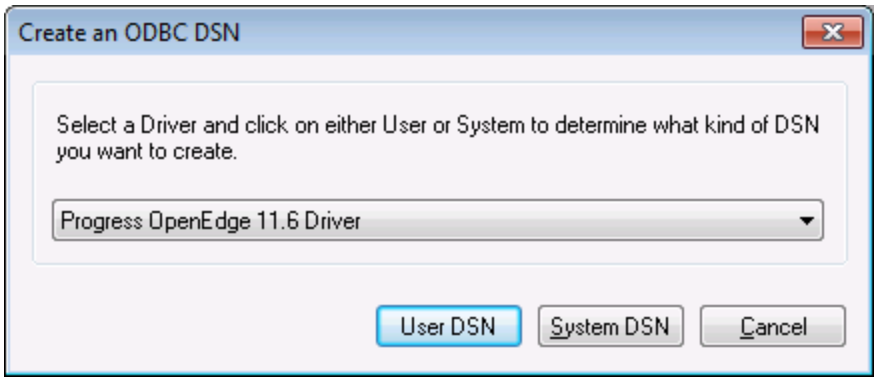
- Der *ODBC Connector for Progress OpenEdge*-Treiber muss auf Ihrem Betriebssystem installiert sein. Der Progress OpenEdge ODBC-Treiber kann von der Website des Anbieters heruntergeladen werden (siehe auch Übersicht über Datenbanktreiber[580]). Bei Ausführung der 32-Bit-Version von UModel muss der 32-Bit-Treiber und bei Ausführung der 64-Bit-Version der 64-Bit-Treiber heruntergeladen werden. Überprüfen Sie nach Abschluss der Installation, ob der ODBC-Treiber auf Ihrem Rechner zur Verfügung steht (siehe auch Anzeigen der verfügbaren ODBC-Treiber[598]).



- Sie verfügen über die folgenden Datenbankverbindungsinformationen: Host-Name, Datenbankname, Benutzer-ID und Passwort.

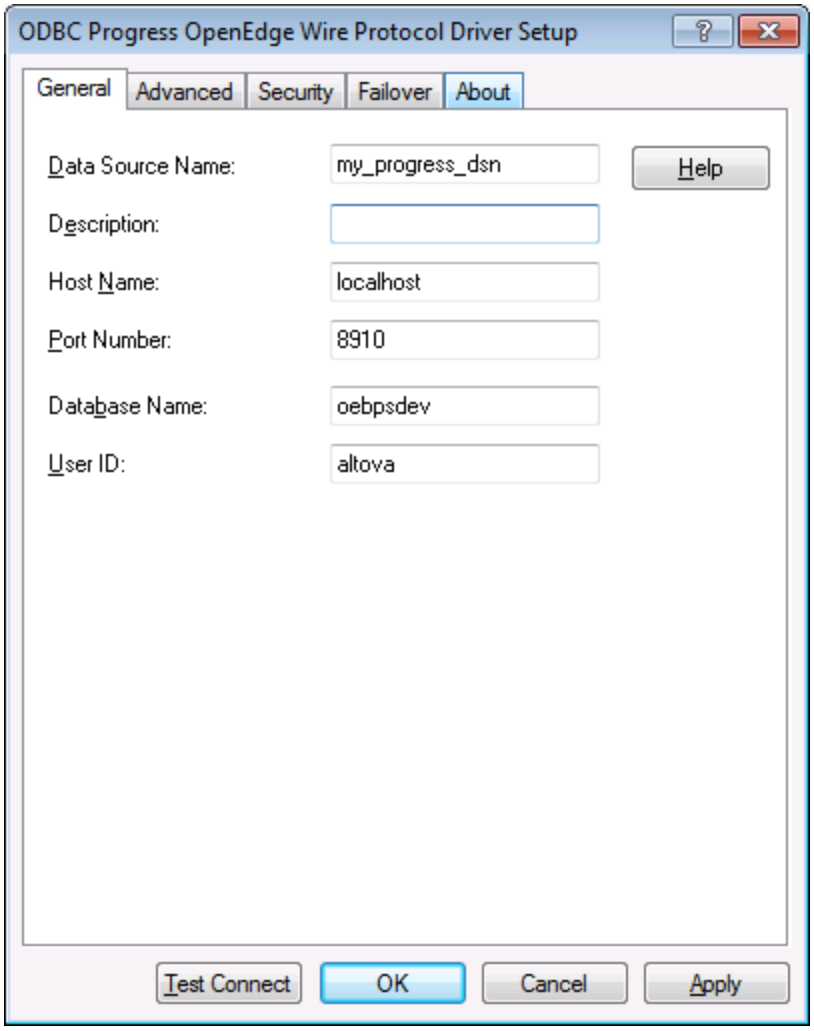
So stellen Sie über ODBC eine Verbindung zu Progress OpenEdge her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **ODBC-Verbindungen**.
3. Wählen Sie Benutzer-DSN (oder alternativ dazu **System-DSN** oder **Datei-DSN** - in diesem Fall sind die nun folgenden Schritte ähnlich).
4. Klicken Sie auf **Hinzufügen** .
2. Wählen Sie aus der Liste **Progress OpenEdge Driver** aus und klicken Sie auf **Benutzer-DSN** (oder gegebenenfalls auf **System-DSN**).

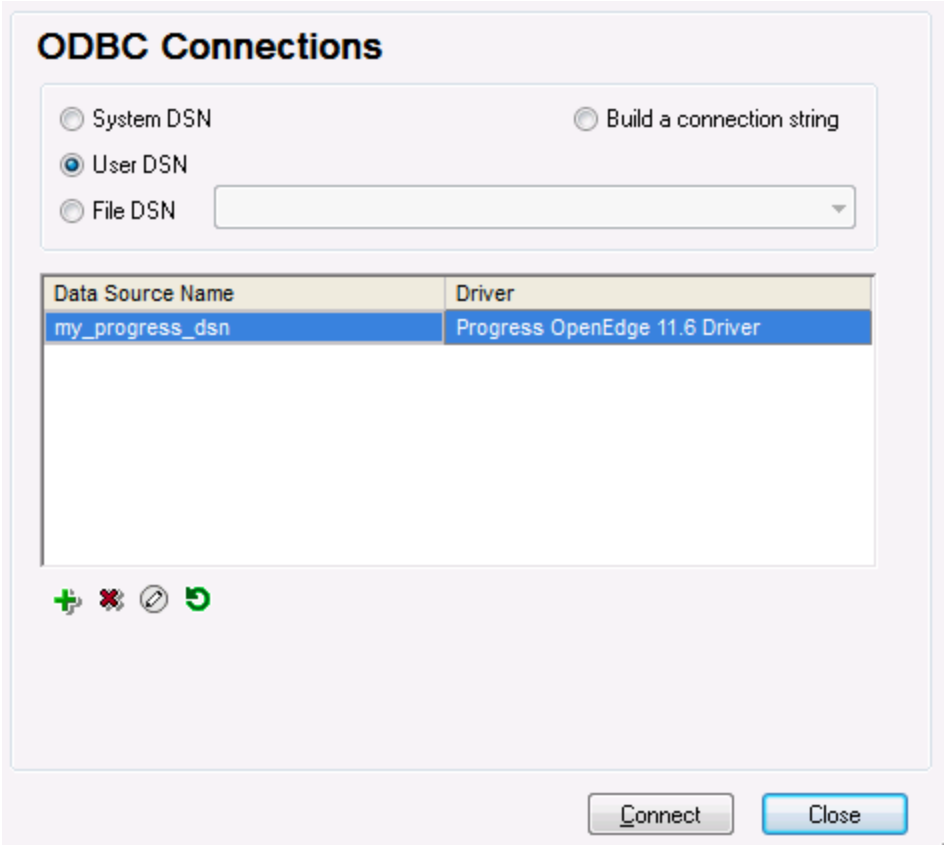


 © 2016-2022 Altova GmbH

3. Füllen Sie die Anmeldeinformationen für die Datenbankverbindung aus (Datenbank, Server, Port, Benutzername und Passwort) und klicken Sie auf **OK**. Um die Verbindung zu überprüfen, bevor Sie die eingegebenen Daten speichern, klicken Sie auf **Verbindung testen**.



4. Klicken Sie auf OK. Die neue Datenquelle wird nun in der Liste der ODBC-Datenquellen angezeigt.



5. Klicken Sie auf **Verbinden**.

10.2.9.18 Sybase (JDBC)

Dieses Kapitel enthält eine Beispielanleitung, wie Sie über JDBC eine Verbindung zu einer Sybase-Datenbank herstellen.

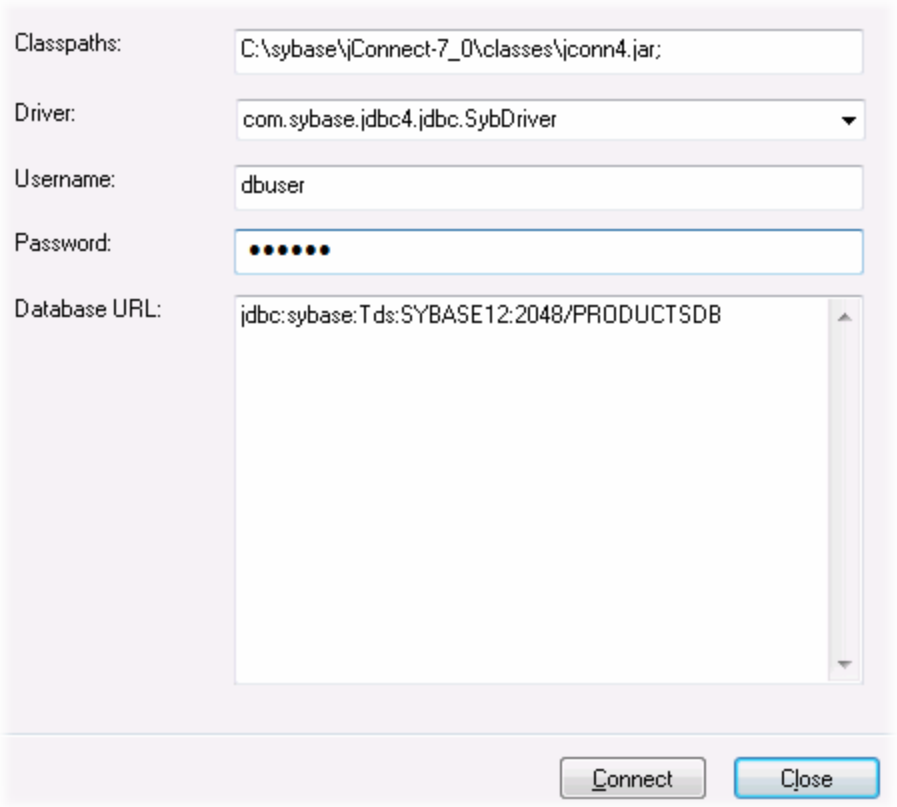
Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- Auf Ihrem Betriebssystem muss die Sybase *jConnect*-Komponente installiert sein (in diesem Beispiel wird *jConnect 7.0* verwendet. Sie wird als Teil des *Sybase Adaptive Server Enterprise PC Client* installiert). Eine Anleitung zur Installation des Datenbank Client finden Sie in der Sybase-Dokumentation.
- Sie haben die folgenden Datenbankverbindungsinformationen zur Verfügung: Host, Port, Datenbankname, Benutzername und Passwort.

 © 2016-2022 Altova GmbH

So stellen Sie über JDBC eine Verbindung zu Sybase her:

1. Starten Sie den Datenbank-Verbindungsassistenten[578].
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. Falls nötig, können Sie auch eine durch Semikola getrennte Liste von .jar-Dateipfaden eingeben. Die benötigte .jar-Datei in diesem Beispiel befindet sich unter dem folgenden Pfad: **C:\sybase\jConnect-7_0\classes\jconn4.jar**. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH[602]).
4. Wählen Sie aus der Liste der verfügbaren JDBC-Treiber den Sybase JDBC-Treiber aus (in diesem Beispiel **com.sybase.jdbc4.jdbc.SybDriver**). Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpaths" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).



5. Geben Sie den Benutzernamen und das Passwort in die entsprechenden Textfelder ein.
6. Geben Sie in das Textfeld "Datenbank-URL" den Connection String zum Datenbankserver ein, indem Sie die hervorgehobenen Werte durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:sybase:Tds:hostName:port/databaseName
```

7. Klicken Sie auf **Verbinden**.

10.2.9.19 Teradata (JDBC)

In diesem Beispiel wird gezeigt, wie Sie über JDBC eine Verbindung zu einem Teradata-Datenbankserver herstellen.

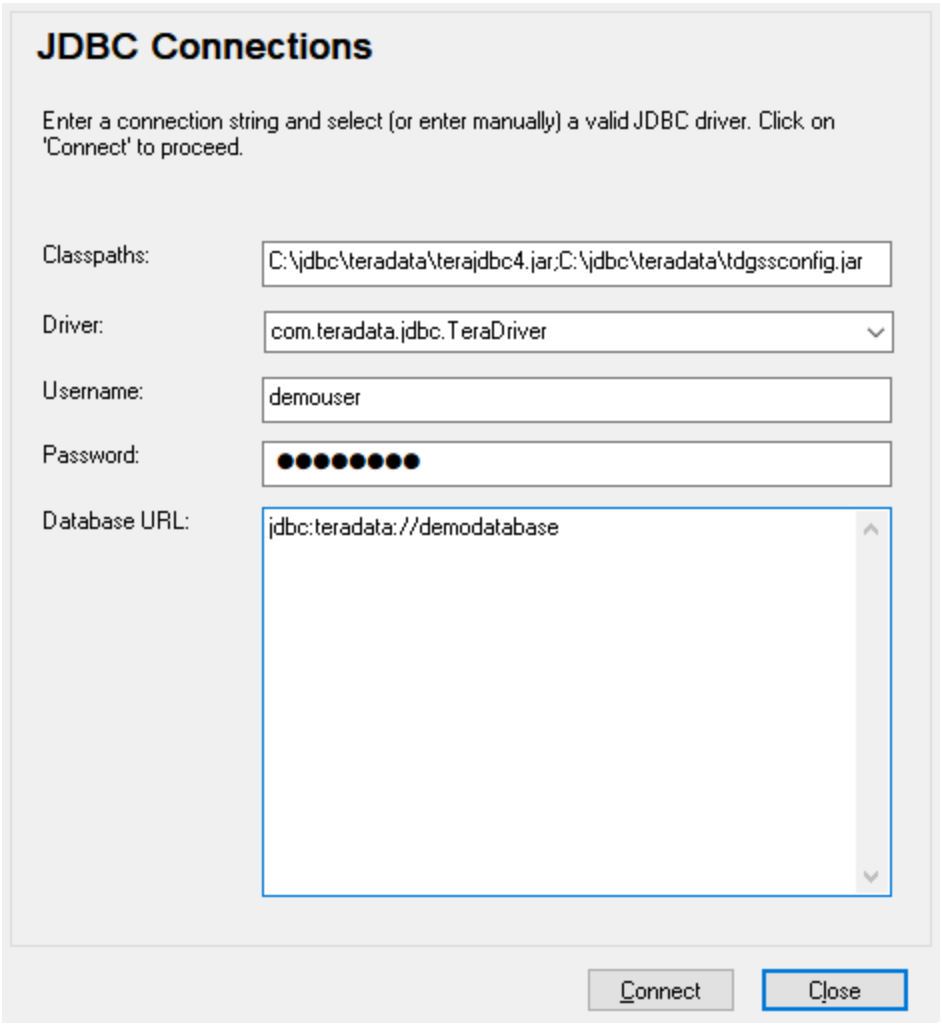
Voraussetzungen:

- Auf Ihrem Betriebssystem muss JRE (Java Runtime Enviroment) oder Java Development KIT (JDK) installiert sein. Dabei muss es sich entweder um Oracle JDK oder einen Open Source Build wie Oracle OpenJDK handeln.UModel ermittelt den Pfad zur Java Virtual Machine (JVM) anhand der folgenden Ordner und zwar in folgender Reihenfolge: a) anhand des benutzerdefinierten JVM-Pfads, den Sie eventuell in den **Applikationsoptionen** definiert haben, siehe Java-Einstellungen 788; b) anhand des JVM-Pfads in der Windows Registry; c) anhand der `JAVA_HOME`-Umgebungsvariablen.
- Stellen Sie sicher, dass die Plattform von UModel (32-Bit, 64-Bit) mit der des JRE/JDK übereinstimmt.
- Der JDBC-Treiber (eine oder mehrere .jar-Dateien, die die Verbindung zur Datenbank herstellen) muss auf Ihrem Betriebssystem installiert sein. In diesem Beispiel wird der Teradata JDBC-Treiber 16.20.00.02 verwendet. Nähere Informationen dazu finden Sie unter https://downloads.teradata.com/download/connectivity/jdbc-driver.
- Sie haben die folgenden Datenbankinformationen zur Verfügung: Host, Datenbank, Port, Benutzername und Passwort.

So stellen Sie über JDBC eine Verbindung zu Teradata her:

1. Starten Sie den Datenbank-Verbindungsassistenten 578.
2. Klicken Sie auf **JDBC-Verbindungen**.
3. Geben Sie neben "Classpaths" den Pfad zur .jar-Datei , die die Verbindung zur Datenbank bereitstellt, ein. Falls nötig, können Sie auch eine durch Semikola getrennte Liste von .jar-Dateipfaden eingeben. Die benötigte .jar-Datei in diesem Beispiel befindet sich unter dem folgenden Pfad:**C:\jdbc\teradata**. Beachten Sie, dass Sie das Textfeld "Classpaths" leer lassen können, wenn Sie den/die .jar-Dateipfad(e) zur Umgebungsvariablen CLASSPATH des Betriebssystems hinzugefügt haben (siehe auch Konfigurieren des CLASSPATH 602).
4. Wählen Sie im Feld "Treiber" **com.teradata.jdbc.TeraDriver** aus. Beachten Sie, dass dieser Eintrag zur Verfügung steht, wenn entweder im Textfeld "Classpath" oder in der Umgebungsvariablen `CLASSPATH` des Betriebssystems eine gültige .jar-Datei gefunden wird (siehe vorheriger Schritt).

 © 2016-2022 Altova GmbH



5. Geben Sie den Benutzernamen und das Passwort in die entsprechenden Textfelder ein.
6. Geben Sie in das Textfeld "Datenbank-URL" den Connection String zum Datenbankserver ein, indem Sie die hervorgehobenen Werte durch die entsprechenden Werte für Ihren Datenbankserver ersetzen.

```
jdbc:teradata://databaseServerName
```

7. Klicken Sie auf **Verbinden**.

10.2.9.20 Teradata (ODBC)

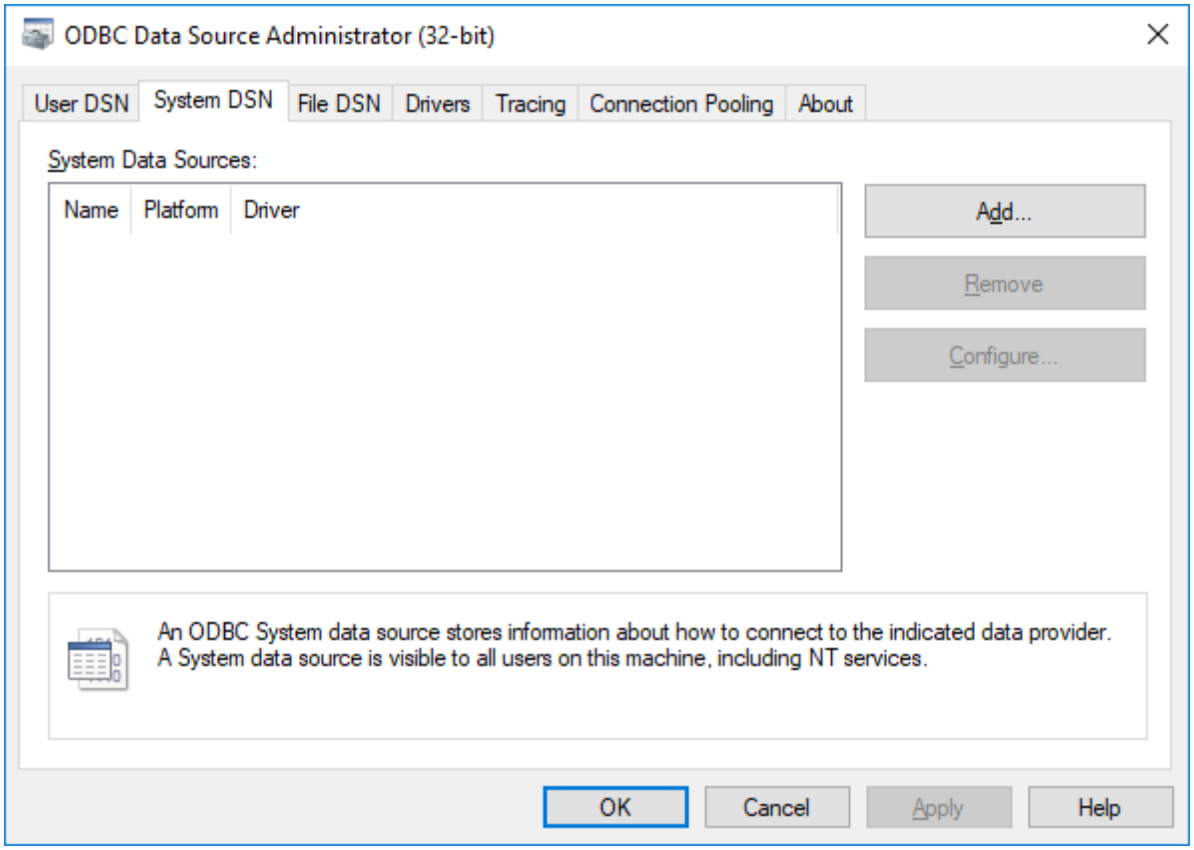
In diesem Beispiel wird gezeigt, wie Sie über ODBC eine Verbindung zu einem Teradata-Datenbankserver herstellen.

Voraussetzungen:

- Der Teradata ODBC-Treiber (https://downloads.teradata.com/download/connectivity/odbc-driver/windows) muss installiert sein. In diesem Beispiel wird der Teradata ODBC-Treiber für Windows Version 16.20.00 verwendet.
- Sie haben die folgenden Datenbankinformationen zur Verfügung: Host, Benutzername und Passwort.

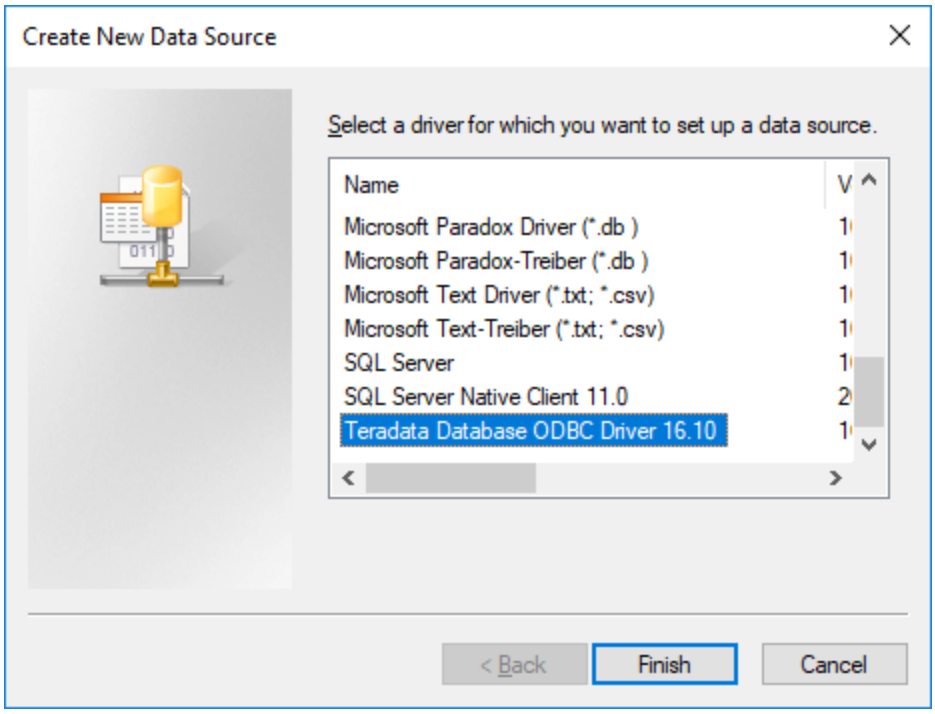
So stellen Sie über ODBC eine Verbindung zu Teradata her:

1. Drücken Sie die **Windows**-Taste, beginnen Sie mit der Eingabe von "ODBC" und wählen Sie aus der Liste der Vorschläge **ODBC Datenquellen einrichten (32-Bit)** . Wenn Sie einen 64-Bit-ODBC-Treiber haben, wählen Sie **ODBC Datenquellen einrichten (64-Bit)** und verwenden Sie in den nachfolgenden Schritten die 64-Bit-Version von UModel.

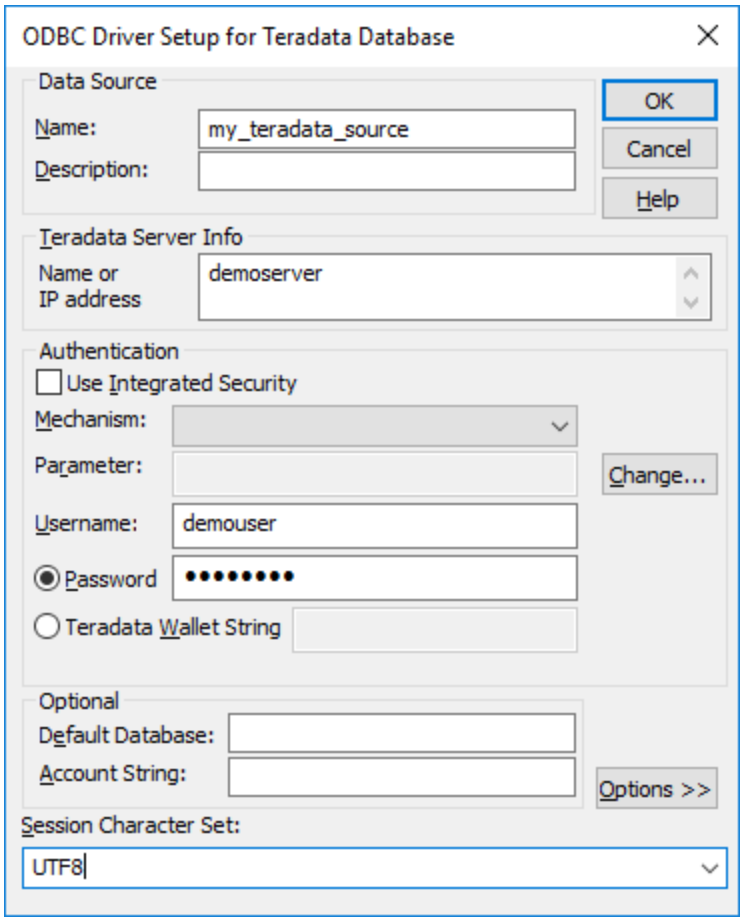


2. Klicken Sie auf das Register **System DSN** und anschließend auf **Hinzufügen**.

© 2016-2022 Altova GmbH

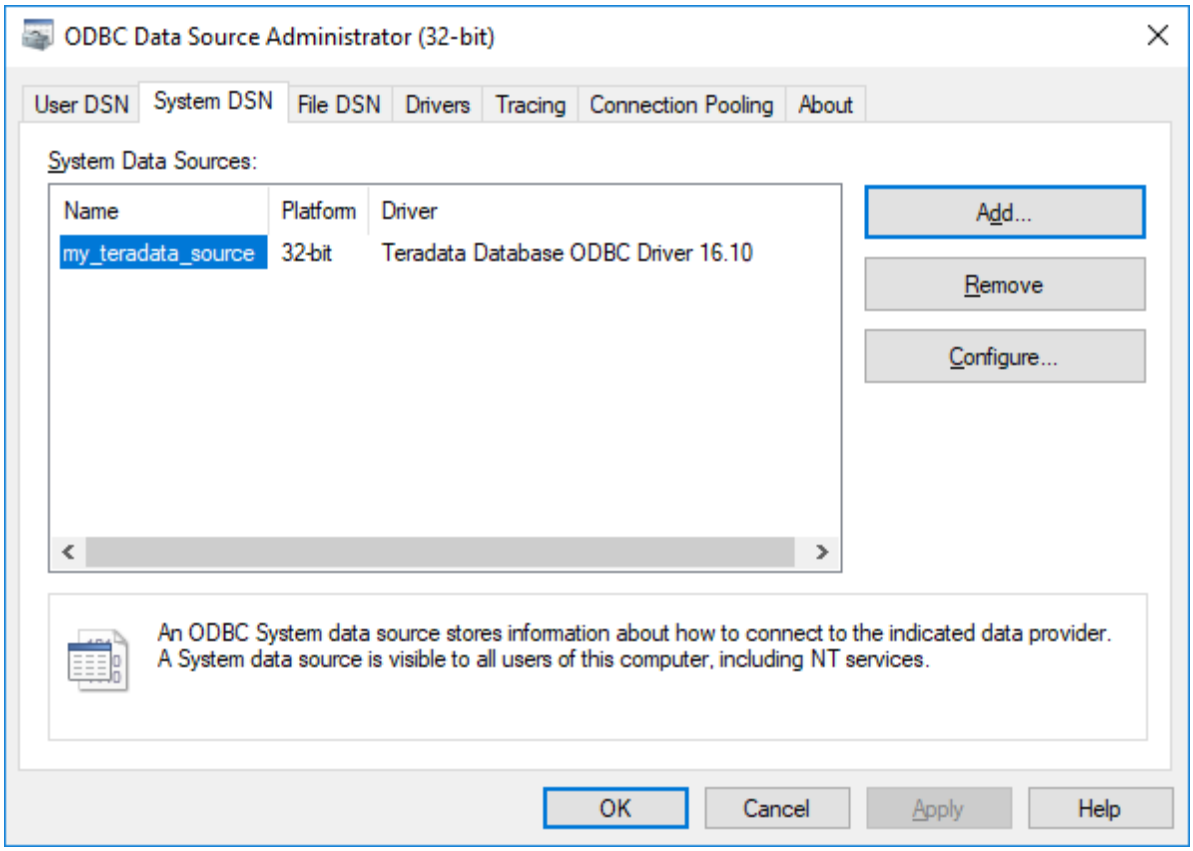


3. Wählen Sie **Teradata Database ODBC Driver** aus und klicken Sie auf **Fertig stellen**.



4. Geben Sie einen Namen und optional eine Beschreibung für diese ODBC-Datenquelle ein. Geben Sie außerdem die Anmeldeinformationen für die Datenbankverbindung ein (Datenbankserver, Benutzer, Passwort) und wählen Sie optional eine Datenbank aus.
5. Klicken Sie auf **OK**. Die Datenquelle wird nun in der Liste angezeigt.

© 2016-2022 Altova GmbH



6. Starten Sie UModel und anschließend den Datenbankverbindungsassistenten[578].
7. Klicken Sie auf **ODBC-Verbindungen**.



8. Klicken Sie auf **System DSN,** wählen Sie die zuvor erstellte Datenquelle aus und klicken Sie auf **Verbinden**.

Anmerkung: Wenn Sie die folgende Fehlermeldung erhalten: "Der Treiber hat eine ungültige SQL_DRIVER_ODBC_VER: 03.80 zurückgegeben (oder ...konnte nicht zurückgegeben werden)", überprüfen Sie, ob der Pfad zum ODBC-Client (z.B. **C:\Programme\Teradata\Client\16.10\bin**, falls unter diesem Pfad installiert) in der PATH-Umgebungsvariablen Ihres Systems vorhanden ist. Fügen Sie den Pfad manuell hinzu, falls der Pfad fehlt.

© 2016-2022 Altova GmbH

11 Austausch von Metadaten zwischen XMI und XML

Altova Website: Austausch von UModel-Projekten über XMI

Sie können UModel-Projekte in XML Metadata Interchange (XMI)-Dateien exportieren und XMI-Dateien als UModel-Projekte importieren. Dies ermöglicht die Interoperabilität mit anderen UML-Tools, die XMI unterstützen. Die folgenden XMI-Versionen werden unterstützt:

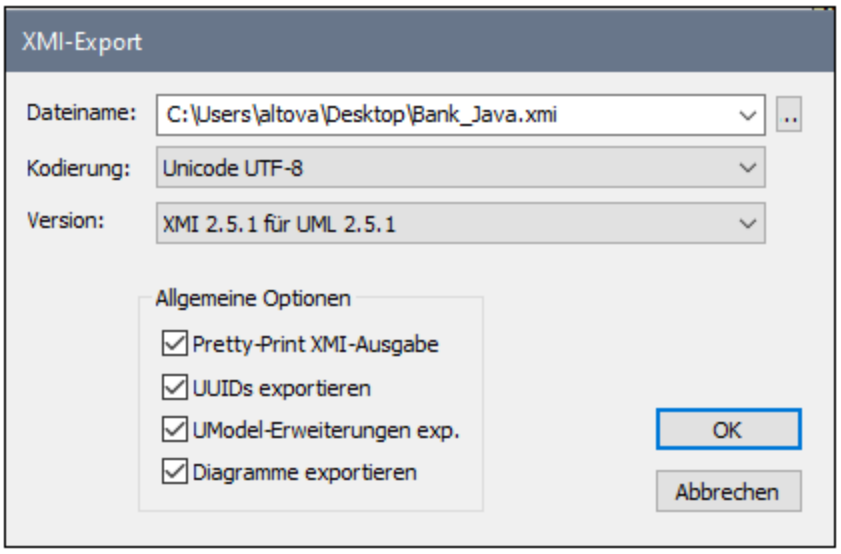
- XMI 2.1 für UML 2.0
- XMI 2.1 für UML 2.1.2
- XMI 2.1 für UML 2.2
- XMI 2.1 für UML 2.3
- XMI 2.4.1 für UML 2.4.1
- XMI 2.4.1 für UML 2.5
- XMI 2.5.1 für UML 2.5.1

So importieren Sie eine XMI-Datei in UModel:

- Klicken Sie im Menü **Datei** auf **Aus XMI-Datei importieren**.

So exportieren Sie ein UModel-Projekt in eine XMI-Datei:

- Klicken Sie im Menü **Datei** auf **In XMI-Datei exportieren**.



Anmerkungen:

- Beim Export inkludierte Dateien werden selbst solche, die als "Durch Referenz 176" definiert werden, ebenfalls exportiert.
- Wenn Sie beabsichtigen, generierten XMI-Code wieder in UModel zu importieren, stellen Sie sicher, dass Sie das Kontrollkästchen **UModel-Erweiterungen exportieren** aktivieren.

Im nachstehenden Abschnitt werden die Optionen, die beim Export von Projekten in XMI zur Verfügung stehen, beschrieben.

Pretty-Print XMI-Ausgabe

Wenn Sie diese Option aktivieren, wird die XMI-Datei mit Einrückung der XML-Tags und mit Zeilenschaltungen ausgegeben.

UUIDs exportieren

In XMI sind drei Elementidentifikationensversionen definiert: IDs, UUIDs und Labels.

- IDs sind innerhalb des XMI-Dokuments eindeutig und werden von den meisten UML-Tools unterstützt. UModel exportiert standardmäßig diese ID-Typen, d.h. keines der Kontrollkästchen muss aktiviert werden.
- UUID sind Universally Unique Identifiers und bieten eine Methode, um jedem Element eine GUID (Global Unique Identification) zuzuweisen, d.h. UUIDs sind nicht auf bestimmte XMI-Dokumente beschränkt. UUIDs werden durch Auswahl des Kontrollkästchens "UUIDs exportieren" generiert.
- UUIDs werden im Standardformat UUID/GUID gespeichert (z.B. "6B29FC40-CA47-1067-B31D-00DD010662DA", "550e8400-e29b-41d4-a716-446655440000",...)
- Labels werden von UModel nicht unterstützt.

Anmerkung: Beim XMI-Importvorgang werden automatisch beide ID-Typen unterstützt.

UModel-Erweiterungen exportieren

In XMI ist ein "Erweiterungsmechanismus" definiert, mit Hilfe dessen jede Applikation ihre toolspezifischen Erweiterungen zur UML-Spezifikation exportieren kann. Andere UML-Tools können nur die Standard-UML-Daten importieren (die UModel-Erweiterungen werden ignoriert). Diese UModel-Erweiterungsdaten stehen nur bei Import in UModel zur Verfügung.

Daten wie z.B. die Dateinamen von Klassen oder Elementfarben sind nicht Teil der UML-Spezifikation und müssen daher in XMI gelöscht oder unter "Extensions" gespeichert werden. Wenn sie als Erweiterungen exportiert wurden und wieder importiert werden, werden alle Dateinamen und Farben, wie definiert, importiert. Wenn für den Export keine Erweiterungen verwendet werden, gehen diese UModel-spezifischen Daten verloren.

Beim Import eines XMI-Dokuments wird das Format automatisch ermittelt und das Modell generiert.

Diagramme exportieren

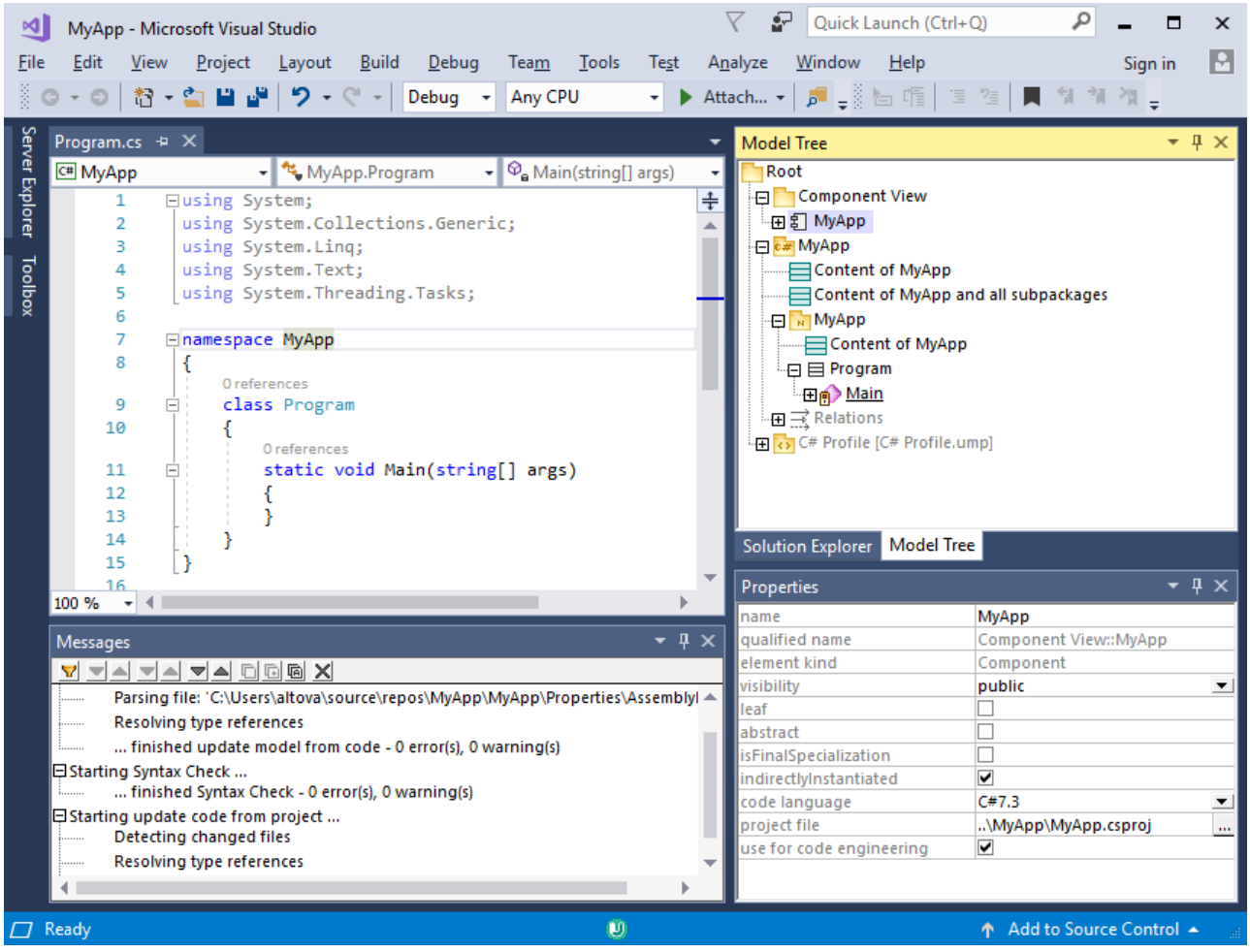
Exportiert UModel-Diagramme als "Erweiterungen" in die XMI-Datei. Die Option **UModel-Erweiterungen exportieren** muss aktiv sein, damit das Diagramm als Erweiterung gespeichert werden kann.

© 2016-2022 Altova GmbH

12 UModel Plug-in für Visual Studio

UModel 2023 kann in die Microsoft Visual Studio-Versionen 2012/2013/2015/2017/2019/2022 integriert werden. Auf diese Art steht Ihnen das Beste aus beiden Welten zur Verfügung: die Modellierungsfunktionen von UModel und die Entwicklungsumgebung von Visual Studio.

Einer der wichtigsten Vorteile bei der Verwendung von UModel als Visual Studio Plug-in ist die automatische Synchronisierung zwischen dem C#- oder VB.NET-Code und dem UML-Modell. Wenn Sie daher an Ihrem Code in Visual Studio Änderungen vornehmen, werden diese automatisch im Modell übernommen. Wenn Sie umgekehrt Änderungen am Modell vornehmen (z.B. durch Bearbeitung von Klassendiagrammen), werden diese auch im Code übernommen. Bei Bedarf können sie die automatische Synchronisierung auch deaktivieren und Code und Modell (bidirektional) manuell miteinander synchronisieren.



Visual Studio 2017-Beispielprojekt mit UModel Plug-in-Unterstützung

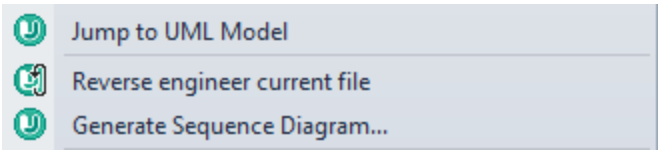
Das UModel Plug-in für Visual Studio weist im Vergleich zur Standalone Edition von UModel folgende Unterschiede auf:

- Automatische, bidirektionale Synchronisierung zwischen UModel-Modell und Projektcode (siehe Synchronisieren von Modell und Code 673).

- In Visual Studio 2019 können die Funktionalitäten von UModel über das Menü "Extensions" aufgerufen werden. In älteren Versionen von Visual Studio stehen die UModel-Funktionalitäten in den folgenden Menüs zur Verfügung:

Datei	Enthält Menübefehle aus UModel und Visual Studio.
Bearbeiten	Enthält Menübefehle aus UModel und Visual Studio.
Ansicht	Die UModel-spezifischen Befehle sind unter **Ansicht | UModel** gruppiert.
Projekt	Die UModel-spezifischen Befehle sind unter **Projekt | UModel** gruppiert.
Layout	Identisch mit der Standalone Edition von UModel.
Extras	Enthält Menübefehle aus UModel und Visual Studio. Die UModel-Optionen stehen unter **Extras | UModel-Optionen** zur Verfügung.
Hilfe	Die UModel-Hilfe steht unter **Hilfe | UModel-Hilfe** zur Verfügung.

- Wenn sich der Cursor im Visual Studio Code-Editor befindet, stehen die folgenden neuen Kontextmenüelemente (dort, wo diese Befehle angewendet werden können) zur Verfügung:
  - Springe zu UML-Modell
  - Reverse Engineering-Vorgang an aktueller Datei ausführen
  - Sequenzdiagramm generieren...



  Wenn sich der Cursor allerdings in einem Element im Fenster "Modell-Struktur" befindet, steht das Kontextmenü **Springe zu Code** (dort, wo dieser Befehl angewendet werden kann) zur Verfügung.
- Bei Ausführung von UModel als Visual Studio Plug-in stehen Ihnen die Versionskontrollfunktionen von Visual Studio zur Verfügung. Die Versionskontrollbefehle aus der Standalone Edition von UModel, die über die Microsoft Source Control Plug-in API unterstützt werden, stehen nicht zur Verfügung.
- Die mit den Befehlen **UModel | Quellverzeichnis importieren** und **UModel | Quellprojekt importieren** aufgerufenen Dialogfelder bieten in der Auswahlliste "Sprache" keine Möglichkeit zur Auswahl von "C#" und "Visual Basic". Der Import vorhandener Projekte erfolgt mit Hilfe der Visual Studio-Befehle (z.B. in Versionen vor Version 2019 **Datei | Hinzufügen | Vorhandenes Projekt**).
- Der Skript-Editor (**Extras | Skript-Editor**) und die Menüoption **Extras | Symbolleisten und Fenster wiederherstellen** stehen nicht zur Verfügung.

© 2016-2022 Altova GmbH

12.1 Installieren des UModel Plug-in für Visual Studio

Um das UModel Plugin für Visual Studio zu installieren, gehen Sie folgendermaßen vor:

1. Installieren Sie Microsoft Visual Studio 2012/2013/2015/2017/2019/2022. Beachten Sie, dass Visual Studio ab Visual Studio 2022 nur als 64-Bit-Applikation zur Verfügung steht.
2. Installieren Sie UModel (Enterprise oder Professional Edition). Wenn Sie Visual Studio 2022+ installiert haben, müssen Sie die 64-Bit-Version von UModel installieren.
3. Downloaden Sie das UModel Integrationspaket für Microsoft Visual Studio. Dieses Paket steht auf der UModel (Enterprise und Professional Edition) Download-Seite unter www.altova.com zur Verfügung.

Nach Installation des Integrationspakets können Sie UModel in der Visual Studio-Umgebung verwenden.

Achtung

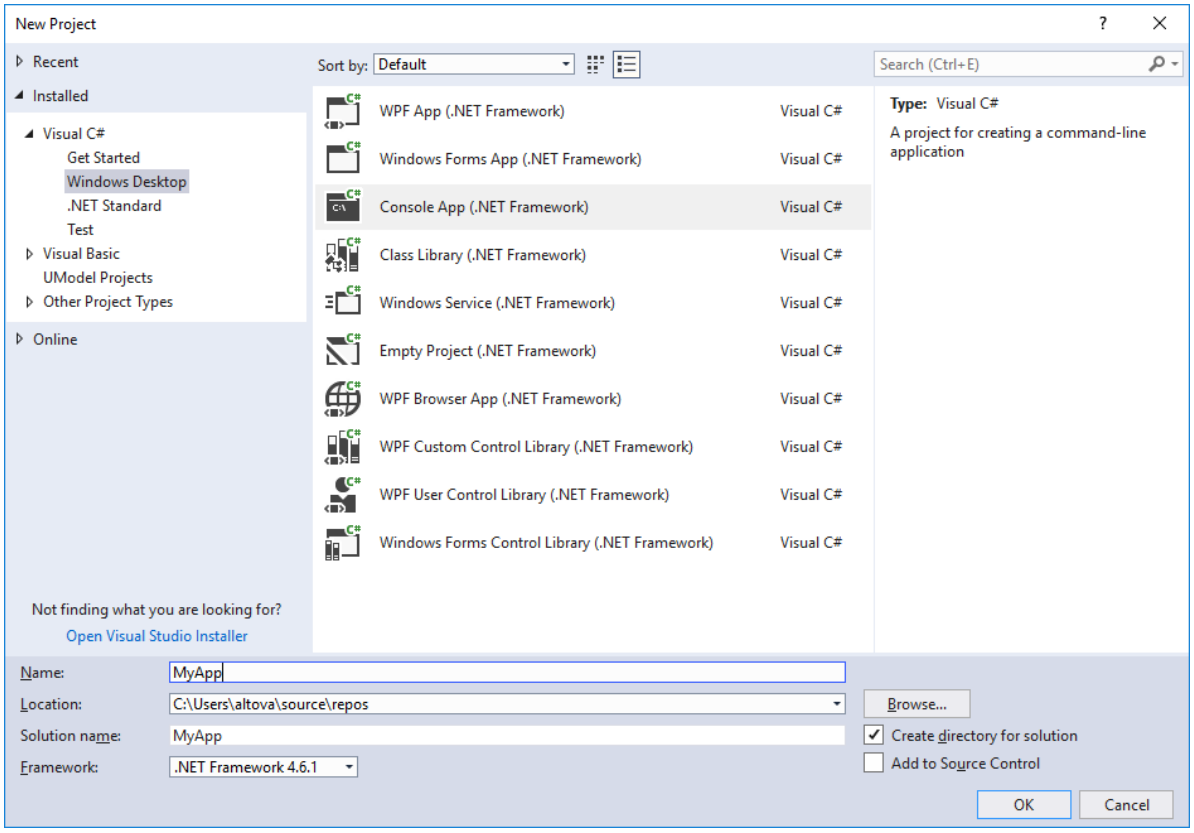
Sie müssen das richtige Integrationspaket für Ihre jeweilige UModel Version (die aktuelle Version ist 2023) verwenden. Das Integrationspaket ist nicht editionsspezifisch und daher sowohl für die Enterprise als auch die Professional Edition verwendet werden.

12.2 Hinzufügen von UModel-Unterstützung zu Visual Studio-Projekten

Wenn Sie UModel-Unterstützung zu einem neuen oder vorhandenen Visual Studio-Projekt hinzufügen, können Sie die automatische Synchronisierung zwischen Ihrem Visual Studio-Projekt und dem UModel-Modell einrichten. Eine Visual Studio-Projektmappe kann immer nur jeweils ein UModel-Projekt enthalten.

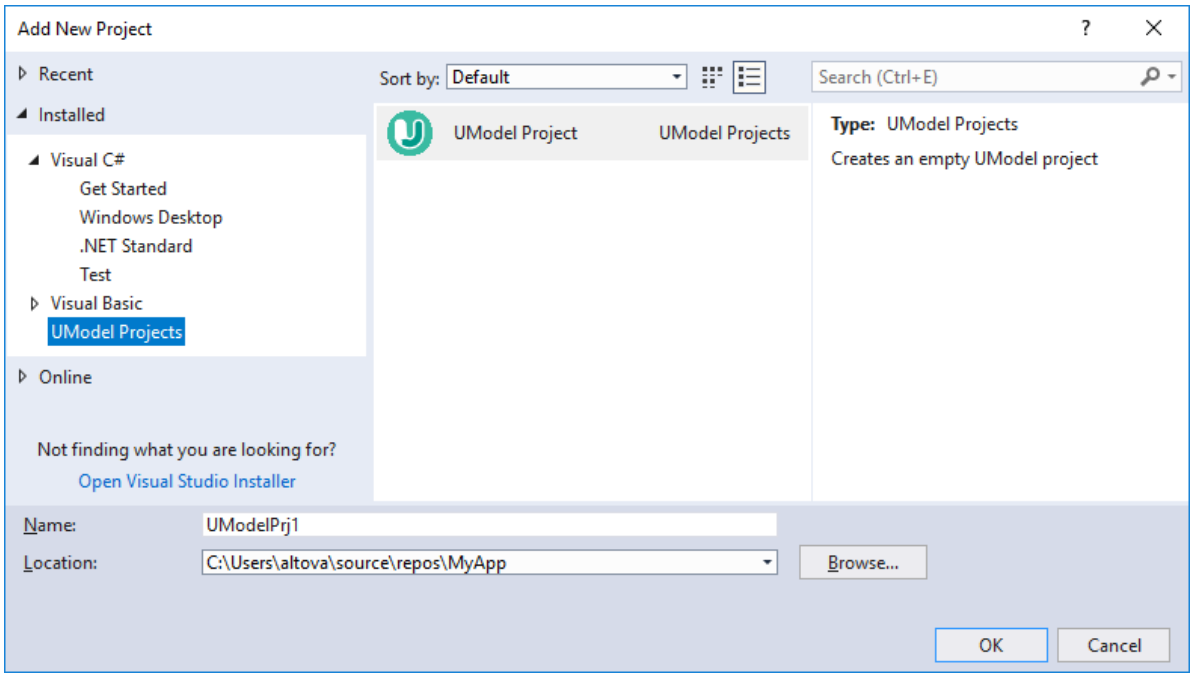
So fügen Sie UModel-Unterstützung zu einem Visual Studio-Projekt hinzu:

1. Erstellen Sie ein neues Visual Studio-Projekt oder öffnen Sie ein vorhandenes. (In diesem Beispiel wurde ein neues C#-Projekt namens "MyApp" mit Visual Studio 2017 erstellt).

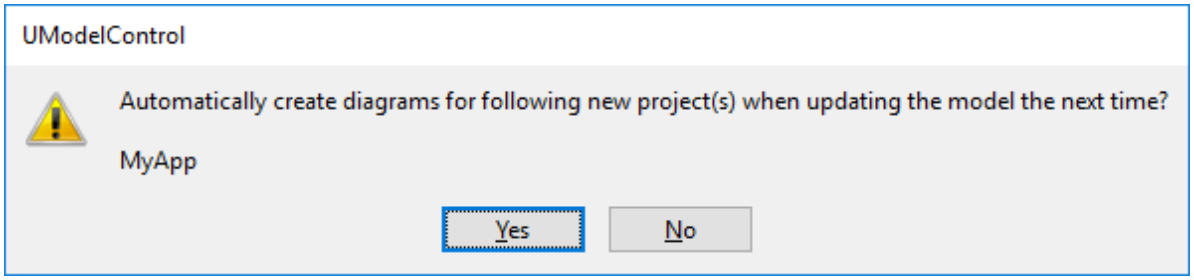


2. Klicken Sie im Menü **Datei** auf **Hinzufügen** und anschließend auf **Neues Projekt**.
3. Wählen Sie **UModel-Projekte** und klicken Sie auf **OK**.

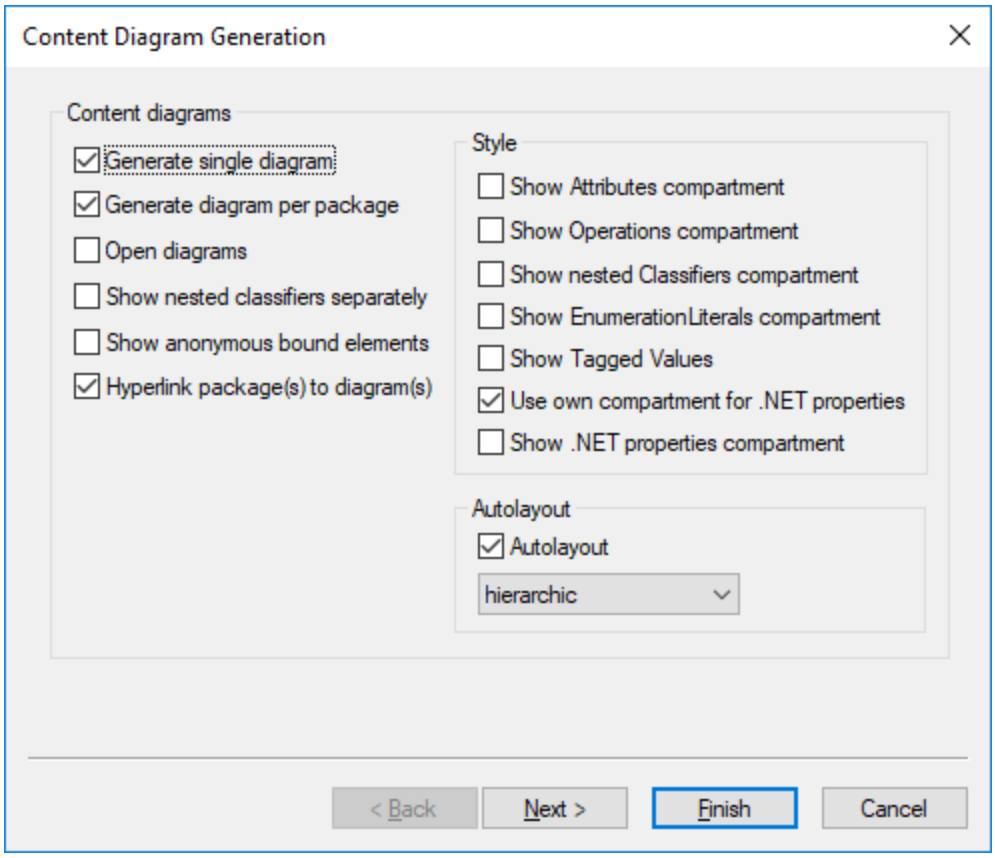
© 2016-2022 Altova GmbH



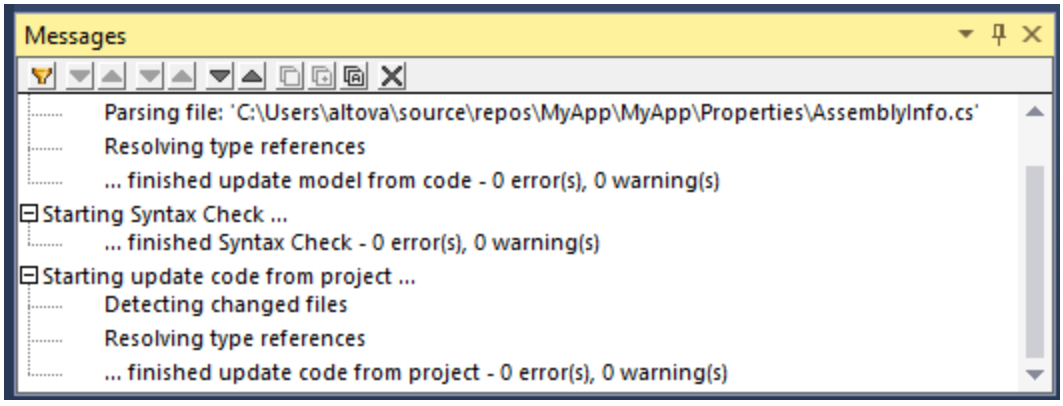
4. Wenn die Diagramme in dem auf dem Code basierenden Modell automatisch erstellt werden sollen, klicken Sie auf **Ja**, wenn Sie gefragt werden (dies ist die empfohlene Option).



5. Wenn Sie vom Assistenten aufgefordert werden, die Diagrammgenerierungsoptionen auszuwählen, aktivieren Sie die gewünschten Optionen und klicken Sie auf **Fertig stellen**. Diese Schritte sind dieselben wie in der Standalone Edition von UModel.



Nachdem Sie auf **Fertig stellen** geklickt haben, startet UModel den Synchronisierungsvorgang und blendet ein entsprechends Dialogfeld ein. Klicken Sie auf **OK**, um das Dialogfeld zu schließen. Die Synchronisierungsinformationen werden im Fenster "Meldungen" angezeigt.

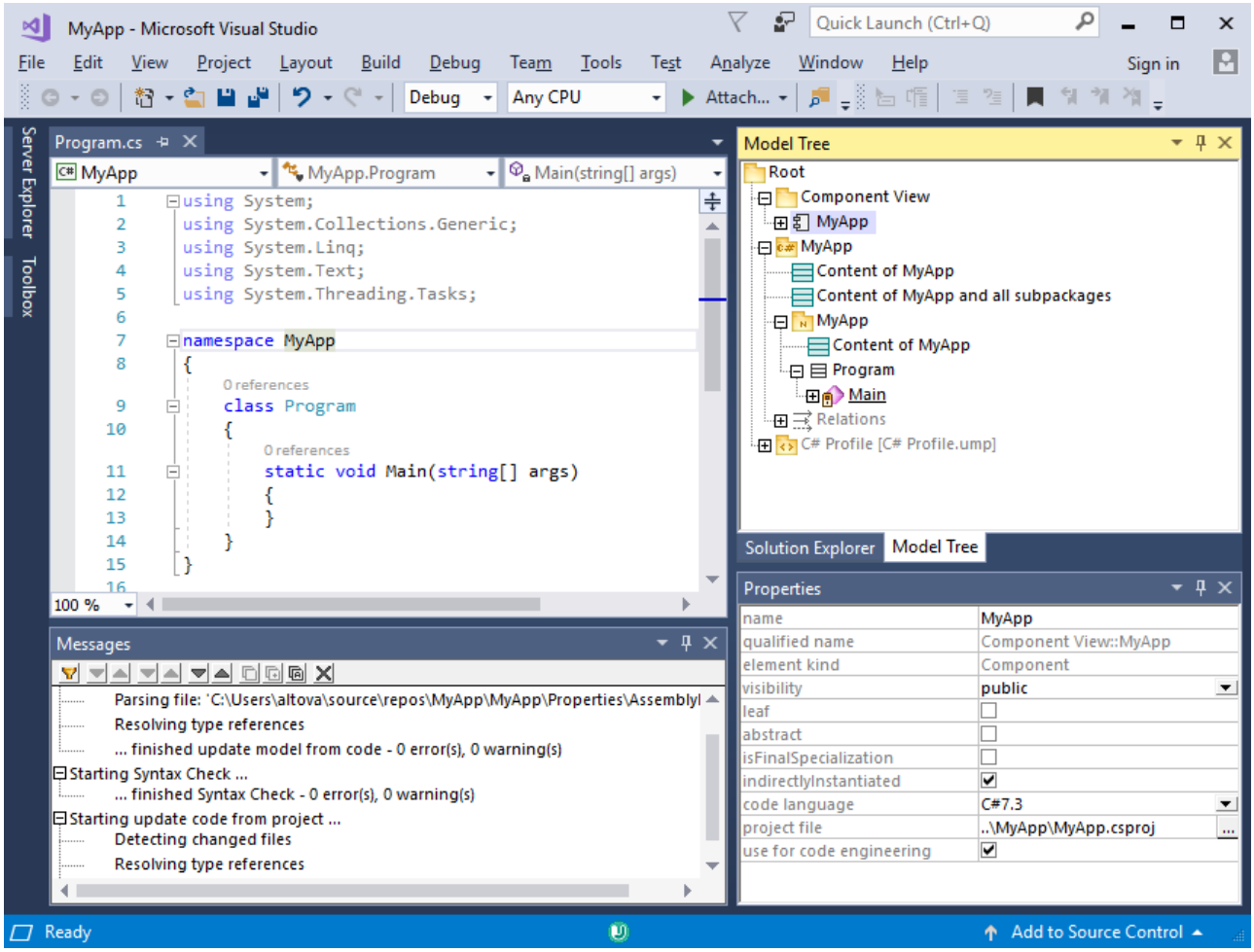


Beachten Sie, dass das Fenster **Meldungen** in Visual Studio eventuell standardmäßig nicht angezeigt wird. Sie können das Fenster (und alle anderen UModel-spezifischen Fenster) durch Auswahl des Menübefehls **Ansicht | UModel | [*Name des Fensters*]** einblenden.

Wenn Sie ein neues UModel-Projekt zu einer Visual Studio-Projektmappe hinzufügen, werden die für das Code Engineering benötigten Einstellungen (wie z.B. die Komponentenrealisierung und das C#- oder VB.NET-Profil) automatisch definiert. Um diese Einstellungen anzuzeigen, öffnen Sie die Fenster **Modell-Struktur** und

© 2016-2022 Altova GmbH

Eigenschaften (Klicken Sie im Menü Ansicht auf **UModel | Model-Struktur** bzw. **UModel | Eigenschaften**). Vergewissern Sie sich, dass Sie im Fenster "Modell-Struktur" auf die Code Engineering-Komponente geklickt haben (in diesem Fall auf "MyApp"), damit das Fenster "Eigenschaften" befüllt wird.



12.3 Laden/Deaktivieren von UModel-Projekten

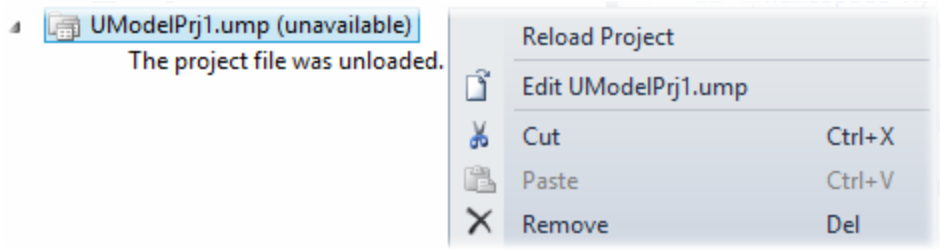
Nachdem Sie ein UModel-Projekt zu einer Visual Studio-Projektmappe hinzugefügt haben, wird es zusammen mit anderen Projekten, die Teil der Projektmappe bilden, im **Projektmappen-Explorer** von Visual Studio angezeigt. Bei Bedarf können Sie das UModel-Projekt vorübergehend aus der Projektmappe entladen. Wenn ein UModel-Projekt aus der Projektmappe entladen wird, bleiben seine Dateien auf der Festplatte und im **Projektmappen-Explorer** gespeichert. Auf diese Art können Sie das Projekt später wieder in die Projektmappe laden.

So entladen Sie ein UModel-Projekt aus einer Visual Studio-Projektmappe:

1. Klicken Sie im Projektmappen-Explorer von Visual Studio auf das UModel-Projekt.
2. Klicken Sie im Menü **Projekt** auf **Projekt deaktivieren**.

So laden Sie das UModel-Projekt wieder in der Projektmappe:

- Klicken Sie mit der rechten Maustaste im Projektmappen-Explorer auf das Projekt und klicken Sie auf **Projekt neu laden.**



So entfernen Sie das UModel-Projekt aus der Visual Studio-Projektmappe:

- Entladen Sie das Projekt, wie oben gezeigt.
- Klicken Sie mit der rechten Maustaste im Projektmappen-Explorer auf das Projekt und klicken Sie auf **Entfernen**.

© 2016-2022 Altova GmbH

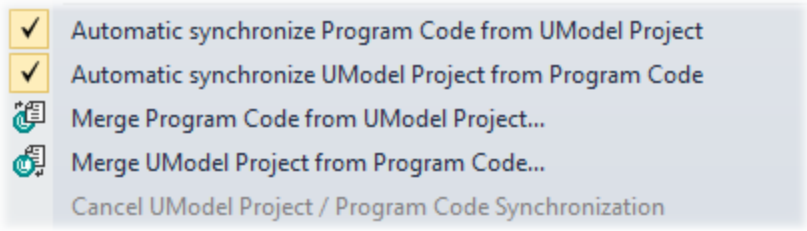
12.4 Synchronisieren von Modell und Code

Die Synchronisierung zwischen der UModel .ump-Datei (dem Modell) und dem C#- oder VB.NET-code kann manuell oder automatisch erfolgen.

Die automatische Synchronisierung wird durchgeführt, sobald Sie UModel-Unterstützung zu ihrem Visual Studio-Projekt hinzugefügt haben (siehe Hinzufügen von UModel-Unterstützung zu Visual Studio-Projekten 668). Bei der automatischen Synchronisierung wird von Ihnen neu bearbeiteter Code vom UModel Plug-in für Visual Studio geparst und das Modell wird anhand dieses Codes aktualisiert. Wenn Sie umgekehrt Änderungen am Modell vornehmen (z.B. durch Bearbeiten eines Diagramms) wird der Code entsprechend aktualisiert. Die manuelle Synchronisierung wird hingegen, wie unten gezeigt, nur bei Bedarf gestartet.

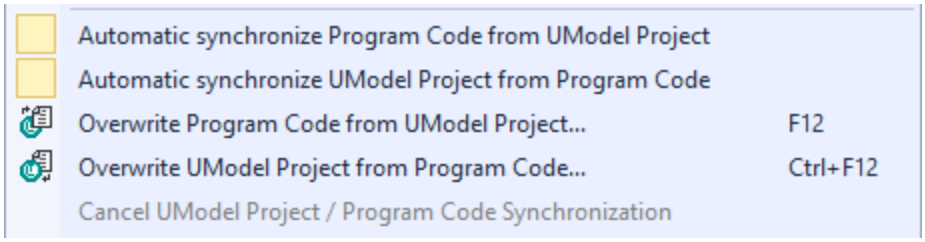
Sowohl bei der automatischen als auch bei der manuellen Synchronisierung werden die Änderungen im Bulk für das gesamte Projekt aktualisiert. Wenn UModel als Visual Studio-Plug-in ausgeführt wird, steht die Option zum Zusammenführen oder Aktualisieren einer einzigen Klasse in der Modell-Struktur nicht zur Verfügung.

Die Befehle zur Durchführung der automatischen oder manuellen Synchronisierung stehen im Menü **Projekt | UModel-Projekt** zur Verfügung:



Menübefehle zur Codesynchronisierung (Visual Studio 2010)

In neueren Versionen von Visual Studio ausgewählte Einträge sehen etwas anders aus.



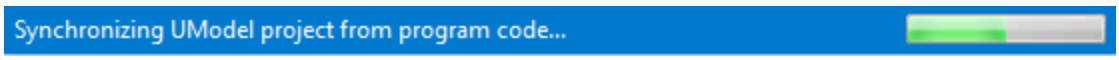
Menübefehle zur Codesynchronisierung (Visual Studio 2017)

Die einzelnen Befehle haben die folgende Bedeutung:

Programmcode anhand von UModel-Projekt automatisch synchronisieren	Diese Menüoption ist standardmäßig aktiv, d.h. die Synchronisierung des Modells anhand von Code wird automatisch durchgeführt. Um die automatische Synchronisierung zu aktivieren bzw. zu deaktivieren, deaktivieren Sie die Option.
UModel-Projekt anhand von Programmcode automatisch	Wie oben, doch im umgekehrter Richtung (Code wird im Modell übernommen).

synchronisieren	
Merge Programmcode aus UModel-Projekt	Aktualisiert den Programmcode anhand der im UModel-Projekt vorgenommenen Änderungen (dieselbe Funktionalität wie in der Standalone-Version). Der Name dieses Befehls ändert sich in **Überschreibe Programmcode aus UModel-Projekt**, wenn Sie diese Option unter **Projekt | UModel Projekt | Synchronisierungseinstellungen** ausgewählt haben.
Merge UModel-Projekt aus Programmcode	Aktualisiert das UModel-Projekt anhand der im Programmcode vorgenommenen Änderungen (dieselbe Funktionalität wie in der Standalone-Version). Der Name dieses Befehls ändert sich in **Überschreibe UModel-Projekt aus Programmcode**, wenn Sie diese Option unter **Projekt | UModel Projekt | Synchronisierungseinstellungen** ausgewählt haben.
UModel-Projekt- / Programmcodesynchronisierung abbrechen	Dient zum Abbrechen einer laufenden Synchronisierung. Wenn gerade keine Synchronisierung im Gang ist, ist diese Option deaktiviert.

Während der Synchronisierung wird der Fortschritt der Operation in der Visual Studio-Statusleiste angezeigt, z.B.



In den folgenden Fällen kann keine Synchronisierung zwischen Code und Modell durchgeführt werden:

- Code kann nicht geparst werden
- Beim letzten Reverse Engineering oder Forward Engineering ist ein Fehler aufgetreten
- Bei der Syntaxüberprüfung in UModel wird ein Fehler gefunden.

In solchen Fällen werden die Fehlerinformationen im Fenster **Meldungen** angezeigt. Um die Quelldatei, die den Fehler enthält, zu öffnen, klicken Sie im Fenster **Meldungen** auf die entsprechende Zeile. Der Cursor wird in die Zeile, die den Fehler enthält, positioniert.

Einschränkungen bei der automatischen Synchronisierung

Bei einigen Änderungen am C#- und VB.NET-Code wird kein Visual Studio-Ereignis ausgelöst. Diese Änderungen werden daher nicht automatisch in UModel aktualisiert. In diesen Fällen können Sie entweder eine manuelle Synchronisierung durchführen oder eine andere Änderung vornehmen, aufgrund welcher die Quelldatei aktualisiert wird. Eine manuelle Synchronisierung muss durchgeführt werden, wenn Sie die folgenden Elemente hinzufügen oder ändern:

- Standardwerte für Attribute
- Standardwerte für Operationsparameter
- Vorlagenparameter
- Vorlagen-Bindings
- Summary-Abschnitt für alle Elemente

© 2016-2022 Altova GmbH

- Remark-Abschnitt für alle Elemente
- Alle Änderungen in Method Bodies

Beachten Sie, dass die automatische Codesynchronisierung bei Änderung eines der oben angeführten Modellierungselementen ganz normal durchgeführt wird. Bei der automatischen Synchronisierung des Codes anhand des Modells gibt es keine Einschränkungen.

Um das Modell manuell anhand des Codes zu synchronisieren, klicken Sie im Code-Editor mit der rechten Maustaste auf die Quellcodedatei und wählen Sie im Kontextmenü den Befehl **Reverse Engineering-Vorgang an aktueller Datei ausführen**.

Wenn Ihr UModel-Projekt das Sprachprofil für Java enthält, so ist die automatische Synchronisierung für dieses Projekt in Visual Studio automatisch deaktiviert und es wird ein entsprechendes Meldungsfeld angezeigt. Solche Projekte müssen (mit den Befehle **UModel | Merge Programmcode aus UModel-Projektect** und **UModel | Merge UModel-Projekt aus Programmcode**) manuell synchronisiert werden. Alternativ dazu können Sie auch das UModel-Plug-in für Eclipse verwenden (siehe UModel Plug-in für Eclipse 676).

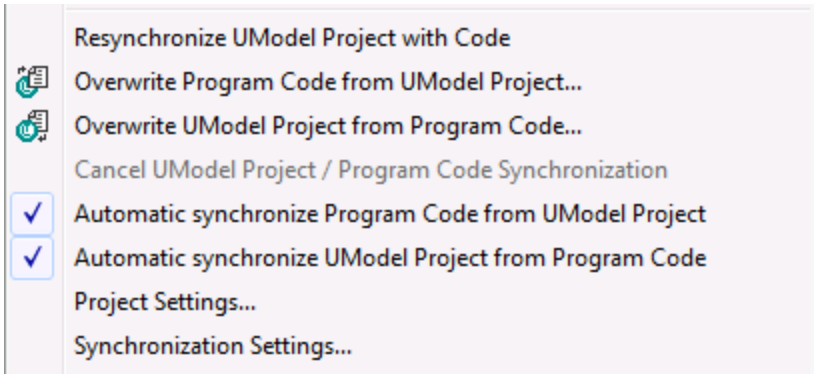
13 UModel Plug-in für Eclipse

Eclipse ist ein Open Source Framework, in das unterschiedliche Arten von Applikationen in Form von Plug-ins integriert werden. Über das UModel Plug-In für die Eclipse-Plattform können Sie UModel-Funktionalitäten direkt von Eclipse aus (Versionen 2022-09, 2022-06, 2022-03, 2021-12) aufrufen. Das Eclipse-spezifische Verhalten wird in diesem Kapitel beschrieben.

Einer der Hauptvorteile der Verwendung von UModel als Eclipse Plug-in ist die automatische Synchronisierung zwischen dem Java-Code und dem UModel-Modell. Das bedeutet, dass Änderungen, die Sie in Eclipse an Ihrem Java-Code vornehmen, automatisch im Modell übernommen werden. Wenn Sie umgekehrt Änderungen am Modell vornehmen (z.B. durch Bearbeiten von Klassendiagrammen), so werden diese im Code übernommen. Falls nötig, können Sie die automatische Synchronisierung deaktivieren und Code und Modell manuell synchronisieren (bidirektional).

Das UModel-Plug-in für Eclipse weist folgende Unterschiede zur Standalone-Version von UModel auf:

- Einige der Elemente der grafischen Benutzeroberflächen entsprechen in Eclipse den Vorgaben der Eclipse-Entwicklungsumgebung (siehe Die UModel-Perspektive 681). Wie in der Standalone-Version sind einige Elemente der Benutzeroberfläche unter Umständen deaktiviert oder stehen nicht zur Verfügung, wenn der Kontext nicht relevant ist. So hängt es z.B. davon ab, welche Art von Diagramm im Haupt-Editor aktiv ist, welche UModel-Symbolleisten-Schaltflächen zu sehen sind.
- In Eclipse steht ein **UModel** -Menü zur Verfügung. Es entspricht dem Menü **Projekt** in der Standalone-Version von UModel. Die meisten der Befehle in diesem Menü sind mit denen in der Standalone-Version identisch, doch gibt es einige neue Befehle, mit denen Sie die automatische Synchronisierung steuern können:

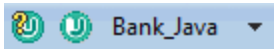


UModel-Projekt mit Code neu synchronisieren	Mit diesem Befehl können Sie die Synchronisierung zwischen dem UModel-Projekt und dem Programmcode explizit starten (Dies kann nützlich sein, wenn die letzte automatische Synchronisierung aus irgendeinem Grund fehlgeschlagen ist).
Merge Programmcode aus UModel-Projekt	Aktualisiert den Programmcode anhand der im UModel-Projekt vorgenommenen Änderungen (dieselbe Funktionalität wie in der Standalone-Version).
Merge UModel-Projekt aus Programmcode	Aktualisiert das UMode-Projekt anhand der im Programmcode vorgenommenen Änderungen (dieselbe Funktionalität wie in der Standalone-Version).

© 2016-2022 Altova GmbH

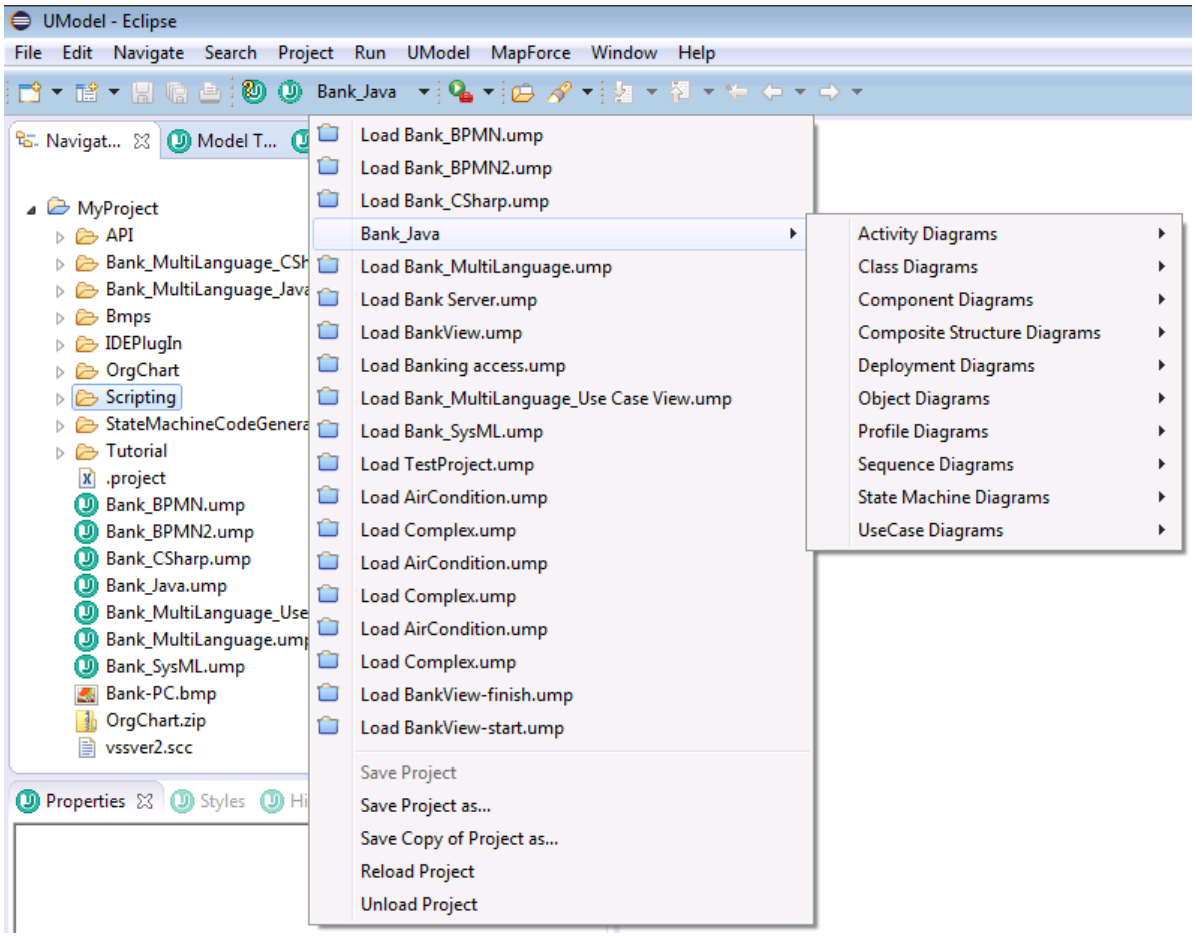
UModel-Projekt- / Programmcodesynchronisierung abbrechen	Dient zum Abbrechen einer laufenden Synchronisierung. Wenn gerade keine Synchronisierung im Gang ist, ist diese Option deaktiviert.
Programmcode anhand von UModel-Projekt automatisch synchronisieren	Diese Menüoption ist standardmäßig aktiv, d.h. die Synchronisierung des Codes anhand des Modells wird automatisch durchgeführt. Um die automatische Synchronisierung zu deaktivieren, deaktivieren Sie die Option.
UModel-Projekt anhand von Programmcode automatisch synchronisieren	Wie oben, aber in die umgekehrte Richtung (Synchronisierung des Codes anhand des Modells).

- Die in der Standalone-Version von UModel über die Microsoft Source Control Plug-in API verfügbaren Versionskontrollbefehle werden in Eclipse nicht unterstützt. Stattdessen haben Sie die Möglichkeit, Drittanbieter-Versionskontrollsysteme zu verwenden, die mit Eclipse integriert werden können.
- In den mit den Befehlen **UModel | Quellverzeichnis importieren** und **UModel | Quellprojekt importieren** aufgerufenen Dialogfeldern steht in der Auswahlliste "Sprache" die Option "Java" nicht zur Verfügung. Um Java-Quellcode in ein Eclipse-Projekt zu importieren, verwenden Sie die Eclipse-Standardbefehle (z.B. **Datei | Importieren**).
- In Eclipse steht eine neue Symbolleiste, nämlich die UModel-Symbolleiste, zur Verfügung. Sie enthält einige allgemeine sowie projektspezifische Befehle.



Über die Symbolleisten-Schaltfläche wird die Hilfedatei aufgerufen. Über die Symbolleisten-Schaltfläche wird der aktuelle Status des Code Engineering-Vorgangs angezeigt (Wenn die Schaltfläche rot angezeigt wird, weist dies auf einen Fehler hin, den Sie im Fenster "Meldungen" ansehen können). Die Dropdown-Liste in der Symbolleiste schließlich hat eine Reihe von Funktionen:

  - Sie können damit schnell eine bestimmte UModel-Projektdatei (.ump) in Eclipse laden oder entfernen. Ihr Eclipse-Projekt muss mindestens eine UModel-Projektdatei (.ump) enthalten, andernfalls ist die Dropdown-Liste deaktiviert.
  - Wenn ein UModel-Projekt geladen ist, stehen mehrere Kontextmenübefehle wie z.B. der Schnellzugriff auf eines der Diagramme des geladenen Projekts zur Verfügung:



Klicken Sie, um das Bild größer anzuzeigen

- Der Skript-Editor (**Extras | Skript-Editor**) und die Menüoption **Extras | Symbolleisten und Fenster wiederherstellen** werden nicht unterstützt.
- Die UModel-Befehle Hilfe, Support Center, Auf Updates überprüfen und Über UModel stehen im Menü **Hilfe | UModel-Hilfe** von Eclipse zur Verfügung. Auch die Versionsinformationen zum UModel Plug-in für Eclipse stehen im Eclipse-Menü zur Verfügung (wählen Sie **Hilfe | Info über Eclipse**, und klicken Sie anschließend auf das UModel-Symbol).

© 2016-2022 Altova GmbH

13.1 Installieren des UModel Plug-in für Eclipse

Voraussetzungen

- Eclipse 2022-09, 2022-06, 2022-03, 2021-12 (http://www.eclipse.org), 64-Bit.
- Java Runtime Environment (JRE) oder Java Development Kit (JDK) für die 64-Bit-Plattform.
- UModel Enterprise oder Professional Edition 64-Bit.

Anmerkung: Alle oben aufgelisteten Programme müssen die 64-Bit-Plattform haben. Die Integration mit älteren Eclipse 32-Bit-Plattformen wird nicht mehr unterstützt, funktioniert aber eventuell noch.

Mit den obigen Voraussetzungen können Sie das UModel Integrationspaket (64-Bit) installieren, um UModel in Eclipse zu integrieren. Die Integration kann entweder während der Installation des Installationspakets oder nach Installation des Integrationspakets manuell von Eclipse aus durchgeführt werden. Das UModel Integrationspaket kann von https://www.altova.com/de/components/download heruntergeladen werden.

Anmerkung: Eclipse muss während der Installation bzw. Deinstallation des UModel Integrationspakets geschlossen sein.

Integration von UModel während der Installation des Integrationspakets

Sie können UModel während der Installation des UModel-Integrationspakets in Eclipse integrieren. Gehen Sie dazu folgendermaßen vor:

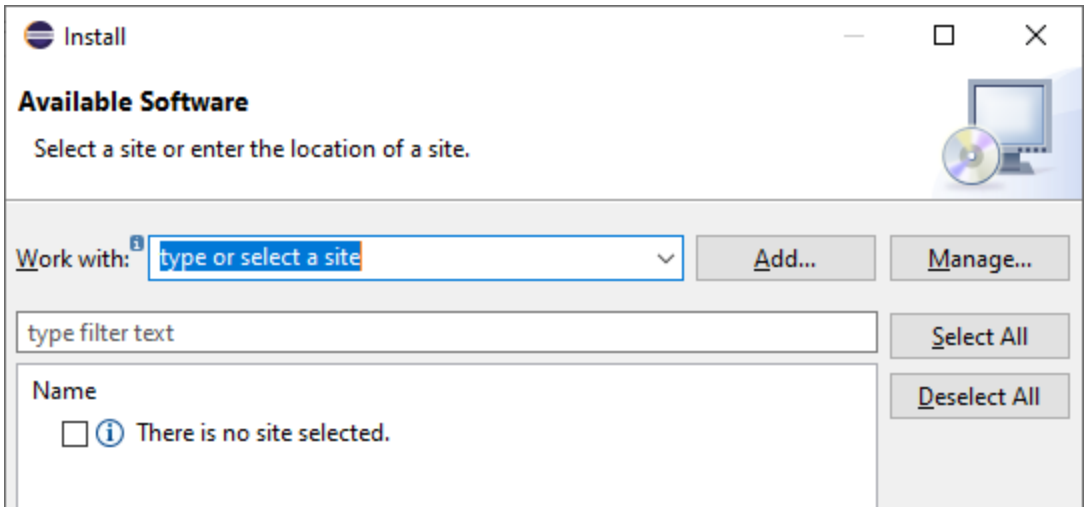
1. Starten Sie das UModelIntegrationspaket, um den Installationsassistenten aufzurufen.
2. Führen Sie mit dem Assistenten die ersten Schritte der Installation durch.
3. Aktivieren Sie beim Integrationsschritt die Option *Altova UModel Plug-in mit diesem Assistenten in Eclipse integrieren* und suchen Sie nach dem Ordner, in dem sich die ausführbare Datei von Eclipse befindet (`eclipse.exe`).
4. Klicken Sie auf **Weiter** und schließen Sie die Installation ab.

Die UModel-Perspektive und die entsprechenden Menüs stehen daraufhin beim nächsten Start von Eclipse zur Verfügung.

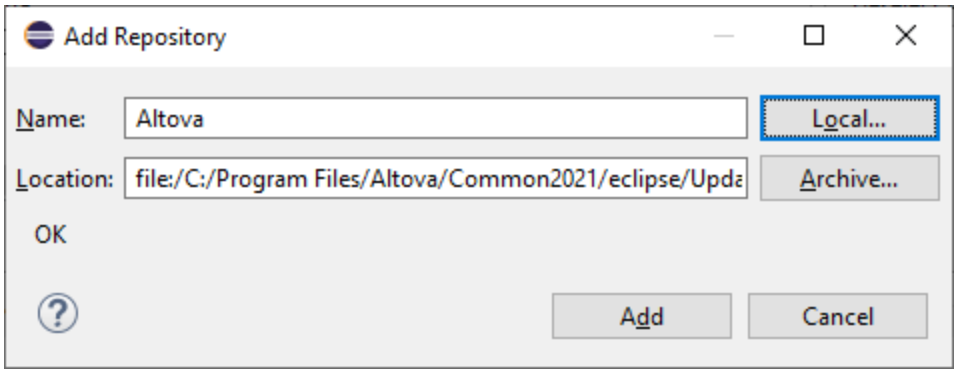
Manuelle Integration von UModel in Eclipse

Nach Installation des UModel-Integrationspakets können Sie UModel folgendermaßen manuell in Eclipse integrieren:

1. Klicken Sie in Eclipse auf den Menübefehl **Help | Install new Software**.
2. Klicken Sie im Dialogfeld "Install", das daraufhin angezeigt wird, auf die Schaltfläche **Add**.



3. Klicken Sie im Dialogfeld "Add Repository" auf **Local**. Navigieren Sie zum Ordner `C:\Programme\Altova\Common2023\eclipse\UpdateSite`, wählen Sie ihn aus und klicken Sie auf OK. Geben Sie einen Namen dafür ein (z.B. "Altova").



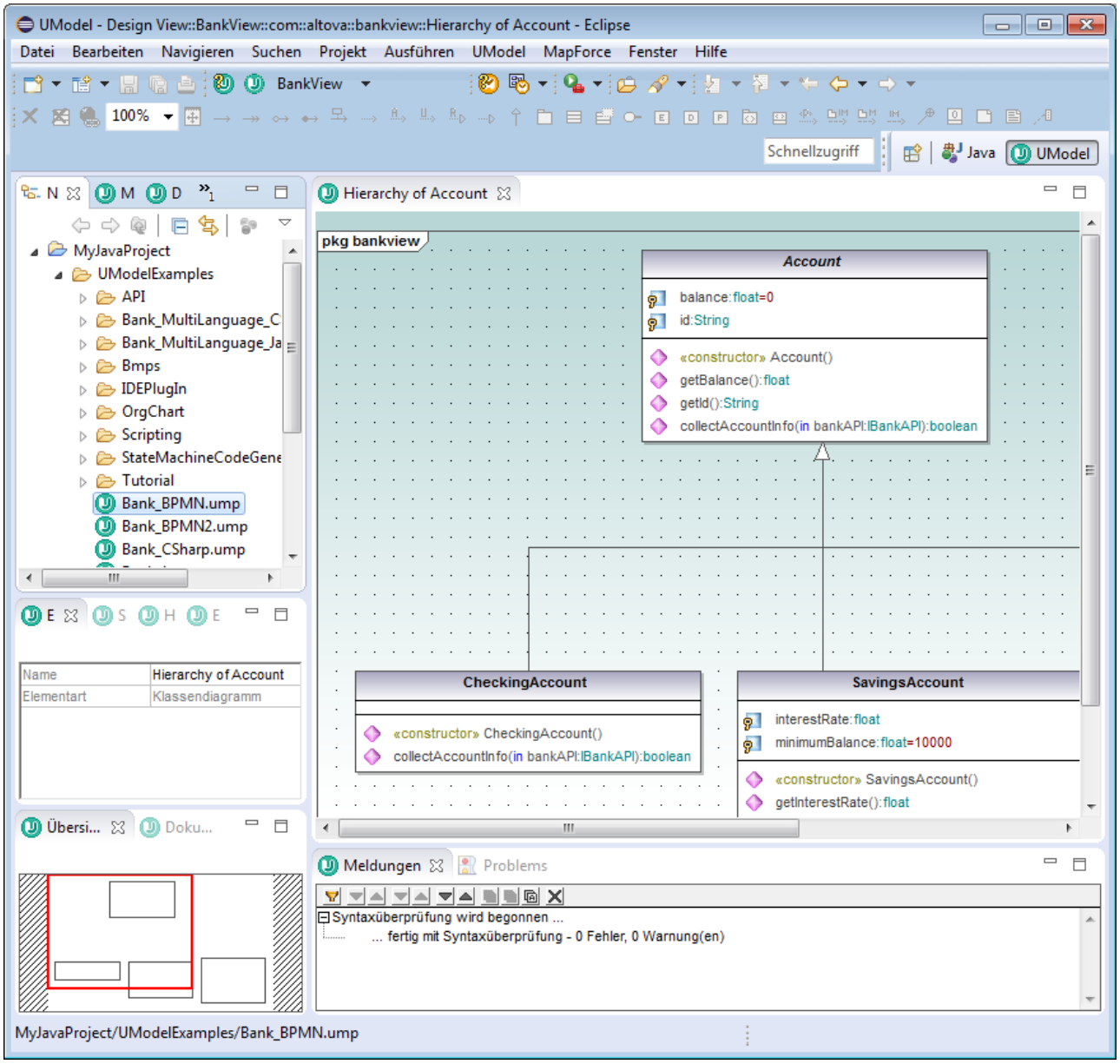
4. Wiederholen Sie die Schritte 2 bis 3 und wählen Sie diesmal den Ordner `C:\Programme\Altova\<%APPNAMESHORT%>\eclipse\UpdateSite`aus. Geben Sie einen Namen wie z.B. "Altova UModel" ein.
5. Wählen Sie im Installationsdialogfeld *Only Local Sites* aus. Wählen Sie als nächstes den "Altova category"-Ordner aus und klicken Sie auf **Next**.
6. Überprüfen Sie die zu installierenden Objekte und klicken Sie zum Fortfahren auf **Next**.
7. Aktivieren Sie das entsprechende Kontrollkästchen, um die Lizenzvereinbarung zu akzeptieren.
8. Klicken Sie anschließend auf **Finish**, um die Installation fertig zu stellen.

Anmerkung: Falls es Probleme mit dem Plug-in gibt (z.B. fehlende Symbole), starten Sie Eclipse über die Befehlszeile mit dem `-clean` Flag.

© 2016-2022 Altova GmbH

13.2 Die UModel-Perspektive

Nachdem Sie das UModel Plug-in für Eclipse installiert haben, steht in Eclipse eine neue Perspektive ("UModel") zur Verfügung. Das Layout dieser Perspektive ist der Benutzeroberfläche der Standalone Edition von UModel sehr ähnlich. Um in die UModel-Perspektive zu wechseln, klicken Sie auf **Fenster | Perspektive öffnen | Andere** und wählen Sie UModel in der Liste aus. In der Abbildung unten sehen Sie ein in Eclipse geladenes UModel-Beispielprojekt (BankView.ump), wobei die UModel-Perspektive aktiviert ist.



Die UModel-Perspektive in Eclipse ist folgendermaßen gegliedert:

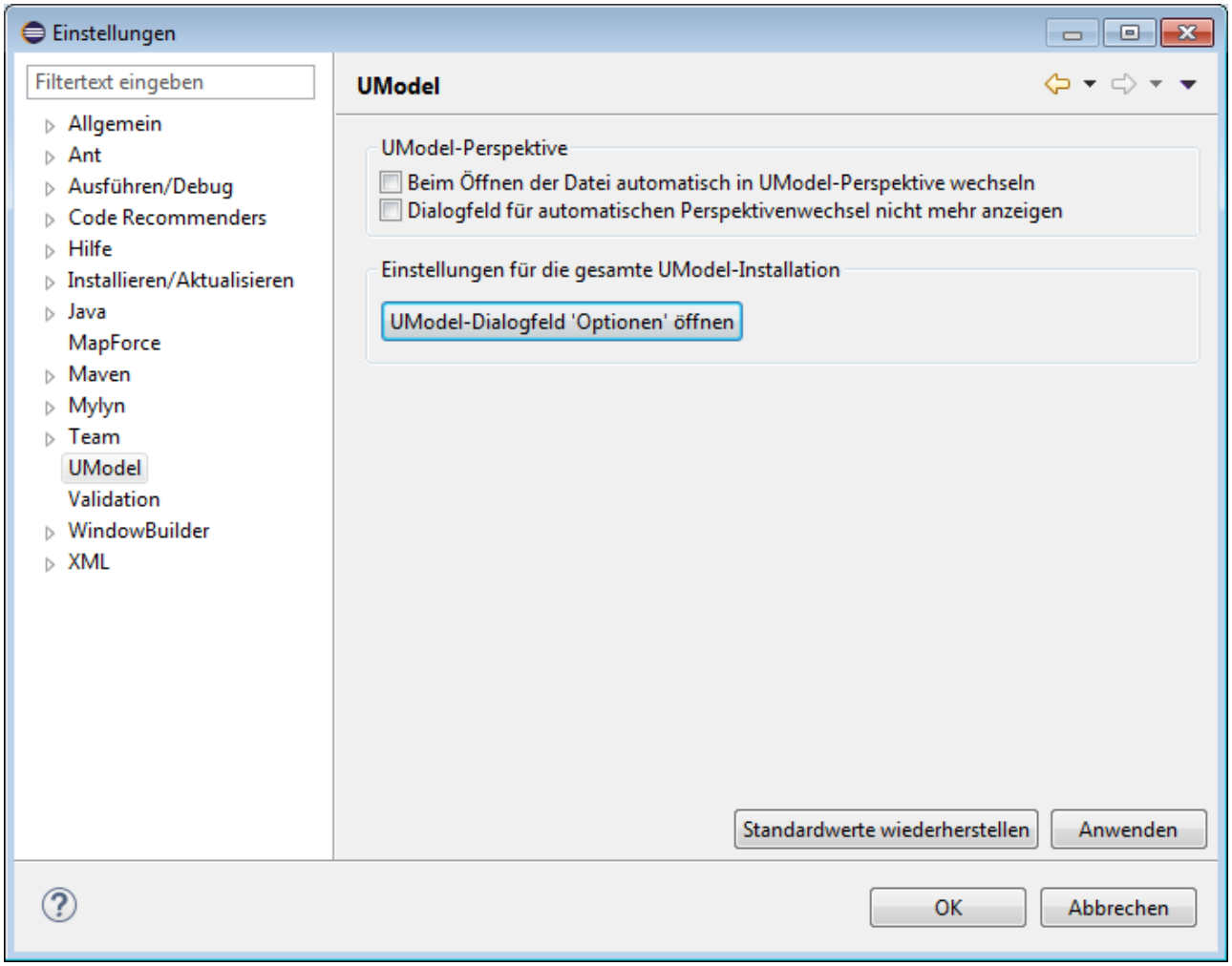
- Das als Eclipse-Editor angezeigte Diagramm-Fenster. Wenn mehrere Diagramme offen sind, werden diese wie in der Standalone-Edition in eigenen Editor-Fenstern angezeigt.
- Alle der folgenden UModel-Fenster stehen (standardmäßig links vom Haupt-Editor) in Form von Eclipse-Ansichten zur Verfügung:
  - Diagramm-Struktur
  - Favoriten
  - Eigenschaften
  - Stile
  - Hierarchie
  - Übersicht
  - Dokumentation
  - Ebene
- Auch das Fenster "Meldungen" steht (standardmäßig unterhalb des Haupt-Editor-Fensters) in Form einer Eclipse-Ansicht zur Verfügung .

Die UModel-Perspektive funktioniert wie jede andere Eclipse-Perspektive. Sie können in Eclipse jederzeit über **Fenster | Navigation | Nächste Perspektive** zur UModel-Perspektive wechseln.

So konfigurieren Sie die Einstellungen für die UModel-Perspektive:

1. Klicken Sie im Menü **Fenster** auf **Benutzervorgaben**.
2. Wählen Sie im Dialogfeld "Einstellungen" den Eintrag **UModel** aus.

© 2016-2022 Altova GmbH



Um das Aussehen der UModel-Perspektive (Sichtbarkeit der Symbolleiste, der Menüs, usw.) anzupassen, wechseln Sie in die UModel-Perspektive und wählen Sie anschließend den Menübefehl **Fenster | Perspektive anpassen**. Um die Einstellungen auf die Standardeinstellungen zurückzusetzen, wählen Sie **Fenster | Perspektive zurücksetzen**.

Um eine bestimmte Ansicht in der UModel-Perspektive anzuzeigen, wechseln Sie in die UModel-Perspektive und wählen Sie anschließend die gewünschte Ansicht aus dem Menü **Fenster | Sicht anzeigen**.

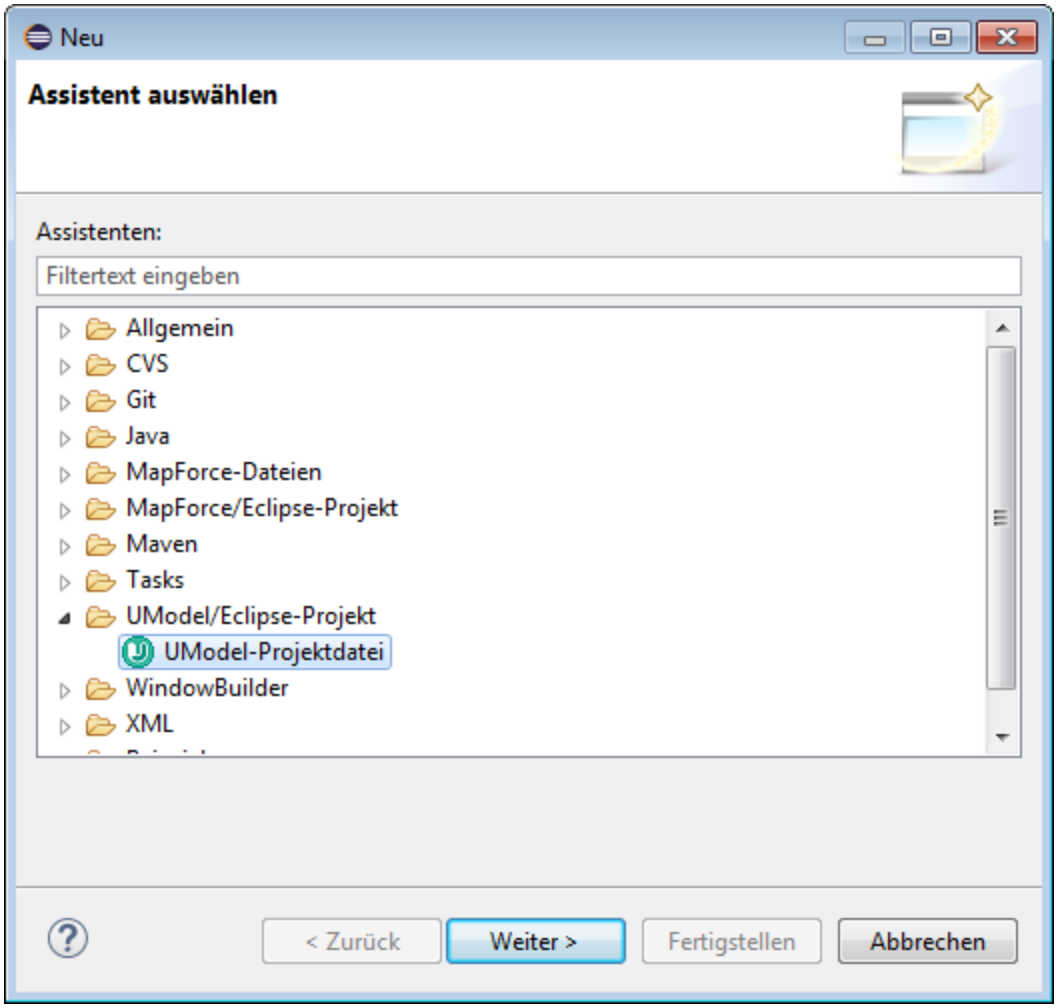
Allgemeine Informationen zu Eclipse-Perspektiven finden Sie in der Dokumentation zu Eclipse.

13.3 Hinzufügen von UModel-Unterstützung zu Eclipse-Projekten

Bevor Sie in der Eclipse-Umgebung mit UModel-Projekten (.ump-Datei) arbeiten können, müssen Sie zuerst ein Eclipse-Projekt öffnen oder erstellen. Dabei kann es sich um ein neues oder vorhandenes Java-Projekt handeln, zu dem Sie UML-Unterstützung hinzufügen möchten). In diesem Kapitel wird beschrieben, wie Sie ein neues UModel-Projekt innerhalb eines Eclipse-Projekts erstellen. Eine Anleitung zum Importieren eines vorhandenen UModel-Projekts in ein Eclipse-Projekt finden Sie unter Importieren vorhandener UModel-Projekte 686.

So fügen Sie ein UModel-Projekt zu einem Eclipse-Projekt hinzu:

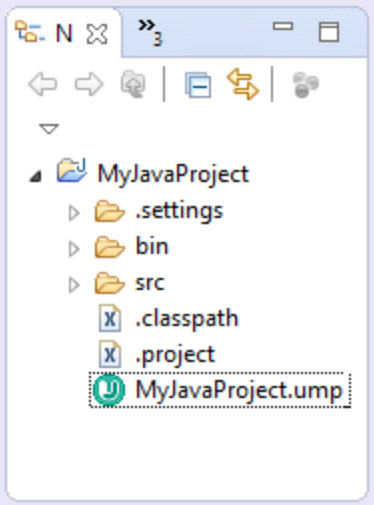
1. Erstellen Sie mit den Eclipse-Befehlen (**Datei | Neu** | **Projekt** oder **Datei | Datei öffnen**) ein neues Projekt bzw. öffnen Sie ein bestehendes.
2. Klicken Sie im Menü **Datei** auf **Neu | Andere** und wählen Sie anschließend den Typ **UModel-Projektdatei** im Dialogfeld aus.



3. Klicken Sie auf **Weiter**.

© 2016-2022 Altova GmbH

4. Wenn Sie dazu aufgefordert werden, wählen Sie einen übergeordneten Ordner für das neue UModel-Projekt auf und klicken Sie auf **Fertigstellen**. Daraufhin steht das neue UModel-Projekt in der Navigator-Ansicht unterhalb des angegebenen Ordners zur Verfügung.

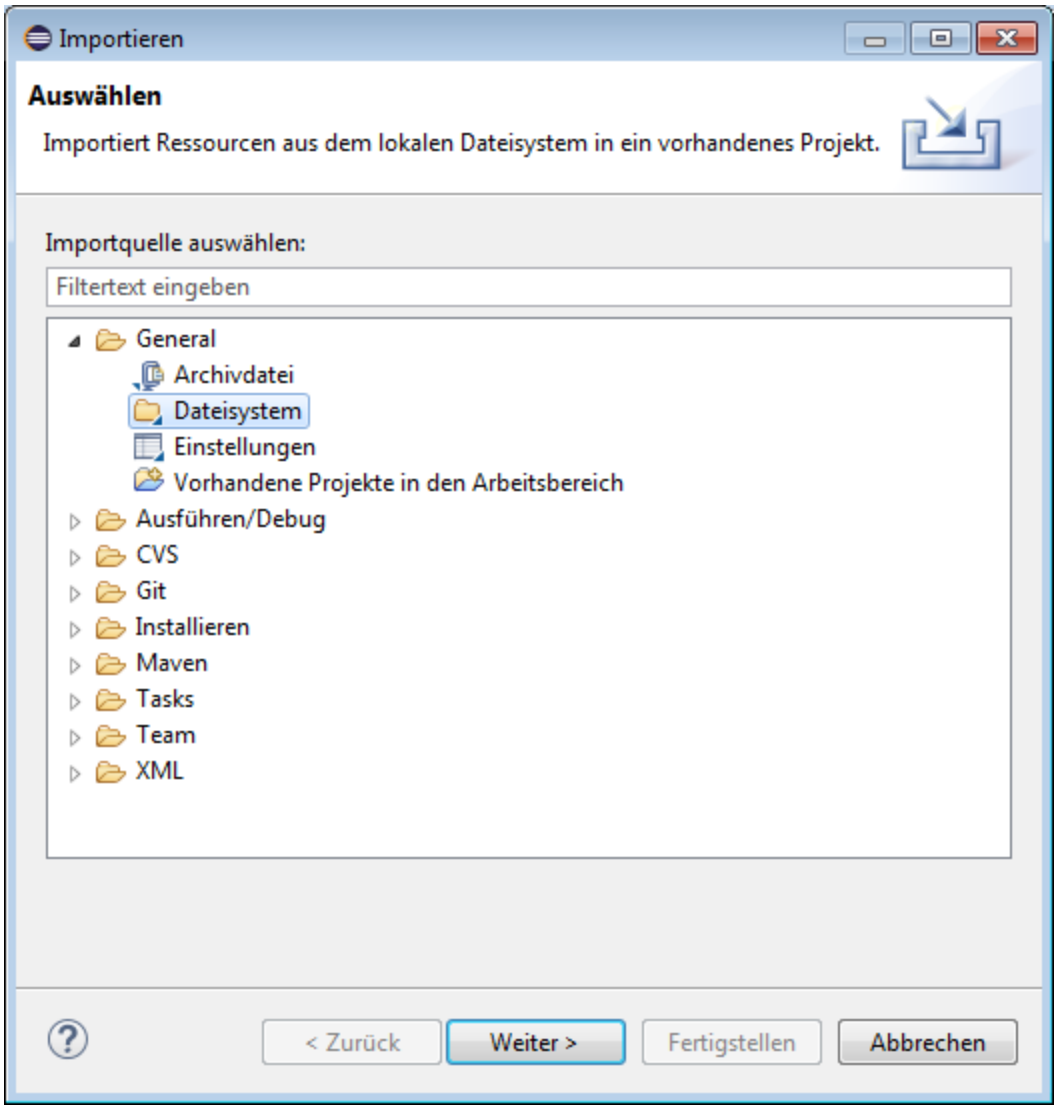


UModel-Projekte können nicht in einem Editor geöffnet werden. Um Aktionen am Projekt vorzunehmen (wie z.B. Speichern oder Laden des Projektinhalts in Eclipse), klicken Sie mit der rechten Maustaste auf die .ump-Datei und wählen Sie den gewünschten Befehl aus.

13.4 Importieren vorhandener UModel-Projekte

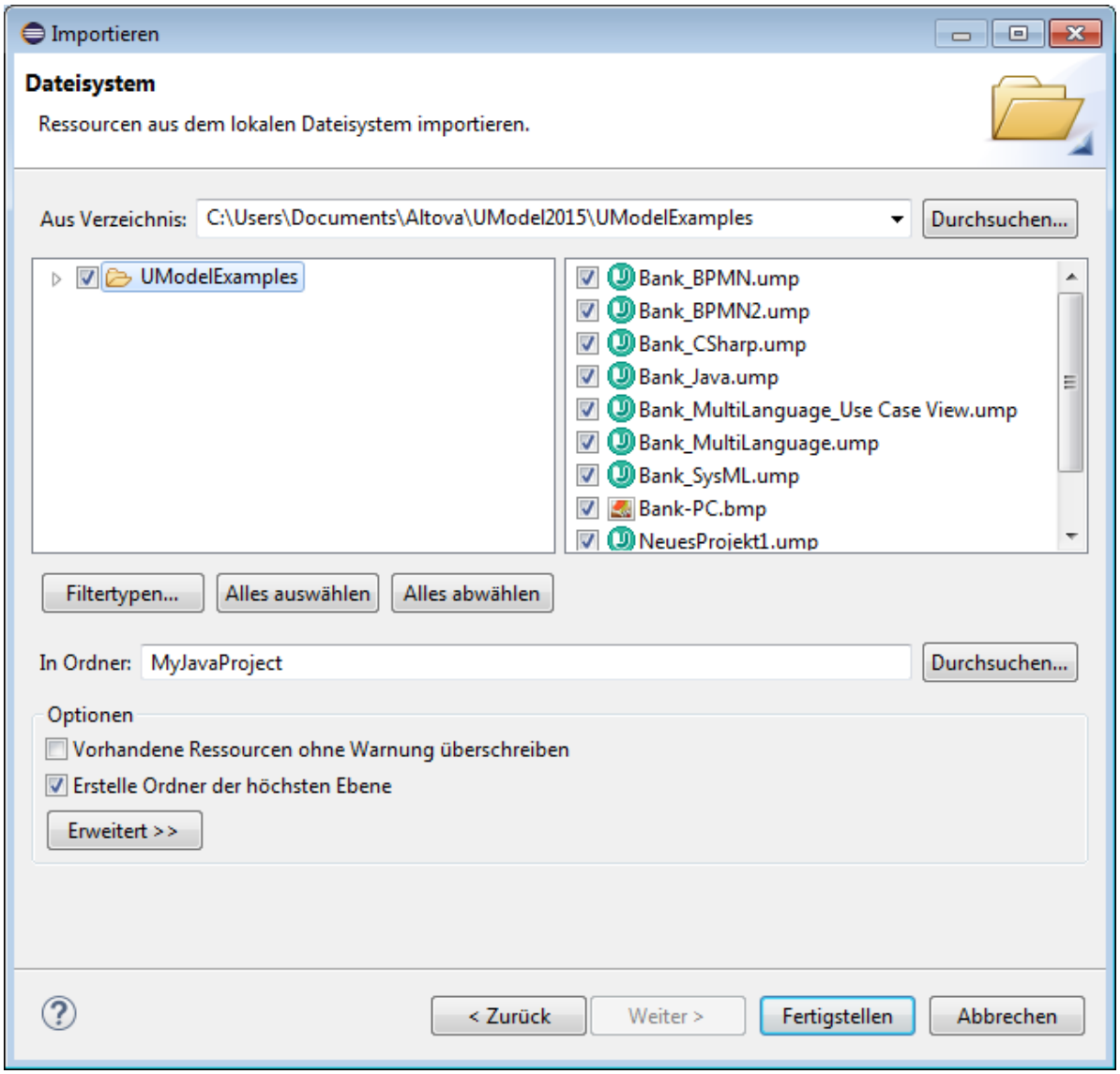
So importieren Sie vorhandene UModel-Projekte in Eclipse:

1. Erstellen Sie ein neues Eclipse-Projekt oder öffnen Sie ein vorhandenes.
2. Klicken Sie im Menü **Datei** auf **Importieren**.
3. Wählen Sie **General | Dateisystem**.



4. Klicken Sie auf **Weiter**.
5. Klicken Sie auf **Durchsuchen** und wählen Sie die zu importierenden UModel-Projektordner aus (z.B. den Ordner "UModel Examples").

© 2016-2022 Altova GmbH



6. Klicken Sie auf **Fertigstellen**.

13.5 Laden/Deaktivieren von UModel-Projekten

Nachdem Sie eine oder mehrere UModel-Projektdateien erstellt oder importiert haben, werden diese in der Navigator-Ansicht von Eclipse angezeigt. Ein Eclipse-Projekt kann zwar mehrere UModel-Projekte enthalten, doch kann immer nur ein UModel-Projekt aktiv (geladen) sein. Sie haben folgende Möglichkeiten, um ein Projekt zu laden:

- Klicken Sie mit der rechten Maustaste auf die Datei im Navigator und wählen Sie **UModel | Laden**.
- Wählen Sie in der UModel-Symbolleiste **Laden IhrenProjektnamen.ump**.

So deaktivieren Sie ein Projekt:

- Klicken Sie mit der rechten Maustaste auf die Datei im Navigator und wählen Sie **UModel | Deaktivieren**.
- Wählen Sie in der UModel-Symbolleiste den Befehl **Projektdatei deaktivieren**.

© 2016-2022 Altova GmbH

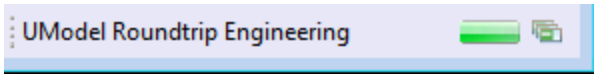
13.6 Funktionsweise der automatischen Synchronisierung

Die automatische Synchronisierung wird durchgeführt, nachdem Sie UModel-Unterstützung zu eine Java-Projekt hinzugefügt haben (siehe Hinzufügen von UModel-Unterstützung zu Eclipse-Projekten 684). Automatische Synchronisierung bedeutet, dass das UModel Plug-in für Eclipse den Code jedes Mal parst, wenn Sie den Code in der Eclipse-Umgebung bearbeiten, und das Modell aktualisiert. Wenn Sie im Modell Änderungen an einem Diagramm vornehmen, so wird umgekehrt der Code entsprechend aktualisiert.

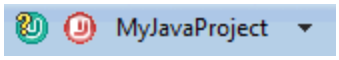
Wenn Ihr UModel-Projekt das Programmiersprachenprofil für C# oder Visual Basic enthält, so wird die automatische Synchronisierung für dieses Projekt automatisch deaktiviert und eine entsprechende Meldung wird angezeigt. Solche Projekte müssen manuell synchronisiert werden (Menübefehle **UModel | Merge Programmcode von UModel-Projekt** und **UModel | Merge UModel-Projekt von Programmcode**).

Bei der automatischen bzw. manuellen Synchronisierung werden die Änderungen auf einmal im gesamten Projekt aktualisiert. Eine Option, nur eine einzige Klasse zusammenzuführen oder zu aktualisieren, steht in der UModel-Struktur nicht zur Verfügung.

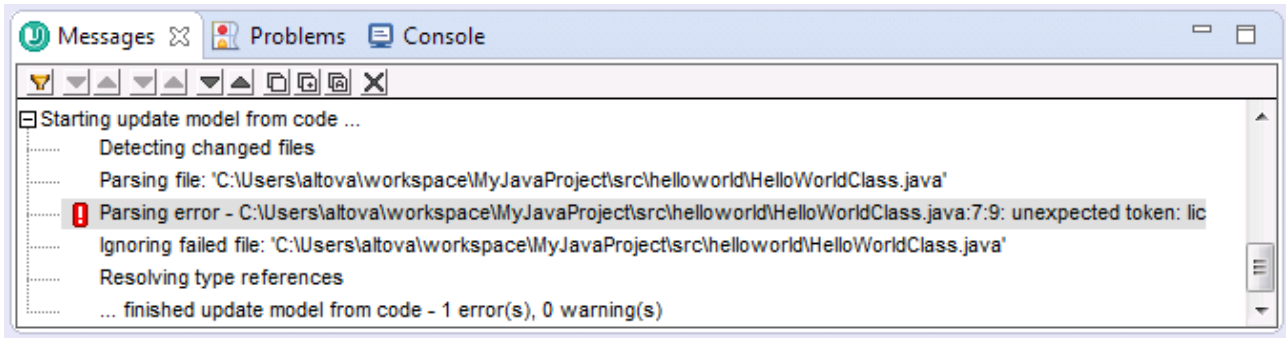
Während der Synchronisierung wird der Fortschritt der Operation in der Eclipse-Statusleiste angezeigt.



Wenn Code nicht geparst werden kann, so wird die Code Engineering-Statusleistenschaltfläche rot angezeigt. Dies passiert auch, wenn beim letzten Reverse Engineering- oder Forward Engineering-Vorgang ein Fehler aufgetreten ist oder wenn bei der Syntaxüberprüfung in UModel ein Fehler gefunden wird.



Im Fenster "Meldungen" werden die Einzelheiten zu den Fehlern angezeigt.



Um die Datei, welche den Fehler verursacht hat, zu öffnen, klicken Sie im Fenster "Meldungen" auf die entsprechende Zeile. Der Cursor erscheint in der Zeile, die den Fehler enthält.

13.7 Beispiel: Einrichten der automatischen Synchronisierung

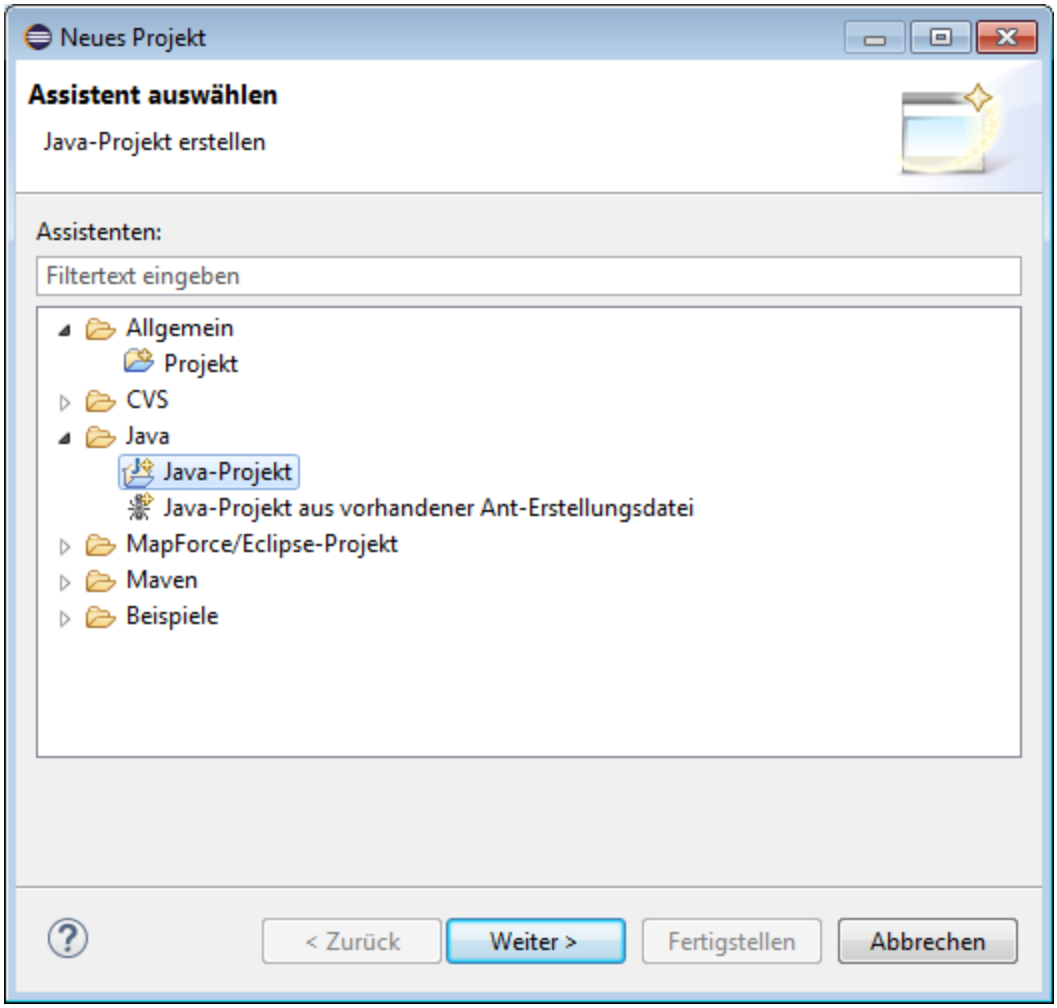
In diesem Tutorial wird gezeigt, wie Sie die automatische Synchronisierung zwischen einem Java-Projekt und dem entsprechenden UML-Modell einrichten. Bevor Sie fortfahren, stellen Sie sicher, dass Sie das UModel Plug-in für Eclipse sowie das von Eclipse benötigte Java Development Kit (nicht nur das Java Runtime Environment) installiert haben.

Schritt 1: Erstellen eines neuen Java-Projekts

Zunächst wollen wir ein neues Java-Projekt in Eclipse erstellen. Für den Zweck dieses Beispiels handelt es sich hierbei um eine einfache Applikation, in der bei Ausführung der Text "Hello World" angezeigt wird.

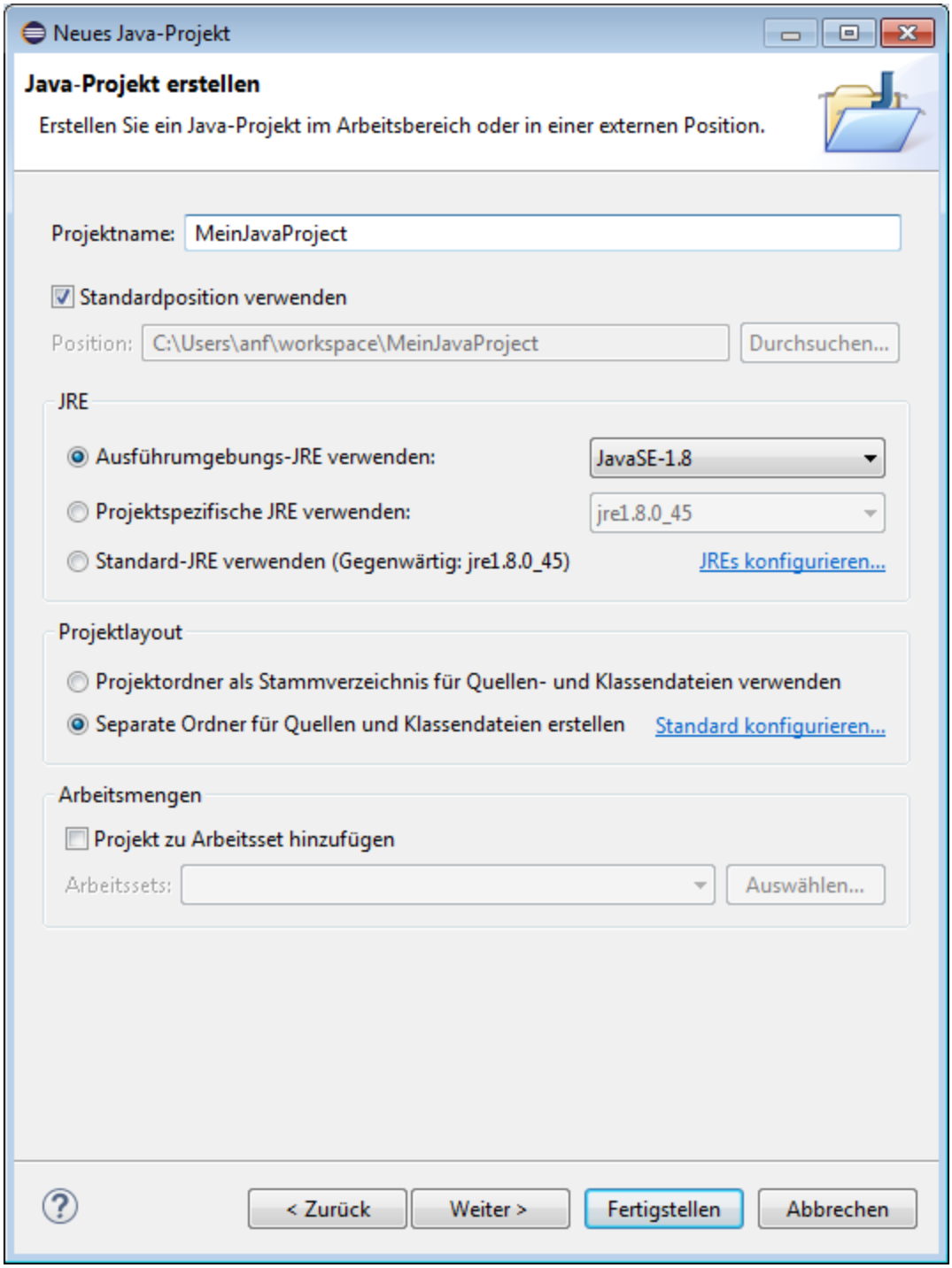
So erstellen Sie die Applikation "Hello, World":

1. Starten Sie Eclipse und wechseln Sie in die Java-Perspektive.
2. Klicken Sie im Menü **Datei** auf **Neu | Projekt**.

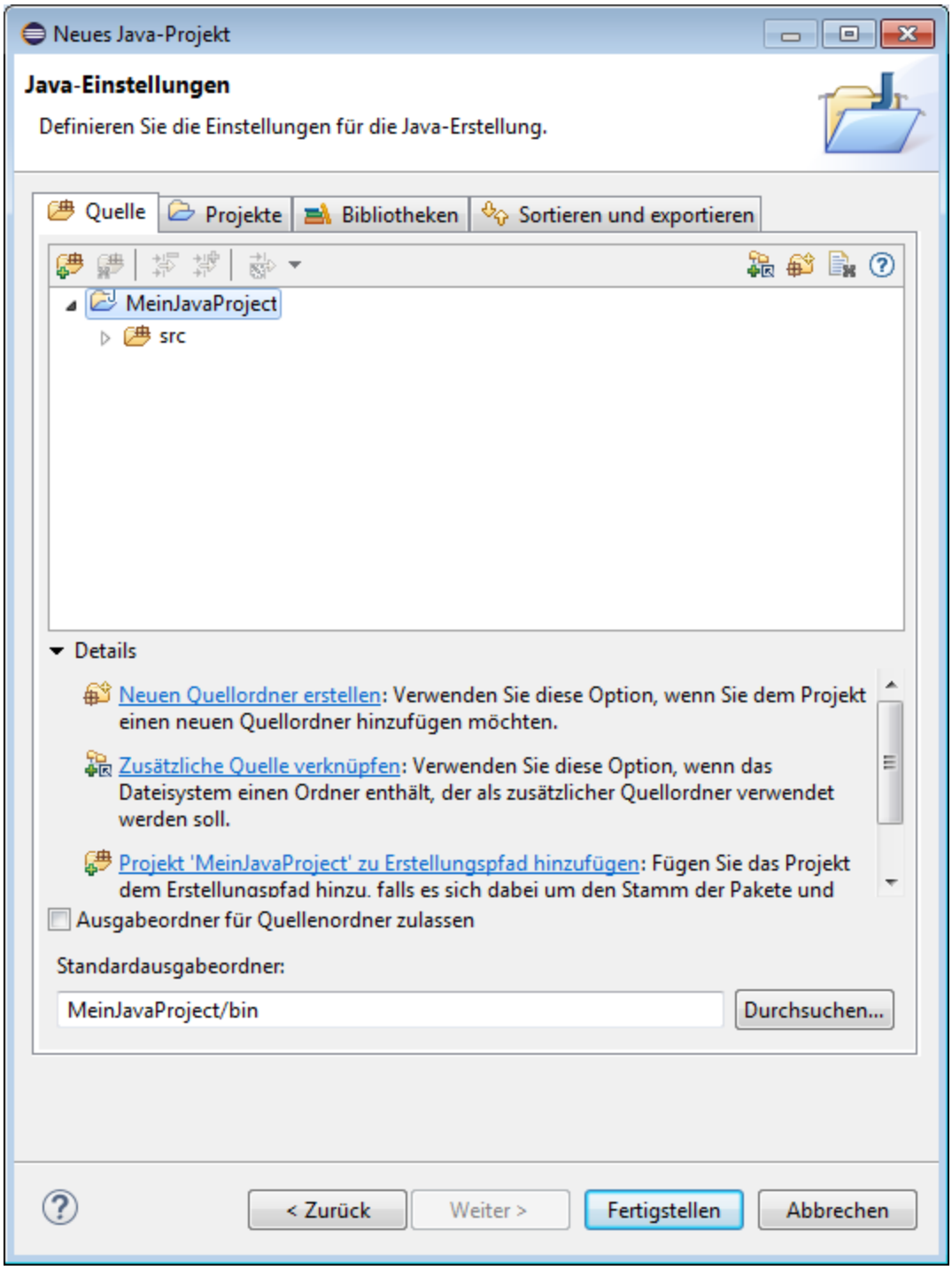


© 2016-2022 Altova GmbH

3. Wählen Sie **Java | Java-Projekt** aus und klicken Sie auf **Weiter**.

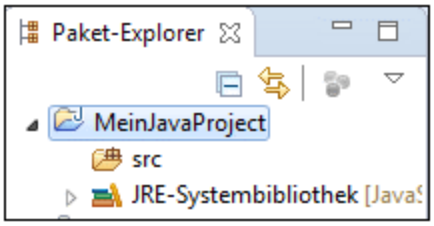


4. Geben Sie als Projektnamen "MeinJavaProjekt" ein und klicken Sie auf **Weiter**.

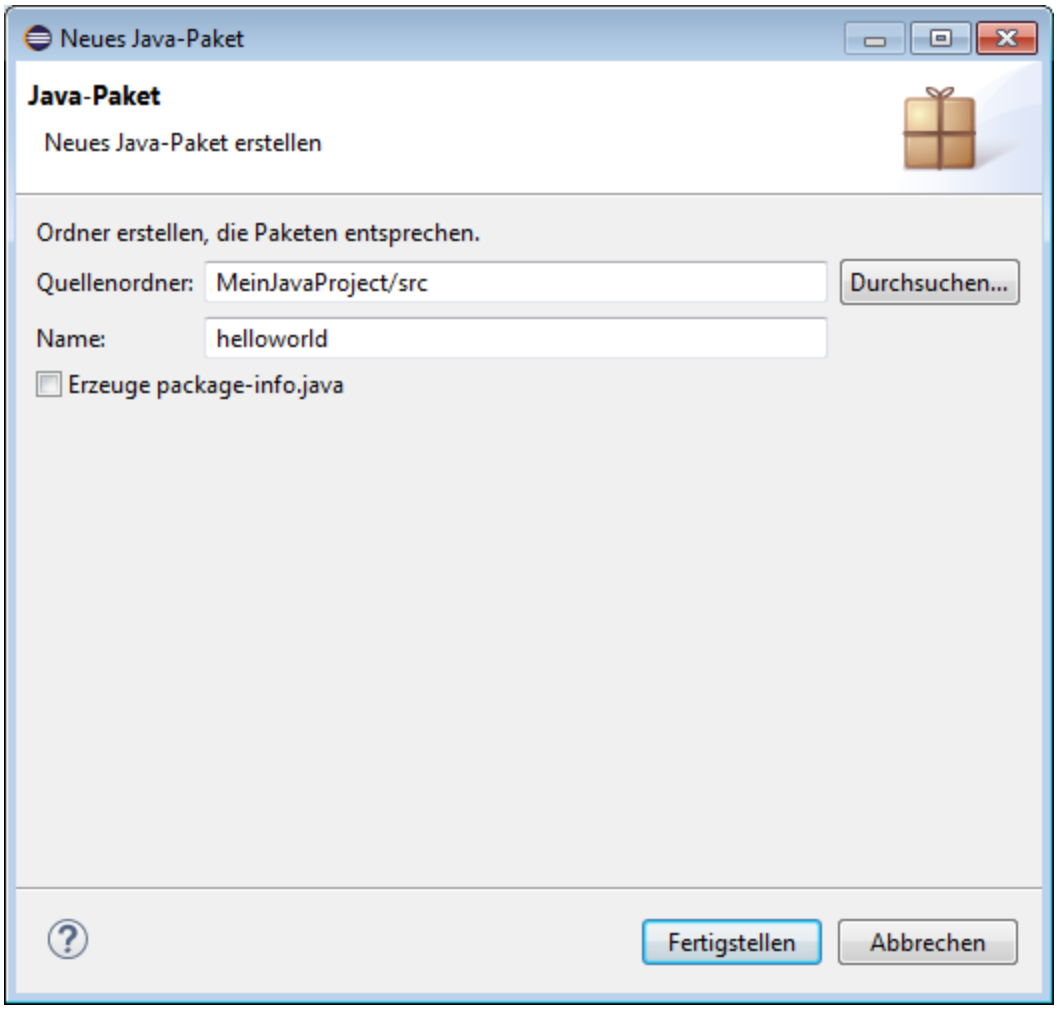


5. Belassen Sie die Standardeinstellungen unverändert und klicken Sie auf **Fertigstellen**. Ihr Projekt wird nun im Paket-Explorer angezeigt.

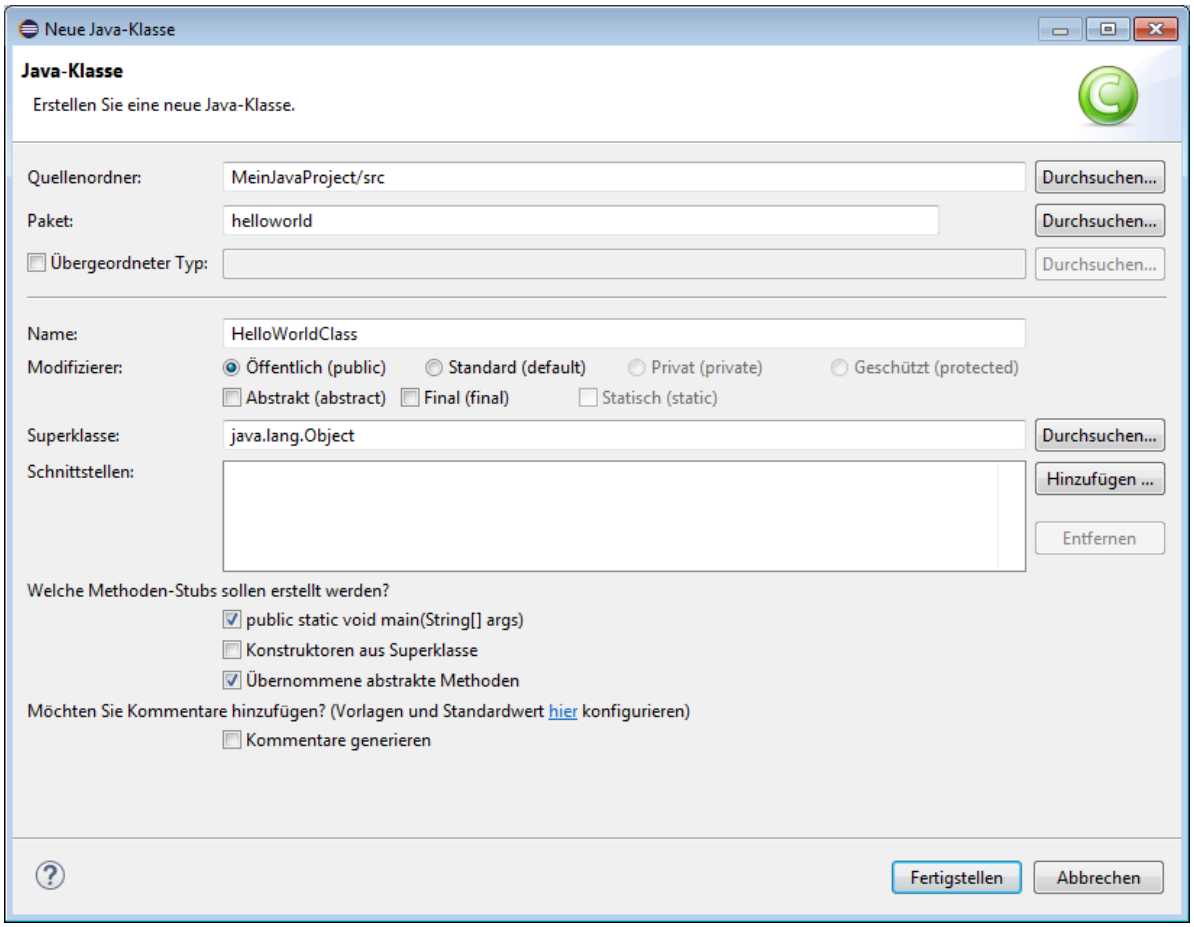
© 2016-2022 Altova GmbH



6. Klicken Sie im Menü **Datei** auf **Neu | Paket**.



7. Geben Sie als Paketnamen "helloworld" ein und klicken Sie auf **Fertigstellen**.
8. Klicken Sie im Menü **Datei** auf **Neu | Klasse**. Geben Sie als Klassennamen "HelloWorldClass" ein und aktivieren Sie die Option **public static_void main(String[] args)**.



9. Öffnen Sie die Klassendatei und fügen Sie zum Body der Klasse den folgenden Text hinzu:

```
package helloworld;

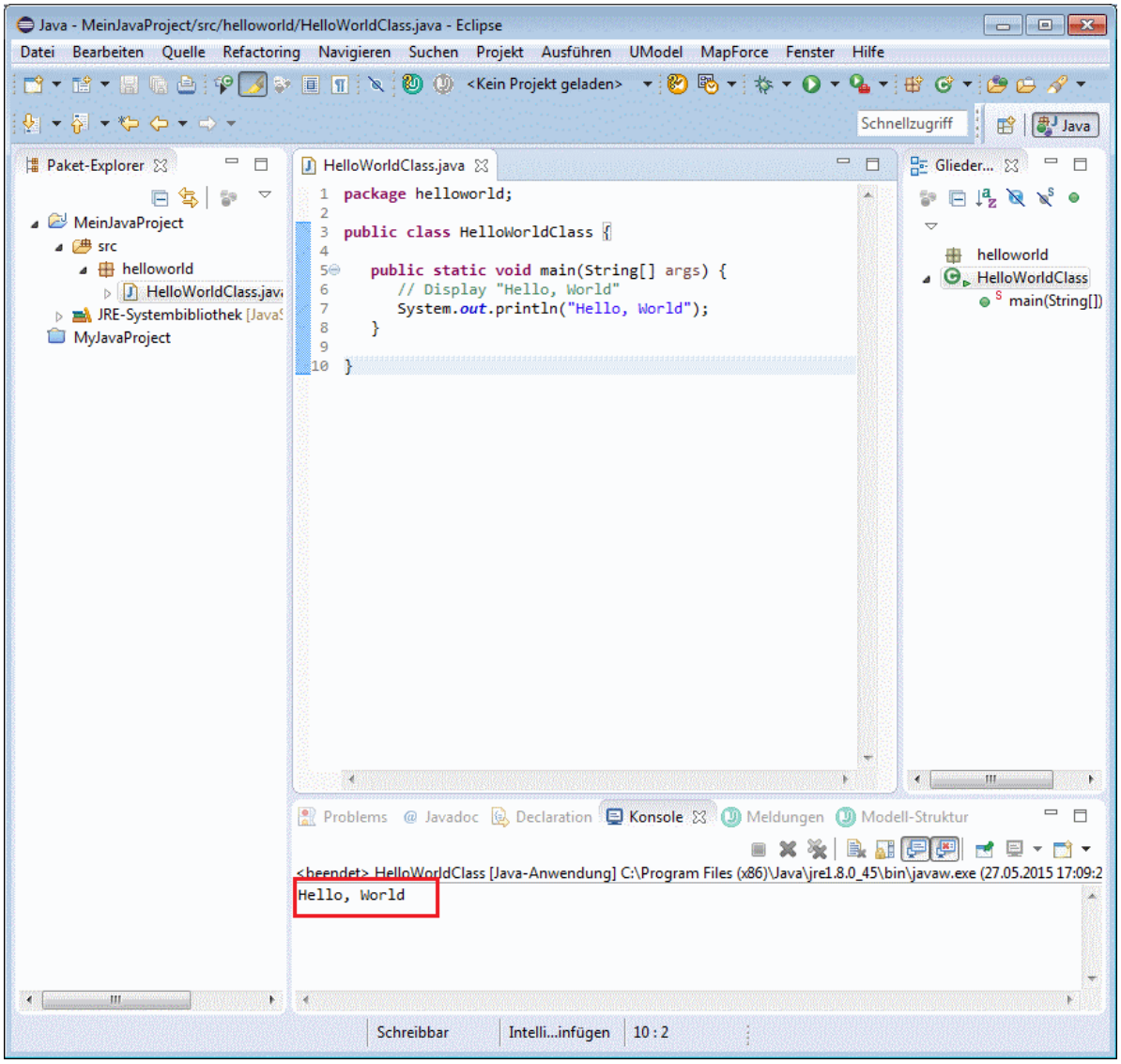
public class HelloWorldClass {

   public static void main(String[] args) {
      // Display "Hello, World"
      System.out.println("Hello, World");
   }

}
```

10. Führen Sie die Applikation aus. In der Konsolenansicht wird der Text "Hello World" angezeigt (siehe Abbildung unten).

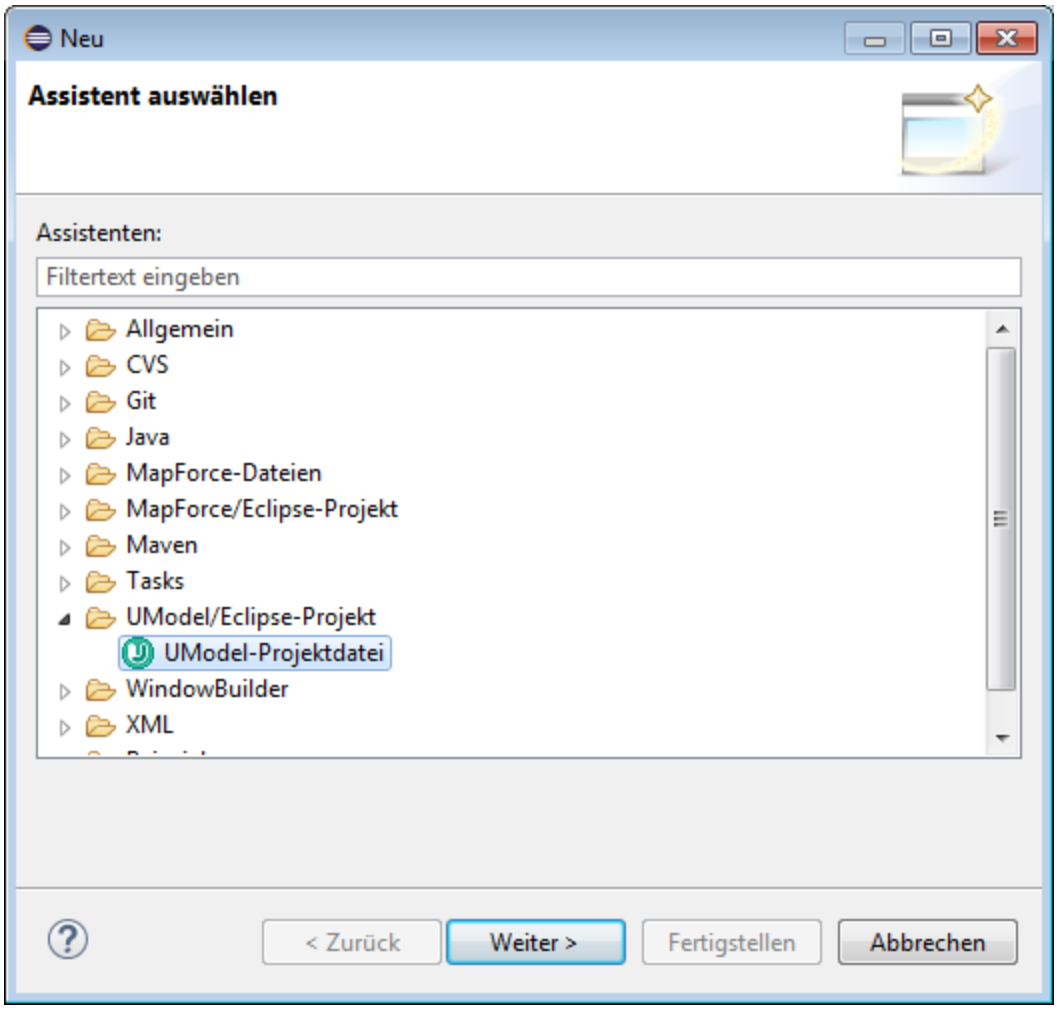
© 2016-2022 Altova GmbH



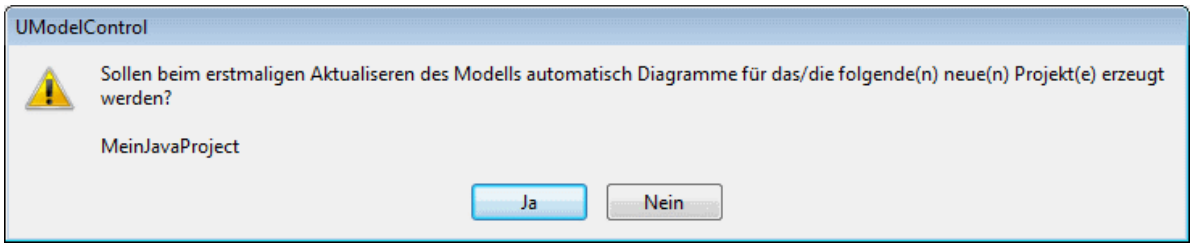
Schritt 2: Hinzufügen des UModel-Projekts zum Java-Projekt

Sie müssen nun das UModel-Projekt zum Eclipse-Projekt hinzufügen. Dadurch wird eine Synchronisierungsbeziehung zwischen dem Modell und dem Code erstellt.

1. Klicken Sie im Menü **Datei** auf **Neu | Andere** und wählen Sie **UModel-Projektdatei.**

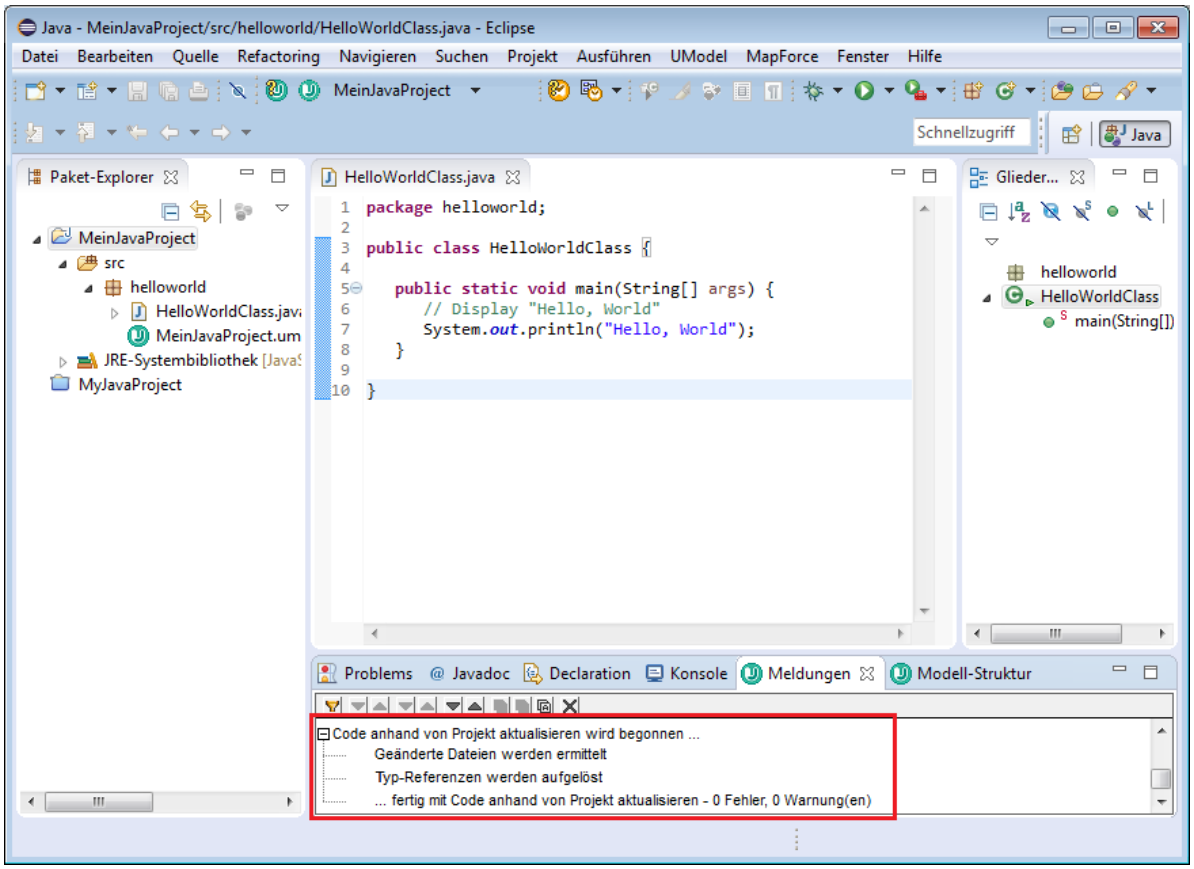


2. Klicken Sie auf **Weiter**. Wenn Sie aufgefordert werden, einen Pfad für das neue UModel-Projekt zu definieren, lassen Sie die Standardeinstellungen unverändert und klicken Sie auf **Fertigstellen**.
3. Wenn Sie von UModel gefragt werden, ob Diagramme für das Projekt erstellt werden sollen, klicken Sie auf **Ja**.



4. Befolgen Sie die Anweisungen des Assistenten und behalten Sie die Standardeinstellungen bei. Wenn Sie auf **Fertigstellen** klicken, wird das neue UModel-Projekt zum Eclipse-Projekt hinzugefügt und der Code wird automatisch mit dem Modell synchronisiert. Beachten Sie die in der UModel-Ansicht "Meldungen" angezeigten Meldungen.

© 2016-2022 Altova GmbH

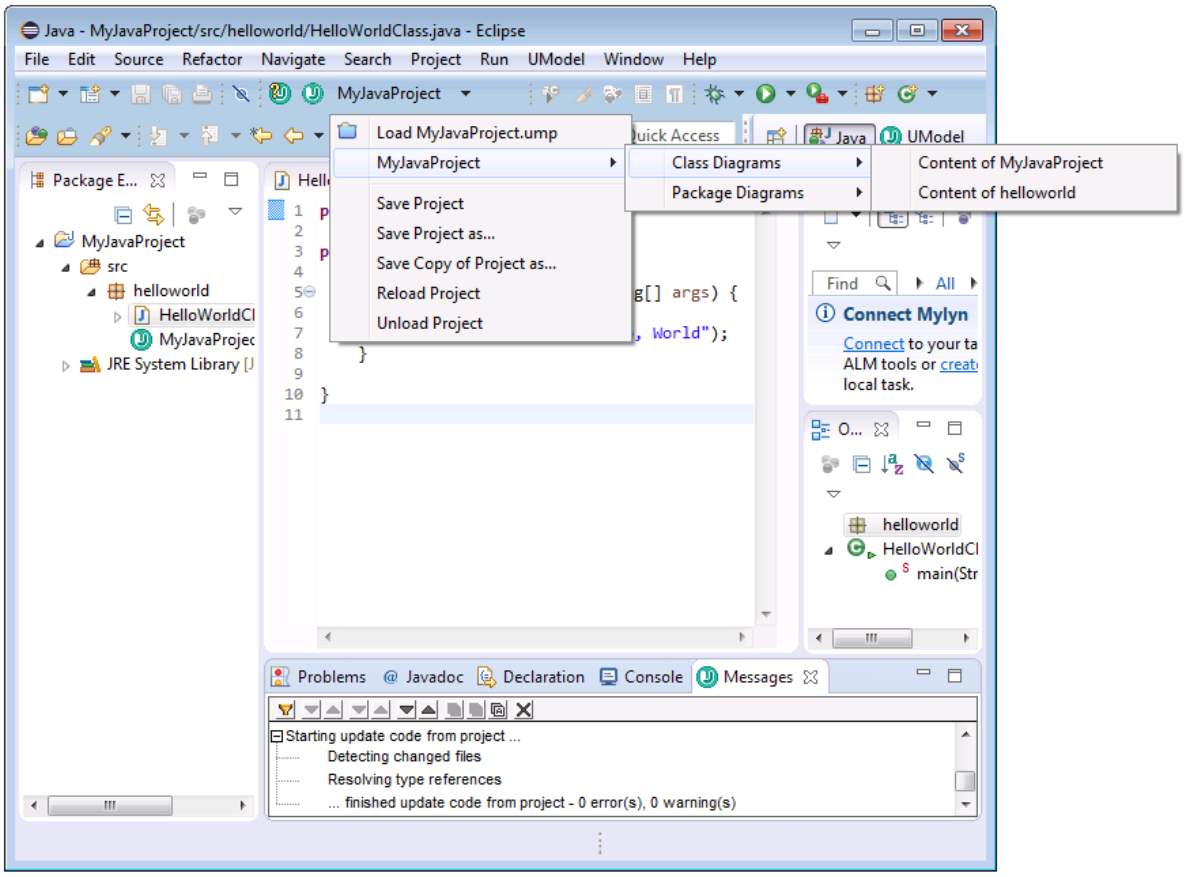


Schritt 3: Auslösen der automatischen Synchronisierung des Codes anhand des Modells

Um die automatische Synchronisierung des Codes anhand des Modells auszulösen, werden wir im Modell im Klassendiagramm einige Änderungen vornehmen. Dabei werden wir eine neue Eigenschaft namens "Property1" vom Typ "Boolean" zur Klasse hinzufügen.

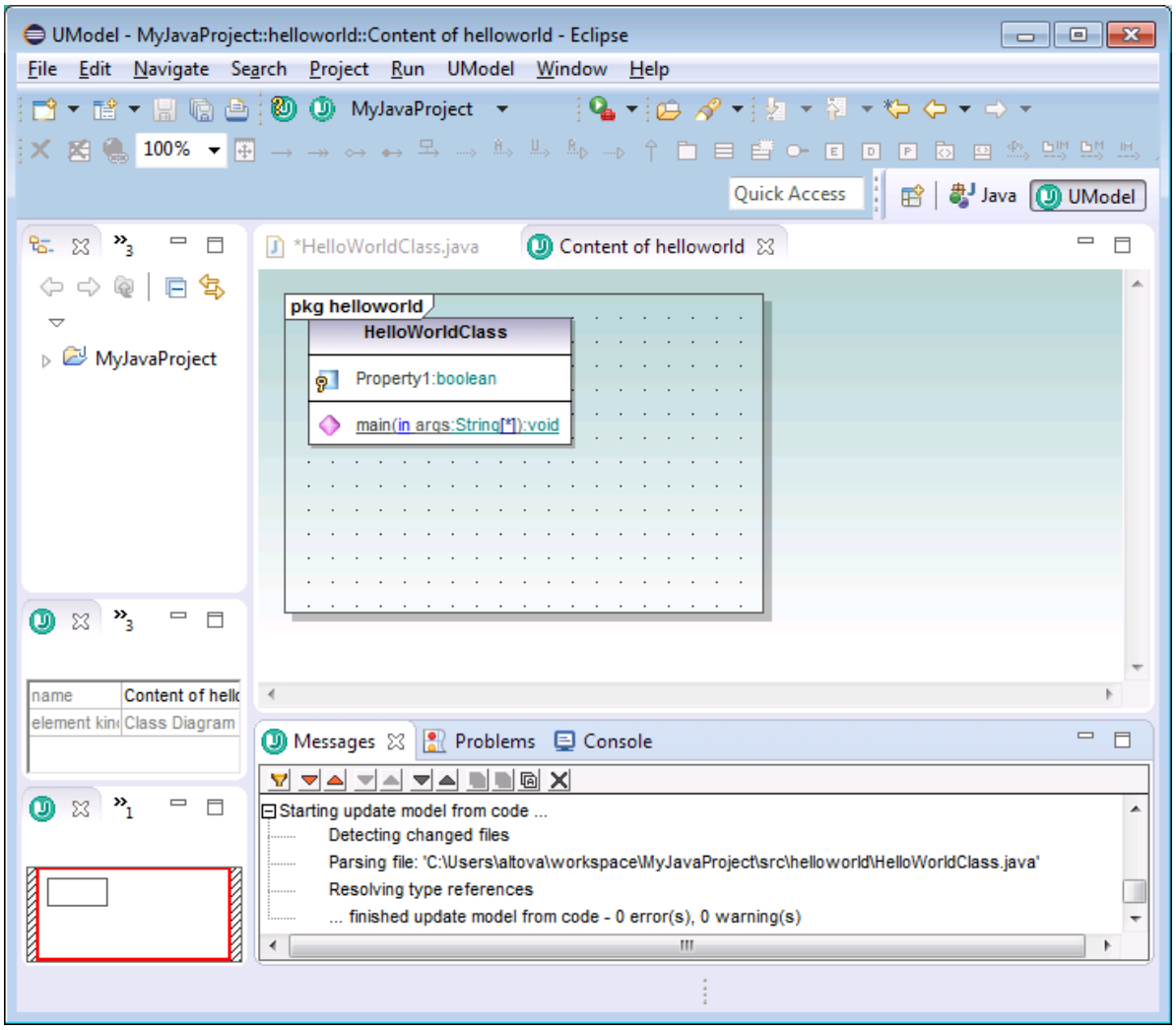
So fügen Sie die Eigenschaft zur Klasse hinzu:

1. Erweitern Sie die Projekt-Dropdown-Liste in der UModel-Symbolleiste und öffnen Sie das generierte Klassendiagramm "content of helloworld".

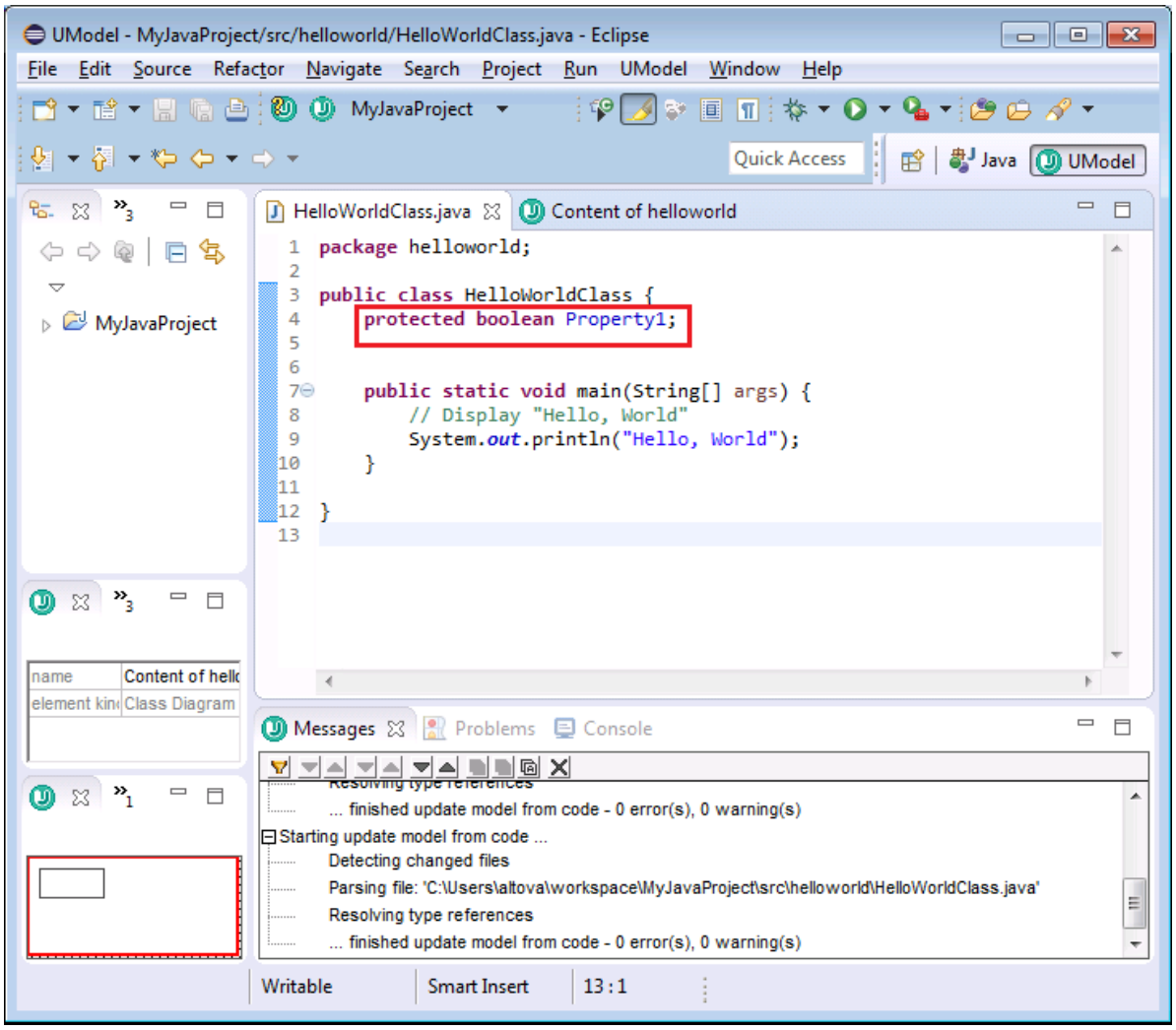


2. Klicken Sie mit der rechten Maustaste auf die Klasse und wählen Sie im Kontextmenü den Befehl **Neu | Eigenschaft**.
3. Geben Sie den Eigenschaftsnamen ("Property1"), gefolgt von einem Doppelpunkt (:), gefolgt vom Typ ("boolean") ein.

© 2016-2022 Altova GmbH



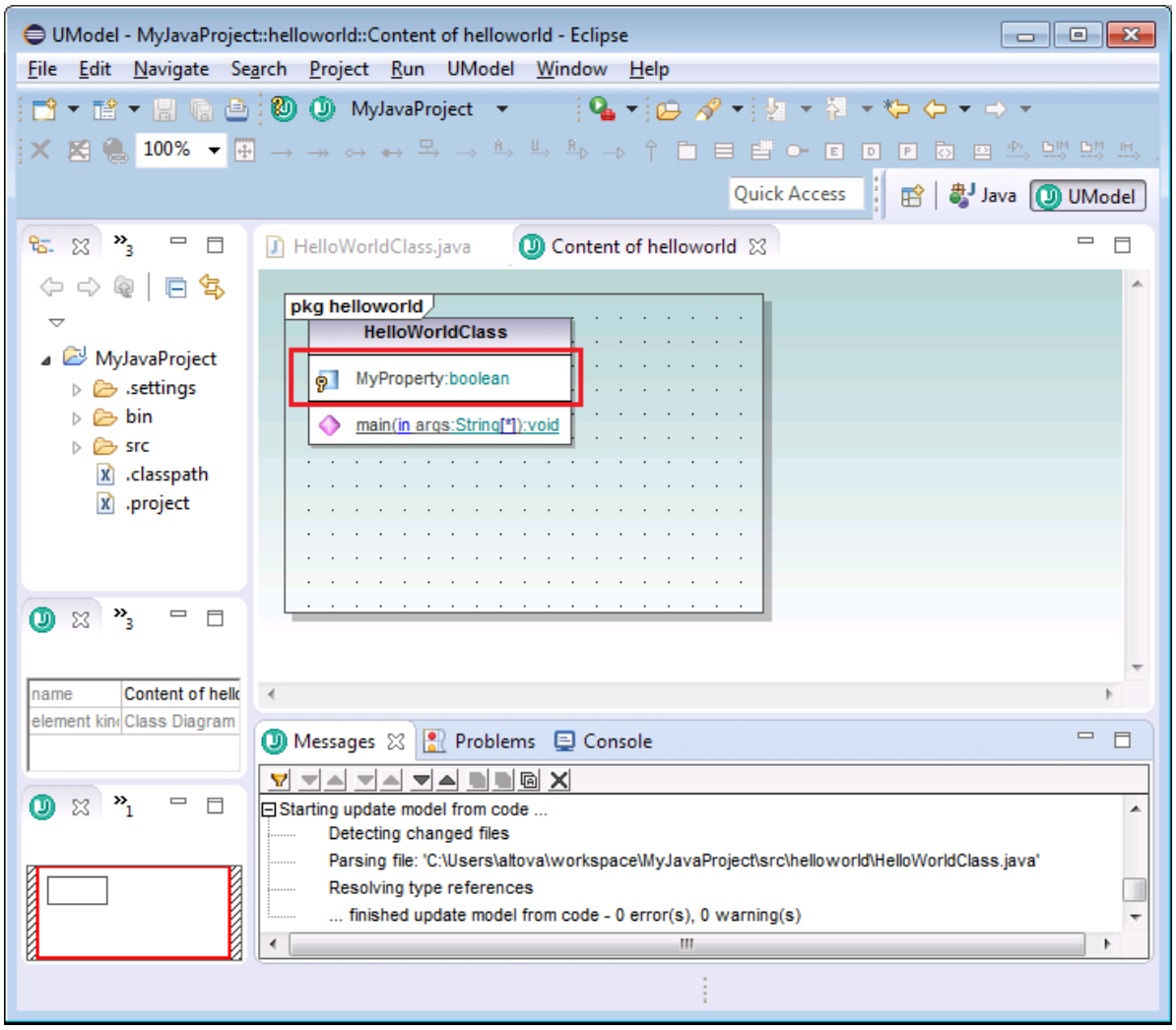
4. Wechseln Sie zurück in den Code-Editor. Beachten Sie, dass die neu hinzugefügte Eigenschaft nun auch im Code zu sehen ist.



Schritt 4. Auslösen der automatischen Synchronisierung des Modells anhand des Codes

Wir wollen nun die automatische Synchronisierung der Änderungen in der umgekehrten Richtung (vom Code ins Modell) vornehmen. Ändern Sie dazu im Code den Namen der Eigenschaft "Property1" in "MyProperty" und speichern Sie das Projekt anschließend. Beachten Sie, dass die Änderungen nun auch im Diagramm angezeigt werden.

© 2016-2022 Altova GmbH


UModel - MyJavaProject::helloworld::Content of helloworld - Eclipse
File Edit Navigate Search Project Run UModel Window Help
MyJavaProject
100%
Quick Access
Java
UModel
HelloWorldClass.java
Content of helloworld
pkg helloworld
HelloWorldClass
MyProperty:boolean
main(in args:String[*]):void
MyJavaProject
.settings
bin
src
.classpath
.project
name
Content of hello
element kin
Class Diagram
Messages
Problems
Console
Starting update model from code ...
Detecting changed files
Parsing file: 'C:\Users\altova\workspace\MyJavaProject\src\helloworld\HelloWorldClass.java'
Resolving type references
... finished update model from code - 0 error(s), 0 warning(s)

14 Versionskontrolle

Die Versionskontrollunterstützung in UModel steht über die Microsoft Source Control Plug-in API (vormals bekannt als MSSCCI API), Version 1.1, 1.2 sowie 1.3 zur Verfügung. Dadurch können Sie Versionskontrollbefehle wie z.B. "Einchecken" oder "Auschecken" direkt über UModel an praktisch jedem Versionskontrollsystem ausführen, das nativen Clients oder Drittanbieter-Clients über das Microsoft Source Control Plug-in API die Verbindung zu UModel gestattet.

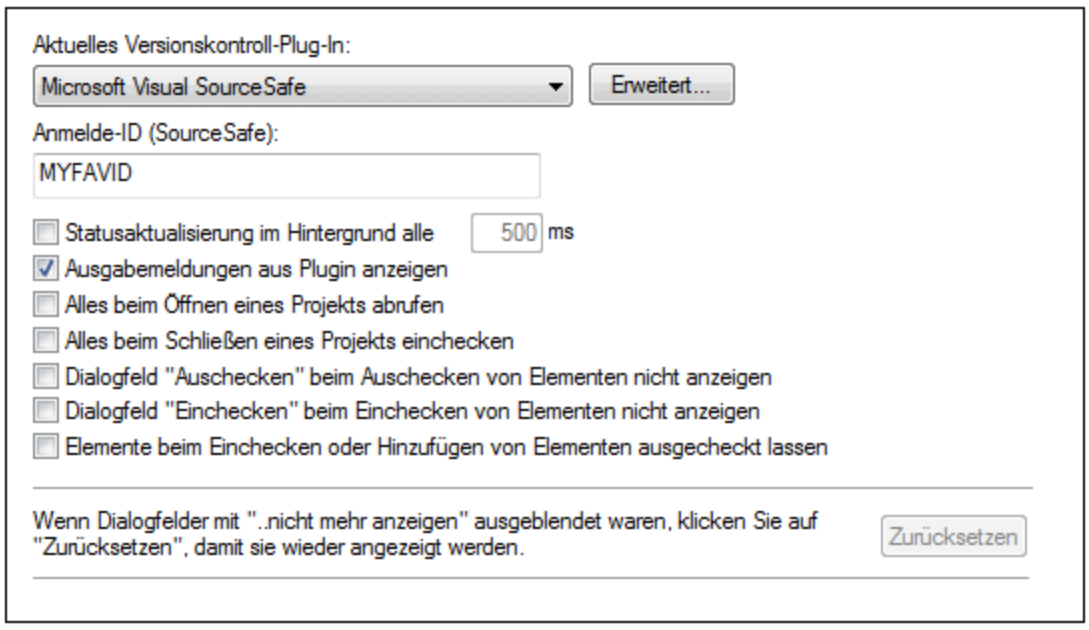
Als Versionskontrollsystemanbieter kann jedes kommerzielle oder nicht kommerzielle Plug-in, das die Microsoft Source Control Plug-in API unterstützt und mit einem kompatiblen Versionskontrollsystem verbunden werden kann, verwendet werden. Eine Liste von von Altova getesteten Versionskontrollsystemen und Plug-ins finden Sie unter Unterstützte Versionskontrollsysteme 705.

Installieren und Konfigurieren des Versionskontrollanbieters

Um die auf Ihrem System verfügbaren Versionskontrollsysteme anzuzeigen, gehen Sie folgendermaßen vor:

1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Versionskontrolle**.

Alle mit der Microsoft Source Code Control Plug-in API kompatiblen Versionskontroll-Plug-ins werden in der Dropdown-Liste **Aktuelles Versionskontroll-Plug-in** angezeigt.



Wenn auf Ihrem System kein kompatibles Plug-in gefunden wurde, wird die folgende Meldung angezeigt:

```
"Die Registrierung des installierten Versionskontrollproviders konnte nicht gefunden werden
oder ist unvollständig."
```

Bei einigen Versionskontrollsystemen wird das Versionskontroll-Plug-in nicht automatisch installiert. In diesem Fall müssen Sie es gesondert installieren. Eine genauere Anleitung dazu finden Sie in der Dokumentation zum

© 2016-2022 Altova GmbH

jeweiligen Versionskontrollsystem. Ein Plug-in (Anbieter), das mit der Microsoft Source Code Control Plug-in API kompatibel ist, sollte auf Ihrem Betriebssystem unter dem folgenden Registrierdateieintrag zu finden sein:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders
```

Wenn das Plug-in korrekt installiert wurde, steht es automatisch in der Liste der Plug-ins in UModel zur Verfügung.

Aufrufen der Versionskontrollbefehle

Die Versionskontrollbefehle stehen über das Menü **Projekt | Versionskontrolle** zur Verfügung.

Probleme im Zusammenhang mit den Ressourcen / der Geschwindigkeit

Bei sehr großen Versionskontrolldatenbanken kann die automatische Durchführung von Statusaktualisierungen im Hintergrund etwas länger dauern.

Um schneller arbeiten zu können, können Sie versuchen, auf dem Register "Versionskontrolle" (Aufruf über **Extras | Optionen**) das Feld "**Statusaktualisierung im Hintergrund alle xxx Sek**." zu deaktivieren oder das Aktualisierungsintervall zu vergrößern.

Anmerkung: Die **64-Bit**-Version Ihrer Altova-Applikation unterstützt automatisch alle in dieser Dokumentation aufgelisteten 32-Bit-Versionskontrollsysteme. Bei Verwendung einer 64-Bit-Altova-Applikation zusammen mit einem 32-Bit-Versionskontrollsystem ist die Option **Statusaktualisierung im Hintergrund alle xxx Sek** automatisch deaktiviert und kann nicht ausgewählt werden.

Vergleich mit Altova DiffDog

Sie können viele Versionskontrollsysteme (einschließlich Git und TortoiseSVN) für die Verwendung mit Altova DiffDog als Vergleichstool konfigurieren. Nähere Informationen zu DiffDog finden Sie unter https://www.altova.com/de/diffdog.html. Die Dokumentation zu DiffDog finden Sie unter https://www.altova.com/de/documentation.html.

14.1 Einrichten der Versionskontrolle

So richten Sie eine Versionskontrolle ein und stellen Dateien in einem UModel-Projekt unter eine Versionskontrolle:

1. Installieren Sie ein Versionskontrollprogramm (siehe Unterstützte Versionskontrollsysteme 705), falls noch keines installiert ist. Richten Sie die Versionskontrolldatenbank (das Repository), in der Sie Ihre Arbeit speichern möchten, ein.
2. Erstellen Sie einen Ordner für den lokalen Arbeitsbereich, der die Arbeitsdateien enthalten soll, die unter Versionskontrolle gestellt werden sollen. Der Ordner, der alle Ihre Arbeitsbereichordner und -Dateien enthält, wird als ihr lokaler Ordner und der Pfad zum lokalen Ordner als der lokale Pfad bezeichnet. Der Ordner wird an einen bestimmten Ordner im Repository gebunden.
3. Erstellen Sie in Ihrer Altova-Applikation einen Applikationsprojektordner, zu dem Sie die unter Versionskontrolle zu stellenden Dateien hinzufügen müssen. Diese Gliederung der Dateien in einem Applikationsprojekt ist abstrakt. Die Dateien in einem Projekt referenzieren physische lokal (vorzugsweise in einem einzigen Ordner, falls nötig mit Unterordnern) gespeicherte Dateien.
4. In der Datenbank des Versionskontrollsystems (die auch als Versionskontrolle oder Repository bezeichnet wird) wird ein Ordner erstellt, der an den lokalen Ordner gebunden ist. Dieser (als gebundener Ordner bezeichnete) Ordner repliziert die Struktur des lokalen Ordners, sodass sich alle unter Versionskontrolle zu stellenden Dateien an der richtigen Stelle im gebundenen Ordner befinden. Der gebundene Ordner wird normalerweise erstellt, wenn Sie zum ersten Mal eine Datei oder ein Applikationsprojekt zur Versionskontrolle hinzufügen.

© 2016-2022 Altova GmbH

14.2 Unterstützte Versionskontrollsysteme

In der nachfolgenden Liste sind die von UModel unterstützten Versionskontrollsysteme zusammen mit ihren entsprechenden Versionskontroll-Clients (SCCs) aufgelistet. Die Liste ist alphabetisch nach Versionskontrollsystem geordnet.

- Altova hat in UModel die Microsoft Source Control Plug-in API (Version 1.1, 1.2 und 1.3) implementiert und die Unterstützung für die aufgelisteten Treiber und Versionskontrollsysteme getestet. Aller Voraussicht nach wird UModel diese Produkte, falls diese aktualisiert werden, weiterhin unterstützen.
- Auch Versionskontroll-Clients, die nicht in der nachstehenden Liste erwähnt sind, die aber die Microsoft Source Control Plug-in API implementieren, sollten ebenfalls mit UModel funktionieren.

Versionskontrollsystem	Versionskontroll-Clients
AccuRev 4.7.0 Windows	AccuBridge for Microsoft SCC 2008.2
Bazaar 1.9 Windows	Aigenta Unified SCC 1.0.6
Borland StarTeam 2008	Borland StarTeam Cross-Platform Client 2008 R2
Codice Software Plastic SCM Professional 2.7.127.10 (Server)	Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)
Collabnet Subversion 1.5.4	• Aigenta Unified SCC 1.0.6 • PushOK SVN SCC 1.5.1.1 • PushOK SVN SCC x64 Version 1.6.3.1 • TamTam SVN SCC 1.2.24
ComponentSoftware CS-RCS (PRO) 5.1	ComponentSoftware CS-RCS (PRO) 5.1
Dynamsoft SourceAnywhere for VSS 5.3.2 Standard/Professional Server	Dynamsoft SourceAnywhere für VSS 5.3.2 Client
Dynamsoft SourceAnywhere Hosted	Dynamsoft SourceAnywhere Hosted Client (22252)
Dynamsoft SourceAnywhere Standalone 2.2 Server	Dynamsoft SourceAnywhere Standalone 2.2 Client
Git	PushOK GIT SCC Plug-in (siehe Versionskontrolle mit Git 728)
IBM Rational ClearCase 7.0.1 (LT)	IBM Rational ClearCase 7.0.1 (LT)
March-Hare CVSNT 2.5 (2.5.03.2382)	Aigenta Unified SCC 1.0.6
March-Hare CVS Suite 2008	• Jalindi Igloo 1.0.3 • March-Hare CVS Suite Client 2008 (3321) • PushOK CVS SCC NT 2.1.2.5 • PushOK CVS SCC x64 Version 2.2.0.4 • TamTam CVS SCC 1.2.40
Mercurial 1.0.2 für Windows	Sergey Antonov HgSCC 1.0.1

Versionskontrollsystem	Versionskontroll-Clients
Microsoft SourceSafe 2005 mit CTP	Microsoft SourceSafe 2005 mit CTP
Microsoft Visual Studio Team System 2008/2010 Team Foundation Server	Microsoft Team Foundation Server 2008/2010 MSSCCI Provider
Perforce 2008 P4S 2008.1	Perforce P4V 2008.1
PureCM Server 2008/3a	PureCM Client 2008/3a
QSC Team Coherence Server 7.2.1.35	QSC Team Coherence Client 7.2.1.35
Reliable Software Code Co-Op 5.1a	Reliable Software Code Co-Op 5.1a
Seapine Surround SCM Client/Server für Windows 2009.0.0	Seapine Surround SCM Client 2009.0.0
Serena Dimensions Express/CM 10.1.3 für Win32 Server	Serena Dimensions 10.1.3 for Win32 Client
Softimage Alienbrain Server 8.1.0.7300	Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300
SourceGear Fortress 1.1.4 Server	SourceGear Fortress 1.1.4 Client
SourceGear SourceOffsite Server 4.2.0	SourceGear SourceOffsite Client 4.2.0 (Windows)
SourceGear Vault 4.1.4 Server	SourceGear Vault 4.1.4 Client
VisualSVN Server 1.6	• Aigenta Unified SCC 1.0.6 • PushOK SVN SCC 1.5.1.1 • PushOK SVN SCC x64 Version 1.6.3.1 • TamTam SVN SCC 1.2.24

© 2016-2022 Altova GmbH

14.3 Versionskontrollbefehle

Die Versionskontrollbefehle von UModel werden in den folgenden Abschnitten anhand von Visual SourceSafe angezeigt. In den Beispielen in diesem Abschnitt wird das UModel-Projekt **Bank_CSharp.ump** (und damit verknüpfte Codedateien) aus dem Ordner C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples verwendet. Beachten Sie, dass ein Versionskontrollprojekt nicht dasselbe ist wie ein UModel-Projekt. Versionskontrollprojekte sind verzeichnisunabhängig, während UModel-Projekte logische Konstruktionen ohne direkte Verzeichnisabhängigkeit sind.

Gehen Sie folgendermaßen vor, um die Versionskontrollbefehle aufzurufen:

- Verwenden Sie den Menübefehl **Projekt | Versionskontrolle**
- Verwenden Sie das **Kontextmenü** in der Modell-Struktur
- Klicken Sie auf die Schaltflächen der Versionskontroll-Symbolleiste. Die Symbolleiste wird über **Extras | Anpassen | Symbolleisten** aktiviert.

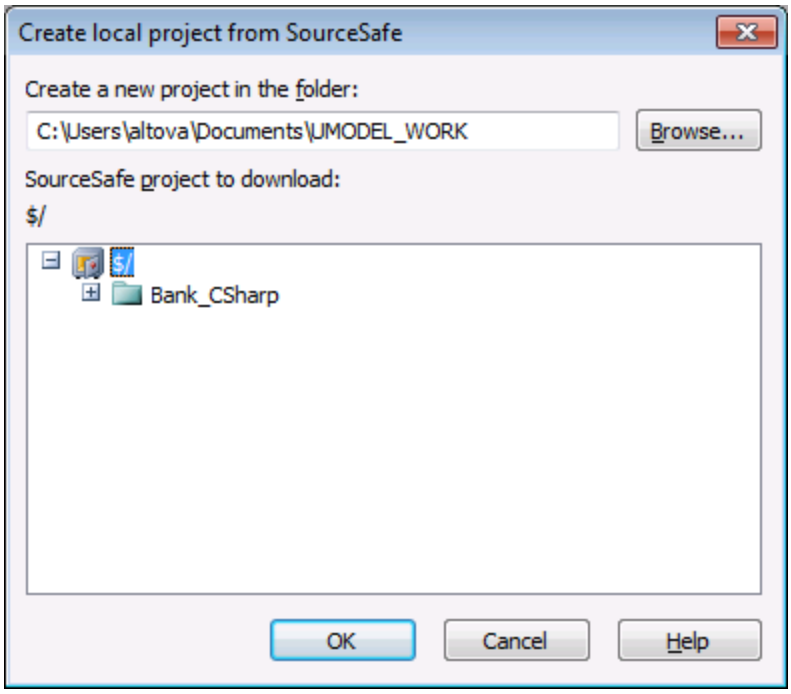
Die Beschreibung der im Folgenden beschriebenen Versionskontrollbefehle gelten für die **Standalone**-Version von UModel. In der Visual Studio und der Eclipse Version von UModel werden die Versionskontrollbefehle und Menübefehle verwendet, die in diesen IDEs zur Verfügung stehen.

Aus Versionskontrolle öffnen 707
Versionskontrolle aktivieren 710
Aktuellste Version holen 711
Abrufen 711
Ordner abrufen 712
Auschecken 714
Einchecken 715
Auschecken rückgängig... 716
Zu Versionskontrolle hinzufügen 717
Von Versionskontrolle ausgliedern 720
Aus Versionskontrolle freigeben 721
Verlauf anzeigen 722
Unterschiede anzeigen 724
Eigenschaften anzeigen 725
Status aktualisieren 726
Versionskontrollmanager 726
Versionskontrolle wechseln 726

14.3.1 Aus Versionskontrolle öffnen

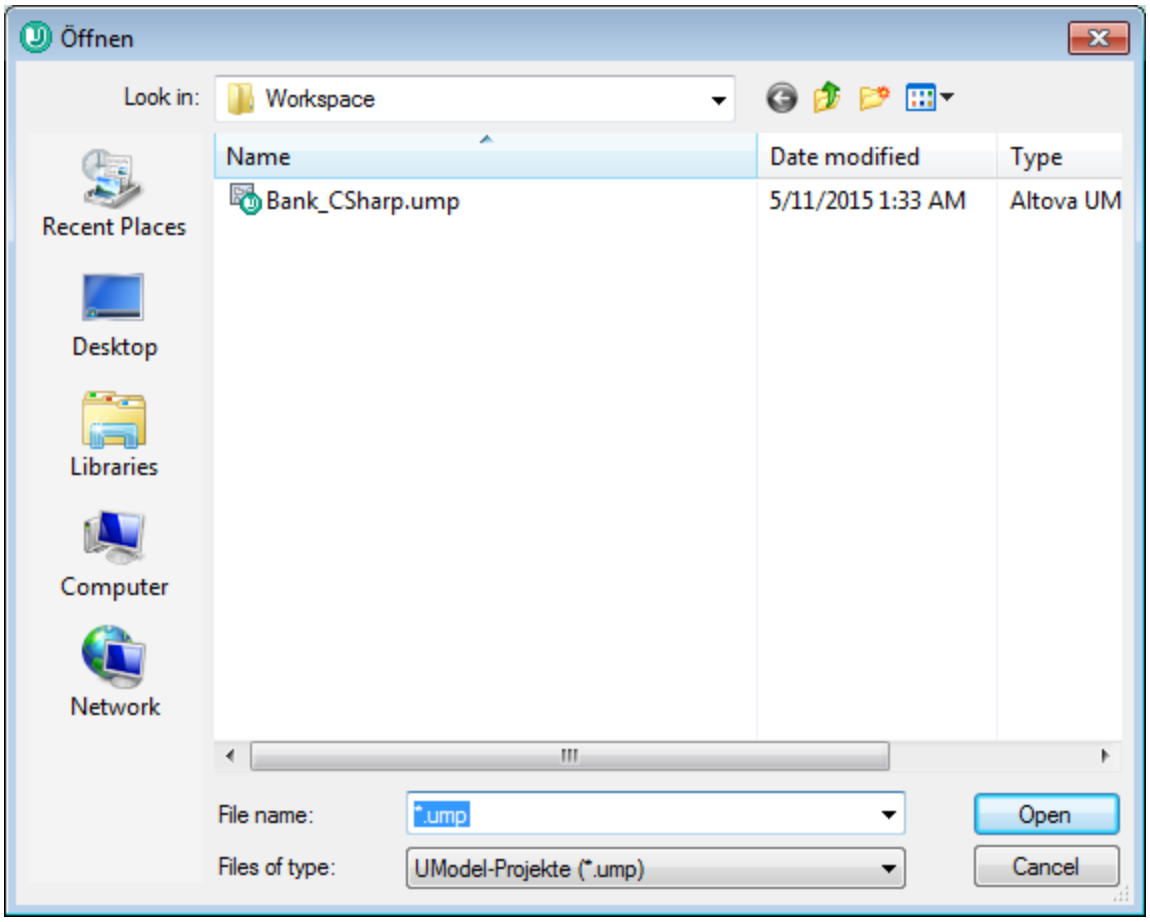
Mit dem Befehl "Aus Versionskontrolle öffnen" wird ein lokales Projekt anhand einer bestehenden Versionskontrolldatenbank erstellt und unter Versionskontrolle - in diesem Fall SourceSafe - gestellt.

1. Wählen Sie **Projekt | Versionskontrolle | Aus Versionskontrolle öffnen**. Daraufhin wird das Anmeldedialogfeld geöffnet, in dem Sie Ihre Anmeldedaten eingeben, bevor Sie fortfahren können. Daraufhin wird das Dialogfeld "Create local project from SourceSafe" angezeigt.
2. Definieren Sie das Verzeichnis, das das neue lokale Projekt enthalten soll, z.B. c:\temp\ssc. Dies wird zum neuen **Arbeitsverzeichnis** oder dem Auscheck-Ordner.



3. Wählen Sie das gewünschte SourceSafe-Projekt aus, z.B. Bank_CSharp. Wenn der hier von Ihnen definierte Ordner noch nicht existiert, erscheint ein Dialogfeld, in dem Sie aufgefordert werden, ihn zu erstellen.
4. Klicken Sie auf **Ja,** um das neue Verzeichnis zu erstellen. Das Dialogfeld "Öffnen" wird angezeigt.

© 2016-2022 Altova GmbH



5. Wählen Sie die UModel-Projektdatei **Bank_CSharp.ump** aus und klicken Sie auf "Öffnen".

   Bank_CSharp.ump wird nun in UModel geöffnet und die Datei wird unter Versionskontrolle gestellt. Angezeigt wird dies in der Modell-Struktur durch ein Schlosssymbol über dem Root-Ordner. Der Root-Ordner repräsentiert sowohl die Projektdatei als auch das Arbeitsverzeichnis für Versionskontrolloperationen.



   Das Verzeichnis "BankCSharp" wurde lokal erstellt. Sie können nun mit diesen Dateien normal arbeiten.

Anmerkung:

Informationen, wie Sie die beim Synchronisieren von Code generierten **Codedateien** unter Versionskontrolle stellen, finden Sie unter: Hinzufügen zur Versionskontrolle 717

Symbole der Versionskontrolle:

 oder

Das Vorhängeschloss-Symbol zeigt an, dass diese Datei oder dieser Ordner **unter Versionskontrolle steht**, aber derzeit nicht ausgecheckt ist.

 oder

Das **rote** Kontrollhäkchen zeigt an, dass diese Datei **ausgecheckt** ist, d.h. dass die UModel-Projektdatei (oder Codedatei) zur Bearbeitung ausgecheckt wurde.
Das Sternchen in der Titelleiste der Applikation bedeutet, dass diese Datei verändert wurde. Beim Schließen werden Sie aufgefordert, die Datei zu speichern.

 oder

Das Personensymbol zeigt an, dass die Datei(en) von jemand anderem im Netzwerk oder von Ihnen in ein anderes Arbeitsverzeichnis ausgecheckt wurde(n).

14.3.2 Versionskontrolle aktivieren

Mit Hilfe dieses Befehls können Sie die Versionskontrolle für ein UModel-Projekt aktivieren oder deaktivieren. Der Befehl steht über das Projekt-Menü **Projekt | Versionskontrolle | Versionskontrolle aktivieren** zur Verfügung. Bei Auswahl dieser Option für eine Datei oder einen Ordner, wird die Versionskontrolle für das gesamte UModel-Projekt aktiviert/deaktiviert.

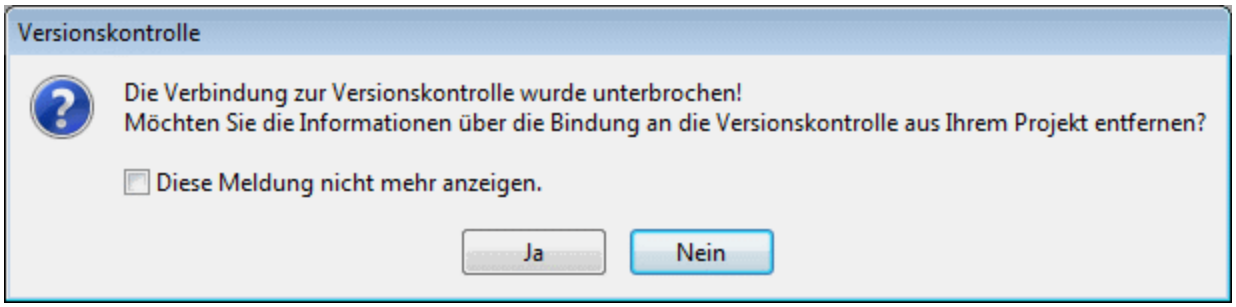
So aktivieren Sie die Versionskontrolle für ein Projekt:

1. Wählen Sie die Menüoption **Projekt | Versionkontrolle** und aktivieren Sie das Kontrollkästchen **Versionskontrolle aktivieren**. Der vorherige Ein-/Auscheck-Status der verschiedenen Dateien wird abgerufen und im Fenster "Modell-Struktur" angezeigt.

So deaktivieren Sie die Versionskontrolle für ein Projekt:

1. Wählen Sie die Menüoption **Projekt | Versionkontrolle** und deaktivieren Sie das Kontrollkästchen **Versionskontrolle aktivieren**.

© 2016-2022 Altova GmbH



Sie werden nun gefragt, ob Sie die Binding-Informationen aus dem Projekt entfernen möchten.

Um die Versionskontrolle **provisorisch** zu deaktivieren, wählen Sie **Nein**.

Um die Versionskontrolle für dieses Projekt **permanent** zu deaktivieren, wählen Sie **Ja**.

14.3.3 Aktuellste Version holen

Mit diesem Befehl wird die aktuellste Version der ausgewählten Datei(en) aus dem Versionskontrollspeicher in das Arbeitsverzeichnis geholt. Die Dateien werden als schreibgeschützte Dateien und nicht ausgecheckt abgerufen.

Wenn die betroffenen Dateien derzeit ausgecheckt sind, geschehen je nach Version des Versionskontroll-Plugin verschiedene Dinge: Es geschieht nichts, neue Daten werden in Ihrer lokalen Datei zusammengeführt oder Ihre Änderungen werden überschrieben.

Dieser Befehl funktioniert ähnlich wie der Befehl "Abrufen", doch wird das Dialogfeld "Versionskontrolle - Abrufen" nicht angezeigt. Sie können daher keine erweiterten Abrufoptionen definieren.

Beachten Sie, dass dieser Befehl bei Ausführung an einem Ordner automatisch eine rekursive Operation "Aktuellste Version holen" ausführt, d.h. alle anderen Dateien, die sich in der Pakethierarchie unterhalb der aktuellen befinden, sind davon betroffen.

So rufen Sie die neueste Version einer Datei ab:

1. Wählen Sie in der Modell-Struktur die Dateien aus, für die Sie die neueste Version holen möchten.
2. Wählen Sie **Projekt** | **Versionskontrolle** | **Aktuellste Version holen**.

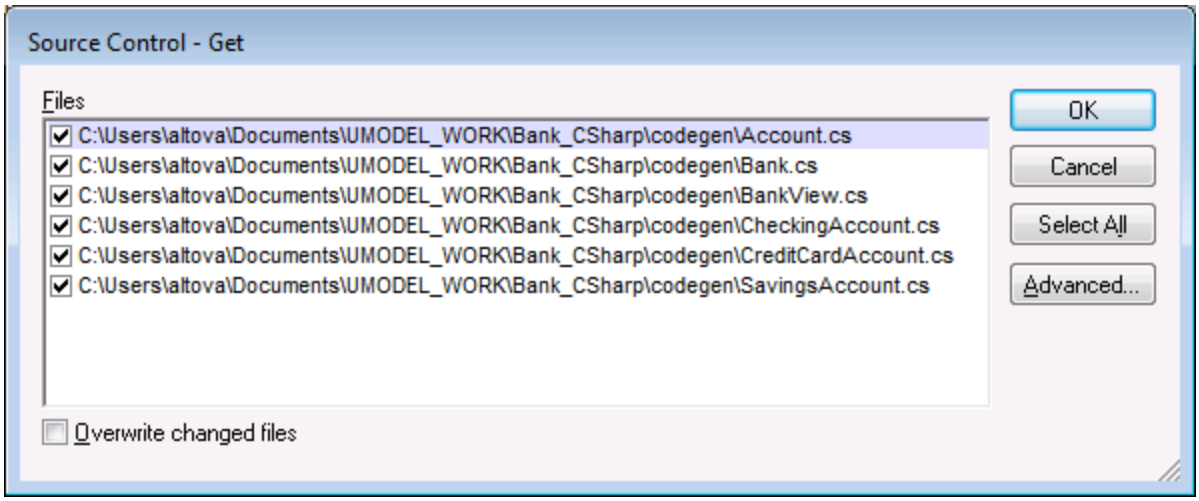
14.3.4 Abrufen

Ruft eine schreibgeschützte Kopie der ausgewählten Dateien ab und platziert Sie in den Arbeitsordner. Standardmäßig sind die Dateien nicht zur Bearbeitung ausgecheckt.

Verwenden von "Abrufen":

- Wählen Sie in der Modell-Struktur die gewünschten Dateien aus.

- Wählen Sie den Befehl **Projekt** | **Versionskontrolle** | **Abrufen.**

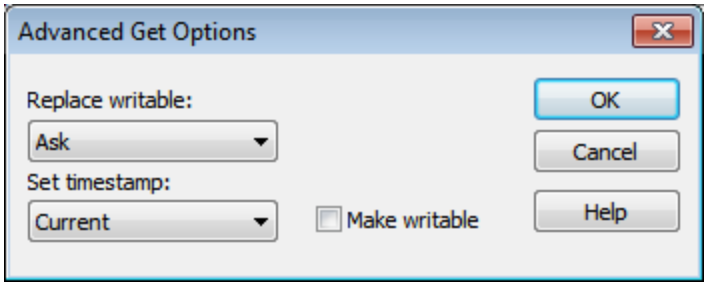


Geänderte Dateien überschreiben
Überschreibt lokal geänderte Dateien mit jenen aus der Versionskontrolldatenbank.

Alle auswählen
Wählt alle Dateien im Listenfeld aus.

Erweitert

Mit Hilfe dieser Option können Sie die Optionen **Replace writable** und **Set timestamp** in den entsprechenden Auswahllisten auswählen.



Mit dem Kontrollkästchen "Make writable" wird das "read-only" Attribut der abgerufenen Dateien entfernt.

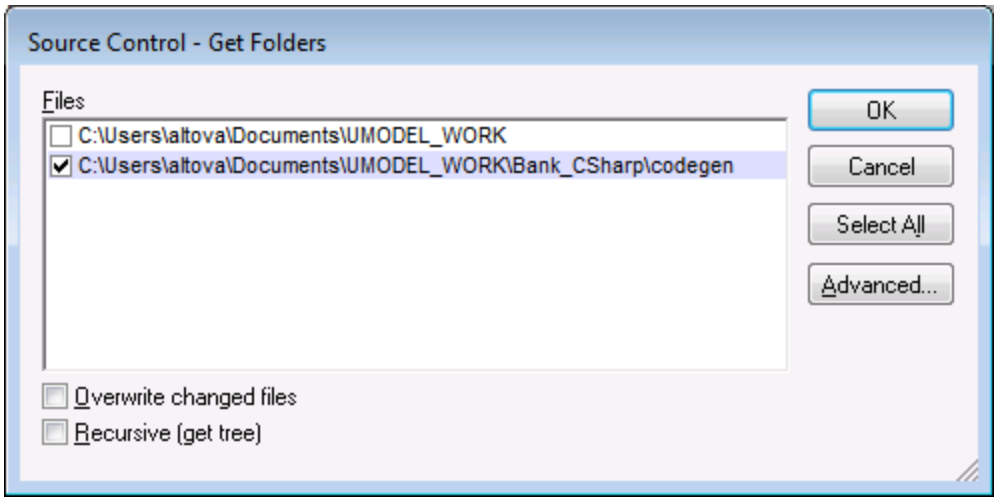
14.3.5 Ordner abrufen

Ruft schreibgeschützte Kopien von Dateien in den ausgewählten Ordnern ab und platziert Sie in den Arbeitsordner. Die Dateien werden standardmäßig nicht zur Bearbeitung ausgecheckt.

© 2016-2022 Altova GmbH

Verwenden von "Ordner abrufen":

- Wählen Sie in der Modell-Struktur die gewünschten Dateien aus.
- Wählen Sie den Befehl **Projekt** | **Versionskontrolle** | **Abrufen.**

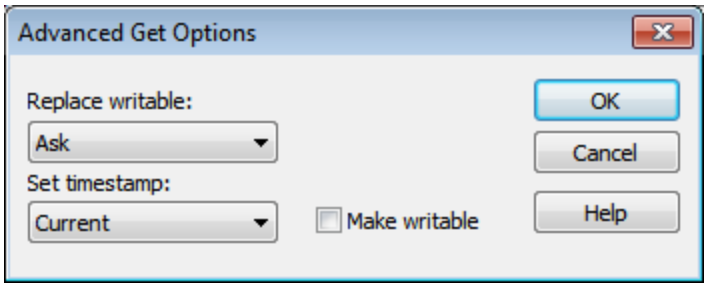


Geänderte Dateien überschreiben
Überschreibt Dateien, die lokal geändert wurden, mit den Dateien aus der Versionskontrolldatenbank.

Rekursiv (Struktur abrufen)
Ruft alle Dateien aus der Ordnerstruktur ab, die sich unterhalb des ausgewählten Ordners befinden.

Erweitert

Mit Hilfe dieser Option können Sie die Optionen **Replace writable** und **Set timestamp** in den entsprechenden Auswahllisten auswählen.



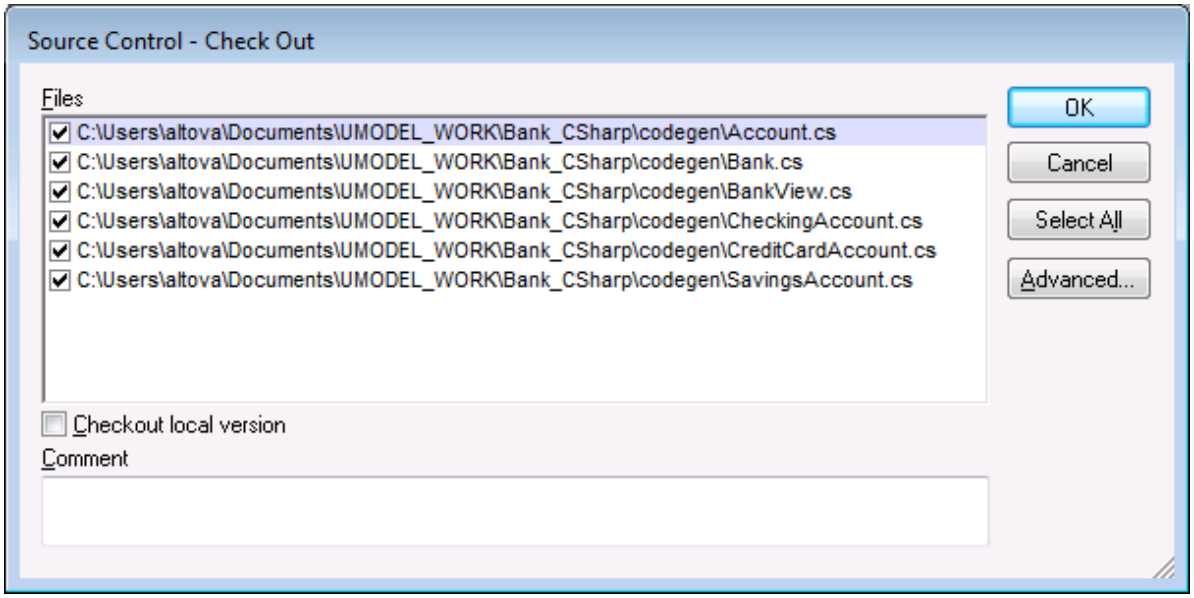
Mit dem Kontrollkästchen "Make writable" wird das "read-only" Attribut der abgerufenen Dateien entfernt.

14.3.6 Auschecken

Mit diesem Befehl checken Sie die aktuellste Version der ausgewählten Datei(en) aus und platzieren beschreibbare Kopien davon in das Arbeitsverzeichnis. Die Dateien werden für andere Benutzer als "ausgecheckt" markiert.

So checken Sie Dateien aus:

- Wählen Sie die gewünschte Datei bzw. den gewünschten Ordner in der Modell-Struktur aus.
- Wählen Sie **Projekt** | **Versionskontrolle** | **Auschecken**.



Anmerkung:
Sie können die Anzahl der auszucheckenden Dateien ändern, indem Sie die einzelnen Kontrollkästchen in der Liste der Dateien aktivieren.

Aktivieren Sie die Option **Lokale Version auschecken,** um nur die lokale Version der Dateien, nicht die aus der Versionskontrolldatenbank auszuchecken.

Folgende Objekte können ausgecheckt werden:

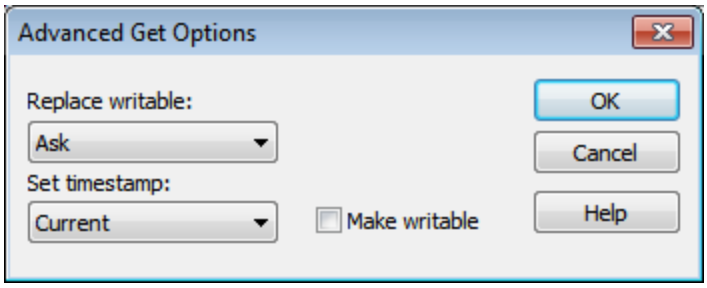
- Einzelne Dateien - klicken Sie in der Modell-Struktur auf die Datei (oder mehrere Dateien mittels Strg + Klick)
- Projektordner - klicken Sie in der Modellstruktur auf den Ordner (oder mehrere Ordner mittels Strg + Klick)

 oder 
Das rote Kontrollhäkchen zeigt an, dass die Datei/der Ordner ausgecheckt ist.#

© 2016-2022 Altova GmbH

Erweitert

Mit Hilfe dieser Option können Sie die Optionen **Replace writable** und **Set timestamp** in den entsprechenden Auswahllisten auswählen.



Mit dem Kontrollkästchen "Make writable" wird das "read-only" Attribut der abgerufenen Dateien entfernt.

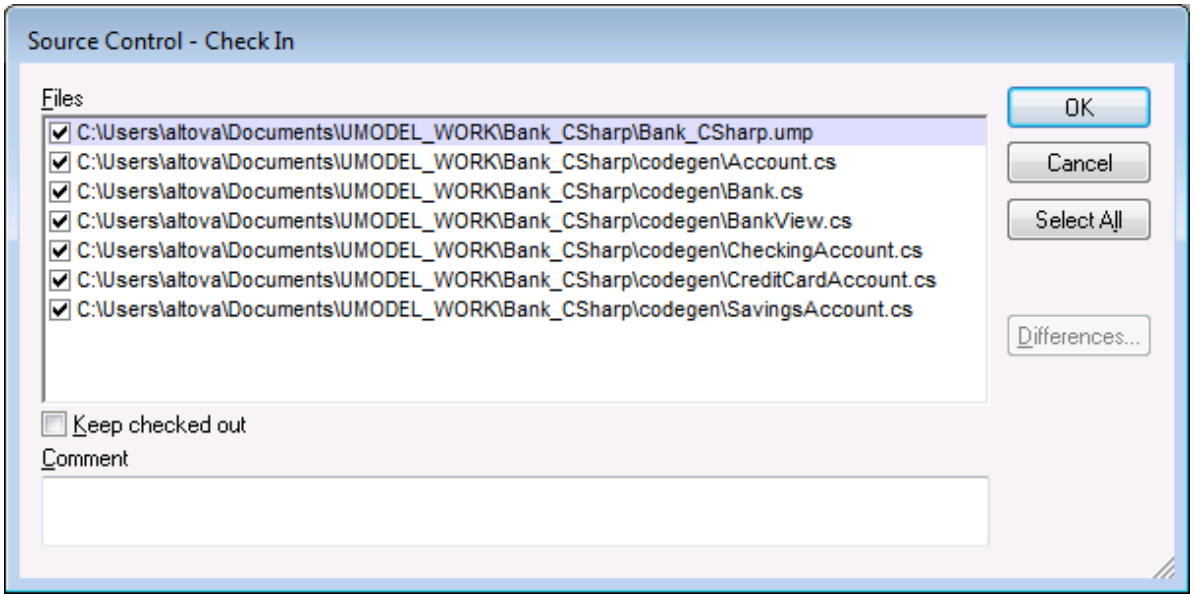
14.3.7 Einchecken

Mit diesem Befehl checken Sie zuvor ausgecheckte Dateien, d.h. Ihre lokal aktualisierten Dateien, wieder ein und stellen die Dateien zurück in das Versionskontrollprojekt.

So checken Sie Dateien ein:

- Wählen Sie die gewünschten Dateien in der Modell-Struktur aus.
- Wählen Sie **Projekt** | **Versionskontrolle** | **Einchecken**.

Kürzel: Rechtsklicken Sie im Projektfenster auf ein ausgechecktes Objekt, und wählen Sie "Einchecken" im Kontextmenü.

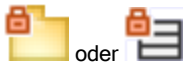


Anmerkung:

Sie können die Anzahl der einzucheckenden Dateien ändern, indem Sie die einzelnen Kontrollkästchen in der Liste der Dateien aktivieren.

Folgende Objekte können eingecheckt werden:

- Einzelne Dateien - klicken Sie in der Modell-Struktur auf die Datei (oder mehrere Dateien mittels Strg + Klick)
- Ordner - klicken Sie in der Modellstruktur auf die Ordner (mittels Strg + Klick)

 oder

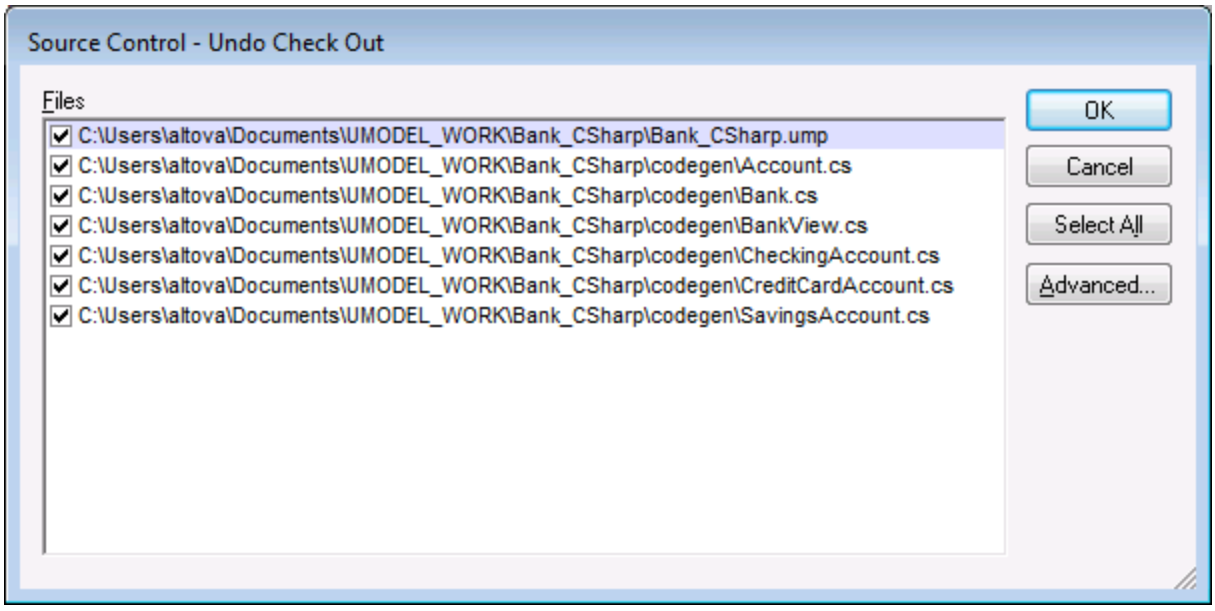
Das Vorhängeschloss-Symbol zeigt an, dass sich die Datei/der Ordner **unter Versionskontrolle** befindet, aber derzeit nicht ausgecheckt ist.

14.3.8 Auschecken rückgängig...

Mit diesem Befehl werden Änderungen in zuvor ausgecheckten Dateien, also lokal aktualisierten Dateien **verworfen**. Es werden die alten Dateien aus der Versionskontrolldatenbank beibehalten.

So machen Sie das Auschecken rückgängig:

- Wählen Sie die Dateien in der Modellstruktur aus
- Wählen Sie **Projekt | Versionskontrolle | Auschecken rückgängig**.



Anmerkung:

Sie können die Anzahl der Dateien ändern, indem Sie die einzelnen Kontrollkästchen in der Liste der Dateien aktivieren.

Die Option "Auschecken rückgängig" kann auf die folgenden Dateien angewendet werden:

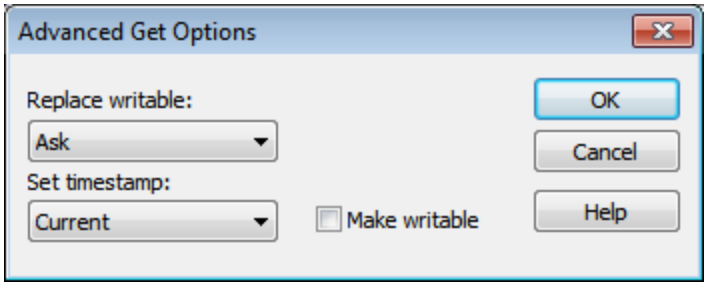
- Einzelne Dateien - klicken Sie in der Modell-Struktur auf die Datei (oder mehrere Dateien mittels Strg + Klick)

© 2016-2022 Altova GmbH

- Ordner - klicken Sie in der Modellstruktur auf die Ordner (mittels Strg + Klick)

Erweitert

Mit Hilfe dieser Option können Sie die Optionen **Replace writable** und **Set timestamp** in den entsprechenden Auswahllisten auswählen.



Mit dem Kontrollkästchen "Make writable" wird das "read-only" Attribut der abgerufenen Dateien entfernt.

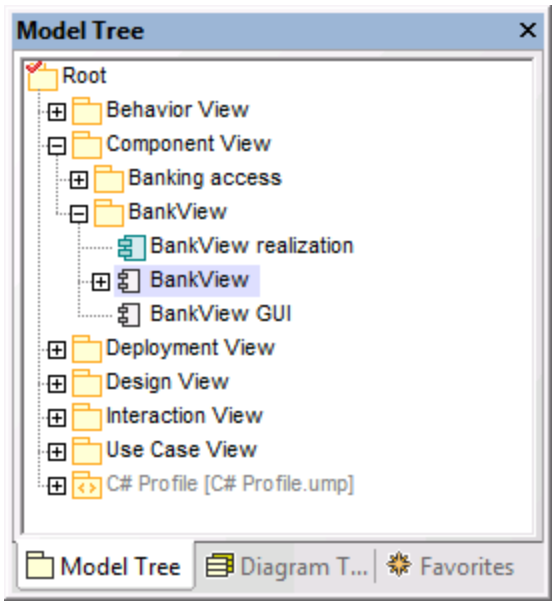
14.3.9 Zu Versionskontrolle hinzufügen

Fügt die ausgewählten Dateien oder Ordner zur Versionskontrolldatenbank hinzu und stellt sie unter Versionskontrolle. Wenn Sie ein neues UModel-Projekt hinzufügen, werden Sie aufgefordert den Arbeitsbereichsordner und den Pfad anzugeben, unter dem das Projekt gespeichert werden soll.

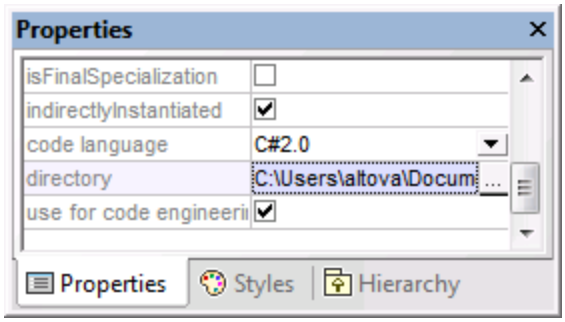
Nachdem Sie die UModel-Projektdatei (*.**ump**) unter Versionskontrolle gestellt haben, können Sie anschließend die beim Code Engineering erzeugten **Codedateien** ebenfalls zur Versionskontrolle hinzufügen. Bitte beachten Sie, dass die generierten Codedateien und das UModel-Projekt in das oder unterhalb desselben SourceSafe-Arbeitsverzeichnisses gelegt werden müssen. Das in diesem Abschnitt verwendete Arbeitsverzeichnis ist **c:\temp\ssc\Bank_CSharp**.

So fügen Sie mit UModel generierte Codedateien zur Versionskontrolle hinzu:

1. Erweitern Sie in der Modell-Struktur den Ordner "Component View" und navigieren Sie zur Komponente "BankView".

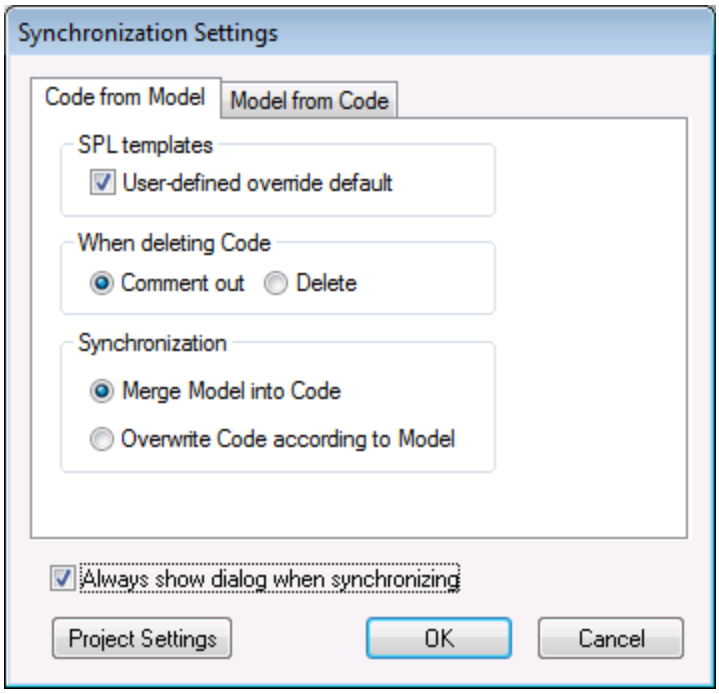


2. Klicken Sie auf die BankView-Komponente und klicken Sie im Fenster "Eigenschaften" auf das Durchsuchen-Symbol neben dem Feld "Verzeichnis".

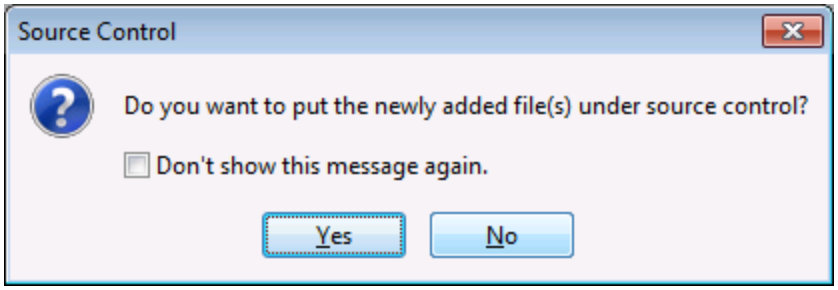


3. Ändern Sie das Code Engineering-Verzeichnis z.B. in **C:\Users\Altova\Documents\UMODEL_WORK\codegen**.
4. Wählen Sie den Menübefehl **Projekt | Merge Programmcode aus UModel-Projekt**.
5. Ändern Sie gegebenenfalls die Synchronisierungseinstellungen und bestätigen Sie mit OK.

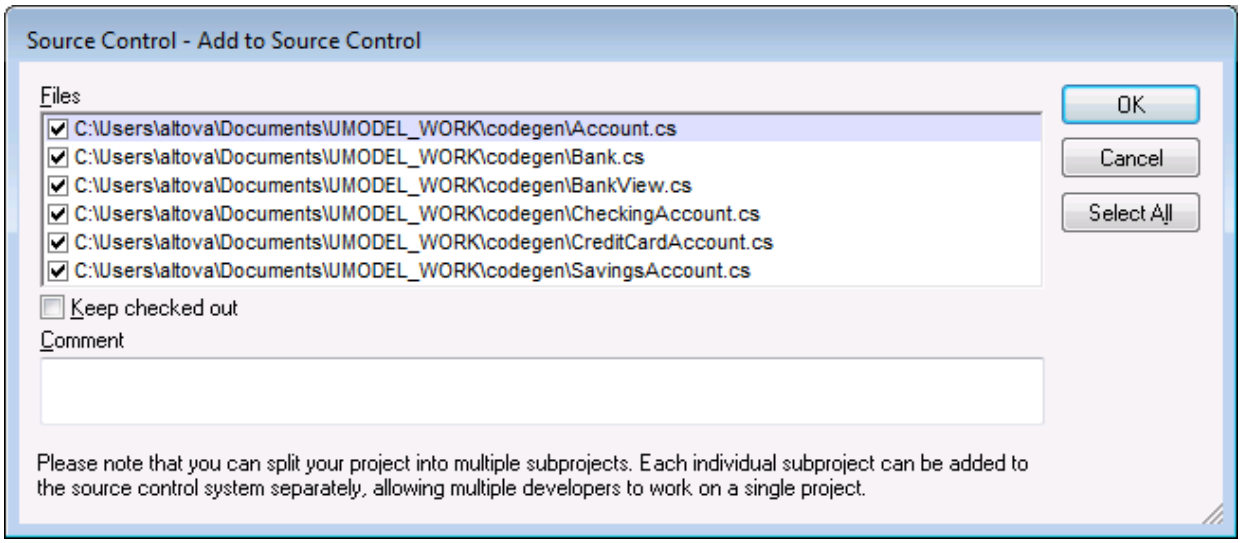
© 2016-2022 Altova GmbH



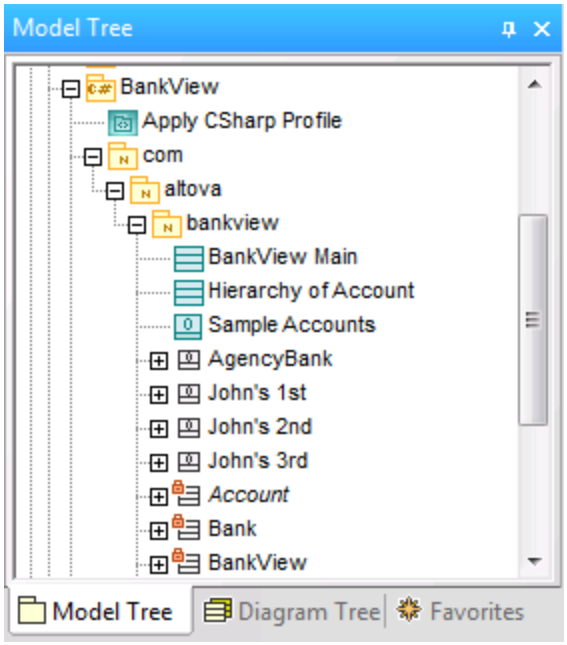
Im Meldungsfenster wird der Code aus dem Projektvorgang angezeigt.
Es erscheint ein Meldungsfeld, in dem Sie gefragt werden, ob Sie die neu erstellten Dateien unter Versionskontrolle stellen möchten.



6. Klicken Sie auf Ja.
7. Daraufhin wird das Dialogfeld "Zu Versionskontrolle hinzufügen" geöffnet, in dem Sie die Dateien auswählen können, die unter Versionskontrolle gestellt werden sollen.



8. Klicken Sie auf OK, sobald Sie die gewünschten Dateien ausgewählt haben.
   Neben den einzelnen unter Versionskontrolle gestellten Klassen/Dateiquellen erscheint nun ein Vorhängeschloss-Symbol.



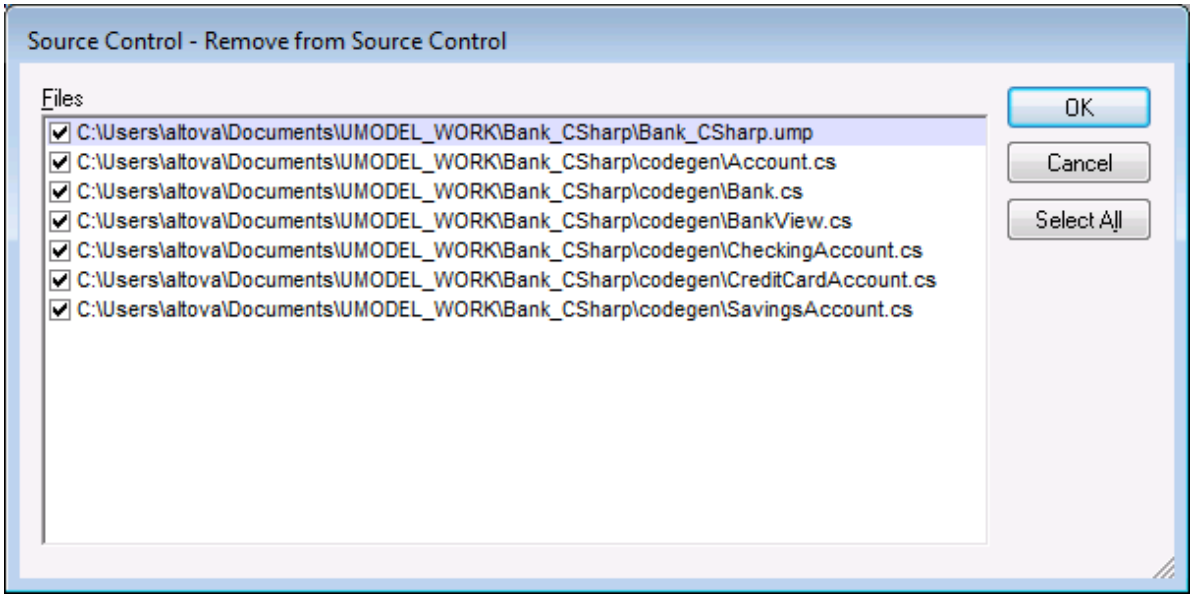
14.3.10 Von Versionskontrolle ausgliedern

Mit diesem Befehl **entfernen** Sie zuvor hinzugefügte Dateien aus dem Versionskontrollprodukt. Diese Dateiarten werden zwar in der Modell-Struktur weiterhin angezeigt, können aber nicht ausgecheckt werden. Mit Hilfe des Befehls "Zu Versionskontrolle hinzufügen" können Sie die Dateien wieder unter Versionskontrolle stellen.

© 2016-2022 Altova GmbH

So entfernen Sie Dateien aus der Versionskontrolle:

- Wählen Sie die gewünschten Dateien in der Modellstruktur aus
- Wählen Sie **Projekt** | **Versionskontrolle** | **Von Versionskontrolle ausgliedern**.



Anmerkung:
Sie können die Anzahl der auszugliedernden Dateien ändern, indem Sie die einzelnen Kontrollkästchen in der Liste der Dateien aktivieren.

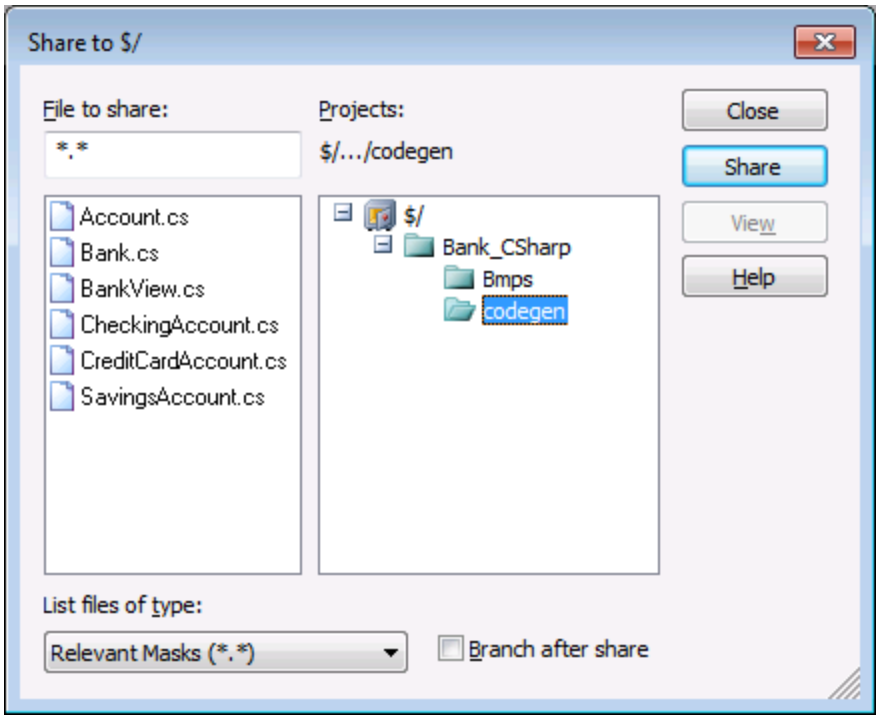
Folgende Objekte könne aus der Versionskontrolle ausgegliedert werden:
- Einzelne Dateien - klicken Sie auf die Datei (oder mehrere Dateien mittels Strg + Klick)
- Ordner - klicken Sie auf das Ordnersymbol

14.3.11 Aus Versionskontrolle freigeben

Mit diesem Befehl werden Dateien aus anderen Projekten/Ordnern im ausgewählten Ordner in die Versionskontrollspeicherschnittstelle in Form von freigegebenen Dateien einem Branch eingefügt.
Um dem Freigeben-Befehl verwenden zu können, müssen Sie für das Projekt, das Sie freigeben möchten, Ein- und Auscheck-Rechte besitzen.

So geben Sie eine Datei aus der Versionskontrolle frei:

1. Wählen Sie den gewünschten Ordner in der Modellstruktur aus und wählen Sie den Befehl **Projekt | Versionskontrolle | Aus Versionskontrolle freigeben,** z.B. die Komponente BankView im Ordner Component View.
2. Wählen Sie im Listenfeld "Projekte" den Projektordner aus, der die Datei enthält.



3. Wählen Sie im Listenfeld "Files to share" die gewünschte Datei aus und klicken Sie auf die Schaltfläche "Freigeben".
   Die Datei wird nun aus der Liste "Files to share" entfernt.
4. Klicken Sie zum Fortfahren auf "Schließen".

Branch after share
Gibt die Datei frei und erstellt einen neuen Branch zum Erstellen einer separaten Version.

14.3.12 Verlauf anzeigen

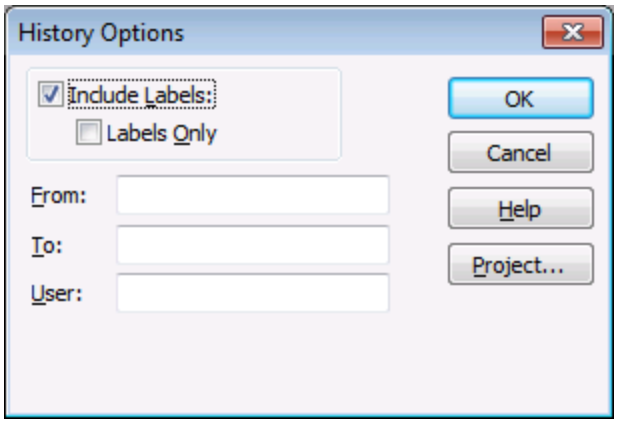
Mit diesem Befehl zeigen Sie den Verlauf einer unter Versionskontrolle stehenden Datei an. Sie können damit eine detaillierte Verlaufsliste anzeigen, genauere Verlaufsinformationen anzeigen, einen Vergleich durchführen oder frühere Versionen der Datei abrufen.

So zeigen Sie den Verlauf einer Datei an:

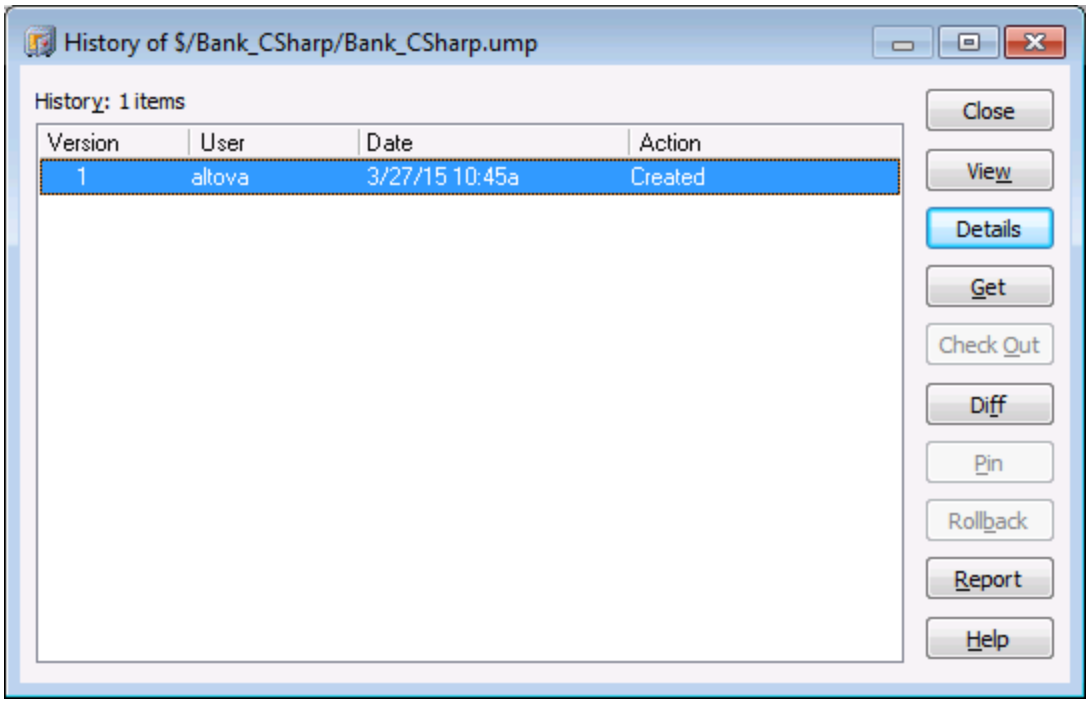
1. Klicken Sie im Fenster "Modell-Struktur" auf die Datei, deren Verlauf Sie ansehen möchten.
2. Wählen Sie die Menüoption **Projekt | Versionskontrolle | Verlauf anzeigen**.

   Es erscheint ein Dialogfeld, in dem Sie nähere Informationen eingeben müssen.

© 2016-2022 Altova GmbH



3. Wählen Sie den gewünschten Eintrag und bestätigen Sie mit OK.



Dieses Dialogfeld bietet verschiedene Möglichkeiten, um bestimmte Versionen der ausgewählten Datei zu vergleichen und abzurufen. Wenn Sie auf einen Eintrag in der Liste doppelklicken, wird das Dialogfeld "Verlauf" für diese Datei geöffnet.

Schließen
Schließt dieses Dialogfeld.

Ansicht
Öffnet ein weiteres Dialogfeld, in dem Sie die Art des Ansichtsprogramms auswählen können, in dem Sie die Datei anzeigen möchten.

Details
Öffnet ein Dialogfeld, in dem Sie die Eigenschaften[725] der gerade aktiven Datei sehen.

Abrufen

Damit können Sie eine der vorherigen Versionen der Datei aus der Versionsliste in das Arbeitsverzeichnis holen.

Auschecken
Damit können Sie die **neueste** Version der Datei auschecken.

Diff
Öffnet das Dialogfeld Vergleichsoptionen[724], in dem Sie die Vergleichsoptionen zur Anzeige der Unterschiede zwischen den beiden Dateiversionen definieren können.

Mit Hilfe von Strg + Klick können Sie zwei Dateiversionen in diesem Fenster markieren. Klicken Sie anschließend auf Diff, um die Unterschiede zwischen den beiden Dateien anzuzeigen.

Pin
Markiert eine Version der Datei bzw. hebt die Markierung auf. Damit können Sie die Dateiversion definieren, die für den Dateivergleich verwendet werden soll.

Rollback
Führt ein Rollback für die ausgewählte Version der Datei durch.

Report
Generiert einen Verlaufsbericht, den Sie an den Drucken, die Datei oder die Zwischenablage senden können.

Hilfe
Öffnet die Online-Hilfe des Versionskontrollanbieters für das Plugin.

14.3.13 Unterschiede anzeigen

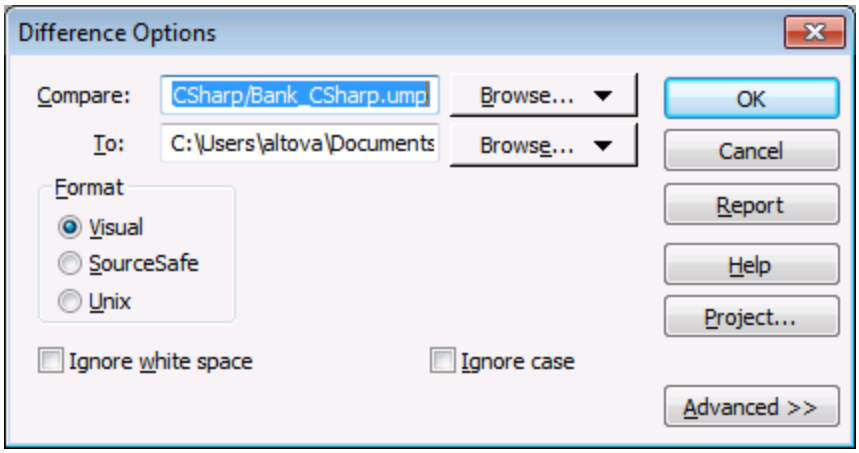
Dieser Befehl ermöglicht die Anzeige von Unterschieden zwischen der in der Versionskontroll-Ablage befindlichen und der gleichnamigen Datei, die Sie **ein-/ausgecheckt** haben.

Wenn Sie eine der Dateien im Dialogfeld "Verlauf" mit einem Pin markiert haben, so wird diese Datei im Textfeld "Vergleichen" verwenden. Sie können mit Hilfe der Durchsuchen-Schaltfläche 2 beliebige Dateien auswählen.

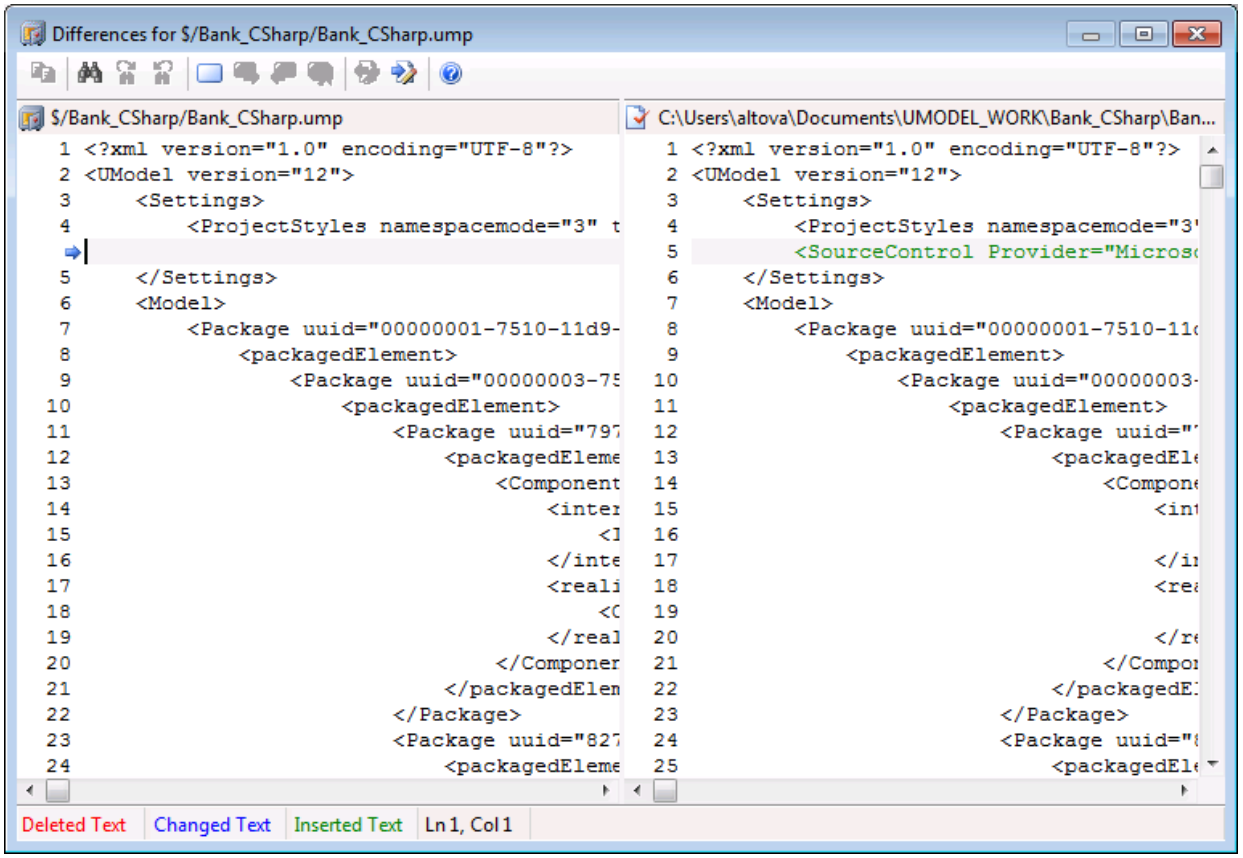
So zeigen Sie die Unterschiede zwischen zwei Dateien an:

1. Klicken Sie in der Modell-Struktur auf eine Datei.
2. Wählen Sie die Menüoption **Projekt | Versionskontrolle | Unterschiede anzeigen.**
   Es erscheint ein Dialogfeld, in dem Sie weitere Informationen eingeben können.

© 2016-2022 Altova GmbH



3. Wählen Sie die gewünschten Einträge und bestätigen Sie mit OK.



Die Unterschiede zwischen beiden Dateien werden in beiden Fenstern farblich markiert (in diesem Beispiel wird MS Source-Safe verwendet).

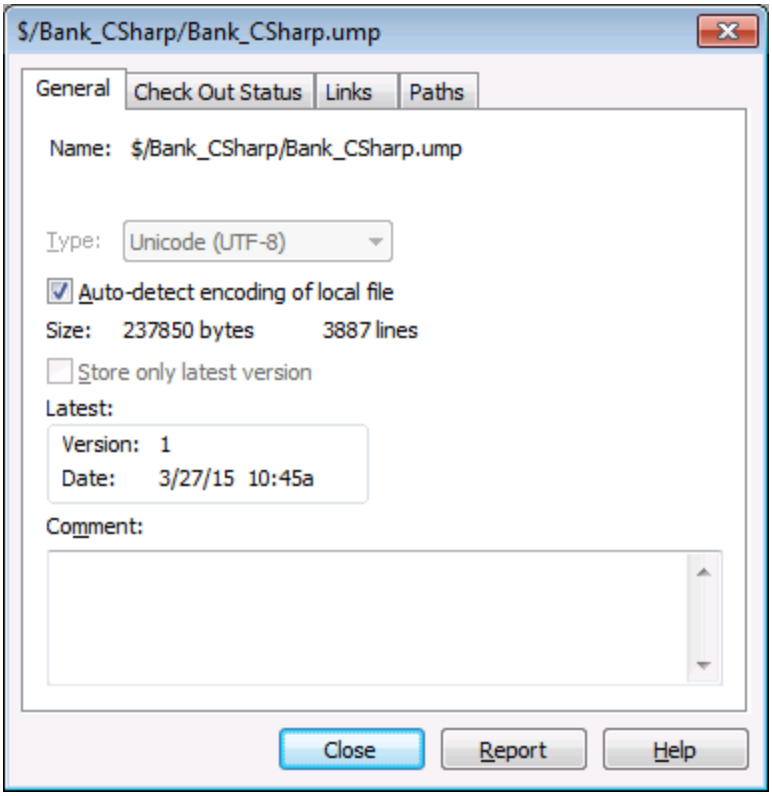
14.3.14 Eigenschaften anzeigen

Mit diesem Befehl können Sie die Eigenschaften der ausgewählten Datei anzeigen. Die Anzeige kann je nach verwendetem Versionskontrollprodukt unterschiedlich sein.

So zeigen Sie die Eigenschaften der aktuell ausgewählten Datei an:

- Wählen Sie **Projekt** | **Versionskontrolle** | **Eigenschaften**.

Dieser Befehl kann jeweils nur an einer einzelnen Datei ausgeführt werden.



14.3.15 Status aktualisieren

Mit diesem Befehl **aktualisieren** Sie den Status aller Projektdateien unabhängig von ihrem derzeitigen Status.

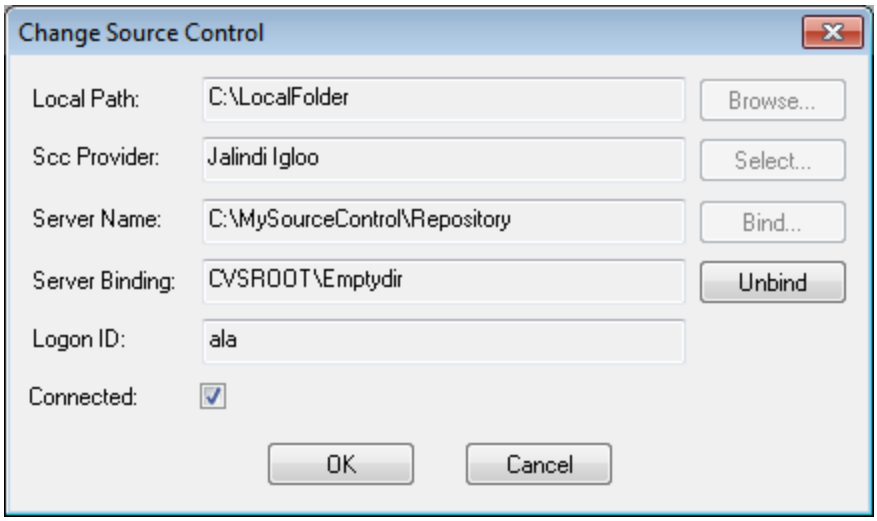
14.3.16 Versionskontrollmanager

Mit diesem Befehl **starten** Sie die ursprüngliche Oberfläche Ihrer Versionskontroll-Software.

14.3.17 Versionskontrolle wechseln

Über dieses Dialogfeld können Sie das Versionskontrollsystem-Binding wechseln. Klicken Sie dazu zuerst auf die Schaltfläche "Bindung aufheben" und anschließend (optional) auf die Schaltfläche "Auswählen", um ein neues Versionskontrollsystem auszuwählen. Klicken Sie anschließend auf die Schaltfläche "Binden", um eine Bindung zu einem neuen Pfad in der Speicherschnittstelle zu erstellen.

© 2016-2022 Altova GmbH

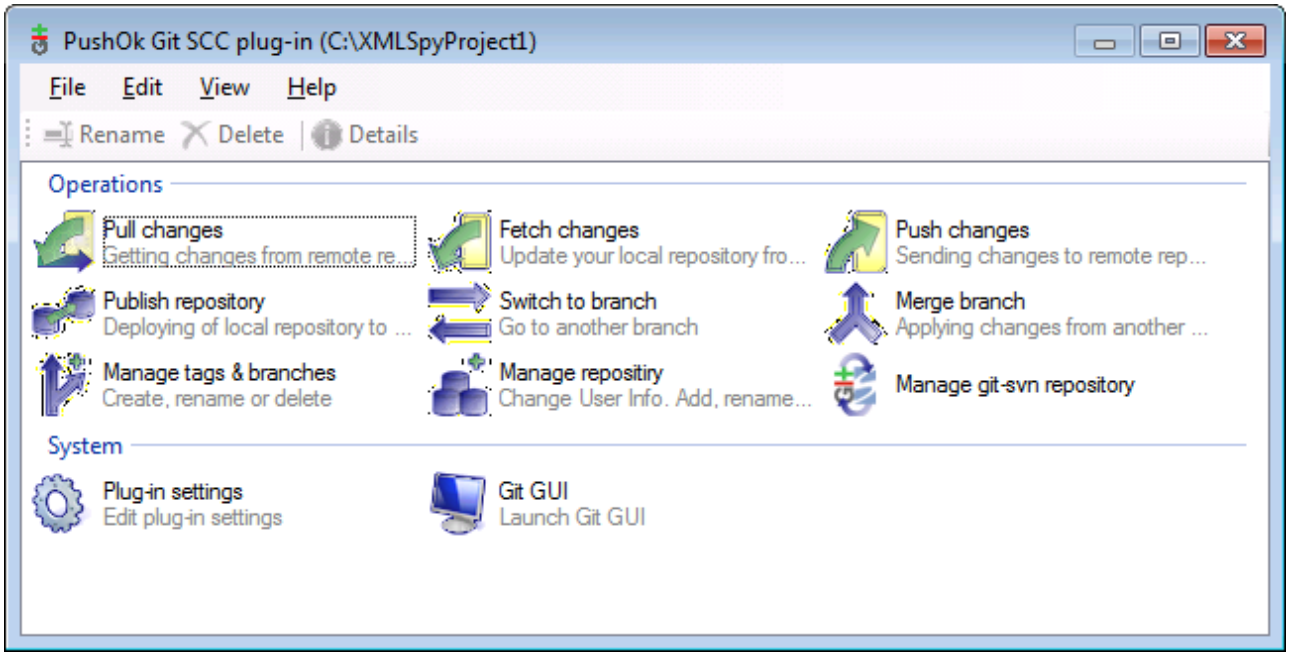

Change Source Control
Local Path:
C:\LocalFolder
Browse...
Scc Provider:
Jalindi Igloo
Select...
Server Name:
C:\MySourceControl\Repository
Bind...
Server Binding:
CVSROOT\Emptydir
Unbind
Logon ID:
ala
Connected:
OK
Cancel

14.4 Versionskontrolle mit Git

Unterstützung für Git als Versionskontrollsystem in UModel steht in Form eines Drittanbieter-Plug-in namens **GIT SCC Plug-in** (http://www.pushok.com/software/git.html) zur Verfügung.

> Zum Zeitpunkt der Verfassung dieser Dokumentation steht das **GIT SCC Plug-in** zum Experimentieren zur Verfügung. Um das Plug-in verwenden zu können, müssen Sie beim Plug-in-Anbieter registriert sein.

Mit Hilfe des GIT SCC Plug-in können Sie über die Befehle im Menü **Projekt | Versionskontrolle** von UModel mit einem Git Repository arbeiten. Beachten Sie, dass die Befehle im Menü **Projekt | Versionskontrolle** von UModel von der Microsoft Source Control Plug-in API (MSSCCI API), für die eine andere Art von Design als von Git verwendet wird, bereitgestellt werden. Daher bildet das Plug-in eine Zwischenschaltung zwischen "Visual Source Safe"-Funktionalitäten und Git-Funktionalitäten. Das bedeutet einerseits, dass ein Befehl wie z.B. **Aktuellste Version holen** für Git eventuell so nicht verwendet werden kann. Andererseits gibt es einige neue Git-spezifische Aktionen, die über das Plug-in im Dialogfeld "Versionskontrollmanager" zur Verfügung gestellt werden (Menü **Projekt | Versionskontrolle | Versionskontrollmanager** von UModel).



Das Dialogfeld "Versionskontrollmanager"

Andere häufig benötigte Versionskontrollbefehle stehen direkt im Menü **Projekt | Versionskontrolle** zur Verfügung.

In den folgenden Abschnitten wird die Anfangskonfiguration des Plug-in sowie der grundlegende Arbeitsablauf beschrieben:

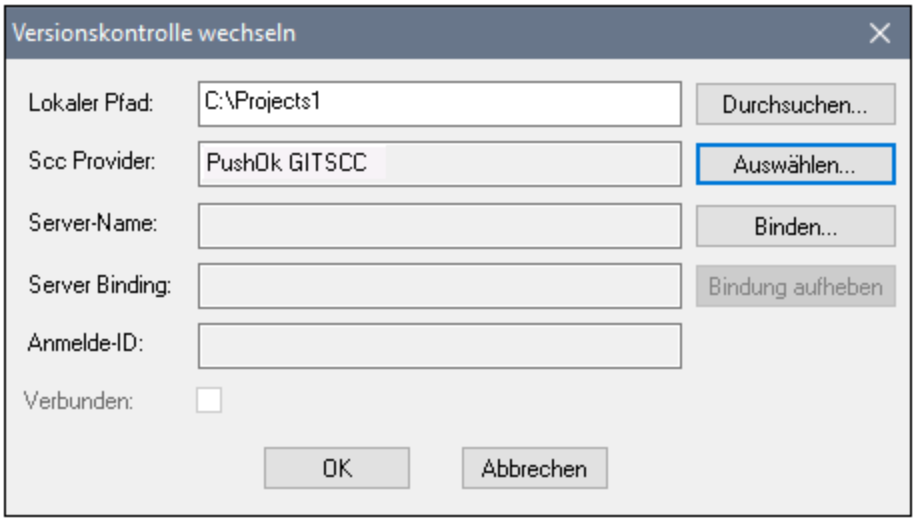
- Aktivieren der Git-Versionskontrolle mit dem Git SCC Plug-in 729
- Hinzufügen eines Projekts zur Git-Versionskontrolle 729
- Klonen eines Projekts anhand der Git-Versionskontrolle 731

© 2016-2022 Altova GmbH

14.4.1 Aktivieren der Git-Versionskontrolle mit dem Git SCC Plug-in

Um die Git-Versionskontrolle mit UModel zu aktivieren, muss zuerst das Drittanbieter-**PushOK GIT SCC Plug-in** installiert, registriert und als Versionskontrollanbieter ausgewählt werden. Dies geschieht folgendermaßen:

1. Laden Sie die Installationsdatei für das Plug-in von der Website des Anbieters (http://www.pushok.com) herunter, starten Sie sie und befolgen Sie die Installationsanweisungen.
2. Klicken Sie im Menü **Projekt** von UModel auf **Versionskontrolle wechseln** und vergewissern Sie sich, dass **PushOk GITSCC** als Versionskontroll-Provider ausgewählt ist. Wenn **Push Ok GITSCC** in der Liste der Provider nicht angezeigt wird, war die Installation des Plug-in wahrscheinlich nicht erfolgreich. Lesen Sie in diesem Fall in der Dokumentation des Anbieters nach, wie Sie das Problem lösen können.



3. Wenn ein Dialogfeld angezeigt wird, in dem Sie aufgefordert werden, das Plug-in zu registrieren, klicken Sie auf **Registrierung** und befolgen Sie die Anweisungen des Assistenten, um die Registrierung abzuschließen.

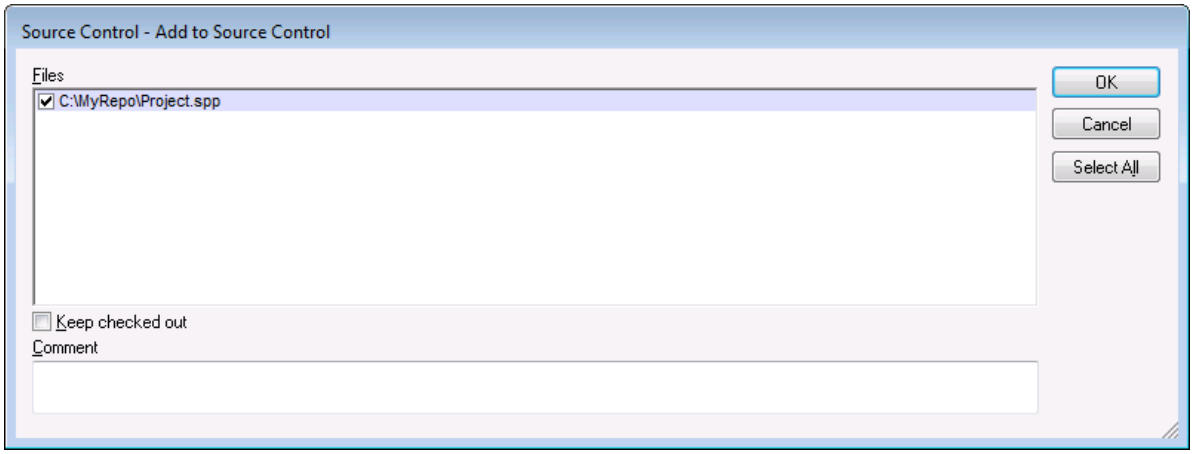
14.4.2 Hinzufügen eines Projekts zur Git-Versionskontrolle

Sie können UModel -Projekte als Git Repositories speichern. Die Struktur der zum Projekt hinzugefügten Dateien oder Ordner würde anschließend der Struktur des Git Repository entsprechen.

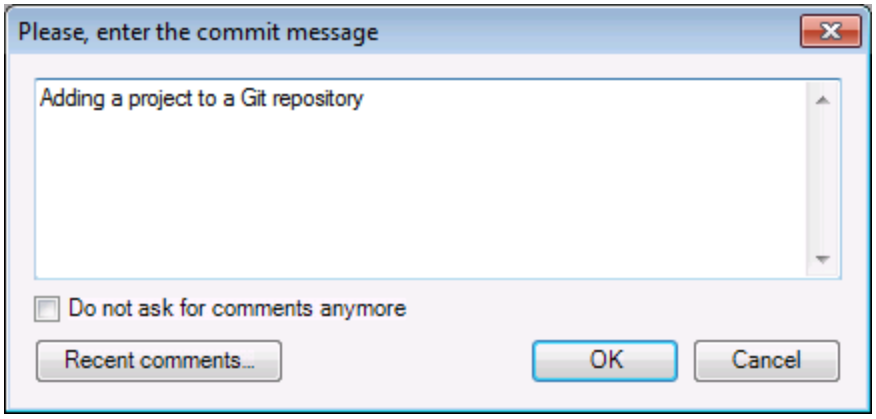
So fügen Sie ein Projekt zu einer Git-Versionskontrolle hinzu:

1. Stellen Sie sicher, dass das **PushOK GIT SCC Plug-in** als Versionskontrollanbieter ausgewählt ist (siehe Aktivieren der Git-Versionskontrolle mit dem Git SCC Plug-in 729).
2. Erstellen Sie ein neues leeres Projekt und vergewissern Sie sich, dass es keine Validierungsfehler aufweist (d.h. dass bei Auswahl des Befehls **Projekt | Projektsyntax überprüfen** keine Fehler oder Warnungen angezeigt werden).
3. Speichern Sie das Projekt in einem lokalen Ordner, z.B. unter `C:\MyRepo\Project.ump`.
4. Klicken Sie im Fenster der **Modell-Struktur** auf den **Root**-Node.

5. Klicken Sie im Menü **Projekt** unter **Versionskontrolle** auf **Zu Versionskontrolle hinzufügen**.

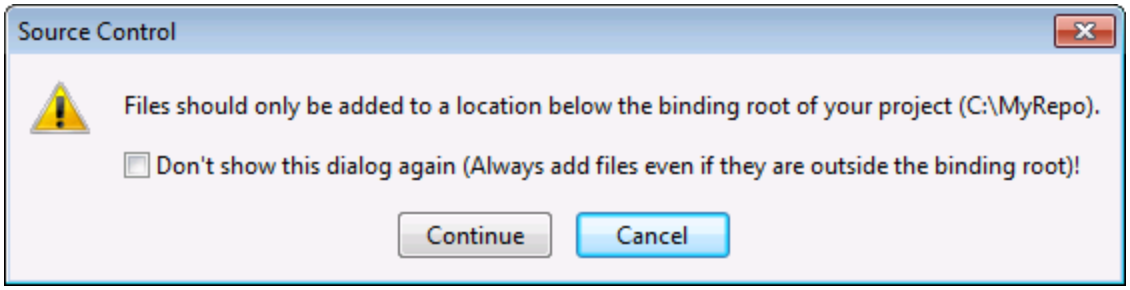


6. Klicken Sie auf **OK**.



7. Geben Sie den Text Ihrer Commit-Meldung ein und klicken Sie auf **OK**.

Sie können jetzt Modellierungselemente (Diagramme, Klassen, Pakete usw.) zu Ihrem Projekt hinzufügen. Beachten Sie, dass alle Projektdateien und -ordner sich unter dem Root-Ordner des Projekts befinden müssen. Wenn das Projekt z.B. im Ordner `C:\MyRepo` angelegt wurde, so sollten nur Dateien unter `C:\MyRepo` zur Projekt hinzugefügt werden. Wenn Sie versuchen, Dateien, die sich außerhalb des Projekt-Root-Ordners befinden, zum Projekt hinzuzufügen, wird eine Warnmeldung angezeigt:

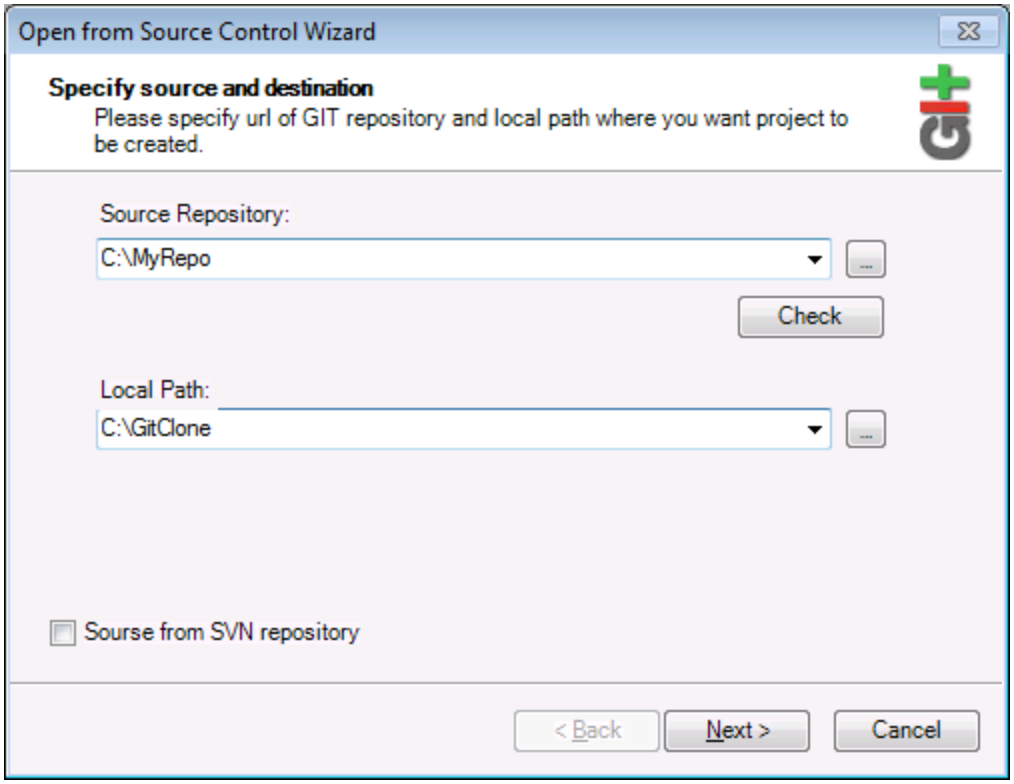


© 2016-2022 Altova GmbH

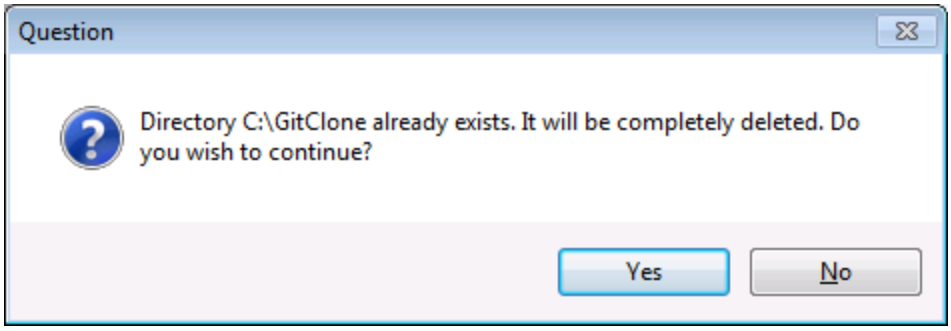
14.4.3 Klonen eines Projekts anhand der Git-Versionskontrolle

Projekte, die bereits zur Git-Versionskontrolle hinzugefügt wurden (siehe Hinzufügen eines Projekts zur Git-Versionskontrolle 729) können folgendermaßen über das Git Repository geöffnet werden:

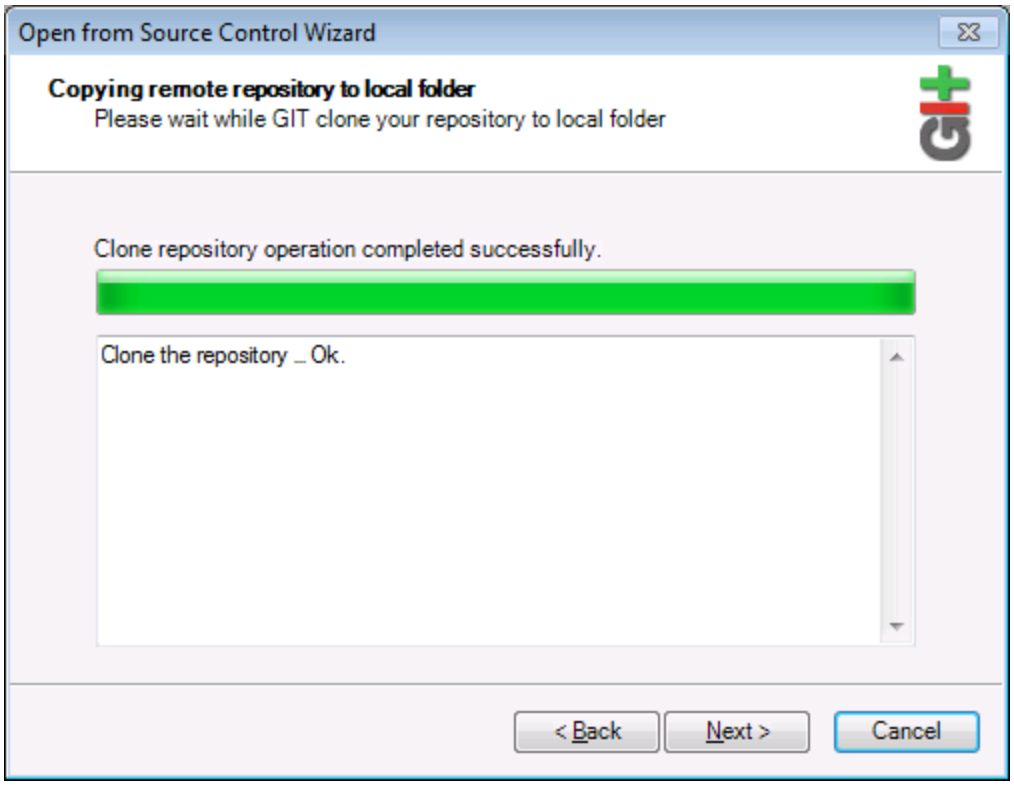
1. Vergewissern Sie sich, dass das **PushOK GIT SCC Plug-in** als Versionskontroll-Provider ausgewählt ist (siehe Aktivieren der Git-Versionskontrolle mit dem Git SCC Plug-in 729).
2. Klicken Sie im Menü **Projekt** auf **Versionskontrolle | Aus Versionskontrolle öffnen**.
3. Geben Sie den Pfad oder die URL des Versionskontroll-Repository ein. Klicken Sie auf **Überprüfen,** um die Gültigkeit des Pfads oder der URL zu überprüfen.



4. Geben Sie unter **Lokaler Pfad** den Pfad zu dem lokalen Ordner ein, in dem das Projekt erstellt werden soll und klicken Sie auf **Weiter**. Wenn der lokale Ordner vorhanden ist, wird (auch wenn er leer ist) das folgende Dialogfeld aufgerufen:



5. Klicken Sie zur Bestätigung auf **Ja** und anschließend auf **Weiter**.



6. Stellen Sie die restlichen Schritte des Assistenten fertig, wie für Ihr Projekt erforderlich.
7. Nach der Fertigstellung wird ein Durchsuchen-Dialogfeld angezeigt, in dem Sie aufgefordert werden, das UModel-Projekt (*.ump)-Datei zu öffnen. Wählen Sie die Projektdatei aus, um den Projektinhalt in UModel zu laden.

© 2016-2022 Altova GmbH

15 UModel Diagrammsymbole

Der folgende Abschnitt enthält eine Kurzübersicht über die Symbole, die in den einzelnen Modelldiagrammen zur Verfügung stehen.

Die Symbole sind in zwei Gruppen unterteilt:

- **Hinzufügen** - Zeigt eine Liste von Elementen an, die zum Diagramm hinzugefügt werden können.
- **Beziehung** - Zeigt eine Liste von Beziehungsarten an, die zwischen Elementen im Diagramm erstellt werden können.

15.1 Aktivitätsdiagramm



Hinzufügen

Aktion (Aufrufverhalten-Aktion)
Aktion (Aufrufoperationsaktion)
Ereignisannahmeaktion
Ereignisannahmeaktion (Zeitereignis)
Signalsendeaktion

Verzweigungsknoten (Verzweigung)
Verbindungsknoten
Startknoten
Aktivitätsendknoten
Endknoten für Kontrollflüsse
Parallelisierungsknoten (vertikal)
Parallelisierungsknoten (horizontal)
Synchronisationsknoten
Synchronisationsknoten (horizontal)

Inputpin
Outputpin
Wert-Pin

Objektknoten
Pufferknoten
Datenspeicherknoten
Aktivitätsbereich (horizontal)
Aktivitätsbereich (vertikal)
Aktivitätsbereich 2-dimensional

Kontrollfluss
Objektfluss
Ausnahme-Handler

Aktivität
Aktivitätsparameterknoten
StrukturierterAktivitätsknoten
Mengenverarbeitungsbereich
Erweiterungsknoten
Unterbrechungsbereich

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.2 Klassendiagramm



Beziehung

Assoziation
Aggregation
Komposition
Assoziationsklasse
Abhängigkeit
Verwendung
Schnittstellenrealisierung
Generalisierung

Hinzufügen

Paket
Klasse
Schnittstelle
Enumeration
Datentyp
Primitivtyp
Profil
Stereotyp
Profilzuweisung
Instanzspezifikation

Anmerkung
Anmerkung verknüpfen

15.3 Kommunikationsdiagramm



Hinzufügen

Lebenslinie
Nachricht (Aufruf)
Nachricht (Antwort)
Nachricht (Erstellung)
Nachricht (Löschung)

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.4 Kompositionsstrukturdiagramm



Hinzufügen

Kollaboration
Kollaborationsanwendung
Part (Eigenschaft)
Klasse
Schnittstelle
Port

Beziehung

Konnektor
Abhängigkeit (Rollenbindung)
Schnittstellenrealisierung
Verwendung

Anmerkung
Anmerkung verknüpfen

15.5 Komponentendiagramm



Hinzufügen

Paket
Schnittstelle
Klasse
Komponente
Artefakt

Beziehung

Realisierung
Schnittstellenrealisierung
Verwendung
Abhängigkeit

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.6 Deployment-Diagramm



Hinzufügen

Paket
Komponente
Artefakt
Knoten
Gerät
Ausführungsumgebung

Beziehung

Manifestation
Deployment
Assoziation
Generalisierung
Abhängigkeit

Anmerkung
Anmerkung verknüpfen

15.7 Interaktionsübersichtsdiagramm



Hinzufügen

Aufrufverhalten - Aktion (Interaktion)
Aufrufverhalten - Aktion (Interaktionsverwendung)
Verzweigungsknoten
Verbindungsknoten
Startknoten
Aktivitätsendknoten
Parallelisierungsknoten
Parallelisierungsknoten (Horizontal)
Synchronisationsknoten
Synchronisationsknoten (Horizontal)
Zeitdauerbedingung

Beziehung

Kontrollfluss

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.8 Objektdiagramm



Beziehung

Assoziation
Assoziationsklasse
Abhängigkeit
Verwendung
Schnittstellenrealisierung
Generalisierung

Hinzufügen

Paket
Klasse
Schnittstelle
Enumeration
Datentyp
Primitivtyp
Instanzspezifikation

Anmerkung
Anmerkung verknüpfen

15.9 Paketdiagramm



Hinzufügen

Paket
Profil

Beziehung

Abhängigkeit
Paketimport
Paketmerge
Profilzuweisung

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.10 Profildiagramm



Hinzufügen

Profil
Stereotyp

Beziehung

Generalisierung
Profilzuweisung
Paketimport
Elementimport

Anmerkung
Anmerkung verknüpfen

15.11 Protokoll-Zustandsdiagramm



Hinzufügen

Einfacher Zustand
Zusammengesetzter Zustand
Orthogonaler Zustand
Unterautomatenzustand

Endzustand
Anfangszustand

Eintrittspunkt
Austrittspunkt
Entscheidung
Kreuzung
Beendung
Gabelung
Gabelung (horizontal)
Vereinigung
Vereinigung (horizontal)
Verbindungspunktreferenz

Beziehung

Protocol Transition

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.12 Sequenzdiagramm



Hinzufügen

Lebenslinie
Combined Fragment
Combined Fragment (Alternativen)
Combined Fragment (Loop)
Interaktionsverwendung
Gate
Zustandsinvariante
Zeitdauerbedingung
Zeitbedingung

Nachricht (Aufruf)
Nachricht (Antwort)
Nachricht (Erstellung)
Nachricht (Löschung)

Asynchrone Nachricht (Aufruf)
Asynchrone Nachricht (Antwort)
Asynchrone Nachricht (Löschung)

Anmerkung
Anmerkung verknüpfen

Keine Nachrichtennummerierung
Einfache Nachrichtennummerierung
Hierarchische Nachrichtennummerierung

Verschieben zusammengehöriger Nachrichten ein/aus
Automatische Erstellung von Antworten auf Nachrichten (Call) ein/aus
Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten

15.13 Zustandsdiagramm



Hinzufügen

Einfacher Zustand
Zusammengesetzter Zustand
Orthogonaler Zustand
Unterautomatenzustand

Anfangszustand
Endzustand
Eintrittspunkt
Austrittspunkt
Entscheidung
Kreuzung
Beendung
Gabelung
Gabelung (horizontal)
Vereinigung
Vereinigung (horizontal)
Deep history
Shallow history
Verbindungspunktreferenz

Beziehung

Transition

Anmerkung
Anmerkung verknüpfen

Automatische Erstellung von Operationen in der Zielkomponente durch Eingabe von Operationsnamen ein-/ausschalten

© 2016-2022 Altova GmbH

15.14 Zeitverlaufsdiagramm



Hinzufügen

Lebenslinie (Zustand/Bedingung)
Lebenslinie (Allgemeiner Wert)
Tick-Symbol
Auslösendes Ereignis
Zeitdauerbedingung
Zeitbedingung

Nachricht (Aufruf)
Nachricht (Antwort)
Asynchrone Nachricht (Aufruf)

Anmerkung
Anmerkung verknüpfen

15.15 Use Case-Diagramm



Hinzufügen

Paket
Akteur
Use Case

Beziehung

Assoziation
Generalisierung
Include
Extend

Anmerkung
Anmerkung verknüpfen

© 2016-2022 Altova GmbH

15.16 XML-Schema-Diagramm



Hinzufügen

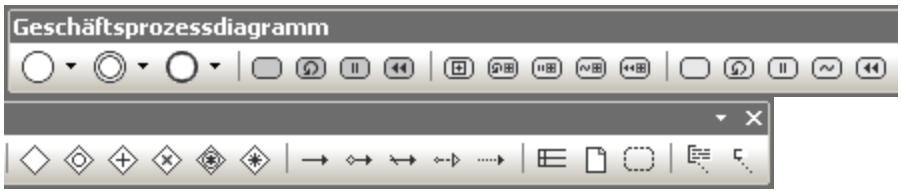
XSD Target Namespace
XSD Schema
XSD Element (global)
XSD Group
XSD ComplexType
XSD ComplexType (simpleContent)
XSD SimpleType
XSD List
XSD Union
XSD Enumeration
XSD Attribute
XSD AttributeGroup
XSD Notation
XSD Import

Beziehung

XSD Include
XSD Redefine
XSD Restriction
XSD Extension
XSD Substitution

Anmerkung
Anmerkung verknüpfen

15.17 Business Process Modeling Notation



Hinzufügen

Startereignis
Zwischenereignis
Endereignis

Task
Schleifen-Task
Mehrfachinstanz-Task
Kompensations-Task

Eingeklappter Unterprozess
Eingeklappter Schleifenunterprozess
Eingeklappter Mehrfachinstanz-Unterprozess
Eingeklappter Ad Hoc-Prozess
Eingeklappter Kompensations-Unterprozess

Aufgeklappter Unterprozess
Aufgeklappter Schleifenunterprozess
Aufgeklappter Mehrfachinstanz-Unterprozess
Aufgeklappter Ad Hoc-Prozess
Aufgeklappter Kompensations-Unterprozess

Gateway
Inklusiver Gateway (OR)
Paralleler Gateway (AND)
Datenbasierter exklusiver Gateway (XOR)
Ereignisbasierter exklusiver Gateway (XOR)
Komplexer Gateway (Entscheidung/Zusammenfluss)

Beziehung

Sequenzfluss
Bedingter Fluss
Standardfluss
Nachrichtenfluss
Assoziation

Pool
Datenobjekt
Gruppe

© 2016-2022 Altova GmbH

Textannotation
Annotationsassoziation

15.18 Business Process Modeling Notation 2.0



Hinzufügen

Startereignis
Empfangsereignis
Sendeereignis
Endereignis

Task
Aufgeklappter Unterprozess
Eingeklappter Unterprozess
Aufruf-Aktivität
Gateway

Beziehung

Sequenzfluss
Standardsequenzfluss
Bedingter Sequenzfluss
Nachrichtenfluss
Assoziation

Pool
Gruppe
Datenobjekt
Datenoutput
Dateninput
Listen-Datenobjekt
Datenspeicher
Nachricht

Textannotation
Annotationsassoziation

© 2016-2022 Altova GmbH

15.19 Datenbankmodellierung



Hinzufügen

Tabelle
CheckConstraint
PrimaryKey
ForeignKey
UniqueKey
Index

Beziehung

Datenbankbeziehungs-Assoziation
Datenbankbeziehung mit Attributen

16 Menüreferenz

Im folgenden Abschnitt sind alle Menüs und Menüoptionen in UModel mit einer kurzen Beschreibung dazu aufgelistet.

© 2016-2022 Altova GmbH

16.1 Datei

Neu

Löscht das Diagrammregister, falls ein früheres Projekt vorhanden ist, und erstellt ein neues UModel-Projekt.

Öffnen

Öffnet ein zuvor definiertes Modellierungsprojekt. Wählen Sie im Dialogfeld "Öffnen" eine zuvor gespeicherte Projektdatei (***.ump**) aus. Siehe Erstellen, Öffnen und Speichern von Projekten[163] und Öffnen von Projekten über eine URL[164].

Neu laden

Dient zum Neuladen des aktuellen Projekts und zum Speichern oder Verwerfen der Änderungen, die seit dem letzten Öffnen der Projektdatei vorgenommen wurden.

Speichern

Speichert das aktuell aktive Modellierungsprojekt unter dem aktiven Dateinamen.

Speichern unter

Speichert das aktuell aktive Modellierungsprojekt unter einem anderen Namen bzw. dient zum Speichern des Projekts unter einem Namen, wenn Sie das Projekt zum ersten Mal speichern.

Kopie speichern unter

Damit können Sie eine Kopie des aktuell aktiven UModel-Projekts unter einem anderen Dateinamen speichern.

Diagramm als Bild speichern

Öffnet das Dialogfeld "Speichern unter..." und speichert das aktuell aktive Diagramm als .png-Datei. Es können auch sehr große .png-Dateien im Gigabyte-Bereich gespeichert werden.

Alle Diagramme als Bilder speichern

Speichert alle Diagramme des derzeit aktiven Projekts als .png-Dateien.

Aus XMI-Datei importieren

Importiert eine zuvor exportierte XMI-Datei. Wenn die Datei mit UModel erzeugt wurde, werden alle Erweiterungen usw. beibehalten.

In XMI-Datei exportieren

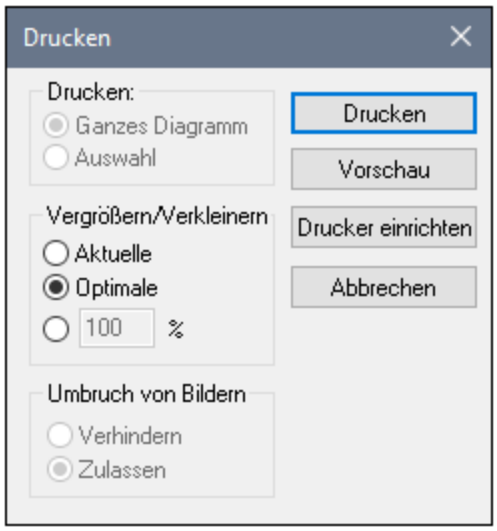
Exportiert das Modell als XMI-Datei. Sie können die UML-Version sowie die zu exportierenden IDs auswählen. Nähere Informationen siehe XMI - XML Metadata Interchange[663].

Als Mail senden

Öffnet Ihre Standard-Mailapplikation und fügt das aktuelle UModel Projekt als Anhang ein.

Drucken

Öffnet das Dialogfeld "Drucken", über das Sie das aktuelle Diagramm (oder eine Auswahl aus dem Diagramm) drucken können.



Bei Auswahl der Option **Aktuelle** wird der aktuell definierte Zoomfaktor des Modellierungsprojekts beibehalten und die Gruppe "Umbruch von Bildern" wird aktiviert. Bei Auswahl von **Optimale** wird das Modellierungsprojekt auf Seitengröße vergrößert/verkleinert. Sie können auch einen numerischen Zoom-Faktor angeben. Bei Auswahl der Option **Verhindern** wird verhindert, dass Modellelemente über Seiten hinweg umbrochen werden. Sie werden auf einer Seite angezeigt.

Alle Diagramme drucken

Öffnet das Dialogfeld "Drucken" und druckt alle in der aktuellen Projektdatei enthaltenen UML-Diagramme.

Druckvorschau

Öffnet dasselbe Druckdialogfeld mit denselben Einstellungen wie oben beschrieben.

Druckereinrichtung

Öffnet das Dialogfeld "Druckereinrichtung", in dem Sie einstellen können, welchen Drucker und welche Papiereinstellungen Sie verwenden möchten.

Letzte Dateien

In diesem Abschnitt des Menüs **Datei** werden bis zu vier zuletzt verwendete Dateien aufgelistet.

Beenden

Mit dem Befehl **Beenden** wird UModel beendet. Wenn eine Ihrer aktuellen Dateien nicht gespeicherte Änderungen enthält, fordert Sie UModel auf, die Änderungen zu speichern.

© 2016-2022 Altova GmbH

16.2 Bearbeiten

Rückgängig 

UModel gestattet eine unbegrenzte Anzahl an Rückgängig-Schritten, sodass Sie Ihren Modellierungsvorgang Schritt für Schritt zurückverfolgen können.

Wiederherstellen 

Mit Hilfe des Befehls "Wiederherstellen" können Sie zuvor rückgängig gemachte Befehle wiederholen. Sie können sich innerhalb des Verlaufs der rückgängig gemachten Schritte vorwärts und rückwärts bewegen.

Ausschneiden/Kopieren/Löschen

Diese sind die Windows-Standardtextbearbeitungsbefehle. Sie können diese nicht nur für Text, sondern auch für Modellierungselemente verwenden, siehe Umbenennen, Verschieben und Kopieren von Elementen 116.

Nur in Diagramm einfügen

Fügt einen Link (oder eine "Ansicht") des kopierten Elements zum aktuellen Diagramm nicht aber zur Modell-Struktur hinzu, siehe Umbenennen, Verschieben und Kopieren von Elementen 116.

Nur aus Diagramm löschen

Löscht die ausgewählten Modellelemente aus dem aktuell aktiven Diagramm. Die gelöschten Elemente werden nicht aus dem Modellierungsprojekt gelöscht und stehen auf dem Register "Modell-Struktur" zur Verfügung. Beachten Sie, dass diese Option nicht zum Löschen von Eigenschaften oder Operationen aus einer Klasse verfügbar ist. Eigenschaften oder Operationen können ausgewählt und direkt aus der Klasse gelöscht werden.

Alles markieren

Markiert alle Modellelemente des aktuell aktiven Diagramms. Entspricht der Tastenkombination Strg+A .

Suchen

Dient zum Suchen von bestimmtem Text im aktuellen Fenster, siehe Suchen und Ersetzen von Text 118.

Weitersuchen  F3

Sucht die nächste Instanz desselben Suchstrings im derzeit aktiven Fenster.

Vorheriges suchen (Umschalt+F3)

Sucht die vorherige Instanz desselben Suchstrings auf dem derzeit aktiven Register oder im derzeit aktiven Diagramm.

Ersetzen

Dient zum Suchen und Ersetzen eines beliebigen Modellierungselements im Projekt, siehe Suchen und Ersetzen von Text 118.

Als Bitmap kopieren

Kopiert das derzeit aktive Diagramm in die Zwischenablage, von wo aus Sie es in die Applikation Ihrer Wahl einfügen können.

Auswahl als Bitmap kopieren

Kopiert die aktuell ausgewählten Diagrammelemente in die Zwischenablage, von wo aus Sie sie in die Applikation Ihrer Wahl einfügen können.

© 2016-2022 Altova GmbH

16.3 Projekt

Projektsyntax überprüfen

Überprüft die UModel-Projektsyntax. Siehe Überprüfen der Projektsyntax 183.

Versionskontrolle

Nähere Informationen und Anleitungen zu Versionskontrollservern und Clients finden Sie unter Versionskontrolle 702.

Quellverzeichnis importieren

Öffnet den "Quellverzeichnis importieren"-Assistenten. Ein Beispiel dazu finden Sie unter Reverse Engineering (Code zu Modell) 75.

Quellprojekt importieren

Öffnet den Assistenten "Quellprojekt importieren", siehe Importieren von Quellcode 209.

Binärtypen importieren

Öffnet das Dialogfeld "Binärtypen importieren", über das Sie Java-, C#- und VB-Binärdateien importieren können. Nähere Informationen dazu finden Sie unter Importieren von Java-, C#- und VB-Binärdateien 225.

XML-Schemaverzeichnis importieren

Öffnet das Dialogfeld "XML-Schemaverzeichnis importieren", über das Sie alle XML-Schemas in diesem Verzeichnis und optional dazu alle XML-Schemas in den Unterverzeichnissen dieses Verzeichnisses importieren können.

XML-Schema-Datei importieren

Öffnet das Dialogfeld "XML-Schema-Datei importieren" zum Importieren von Schema-Dateien. Nähere Informationen finden Sie unter XML-Schema-Diagramme 490.

SQL-Datenbank importieren

Ruft das Dialogfeld "Datenbank importieren" auf, über das Sie eine Datenbankstruktur in das Modell importieren können, siehe Importieren von SQL-Datenbanken in UModel 558.

Sequenzdiagramme von Code generieren...

Siehe Generieren mehrerer Sequenzdiagramme 432.

Code von Sequenzdiagrammen generieren

UModel kann Code anhand eines mit mindestens einer Operation verknüpften Sequenzdiagramms generieren. Nähere Informationen dazu finden Sie in diesem Abschnitt 434.

Zustandsautomatencode generieren

Sie können in UModel einen oder mehrere Zustandsautomaten auswählen, in denen Code generiert werden soll. Nähere Informationen finden Sie in diesem Kapitel 386.

Merge Programmcode aus UModel-Projekt / Überschreibe Programmcode aus UModel-Projekt

Aktualisiert Programmcode anhand des Modells (vorausgesetzt Ihr Projekt wurde für das Code Engineering vorbereitet, siehe Generieren von Programmcode 180). Der Name dieses Befehls kann entweder **Merge Programmcode aus UModel-Projekt** oder **Überschreibe Programmcode aus UModel-Projekt** lauten, je nachdem welche Einstellung im Dialogfeld "Synchronisierungseinstellungen" gewählt wurde. Das Dialogfeld "Synchronisierungseinstellungen" wird standardmäßig jedes Mal geöffnet, wenn Sie diesen Befehl aufrufen. Nähere Informationen dazu finden Sie unter Codesynchronisierungseinstellungen 244.

Merge UModel-Projekt aus Programmcode / Überschreibe UModel-Projekt aus Programmcode

Aktualisiert das Modell (das UModel-Projekt) anhand des Programmcodes. Der Name dieses Befehls kann entweder **Merge UModel-Projekt aus Programmcode** oder **Überschreibe UModel-Projekt aus Programmcode** lauten, je nachdem welche Einstellung im Dialogfeld "Synchronisierungseinstellungen" gewählt wurde. Das Dialogfeld "Synchronisierungseinstellungen" wird standardmäßig jedes Mal geöffnet, wenn Sie diesen Befehl aufrufen. Nähere Informationen dazu finden Sie unter Codesynchronisierungseinstellungen 244 .

Projekteinstellungen

Wenn Sie anhand von Programmcode ein UModel-Projekt generieren, können Sie die Projekteinstellungen 186 definieren bzw. ändern.

Synchronisierungseinstellungen

Öffnet das Dialogfeld "Synchronisierungseinstellungen", siehe Codesynchronisierungseinstellungen 244.

Modelltransformation

Ruft einen Assistenten auf, über den Sie das Modell von einer Programmiersprache in eine andere konvertieren können (z.B. von Java in C#), siehe Transformieren von UML-Modellen 317.

Projekt zusammenführen

Führt zwei UModel-Projektdateien in einem Modell zusammen. Die erste Datei, die Sie öffnen, ist die Datei, in die die zweite Datei überführt wird. Nähere Informationen dazu finden Sie unter Zusammenführen von UModel-Projekten 307.

Projekt zusammenführen (3-Weg)

UModel unterstützt die Zusammenführung mehrerer UModel-Projekte, die gleichzeitig von mehreren Entwicklern bearbeitet wurden, in einer 3-Weg-Projektzusammenführung 308.

© 2016-2022 Altova GmbH

Unterprojekt inkludieren

Siehe Inkludieren anderer UMdoel-Projekte 174.

Unterprojekt separat öffnen

Öffnet das ausgewählte Unterprojekt als neues Projekt.

Meldungen löschen

Löscht die Meldungen, Warnungen und Fehler, die im Fenster "Meldungen" 99 zur Syntaxüberprüfung und Codezusammenführung angezeigt werden.

Anmerkung: Fehlermeldungen weisen im Allgemeinen auf Probleme hin, die vor der Codegenerierung bzw. vor der Aktualisierung des Modellcodes während der Codegenerierung behoben werden müssen. Warnmeldungen weisen im Allgemeinen auf Probleme hin, die auch später behoben werden können. Fehlermeldungen und Warnmeldungen werden von der Syntaxüberprüfung, dem Compiler für die spezifische Sprache, dem UModel Parser, der die neu generierte Quelldatei liest, sowie beim Import von XMI-Dateien, generiert.

Dokumentation generieren

Dient zum Generieren von Dokumentation in den Formaten HTML, Microsoft Word und RTF für das derzeit offene Projekt. Nähere Informationen dazu finden Sie unter Generieren von UML-Dokumentation 345.

In keinem Diagramm verwendete Elemente auflisten

Erstellt eine Liste aller Elemente, die in keinem Diagramm des Projekts verwendet werden, siehe Überprüfen, wo und ob Elemente verwendet werden 120.

Freigegebene Pakete auflisten

Listet alle freigegebenen Pakete des aktuellen Projekts auf.

Inkludierte Pakete auflisten

Listet alle im aktuellen Projekt inkludierten Pakete auf.

16.4 Layout

Die Befehle des Menüs "Layout" gestatten Ihnen, die Elemente Ihrer Modelldiagramme anzuordnen und aneinander auszurichten, siehe Ausrichten von Modellierungselementen und Anpassen der Größe 136.

Ausrichten

Dieser Befehl dient je nach Auswahl des jeweiligen Befehls zum Ausrichten von Modellelementen entlang ihrer Ränder oder Mittelpunkte.

Gleichmäßig anordnen

Mit dieser Gruppe von Befehlen können Sie ausgewählte Elemente sowohl in horizontaler als auch vertikaler Richtung gleichmäßig anordnen.

Größe angleichen

Mit Hilfe dieser Befehle können Sie die Breite und Höhe der ausgewählten Elemente anhand des aktiven Elements anpassen.

Anordnen

Mit Hilfe dieser Gruppe von Befehlen können Sie die ausgewählten Elemente vertikal oder horizontal in einer Linie anordnen.

Linienart

Mit Hilfe dieser Gruppe von Befehlen können Sie die Art der Linie auswählen, mit der die verschiedenen Modellelemente verbunden werden sollen. Die Linien können jede Art von Abhängigkeit oder Assoziation darstellen, die in den verschiedenen Diagrammen verwendet werden.

Größe automatisch anpassen

Mit diesem Befehl werden die ausgewählten Elemente an die jeweils optimale Größe angepasst.

Automatisches Layout

Dieser Befehl dient zum Auswählen der Art, wie Modellelemente auf dem UML-Diagrammregister dargestellt werden sollen.

Zentriert	Zeigt die Modellierungselemente in der Mitte zentriert an.
Hierarchisch	Zeigt die Elemente nach ihren hierarchischen Beziehungen an. So wird z.B. eine übergeordnete Klasse oberhalb der davon abgeleiteten Klassen angezeigt. Die Optionen für hierarchisches Layout können über das Menü **Extras | Optionen,** Register **Ansicht,** Gruppe **Hierarchisch anordnen** angepasst werden.
Block	Zeigt die Elemente rechteckig nach Elementgröße gruppiert an.

© 2016-2022 Altova GmbH

Textlabels neu positionieren

Positioniert die Namen der (ausgewählten) Modellelemente zurück an ihre Standardposition.

16.5 Ansicht

Mit Hilfe der Befehle in diesem Menü können Sie:

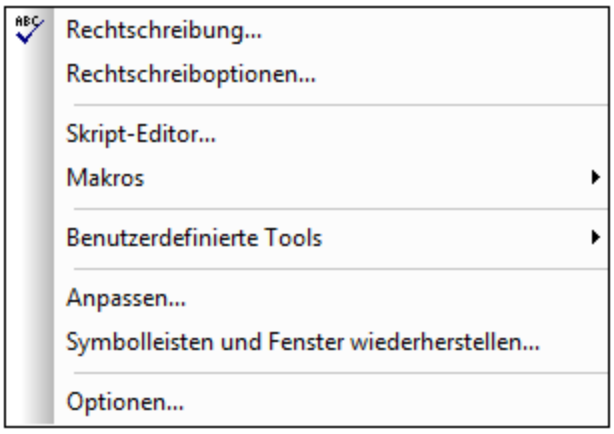
- jedes beliebige der UModel-Hilfsfenster ein- oder ausblenden, siehe Grafische Benutzeroberfläche von UModel 84
- die Sortierkriterien für die Modellelemente der Fenster "Modell-Struktur" 86 und "Favoriten" 91 definieren
- die Gruppierungskriterien der Diagramme im Fenster "Diagramm-Struktur" 90 definieren
- bestimmte UML-Elemente in den Fenstern "Favoriten" und "Modell-Struktur" ein- oder ausblenden
- den Zoomfaktor des aktuellen Diagramms definieren, siehe Vergrößern und Verkleinern von Diagrammen 142.

© 2016-2022 Altova GmbH

16.6 Extras

Mit den Befehlen im Menü "Extras" können Sie:

- die Rechtschreibprüfung für Ihr UModel-Projekt starten und die Rechtschreibprüfungsoptionen definieren.
- die Skripting-Umgebung[809] von UModel aufrufen. Sie können darin Ihre eigenen Formulare, Makros und Event Handler erstellen, verwalten und speichern.
- die aktuelle definierten Makros anzeigen und ausführen.
-
- Ihre Benutzeroberfläche anpassen[771]: Ihre eigenen Symbolleisten, Tastaturkürzel, Menüs und Makros definieren
- die Symbolleisten und Fenster wieder in den Originalzustand versetzen.
- die globalen Programmeinstellungen/optionen[781] definieren.

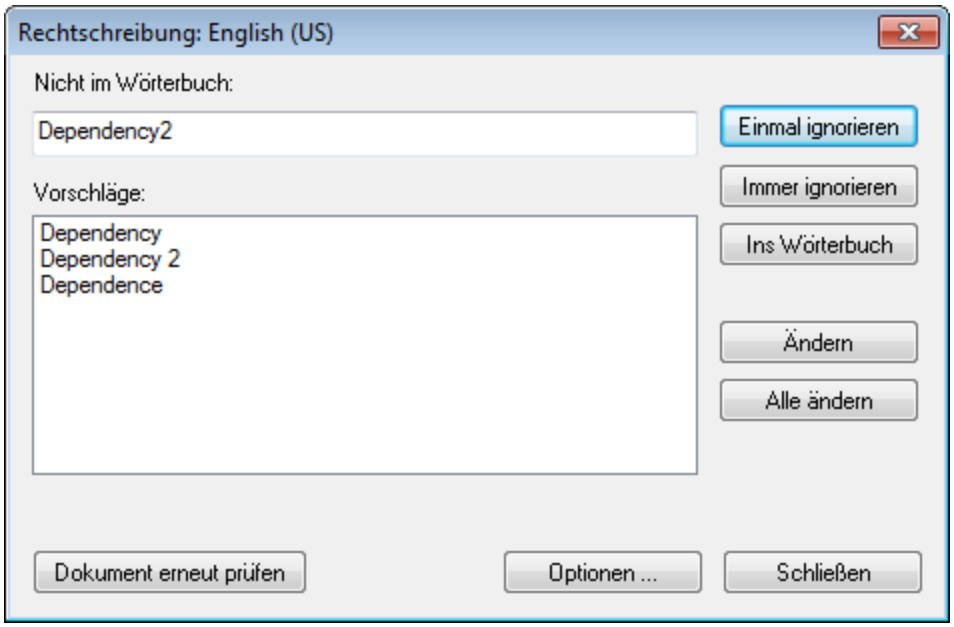


16.6.1 Rechtschreibung

Wählen Sie **Extras | Rechtschreibung** um die Rechtschreibprüfung zu starten. Die Standardoptionen für die Rechtschreibprüfung stehen in diesem Dialogfeld zur Verfügung.

Um die einzelnen Rechschreibprüfoptionen zu definieren, klicken Sie im Dialogfeld **Rechtschreibung** auf die Schaltfläche "**Optionen**" oder wählen Sie die Menüoption **Extras | Rechtschreiboptionen**.

Sie können sowohl die Rechtschreibung sowohl für Einträge in dier Modellstruktur als auch in UML-Diagrammen überprüfen. Wenn Sie mit der rechten Maustaste in die Modell-Struktur klicken und den Befehl "Rechtschreibung der Dokumentation" auswählen, wird die Rechtschreibung der Kommentare und Notizen in der Modell-Struktur überprüft.



Nicht im Wörterbuch

Dieses Textfeld enthält das Wort, das weder im ausgewählten Wörterbuch noch im Benutzerwörterbuch gefunden wurde.

Vorschläge

Dieses Listenfeld zeigt Wörter an, die Ähnlichkeiten mit dem unbekannten Wort haben (die Wörter stammen aus dem Wörterbuch und dem Benutzerwörterbuch). Durch Doppelklicken auf einen Eintrag wird dieser anstelle des unbekannten Wortes eingefügt und die Rechtschreibprüfung wird fortgesetzt.

Einmal ignorieren

Mit diesem Befehl können Sie das unbekannte Wort bei seinem ersten Auftreten ignorieren und die Rechtschreibprüfung fortsetzen. Sollte das Wort ein weiteres Mal im Dokument vorkommen, wird die Rechtschreibprüfung es dann wieder als unbekannt markieren.

Immer ignorieren

Mit diesem Befehl werden alle Instanzen des unbekannten Worts im gesamten Dokument ignoriert.

Ins Wörterbuch

Mit diesem Befehl können Sie das unbekannte Wort in das **Benutzerwörterbuch** einfügen. Das Benutzerwörterbuch kann über das Dialogfeld **Optionen** aufgerufen und dort bearbeitet werden..

Ändern

Mit diesem Befehl ersetzen Sie das im XML-Dokument markierte Wort durch das bearbeitete Wort aus dem Textfeld *Nicht im Wörterbuch*.

© 2016-2022 Altova GmbH

Alle ändern

Mit diesem Befehl ersetzen Sie das Wort an allen Stellen im XML-Dokument, an denen es vorkommt, durch das bearbeitete Wort aus dem Textfeld *Nicht im Wörterbuch*.

Dokument erneut prüfen

Das Dokument wird vom Anfang an neu überprüft.

Hinzufügen von Wörterbüchern für die Rechtschreibprüfung

Für jede Wörterbuchsprache stehen zwei Hunspell-Wörterbuchdateien zur Verfügung, die miteinander verwendet werden: eine `.aff` Datei und eine `.dic` Datei. Alle Wörterbücher werden unter dem folgenden Pfad im Ordner `Lexicons` installiert: `C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons`.

Im Ordner `Lexicons` werden unterschiedliche Sprachwörterbücher in jeweils anderen Ordnern gespeichert: `<Sprachenname>\<Wörterbuchdateien>`. So werden z.B. Dateien für die beiden englischsprachigen Wörterbücher (`English (British)` und `English (US)`) folgendermaßen gespeichert:

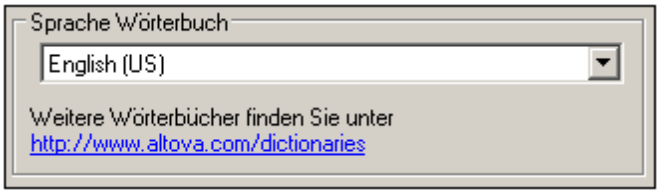
```
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)
\en_GB.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (British)
\en_GB.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)\en_US.dic
```

Die Sprachwörterbücher werden im Dialogfeld "Rechtschreiboptionen" in der Dropdown-Liste der Auswahlliste *Sprache Wörterbuch* aufgelistet. Diese Wörterbücher sind diejenigen, die im Ordner `Lexicons` zur Verfügung stehen, und haben dieselben Namen wie die Unterordner für die jeweilige Sprache im Ordner `Lexicons`. So würden z.B. im Fall der oben angegebenen englischsprachigen Wörterbücher die Wörterbücher in der Auswahlliste "Sprache Wörterbuch" als *English (British)* und *English (US)* angezeigt werden.

Alle installierten Wörterbücher stehen für alle Benutzer auf dem Rechner und alle Altova-Produktversionen (ob 32-Bit oder 64-Bit) zur Verfügung

Sie können Wörterbücher für die Rechtschreibprüfung auf zwei Arten hinzufügen, wobei die Dateien in keinem Fall im System registriert werden müssen:

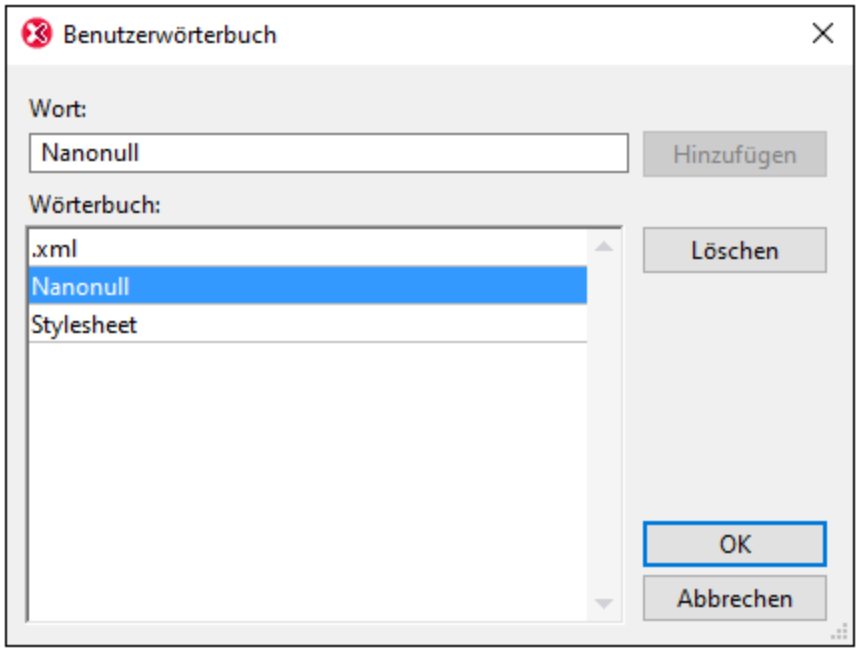
- Durch Hinzufügen von Hunspell-Wörterbüchern in einem neuen Unterordner des Ordners `Lexicons`. Hunspell-Wörterbücher können z.B. von https://wiki.services.openoffice.org/wiki/Dictionaries oder http://extensions.services.openoffice.org/en/dictionaries heruntergeladen werden. (Beachten Sie bitte, dass in OpenOffice das Zip-Format `OXT` verwendet wird. Ändern Sie also die Erweiterung in `.zip` und entpacken Sie die `.aff`- und `.dic`-Datei in die Sprachordner im Ordner `Lexicons`. Beachten Sie außerdem, dass Hunspell-Wörterbücher auf Myspell-Wörterbüchern basieren. Daher können auch Myspell-Wörterbücher verwendet werden.)
- Durch Verwendung des Altova-Wörterbuchinstallationsprogramms, das ein Paket mit mehreren Sprachwörterbüchern standardmäßig im richtigen Ordner auf Ihrem Rechner installiert. Das Installationsprogramm kann durch Klicken auf den Link im Bereich "Sprache Wörterbuch" des Dialogfelds "Rechtschreibung Optionen" (*siehe Abbildung unten*) heruntergeladen werden. Für die Installation benötigen Sie Administratorrechte, da die Installation sonst nicht durchgeführt werden kann.



Anmerkung: Es bleibt Ihnen überlassen, ob Sie mit den für das jeweilige Wörterbuch geltenden Lizenzbedingungen einverstanden sind und ob das Wörterbuch sich für die Verwendung mit der auf Ihrem Rechner installierten Software eignet.

Arbeiten mit dem Benutzerwörterbuch

Jeder Benutzer hat ein Benutzerwörterbuch, in dem vom Benutzer genehmigte Wörter gespeichert werden können. Bei der Rechtschreibprüfung wird die Rechtschreibung anhand einer Wörterliste bestehend aus den Wörtern im Sprachwörterbuch und denen im Benutzerwörterbuch durchgeführt. Sie können Wörter über das Dialogfeld "Benutzerwörterbuch" (*Abbildung unten*) zum Benutzerwörterbuch hinzufügen bzw. diese daraus löschen. Dieses Dialogfeld wird durch Klick auf die Schaltfläche "Benutzerwörterbuch" im Dialogfeld "Rechtschreiboptionen" aufgerufen (*siehe zweite Abbildung in diesem Abschnitt*).



Um ein Wort zum Benutzerwörterbuch hinzuzufügen, geben Sie das Wort in das Textfeld "Wort" ein und klicken Sie auf **Hinzufügen**. Daraufhin wird das Wort zur alphabetischen Liste im Fenster "Wörterbuch" hinzugefügt. Um ein Wort aus dem Wörterbuch zu löschen, wählen Sie das Wort im Fenster "Wörterbuch" aus und klicken Sie auf **Löschen**. Daraufhin wird das Wort aus dem Fenster "Wörterbuch" gelöscht. Wenn Sie mit der Bearbeitung des Benutzerwörterbuchs fertig sind, klicken Sie auf **OK**, damit die Änderungen im Benutzerwörterbuch gespeichert werden.

Wörter können auch während einer Rechtschreibprüfung zum Benutzerwörterbuch hinzugefügt werden. Wenn bei der Rechtschreibprüfung ein unbekanntes Wort gefunden wird, wird das Dialogfeld Rechtschreibprüfung 765

© 2016-2022 Altova GmbH

aufgerufen und Sie werden aufgefordert zwischen verschiedenen Aktionen zu wählen. Wenn Sie auf die Schaltfläche **Ins Wörterbuch** klicken, wird das unbekannte Wort zum Benutzerwörterbuch hinzugefügt.

Das Benutzerwörterbuch befindet sich im folgenden Ordner: `C:\Benutzer\<Benutzer>\Dokumente\Altova\SpellChecker\Lexicons\user.dic`

16.6.2 Rechtschreiboptionen

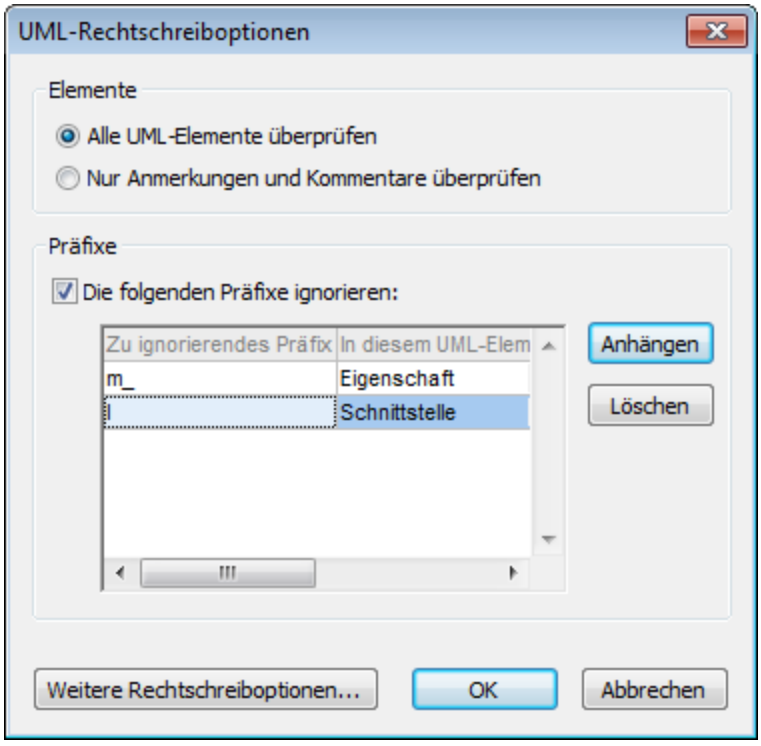
Elemente

In dieser Gruppe können Sie auswählen, ob alle UML-Elemente oder nur Anmerkungen und Kommentarobjekte überprüft werden sollen.

Präfixe

Doppelklicken Sie in die Spalte "Zu ignorierendes Präfix", um Präfixe oder bestimmte UML-Elemente, die bei der Rechtschreibprüfung ignoriert werden sollen, zu ignorieren (z.B. m_ für Eigenschaften und I für Interfaces).

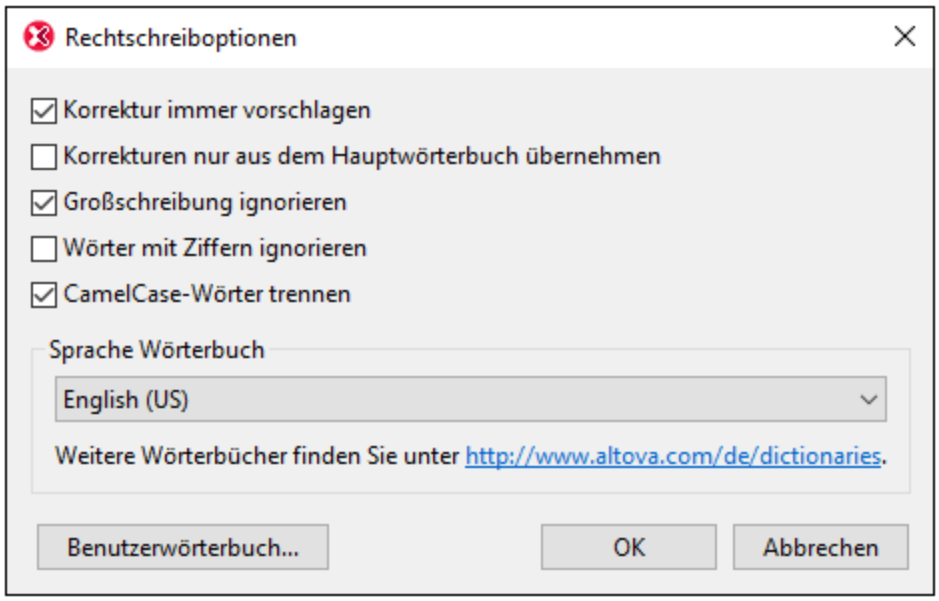
Über die Schaltfläche "Anhängen" können Sie eine neue Zeile zur Tabelle "Präfixe" hinzufügen. Mit "Löschen" wird die aktive Zeile gelöscht.



Wenn Sie auf die Schaltfläche "Weitere Rechtschreiboptionen..." klicken, wird das unten gezeigte Dialogfeld "Optionen" geöffnet.

Weitere Rechtschreiboptionen

Im Dialogfeld "Rechtschreiboptionen" können Sie globale Optionen für die Rechtschreibprüfung definieren.



Korrektur immer vorschlagen:
Wenn Sie dieses Kontrollkästchen aktivieren, werden die Vorschläge (sowohl aus dem Wörterbuch als auch aus dem Benutzerwörterbuch) in der Vorschlagsliste angezeigt. Wenn Sie diese Option deaktivieren, werden keinen Vorschläge angezeigt.

Korrekturen nur aus dem Hauptwörterbuch übernehmen:
Bei Aktivierung dieser Option werden nur Vorschläge angezeigt, die im Hauptwörterbuch zur Verfügung stehen. Das Benutzerwörterbuch wird nicht durchsucht. Die Schaltfläche **Benutzerwörterbuch** ist deaktiviert, sodass das Benutzerwörterbuch nicht bearbeitet werden kann.

Großschreibung ignorieren:
Großgeschriebene Wörter werden in der Rechtschreibprüfung ignoriert.

Wörter mit Ziffern ignorieren:
Wörter, die Ziffern enthalten, werden in der Rechtschreibprüfung ignoriert.

CamelCase-Wörter trennen
CamelCase-Wörter sind Wörter, die einen Großbuchstaben innerhalb des Worts haben. So ist z.B. im Wort "CamelCase" das "C" von "Case" groß geschrieben, d.h. es handelt sich hierbei um ein CamelCase-Wort. Da man derartige Wörte selten in einem Wörterbuch findet, würden Sie von der Rechtschreibprüfung als Fehler markiert. Um dies zu vermeiden, werden CamelCase-Wörter mit dieser Option in ihre beiden Komponenten aufgeteilt, sodass die Komponenten einzeln überprüft werden können. Diese Option ist standardmäßig aktiviert.

Sprache Wörterbuch
Wählen Sie im Dropdown-Listenfeld eine Sprache, in der die Rechtschreibprüfung erfolgen soll. Die Standardeinstellung ist US-Englisch. Wörterbücher in weiteren Sprachen können kostenlos von der Altova Website heruntergeladen werden.

© 2016-2022 Altova GmbH

16.6.3 Skript-Editor

Mit dem Befehl "Skript-Editor" wird das Skript-Editor-Fenster geöffnet, siehe Skript-Editor[809].

Anmerkung: Um den Skript-Editor ausführen zu können, muss die .NET Framework Version 2.0 oder höher auf Ihrem Rechner installiert sein.

16.6.4 Makros

Ruft eine Liste der derzeit im Skripting-Projekt definierten Makros auf, siehe Skript-Editor[809]. Das aktive Skripting-Projekt ist im Dialogfeld "Optionen" auf dem Register Skripting-Umgebung[787] definiert.

16.6.5 Benutzerdefinierte Tools

Wenn Sie den Mauszeiger über den Befehl **Benutzerdefinierte Tools** platzieren, wird ein Untermenü mit benutzerdefinierten Befehlen angezeigt, die externe Applikationen verwenden. Sie können diese Befehle im Dialogfeld "Anpassen" auf dem Register "Extras"[774] erstellen. Wenn Sie auf einen dieser benutzerdefinierten Befehle klicken, wird die mit diesem Befehl verknüpfte Aktion ausgeführt.

Der Befehl **Benutzerdefinierte Tools | Anpassen** öffnet das Register "Extras"[774] im Dialogfeld "Anpassen" (Hier können Sie die benutzerdefinierten Befehle, die im Menü des Befehls **Benutzerdefinierte Tools** angezeigt werden, erstellen).

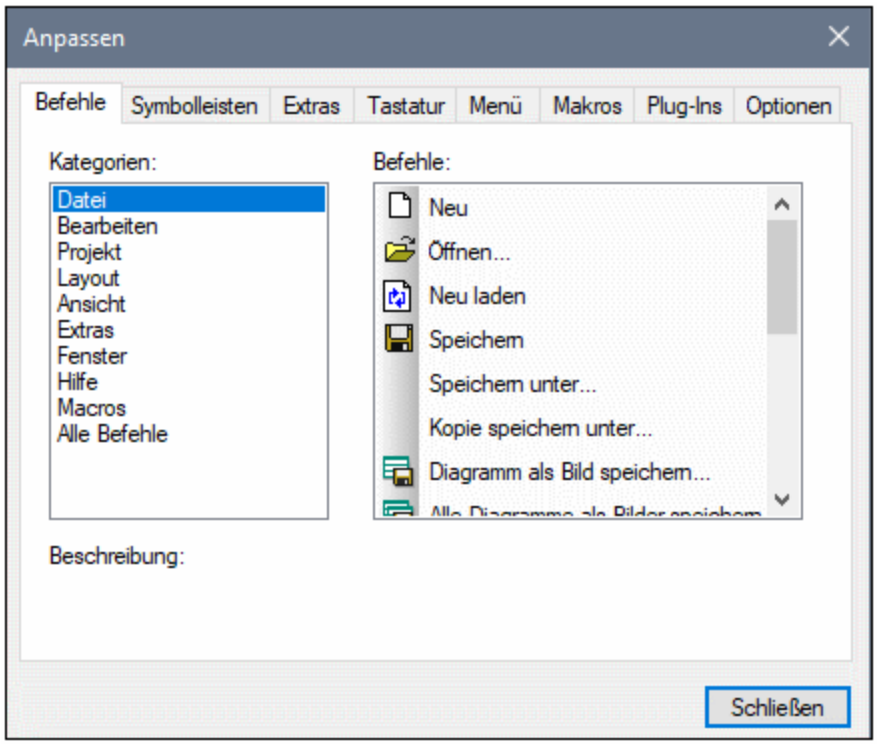
16.6.6 Anpassen

Mit dem Befehl **Anpassen** rufen Sie ein Dialogfeld auf, in dem Sie UModel an Ihre persönlichen Bedürfnisse anpassen können. Sie können folgende Dinge anpassen:

- Befehle[771]
- Symbolleisten[773]
- Extras[774]
- Tastatur[777]
- Menü[779]
- Makros[780]
- Plug-in[780]
- Optionen[781]

16.6.6.1 Befehle

Auf dem Register "**Befehle**" können Sie Ihre UModel-Menüs oder Symbolleisten anpassen.



So fügen Sie einen Befehl zu einer Symbolleiste oder einem Menü hinzu:

1. Klicken Sie im Menü **Extras** auf **Anpassen.**
2. Wählen Sie im Listenfeld "**Kategorien**" die Befehlskategorie aus. Die verfügbaren Befehle werden im Listenfeld "**Befehle**" angezeigt.
3. Klicken Sie auf einen Befehl in der Liste der **Befehle** und ziehen Sie ihn in ein vorhandenes Menü bzw. eine vorhandene Symbolleiste. Wenn Sie den Cursor über eine Position halten, an die der Befehl gezogen werden kann, wird ein I-Zeichen angezeigt.
4. Lassen Sie die Maustaste an der Position los, an der der Befehl eingefügt werden soll. An der Spitze des Mauszeigers erscheint beim Ziehen des Befehls eine kleine Schaltfläche. Das Häkchen unterhalb des Mauszeigers bedeutet, dass der Befehl nicht an der aktuellen Cursorposition abgelegt werden kann. An den Stellen, an die der Befehl gezogen werden kann (also über der Symbolleiste oder einem Menü) verschwindet das Häkchen.

Anmerkungen:

- Wenn Sie den Cursor beim Ziehen über ein Menü ziehen, wird dieses geöffnet, sodass Sie den Befehl an einer beliebigen Stelle im Menü ablegen können.
- Befehle können in Menüs oder Symbolleisten platziert werden. Wenn Sie Ihre eigene Symbolleiste erstellt haben, können Sie darin Ihre eigenen Befehle/Symbole einordnen.
- Auf dieselbe Weise können Sie auch die Befehle in den Kontextmenüs[779] (wird durch Rechtsklick aufgerufen) bearbeiten. Klicken Sie auf das Register "Menü" und wählen Sie anschließend aus der Liste der Kontextmenüs das jeweilige Kontextmenü aus.

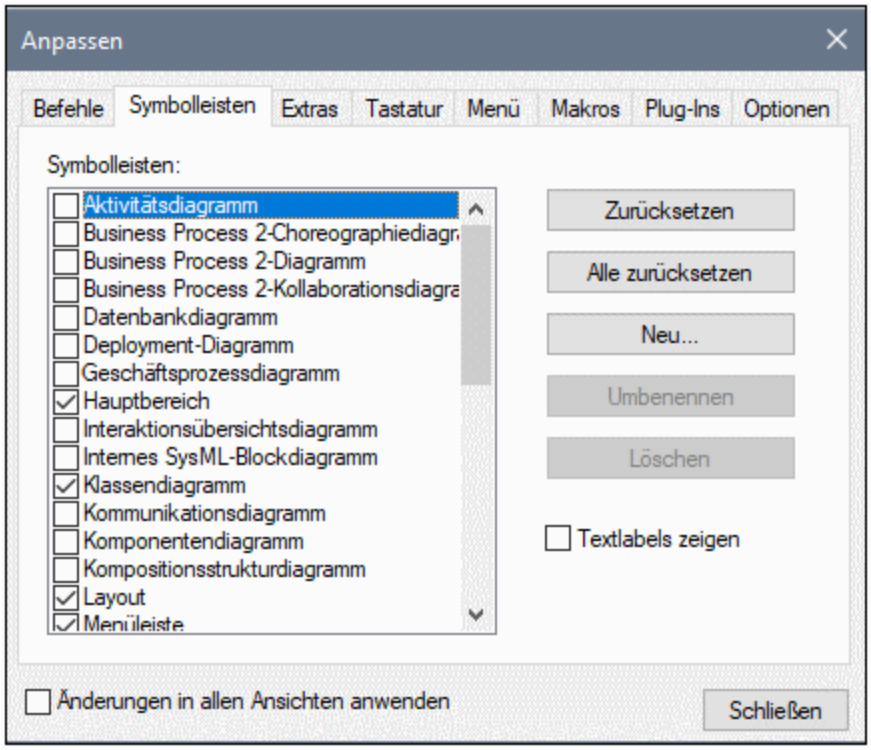
© 2016-2022 Altova GmbH

So löschen Sie einen Befehl oder ein Menü:

1. Klicken Sie im Menü **Extras** auf **Anpassen.**
2. Klicken Sie auf den Menüeintrag bzw. das zu löschende Symbol und ziehen Sie es mit der Maus.
3. Lassen Sie die Maustaste los, sobald das Häkchensymbol unterhalb des Mauszeigers erscheint. Der Befehl bzw. der Menüeintrag wird aus dem Menü bzw. der Symbolleiste gelöscht.

16.6.6.2 Symbolleisten

Auf dem Register "**Symbolleisten**" können Sie bestimmte Symbolleisten aktivieren oder deaktivieren sowie Ihre eigenen Symbolleisten erstellen.



Symbolleisten enthalten Befehle für die am häufigsten verwendeten Menübefehle. Zu jedem Symbol erhalten Sie eine kurze Erklärung in Form eines Tooltipps, wenn Sie den Mauszeiger direkt über das Element platzieren. In der Statusleiste wird eine detailliertere Beschreibung des Befehls angezeigt. Sie können die Symbolleisten von ihrer Standardposition an eine beliebige Stelle auf dem Bildschirm ziehen, wo sie als abgedocktes Fenster angezeigt werden. Alternativ dazu können Sie die Symbolleisten auch am linken oder rechten Rand des Hauptfensters andocken.

So aktivieren oder deaktivieren Sie eine Symbolleiste:

- Klicken Sie auf das Kontrollkästchen, um die jeweilige Symbolleiste zu aktivieren (bzw. zu deaktivieren).

So erstellen Sie eine neue Symbolleiste:

1. Klicken Sie auf die Schaltfläche **Neu...** und geben Sie der Symbolleiste im Dialogfeld "Symbolleistenname" einen Namen.
2. Fügen Sie über das Register Befehle[771] des Dialogfelds "Anpassen" Befehle zur Symbolleiste hinzu.

So setzen Sie die Menüleiste zurück

1. Klicken Sie auf den Menüleisteneintrag und
2. anschließend auf die Schaltfläche **Zurücksetzen**, um die Menübefehle auf den Zustand zum Zeitpunkt der Installation zurückzusetzen

So setzen Sie alle Symbolleisten- und Menübefehle zurück

1. Klicken Sie auf die Schaltfläche **Alle zurücksetzen**, um alle Symbolleistenbefehle auf den Zustand zum Zeitpunkt der Installation des Programms zurückzusetzen. Sie werden gefragt, ob alle Symbolleisten und Menüs zurückgesetzt werden sollen.
2. Klicken Sie auf "**Ja**", um den Vorgang zu bestätigen.

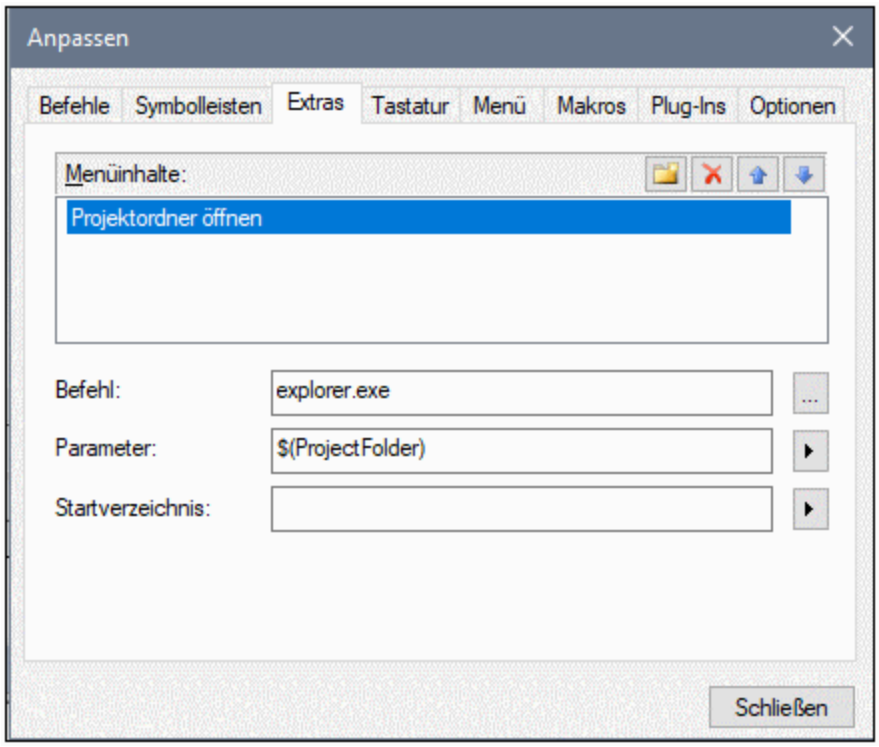
Bei Aktivierung der Option **Textlabels zeigen** wird erklärender Text unterhalb der Symbolleistenbefehle angezeigt.

16.6.6.3 Extras

Über das Register **Extras** können Sie benutzerdefinierte Menübefehle erstellen, mit denen Sie externe Tools direkt von UModel aus starten können. Die hier definierten Menübefehle werden im Menü **Extras | Benutzerdefinierte Tools** angezeigt. Bei externen Tools kann es sich um in Windows enthaltene Programme, wie Windows Explorer (**explorer.exe**), Notepad (**notepad.exe**) oder andere benutzerdefinierte ausführbare Dateien handeln. Sie können jedem benutzerdefinierten Tool optional Parameter zuweisen und das Verzeichnis, in dem das externe Tool initialisiert werden soll, definieren (um nach relativen Dateinamen suchen zu können).

So wird etwa mit der unten gezeigten Konfiguration ein neuer Menübefehl namens "Projektordner öffnen" hinzugefügt. Wenn Sie den Befehl starten, wird damit das Verzeichnis des aktuellen UModel-Projekts in Windows Explorer geöffnet.

© 2016-2022 Altova GmbH



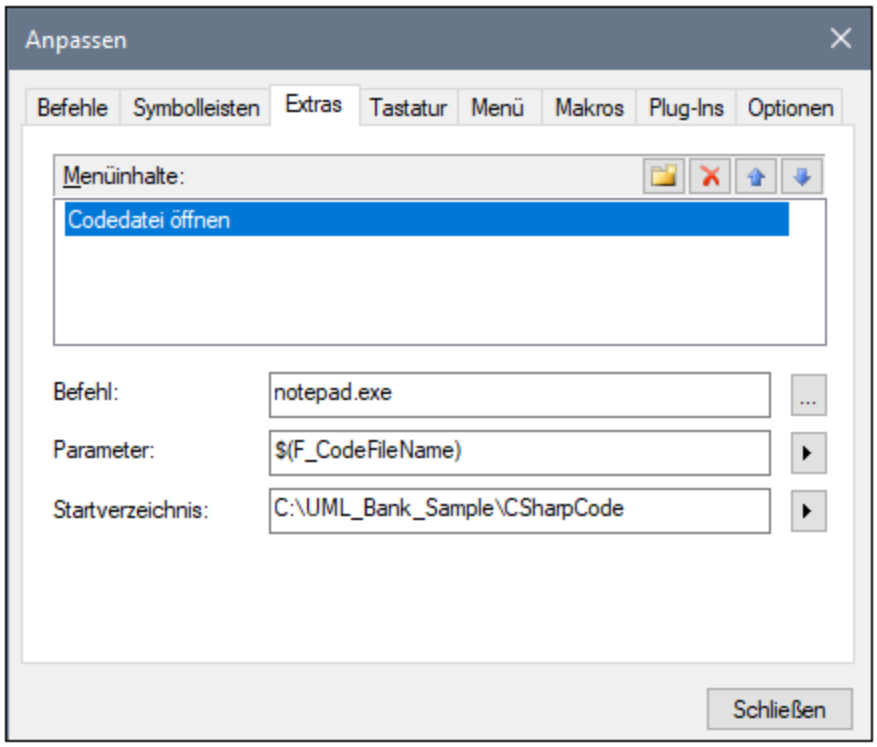
Wenn ein externes Tool Parameter (wie z.B. im Beispiel oben Windows Explorer erhält), können diese in das Eingabefeld "Parameter" eingegeben werden. Bei Eingabe mehrerer Parameter, müssen diese durch ein Leerzeichen getrennt werden. Bei den Werten, die Sie als Parameter bereitstellen können, kann es sich um reinen Text (hartcodierte Werte) handeln oder sie können mit der Schaltfläche ▸ aus einer Liste vordefinierte UModel-Variablen ausgewählt werden. Sie können jede der folgenden vordefinierten UModel-Variablen als Parameter verwenden:

Vordefinierte UModel-Variable	Aufgabe
Projektdateiname	Der Dateiname der aktiven UModel-Projektdatei, z.B. **Test.ump**.
Projektdateipfad	Der absolute Dateipfad der aktiven UModel-Projektdatei, z.B. **C:\MyDirectory\Test.ump**.
Fokussierte UML-Daten - Name	Der Name des UML-Elements, das sich im Fokus befindet, z.B. **Klasse1**.
Fokussierte UML-Daten – Qualifizierter UML-Name	Der qualifizierte Name des UML-Elements, das sich im Fokus befindet, z.B. **Paket1::Paket2::Klasse1**.
Fokussierte UML-Daten – Codedateiname	Der Codedateiname der UML-Klasse, -Schnittstelle oder Enumeration, die sich gerade im Fokus befindet, der im Fenster "Eigenschaften" (relativ zur realisierenden

Vordefinierte UModel-Variable	Aufgabe
	Komponente) angezeigt wird, z.B. **Klasse1.cs** oder **MeinNamespace\Klasse1.Java**.
Fokussierte UML-Daten – Codedateipfad	Der Codedateipfad der UML-Klasse, -Schnittstelle oder Enumeration, die sich gerade im Fokus befindet, der im Fenster "Eigenschaften" angezeigt wird, z.B., **C:\Temp\MySource\Klasse1.cs**.
Fokussierte UML-Daten – Codeprojektdateiname	Der Dateiname des Codeprojekts, zu dem die UML-Klasse, Schnittstelle oder Enumeration, die sich gerade im Fokus befindet, gehört. Der Codeprojektdateiname kann relativ zur UModel-Projektdatei sein und ist mit dem in den Eigenschaften der Komponente angezeigten identisch, z.B. **C:\Temp\MySource\MeinProjekt.vcproj** oder **MySource\MeinProjekt.vcproj**.
Fokussierte UML-Daten – Codeprojektdateipfad	Der Dateipfad des Codeprojekts, zu dem die UML-Klasse, Schnittstelle oder Enumeration, die sich gerade im Fokus befindet, gehört, z.B. **C:\Temp\MySource\MeinProjekt.vcproj**.
Projektordner	Das Verzeichnis, in dem das aktuelle UModel-Projekt gespeichert ist, z.B. **C:\Benutzer\<Benutzer>\Dokumente\Altova\UModel2023\UModelExamples**.
Temporärer Ordner	Das Verzeichnis, in dem die temporären Dateien der Applikation gespeichert sind, z.B. **C:\Benutzer\<Benutzer>\AppData\Local\Temp**.

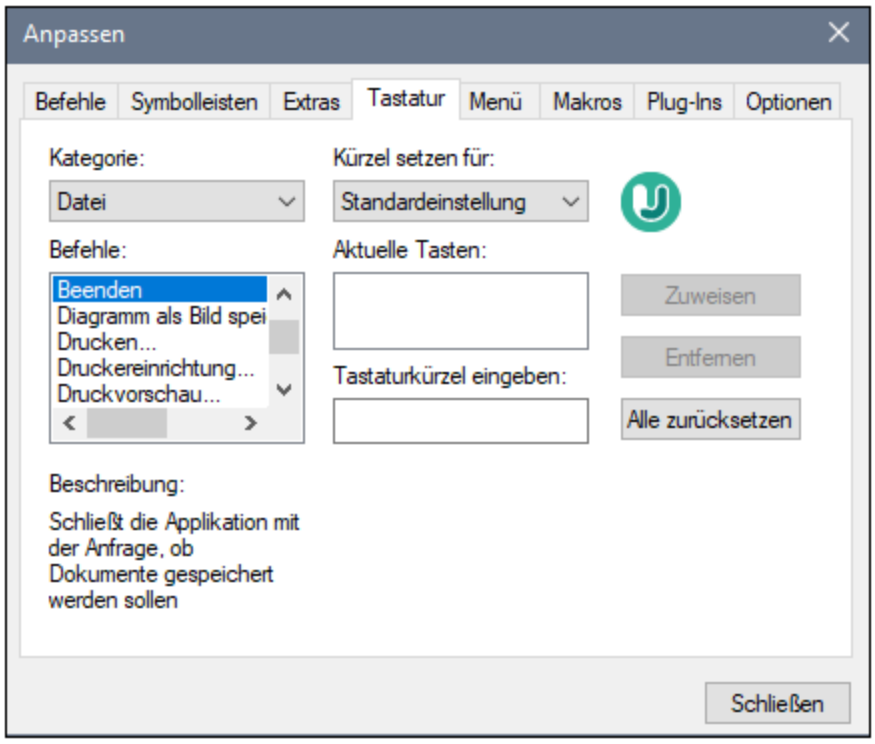
In einigen Fällen müssen Sie eventuell auch einen Wert in das Eingabefeld **Startverzeichnis** eingeben. So wird z.B. in der Konfiguration unten die Codedatei des aktuell in einem Diagramm ausgewählten Elements in Notepad geöffnet. (Beachten Sie: Damit dieser Befehl funktioniert, muss für das im Diagramm aktuell ausgewählte Element ein Wert (Dateiname) im Feld Codedateiname des Fensters "Eigenschaften" 92 definiert sein und diese Datei muss im Verzeichnis **C:\UML_Bank_Sample\CSharpCode** vorhanden sein).

© 2016-2022 Altova GmbH



16.6.6.4 Tastatur

Auf dem Register "**Tastatur**" können Sie Tastaturkürzel für jeden beliebigen Befehl definieren (oder ändern).



So weisen Sie einem Befehl ein neues Tastaturkürzel zu:

1. Wählen Sie in der Auswahlliste **Kategorie** die Befehlskategorie aus.
2. Wählen Sie in der Liste **Befehle** den **Befehl** aus, dem Sie ein neues Tastaturkürzel zuweisen möchten.
3. Klicken Sie in das Textfeld **Tastaturkürzel eingeben**: und drücken Sie die Tasten, mit denen dieser Befehl aufgerufen werden soll. Die Tastenkombination wird sofort im Textfeld angezeigt. Falls diese Tastenkombination bereits zugewiesen wurde, so wird diese Funktion unterhalb des Textfelds angezeigt.
4. Klicken Sie auf die Schaltfläche **Zuweisen**, um das Tastaturkürzel permanent zuzuweisen. Das Tastaturkürzel erscheint nun im Listenfeld **Aktuelle Tasten**. (Um den Eintrag in diesem Textfeld zu **löschen,** drücken Sie eine der Steuerungstasten: **Strg, Alt** oder **Umschalttaste**.)

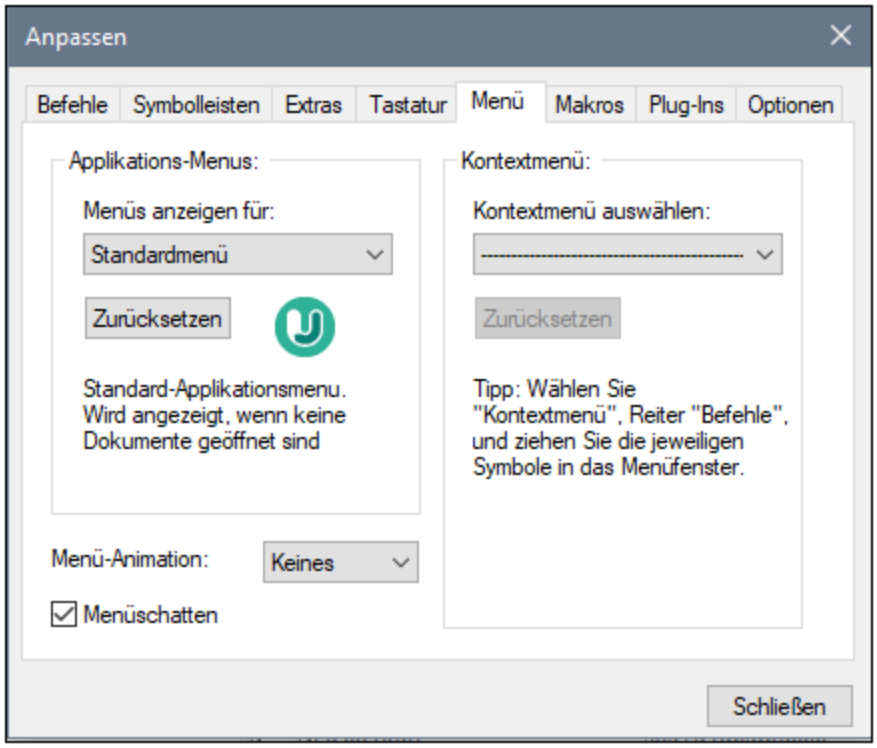
So heben Sie eine Tastenzuweisung auf (oder löschen ein Tastaturkürzel):

1. Klicken Sie in der Liste **Aktuelle Tasten** auf das zu löschende Tastaturkürzel.
2. Klicken Sie auf die jetzt aktiv gewordene Schaltfläche **Entfernen.**
3. Klicken Sie auf die Schaltfläche **Schließen**, um alle im Dialogfeld "Anpassen" vorgenommenen Änderungen zu bestätigen.

© 2016-2022 Altova GmbH

16.6.6.5 Menü

Auf dem Register **Menü** können Sie die Menüleisten sowie die Kontextmenüs anpassen.



Anpassen von Menüs

Die Menüleiste **Standardmenü** ist die Menüleiste, die angezeigt wird, wenn kein Projektfenster geöffnet ist. Die Menüleiste **UModel-Projekt** ist die Menüleiste, die angezeigt wird, wenn ein Projekt geöffnet ist. Jede Menüleiste kann separat angepasst werden. Änderungen, die an einer der Leisten vorgenommen wurden, haben keine Auswirkung auf andere Menüleisten.

Um eine Menüleiste anzupassen, wählen Sie diese in der Auswahlliste **Menüs anzeigen für** aus. Klicken Sie anschließend auf das Register **Befehle** und ziehen Sie die Befehle aus dem Listenfeld **Befehle** in die Menüleiste oder in eines der Menüs.

Löschen von Befehlen aus Menüs und Zurücksetzen der Menüleisten

So löschen Sie ein ganzes Menü oder einen Befehl in einem Menü:

1. Wählen Sie die gewünschte Menüleiste aus der Dropdown-Liste **Menüs anzeigen für** aus.
2. Während das Dialogfeld "Anpassen" geöffnet ist, (i) wählen Sie das Menü, das Sie aus der Menüleiste der Applikation löschen möchten, aus oder (ii) wählen Sie den Befehl aus, den Sie aus einem dieser Menüs löschen möchten.
3. Ziehen Sie entweder das Menü aus der Menüleiste oder den Menübefehl aus dem Menü oder (ii) klicken Sie mit der rechten Maustaste auf das Menü oder den Menübefehl und wählen Sie den Befehl **Löschen**.

Sie können jede Menüleiste in den Originalzustand zurücksetzen. Wählen Sie sie dazu aus der Dropdown-Liste **Menüs anzeigen für** aus und klicken Sie anschließend auf die Schaltfläche **Zurücksetzen**.

Anpassen der Kontextmenüs der Applikation

Die Kontextmenüs sind die Menüs, die angezeigt werden, wenn Sie mit der rechten Maustaste auf bestimmte Objekte auf der Benutzeroberfläche der Applikation klicken. Jedes dieser Kontextmenüs kann folgendermaßen angepasst werden:

1. Wählen Sie das Kontextmenü in der Dropdown-Liste **Kontextmenü auswählen** aus. Daraufhin wird das Kontextmenü angezeigt.
2. Klicken Sie auf das Register **Befehle**.
3. Ziehen Sie den gewünschten Befehl aus dem Listenfeld **Befehle** in das Kontextmenü.
4. Um einen Befehl aus dem Kontextmenü zu löschen, klicken Sie mit der rechten Maustaste auf diesen Befehl im Kontextmenü und wählen Sie den Befehl **Löschen**. Ziehen Sie den Befehl alternativ dazu mit der Maus aus dem Kontextmenü heraus.

Sie können jedes Kontextmenü in den Originalzustand zurücksetzen. Wählen Sie es dazu aus der Dropdown-Liste **Kontextmenü auswählen** aus und klicken Sie anschließend auf die Schaltfläche **Zurücksetzen**.

Menüschatten

Aktivieren Sie das Kontrollkästchen **Menüschatten**, wenn Menüs mit Schatten dargestellt werden sollen.

Wenn Sie animierte Menüs vorziehen, haben Sie die Wahl zwischen einer Reihe von Menüanimationen. Die Dropdown-Liste **Menu-Animation** enthält die folgenden Optionen:

- Keines (Standard)
- Entfalten
- Entrollen
- Abblenden

16.6.6.6 Makros

Auf dem Register "Makros" können Sie zwischen den Makros auswählen, die in dem aktuell aktiven Skriptingprojekt definiert sind.

Die aktiven Skriptingprojekte sind im Dialogfeld "Optionen" auf dem Register "Skripting" oder in den Projekteinstellungen auf dem Register "Skripting" 787 definiert.

16.6.6.7 Plug-ins

Auf dem Register **Plug-ins** können Sie ein UModel Plugin (.dl-Dateil), das mit UModel integriert werden kann, hinzufügen oder entfernen, siehe UModel IDE Plug-Ins 837.

© 2016-2022 Altova GmbH

16.6.6.8 Optionen

Auf dem Register **Optionen** können Sie allgemeine Einstellungen vornehmen.

Wenn das Kontrollkästchen **Tooltip in Symbolleiste einblenden** aktiv ist, wird ein Tooltipp angezeigt, wenn der Mauszeiger über eine Symbolleisten-Schaltfläche platziert wird. In diesem Tooltipp wird eine kurze Beschreibung der Schaltflächenfunktion sowie das dazugehörige Tastaturkürzel, falls eines zugewiesen wurde, angezeigt.

Wenn das Kontrollkästchen **Große Symbole** aktiv ist, werden anstatt der Standardsymbole größere Symbole angezeigt.

16.6.7 Symbolleisten und Fenster wiederherstellen

Mit diesem Befehl wird UModel geschlossen und mit den Standardeinstellungen neu gestartet. Vor dem Schließen wird ein Dialogfeld angezeigt, in dem Sie gebeten werden, den Befehl zu bestätigen

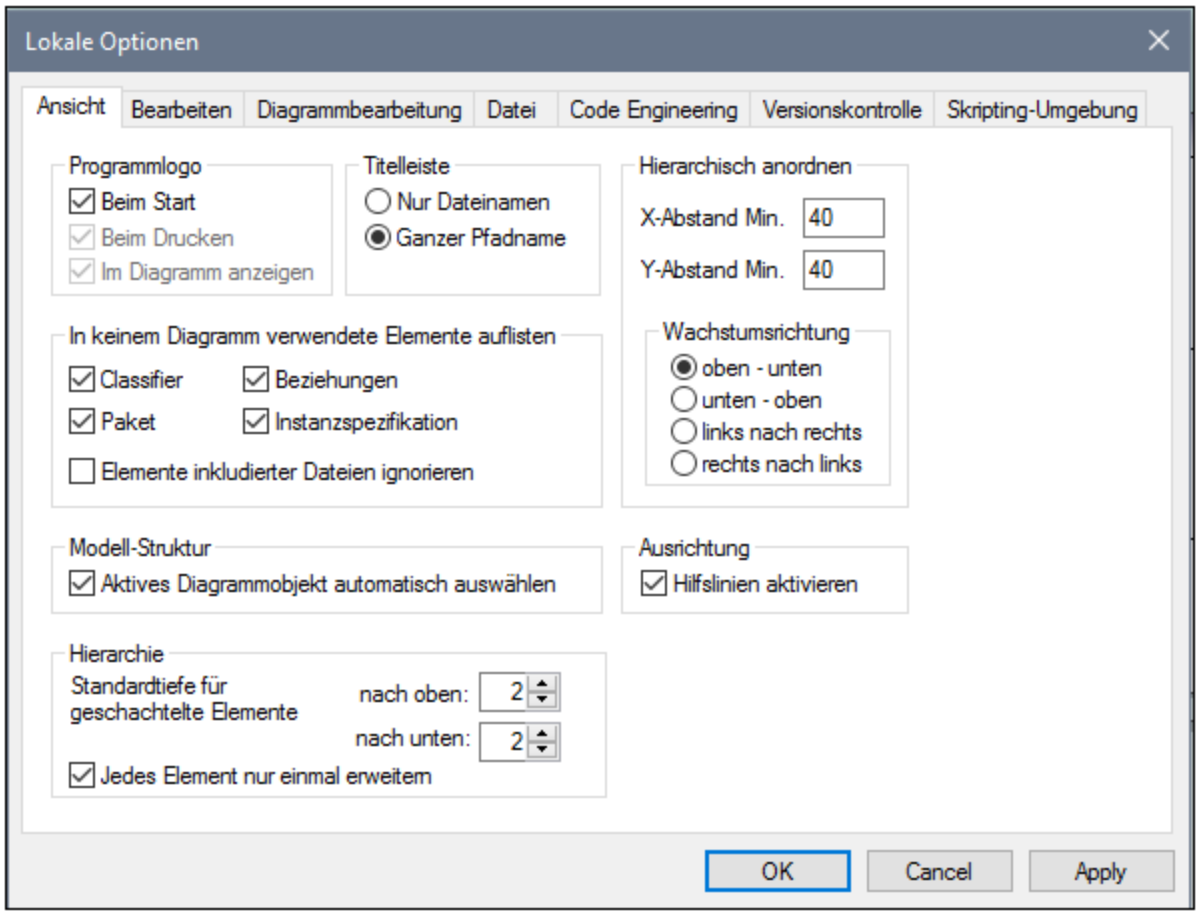
Dieser Befehl ist hilfreich, wenn Sie Symbolleisten oder Fenster verschoben oder ausgeblendet oder deren Größe angepasst haben und nun alle Originaleinstellungen wiederherstellen möchten.

16.6.8 Optionen

Wählen Sie den Menüeintrag **Extras | Optionen,** um Ihre Projektoptionen zu definieren.

Auf dem Register **Ansicht** können Sie folgende Einstellungen festlegen:

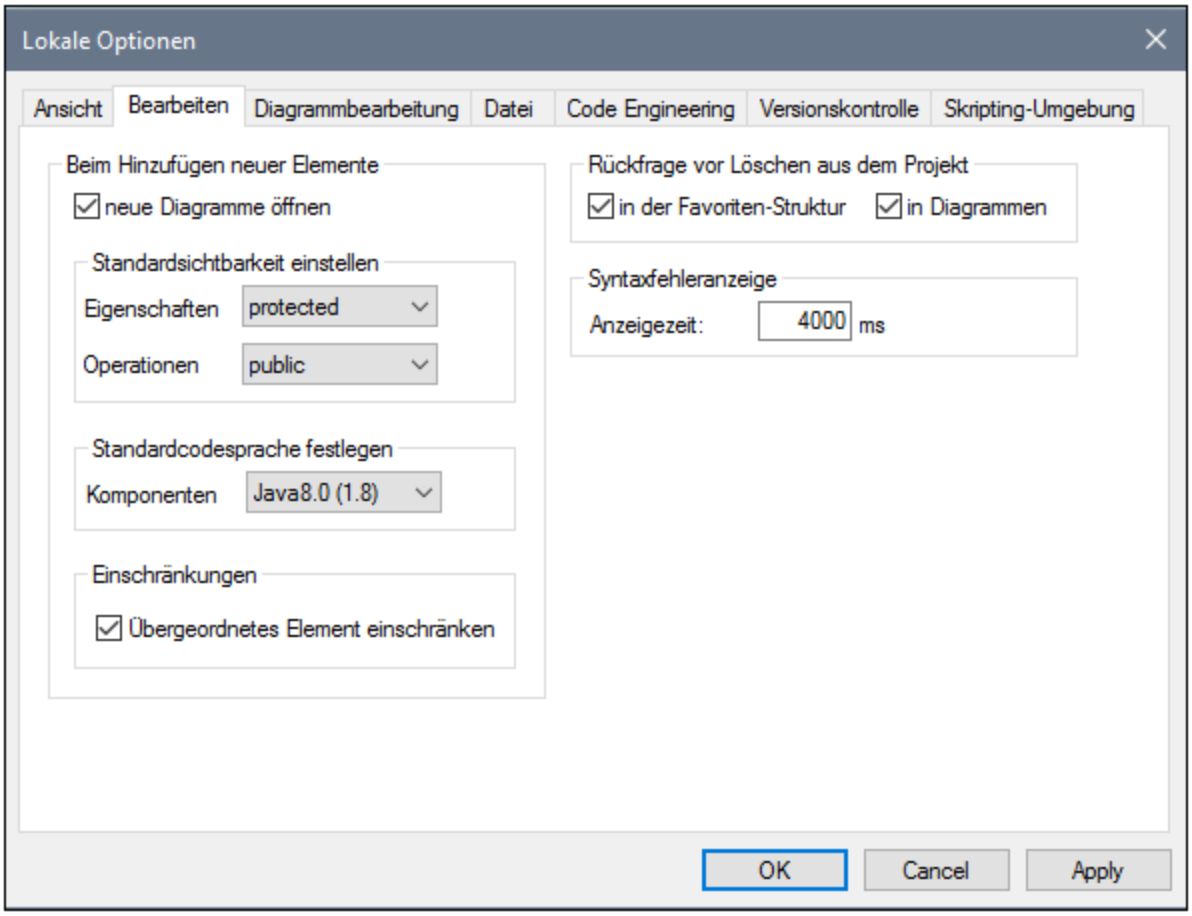
- wo das Programmlogo angezeigt werden soll.
- den Inhalt der Applikationstitelleiste
- die Elementarten, die aufgelistet werden sollen, wenn Sie auf dem Register "Model-Struktur" oder "Favoriten" auf die Kontextmenüoption "In keinem Diagramm verwendete Elemente auflisten" klicken. Sie haben auch die Möglichkeit, Elemente in **inkludierten** Dateien zu ignorieren.
- ob ein in einem Diagramm ausgewähltes Element in der Modell-Struktur automatisch ausgewählt/synchronisiert werden soll.
- die Standardtiefe der hierarchischen Ansicht auf dem Register "Hierarchie" bei Verwendung der Option **Graph. Ansicht anzeigen** .
- Die Einstellungen für automatisches hierarchisches Layout, über Sie für das Hierarchiefenster die Verschachtelungstiefe nach oben und unten festlegen können.
- "Jedes Element nur einmal erweitern". Dadurch kann im selben Bild/Diagramm kann nur eines derselben Classifier-Elemente erweitert werden.
- ob Sie Elemente beim Ziehen mit der Maus in einem Diagramm an Hilfslinien ausrichten möchten.



Auf dem Register **Bearbeiten** können Sie folgende Einstellungen definieren:

- Ob ein neues auf dem Register "Modell-Struktur" erstelltes Diagramm im Hauptfenster automatisch geöffnet werden soll.
- Standardsichtbarkeit beim Hinzufügen neuer Elemente - Eigenschaften oder Operationen.
- die Standard-Codesprache, wenn eine neue Komponente hinzugefügt wird.
- Ob eine neu hinzugefügte Einschränkung auch ihren Besitzer (Owner) einschränken soll.
- Ob Sie gefragt werden möchten, bevor Elemente aus einem Projekt, vom Register **Favoriten** oder aus einem der Diagramme gelöscht werden. Wenn Sie Elemente hier löschen, kann diese Eingabeaufforderung deaktiviert werden. Sie können die Eingabeaufforderung "vor dem Löschen fragen" hier zurücksetzen.
- Die Zeit nach der Syntaxfehler-Popup-Fenster geschlossen werden sollen.

© 2016-2022 Altova GmbH

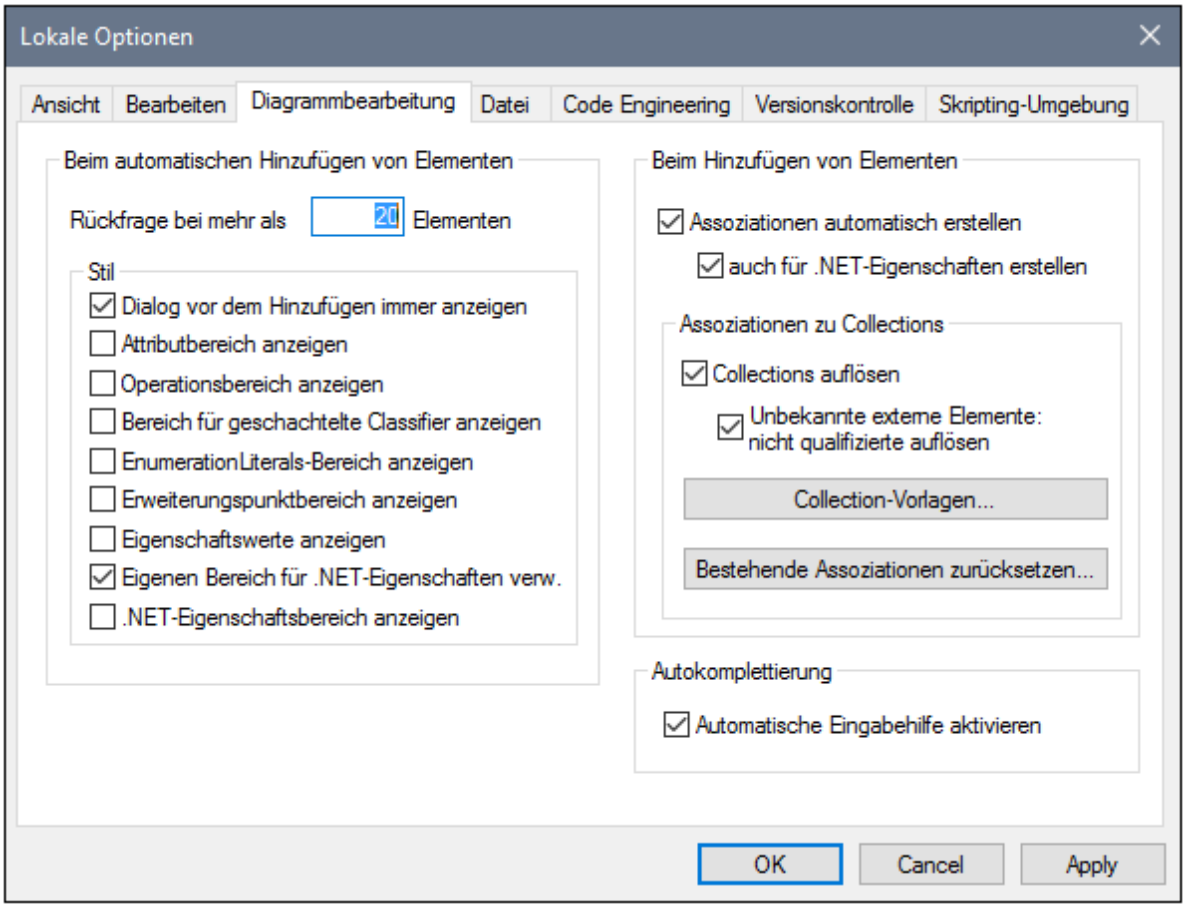


Auf dem Register **Diagrammbearbeitung** können Sie folgende Einstellungen definieren:

- wie viele Objekte automatisch zu einem Diagramm hinzugefügt werden können, bevor eine Eingabeaufforderung erscheint.
- die Anzeige von Stilen, wenn diese automatisch zu einem Diagramm hinzugefügt werden.
- ob automatisch Assoziationen zwischen Modellelementen erstellt werden sollen, wenn Elemente zu einem Diagramm hinzugefügt werden.
- ob die Assoziationen zu Collections aufgelöst werden sollen.
- ob Vorlagen von unbekannten externen Elementen als nicht vollständig qualifiziert aufgelöst werden sollen.
- ob bestehende Collection-Vorlagen verwendet oder neue definiert werden sollen.

  Collection-Vorlagen sollten als voll qualifiziert definiert werden, also a.b.c.List. Wenn die Vorlage diesen Namespace hat, erstellt UModel automatisch eine Collection-Assoziation. Ausnahme: Wenn die Vorlage zum Paket der unbekannten externen Elemente gehört und die Option "Unbekannte externe Elemente: nicht qualifizierte auflösen" aktiviert ist, so wird nur der Vorlagenname berücksichtigt (d.h. List anstelle von a.b.c.List).

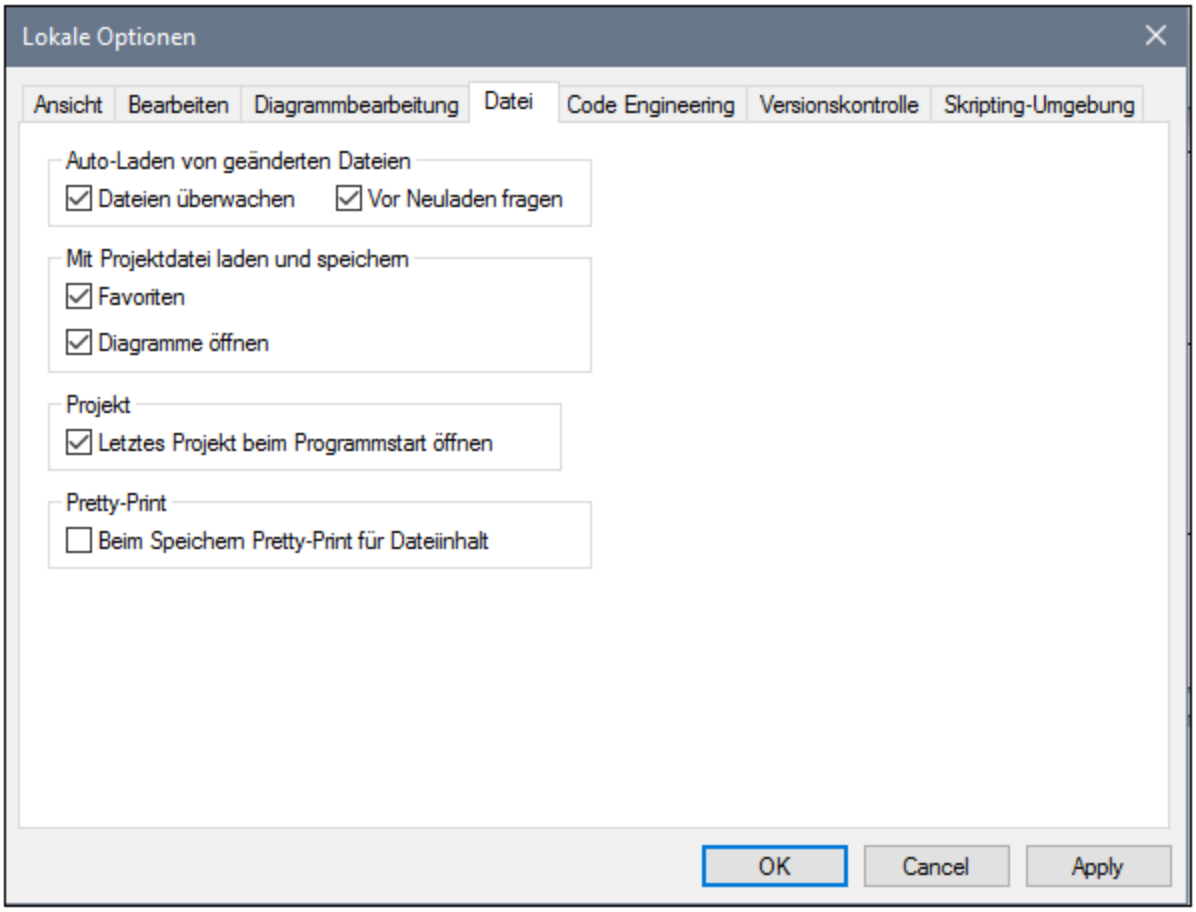
- Ob das Autokomplettierungsfenster bei der Bearbeitung von Attributen oder Operationen im Klassendiagramm verfügbar sein soll.



Auf dem Register **Datei** können Sie folgende Einstellungen definieren:

- welche Aktionen durchgeführt werden sollen, wenn Dateien geändert werden.
- ob der Inhalt des Registers "Favoriten" sowie alle derzeit offenen Diagramme mit dem aktuellen Projekt geladen und gespeichert werden soll.
- ob das zuletzt geöffnete Projekt beim Starten der Applikation automatisch geöffnet werden soll.
- ob die Projektdatei mit CR/LF-Zeilen und Tabulatoreinrückungen im Pretty-Print-Format gespeichert werden soll.

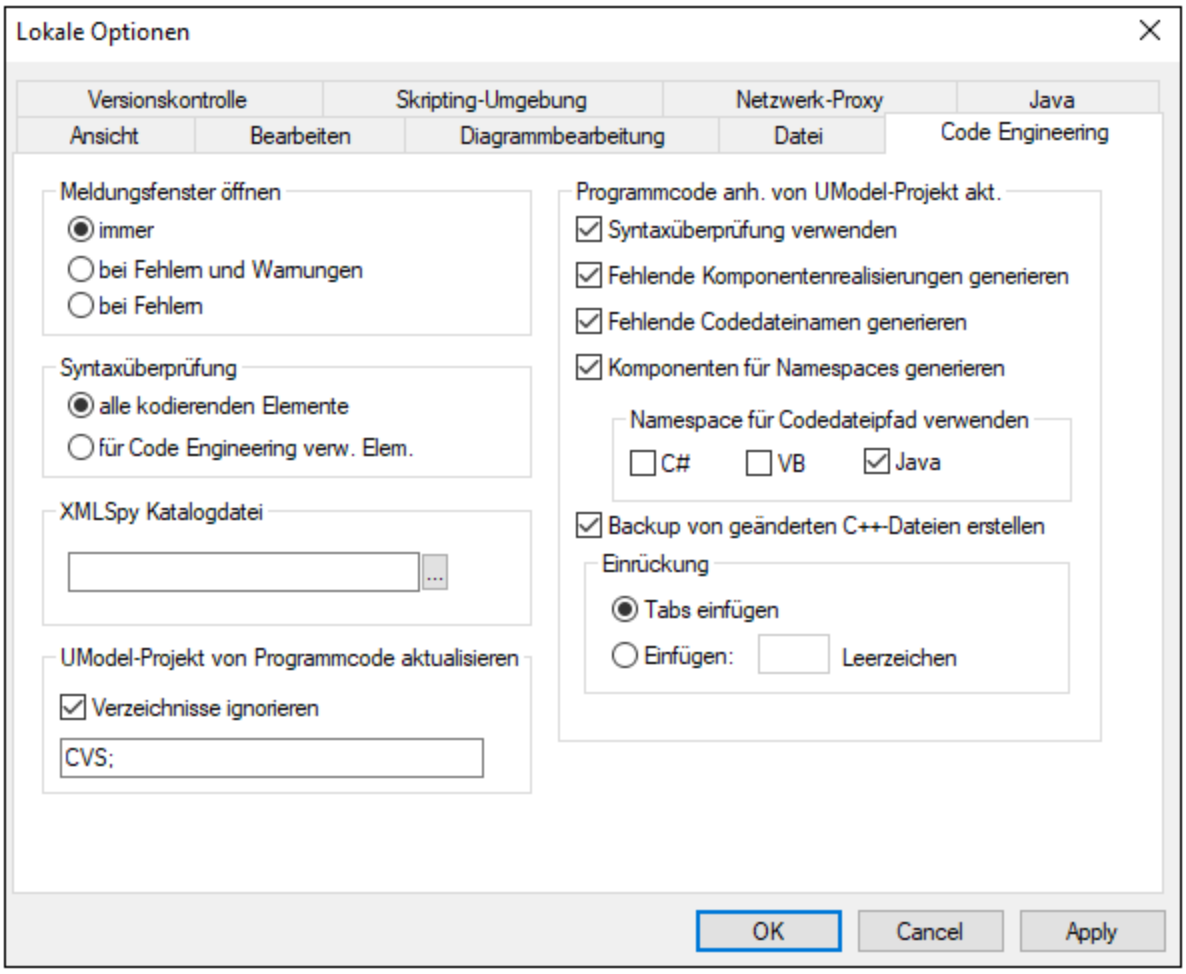
© 2016-2022 Altova GmbH



Auf dem Register **Code Engineering** können Sie folgende Parameter definieren:

- die Umstände, unter denen das Meldungsfenster angezeigt werden soll.
- ob **alle Code-Elemente**, d.h. jene, die in einer Java-, C# - oder VB-Namespace Root enthalten sind, sowie jene, die einer Java-, C#- oder VB-Komponente zugewiesen sind, überprüft werden sollen oder ob nur **Elemente, die für das Code Engineering verwendet werden**, d.h. Elemente, bei denen das Kontrollkästchen "für Code Engineering verwenden" aktiv ist, überprüft werden sollen.
- ob im zusammenzuführenden Code fehlende Codedateinamen generiert werden sollen.
- Beim Aktualisieren von Programmcode:
  - ob eine Syntaxüberprüfung durchgeführt werden soll.
  - ob fehlende Komponentenrealisierungen automatisch generiert werden sollen.
  - ob im zusammengeführten Code fehlende Codedateinamen generiert werden sollen
  - ob im Codedateipfad Namespaces verwendet werden sollen
- Die im Code verwendete Einrückungsmethode, d.h. Tabulatoren oder beliebig viele Leerzeichen
- welche Verzeichnisse beim Aktualisieren eines UModel-Projekts anhand von Code oder einem Verzeichnis ignoriert werden sollen. Trennen Sie die einzelnen Verzeichnisse durch ein Semikolon ";". Untergeordnete Verzeichnisse desselben Namens werden ebenfalls ignoriert.
- den Pfad zur XMLSpy-Datei **RootCatalog.xml**, welche es UModel sowie XMLSpy erlaubt, allgemein verwendete Schemas (sowie Stylesheets und andere Dateien) aus lokalen Benutzerordnern aufzurufen. Dadurch wird die allgemeine Verarbeitungsgeschwindigkeit erhöht und Benutzer können auf diese Art auch offline arbeiten.

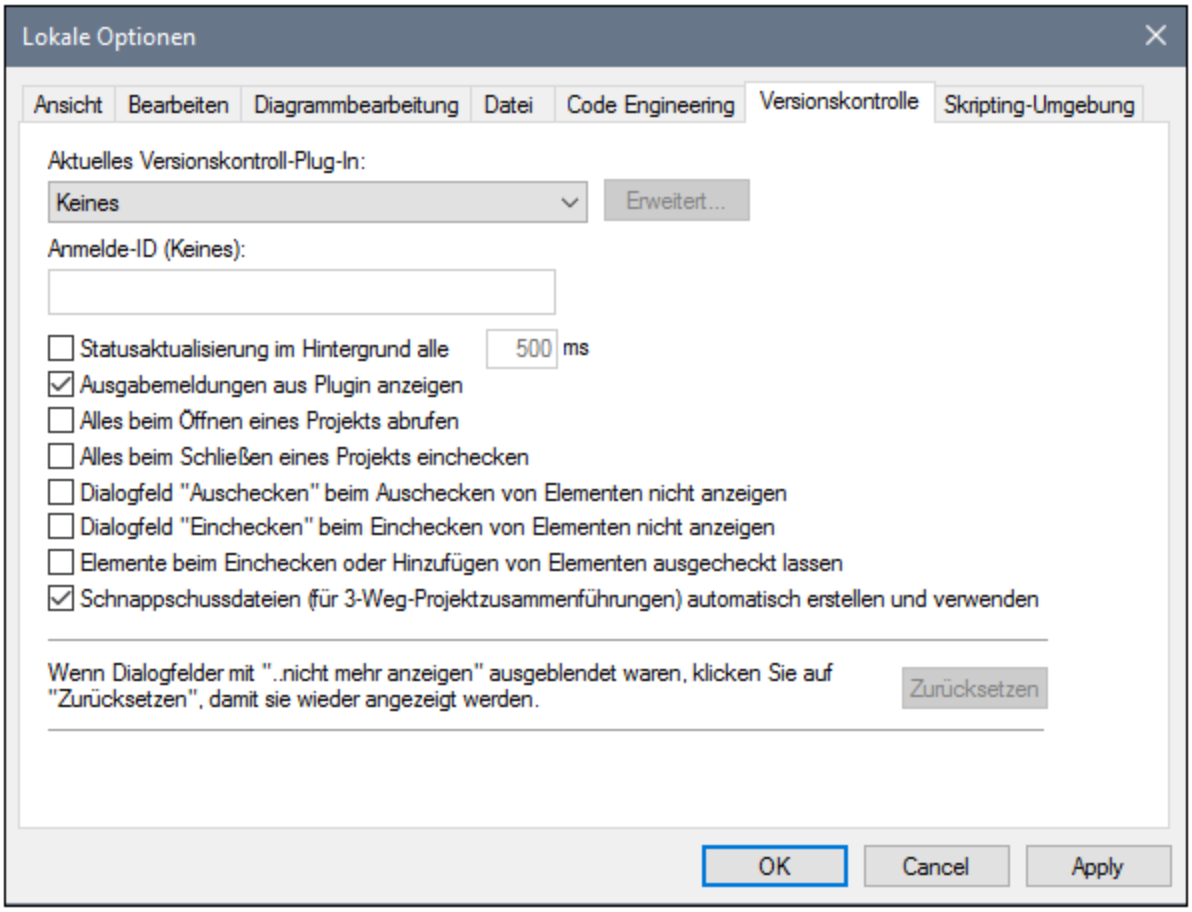
- Außerdem können Sie festlegen, ob eine Sicherungskopie von geänderten C++-Dateien angelegt werden soll.



Auf dem Register **Versionskontrolle** können Sie folgende Einstellungen vornehmen:

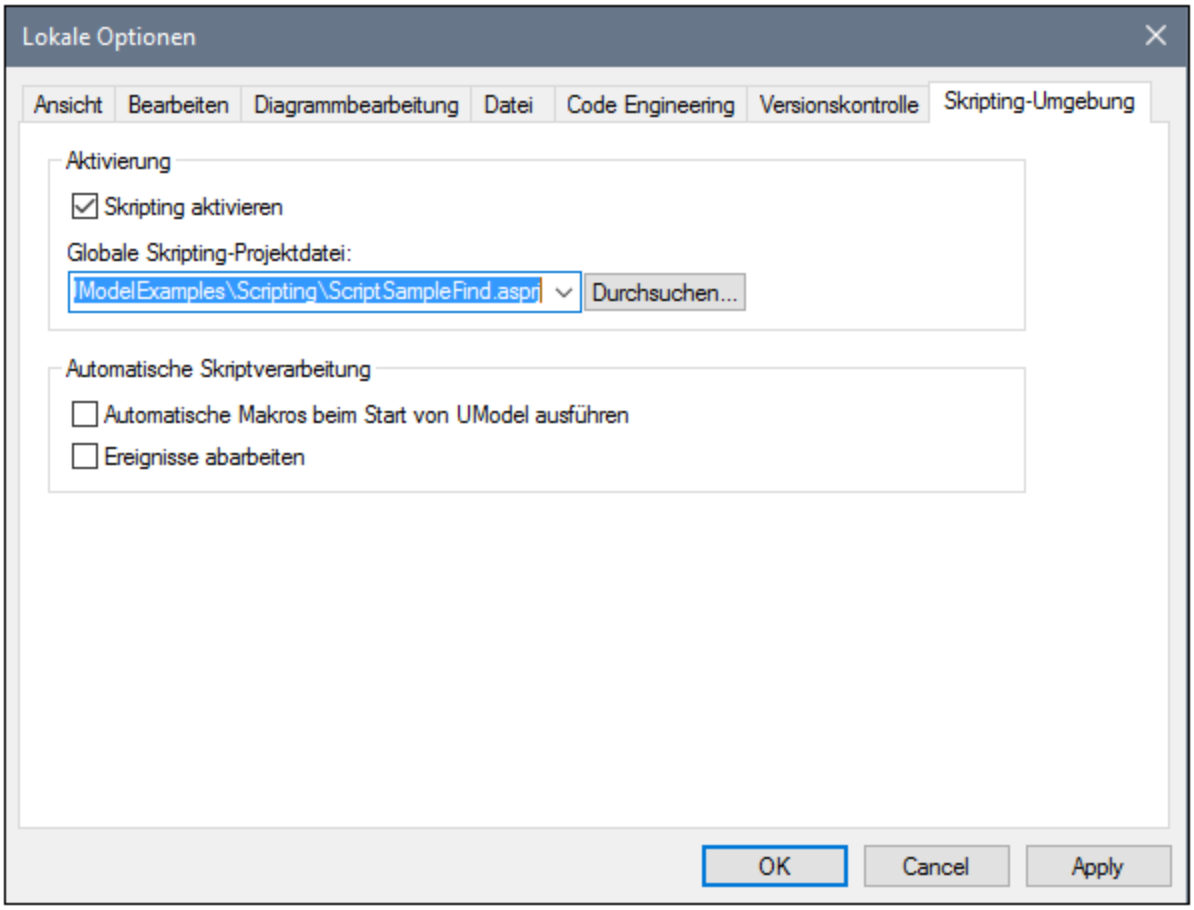
- Auswahl des aktuellen Versionskontroll-Plug-in über die Auswahlliste. Über die Schaltfläche "Erweitert" können Sie die genauen Einstellungen für das ausgewählte Versionskontroll-Plug-in definieren. Diese Einstellungen sind unterschiedlich, je nachdem, welches Versionskontroll-Plug-in Sie verwenden.
- Die Anmelde-ID für das Versionskontrollsystem.
- Spezifische Ein- und Auscheck-Einstellungen.
- Die Schaltfläche "Zurücksetzen" steht zur Verfügung, wenn Sie in einem der Dialogfelder die Option "Wenn Dialogfelder mit "..nicht mehr anzeigen" ausgeblendet waren, klicken Sie auf "Zurücksetzen", damit sie wieder angezeigt werden" aktiviert haben. Daraufhin wird diese "nicht mehr anzeigen"-Meldung wieder angezeigt.

© 2016-2022 Altova GmbH



Auf dem Register **Skripting-Umgebung** können Sie folgende Einstellungen vornehmen:

- ob die Skripting-Umgebung[809] für das aktuelle UModel-Projekt aktiv sein soll.
- welche globale Skripting-Datei Sie verwenden möchten
- wenn beim Start von UModel automatische Makros ausgeführt werden sollen
- ob Skripting-Ereignisse verarbeitet werden sollen.



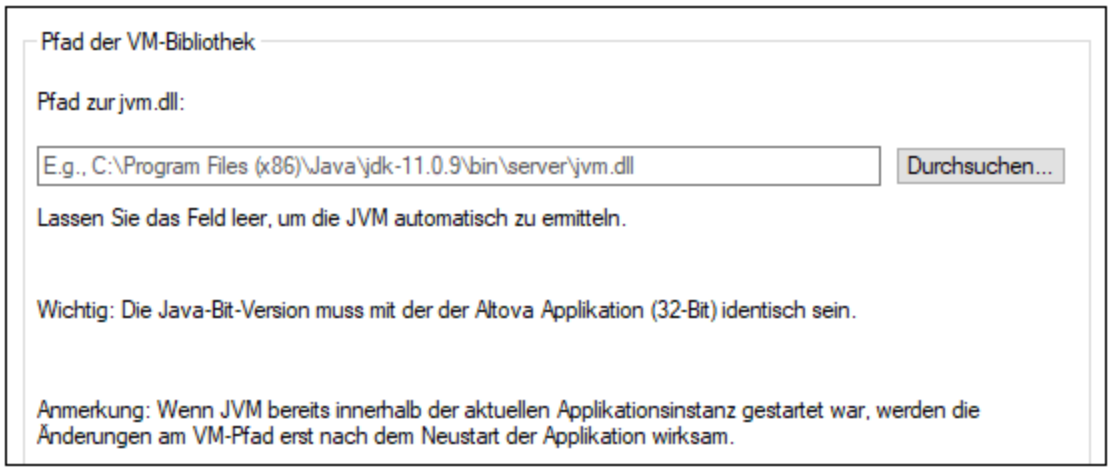
Informationen zu den Einstellungen auf dem Register **Netzwerk-Proxy** finden Sie unter Netzwerk-Proxy-Einstellungen [789]. Nähere Informationen zu Java VM-Einstellungen finden Sie unter Java Virtual Machine-Einstellungen [788].

16.6.8.1 Java Virtual Machine-Einstellungen

Im Abschnitt *Java* (*siehe Abbildung unten*) haben Sie die Möglichkeit, den Pfad zu einer Java VM (Virtual Machine) auf Ihrem Dateisystem einzugeben. Beachten Sie, dass dies nicht immer notwendig ist. UModel versucht standardmäßig den Java VM-Pfad automatisch zu ermitteln. Dazu wird zuerst die Windows Registry und anschließend die JAVA_HOME-Umgebungsvariable gelesen. Ein in dieses Dialogfeld eingegebener benutzerdefinierter Pfad hat Vorrang vor allen automatisch ermittelten Java VM-Pfaden.

Wenn Sie eine Java Virtual Machine verwenden, die keinen Installer hat und keine Registry-Einträge erstellt (z.B. OpenJDK von Oracle), müssen Sie eventuell einen benutzerdefinierten Java VM-Pfad angeben. Auch wenn Sie automatisch von UModel ermittelte Java VM-Pfade aus irgendeinem Grund außer Kraft setzen müssen, müssen Sie diesen Pfad eventuell definieren.

© 2016-2022 Altova GmbH



Beachten Sie dazu Folgendes:

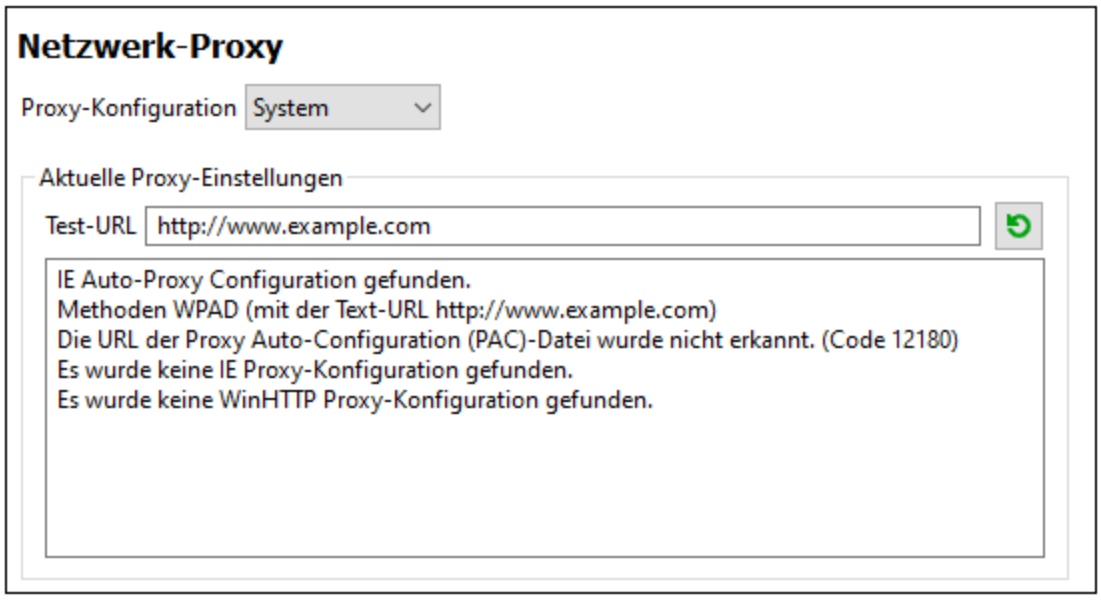
- Der Java VM-Pfad wird gemeinsam von allen Altova Desktop-Applikationen (nicht aber den Server-Applikationen) verwendet. Wenn Sie den Pfad daher in einer Applikation ändern, gilt dies automatisch auch für alle anderen Altova-Applikationen.
- Der Pfad muss auf die Datei **`jvm.dll`** im Verzeichnis **`\bin\server`** oder **`\bin\client`** (relativ zum Verzeichnis, in dem JDK installiert ist) verweisen.
- Die UModel-Plattform (32-Bit, 64-Bit) muss mit der des JDK identisch sein.
- Nachdem Sie den Java VM-Pfad geändert haben, müssen Sie UModel eventuell neu starten, damit die neuen Einstellungen wirksam werden.

Eine Änderung des Java VM-Pfads wirkt sich auf Datenbankverbindungen über JDBC aus.Diese Einstellung hat keine Auswirkung auf die Java-Codegenerierung und den Import von Java-Code. Beachten Sie, dass die für den Import von Java-Binärdateien in UModel verwendeten Java Runtimes separat konfiguriert werden können, siehe Hinzufügen von benutzerdefinierten Java Runtimes 226.

16.6.8.2 Netzwerk-Proxy-Einstellungen

Im Abschnitt *Netzwerk-Proxy* können Sie die benutzerdefinierten Proxy-Einstellungen konfigurieren. Diese Einstellungen beeinflussen, wie die Applikation eine Verbindung mit dem Internet herstellt (z.B. zur XML-Validierung). Standardmäßig werden die Proxy-Einstellungen des Systems verwendet, d.h. die Einstellungen funktionieren, ohne dass der Benutzer etwas daran ändern muss. Falls nötig, können Sie jedoch einen anderen Netzwerk-Proxy-Server definieren. Wählen Sie dazu in der Auswahlliste *Proxy-Konfiguration* entweder die Option *Automatisch* oder *Manuell*, um die Einstellungen entsprechend zu konfigurieren.

Anmerkung: Die Nethwerk-Proxy-Einstellungen werden von allen Altova MissionKit-Applikationen gemeinsam verwendet. Wenn Sie daher die Einstellungen in einer Applikation ändern, wirkt sich dies automatisch auf alle anderen Applikationen aus.



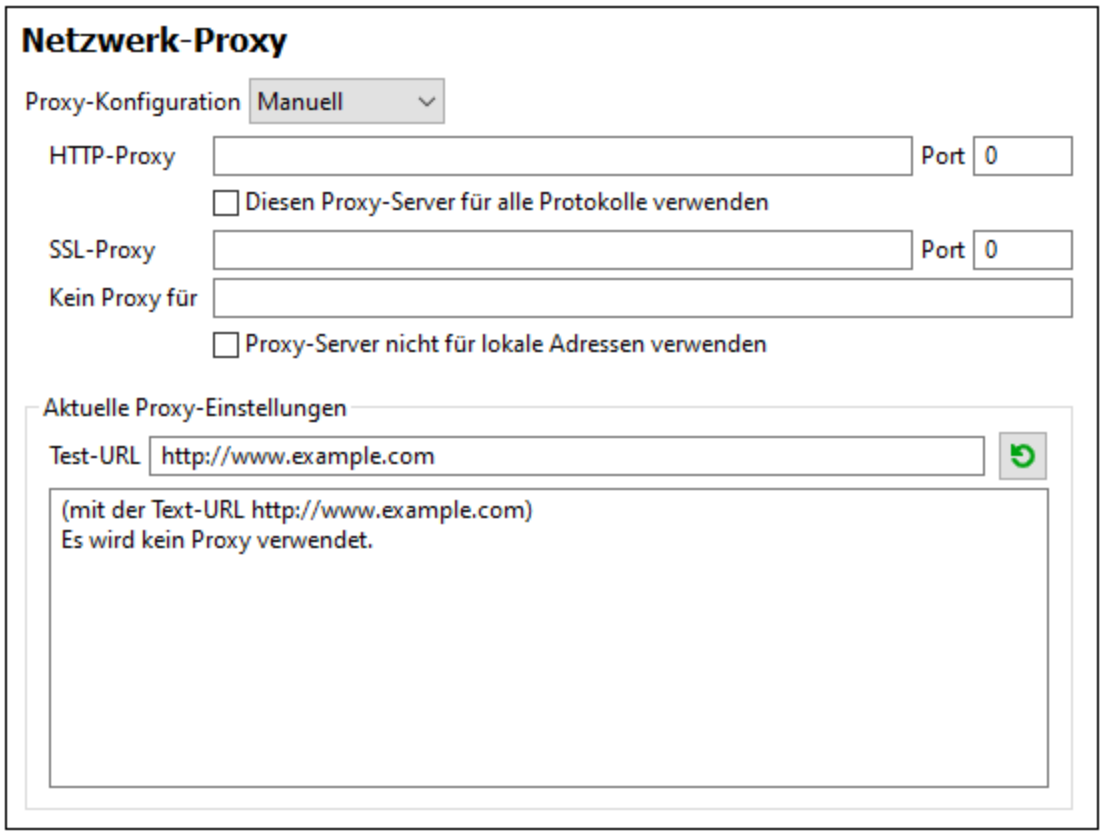
System-Proxy-Einstellungen verwenden
Dadurch werden die über die System-Proxy-Einstellungen konfigurierbaren Internet Explorer (IE)-Einstellungen verwendet. Führt außerdem eine Abfrage der mit **`netsh.exe winhttp`** konfigurierten Einstellungen durch.

Automatische Proxy-Konfiguration
Es stehen die folgenden Optionen zur Verfügung:

- *Einstellungen automatisch ermitteln:* **`Verwendet ein WPAD-Skript`** (http://wpad.LOCALDOMAIN/wpad.dat) über DHCP oder DNS, um die Einrichtung des Proxy-Servers zu konfigurieren.
- *Skript-URL:* Definieren Sie eine HTTP URL zu einem automatischen Proxy-Konfigurationsskript (**`.pac`**), mit dem der Proxy-Server eingerichtet wird.
- *Neu laden:* Setzt die aktuelle automatische Proxy-Konfiguration zurück und lädt sie neu. Dafür ist Windows 8 oder neuer erforderlich. Die Rücksetzung kann bis zu 30 Sekunden dauern.

Manuelle Proxy-Konfiguration
Definieren Sie den vollständig qualifizierten Host-Namen und Port für die Proxy-Server der jeweiligen Protokolle manuell. Im Host-Namen kann ein unterstütztes Schema inkludiert werden (z.B.: **`http://hostname`**). Das Schema muss nicht mit dem entsprechenden Protokoll übereinstimmen, wenn der Proxy-Server das Schema unterstützt.

© 2016-2022 Altova GmbH



Es stehen die folgenden Optionen zur Verfügung:

- *HTTP-Proxy:* Verwendet den angegebenen Host-Namen und Port für das HTTP-Protokoll. Wenn *Diesen Proxy-Server für alle Protokolle verwenden* aktiviert ist, wird der angegebene HTTP-Proxy-Server für alle Protokolle verwendet.
- *SSL-Proxy:* Verwendet den angegebenen Host-Namen und Port für das SSL-Protokoll.
- *Kein Proxy für:* eine durch Semikola (`;`) getrennte Liste von voll qualifizierten Host-Namen, Domain-Namen oder IP-Adressen für Hosts, die ohne einen Proxy-Server verwendet werden sollen. IP-Adressen dürfen nicht abgeschnitten werden und IPv6-Adressen müssen innerhalb von eckige Klammern gesetzt werden (z.B.: `[2606:2800:220:1:248:1893:25c8:1946]`). Domain-Namen muss ein Punkt vorangestellt werden (z.B: `.example.com`).
- *Proxy-Server nicht für lokale Adressen verwenden:* Falls dieses Kontrollkästchen aktiviert ist, wird `<local>` zur *Kein Proxy für*-Liste hinzugefügt. Falls diese Option ausgewählt ist, wird für die folgenden Adressen kein Proxy-Server verwendet: (i) `127.0.0.1`, (ii) [::1], (iii) alle Host-Namen, die kein Punktzeichen (`.`) enthalten.

Aktuelle Proxy-Einstellungen
Stellt ein ausführliches Protokoll der Proxy-Ermittlung bereit. Es kann über die Schaltfläche **Aktualisieren** rechts vom Feld *Test-URL* aktualisiert werden (z.B. bei Wechsel zu einer anderen Test-URL oder wenn die Proxy-Einstellungen geändert wurden).

- *Test-URL:* Anhand einer Test-URL kann ermittelt werden, welcher Proxy-Server für diese bestimmte URL verwendet wird. Mit dieser URL erfolgt kein I/O. Dieses Feld darf nicht leer sein, wenn die automatische Proxy-Konfiguration verwendet wird (entweder über *System-Proxy-Einstellungen*

verwenden oder *Automatische Proxy-Konfiguration*).

© 2016-2022 Altova GmbH

16.7 Fenster

Überlappend

Mit diesem Befehl werden alle offenen Dokumentenfenster so angeordnet, dass sie einander überlappend angezeigt werden.

Horizontal anordnen

Mit diesem Befehl werden alle offenen Dokumentenfenster horizontal nebeneinander angeordnet, sodass alle gleichzeitig sichtbar sind.

Vertikal anordnen

Mit diesem Befehl werden alle offenen Dokumentenfenster vertikal übereinander angeordnet, sodass alle gleichzeitig sichtbar sind.

Symbole anordnen

Damit werden Diagramme, die beliebig und in Symbolform angeordnet sind, entlang des unteren Rands des Diagrammansichtsbereichs positioniert.

Schließen

Schließt das derzeit aktive Diagrammregister.

Alle schließen

Schließt alle derzeit geöffneten Diagrammregister.

Alle bis auf das aktive schließen

Schließt alle Diagrammregister mit Ausnahme des derzeit aktiven.

Vorwärts

Immer wenn Sie den Fokus von einem Diagrammfenster auf ein anderes verlegen oder auf einen Hyperlink klicken, merkt sich UModel dies als Ereignis. Mit diesem Befehl gelangen Sie im Verlauf solcher Ereignisse vorwärts. Der Befehl steht nur dann zur Verfügung, wenn Sie den Befehl **Zurück** bereits verwendet haben (siehe unten).

Zurück

Mit diesem Befehl gelangen Sie zu dem Fenster zurück, das sich zuvor im Fokus befunden hat. Dies kann nützlich sein, wenn Sie gleichzeitig mit mehreren Diagrammfenstern arbeiten oder wenn Sie mittels Hyperlinks navigieren, siehe Erstellen von Hyperlinks zu Elementen 122.

Fensterliste (1,2)

In dieser Liste werden alle derzeit offenen Fenster angezeigt, sodass Sie zwischen ihnen wechseln können. Sie können auch mit Hilfe der Tastaturkürzel **Strg-Tab** oder **Strg F6** zwischen den offenen Fenstern hin- und herwechseln.

Fenster

Ruft ein Dialogfeld auf, in dem Sie das Layout für mehrere Diagrammfenster gleichzeitig bestimmen bzw. mehrere Fenster gleichzeitig schließen können, siehe auch Diagrammbereich[102].

© 2016-2022 Altova GmbH

16.8 Hilfe

Inhaltsverzeichnis

Mit dem Befehl "Inhaltsverzeichnis" wird die Bildschirmhilfe für UModel geöffnet, wobei das Inhaltsverzeichnis auf der linken Seite des Hilfefensters angezeigt wird. Das Inhaltsverzeichnis bietet eine Übersicht über das gesamte Hilfedokument. Wenn Sie auf einen Eintrag im Inhaltsverzeichnis klicken, gelangen Sie zu diesem Thema.

Index

Mit dem Befehl Index wird die Bildschirmhilfe für UModel geöffnet, wobei auf der linken Seite des Hilfefensters der Stichwortindex angezeigt wird. Im Index werden Stichwörter aufgelistet und Sie können durch Doppelklick auf das Stichwort zum jeweiligen Thema navigieren. Wenn ein Stichwort mit mehr als einem Thema verknüpft ist, können Sie das Thema aus einer Liste von Themen auswählen.

Suchen

Mit dem Befehl Suchen wird die Bildschirmhilfe für UModel geöffnet, wobei auf der linken Seite des Hilfefensters das Suchdialogfeld angezeigt wird. Um nach einem Begriff zu suchen, geben Sie ihn in das Eingabefeld ein und drücken Sie die **Eingabetaste** oder klicken Sie auf **Kapitel auflisten**. Das Hilfesystem führt eine Volltextsuche am gesamten Hilfedokument durch und gibt eine Trefferliste zurück. Doppelklicken Sie auf einen Eintrag, um das Thema anzuzeigen.

Software-Aktivierung

Lizenzieren Ihres Produkts
Nachdem Sie Ihre Altova-Software heruntergeladen haben, können Sie sie entweder mit Hilfe eines kostenlosen Evaluierungs-Keycode oder eines käuflich erworbenen permanenten Lizenzkeycode lizenzieren oder aktivieren.

- ***Kostenlose Evaluierungs-Lizenz.*** Wenn Sie die Software zum ersten Mal starten, wird das Dialogfeld **Software-Aktivierung** angezeigt. Es enthält eine Schaltfläche, über die Sie eine kostenlose Evaluierungs-Lizenz anfordern können. Geben Sie in das Dialogfeld Ihren Namen, den Namen Ihrer Firma und Ihre E-Mail-Adresse ein und klicken Sie auf **Anfordern**. Daraufhin wird an die von Ihnen angegebene E-Mail-Adresse eine Lizenz gesendet und sollte in wenigen Minuten eintreffen. Speichern Sie die Lizenzdatei in einem geeigneten Ordner.

  Wenn Sie auf **Anfordern** klicken, wird am unteren Rand des Dialogfelds ein Eingabefeld angezeigt. Über dieses Feld gelangen Sie zum Pfad der Lizenzdatei. Navigieren Sie zur Lizenzdatei oder geben Sie den Pfad dazu ein und klicken Sie auf **OK**. (Sie können im Dialogfeld **Software-Aktivierung** auch auf **Neue Lizenz hochladen** klicken, um das Dialogfeld, in das der Pfad zur Lizenzdatei eingegeben wird, aufzurufen). Die Software wird für einen Zeitraum von 30 Tagen aktiviert.

- ***Permanenter Lizenz-Keycode.*** Über das Dialogfeld **Software-Aktivierung** können Sie einen permanenten Lizenz-Keycode erwerben. Wenn Sie auf diese Schaltfläche klicken, gelangen Sie zum Altova Online Shop, in dem Sie einen permanenten Lizenzschlüssel für Ihr Produkt erwerben können. Ihre Lizenz wird Ihnen in Form einer Lizenzdatei, die Ihre Lizenzdaten enthält, per E-Mail zugesendet.

  Es gibt drei Arten von permanenten Lizenzen: *Einzelplatzlizenzen*, *Parallellizenzen* und *Named User-Lizenzen* (benutzerdefinierte Nutzung). Mit einer Einzelplatzlizenz wird die Software auf

einem einzigen Rechner freigeschaltet. Wenn Sie eine Einzelplatzlizenz für *N* Rechner erwerben, gestattet Ihnen die Lizenz, die Software auf bis zu *N* Rechnern zu verwenden. Mit einer Parallellizenz für *N* Parallelbenutzer dürfen *N* Benutzer die Software gleichzeitig ausführen. (Die Software darf auf 10N Rechnern installiert sein.) Mit einer Named User-Lizenz darf ein bestimmter Benutzer die Software auf bis zu 5 verschiedenen Rechnern verwenden. Um Ihre Software zu aktivieren, klicken Sie auf **Neue Lizenz hochladen** und geben Sie im daraufhin angezeigten Dialogfeld den Pfad zur Lizenzdatei ein und klicken Sie auf **OK**.

Anmerkung: Bei Mehrplatzlizenzen wird jeder Benutzer aufgefordert, seinen eigenen Namen einzugeben.

Ihre Lizenz-E-Mail und die verschiedenen Methoden, Ihr Altova-Produkt zu lizenzieren
Die Lizenz-E-Mail, die Sie von Altova erhalten, enthält Ihre Lizenzdatei im Anhang. Die Lizenzdatei hat die Dateierweiterung `.altova_licenses`.

Sie haben folgende Möglichkeiten, Ihr Altova-Produkt zu aktivieren:

- Speichern Sie die Lizenzdatei (`.altova_licenses`) in einem geeigneten Ordner, doppelklicken Sie auf die Lizenzdatei, geben Sie etwaige erforderliche Informationen in das Dialogfeld ein, das daraufhin angezeigt wird und beenden Sie den Vorgang durch Klicken auf **Lizenzschlüssel anwenden**.
- Speichern Sie die Lizenzdatei (`.altova_licenses`) in einem geeigneten Ordner. Wählen Sie in Ihrem Altova-Produkt den Menübefehl **Hilfe | Software-Aktivierung** und klicken Sie anschließend auf **Neue Lizenz hochladen.** Navigieren Sie zur Lizenzdatei oder geben Sie den Pfad dazu ein und klicken Sie auf **OK**.
- Speichern Sie die Lizenzdatei (`.altova_licenses`) in einem geeigneten Ordner und laden Sie diese von dort aus in den Lizenz-Pool Ihres Altova LicenseServer hoch. Sie können die Lizenz anschließend (i) entweder von Ihrem Altova-Produkt über das Dialogfeld "Software-Aktivierung" abrufen (*siehe unten*) oder (ii) dem Produkt die Lizenz von Altova LicenseServer aus zuweisen. *Nähere Informationen zur Lizenzierung über LicenseServer finden Sie weiter unten in diesem Kapitel.*

Das Dialogfeld **Software-Aktivierung** (*Abbildung unten*) kann über den Befehl **Hilfe | Software-Aktivierung** aufgerufen werden.

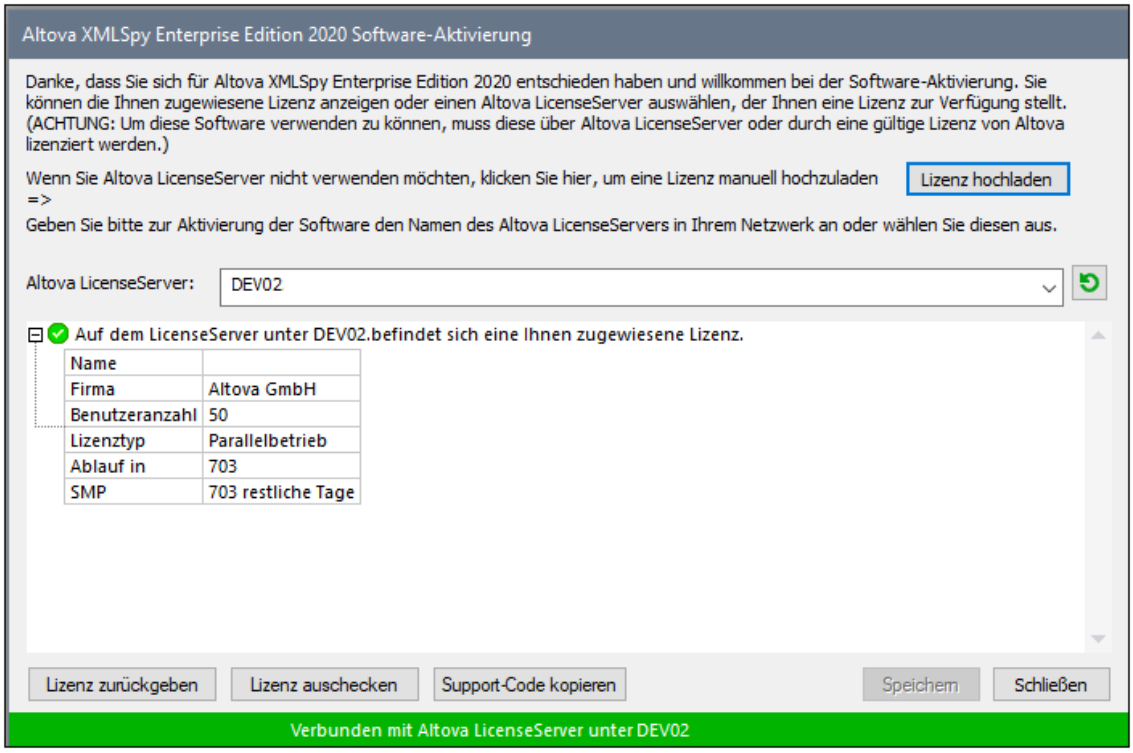
Aktivieren Ihrer Software
Sie können die Software durch Registrieren der Lizenz im Dialogfeld "Software-Aktivierung" oder durch Lizenzierung über Altova LicenseServer (*nähere Informationen siehe unten*) aktivieren.

- *Registrierung der Lizenz im Dialogfeld "Software-Aktivierung".* Klicken Sie im Dialogfeld auf **Neue Lizenz hochladen** und navigieren Sie zur Lizenzdatei. Klicken Sie auf **OK,** um den Pfad zur Lizenzdatei und alle eingegebenen Daten (im Fall einer Mehrplatzlizenz Ihren Namen) zu bestätigen und abschließend auf **Speichern**.

- *Lizenzierung über einen Altova LicenseServer in Ihrem Netzwerk:* Um eine Lizenz über einen Altova LicenseServer in Ihrem Netzwerk abzurufen, (klicken Sie am unteren Rand des Dialogfelds **Software-Aktivierung** auf **Altova LicenseServer verwenden**). Wählen Sie den Rechner aus, auf dem der gewünschte LicenseServer installiert wurde. Beachten Sie, dass die automatische Ermittlung von License Servern durch die Aussendung eines Signals ins LAN erfolgt. Da diese

© 2016-2022 Altova GmbH

Aussendung auf ein Subnetz beschränkt ist, muss sich der LicenseServer im selben Subnetz wie der Client-Rechner befinden, damit die Ermittlung von License Servern funktioniert. Falls die automatische Ermittlung nicht funktioniert, geben Sie den Namen des Servers ein. Der Altova LicenseServer muss in seinem Lizenzpool eine Lizenz für Ihre Altova-Produkt haben. Wenn im LicenseServer-Pool eine Lizenz verfügbar ist, wird dies im Dialogfeld **Software-Aktivierung** angezeigt (*siehe Abbildung unten, in der Sie das Dialogfeld in Altova XMLSpy sehen*) und Sie können auf **Speichern** klicken, um die Lizenz abzurufen.



Eine rechnerspezifische Lizenz (Einzelplatzlizenz) kann erst nach Ablauf von sieben Tagen wieder an LicenseServer zurückgegeben werden. Danach können Sie die rechnerspezifische Lizenz durch Klick auf **Lizenz zurückgeben** an den Server zurückgeben, sodass sie von einem anderen Client vom LicenseServer abgerufen werden kann. Ein LicenseServer-Administrator kann die Zuweisung einer abgerufenen Lizenz jedoch über die Web-Benutzeroberfläche von LicenseServer jederzeit aufheben. Beachten Sie, dass eine Rückgabe von Lizenzen nur bei rechnerspezifischen Lizenzen, nicht aber bei Parallellizenzen möglich ist.

Lizenz-Check-Out
Über den Lizenzpool können Sie eine Lizenz für einen Zeitraum von bis zu 30 Tagen auschecken, sodass die Lizenz auf dem lokalen Rechner gespeichert wird. Dadurch können Sie offline arbeiten, was nützlich ist, wenn Sie z.B. in einer Umgebung arbeiten möchten, in der Sie keinen Zugriff auf Ihren Altova LicenseServer haben (z.B. wenn Ihr Altova-Produkt auf einem Laptop installiert ist und Sie gerade unterwegs sind). Solange die Lizenz ausgecheckt ist, zeigt LicenseServer die Lizenz als in Verwendung an. Diese Lizenz kann dann von keinem anderen Rechner verwendet werden. Die Lizenz wird nach Ablauf des Check-Out-Zeitraums automatisch wieder eingecheckt. Alternativ dazu kann eine ausgecheckte Lizenz jederzeit über die Schaltfläche **Einchecken** des Dialogfelds **Software-Aktivierung** wieder eingecheckt werden.

Um eine Lizenz auszuchecken, gehen Sie folgendermaßen vor: (i) Klicken Sie im Dialogfeld

Software-Aktivierung auf **Lizenz auschecken** (*siehe Abbildung oben*); (ii) Wählen Sie im daraufhin angezeigten Dialogfeld **Lizenz-Check-Out** den gewünschten Check-Out-Zeitraum aus und klicken Sie auf **Auschecken**. Daraufhin wird die Lizenz ausgecheckt. Nachdem Sie eine Lizenz ausgecheckt haben, geschehen zwei Dinge: (i) Die Check-Out-Informationen und das Ende des Check-Out-Zeitraums werden im Dialogfeld **Software-Aktivierung** angezeigt; (ii) Die Schaltfläche **Lizenz auschecken** im Dialogfeld ändert sich nun in **Einchecken**. Sie können die Lizenz jederzeit durch Klicken auf **Einchecken** einchecken. Da die Lizenz nach Ablauf des Check-Out-Zeitraums automatisch wieder in den Zustand "Eingecheckt" zurück wechselt, sollte der von Ihnen ausgewählte Zeitraum für das Check-Out den gewünschten Zeitraum, in dem Sie offline arbeiten möchten, entsprechend abdecken.

Wenn eine Lizenz wieder eingecheckt wird, muss diese Lizenz für dieselbe Hauptversion eines Altova-Produkts ausgestellt sein, wie die Lizenz, die ausgecheckt wurde. Stellen Sie daher sicher, dass die Lizenz eingecheckt ist, bevor Sie für Ihr Altova-Produkt ein Upgrade auf die nächste Hauptversion installieren.

Anmerkung: Damit Lizenzen ausgecheckt werden können, muss die Check-Out-Funktion auf dem LicenseServer aktiviert werden. Wenn diese Funktion nicht aktiviert wurde, erhalten Sie eine entsprechende Fehlermeldung, wenn Sie versuchen die Lizenz auszuchecken. Wenden Sie sich in diesem Fall an Ihren LicenseServer-Administrator.

Support-Code kopieren
Klicken Sie auf **Support-Code kopieren,** um Lizenzinformationen in die Zwischenablage zu kopieren. Dies sind die Daten, die Sie bei einer Support-Anfrage über das Online Support-Formular benötigen.

Altova LicenseServer bietet IT-Administratoren einen Echtzeitüberblick über alle Altova-Lizenzen in einem Netzwerk. Dazu werden die Einzelheiten zu jeder Lizenz sowie Client-Zuweisungen und die Verwendung von Lizenzen durch Clients angezeigt. Der Vorteil der Verwendung von LicenseServer liegt in seinen Funktionen zur Verwaltung großer Altova-Lizenzpools. Altova LicenseServer steht kostenlos auf der Altova Website zur Verfügung. Nähere Informationen zu Altova LicenseServer und der Lizenzierung mittels Altova LicenseServer finden Sie in der Dokumentation zu Altova LicenseServer.

- Bestellformular

  Sobald Sie eine lizenzierte Version des Software-Produkts bestellen möchten, klicken Sie im Dialogfeld **Software-Aktivierung** (*siehe oben*) auf die Schaltfläche **Permanenten Key-Code erwerben...** oder wählen Sie den Befehl **Bestellformular**, um zum sicheren Online-Shop von Altova weitergeleitet zu werden.

- Registrierung

  Bei Aufruf dieses Befehls wird die Altova-Produktregistrierungsseite auf einem Register Ihres Browsers geöffnet. Durch Registrierung Ihrer Altova-Software stellen Sie sicher, dass Sie immer die neuesten Produktinformationen erhalten.

- Auf Updates überprüfen

  Überprüft, ob am Altova Server eine neuere Version Ihres Produkts vorhanden ist und zeigt eine entsprechende Meldung an.

- Support Center

© 2016-2022 Altova GmbH

Der Befehl "Support Center" ist ein Link zum Altova Support Center im Internet. Im Support Center finden Sie Antworten auf häufig gestellt Fragen, Diskussionsforen, in denen Sie Software-Probleme besprechen können und ein Formular, um unsere Mitarbeiter vom technischen Support zu kontaktieren.

Fragen und Antworten im Web

Der Befehl "Fragen und Antworten im Web" ist ein Link zur Altova-Fragen- und Antworten-Datenbank im Internet. Diese Datenbank wird ständig aktualisiert, wenn die Mitarbeiter von Altova von unseren Kunden mit neuen Fragen konfrontiert werden.

Komponenten und Gratistools downloaden

Dieser Befehl ist ein Link zum Komponenten Download Center von Altova im Internet. Von hier können Sie Software-Komponenten verschiedener anderer Anbieter herunterladen, die Sie mit Altova Produkten verwenden können. Dabei handelt es sich um XSLT- und XSL-FO-Prozessoren, Applikationsserverplattformen usw. Die im Komponenten Download Center verfügbare Software ist normalerweise kostenlos.

UModel im Internet

Der Befehl UModel im Internet ist ein Link zur Altova Website im Internet. Hier erfahren Sie mehr über UModel und verwandte Technologien und Produkte auf der Altova Website.

Über UModel

Mit dem Befehl Über UModel wird das Willkommensfenster und die Versionsnummer Ihres Produkts angezeigt. Wenn Sie die 64-Bit-Verson von UModel verwenden, wird dies durch das Suffix (x64) nach dem Applikationsnamen angezeigt. Die 32-Bit-Version hat kein Suffix.

17 UModel Referenz für Programmierer

UModel ist ein Automation Server, d.h. eine Applikation, die anderen Applikationen, so genannten Automation Clients, programmierbare Objekte zur Verfügung stellt. Ein Automation Client hat direkten Zugriff auf die vom Automation Server bereitgestellten Objekte und Funktionalitäten, sodass sich der Automation Client die Funktionalitäten von UModel zunutze machen kann. Dadurch können Sie von anderen Applikationen aus die fertigen Funktionen von UModel nutzen. Dadurch können Sie von anderen Applikationen aus die fertigen Funktionen von UModel nutzen. Entwickler können daher Ihre Applikationen verbessern, indem Sie die fertigen Funktionalitäten von UModel verwenden.

Die programmierbaren Objekte von UModel stehen den Automation Clients über die UModel API, eine COM API, zur Verfügung. Das Objektmodell der API und eine vollständige Beschreibung aller verfügbaren Objekte finden Sie in dieser Dokumentation (siehe Abschnitt UModel API-Referenz 919).

Die UModel-API kann von den folgenden Umgebungen aus aufgerufen werden:

- Skrip-Editor 809
- IDE Plug-ins 837
- Externe Programme 857

Im Folgenden finden Sie eine Beschreibung der einzelnen Umgebungen.

Skrip-Editor

Sie können Ihre Installation von UModel anpassen, indem Sie sie ändern und Funktionalitäten dazu hinzufügen. Sie können auch Formulare für die Benutzereingabe erstellen und neue Menübefehle und Symbolleisten-Schaltflächen zur Benutzeroberfläche hinzufügen. Zu diesem Zweck werden Skripts geschrieben, die mit Objekten der Applikations-API interagieren. Zur effizienten Ausführung dieser Aufgaben steht in UMod ein integrierter Skript-Editor zur Verfügung. Eine ausführliche Beschreibung der im Skript-Editor verfügbaren Funktionalitäten und eine Anleitung zur Verwendung der Scripting-Umgebung finden Sie im Abschnitt Skript-Editor 809 dieser Dokumentation. Unterstützt werden die Programmiersprachen **JScript** und **VBScript**.

IDE Plug-ins

Sie haben in UModel die Möglichkeit Ihre eigenen Plug-Ins als DLL-Dateien zu schreiben und in UModel zu integrieren. Die grafische Benutzeroberfläche von UModel bietet Befehle, um Plug-ins zu aktivieren oder zu deaktivieren. Normalerweise werden zum Implementieren von IDE-Plug-ins die Sprachen **C#** und **C++** verwendet. Nähere Informationen dazu finden Sie im Abschnitt IDE Plug-ins 837.

Externe Programme

Außerdem können Sie UModel mittels externer Skripts bedienen. So könnten Sie z.B. ein Skript schreiben, um UModel zu einem bestimmten Zeitpunkt zu öffnen, dann ein UModel-Projekt zu öffnen, UML-Dokumentation zu generieren und auszudrucken. Externe Skripts würden sich zur Ausführung dieser Aufgaben wiederum der API bedienen. Eine Beschreibung der API finden Sie im Abschnitt UModel API 857.

Um die UModel-API außerhalb von UModel verwenden zu können, muss zuerst eine Instanz von UModel gestartet werden, siehe Aufruf der API 857.

Im Wesentlichen wird UModel über seine COM Registrierung gestartet. Anschließend wird das mit der UModel-Instanz verknüpfte `Application` Objekt zurückgegeben. Je nach COM-Einstellungen kann ein mit einer bereits

© 2016-2022 Altova GmbH

laufenden UModel-Instanz verknüpftes Objekt zurückgegeben werden. Es kann jede Programmiersprache verwendet werden, die die Erstellung und den Aufruf von COM-Objekten unterstützt. Die am häufigsten dazu verwendeten Sprachen sind unten aufgelistet.

- JScript[904]- und VBScript-Skriptdateien haben eine einfache Syntax und wurden für den Aufruf von COM-Objekten entworfen. Sie können direkt über die Befehlszeile oder durch Doppelklick im Windows Explorer ausgeführt werden. Am besten eignen sich diese Sprachen für einfache Automationsaufgaben.
- C#[876] ist eine umfangreiche Programmiersprache, die Unterstützung für die COM-Interoperabilität bietet.
- Java[903]: Im Lieferumfang von Altova-Produkten sind native Java-Klassen inkludiert, die als Wrapper für die Applikations-API verwendet werden und dadurch eine Java-Umgebung ermöglichen.
- Weitere nützliche Alternativen sind Visual Basic für Applikationen, Perl und Python.

17.1 Anmerkungen zur Release

In der Folge sind für jede Release der UModel API die wichtigsten Änderungen seit der letzten Version aufgelistet. Eine Änderung an der Hauptversion der Typbibliothek (z.B. von 4.0 in 5.0) bedeutet, dass Nicht-Skripting-Clients (z.B. UModel-IDE Plug-Ins, die in C#, VB.NET, C++, usw. geschrieben wurden) neu kompiliert werden sollten.

Automation Interface für UModel 2021r2 - Typbibliothek Version 5.9

UModel API	• Die Enumeration `ENUMCodeLangVersion` hat neue Mitglieder für die Java-Version 15 und für C# 9.0. • Die Enumeration `ENUMUMLPredefinedElement` hat im Zusammenhang mit der Unterstützung für Java 15 eine Reihe neuer Mitglieder.

Automation Interface für UModel 2021 - Typbibliothek Version 5.8

UModel API	• Die Enumeration `ENUMCodeLangVersion` hat neue Mitglieder für die Java-Version 14.0 und für die Datenbankart MariaDB. • Die Enumeration `ENUMUMLPredefinedElement` hat neue Mitglieder für die Java-Version 14.0 und für die Datenbankart MariaDB.

Automation Interface für UModel 2020r2 - Typbibliothek Version 5.7

UModel API	• Die Enumeration `ENUMCodeLangVersion` hat ein neues Mitglied für die Java-Version 13.0. • Die Enumeration `ENUMExportXMIType` hat neue Mitglieder: `eXMI24ForUML25` and `eXMI251ForUML251`. Diese unterstützen den XMI-Export in die entsprechenden XMI-Versionen, siehe auch XMI (XML Metadata Interchange) 663. • Die Schnittstelle `IApplication` hat neue Operationen: `LogMessage`, `LogMessageWithUMLDataLink`. Außerdem gibt es die neue Enumeration `ENUMMessageLogType`. Dadurch können Fehlermeldungen (z.B. aus einem UModel IDE Plug-in) im UModel-Meldungsfenster ausgegeben werden.
UModel API - UMLData	• Es gibt die neue Schnittstelle `IUMLReception` sowie eine Reihe neuer Eigenschaften und Methoden, die UML Receptions unterstützen. Die neue Enumeration `ENUMUMLGuiStyleKind` hat das neue Mitglied `eUMLGuiStyle_ShowReceptions` • Mit Hilfe der neuen Operation `InsertArgumentOfKindAt` können Sie nun **ValuePin** zu **CallBehaviorAction** hinzufügen.

© 2016-2022 Altova GmbH

Automation Interface für UModel 2020 - Typbibliothek Version 5.6

UModel API	• Die Enumeration `ENUMCodeLangVersion` hat neue Einträge für die Sprachversionen C# 8.0 und C++17.

Automation Interface für UModel 2019r3 - Typbibliothek Version 5.5

UModel API	• Die Eigenschaft `ImageFormat` wurde aus der `ISaveAllDiagramsAsImagesDlg`-Schnittstelle entfernt. • Die Enumeration `ENUMCodeLangVersion` hat ein neues Mitglied für die Sprachversion "Java 12".

Automation Interface für UModel 2018r2 - Typbibliothek Version 5.4

UModel API	• Die `IBinaryTypeEntry`-Schnittstelle hat eine neue Eigenschaft: `TypesToImport`. Mit Hilfe dieser Eigenschaft können Sie eine Liste von Binärtypen für den Import definieren (die Binärtypen können durch ein Komma, ein Semikolon oder ein Leerzeichen getrennt werden). • Die `IProjectSettingsDlg`-Schnittstelle hat mehrere neue Eigenschaften für das C++ Code Engineering. • Die Enumeration `ENUMCodeLang` hat ein neues Mitglied `eCodeLang_Cpp`, das die C++-Sprache für das Code Engineering definiert. • Die Enumeration `ENUMCodeLangVersion` hat neue Mitglieder, die C++-Sprachversionen definieren. • Die Enumeration `ENUMUMLPredefinedElement` hat neue Mitglieder für das C++ Code Engineering. Anmerkung: Für das C++ Code Engineering wird die UModel Enterprise benötigt.

Automation Interface für UModel 2017 - Typbibliothek Version 5.3

UModel API - UMLData	Es wurden die folgenden Eigenschaften hinzugefügt: • `IUMLOpaqueAction::Body` • `IUMLOpaqueAction::Language`

Automation Interface für UModel 2016 - Typbibliothek Version 5.2

UModel API - UMLData	Es wurden die folgenden Methoden hinzugefügt: • `IUMLInstanceSpecification::SetSlotInstanceValueAt` • `IUMLSlot::InsertSlotInstanceValueAt` • `IUMLDataAll::InsertSlotInstanceValueAt` • `IUMLDataAll::SetSlotInstanceValueAt`

Automation Interface für UModel 2015r4 - Typbibliothek Version 5.1

UModel API - UMLData	Es wurden die folgenden Methoden hinzugefügt: • `IUMLGuiNodeLink::AddOwnedGuiNodeLink` • `IUMLDataAll::AddOwnedGuiNodeLink`

Automation Interface für UModel 2013 - Typbibliothek Version 5.0

UModel API	• `IDocument` hat eine neue Methode `GenerateSequenceDiagramsForAllOperations`
UModel API - UMLData	• `ENUMExportXMIType` hat einen neuen Eintrag für UML2.4 `(eXMI24ForUML24)`. • `ENUMUMLGuiTextLabelKind` hat ein neues Literal `eTextLabel_DotNetPropertyName` • `ENUMUMLPredefinedElement` hat neue Literale für SysML 1.2. • `IUMLGuiSequenceDiagram` hat für die Codegenerierung die neuen Eigenschaften `UseForForwardEngineering` und `CodeOperation`. • `IUMLExecutionEvent`, `IUMLCreationEvent`, `IUMLDestructionEvent`, `IUMLSendOperationEvent`, `IUMLSendSignalEvent`, `IUMLReceiveOperationEvent` und `IUMLReceiveSignalEvent` wurden entfernt, da die entsprechenden Klasse nicht mehr Teil von UML2.4 sind.

Automation Interface für UModel 2012 - Typbibliothek Version 4.1

UModel API	• Für Modelltransformationen wurden `IModelTransformationDlg`, `IModelTransformationTypeMappings` und `IModelTransformationTypeMapping` hinzugefügt. `IDocument` hat eine neue Methode `ModelTransformation`. • `ILocalOptionsView` hat eine neue Eigenschaft `EnableSnapLines`.
UModel API - UMLData	• `ENUMCodeLang` hat ein neues Literal `eCodeLang_UML`. • `ENUMUMLPredefinedElement` hat eine Reihe neuer Literale für die Modelltransformation.

Automation Interface für UModel 2011r3 - Typbibliothek Version 4.0

UModel API	• `ILocalOptionsDiagramEditing` hat für die Behandlung von .NET-Eigenschaften die neuen Eigenschaften `UseDotNetPropertyCompartment` und `ShowDotNetPropertyCompartment`. • `IDialog` hat die neuen Eigenschaften `Application` und `Parent` • `IImportSourceDlg` hat für die Behandlung von .NET-Eigenschaften die neuen Eigenschaften `Content_UseDotNetPropertyCompartment` und `Content_ShowDotNetPropertyCompartment`.

© 2016-2022 Altova GmbH

UModel API - UMLData	• Die Eigenschaft `BehaviorSpecification` kann für `IUMLBehavior` definiert werden • `ENUMUMLGuiStyleKind` hat ein neues Literal `eUMLGuiStyle_ShowDotNetPropertyCompartment` für die Behandlung von .NET-Eigenschaften

Automation Interface für UModel 2011r2 - Typbibliothek Version 3.2

UModel API	• Für die Zustandsautomaten-Codegenerierung wurde `IGenerateStateMachineCodeDlg` hinzugefügt und `IDocument` hat die Methode `GenerateStateMachineCode`. • `IGenerateDocumentationDlg` hat für die Generierung von SPS-Dokumentation die die neuen Eigenschaften `UseFixedDesign` und `SPSFile`; eine neue Eigenschaft `Include_IncludedPredefinedSubprojects` und eine neue Methode `Fonts_SetDefaults`. • `ENUMDocumentationOutputFormat` hat für die Dokumentationsgenerierung im PDF-Format ein neues Literal • `ENUMUMLPredefinedElement` hat eine Reihe neuer Literale für die BPMN2-Unterstützung
UModel API - UMLData	• Für BPMN2 wurden `IUMLGuiBPMN2Diagram`, `IUMLGuiBPMN2ChoreographyDiagram` und `IUMLGuiBPMN2CollaborationDiagram` hinzugefügt

Automation Interface für UModel 2011 - Typbibliothek Version 3.1

Änderungen seit der letzten Release (UModel 2010r3 - Typbibliothekversion 3.0):

UModel API	• `IImportDatabaseDlg` zum Importieren von Datenbanken ist neu hinzugekommen. • `IDocument` hat die neuen Methoden `ImportDatabase` zum `Importieren` von Datenbanken und `MergeProject3Way` für 3-Weg-Projektzusammenführungen
UModel API - UMLData	• `ENUMCodeLangVersion` und `ENUMUMLPredefinedElement` haben neue Literale zur Datenbankunterstützung • `ENUMUMLDBDataSourceMethod` ist neu hinzugekommen und dient zur Datenbankunterstützung

Automation Interface für UModel 2010r3 - Typbibliothek Version 3.0

Änderungen seit der letzten Release (UModel 2010r2 - Typbibliothekversion 2.1):

UModel API	• `IGenerateSequenceDiagramDlg` hat die Eigenschaft OperationIgnoreList, um beim Generieren eines Sequenzdiagramms anhand von Quellcode bestimmte Eigenschaften zu ignorieren.

	• Die IGenerateDocumentationDlg Schnittstelle hat die neue Eigenschaft Details_Constraints. • ENUMDiagramLayoutKind hat einen neuen Eintrag für die Layoutart “Block” • ENUMCodeLangVersion hat einen neuen Eintrag für C# 4.0
UModel API - UMLData	• IUMLElement hat die Methode GetOwnedElementsOfKind, um alle "owned" Elemente einer bestimmten Art abzurufen. • IUMLClass, IUMLEnumeration und IUMLInterface haben die Methode GetCodeFilePath, um den vollständigen Codedateipfad abzurufen (siehe auch GetCodeFileName, womit nur der Dateiname zurückgegeben wird). • IUMLInterface hat die Eigenschaft Protocol und die Methode SetNewProtocol, um ein IUMLProtocolStateMachine zu referenzieren. • IUMLConstraint hat die neuen Eigenschaften OwningTransition und OwningState. • IUMLState hat die Eigenschaft StateInvariant und die Methode SetNewStateInvariant. • IUMLPort hat die neue Eigenschaft Protocol • IUMLStructuredClassifier hat die Methode InsertOwnedPortAt. • ENUMUMLPredefinedElement hat mehrere neue Literale für C# 4.0 • Des Weiteren gibt es die folgenden neuen Schnittstellen: IUMLValueSpecificationAction, IUMLProtocolStateMachine, IUMLProtocolTransition, IUMLGuiProtocolStateMachineDiagram

Automation Interface für UModel 2010r2 - Typbibliothek Version 2.1

UModel API	• IGenerateSequenceDiagramDlg hat für die neue Aufteilung in Sequenzdiagramme die Eigenschaft SplitIntoSmallerDiagrams.
UModel API - UMLData	• ENUMExportXMIType hat einen neuen Eintrag für UML2.3. • IUMLAction hat die Eigenschaft IsLocallyReentrant. • IUMLPort hat die Eigenschaft IsConjugated. • Die Eigenschaft ConnectorKind von IUMLConnector ist jetzt schreibgeschützt. • IUMLClassifier hat die Eigenschaft IsFinalSpecialization. • IUMLActivityGroup wird nun von IUMLNamedElement abgeleitet.

Automation Interface für UModel 2010 - Typbibliothek Version 2.0

Änderungen seit der letzten Release (UModel 2009 - Typbibliothek Version 1.0):

UModel Plug-ins	• Unterstützung für ActiveX Controls. Jedes IDE Plug-in, das auch ein ActiveX Control ist, wird in einer Dialogfeldsteuerleiste in UModel angezeigt (siehe ActiveX Controls 852 oder das Beispiel StatisticsActiveX 897)
UModel API	• IApplication hat die Eigenschaft ServicePackVersion und die Methode RunMacro, um ein Makro eines (bereits geladenen)

© 2016-2022 Altova GmbH

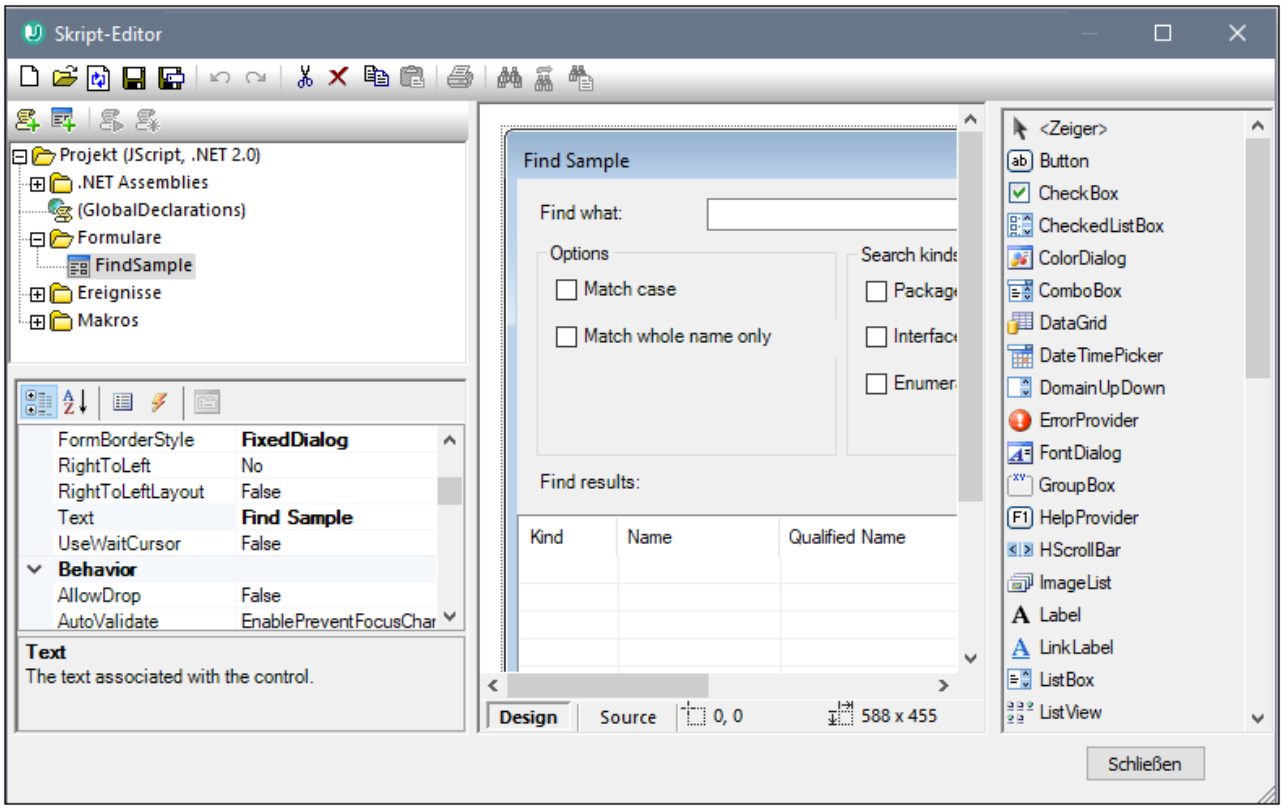
	Skripting-Projekts zu starten. • `IDocument` hat die neuen Methoden: `SaveCopyAs`, `CanFocusUMLDataInModelTree`, `FocusUMLDataInModelTree`, `Reload`. • Mit Hilfe der Eigenschaft `FocusedUMLDataNotifier` von `IDocument` kann die neue Schnittstelle `_IFocusedUMLDataEvents` für Fokusänderungsereignisse aufgerufen werden. • Die Methode `OnModifiedFlagChanged` von `_IDocumentEvents` hat die Schnittstelle `IDocument` als zweiten Parameter. • Die Schnittstelle `IDiagramWindow` hat die folgenden Methoden für automatisches Layout: `Autolayout`, `AutolayoutSelection`. • Die Schnittstelle `IProjectSettingsDlg` hat die neuen Eigenschaften `CSharp_ResolveAliases` und `VBasic_ResolveAliases`. • Die Schnittstelle `IGenerateDocumentationDlg` hat die neuen Eigenschaften `EmbedCSSinHTML` und `CreateFolderForDiagrams`. • `ILocalOptionsCodeEngineering` hat die folgenden neuen Eigenschaften: `CodeFromModel_Indentation_InsertTabs`, `CodeFromModel_Indentation_InsertNSpaces`.
UModel API - UMLData	• Über die neue Schnittstelle `IUMLHyperlink2Model` können Hyperlinks zu Modellelementen (in der Modell-Struktur) angelegt werden. `IUMLNamedElement` hat `InsertOwnedHyperlink2ModelAt`, `IUMLGuiTextHyperlink` hat `SetHyperlinkModelElementAddress` zum Setzen von Links zu Modellelementen. • `IUMLCommentTextHyperlink` ist neu und ermöglicht das Setzen von Hyperlinks für IUMLComments (siehe auch Anleitung zum Erstellen und Verwenden von Hyperlinks). `IUMLComment` wird durch `InsertOwnedCommentTextHyperlinkAt` und `OwnedHyperlinks` erweitert, mit denen Sie diese Hyperlinks einfügen und aufrufen können. • Auf dieselbe Art wurde `IUMLGuiDiagram` durch `InsertOwnedGuiTextHyperlinkAt` und `OwnedHyperlinks` erweitert. • Die Schnittstelle `IUMLElement` hat die Eigenschaften `OwnedDocComment` und `OwnedDocCommentBody`, um den "Body"-Teil des Kommentars, der im Dokumentationsfenster angezeigt wird, wenn das Element markiert ist, aufzurufen. • `IUMLGuiDiagram` hat eine Methode `AddUMLGuiContainmentLink` um Enthältbeziehungslinien in Diagramme einzufügen. • `ENUMUMLGuiTextLabelKind` hat das neuen Literal `eTextLabel_InformationFlow`. • `ENUMUMLPredefinedElement` hat neue Literale für XSD-Datentypen und das SysML-Profil. • Zusätzlich dazu wurden die folgenden neuen Schnittstellen hinzugefügt: `IUMLInformationFlow`, `IUMLGuiContainmentLink`, `IUMLGuiSysMLActivityDiagram`, `IUMLGuiSysMLBlockDefinitionDiagram`, `IUMLGuiSysMLInternalBlockDiagram`, `IUMLGuiSysMLPackageDiagram`, `IUMLGuiSysMLParametricDiagram`, `IUMLGuiSysMLRequirementDiagram`, `IUMLGuiSysMLSequenceDiagram`,

	IUMLGuiSysMLStateMachineDiagram, IUMLGuiSysMLUseCaseDiagram

© 2016-2022 Altova GmbH

17.2 Skript-Editor

Der Skript-Editor ist eine in UModel integrierte Entwicklungsumgebung, über die Sie die Funktionalitäten von UModel mit Hilfe von JScript- oder VBScript-Skripts anpassen können. So können Sie z.B. einen neuen Menüeintrag zur Durchführung einer benutzerdefinierten Projektaufgabe hinzufügen oder Sie können festlegen, dass jedes Mal, wenn ein Dokument in UModel geöffnet oder geschlossen wird, ein bestimmtes Verhalten ausgelöst wird. Zu diesem Zweck werden Skripting-Projekte - Dateien mit der Erweiterung .asprj (Altova Skripting-Projekt) - erstellt.



Skript-Editor

Skripting-Projekte enthalten normalerweise ein oder mehrere Makros, d.h. Programme, die bei Aufruf verschiedene benutzerdefinierte Aufgaben ausführen. Makros können entweder explizit über einen Menübefehl (oder ggf. eine Symbolleisten-Schaltfläche) gestartet werden oder so konfiguriert werden, dass sie beim Start von UModel automatisch ausgeführt werden. Die Skripting-Umgebung ist auch mit der UModel COM API integriert. So können Ihre VBScript- oder JScript-Skripts etwa Applikations- oder Dokument-Ereignisse wie das Starten oder Beenden von UModel, das Öffnen oder Schließen eines Projekts, usw. behandeln. Skripting-Projekte können Windows-Formulare enthalten, die Sie ähnlich wie in Visual Studio visuell entwerfen können. Zusätzlich dazu steht eine Reihe von vordefinierten Befehlen, die Ihnen dabei helfen, .NET-Klassen aus VBScript oder JScript-Code zu instantiieren und zu verwenden zur Verfügung.

Sobald Ihr Scripting-Projekt fertig gestellt wurde, können Sie es in UModel entweder global oder nur für bestimmte Projekte aktivieren.

Für den Skript-Editor muss vor der Installation von UModel .NET Framework 2.0 oder höher installiert sein.

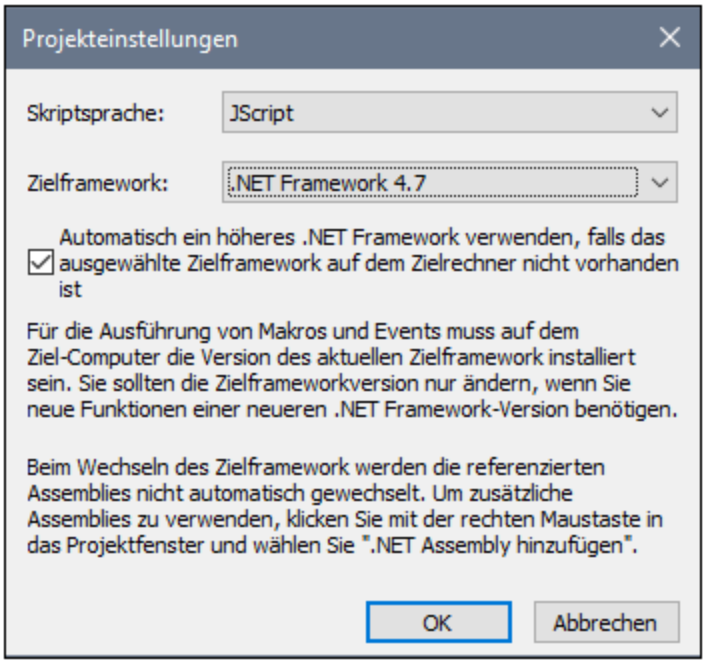
17.2.1 Erstellen eines Skripting-Projekts

Alle im Skript-Editor erstellten Skripting-Informationen werden in Altova Skripting-Projekten (.asprj-Dateien) gespeichert. Ein Skripting-Projekt kann Makros, Applikation Event Handler und Formulare (die wiederum ihre eigenen Event Handler haben können) enthalten. Zusätzlich dazu können Sie globale Variablen und Funktionen zu einem "Global Declarations"-Skript hinzufügen, sodass diese Variablen und Funktionen im gesamten Projekt aufgerufen werden können.

Um ein neues Projekt zu beginnen, klicken Sie auf den Menübefehl **Extras | Skript-Editor**.

Für die Verwendung in einem Skripting-Projekt werden die Sprachen JScript und VBScript unterstützt (nicht zu verwechseln mit Visual Basic, welches nicht unterstützt wird). Diese Skripting-Prozessoren stehen standardmäßig in Windows zur Verfügung. Es müssen keine speziellen Voraussetzungen erfüllt werden, um diese Sprachen auszuführen. Folgendermaßen können Sie eine Skripting-Sprache auswählen:

1. Klicken Sie mit der rechten Maustaste im linken oberen Bereich auf den Eintrag **Projekt** und wählen Sie im Kontextmenü den Befehl **Projekteinstellungen** aus.
2. Wählen Sie eine Sprache (JScript oder VBScript) aus und klicken Sie auf **OK**.



Über das oben gezeigte Dialogfeld "Projekteinstellungen" können Sie auch die .NET-Zielframework-Version ändern. Normalerweise ist dies dann notwendig, wenn für Ihr Skripting-Projekt Funktionen benötigt werden, die in einer neueren .NET Framework-Version zur Verfügung stehen. Beachten Sie, dass dieselbe .NET Framework-Version (oder eine höhere kompatible Version) auch auf allen Clients, auf denen Ihr Skripting-Projekt verwendet wird, installiert sein muss.

© 2016-2022 Altova GmbH

Ein Skripting-Projekt referenziert standardmäßig mehrere .NET Assemblys wie `System`, `System.Data`, `System.Windows.Forms` und andere. Falls nötig, können Sie weitere .NET Assemblys, darunter Assemblys aus .NET Global Assembly Cache (GAC) oder benutzerdefinierte .dll-Dateien importieren. Assemblys können folgendermaßen importiert werden:

1. Statisch durch manuelles Hinzufügen zum Projekt. Klicken Sie mit der rechten Maustaste im linken oberen Bereich auf **Projekt** und wählen Sie im Kontextmenü den Befehl **.NET Assembly hinzufügen**.
2. Dynamisch zur Laufzeit durch Aufruf des Befehls `CLR.LoadAssembly`[827] über den Code.

Gegebenenfalls können Sie mehrere Skripting-Projekte erstellen. Sie können ein Skripting-Projekt auf der Festplatte speichern und es später wieder über die Windows-Standard-Symbolleisten-Schaltflächen **Neu, Öffnen, Speichern, Speichern unter** in den Skript-Editor laden. Nachdem Sie das Skripting-Projekt getestet haben und es bereitgestellt werden kann, können Sie es in UModel laden und jedes beliebige seiner Makros und Event Handler ausführen. Nähere Informationen dazu finden Sie unter Aktivieren von Skripts und Makros[834] .

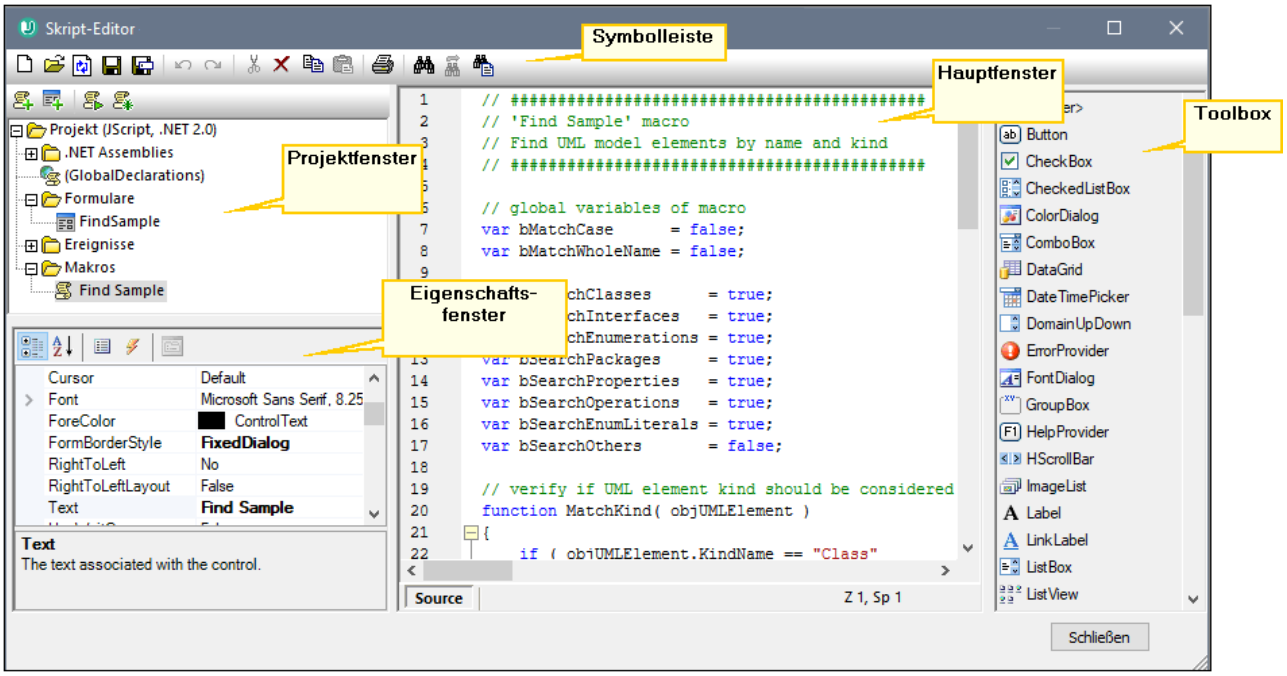
Ein Beispiel für ein Skripting-Projekt finden Sie unter dem folgenden Pfad: **C:\Benutzer\<Benutzer>\Dokumente\Altova\UModel2023\UModelExamples\Scripting\ScriptSampleFind.asprj**.

In den nächsten Abschnitten wird beschrieben, welche Teile in Ihrem Skripting-Projekt eventuell benötigt werden: Globale Deklarationen, Makros, Formulare und Events.

17.2.1.1 Übersicht über die Umgebung

Der Skript-Editor besteht aus den folgenden Teilen:

- Symbolleiste
- Projektfenster
- Eigenschaftsfenster
- Hauptfenster
- Toolbox



Symbolleiste

Die Symbolleiste enthält Windows-Standardbefehle für die Dateiverwaltung (**Neu**, **Öffnen**, **Speichern**, **Speichern unter**) sowie Bearbeitungsbefehle (**Kopieren, Ausschneiden, Löschen, Einfügen**). Bei der Bearbeitung von Quellcode stehen zusätzlich die Befehle **Suchen** und **Ersetzen** sowie **Drucken** zur Verfügung.

Projektfenster

Im Projektfenster können Sie die Struktur des Projekts anzeigen und verwalten. Ein Skripting-Projekt besteht aus mehreren Komponenten, die gemeinsam verwendet und in jeder beliebigen Reihenfolge erstellt werden können.

- Ein Skript *"Global Declarations"*. Wie der Name schon sagt, sind in diesem Skript Informationen gespeichert, die global im gesamten Projekt zur Verfügung stehen. Sie können in diesem Skript alle Variablen oder Funktionen, die in alle Formularen, Event Handlern und Makros zur Verfügung stehen sollen, deklarieren.
- *Formulare.* Formulare dienen normalerweise zum Erfassen von Benutzer-Eingaben oder stellen Informationsdialogfelder zur Verfügung. So kann z.B. mit Hilfe Ihres Skripting-Projekts ein Eingabeformular angezeigt werden, über das der Benutzer einen Elementnamen eingeben und auf eine **Löschen**-Schaltfläche klicken kann. Bei Klick auf diese Schaltfläche werden alle Instanzen dieses Elements aus dem UModel-Projekt entfernt. Ein Formular wird entweder durch einen Aufruf innerhalb einer Funktion (im Globale Deklaration-Skript) oder durch einen Aufruf direkt in einem Makro aufgerufen.
- *Events.* In Ordner "Events" werden UModel Applikations-Events aus der COM API angezeigt. Um ein Skript zu schreiben, das bei Auftreten eines Events ausgeführt werden soll, doppelklicken Sie auf ein beliebiges Event und geben Sie anschließend den behandelnden Code in den Editor ein. Die Applikations-Events sind nicht mit Formular-Events zu verwechseln; letztere werden auf Formularebene behandelt, wie weiter unten beschrieben.

© 2016-2022 Altova GmbH

- *Makros*. Ein Makro ist ein Skript, das entweder bei Bedarf über ein Kontextmenü aufgerufen werden kann oder automatisch beim Start von UModel ausgeführt wird. Makros haben keine Parameter oder Rückgabewerte. Ein Makro kann alle im Global Declarations-Skript deklarierten Variablen und Funktionen aufrufen und auch Formulare anzeigen.

Klicken Sie mit der rechten Maustaste auf eine der Komponenten, um die verfügbaren Kontextmenübefehle und deren Tastenkürzel zu sehen. Doppelklicken Sie auf eine beliebige Datei (wie z.B. ein Formular oder ein Skript), um diese(s) im Hauptfenster zu öffnen.

Über die Symbolleisten-Schaltflächen stehen die folgenden Schnellbefehle zur Verfügung:

	**Neues Makro**	Fügt im Verzeichnis "Makros" ein neues Makro zum Projekt hinzu.
	**Neues Formular**	Fügt im Verzeichnis "Formulare" ein neues Formular zum Projekt hinzu.
	**Makro ausführen**	Führt das ausgewählte Makro aus.
	**Makro debuggen**	Führt das ausgewählte Makro im Debug-Modus aus.

Eigenschaftsfenster

Das Eigenschaftsfenster ist dem Eigenschaftsfenster in Visual Studio sehr ähnlich. Darin werden die folgenden Elemente angezeigt:

- Wenn ein Formular ausgewählt ist, Formulareigenschaften
- Wenn ein Objekt in einem Formular ausgewählt ist, Objekteigenschaften
- Wenn ein Formular ausgewählt ist, Formular-Events
- Wenn ein Objekt in einem Formular ausgewählt ist, Objekt-Events

Um zwischen den Eigenschaften und Events der ausgewählten Komponente hin- und herzuwechseln, klicken Sie auf die Schaltfläche **Properties**  bzw. auf die Schaltfläche **Events**  .

Mit den Schaltflächen **Categorized**  und **Alphabetical**  werden die Eigenschaften oder Events, entweder nach Kategorie oder in aufsteigender alphabetischer Reihenfolge geordnet.

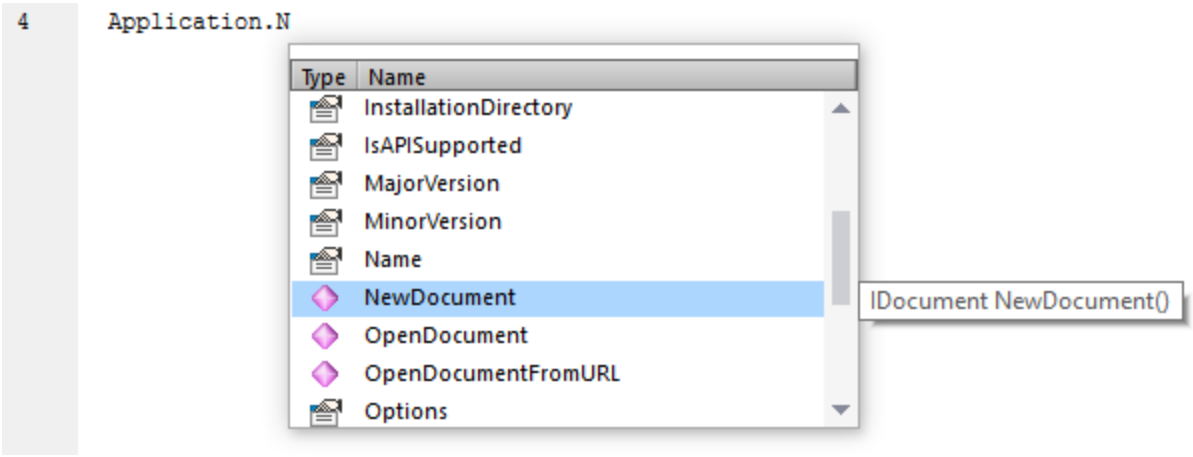
Wird eine Eigenschaft oder ein Event ausgewählt, wird am unteren Rand des Eigenschaftsfensters eine kurze Beschreibung angezeigt.

Hauptfenster

Das Hauptfenster ist der Arbeitsbereich, in den Sie Quellcode eingeben oder das Design des Formulars bearbeiten können. Bei der Bearbeitung von Formularen können Sie auf zwei Registern arbeiten: dem Register **Design** und dem Register **Source**. Auf dem Register **Design** sehen Sie das Layout des Formulars, während das Register **Source** den Quellcode wie z.B. Handler-Methoden für die Formular-Events enthält.

Der Quellcode-Editor bietet Codebearbeitungshillfen wie Syntaxfärbung, Quellcode-Klappleiste, Markierung von öffnenden und schließenden geschwungenen Klammern, Zoomen, Autokomplettierungsvorschläge, Lesezeichen.

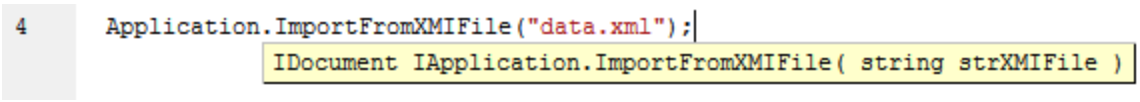
Autokomplettierungsvorschläge
JScript und VBScript sind typenlose Sprachen, daher ist die Autokomplettierung auf COM API-Namen und vordefinierte UModel-Befehle[824] beschränkt. Die vollständige Methode oder Eigenschaftssignatur wird neben der Autokomplettierungseingabehilfe angezeigt.



Wenn Namen mit `objUMLxxx` beginnen, werden Mitglieder der entsprechenden `IUMLxxx` Schnittstelle angezeigt. So hat z.B. die UModel COM API eine Schnittstelle `IUMLClass`. Wenn Sie beim Codieren `objUMLClass`, `objUMLClass123` oder `objUMLClassParent` verwenden, werden die Mitglieder der entsprechenden `IUMLClass` angezeigt.

Wenn Namen mit `objApplication`, `objDocument` oder `objDiagramWindow` beginnen, werden Mitglieder der entsprechenden Schnittstelle angezeigt. Dies gilt auch für alle anderen in der UModel API definierten Schnittstellen.

Wenn Sie den Mauszeiger über eine bekannte Methode oder Eigenschaft platzieren, wird deren Signatur (und, falls vorhanden, Dokumentation) angezeigt, z.B:



Die Autokomplettierungseingabehilfe wird normalerweise automatisch bei der Bearbeitung angezeigt, kann aber auch jederzeit durch Drücken von **Strg+Leerzeichen** angezeigt werden.

Lesezeichen
- Um ein Lesezeichen zu setzen oder zu entfernen, klicken Sie in eine Zeile und drücken Sie **Strg+F2.**
- Um zum nächsten Lesezeichen zu gelangen, drücken Sie **F2.**
- Um zum vorhergehenden Lesezeichen zu gelangen, drücken Sie **Umschalt+F2.**
- Um alle Lesezeichen zu löschen, drücken Sie **Strg+Umschalt+F2.**

Vergrößern/Verkleinern
- Um etwas zu vergrößern oder zu verkleinern, halten Sie die Strg-Taste gedrückt und drücken Sie anschließend die "+" oder "-" Taste oder drehen Sie das Mausrad.

© 2016-2022 Altova GmbH

Einstellungen für Textansicht
Um die Texteinstellungen aufzurufen, klicken Sie mit der rechten Maustaste in den Editor und wählen Sie im Kontextmenü den Befehl **Einstellungen für Textansicht.**

Schriftarteinstellungen
Um die Schriftart zu ändern, klicken Sie mit der rechten Maustaste in den Editor und wählen Sie im Kontextmenü den Befehl **Schriftart Textansicht**.

Toolbox

Die Toolbox enthält alle Objekte, die für das Formulardesign zur Verfügung stehen, wie z.B. Schaltflächen, Textfelder, Auswahllisten, usw.

So fügen Sie ein Element aus der Toolbox zu einem Formular hinzu:

1. Erstellen oder öffnen Sie ein Formular und stellen Sie sicher, dass das Register **Design** ausgewählt ist.
2. Klicken Sie auf das Toolbox-Objekt (z.B.**Button**) und anschließend auf die Stelle im Formular, an der Sie es einfügen möchten. Alternativ dazu können Sie das Objekt auch mit der Maus direkt in das Formular ziehen.

Einige Objekte wie z.B. `Timer` werden nicht zum Formular hinzugefügt, sondern in der Ablageunteren Bereich des Hauptfensters angezeigt. Sie können das Objekt dort auswählen und über das Eigenschaftsfenster dafür Eigenschaften und Event Handler definieren. Ein Beispiel für die Behandlung solcher Komponenten über den Code finden Sie unter Behandeln von Formular-Events 817.

Sie können auch registrierte ActiveX Controls zum Formular hinzufügen. Klicken Sie dazu mit der rechten Maustaste in den Toolbox-Bereich und wählen Sie im Kontextmenü den Befehl **ActiveX Control hinzufügen.**

17.2.1.2 Globale Deklarationen

Das Skript "Global Declarations" ist standardmäßig in jedem Skripting-Projekt vorhanden und muss nicht explizit erstellt werden. Alle Variablen oder Funktionen, die Sie zu diesem Skript hinzufügen, werden als global für das gesamte Projekt behandelt, sodass Sie solche Variablen und Funktionen von jedem beliebigen Makro oder Event des Projekts aus referenzieren können. Im Folgenden sehen Sie ein Beispiel für ein Global Declarations-Skript, das den Namespace `System.Windows.Forms` in das Projekt importiert. Der unten stehende Code ruft zu diesem Zweck den im Skript-Editor vordefinierten Befehl `CLR.Import` auf.

```
// import System.Windows.Forms namespace for all macros, forms and events:
CLR.Import( "System.Windows.Forms" );
```

Anmerkung: Die globalen Deklarationen werden jedes Mal, wenn ein Makro ausgeführt oder ein Event Handler aufgerufen wird, erneut initialisiert.

17.2.1.3 Makros

Makros sind Skripts, die JScript-Anweisungen (oder, je nach Projektsprache, VBScript-Anweisungen) wie z.B. Variablendeklarationen und Funktionen enthalten.

Wenn in Ihren Projekten Makros verwendet werden sollen, können Sie diese folgendermaßen hinzufügen: Klicken Sie mit der rechten Maustaste in das Projektfenster, wählen Sie im Kontextmenü den Befehl Makro hinzufügen aus und geben Sie den Code des Makros in das Hauptformular ein. Beim Makrocode kann es sich z.B. um eine einfache Warnmeldung handeln:

```
alert("Hello, I'm a macro!");
```

Komplexere Makros können Variablen und lokale Funktionen enthalten. Makros können auch Code, der Formulare aus dem Projekt aufruft, enthalten. Im Codefragment unten sehen Sie ein Beispiel für ein Makro, das ein Formular aufruft. Es wird davon ausgegangen, dass dieses Formular im Ordner "Formulare" bereits erstellt wurde und den Namen "SampleForm" hat, siehe auch Formulare 816.

```
// display a form
ShowForm( "SampleForm" );
```

`ShowForm` im obigen Codefragment ist ein im Skript-Editor vordefinierter Befehl. Eine Liste anderer ähnlicher Befehle, die für die Arbeit mit Formularen und .NET-Objekten zur Verfügung stehen, finden Sie unter Vordefinierte Befehle 824.

Sie können mehrere Makros zum selben Projekt hinzufügen und jedes beliebige Makro als "automatisches Makro" definieren. Wenn ein Makro als automatisches Makro definiert ist, wird es beim Start von UModel automatisch ausgeführt. Um ein Makro als automatisches Makro zu definieren, klicken Sie mit der rechten Maustaste darauf und wählen Sie im Kontextmenü den Befehl **Als automatisches Makro definieren**.

Es kann immer nur ein Makro gleichzeitig ausgeführt werden. Nachdem ein Makro (oder Event) ausgeführt wurde, wird das Skript geschlossen und die globalen Variablen verlieren ihre Werte.

Um ein Makro direkt im Skript-Editor auszuführen, klicken Sie auf **Makro ausführen** . Um ein Makro mit dem Visual Studio Debugger zu debuggen, klicken Sie auf **Makro debuggen** . Informationen darüber, wie Sie in UModel Makros aktivieren und ausführen, finden Sie unter Aktivieren von Skripts und Makros 834.

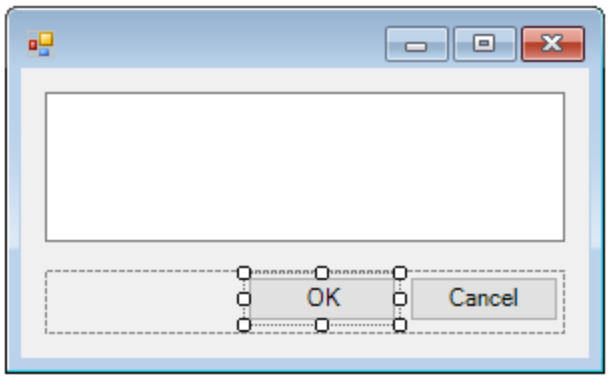
17.2.1.4 Formulare

Formulare eigenen sich vor allem, um Eingabedaten von Benutzern zu erfassen oder Benutzern Daten anzuzeigen. Ein Formular kann diverse Steuerelemente wie z.B. Schaltflächen, Kontrollkästchen, Auswahllisten, usw. enthalten.

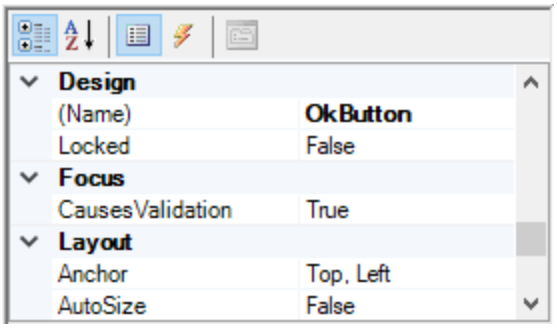
Um ein Formular hinzuzufügen, klicken Sie im Projektfenster mit der rechten Maustaste auf das Formular und wählen Sie im Kontextmenü den Befehl **Formular hinzufügen**. Um ein Steuerelement zu einem Formular hinzuzufügen, ziehen Sie es aus der Toolbox auf der rechten Seite des Skript-Editors in das Formular.

Sie können die Position und Größe der Steuerelemente direkt im Formular über die Ziehpunkte, die angezeigt werden, wenn Sie auf das Steuerelement klicken, ändern, z.B:

© 2016-2022 Altova GmbH



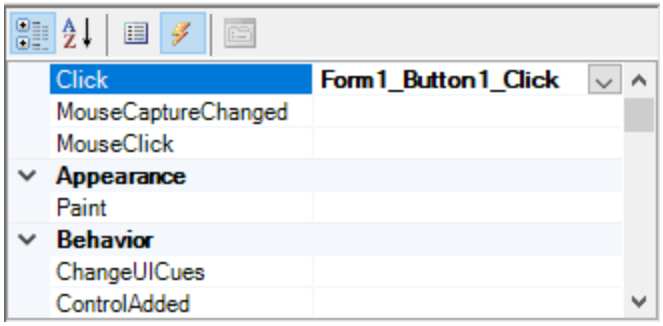
Alle Formularsteuerelemente haben Eigenschaften, die Sie im Eigenschaftsfenster mühelos anpassen können. Wählen Sie dazu das Steuerelement zuerst im Formular aus und bearbeiten Sie anschließend die gewünschten Eigenschaften im Eigenschaftsfenster.



Behandeln von Formular-Events

Jedes Formularsteuerelement stellt außerdem verschiedene Events bereit, an die Ihr Skripting-Projekt gebunden werden kann. So können Sie z.B. jedes Mal, wenn ein Benutzer auf eine Schaltfläche klickt, eine bestimmte UModel COM API-Methode aufrufen lassen. Um eine Funktion zu erstellen, die an ein Formular-Event gebunden ist, gehen Sie folgendermaßen vor:

1. Klicken Sie im Eigenschaftsfenster auf die Schaltfläche **Events** .
2. Doppelklicken Sie in der Spalte **Action** auf das Event, für das Sie die Methode benötigen (in der Abbildung unten ist dies "Click").



Sie können Handler-Methoden auch durch Doppelklick auf ein Steuerelement im Formular hinzufügen. Wenn Sie z.B. im Formulardesign auf eine Schaltfläche doppelklicken, wird eine Handler-Methode für das "Click"-Event dieser Schaltfläche erstellt.

Nachdem der Rumpf der Handler-Methode generiert wurde, können Sie Code eingeben, der dieses Event behandelt, z.B:

```
//Occurs when the component is clicked.
function MyForm_ButtonClick( objSender, e_EventArgs )
{
   alert("A button was clicked");
}
```

Um ein Formular, an dem Sie gerade arbeiten, getrennt vom Skript-Editor anzuzeigen, klicken Sie mit der rechten Maustaste im Projektfenster auf das Formular und wählen Sie im Kontextmenü den Befehl **Formular testen**. Beachten Sie, dass mit dem Befehl **Formular testen** nur das Formular angezeigt wird; die Events des Formulars (wie z.B. Schaltflächenklicks) sind weiterhin deaktiviert. Damit das Formular auf Events reagiert, rufen Sie es von einem Makro aus auf, z.B:

```
// Instantiate and display a form
ShowForm( "SampleForm" );
```

Aufrufen von Formularsteuerelementen

Sie können alle Komponenten eines Formulars von Ihrem Code aus mit Hilfe von Feldaufrufsyntax aufrufen. Angenommen, Sie haben das folgenden Formular:

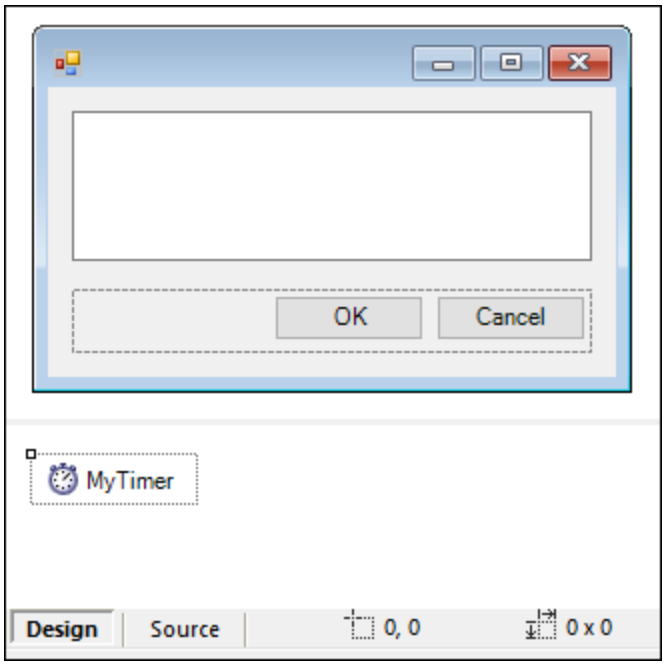
```
// MyForm
//   ButtonPanel
//      OkButton
//      CancelButton
//   TextEditor
//      AxMediaPlayer1
// TrayComponents
//   MyTimer
```

Im unten stehenden Code sehen Sie, wie das Formular instantiiert wird, einige seiner Steuerelemente mittels Feldaufrufsyntax aufgerufen und das Formular anschließend angezeigt wird.

© 2016-2022 Altova GmbH

```
// Instantiate the form
var objForm = CreateForm("MyForm");
// Disable the OK button
objForm.ButtonPanel.OkButton.Enabled = false;
// Change the text of TextEditor
objForm.TextEditor.Text = "Hello";
// Show the form
objForm.ShowDialog();
```

Wenn Sie bestimmte Steuerelemente wie Timer zum Formular hinzufügen, werden diese nicht im Formular, sondern als Komponenten der Ablage am unteren Rand des Formulardesigns angezeigt, z.B:



Mit Hilfe der Methode `GetTrayComponent` können Sie Steuerelemente aus der Ablage aufrufen und den Namen des Steuerelements als Argument bereitstellen. Um in diesem Beispiel eine Referenz zu `MyTimer` zu erhalten und zu aktivieren, verwenden Sie den folgenden Code:

```
var objTimer = objForm.GetTrayComponent("MyTimer");
objTimer.Enabled = true;
```

Bei ActiveX Controls können Sie das zugrunde liegende COM-Objekt über die OCX-Eigenschaft aufrufen:

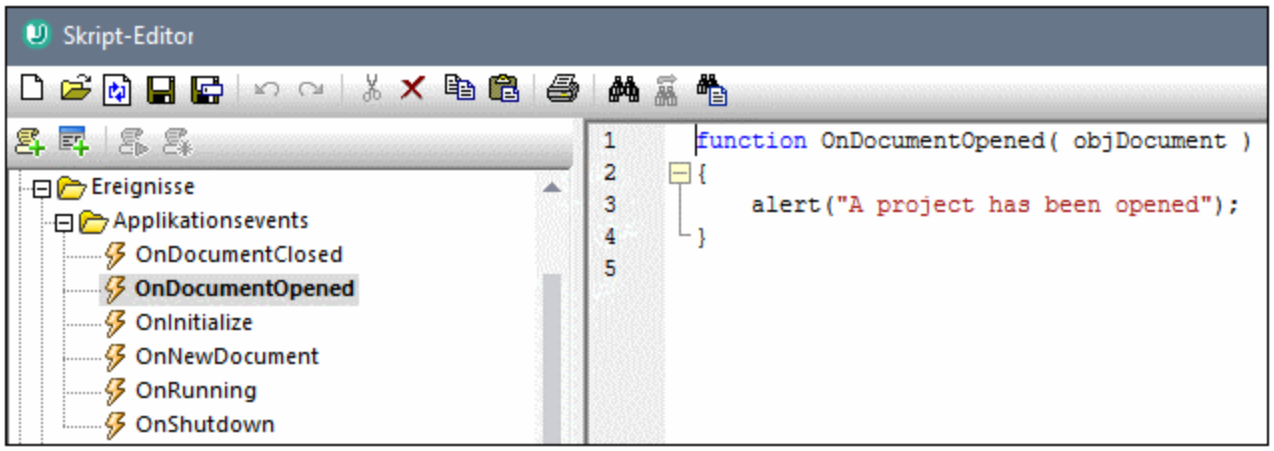
```
var ocx = lastform.AxMediaPlayer1.OCX; // get underlying COM object
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";
```

17.2.1.5 Events

Ihr Skripting-Projekt kann optional Skripts enthalten, die UModel Events wie das Öffnen, Schließen oder Speichern eines Dokuments, das Starten oder Schließen von UModel, das Hinzufügen eines Elements zu einem Diagramm und andere behandeln. Diese Events werden von der UModel COM API bereitgestellt. Sie finden diese Events im Ordner "Ereignisse" Ihres Skripting-Projekts. Beachten Sie, dass diese Events im Gegensatz zu Formular-Events UModel-spezifisch sind. Die Events befinden Sie in den folgenden Ordnern:

- Applikationsevents
- Dokumentevents
- Transaktionsevents
- UMLData Events
- UMLData Fokuswechselevents

Um ein Event Handler-Skript zu erstellen, klicken Sie mit der rechten Maustaste auf ein Event und wählen Sie im Kontextmenü den Befehl **Öffnen** (oder doppelklicken Sie auf das Event). Daraufhin wird das Event Handler-Skript im Hauptfenster, wo Sie es nun bearbeiten können, angezeigt. Der unten gezeigte Event Handler zeigt z.B. jedes Mal, wenn ein Projekt in UModel geöffnet wird, eine entsprechende Benachrichtigung an.



Beachten Sie die folgenden Punkte:

- Der Befehl `alert` ist auf JScript anwendbar. Das VBScript-Äquivalent dazu ist `MsgBox`. Siehe auch alert [825].
- Der Name der Event Handler-Funktion darf nicht geändert werden, da das Event Handler-Skript sonst nicht aufgerufen wird.
- Damit Events verarbeitet werden, muss das Kontrollkästchen **Ereignisse abarbeiten** ausgewählt sein, wenn Sie das Skripting-Projekt in UModel aktivieren. Nähere Informationen dazu finden Sie unter Aktivieren von Skripts und Makros [834].

Sie können in Event Handler-Skripts optional lokale Variablen und Hilfsfunktionen definieren, z.B:

```
var local;

function OnInitialize( objApplication )
{
```


© 2016-2022 Altova GmbH

```
   local = "OnInitialize";
   Helper();
}

function Helper()
{
   alert("I'm a helper function for " + local);
}
```

17.2.1.6 Tipps zur Programmierung mit JScript

Im Folgenden finden Sie einige Tipps zum Programmieren in JScript, die sich bei der Erstellung eines Skripting-Projekts im UModel Skript-Editor als nützlich erweisen könnten.

Out-Parameter

Für Out-Parameter von Methoden des .NET Framework werden in JScript spezielle Variablen benötigt, z.B:

```
var dictionary =
CLR.Create("System.Collections.Generic.Dictionary<System.String,System.String>");
dictionary.Add("1", "A");
dictionary.Add("2", "B");

// use JScript method to access out-parameters
var strOut = new Array(1);
if ( dictionary.TryGetValue("1", strOut) ) // TryGetValue will set the out parameter
  alert( strOut[0] ); // use out parameter
```

Ganzzahl-Argumente

.NET-Methoden, für die Ganzzahl-Argumente benötigt werden, sollten nicht direkt mit JScript-Zahlenobjekten, die Gleitkommawerte enthalten, aufgerufen werden. Verwenden Sie z.B. anstelle von:

```
var objCustomColor = CLR.Static("System.Drawing.Color").FromArgb(128,128,128);
```

Folgendes:

```
var objCustomColor =
CLR.Static("System.Drawing.Color").FromArgb(Math.floor(128),Math.floor(128),Math.floor(12
8));
```

Iterieren über .NET Collections

Für die Iteration über .NET Collections können sowohl der JScript Enumerator als auch .NET-Iterierungstechnologien verwendet werden, z.B:

```
// iterate using the JScript iterator
var itr = new Enumerator( coll );
```

```
for ( ; !itr.atEnd(); itr.moveNext() )
  alert( itr.item() );

// iterate using the .NET iterator
var itrNET = coll.GetEnumerator();
while( itrNET.MoveNext() )
  alert( itrNET.Current );
```

.NET Templates

.NET Templates können, wie unten gezeigt, instantiiert werden:

```
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
```

oder

```
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary<String,Dictionary<String,String>>" );
```

.NET-Enumerationswerte

.NET-Enumerationswerte werden, wie unten gezeigt, aufgerufen:

```
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch;
```

Enumerationsliterale

Die Enumerationsliterale aus der UModel API können, wie unten gezeigt, aufgerufen werden (ihr numerischer Wert muss nicht bekannt sein).

```
objExportXMIFileDlg.XMIType = eXMI21ForUML23;
```

17.2.1.7 Skripting-Beispielprojekt

Unter dem folgenden Pfad steht ein Demo-Projekt, in dem die Skripterstellung mit UModel gezeigt wird, zur Verfügung: **C:\Benutzer\<Benutzer>\Dokumente\Altova\UModel2023\UModelExamples\Scripting\ScriptSampleFind.asprj**.

Dieses Skripting-Projekt besteht aus einem Makro und einem Windows-Formular. Über das Formular können Sie im aktuell geöffneten UModel-Projekt nach UML-Paketen, Schnittstellen, Operationen und anderen Elementarten suchen. Sie können auswählen, nach welchen Elementarten gesucht werden soll und bei der Suche die Groß- und Kleinschreibung ignorieren oder nur nach ganzen Wörtern suchen.

© 2016-2022 Altova GmbH

So laden Sie das Skripting-Projekt in den Skript-Editor:

1. Klicken Sie im Menü **Extras** auf **Skript-Editor**.
2. Klicken Sie auf **Öffnen** und navigieren Sie zur Datei **ScriptSampleFind.asprj** unter dem obigen Pfad.

Beachten Sie, dass das Projekt im Verzeichnis "Makros" ein Makro namens **Find Sample** enthält. außerdem befindet sich im Verzeichnis "Formulare" ein Suchformular, das mehrere Formular-Event Handler enthält.

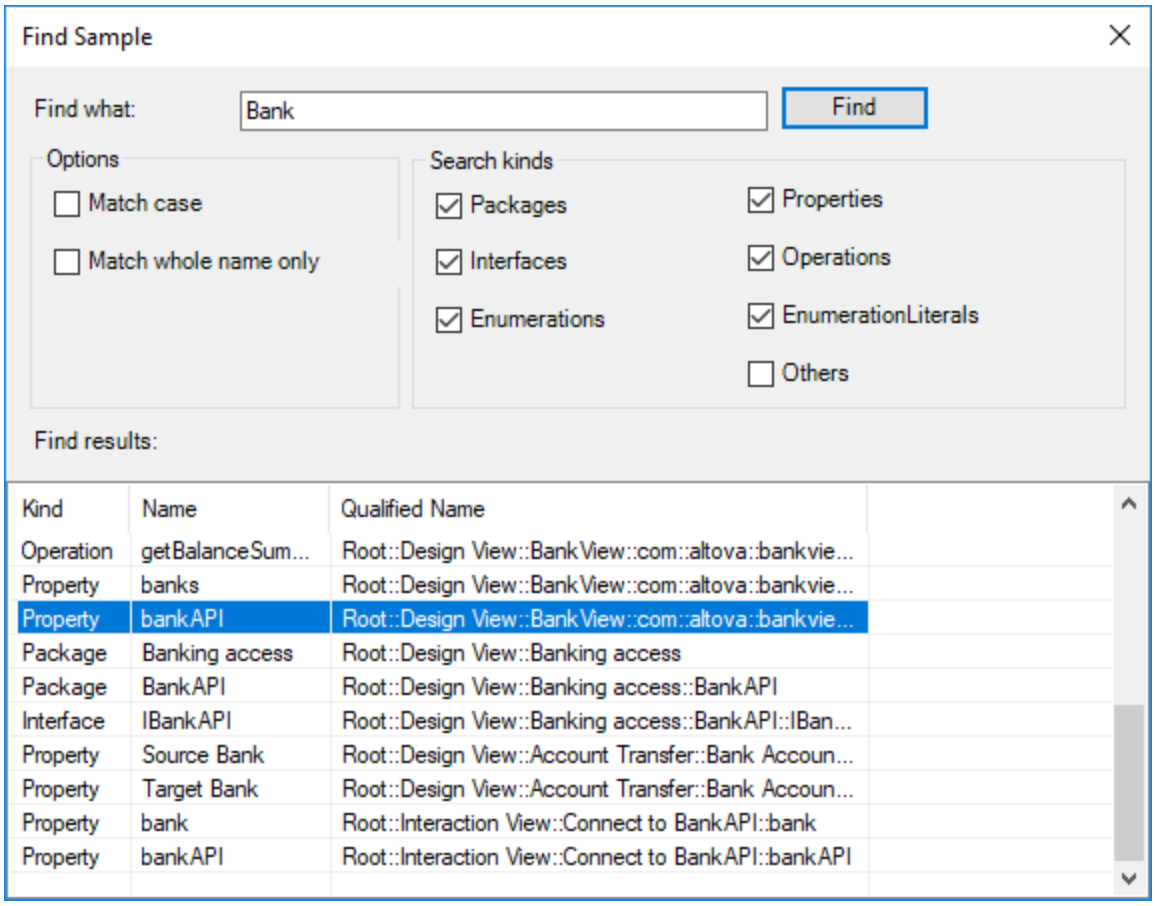
So aktivieren Sie das Skripting-Projekt als globales UModel-Skripting-Projekt:

1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Skripting-Umgebung**.
3. Klicken Sie unter "Globale Skripting-Projektdatei" auf **Durchsuchen** und wählen Sie die Datei **ScriptSampleFind.asprj** aus dem obigen Pfad aus.
4. Dieses Skripting-Projekt hat keine automatischen Makros und Applikations-Event-Handler, daher müssen die Kontrollkästchen **Automatische Makros...ausführen** und **Ereignisse abarbeiten** nicht aktiviert werden.
5. Klicken Sie auf **Anwenden**.

Daraufhin steht unter dem Menü **Extras | Makros** ein neuer Menüeintrag namens **Find Sample** zur Verfügung. Dieser neue Menübefehl ruft das Makro des Skripting-Projekts auf.

So führen Sie das Makro aus:

1. Öffnen Sie ein UModel-Projekt, das mehrere Pakete, Operationen usw. enthält (in diesem Beispiel **C:\Benutzer\<Benutzer>\Dokumente\Altova\UModel2023\UModelExamples\Bank_Java.ump**).
2. Klicken Sie im Menü **Extras** auf **Makros** und anschließend auf **Find Sample**.
3. Geben Sie den Suchbegriff ein und klicken Sie auf **Find**.



Wie oben gezeigt, werden daraufhin alle Projektelemente, deren Name den Suchbegriff enthält, aufgelistet. Sie können auf jedes beliebige Element im Raster klicken, um es im Projektfenster auszuwählen.

17.2.2 Vordefinierte Befehle

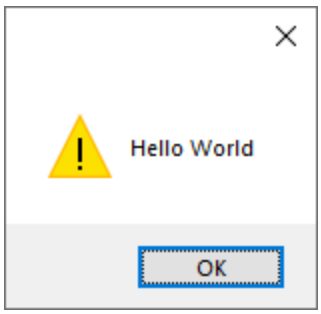
Dieser Abschnitt enthält eine Referenz aller Befehle, die Sie im UModel Skript-Editor verwenden können.

- alert 825
- confirm 825
- CLR.Create 826
- CLR.Import 827
- CLR.LoadAssembly 827
- CLR.ShowImports 828
- CLR.ShowLoadedAssemblies 829
- CLR.Static 830
- CreateForm 830
- doevents 831
- lastform 832
- prompt 832
- ShowForm 833
- watchdog 834

 © 2016-2022 Altova GmbH

17.2.2.1 alert

Zeigt ein Meldungsfeld mit einer angegebenen Meldung und der Schaltfläche "OK" an. Um fortzufahren, muss der Benutzer auf "OK" klicken.



Signatur

Für JScript lautet die Signatur:

```
alert(strMessage : String) -> void
```

Für VBScript lautet die Signatur:

```
MsgBox(strMessage : String) -> void
```

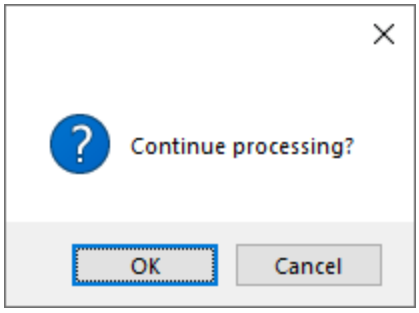
Beispiel

Mit dem folgenden JScript-Code wird ein Meldungsfeld mit dem Text "Hello World" angezeigt.

```
alert("Hello World");
```

17.2.2.2 confirm

Öffnet ein Dialogfeld, in dem eine angegebene Meldung, eine Bestätigungsschaltfläche und eine Abbrechen-Schaltfläche angezeigt werden. Der Benutzer muss entweder auf "OK" oder auf "Cancel" klicken, um fortfahren zu können. Der Rückgabewert ist ein Boolescher Wert, der für die Antwort des Benutzers steht. Wenn der Benutzer auf "OK" klickt, gibt die Funktion **true** zurück, wenn der Benutzer auf "Cancel" klickt, gibt das Dialogfeld **false** zurück.



Signatur

```
confirm(strMessage : String) -> result : Boolean
```

Beispiel (JScript)

```
if ( confirm( "Continue processing?" ) == false )
  alert("You have cancelled this action");
```

Beispiel (VBScript)

```
If ( confirm( "Continue processing?" ) = false ) Then
  MsgBox("You have cancelled this action")
End If
```

17.2.2.3 CLR.Create

Erstellt eine neue .NET-Objektinstanz für den als Argument angegebenen Typnamen. Wenn mehr als ein Argument übergeben wird, werden die nachfolgenden Argumente als Argumente für den Konstruktor des .NET-Objekts interpretiert. Der Rückgabewert ist eine Referenz auf das erstellte .NET-Objekt.

Signatur

```
CLR.Create(strTypeNameCLR : String, constructor arguments ... ) -> object
```

Beispiel

Im folgenden JScript-Code wird gezeigt, wie Sie Instanzen von verschiedenen .NET-Klassen erstellen.

```
// Create an ArrayList
var objArray = CLR.Create("System.Collections.ArrayList");
// Create a ListViewItem
var newItem = CLR.Create( "System.Windows.Forms.ListViewItem", "NewItemText" );
// Create a List<string>
```


© 2016-2022 Altova GmbH

```
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
// Import required namespaces and create a Dictionary object
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String, String > >" );
```

17.2.2.4 CLR.Import

Importiert einen Namespace. Dies ist das Skripting-Äquivalent zu den C# und VB.Net-Schlüsselwörtern `using` und `imports`. Damit können Sie den Namespace-Teil in aufeinander folgenden Aufrufen wie z.B. CLR.`Create()` und `CLR.Static()` auslassen.

Anmerkung: Durch den Import eines Namespace wird die entsprechende Assembly nicht zum Skripting-Projekt hinzugefügt bzw. geladen. Assemblys können mittels CLR.LoadAssembly 827 dynamisch (zur Laufzeit) im Quellcode eingefügt werden.

Signatur

```
CLR.Import(strNamespaceCLR : String) -> void
```

Beispiel

Anstatt vollständig qualifizierte Namespaces wie die folgenden verwenden zu müssen:

```
if ( ShowForm( "FormName" ) == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "System.Windows.Forms.MessageBox" ).Show( "Hello " + sName );
}
```

können Sie Namespaces zuerst importieren und danach die Kurzform verwenden:

```
CLR.Import( "System.Windows.Forms" );

if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.textboxLastName.Text;
    CLR.Static( "MessageBox" ).Show( "Hello " + sName );
}
```

17.2.2.5 CLR.LoadAssembly

Lädt die .NET Assembly mit dem angegebenen Assembly-Namen oder Dateipfad. Gibt den Booleschen Wert **true** zurück, wenn die Assembly geladen werden konnte, **false**, wenn nicht.

Signatur

```
CLR.LoadAssembly(strAssemblyNameCLR : String, showLoadErrors : Boolean) -> result :
Boolean
```

Beispiel

Mit dem folgenden JScript-Code wird versucht, den Text in der Zwischenablage durch dynamisches Laden der erforderlichen Assembly zu definieren.

```
// set clipboard text (if possible)
// System.Windows.Clipboard is part of the PresentationCore assembly, so load this
assembly first:
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35", true ) )
{
    var clipboard = CLR.Static( "System.Windows.Clipboard" );
    if ( clipboard != null )
        clipboard.SetText( "HelloClipboard" );
}
```

17.2.2.6 CLR.ShowImports

Öffnet ein Meldungsfeld, in dem die aktuell importierten Namespaces angezeigt werden. Der Benutzer muss zum Fortfahren auf OK klicken.

Signatur

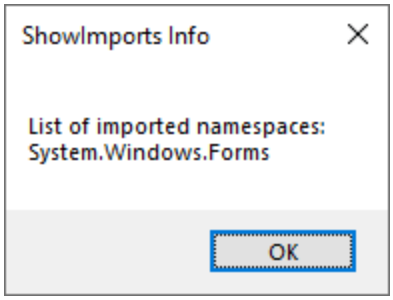
```
CLR.ShowImports() -> void
```

Beispiel

Mit dem folgenden JScript-Code wird zuerst ein Namespace importiert und anschließend die Liste der importierten Namespaces angezeigt:

```
CLR.Import( "System.Windows.Forms");
CLR.ShowImports();
```


© 2016-2022 Altova GmbH



17.2.2.7 CLR.ShowLoadedAssemblies

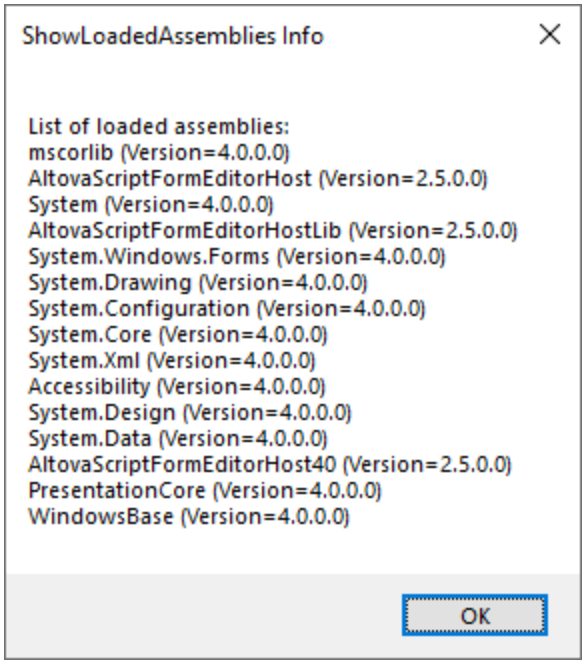
Öffnet ein Meldungsfeld, in dem die aktuell geladenen Assemblys angezeigt werden. Der Benutzer muss zum Fortfahren auf OK klicken.

Signatur

```
CLR.ShowLoadedAssemblies() -> void
```

Beispiel

```
CLR.ShowLoadedAssemblies();
```



17.2.2.8 CLR.Static

Gibt eine Referenz auf ein statisches .NET-Objekt zurück. Verschafft Zugriff auf .NET-Typen, die keine Instanzen haben und nur statische Mitglieder enthalten.

Signatur

```
CLR.Static(strTypeNameCLR : String) -> object
```

Beispiel (JScript)

```
// Get the value of a .NET Enum into a variable
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch

// Set the value of the Windows clipboard
var clipboard = CLR.Static( "System.Windows.Clipboard" );
clipboard.SetText( "HelloClipboard" );

// Check the buttons pressed by the user on a dialog box
if ( ShowForm( "FormName" ) == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

17.2.2.9 CreateForm

Instantiiert das `Form`-Objekt, das durch den als Argument bereitgestellten Namen identifiziert wird. Das Formular muss im Ordner "Formulare" des Skripting-Projekts vorhanden sein. Gibt das Formularobjekt (`System.Windows.Forms.Form`) des angegebenen Namens oder `null` zurück, wenn kein Formular dieses Namens vorhanden ist.

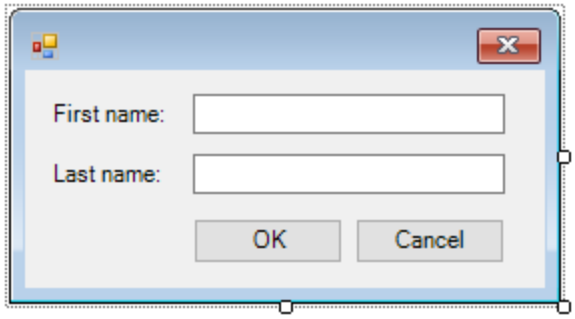
Signatur

```
CreateForm (strFormName : String) -> System.Windows.Forms.Form | null
```

Beispiel

Angenommen, es gibt im Skripting-Projekt ein Formular namens "FormName".

© 2016-2022 Altova GmbH



Der folgende JScript-Code instantiiert das Formular mit einige Standardwerten und zeigt es dem Benutzer an.

```
var myForm = CreateForm( "FormName" );
if ( myForm != null )
{
   myForm.textboxFirstName.Text = "Daniela";
   myForm.textboxLastName.Text = "Heidegger";
   var dialogResult = myForm.ShowDialog();
}
```

Das `dialogResult` kann anschließend folgendermaßen ausgewertet werden:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

Anmerkung: Der oben gezeigte Code funktioniert nur, wenn die Eigenschaft **DialogResult** der Schaltflächen "OK" und "Cancel" über das Eigenschaftsfenster korrekt definiert wurde (für die Schaltfläche "OK" muss er z.B. **OK** sein).

17.2.2.10 doevents

Verarbeitet alle Windows-Meldungen, die sich derzeit in der Meldungswarteschlange befinden.

Signatur

```
doevents() -> void
```

Beispiel (JScript)

```
for ( i=0; i < nLongLastingProcess; ++i )
{
    // do long lasting process
```

```
    doevents(); // process Windows messages; give UI a chance to update
}
```

17.2.2.11 lastform

Dies ist ein globales Feld, das eine Referenz auf das zuletzt mit `CreateForm()` oder `ShowForm()` erstellte Formularobjekt zurückgibt.

Signatur

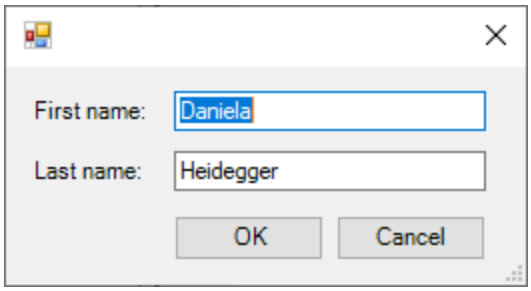
```
lastform -> formObj : System.Windows.Forms.Form
```

Beispiel

Mit dem folgenden JScript-Code wird das Formular "FormName" als Dialogfeld angezeigt.

```
CreateForm( "FormName" );
if ( lastform != null )
{
    lastform.textboxFirstName.Text = "Daniela";
    lastform.textboxLastName.Text = "Heidegger";
    var dialogResult = lastform.ShowDialog();
}
```

Die Werte beider Textfeld-Steuerelemente werden mit Hilfe von `lastform` initialisiert.



17.2.2.12 prompt

Öffnet ein Dialogfeld, in dem einem Meldung sowie ein Textfeld-Steuerelement mit einer Standardantwort angezeigt wird. Damit kann der Benutzer einen einfachen String-Wert eingeben. Der Rückgabewert ist ein String, der den Textfeldwert enthält oder Null, wenn der Benutzer auf "Cancel" geklickt hat.

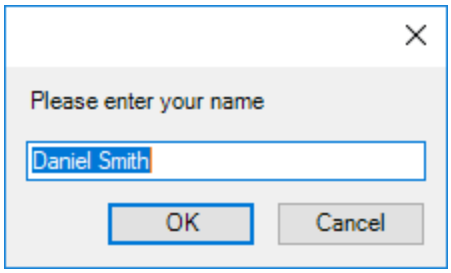
Signatur

```
prompt(strMessage : String, strDefault : String) -> val : String
```


© 2016-2022 Altova GmbH

Beispiel

```
var name = prompt( "Please enter your name", "Daniel Smith" );
if ( name != null )
    alert( "Hello " + name + "!" );
```



17.2.2.13 ShowForm

Instantiiert ein neues Formularobjekt anhand des angegebenen Formularnamens und zeigt es sofort als Dialogfeld an. Der Rückgabewert ist eine Ganzzahl, die für das generierte Dialogfeldergebnis `DialogResult` (`System.Windows.Forms.DialogResult`) steht. Eine Liste der möglichen Werte finden Sie in der Dokumentation zur `DialogResult` Enum (https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.dialogresult?view=netframework-4.8).

Signatur

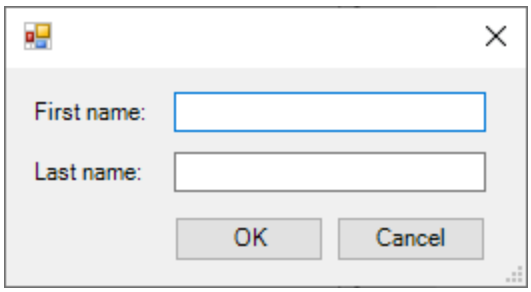
```
ShowForm(strFormName : String) -> result : Integer
```

Beispiel

Der folgende JSCript-Code

```
var dialogResult = ShowForm( "FormName" );
```

zeigt das Formular "FormName" als Dialogfeld an:



Das Ergebnis `DialogResult` kann anschließend ausgewertet werden, z.B:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );
```

Anmerkung: Der oben gezeigte Code funktioniert nur, wenn die Eigenschaft **DialogResult** der Schaltflächen "OK" und "Cancel" über das Eigenschaftsfenster korrekt definiert wurde (für die Schaltfläche "OK" muss er z.B. **OK** sein).

17.2.2.14 watchdog

Bei CPU-intensiven Skripts, deren Verarbeitung lange dauert, fragt der Watchdog den Benutzer, ob das Skript abgebrochen werden soll. Mit Hilfe der `watchdog()` Methode können Sie dieses Verhalten aktivieren bzw. deaktivieren. Standardmäßig ist der Watchdog aktiviert.

Durch Aufruf von `watchdog(true)` können Sie den Watchdog auch zurücksetzen. Dies empfiehlt sich manchmal vor der Ausführung langer speicherintensiver Aufgaben, um sicherzustellen, dass diesen der maximale vom Skript her zulässige Verarbeitungsspeicher zur Verfügung steht.

Signatur

```
watchdog(bEnable : boolean) -> void
```

Beispiel

```
watchdog( false );  // disable watchdog - we know the next statement is CPU intensive but
it will terminate for sure
doCPUIntensiveScript();
watchdog( true );   // re-enable watchdog
```

17.2.3 Aktivieren von Skripts und Makros

Nachdem ein Skripting-Projekt fertig gestellt und getestet wurde, können Sie es auf folgende Arten verwenden:

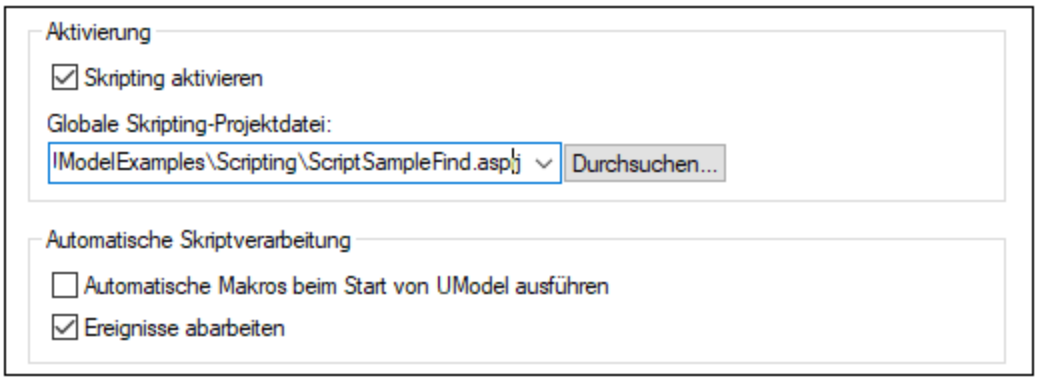
1. Als das globale Skripting-Projekt für UModel. Das bedeutet, dass alle Skripts und Makros aus dem Skripting-Projekt UModel zur Verfügung stehen.
2. Auf UModel-Projektebene. Das bedeutet, es wird eine Referenz auf die .asprj-Datei zusammen mit dem UModel-Projekt gespeichert. Wenn das UModel-Projekt geöffnet wird, können die damit verknüpften Skripts und Makros aufgerufen werden.

So definieren Sie ein Skripting-Projekt als globales Skripting-Projekt:

1. Klicken Sie im Menü **Extras** auf **Optionen**.
2. Klicken Sie auf das Register **Skripting-Umgebung**.

© 2016-2022 Altova GmbH

3. Aktivieren Sie das Kontrollkästchen **Skripting aktivieren** und navigieren Sie zu der .asprj-Datei, die als globales Skripting-Projekt verwendet werden soll.



Sie können optional die folgenden zusätzlichen Skriptverarbeitungsoptionen aktivieren:

Automatische Makros beim Start von UModel ausführen	Wenn Sie dieses Kontrollkästchen aktivieren, werden alle im Projekt als automatische Makros definierten Makros beim Start von UModel automatisch ausgelöst.
Ereignisse abarbeiten	Aktivieren Sie dieses Kontrollkästchen, wenn Ihre Skripts an Applikations-Events gebunden sind. Deaktivieren Sie dieses Kontrollkästchen, damit die Skripts nicht auf Events reagieren.

So aktivieren Sie ein Skripting-Projekt auf Projektebene:

1. Öffnen Sie das Projekt.
2. Klicken Sie im Menü **Projekt** auf **Projekteinstellungen**.
3. Klicken Sie auf das Register **Skripting-Umgebung**.
4. Aktivieren Sie das Kontrollkästchen **Projektskripts aktivieren** und navigieren Sie zur .asprj-Datei.

Das Kontrollkästchen **Beim Laden des... automatische Makros ausführen** hat dieselbe Bedeutung wie oben beschrieben.

17.2.3.1 Ausführen von Makros

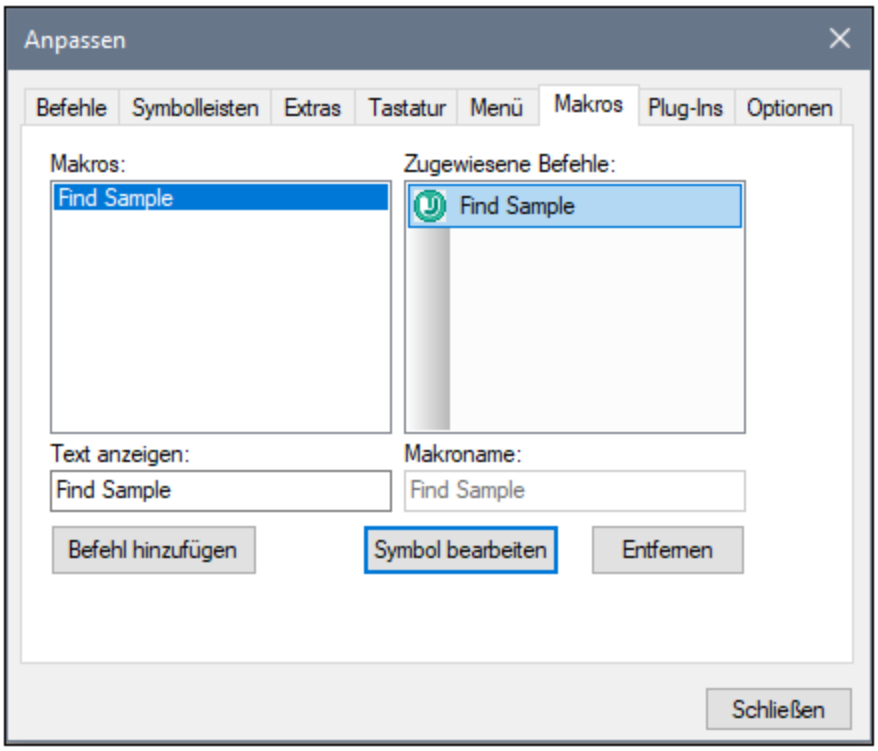
Wenn ein Skripting-Projekt in UModel aktiv ist, werden alle in diesem Projekt verfügbaren Makros im Menü **Extras | Makros** angezeigt. Sie können ein Makro daher durch Ausführen des entsprechenden Menübefehls wie z.B. **Extras | Makros | <Makroname>** jederzeit ausführen.

Makros die als automatische Makros konfiguriert wurden, werden bei jedem Start von UModel automatisch ausgeführt, vorausgesetzt dieses Verhalten wurde in den Optionen, wie unter Aktivieren von Skripts und Makros 834 beschrieben, aktiviert.

Sie können auch Symbolleisten-Schaltflächen für Makros erstellen. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie im Menü **Extras** auf **Anpassen**.

2. Klicken Sie auf das Register **Makros**. Alle auf Applikationsebene (im *globalen* Skripting-Projekt) verfügbaren Makros werden aufgelistet.
3. Klicken sie auf **Befehl hinzufügen**.



4. Klicken Sie optional auf **Symbol bearbeiten** und erstellen Sie ein neues Symbol für das neue Makro. Sie können dem Makro über das Register **Tastatur** auch ein Tastaturkürzel zuweisen.
5. Ziehen Sie das Makro aus dem Bereich **Zugewiesene Befehle** an die gewünschte Stelle in der Symbolleiste.

So entfernen Sie ein Makro aus einer Symbolleiste:

1. Klicken Sie im Menü **Extras** auf **Anpassen**.
2. Klicken Sie auf das Register **Makros**.
3. Ziehen Sie das Makro aus der Symbolleiste, in der es angezeigt wird, zurück in den Bereich **Zugewiesene Befehle.**

© 2016-2022 Altova GmbH

17.3 UModel IDE PlugIns

Eine der Möglichkeiten, programmatisch mit der grafische Benutzeroberfläche von UModel zu interagieren, ist die Erstellung eigener Plug-ins für UModel als DLL-Bibliotheken. UModel Integrated Development Environment (IDE) Plug-ins ermöglichen folgende Dinge:

- Anpassen von UModel (z.B. durch Hinzufügen von Befehlen über benutzerdefinierte Menüs, Schaltflächen)
- Reagieren auf Ereignisse aus UModel
- Ausführen Ihres spezifischen Codes mit UModel, wobei Sie Zugriff auf die komplette UModel API haben
- Integration Ihrer eigenen ActiveX Controls in UModel

Plug-ins können entweder als COM-Applikation (in C++) oder in einer für COM-Interoperabilität geeigneten .NET-Sprache wie C# geschrieben werden. UModel IDE Plug-ins müssen die IUModelPlugIn 853 Schnittstelle implementieren. Außerdem müssen einige weitere Voraussetzungen für die .NET COM-Interoperabilität erfüllt werden, wie später in dieser Dokumentation beschrieben.

Unter dem folgenden Pfad finden Sie einige Visual Studio-Beispielprojekte, anhand derer gezeigt wird, wie Sie UModel-Funktionalitäten über ein benutzerdefiniertes Plug-in aufrufen: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\IDEPlugIn**.

Einschränkungen

Vermeiden Sie bei der Entwicklung eines UModel IDE Plug-in die Definition der Eigenschaft **VisualStyleState** des Objekts **System.Windows.Forms.Application**, z.B:

```
System.Windows.Forms.Application.VisualStyleState = VisualStyleState.NoneEnabled;
```

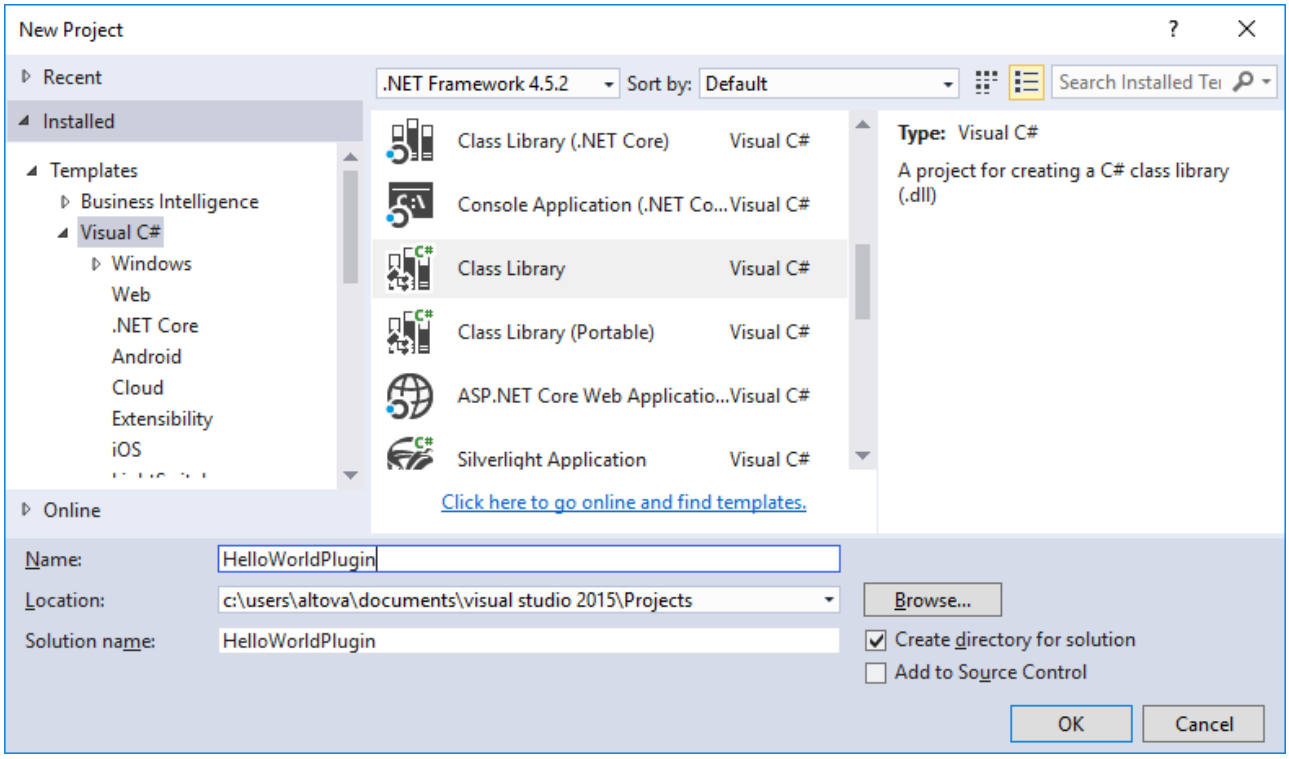
Mit der obigen Einstellung wird verhindert, dass die COM-Klasse erstellt wird, wodurch beim Laden des Plug--in die Menübefehle **Datei | Öffnen** und **Datei | Speichern unter** in UModel nicht geladen werden.

17.3.1 Erstellen eines UModel IDE Plug-in in

In diesem Abschnitt wird beschrieben, wie man in C# und Visual Studio eine einfache UModel IDE Plug-in DLL erstellt.

Anmerkung: Auf Ihrem Computer müssen die UModel Enterprise oder Professional Edition, Visual Studio und das Microsoft .NET Framework installiert sein.

Starten Sie Visual Studio und erstellen Sie ein neues leeres Projekt vom Typ "Klassenbibliothek" (.dll).

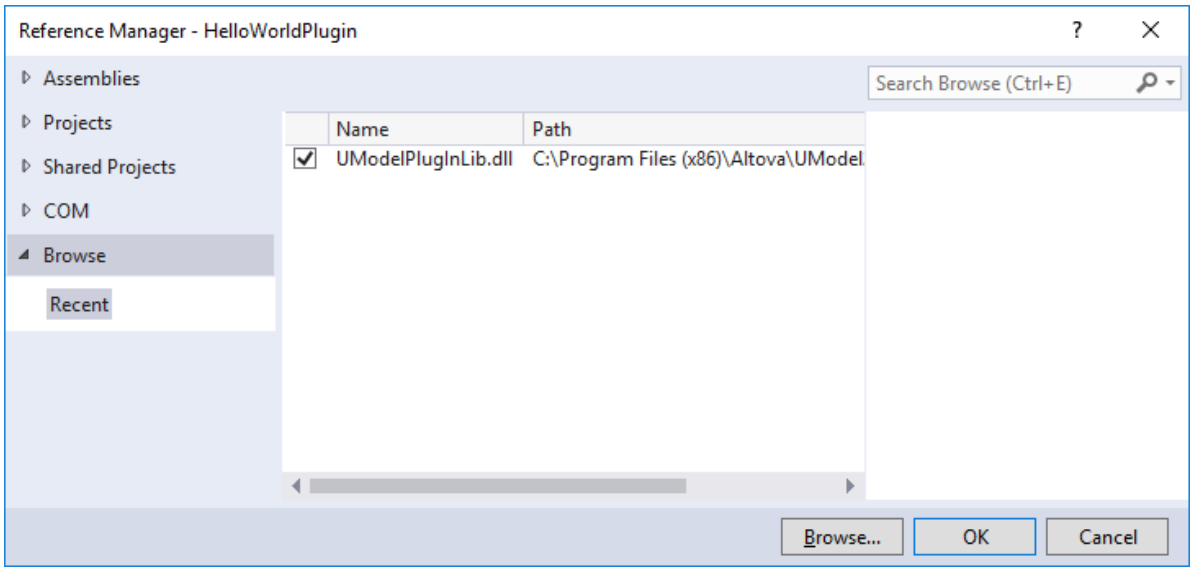


17.3.1.1 Hinzufügen einer Referenz zur UModel Plug-in-Bibliothek

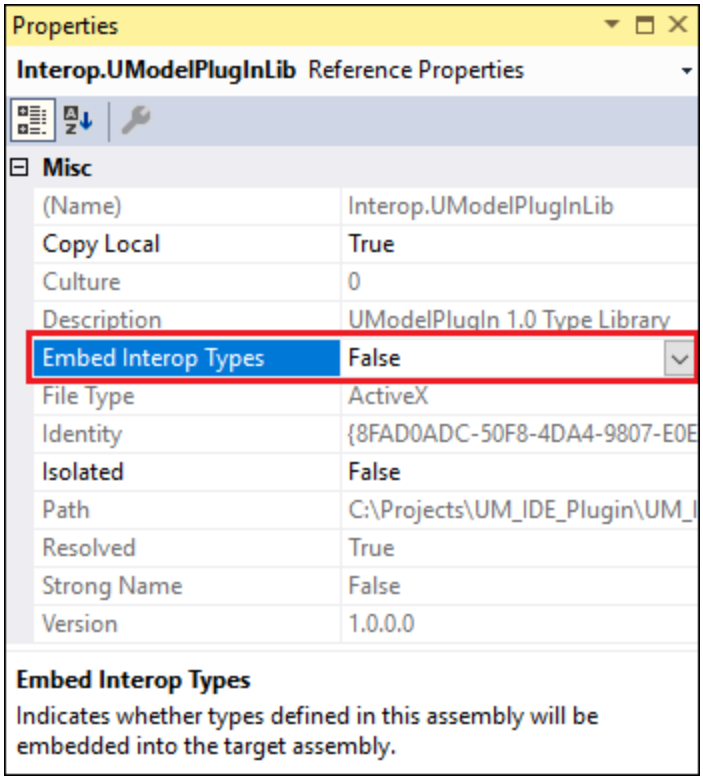
Jede als Plug-in zu UModel hinzugefügte DLL-Bibliothek muss die IUModelPlugIn[853] Schnittstelle implementieren. Zu diesem Zweck muss zuerst in Visual Studio eine Referenz auf die **UModelPlugInLib.dll** hinzugefügt werden. Gehen Sie dazu folgendermaßen vor:

1. Klicken Sie mit der rechten Maustaste im Solution Explorer auf "**Add Reference**".
2. Klicken Sie auf dem Register "**Browse**" auf **Browse** und wählen Sie **UModelPlugInLib.dll** aus dem UModel Installationsverzeichnis (z.B. **C:\Programme (x86)\Altova\UModel2023**) aus.

© 2016-2022 Altova GmbH



3. Klicken Sie im Solution Explorer auf die referenzierte Bibliothek (**UModelPlugInLib**). Suchen Sie im Fenster "Eigenschaften" die Eigenschaft **Embed Interop Types** und stellen Sie sicher, dass diese Eigenschaft auf **False** ist.



UModelPlugInLib.dll ist eine .NET Assembly und wurde mit Hilfe des Microsoft .NET Framework anhand von **IUModelPlugIn.tlb** aus demselben Ordner erstellt.

Wenn Sie Ihr Plug-in in einem . NET Framework mit einer niedrigeren Versionsnummer als 2.0 (z.B. 1.1) installieren möchten, müssen Sie Ihre eigene **UModelPluginLib.dll** in der entsprechenden .NET Framework-Version erstellen.

Sie können Ihre eigene **UModelPluginLib.dll** z.B. mit Hilfe des Type Library Importer Ihrer Wahl erstellen. In .NET können Sie dies mit dem Type Library Importer (**tlbimp.exe**) des Microsoft .NET Framework SDK tun.

```
tlbimp.exe IUmodelPlugIn.tlb
```

Oder Sie erstellen die Assembly mit einem "strong name key pair" und einer bestimmten Version:

```
tlbimp.exe IUmodelPlugIn.tlb /keyfile:UModelPlugIn.snk /asmversion:1.0.0.0
```

wobei `UModelPlugIn.snk` eine mit dem Strong Name Tool (**sn.exe** - ebenfalls Teil des .NET Framework SDK**)** mit einem Befehl wie dem folgenden erstellte Schlüsseldatei ist.

```
sn.exe -k UModelPlugIn.snk
```

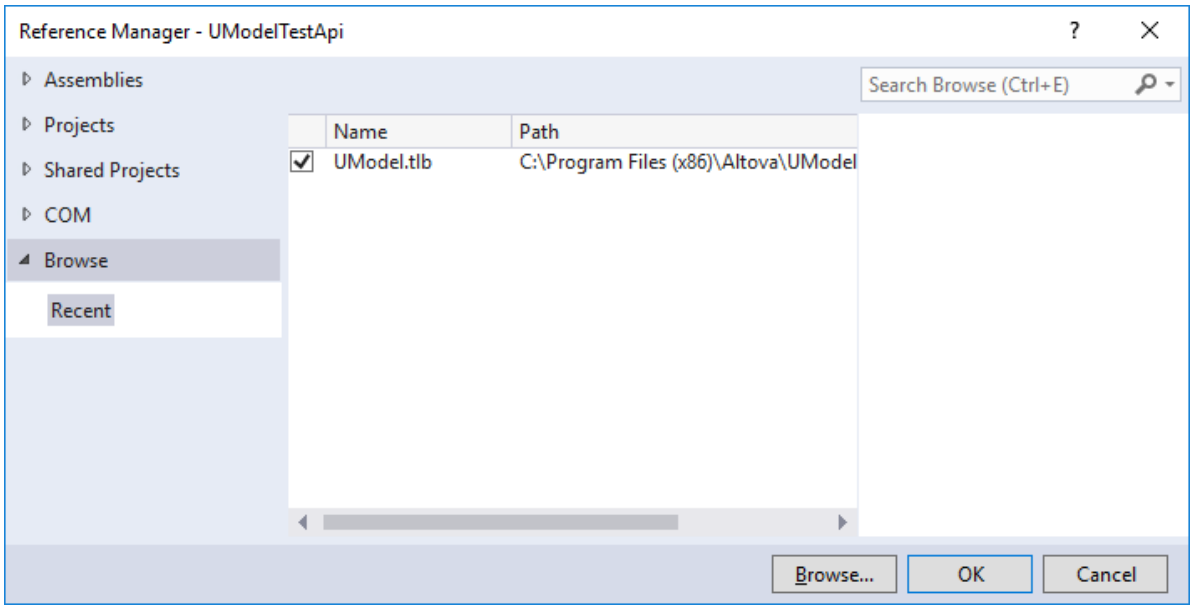
Nähere Informationen zu Tools aus dem .NET Framework finden Sie in der Microsoft-Dokumentation https://docs.microsoft.com/de-de/dotnet/framework/tools/.

17.3.1.2 Hinzufügen einer Referenz zur UModel-Typbibliothek

Um die API-Funktionalitäten von UModel über Ihr Visual Studio-Projekt nutzen zu können, fügen Sie zuerst in Visual Studio eine Referenz auf die UModel-Typbibliothek hinzu:

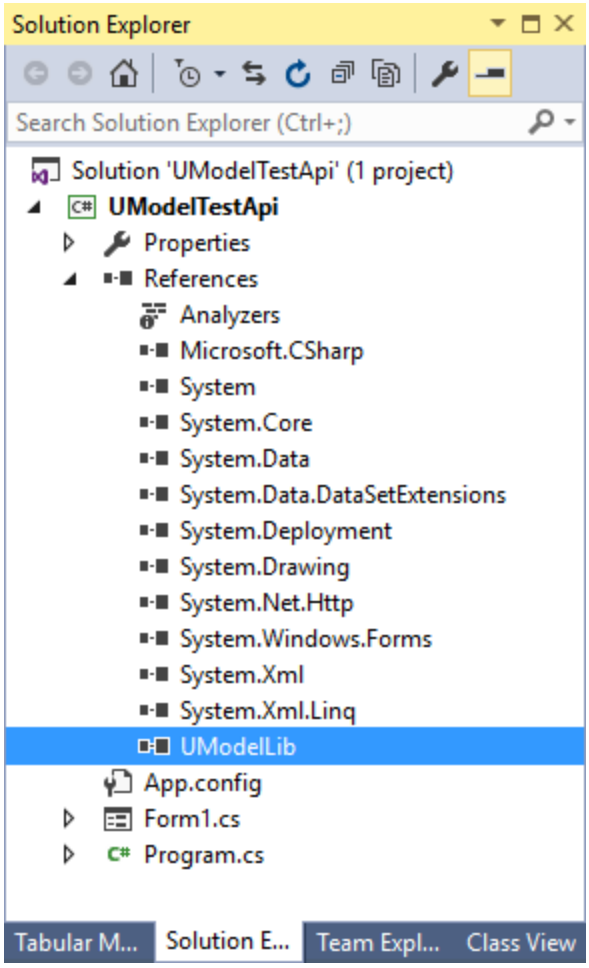
1. Erstellen Sie ein neues Visual Studio-Projekt oder öffnen Sie ein vorhandenes.
2. Klicken Sie im Menü **Project** auf **Add Reference**.
3. Wählen Sie im COM-Abschnitt **UModel Type Library** aus der Liste aus. Wenn dieser Eintrag im COM-Abschnitt nicht zur Verfügung steht, klicken Sie auf **Durchsuchen** und wählen Sie die Datei **UModel.tlb** aus dem UModel-Programmapplikationsordner aus.

© 2016-2022 Altova GmbH



Anmerkung: Die **UModel-Typbibliothek** ist nicht mit der **UModelPlugin-Typbibliothek** zu verwechseln. Mit letzterer können Sie Ihre eigenen Plug-ins erstellen und in UModel integrieren, siehe Hinzufügen einer Referenz zur UModel Plug-in-Bibliothek 838.

Nach Durchführung der oben beschriebenen Schritte sollte die UModel-Typbibliothek in der Liste der Referenzen Ihrer Visual Studio-Projektmappe zur Verfügung stehen, z.B:

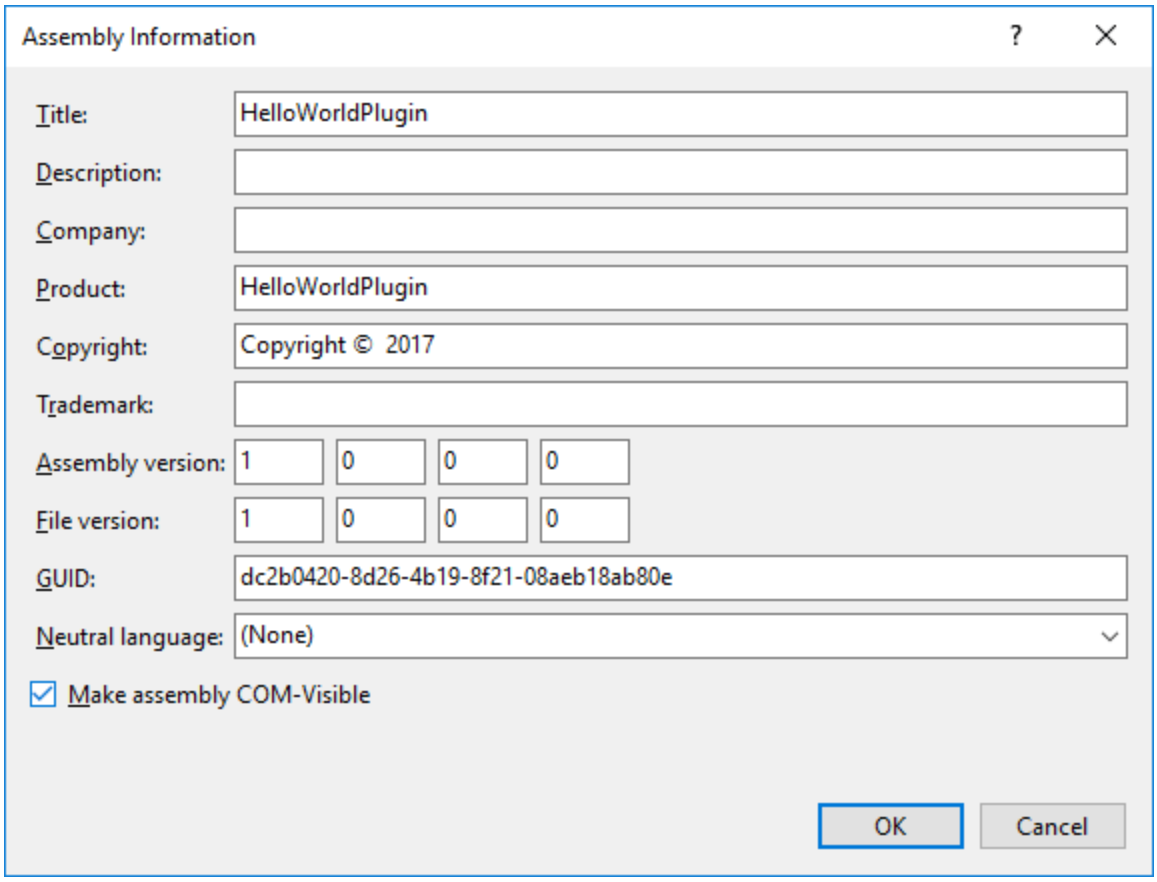


17.3.1.3 Sichtbarmachen der Assenbly für COM

Damit COM auf Ihren Code zugreifen kann, müssen Sie Ihre Compiler-Einstellungen ändern.

1. Klicken Sie mit der rechten Maustaste auf Ihr C#-Projekt und wählen Sie **Properties** aus.
2. Öffnen Sie auf dem Register "Application" das Dialogfeld "**Assembly Information...**" und aktivieren Sie das Kontrollkästchen "**Make assembly COM-Visible**" am unteren Rand des Dialogfelds.

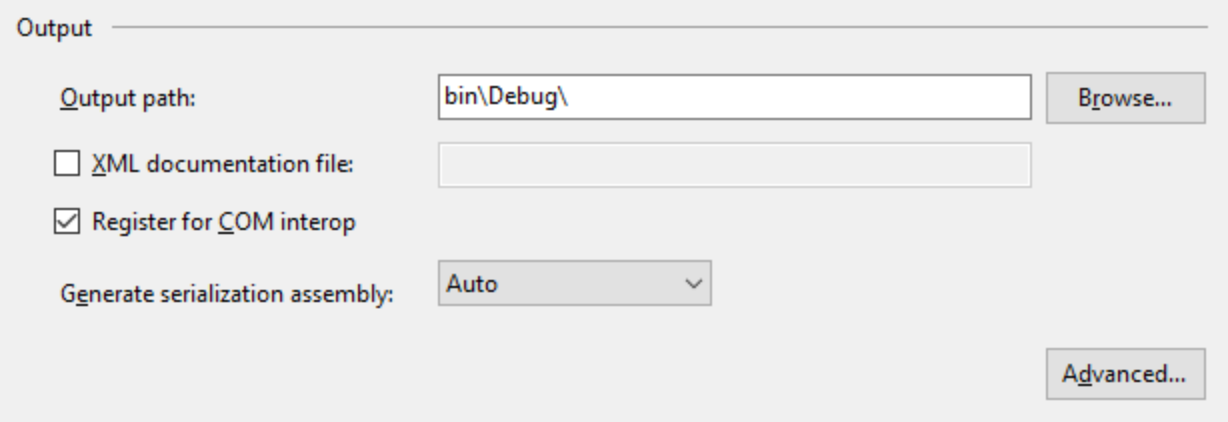
© 2016-2022 Altova GmbH



17.3.1.4 Bereitstellen des COM-Wrappers

So stellen Sie einen durch COM aufrufbaren Wrapper zur Verfügung, der mit COM-Objekten interagieren kann:

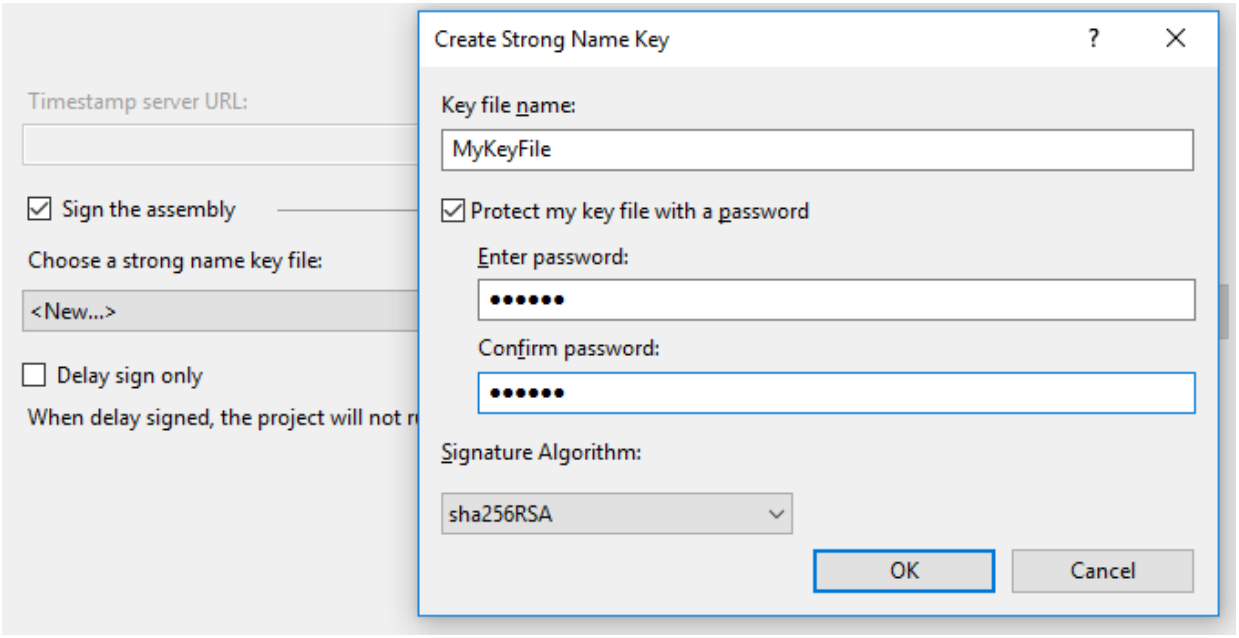
1. Klicken Sie mit der rechten Maustaste auf Ihr C#-Projekt und wählen Sie **Properties** aus.
2. Aktivieren Sie auf dem Register **Build** das Kontrollkästchen **Register for COM interop** für alle Build-Konfigurationen.



17.3.1.5 Signieren des Plug-in mit einem Strong Name (optional)

So signieren Sie Ihre Assembly mit einem Strong Name Schlüsselpaar. (z.B. für das Deployment[847]):

1. Klicken Sie mit der rechten Maustaste auf Ihr C#-Projekt und wählen Sie den Befehl **Properties**.
2. Aktivieren Sie auf dem Register **Signing** das Kontrollkästchen **Sign the assembly.**
3. Wählen Sie entweder über **Browse...** eine bestehende Schlüsseldatei aus oder klicken Sie auf **New...**, um eine neue zu erstellen.



17.3.1.6 Implementieren der IUModelPlugIn-Schnittstelle

UModel IDE Plug-ins müssen die IUModelPlugIn[853] Schnittstelle implementieren. Der nachstehenden Code zeigt eine einfache Implementierung dieser Schnittstelle. Es werden ein Menübefehl und eine (mit UModel verfügbare) Trennlinie zum Menü **Bearbeiten** hinzugefügt. Wenn Sie auf den Menüeintrag klicken, wird ein Meldungsfeld mit dem Text "Hello, World!" angezeigt.

Anmerkung: Da in diesem Beispiel ein Meldungsfeld angezeigt wird, stellen Sie sicher, dass `System.Windows.Forms` in Ihrem C#-Projekt referenziert wird. Klicken Sie dazu mit der rechten Maustaste im Solution Explorer auf **References**, wählen Sie **Add Reference** aus und navigieren Sie zur `System.Windows.Forms`-Assembly.

```
using System;
using System.Collections.Generic;
using System.Text;
using UModelPlugInLib;

namespace HelloWorldPlugIn
{
    public class MyHelloWorldUModelPlugIn : IUModelPlugIn
```


© 2016-2022 Altova GmbH

```
    {
        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "HelloWorldPlugIn;HelloWorldPlugIn demonstrates a simple
implementation of an IDE plug-in for UModel";
        }

        public string GetUIModifications()
        {
            return  "<ConfigurationData>"                          +
                        "<Modifications>"                          +
                            // add "Hello World..." to Edit menu
                            "<Modification>"                       +
                                "<Action>Add</Action>"             +
                                "<UIElement type=\"MenuItem\">"    +
                                    "<ID>1</ID>"                   +
                                    "<Name>Hello world...</Name>"  +
                                    "<Info>My hello world</Info>"  +
                                    "<Place>0</Place>"             +
                                    "<MenuID>101</MenuID>"         +
                                    "<Parent>:Edit</Parent>"       +
                                "</UIElement>"                     +
                            "</Modification>"                      +
                        // add Separator to Edit menu
                            "<Modification>"                       +
                                "<Action>Add</Action>"             +
                                "<UIElement type=\"MenuItem\">"    +
                                    "<ID>0</ID>"                   +
                                    "<Place>1</Place>"             +
                                    "<MenuID>101</MenuID>"         +
                                    "<Parent>:Edit</Parent>"       +
                                "</UIElement>"                     +
                            "</Modification>"                      +
                       // finish modification description
                       "</Modifications>"                          +
                   "</ConfigurationData>";
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
        }

        public void OnShutdown(object pUModel)
        {
            // application will shutdown; release all unused objects
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            if (nID == 1)
                return UModelUpdateAction.UModelUpdateAction_Enable;
```

```
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        public void OnCommand(int nID, object pUModel)
        {
            System.Windows.Forms.MessageBox.Show("Hello world!");
        }

        #endregion
    }
}
```

17.3.1.7 Erstellen und Ausführen des Plug-in

Nach Ausführung der obigen Schritte, erstellen Sie die Lösung mittels Build in Visual Studio (Klicken Sie im Menü **Build** auf **Build Solution**).

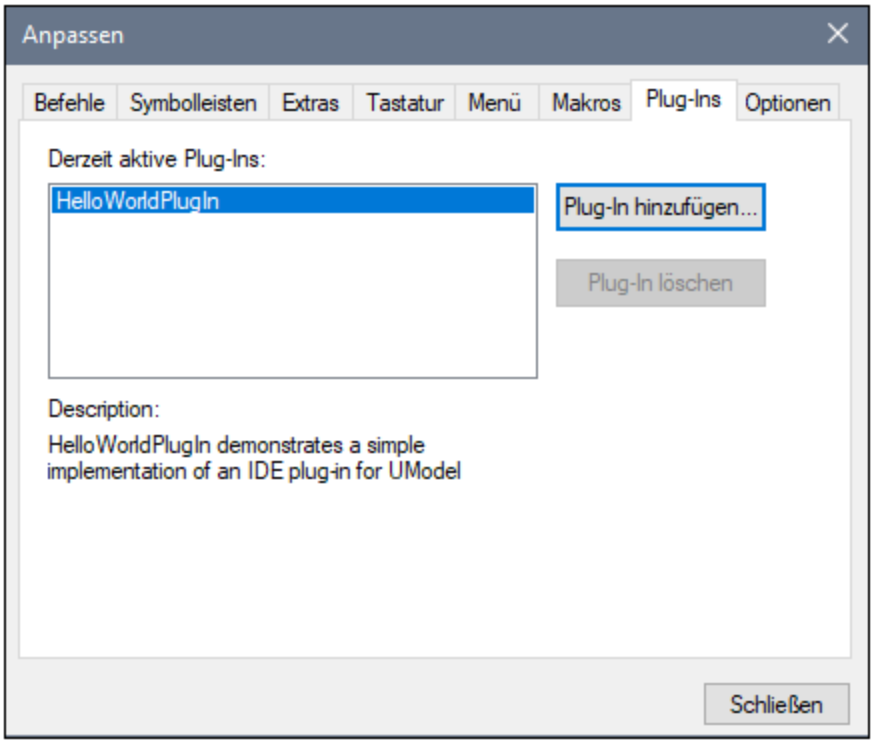
Wichtige Hinweise

- Damit das Plug-in erstellt werden kann, benötigen Sie Zugriff auf die Registry, daher muss Visual Studio von Ihnen als Administrator gestartet werden.
- Wenn Sie ein 64-Bit-Betriebssystem haben und eine 32-Bit-Installation von UModel verwenden, fügen Sie die x86-Plattform im Konfigurationsmanager der Lösung hinzu und erstellen Sie das Beispiel mit dieser Konfiguration. Um den Konfigurationsmanager aufzurufen, klicken Sie auf den Menübefehl **Build | Configuration Manager**
- Klicken Sie im Solution Explorer auf die referenzierte Bibliothek (**UModelPlugInLib**). Suchen Sie im Fenster "Properties" nach der Eigenschaft **Embed Interop Types** und stellen Sie sicher, dass diese Eigenschaft auf **False** gesetzt ist.

Nachdem Sie Ihr C#-Projekt erstellt haben, können Sie das Plug-in zu UModel hinzufügen und es folgendermaßen testen:

1. Starten Sie UModel (oder starten Sie es gegebenenfalls neu; damit stellen Sie sicher, dass die Plug-in-Informationen korrekt aus der Registry ausgelesen werden).
2. Klicken Sie im Menü **Extras** auf **Anpassen**.
3. Klicken Sie auf dem Register **Plug-Ins** auf **Plug-In hinzufügen...** und wählen Sie die Plug-in-DLL-Datei aus (in diesem Beispiel **HelloWorldPlugIn.dll**):

© 2016-2022 Altova GmbH



Anmerkung: Wenn Sie eine Fehlermeldung bekommen, dass in der Typbibliothek keine Implementierung des UModel Plug-In gefunden werden konnte, stellen Sie sicher, dass die Eigenschaft **Embed Interop Types** für die **UModelPlugInLib**-Bibliothek auf **False** gesetzt wurde, wie unter Hinzufügen einer Referenz zur UModel Plug-in-Bibliothek 838 beschrieben.

Das Menü **Bearbeiten** von UModel enthält nun einen neuen Menübefehl namens **Hello world**. Rufen Sie diesen Befehl auf, um ein Dialogfeld mit dem Text "Hello, World!" anzuzeigen.

17.3.2 Bereitstellung von UModel-Plug-ins

Auf einem Entwickler-PC erfolgt die COM-Registrierung, wenn Sie das Plug-in mit Visual Studio erstellen. Unter normalen Umständen ist keine manuelle Registrierung erforderlich. Wenn Sie planen, ein UModel IDE-Plug-in auf einem Client-Zielsystem bereitzustellen, müssen auf diesem Rechner die folgenden Programme installiert sein:

- UModel Professional oder Enterprise Edition
- Wenn das Plug-in in .NET geschrieben wurde, das entsprechende Microsoft .NET Framework.

Auf einem Deployment-PC kann das Plug-in entweder manuell oder bei der Installation registriert werden. Ein Beispiel für ein Visual Studio Setup-Projekt finden Sie im Beispiel "Stile definieren" 882.

So registrieren Sie ein UModel IDE Plug-in manuell:

1. Klicken Sie im Menü **Extras** von UModel auf **Anpassen**.
2. Klicken Sie auf das Register **Plug-Ins**.

3. Klicken Sie auf **Plug-In hinzufügen** und navigieren Sie zur .dll-Datei für das Plug-in.

Durch Ausführen von **regedit.exe** in der Befehlszeile können Sie überprüfen, ob ein UModel Plug-in registriert ist. UModel verwendet den folgenden Registrierschlüssel für alle registrierten Plug-ins:

```
HKEY_CURRENT_USER\Software\Altova\UModel\PlugIns
```

Alle Werte dieses Schlüssels werden als Referenzen auf registrierte Plug-ins behandelt und müssen dem folgenden Format entsprechen:

Wertname:	ProgID des Plug-in
Art des Werts:	muss REG_SZ sein
Daten des Werts:	CLSID der Komponente

Bei jedem Start der Applikation werden die Werte des "PlugIns"-Schlüssels gelesen und die registrierten Plug-ins werden geladen. Wenn es zu Problemen kommt, überprüfen Sie, ob die CLSID Ihres Plug-in korrekt im "PlugIns" Schlüssel registriert wurde. Falls dies nicht der Fall ist, war der Name Ihrer Plug-in-DLL wahrscheinlich nicht eindeutig genug. Verwenden Sie in diesem Fall einen anderen Namen.

Anmerkung: Wenn Sie Ihr UModel IDE Plug-in auf .NET Framework-Versionen vor 2.0 bereitstellen, muss die Plug-in .dll-Datei entweder im selben Verzeichnis wie UModel.exe installiert werden oder mit einem Strong Name Key signiert und im Global Assembly Cache (GAC) registriert werden.

Falls Sie verschiedene Aufgaben im Zusammenhang mit Assemblys manuell durchführen müssen, stehen Ihnen im .NET Framework SDK die folgenden Tools zur Verfügung:

- Assembly Registration Tool (**regasm.exe**). Mit diesem Tool können Sie COM-Assemblys manuell registrieren bzw. deren Registrierung aufheben. Um z.B. die **UModelPlugLib.dll** manuell zu registrieren, verwenden Sie:

  ```
  regasm.exe UModelPlugInLib.dll /codebase
  ```

- Strong Name Tool (**sn.exe**). Dieses Tool kann optional verwendet werden, um Ihren Assembly-Schlüssel mit einem Strong Key zu signieren, z.B:

  ```
  sn.exe -k MyKeyFile.snk
  ```

  Der Schlüssel kann auch mit Visual Studio generiert werden, siehe Signieren des Plug-in mit einem Strong Name (optional) 844.

- Global Assembly Cache Tool (**gacutil.exe**). Mit diesem Tool können Sie eine Assembly zum Global Assembly Cache (GAC) hinzufügen oder daraus entfernen. Um z.B. die **MyPlugin.dll** zum GAC hinzuzufügen, verwenden Sie:

  ```
  gacutil.exe /i MyPlugin.dll
  ```

 © 2016-2022 Altova GmbH

Nähere Informationen zu im .NET Framework inkludierten Tools finden Sie in der Microsoft-Dokumentation https://docs.microsoft.com/de-de/dotnet/framework/tools/.

17.3.3 XML-Konfigurationsdatei

MIt dem Plug-in können Sie die Benutzeroberfläche von UModel ändern. Dabei wird jede einzelne Änderung mit Hilfe eines XML-Datenstroms beschrieben. Die XML-Konfiguration wird mit der `GetUIModifications` Methode der IUModelPlugIn-Schnittstelle 853 an UModel übergeben.

Die XML-Datei für das Plug-in mit den Änderungen an der Benutzeroberfläche muss die folgende Struktur haben:

```
<ConfigurationData>
   <ImageFile>path To image file</ImageFile>
   <Modifications>
      <Modification>
         ...
      </Modification>
      ...
   </Modifications>
</ConfigurationData>
```

Sie können Schaltflächen oder Symbolleisten-Schaltflächen für die neuen Menübefehle definieren, die über das Plug-in zur Benutzeroberfläche von UModel hinzugefügt werden. Der Pfad zur Datei, die die Bilder enthält, wird mit Hilfe des Elements ImageFile definiert. Jedes Bild muss 16 x 16 Pixel groß sein. Die Bildreferenzen müssen in einer einzigen Zeile (<ImageFile>...) von links nach rechts angeordnet werden. Der Indexwert für das am weitesten rechts befindliche Bild ist Null.

Das Modifications Element kann beliebig viele Modification Child-Elemente haben. Jedes Modification Element definiert eine bestimmte Änderung an der Standardbenutzeroberfläche von UModel. Es können auch Elemente der Benutzeroberfläche aus UModel entfernt werden.

Struktur von Modification-Elementen

Alle Modification-Elemente bestehen aus den folgenden beiden Child-Elementen:

```
<Modification>
   <Action>Type of action</Action>
   <UIElement Type="type of UI element">
   </UIElement>
</Modification>
```

Gültige Werte für das Action Element sind:

- Add - um das folgende Benutzeroberflächenelement zu UModel hinzuzufügen.
- Hide - um das folgende Benutzeroberflächenelement in UModel auszublenden.
- Remove - um das Benutzeroberflächenelement im Dialogfeld "Anpassen" aus dem Listenfeld "Befehle" zu entfernen

Sie können Werte des Action Elements kombinieren, z.B. "Hide Remove"

Das UIElement Element beschreibt alle neuen oder bestehenden Benutzeroberflächenelemente für UModel. Mögliche Elemente sind derzeit: neue Symbolleisten, Schaltflächen, Menüs oder Menübefehle. Das **Type** Attribut definiert, welches Benutzeroberflächenelement durch das XML-Element beschrieben wird.

Allgemeine UIElement Child-Elemente

Die Elemente "**ID**" und "**Name**" sind für alle verschiedenen Arten von XML UIElement-Fragmenten gültig. Es besteht jedoch die Möglichkeit, einen der Werte für einen bestimmten Typ von UIElement zu ignorieren z.B. wird "**Name**" bei einem Trennzeichen ignoriert.

```
<ID></ID>
<Name></Name>
```

Wenn UIElement ein bestehendes Element der Benutzeroberfläche beschreibt, ist der Wert des ID-Elements in UModel vordefiniert. Normalerweise sind diese ID-Werte bekannt. Wenn das XML-Fragment einen neuen Teil der Benutzeroberfläche beschreibt, kann die ID beliebig gewählt werden und der Wert sollte kleiner als 1000 sein.

Das Element Name definiert den Textwert. Bestehende Elemente der Benutzeroberfläche, wie z.B. Menüs und Menübefehle mit den dazugehörigen Untermenüs, werden rein anhand ihres Namens erkannt. Bei neuen Elementen der Benutzeroberfläche definiert das Name Element den Titel einer Symbolleiste oder den Text für einen Menübefehl.

Symbolleisten und Menüs

Um eine Symbolleiste zu definieren, muss/müssen die ID und/oder der Name der Symbolleiste angegeben werden. Eine bestehende Symbolleiste kann nur durch Angabe des Namens oder der ID, falls bekannt, definiert werden. Um eine **neue** Symbolleiste zu erstellen, müssen beide Werte definiert werden. Das **Type** Attribut muss "ToolBar" sein.

```
<UIElement Type="ToolBar">
   <ID>1</ID>
   <Name>Styles</Name>
</UIElement>
```

Um ein UModel-Menü zu definieren, benötigen Sie zwei Parameter:

- die ID der Menüleiste, die das Menü enthält. Die ID der Hauptmenüleiste von UModel ist 101.
- den Menünamen. Menüs haben keinen damit verknüpften ID-Wert. Im folgenden Beispiel wird das Menü "Bearbeiten" der Menüleiste definiert:

```
<UIElement Type="Menu">
   <ID>101</ID>
   <Name>Edit</Name>
</UIElement>
```

Wenn Sie ein neues Menü erstellen möchten, wird ein zusätzliches Element verwendet. Das Place Element definiert die Position des neuen Menüs in der Menüleiste:

© 2016-2022 Altova GmbH

```
<UIElement Type="Menu">
   <ID>101</ID>
   <Name>PlugIn Menu</Name>
   <Place>12</Place>
</UIElement>
```

Bei einem Wert von -1 für das Place Element wird die neue Schaltfläche oder der neue Menübefehl an das Ende des Menüs oder der Symbolleiste gesetzt.

Befehle

Wenn Sie einen neuen Befehl über eine Symbolleisten-Schaltfläche oder einen Menübefehl hinzufügen, kann das UIElement Fragment jedes dieser Subelemente enthalten:

```
<Info></Info>
<ImageID></ImageID>
```

Das Info Element enthält eine kurze Beschreibung, die in der Statusleiste angezeigt wird, wenn Sie den Mauszeiger über den dazugehörigen Befehl (Schaltfläche oder Menübefehl) platzieren. ImageID definiert den Index des Symbols in der externen Bilddatei. Beachten Sie bitte, dass alle Symbole in einer Bilddatei gespeichert sind.

Um eine Symbolleisten-Schaltfläche zu definieren, erstellen Sie ein UIElement mit der folgenden Struktur:

```
<UIElement Type="ToolBarItem">
   <!--don't reuse local IDs even the commands do the same-->
   <ID>6</ID>
   <Name>Fill red</Name>
   <!--Set Place To -1 If this is the first button to be inserted-->
   <Place>-1</Place>
   <ImageID>0</ImageID>
   <ToolBarID>1</ToolBarID>
   <!--instead of the toolbar ID the toolbar name could be used-->
   <ToolBarName>Styles</ToolBarName>
</UIElement>
```

Zusätzliche Elemente, die in einer Symbolleisten-Schaltfläche deklariert werden sind: Place, ToolBarID und ToolBarName. ToolBarID und ToolBarName dienen zum Identifizieren der Symbolleiste, die die neue oder bestehende Schaltfläche enthält. Beim Textwert von ToolBarName muss die Groß- und Kleinschreibung beachtet werden. Das **type**-Attribut von UIElement muss "ToolBarItem" sein.

Um einen Menübefehl zu definieren, stehen zusätzlich zu den Standardelementen, die zum Deklarieren eines Befehls verwendet werden, die Elemente MenuID, Place und Parent zur Verfügung. MenuID muss 101 sein. Nähere Informationen zu diesen Werten finden Sie unter "Symbolleisten und Menüs".

Das Parent Element dient zum Festlegen des **Menüs** in das der neue Menübefehl eingefügt werden soll. Da die Befehle von Untermenüs keine eindeutige Windows ID haben, benötigen wir eine andere Methode, um das übergeordnete Element des Menübefehls zu definieren.

Der Wert des Parent Elements ist ein Pfad zum Menübefehl.

Der Textwert des Parent-Elements muss der **Menüname des dem Untermenü übergeordneten Menüs** sein, wobei der Name des Untermenüs durch einen Doppelpunkt getrennt ist. Wenn das Menü kein übergeordnetes Element hat, da es sich beim Menü um kein Untermenü handelt, setzen Sie einen Doppelpunkt vor den Anfang des Namens. Das **type** Attribut muss auf "MenuItem" gesetzt werden.

Beispiele für ein UIElement das einen Menübefehl definiert, sind:

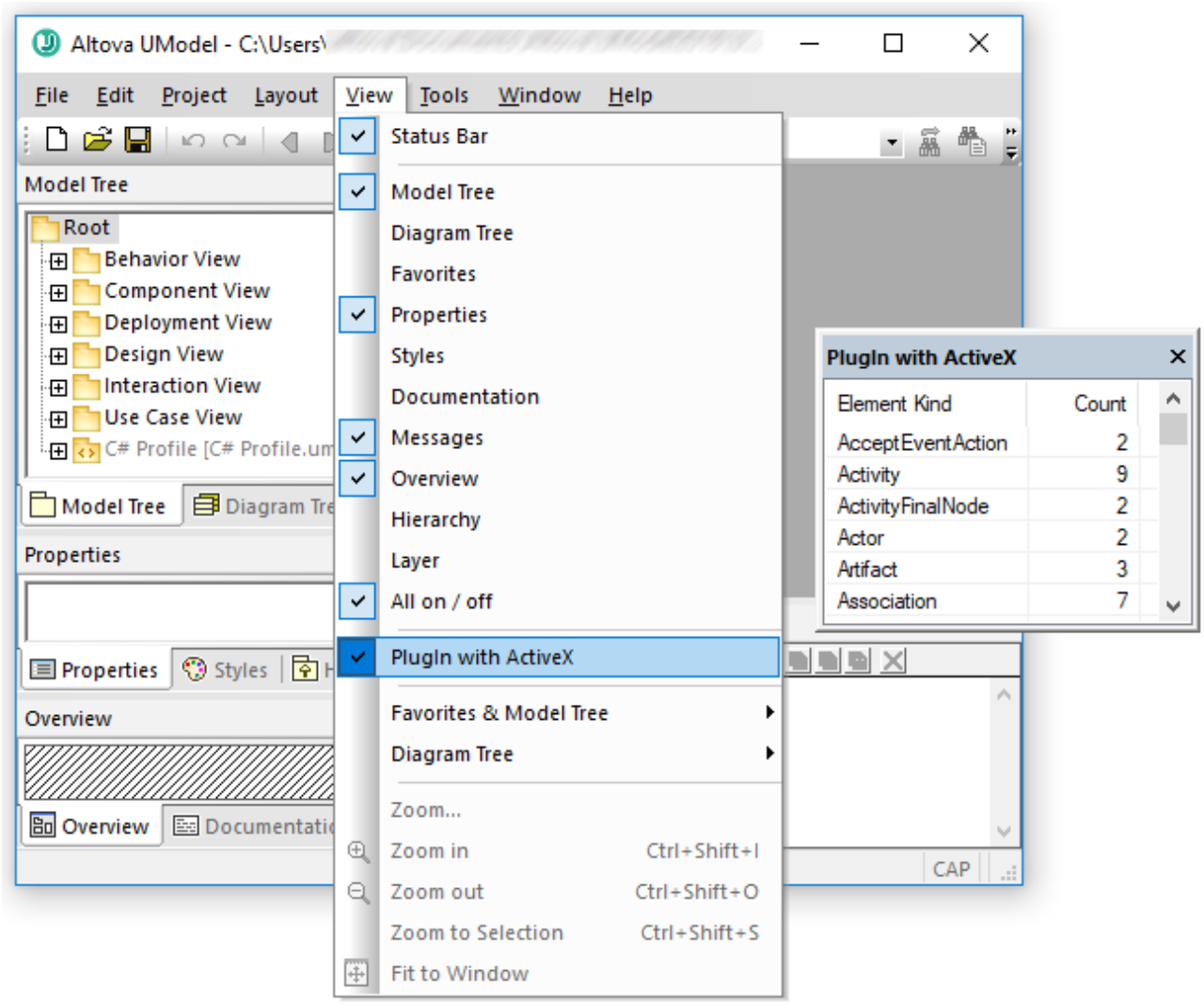
```
<UIElement Type="MenuItem">
   <!--the following element is a Local command ID-->
   <ID>3</ID>
   <Name>Fill red</Name>
   <Place>-1</Place>
   <MenuID>101</MenuID>
   <Parent>:PlugIn Menu1</Parent>
   <ImageID>0</ImageID>
</UIElement>
```

Sie können in UModel Symbolleistentrennzeichen und Menüs hinzufügen, wenn der Wert des ID Elements auf Null gesetzt wird.

17.3.4 Plug-ins als ActiveX Controls

Um als ActiveX Control verwendet werden zu können muss das IDE Plug-in die `IOleControl` Schnittstelle (C++) implementieren oder von `System.Windows.Forms.UserControl` übernehmen (C#, VB.NET). Solche Plug-ins werden auf der grafischen Benutzeroberfläche als neues Fenster angezeigt und erhalten im Menü **Ansicht** auch einen neuen Menübefehl.

© 2016-2022 Altova GmbH



Der Quellcode für das oben gezeigte Plug-in steht unter **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\IDEPlugIn\StatisticsActiveX\StatisticsActiveX.cs**, zur Verfügung, siehe auch das StatisticsActiveX[897] Beispiel.

17.3.5 IUModelPlugIn-Schnittstelle

Wenn eine DLL als Plug-in zu UModel hinzugefügt wird, muss sie als COM-Komponente, die auf eine `IUModelPlugIn`-Schnittstelle antwortet, registriert werden. Die `IUModelPlugin` Schnittstelle stellt die folgenden Methoden bereit, von denen alle von einem Client Plug-in implementiert werden müssen.

- `OnInitialize`
- `OnRunning`
- `OnShutdown`
- `GetUIModifications`
- `GetDescription`
- `OnCommand`

- `OnUpdateCommand`

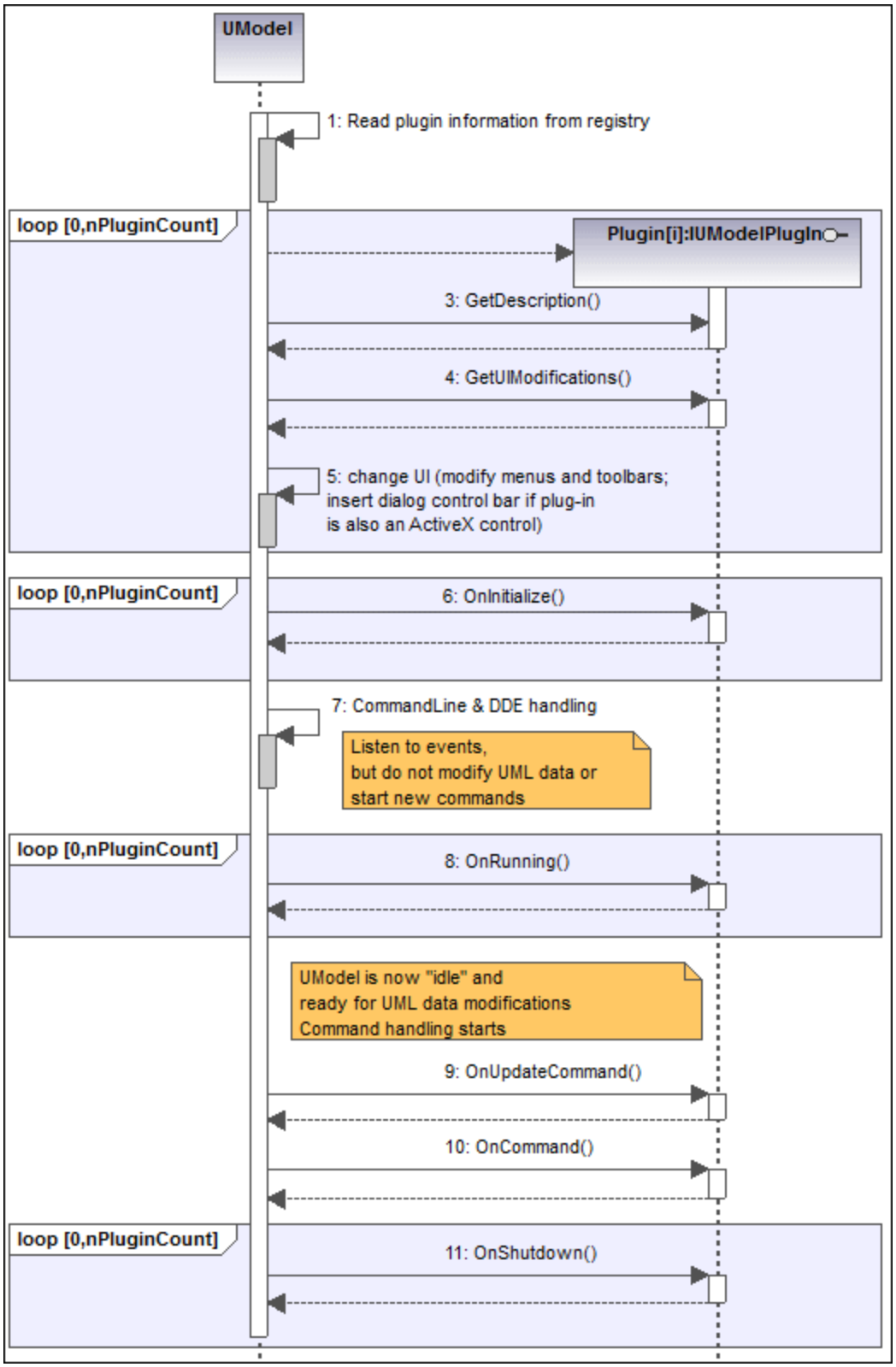
Methodendeklaration	Verwendung
`OnInitialize(pUModel als IDispatch)`	Die `OnInitialize` Methode der Schnittstellenimplementierung wird aufgerufen, wenn das Plug-in initalisiert wird und bevor DDE oder Batch-Befehle verarbeitet werden. Sie können Notifier anhängen und UModel Events mit "listen" beobachten, sollten aber keine neuen Befehle / Änderungen starten, bevor nicht die `OnRunning` Methode aufgerufen wird. `pUModel` enthält eine Referenz zur Sendeschnittstelle des `Application` Objekts von UModel.
`OnRunning(pUModel als IDispatch)`	Die `OnRunning` Methode der Schnittstellenimplementierung wird aufgerufen, wenn das Plug-in initialisiert wird und nachdem DDE oder Batch-Befehle verarbeitet wurden. Die Applikation ist nun vollständig initialisiert und man kann neue Befehle / Änderungen starten und UML-Daten ändern. `pUModel` enthält eine Referenz auf die Sendeschnittstelle des `Application` Objekts von UModel.
`OnShutdown(pUModel als IDispatch)`	Die `OnShutdown` Methode der Schnittstellenimplementierung wird unmittelbar vor dem Entladen des Plug-in (z.B. weil die Applikation beendet wird) aufgerufen. `pUModel` enthält eine Referenz auf die Sendeschnittstelle des `Application` Objekts von UModel.
`GetUIModifications() als String`	Die `GetUIModifications()` Methode wird bei der Initialisierung des Plug-in aufgerufen, um die XML-Konfigurationsdaten aufzurufen, die die Änderungen an der Benutzeroberfläche von UModel definieren. Die Methode wird aufgerufen, wenn das Plug-in das erste Mal geladen wird sowie bei jedem Start von UModel. Eine ausführliche Beschreibung, wie Sie die Benutzeroberfläche ändern können, finden Sie unter XML-Konfigurationsdatei 849.
`GetDescription() als String`	`GetDescription()` dient zum Definieren des Beschreibungsstring für die Plug-in-Einträge, die im Dialogfeld "Anpassen" angezeigt werden.
`OnCommand(nID als long, pUModel als IDispatch)`	Die `OnCommand()` Methode der Schnittstellenimplementierung wird jedes Mal aufgerufen, wenn ein durch das Plug-in hinzugefügter Befehl (Menübefehl oder Symbolleisten-Schaltfläche) verarbeitet wird.

© 2016-2022 Altova GmbH

Methodendeklaration	Verwendung
	`nID` speichert die Befehls-ID, die durch das ID-Element des entsprechenden UIElement definiert wird. `pUModel` enthält eine Referenz auf die Sendeschnittstelle des `Application` Objekts von UModel.
`OnUpdateCommand(nID als long, pUModel als IDispatch) als UModelUpdateAction`	Die `OnUpdateCommand()` Methode wird jedes Mal, wenn der Sichtbarkeitsstatus einer Schaltfläche oder eines Menübefehls definiert werden muss, aufgerufen. `nID` speichert die Befehls-ID, die durch das ID-Element des entsprechenden UIElementdefiniert wird. `pUModel` enthält eine Referenz auf die Sendeschnittstelle des `Application` Objekts von UModel. Mögliche Rückgabewerte zur Definition des Update-Status sind (definiert in UModelUpdateAction)[920]: **UModelUpdateAction_Enable** = 1 **UModelUpdateAction_Disable** = 2 **UModelUpdateAction_Check** = 4 **UModelUpdateAction_Uncheck** = 8 Werte können mit Hilfe des bitwise OR Operators kombiniert werden (z.B. `UModelUpdateAction_Enable | UModelUpdateAction_Check`).

Ein ganz einfaches Beispiel für eine Schnittstellenimplementierung finden Sie unter Implementieren der IUModelPlugIn-Schnittstelle[844]. Weitere Beispielimplementierungen (als Visual Studio-Lösungen) finden Sie unter dem folgenden Pfad: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\IDEPlugIn**.

Im unten gezeigten Sequenzdiagramm sehen Sie, wie UModel mit IUModelPlugIn[919] interagiert:



 © 2016-2022 Altova GmbH

17.4 Die UModel API

Über die COM-basierte API von UModel können Clients die Funktionalitäten von UModel einfach von benutzerdefiniertem Code oder einer Applikation aus aufrufen und verschiedenste Aufgaben automatisieren.

Die UModel-API hält sich an die von Microsoft vorgegebenen allgemeinen Spezifikationen für Automation Server. UModel wird während der Installation automatisch als COM-Server-Objekt registriert. Sobald das COM-Server-Objekt registriert ist, können Sie es von innerhalb einer Applikation aus und über Skripting-Sprachen, die COM-Aufrufe unterstützen, aufrufen. Dadurch kann die UModel API nicht nur von Entwicklungsumgebungen, für die .NET, C++ und Visual Basic verwendet wird, sondern auch von Skripting Sprachen wie JavaScript und VBScript aufgerufen wenden. In Java steht die UModelAPI über Java-COM-Bridge-Bibliotheken zur Verfügung.

Anmerkung: Wenn Sie mit Hilfe der UModel API eine Applikation erstellen, die auch an andere Clients verteilt werden soll, muss UModel auf jedem Client-Rechner installiert sein. Außerdem muss Ihr Integrationscode für jeden Client-Rechner bereitgestellt werden oder Ihre Applikation muss auf jedem Client-Rechner installiert sein.

17.4.1 Aufruf der API

Um die COM API aufrufen zu können, muss in Ihrer Applikation (oder Ihrem Skript) eine neue Instanz des `Application`-Objekts erstellt werden. Anschließend können Sie beginnen, die Funktionalitäten von UModel zu verwenden. Normalerweise werden Sie entweder ein vorhandenes Dokument öffnen, ein neues erstellen oder das aktive Dokument (`IDocument`[938]) aufrufen. `IDocument`[938] entspricht einem UModel-Projekt. Sie können darüber Unterprojekte inkludieren, Dokumentation generieren, Modell und Code miteinander synchronisieren, während Sie darüber Zugriff auf die Haupt-UMLData-Objekte erhalten, siehe auch Objektmodell[858].

Anmerkung: Bei der Implementierung eines UModel IDE Plug-in ist es nicht nötig, eine Instanz das Applikationsobjekts zu erstellen, da UModel bereits ausgeführt wird und die aktuelle Instanz des Applikationsobjekts von `IApplication`[923] als Parameter für alle wichtigen Methoden von `IUModelPlugIn`[853] bereitgestellt wird.

Voraussetzungen

Um das UModel COM-Objekt in Ihrem Visual Studio-Projekt verfügbar zu machen, fügen Sie eine Referenz zur UModel Typbibliotheksdatei (.tlb) hinzu, siehe Referenzieren der UModel-Typbibliothek[877]. Unter **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\API\C#** finden Sie einen UModel API-Beispielclient.

In Java steht die UModel API über Java-COM Bridge-Bibliotheken zur Verfügung. Sie finden diese Bibliotheken im UModel Installationsordner: **C:\Programme (x86)\Altova\UModel2022\JavaAPI** (Beachten Sie, dass dieser Pfad gilt, wenn ein 32-Bit-UModel auf 64-Bit-Windows ausgeführt wird. Passen Sie den Pfad andernfalls entsprechend an).

- `AltovaAutomation.dll`: ein JNI Wrapper für Altova Automation Server
- `AltovaAutomation.jar`: Java-Klassen für den Zugriff auf Altova Automation Server
- `UModelAPI.jar`: Java-Klassen, die den Wrap für die UModel Automation-Schnittstelle bilden
- `UModelAPI_JavaDoc.zip`: eine `Javadoc`-Datei mit der Hilfe zur Java API

Anmerkung: Um direkt Zugriff auf den die Java API zu haben, müssen sich die .dll- und .jar-Dateien im Java `classpath` befinden.

Unter **C:\Benutzer\<Benutzername>\Dokumente\Altova\MapForce2023\UModelExamples\API\Java** steht ein UModel API-Beispiel-Client in Java zur Verfügung.

In Skriptsprachen wie JScript oder VBScript kann das UModel COM-Objekt über den Microsoft Windows Script Host aufgerufen werden (siehe https://msdn.microsoft.com/en-us/library/9bbdkx3k.aspx). Solche Skripts können mit einem Text-Editor geschrieben werden und müssen nicht kompiliert werden, da sie von dem mit Windows verpackten Windows Script Host ausgeführt werden. (Um zu überprüfen, ob der Windows Script Host ausgeführt wird, geben Sie in der Befehlszeile `wscript.exe /?` ein). Unter **C:\Benutzer\<Benutzername>\Dokumente\Altova\MapForce2023\UModelExamples\API\JScript** steht ein UModel API-Beispiel-Client in JScript zur Verfügung.

Anmerkung: Für die 32-Bit-Version von UModel ist der registrierte Name oder der programmatische Identifier (ProgId) des COM-Objekts `UModel.Application`. Für die 64-Bit-Version von UModel ist der Name `UModel_64.Application`. Beachten Sie jedoch, dass das aufrufende Programm die CLASSES Registry-Einträge in seiner eigenen Registry Hive oder -Gruppe (32-Bit oder 64-Bit) aufruft. Wenn Sie daher Skripts über die Standardbefehlszeileneingabe und mit Windows Explorer auf einem 64-Bit-Windows-System ausführen, werden die 64-Bit-Registry-Einträge, welche auf die 64-Bit-Version von UModel verweisen, aufgerufen. Wenn daher sowohl UModel 32-Bit als auch die 64-Bit-Version installiert ist, ist eine spezielle Behandlung erforderlich, damit die 32-Bit-Version von UModel aufgerufen wird. Angenommen, der Windows Skripting Host ist das aufrufende Programm, so gehen Sie folgendermaßen vor:

1. Wechseln Sie in das Verzeichnis **C:\Windows\SysWOW64**.
2. Geben Sie in der Befehlszeile **wscript.exe** gefolgt vom Pfad zum gewünschten Skript ein, z.B:

```
C:\Users\...\Documents\Altova\UModel2023\UModelExamples\API\JScript\Start.js
```

Richtlinien

Es sollten in Ihrem Client Code die folgenden Richtlinien beachtet werden:

- Behalten Sie Referenzen auf Objekte nicht länger im Arbeitsspeicher, als notwendig. Wenn ein Benutzer zwischen zwei Calls Ihres Client eine Eingabe macht, besteht keine Garantie, dass diese Referenzen noch gültig sind.
- Denken Sie daran, dass bei einem Absturz Ihres Client Code Instanzen von UModel möglicherweise noch im System verbleiben. Nähere Informationen, wie man Fehlermeldungen vermeidet, finden Sie unter Behandlung von Fehlern 874.
- Geben Sie Referenzen explizit frei, wenn Sie Sprachen wie C oder C++ verwenden. In C# und Visual Basic können Sie die Speicherbereinigung mit Hilfe von `GC.Collect()` erzwingen.
- UModel API-Collections sind Null-basiert. So wird etwa mit der Anweisung `myPackage.InsertPackagedElementAt(0, "Interface");` eine neue Schnittstelle als erstes Child des Pakets eingefügt.

17.4.2 Objektmodell

Der Ausgangspunkt für jede Applikation, die die UModel API verwendet, ist die IApplication 923 Schnittstelle. Das Application-Objekt besteht aus den folgenden Teilen (jede Einrückung zeigt an, das es sich dabei um eine Child-Parent-Beziehung zur Ebene unmittelbar oberhalb davon handelt):

IApplication 923

 © 2016-2022 Altova GmbH

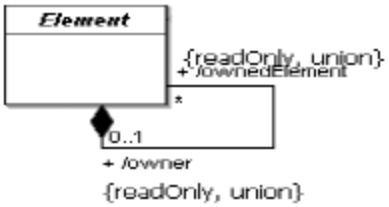
- IDocument 938
  - IDiagramWindows 934
    - IDiagramWindow 931
  - IFocusedUMLDataNotifier 947
  - ITransactionNotifier 996
  - IUMLData 1011 (und alle anderen abgeleiteten UML-Datenschnittstellen)
    - IUMLDataList 1013
- IDialogs 936
  - IExportXMIFileDlg 945
  - IGenerateDocumentationDlg 947
    - IKindSelectionList 972
      - IKindSelection 972
  - IGenerateSequenceDiagramDlg 954
  - IGenerateStateMachineCodeDlg 956
  - IImportBinaryTypesDlg 958
    - IImportBinaryTypeEntries 926
      - IImportBinaryTypeEntry 927
  - IImportDatabaseDlg 962
  - IImportSourceDirectoryDlg 963
  - IImportSourceProjectDlg 967
  - IImportXMLSchemaDirectoryDlg 969
  - IImportXMLSchemaFileDlg 970
  - IIncludeSubprojectDlg 971
    - IModelTransformationDlg 988
      - IModelTransformationTypeMappings 991
        - IModelTransformationTypeMapping 990
  - IProjectSettingsDlg 992
  - ISaveAllDiagramsAsImagesDlg 995
  - ISynchronizationSettingsDlg 995
    - IMatchRenamedDlg 985
      - IMatchRenamedEntries 986
        - IMatchRenamedEntry 987
  - IURLDlg 997
- ILocalOptions 973
  - ILocalOptionsCodeEngineering 975
  - ILocalOptionsDiagramEditing 977
    - ICollectionTemplates 929
      - ICollectionTemplate 928
  - ILocalOptionsEditing 980
  - ILocalOptionsFile 981
  - ILocalOptionsView 983

Enumerations 1003
Events 998

Zusätzlich dazu bildet eine Reihe von Enumerationen 1003 und Events 998 Teil des Modells.

17.4.2.1 Objektmodell UMLData

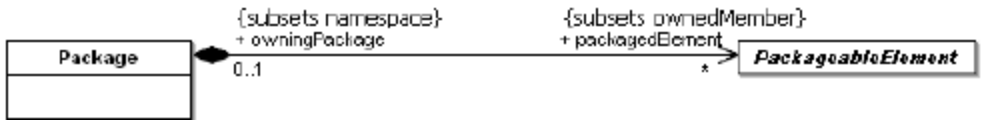
Der Ausgangspunkt zum Aufrufen vom UML-Elementen ist das Root-Paket (IUMLPackage[1238]), welches eine Eigenschaft der IDocument[938] Schnittstelle ist. Alle Child-Elemente des Root-Pakets sind Subtypen von IUMLElement[1090] und werden gespeichert, wie in der UML 2.4 Superstructure Specification der OMG definiert (siehe auch http://www.uml.org). So ist z.B. in der UML Superstructure Specification die folgende Beziehung für UML-Elemente definiert:



D.h. jedes UML-Element kann eine Liste von Elemente dazugehöriger Elemente (="owned elements") haben und jedes UML-Element mit Ausnahme des Root-Pakets hat einen "Owner".

In der UModel API wird ein UML-Element UMLElement[1157] zugeordnet und hat die Eigenschaften `OwnedElement` und `Owner`. Da diese Beziehungen in der UML-Spezifikation schreibgeschützt sind, können beide Eigenschaften in der UModel API nicht geändert werden.

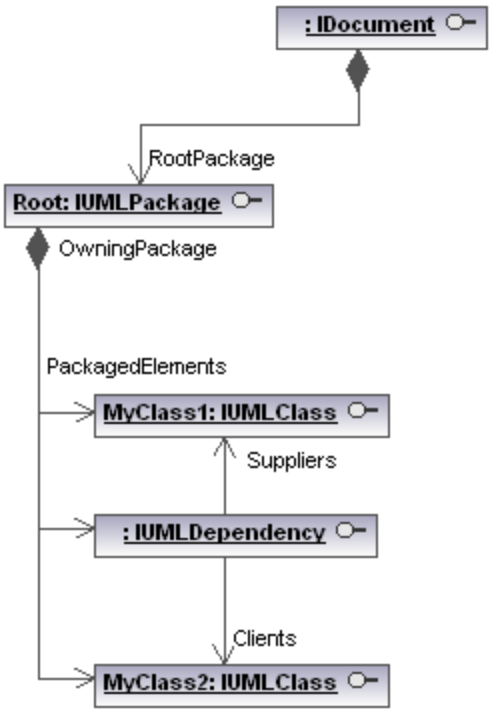
In der UML Superstructure Specification sind außerdem die folgenden Beziehungen zwischen `Package` und `PackageableElement` definiert:



Dies ist IUMLPackageableElement[1241] zugeordnet und hat eine Eigenschaft `OwningPackage` und ein IUMLPackage[1238], das nicht nur eine Eigenschaft `PackagedElements` hat, sondern auch eine Methode `InsertPackagedElementAt`, um neue IUMLPackageableElement[1241]e (an der angegebenen Position) einzufügen. Die Methode `EraseFromModel` löscht alle IUMLElement[1157]e (und deren untergeordnete Elemente) aus dem Modell.

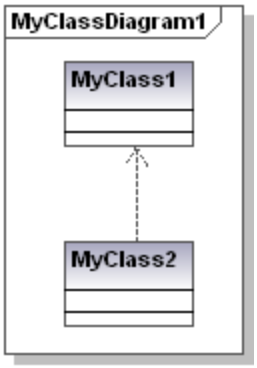
Im Beispiel unten sehen Sie das Mapping eines Projekts, das aus zwei Klassen (IUMLClass[1121]) mit einer Abhängigkeit zwischen diesen Klassen (IUMLDependency[1148]) besteht:

© 2016-2022 Altova GmbH



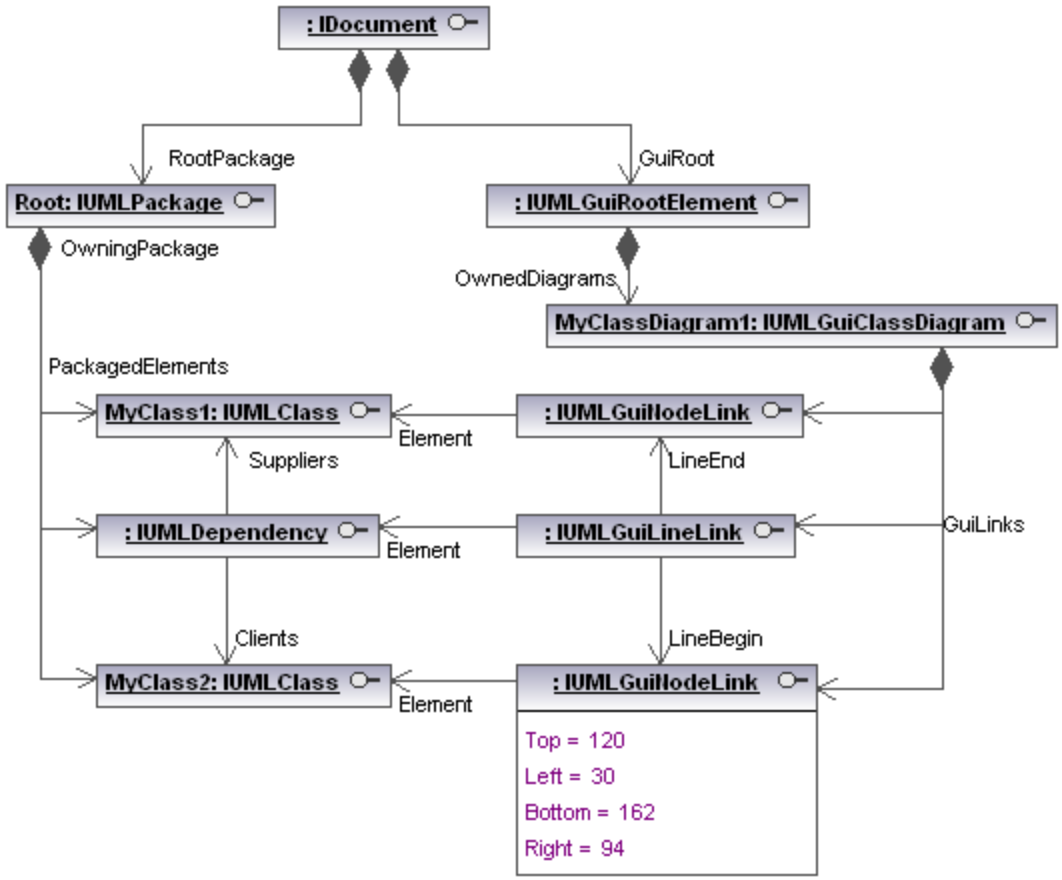
Diese Struktur ist unabhängig davon, ob diese Elemente in einem Diagramm gezeigt werden oder nicht.

Die Darstellung grafischer Objekte in Diagrammen (wie im Bild unten gezeigt) wird in einer zweiten Struktur mit Elementen der Art IUMLGuiElement 1304 (siehe auch Grafische Objekte 863) gespeichert.



Der Ausgangspunkt zu Aufrufen von UML GUI-Elementen ist die GuiRoot (IUMLGuiRootElement 1338), welche eine Eigenschaft der IDocument 938 Schnittstelle ist.

Linien werden durch IUMLGuiLineLink 1326s behandelt, die meisten anderen Objekte (wie Klassen, Schnittstellen, Pakete,...) durch IUMLGuiNodeLink 1331s.

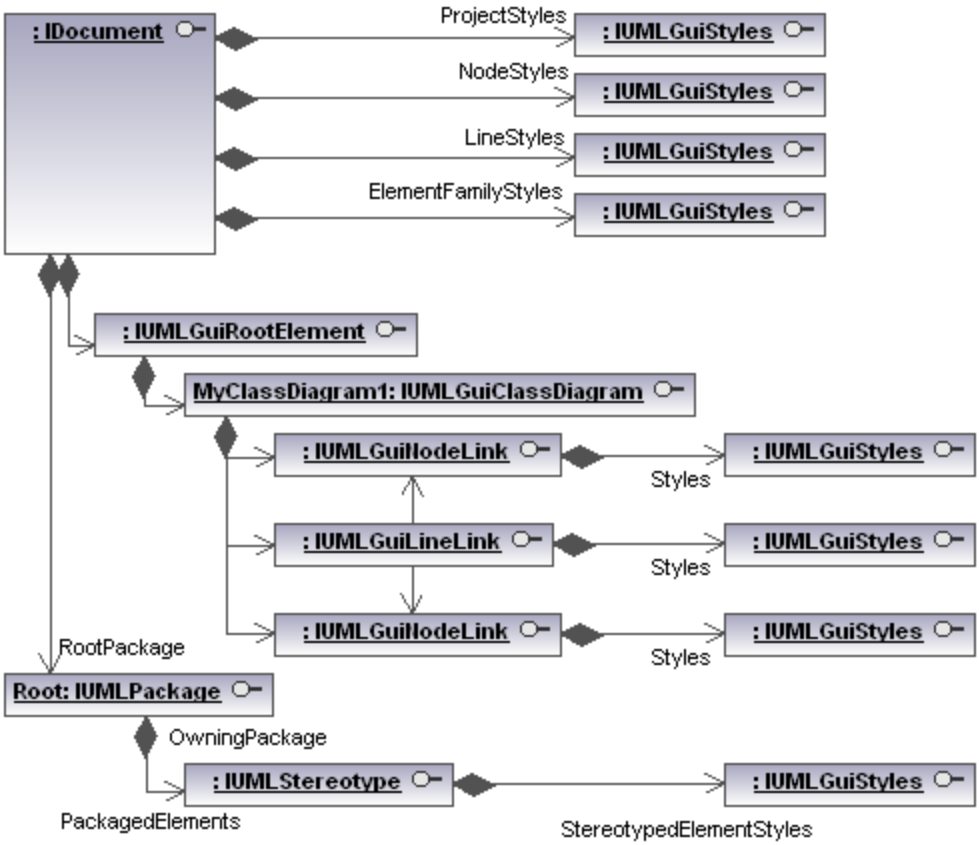


17.4.2.2 Objektmodell UMLData-Stile

UModel verfügt über verschiedene Stile[93], mit denen Sie das Aussehen von Diagrammen (Schriftgröße, Schriftstärke, Farbe, Sichtbarkeitsoptionen) anpassen können.

In der folgenden Abbildung sehen Sie, wie verschiedene Stile (IUMLGuiStyles[1346]) über die UModel API[857] aufgerufen werden können:

© 2016-2022 Altova GmbH



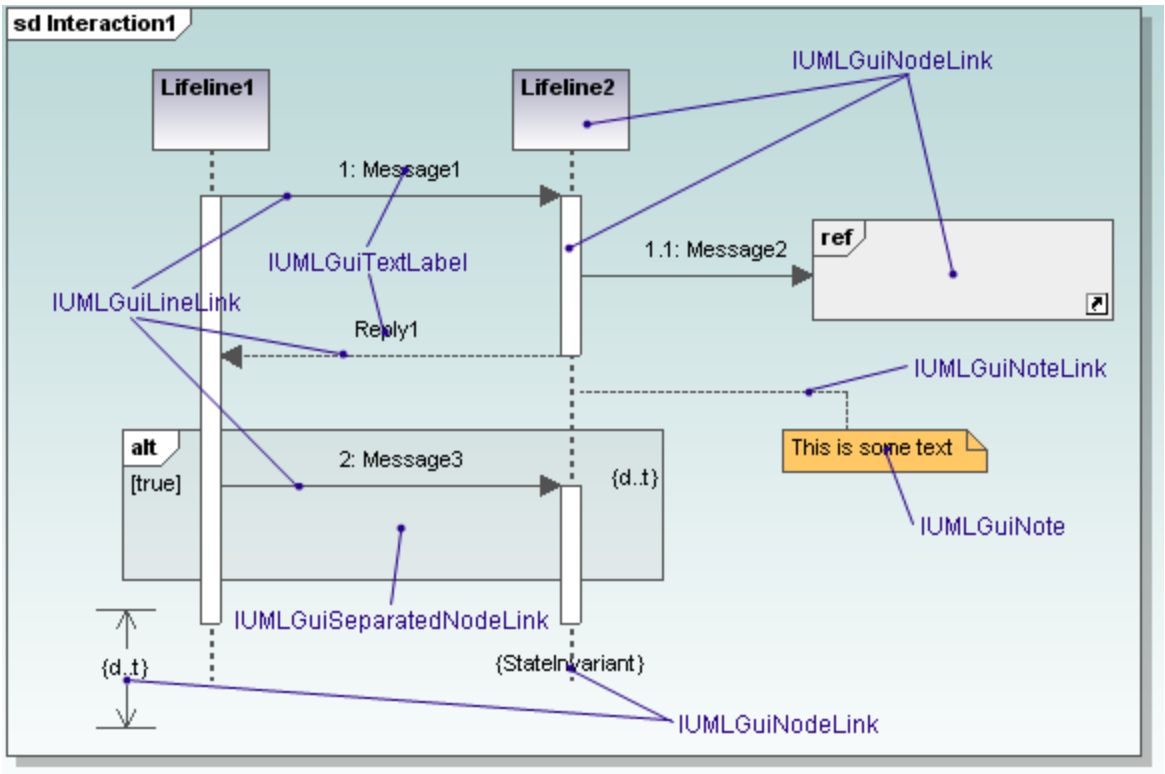
Die verschiedenen Stile können durch ENUMUMLGuiStyleKind[1371] identifiziert werden.

17.4.2.3 Grafische Objekte

In der UModel API[857] werden grafische Objekte in Diagrammen durch Objekte dargestellt, die von der IUMLGuiElement[1304] Schnittstelle abgeleitet werden. Die meisten davon können über die Eigenschaft IUMLGuiDiagram[1316] 'GuiLinks' abgerufen werden.

Bei den meisten Diagrammen sind die meisten Objekte, die Linien darstellen, Instanzen von IUMLGuiLineLink[1326]. Die meisten anderen, festen Objekte oder "Nodes" sind Instanzen von IUMLGuiNodeLink[1331]. Diese Schnittstellen haben Eigenschaften und Methoden zum Bearbeiten der grundlegenden Eigenschaften dieser grafischen Objekte wie z.B. Position, Farbe und Stil.

Es gibt natürlich noch mehr spezialisierte Schnittstellen, die von diesen allgemeinen Schnittstellen abgeleitet werden und die Zugriff auf Sondereigenschaften bieten. In der folgenden Abbildung sehen Sie ein Sequenzdiagramm und die Schnittstelle, die die einzelnen grafische Objekte darin repräsentiert:



17.4.3 Anleitungen

17.4.3.1 Anleitung zum Erstellen von Sequenzdiagrammen

Es gibt zwei Methoden, um über die UModel API in Form von Programmcode Sequenzdiagramme zu erstellen:

- Generieren eines Sequenzdiagramms anhand von bestehendem Quellcode[864], wenn es Code gibt, den Sie mittels Reverse Engineering erzeugen und als UML-Diagramm darstellen möchten.
- Direktes manuelles Erzeugen eines Sequenzdiagramms[865] von Grund auf neu mit Hilfe von IUMLGuiElements

17.4.3.1.1 Anleitung zum Generieren von Sequenzdiagrammen anhand von Code

Sequenzdiagramme können in UModel durch Programmcode über ein IUMLOperation[1236] Element generiert werden. Die Operation muss im Modell vorhanden und mit Quellcode verknüpft sein.

Die Operation wurde möglicherweise bereits früher von der Reverse Engineering-Funktionalität von UModel "gelesen". Die Erstellung neuer Sequenzdiagramme aus einem Programm heraus durch das Reverse Engineering von Quellcode mittels der UModel API[857] erfolgt in zwei kurzen Schritten:

- Definieren der Optionen zur Diagrammgenerierung
- Aufrufen der Diagrammgenerierungsfunktion

© 2016-2022 Altova GmbH

Im folgenden C#-Code wird gezeigt, wie Sie die Optionen definieren und die Generierung des Sequenzdiagramms starten:

```
// starts the sequence diagram generation process based on an operation given as
parameter
public static void reverseEngineerAndCreateSequenceDiagram(IApplication application,
IUMLOperation operation)
{
    GenerateSequenceDiagramDlg dialog = application.Dialogs.GenerateSequenceDiagramDlg;

    // set some options
    dialog.ShowEmptyCombinedFragments = false;
    dialog.UseDedicatedLineForStaticCalls = true;
    dialog.ShowCodeOfMessagesDisplayedDirectlyBelow = true;
    dialog.ShowCodeInNotes = true;

    dialog.ShowDialog = true; // set this to true if you want the dialog to be displayed

    // generated the sequence diagram now
    application.ActiveDocument.GenerateSequenceDiagram(dialog, operation);
}
```

17.4.3.1.2 Anleitung zum manuellen Erstellen von Sequenzdiagrammen

Wenn Sie ein Sequenzdiagramm aus einem Programm heraus über die UModel API[857] von Grund auf neu erstellen, werden im Prinzip einfach Interaktionsfragmente wie z.B. Lebenslinien in ein Diagramm platziert und mit Nachrichten verknüpft.

Nachrichten können ganz einfach mit Hilfe der **`AddUMLLineElement()`** `Methode` von `IUMLGuiLineLink`[1328] erstellt werden, wodurch nicht mehrere zugrunde liegende UML-Elemente wie z.B. MessageEnds, ExecutionOccurrences und ähnliches manuell erstellt werden müssen.

Um das Erstellen von Nachrichten zwischen zwei Interaktionsfragmenten wie z.B. Lebenslinien einfacher zu machen, erstellen Sie eine kleine Hilfsfunktion, die `AddUMLLineElement()` aufruft und die erstellte Linie positioniert:

```
// Creates a message between two interaction fragments (i.e. lifelines, interaction uses,
// combined fragments or gates) and attaches all necessary elements like events and
activation bars.
// Possible values for 'kind': "Message", "Reply", "Create", "Destruct"
protected static IUMLMessage addMessage(int ypos, string kind,
                                IUMLGuiNodeLink from, IUMLGuiNodeLink to,
                                DiagramWindow wnd)
{
    // add message
    IUMLGuiLineLink line = wnd.Diagram.AddUMLLineElement(kind, from, to);

    if (line == null)
        return null;

    // set position of the line where we want it to show up
    wnd.UpdateWindow();
```

```
    if (from == to && line.Waypoints.Count > 3)
    {
        // self-message
        ((IUMLGuiWaypoint)line.Waypoints[1]).SetPos(0, ypos);
        ((IUMLGuiWaypoint)line.Waypoints[4]).SetPos(0, ypos + 25);
    }
    else
    if (line.Waypoints.Count > 1)
    {
        // normal message
        ((IUMLGuiWaypoint)line.Waypoints[1]).SetPos(0, ypos);
        ((IUMLGuiWaypoint)line.Waypoints[2]).SetPos(0, ypos);
    }

    return (IUMLMessage)line.Element;
}
```

Wie Sie sehen, akzeptiert IUMLDiagram.AddUMLLineElement() 1316 als Parameter nicht nur den String "Message", um eine Nachrichtenzeile zu erstellen, sondern auch "Reply", "Create" und "Destruct" für Antwortnachrichten, Erstellungsnachrichten und Löschungsnachrichten.

Um ein einfaches Diagramm zu erstellen, müssen Sie nur im GuiRoot-Objekt ein Sequenzdiagramm erstellen, das Diagramm öffnen, ein paar Lebenslinien hinzufügen und diese mit Hilfe dieser Hilfsfunktion mit Nachrichten verbinden:

```
IDocument document = theapplication.ActiveDocument;

// create diagram and open it
IUMLGuiSequenceDiagram sequenceDiagram =
   (IUMLGuiSequenceDiagram)document.GuiRoot.InsertOwnedDiagramAt(0, document.RootPackage,
 "SequenceDiagram");

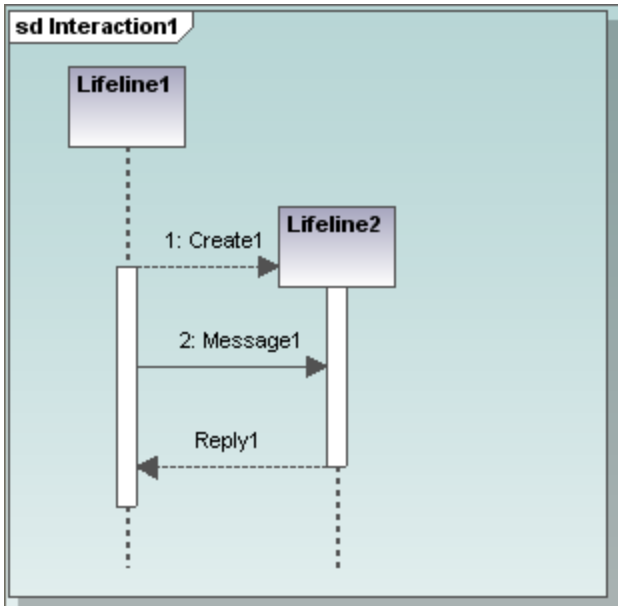
DiagramWindow wnd = document.OpenDiagram(sequenceDiagram);

// create two lifelines
IUMLGuiNodeLink lifeline1 = sequenceDiagram.AddUMLElement("Lifeline", 0, 0);
IUMLGuiNodeLink lifeline2 = sequenceDiagram.AddUMLElement("Lifeline", 100, 70);

// connect these lifelines using some messages
addMessage(100, "Create",  lifeline1, lifeline2, wnd);
addMessage(150, "Message", lifeline1, lifeline2, wnd);
addMessage(200, "Reply",   lifeline2, lifeline1, wnd);
```

Das erzeugte Diagramm sollte folgendermaßen aussehen:

© 2016-2022 Altova GmbH



Definieren des Typs einer Lebenslinie

Um den von einer Lebenslinie repräsentierten Typ - sei es eine Klasse, Schnittstelle, ein Datentyp oder ähnliches - anzuzeigen, verwenden Sie die Eigenschaft `IUMLLifeline.Represents`[1209], die eine `IUMLProperty`[1251] referenziert. Wenn der Typ dieser Eigenschaft definiert ist, wird er ebenfalls im Diagramm angezeigt.

Mit dem folgenden Code wird eine Lebenslinie erstellt, die eine Klasse referenziert:

```
// create a class to be referenced by the lifeline
IUMLClass someclass = (IUMLClass)document.RootPackage.InsertPackagedElementAt(0,
"Class");

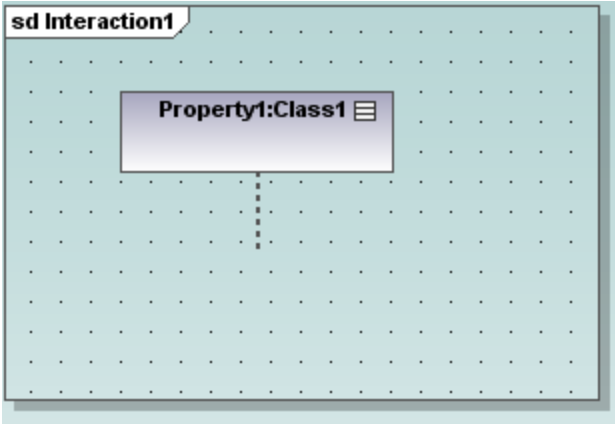
// create a lifeline and a property with the class as type in the interaction
// of the sequence diagram to reference this class
IUMLInteraction interaction = (IUMLInteraction)sequenceDiagram.LinkedOwner;

IUMLProperty prop = interaction.InsertOwnedAttributeAt(0);
prop.Type = someclass;

UModelLib.IUMLLifeline lifeline = interaction.InsertLifelineAt(0);
lifeline.Represents = (IUMLConnectableElement)prop;

// show the lifeline on the diagram
sequenceDiagram.AddUMLGuiNodeLink(lifeline, 200, 0);
```

Die erzeugte Lebenslinie würde folgendermaßen aussehen:



Definieren der Operation einer Nachricht

Nachrichten repräsentieren normalerweise den Aufruf einer Operation eines Objekts. Bitte beachten Sie: Je nach Typ der Nachricht (normale Nachricht, Erstellungsnachricht, Löschungsnachricht oder Antwortnachricht) und dem Vorhandensein bzw. Fehlen eines zugrunde liegenden UML-Elements wie z.B. MessageOccurenceSpecifications oder CallEvents, kann eine Nachricht nicht immer eine Operation repräsentieren und der Abruf des richtigen UML-Elements, das auf die Operation verweist, ist nicht immer so einfach.

Aus diesem Grund bietet die Schnittstelle IUMLMessage[1216] in der UModel API die Methode **SetOperation(),** mit Hilfe derer man eine Nachricht eine Operation referenzieren lassen kann, falls dies möglich ist:

```
// create a message, an operation in a class and let the message refer this operation
IUMLMessage msg = addMessage(250, "Message", lifeline1, lifeline2, wnd);

UModelLib.IUMLOperation someoperation = someclass.InsertOwnedOperationAt(0);
someoperation.Name = "SomeOperation";

msg.SetOperation(someoperation);
```

17.4.3.2 Anleitung zum Rückgängigmachen/Wiederholen und der Behandlung von UMLData-Transaktionen

Wenn Sie die UML-Datenstruktur über die UModel API[857] ändern, müssen Sie sich nicht um das Rückgängigmachen / Wiederholen von Aktionen oder Transaktionen kümmern.

Der folgende Code nimmt drei Änderungen vor:

```
        public void ChangeClass( IUMLClass iClass )
        {
            iClass.SetName("NewName");
            iClass.Visibility = ENUMUMLVisibilityKind.eVisibility_Public;
            iClass.IsAbstract = true;
        }
```

Für jede Änderung wird ein neuer Rückgängig-Schritt erstellt. Anders ausgedrückt: Der Benutzer muss in UModel drei Mal auf die Schaltfläche "Rückgängig" klicken, um diese drei Änderungen rückgängig zu machen.

© 2016-2022 Altova GmbH

Dies ist nicht immer das gewünschte Verhalten, daher unterstützt die UModel API[857] das "Transaction-Handling", sodass mehrere Änderungen in einem einzigen Schritt ausgeführt werden können.

IDocument[938] hat die Aufgabe zu definieren, wann eine Gruppe von Änderungen beginnt ("BeginModification") und wann sie endet ("EndModification"):

```
public void ChangeClass(IUMLClass iClass, IDocument iDoc)
{
    try
    {
        // make all modifications within one UndoStep; start modification here
        if (!iDoc.BeginModification())
            return;

        iClass.SetName("NewName");
        iClass.Visibility = ENUMUMLVisibilityKind.eVisibility_Public;
        iClass.IsAbstract = true;

        // do not forget to end modification and finish UndoStep
        iDoc.EndModification();
    }
    catch (System.Exception)
    {
        // rollback made changes
        iDoc.AbortModification();

        // add error handling
    }
}
```

Diese Art von Transaktionsbehandlung kann nur bei der Änderung von UML-Daten verwendet werden.
Bei anderen Funktionen, wie z.B. "Modell anhand von Code synchronisieren" wird ohnedies ein einziger Rückgängig-Schritt erstellt.

17.4.3.3 Anleitung zur Verwendung vordefinierter UModel-Elemente

In UModel sind einige wichtige Elemente als "vordefiniert" definiert. Dazu gehören eine Reihe von internen Elementen (Root, Komponentenansicht und Paket "unbekannte externe Elemente" sowie die Elemente aller mit UModel installierter Profile (z.B. das C#, VB und Java Profil).

Vordefinierte Elemente können mit Hilfe von ENUMUMLPredefinedElement[1376] eindeutig identifiziert werden. Dadurch haben Sie direkten und einfachen Zugriff auf diese Elemente und zwar für verschiedene Funktionalitäten wie z.B.

- Suchen eines vordefinierten Elementes:

```
// get the CSharp profile
IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc.RootPackage.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_P
rofile, false);
```

- Anwenden eines vordefinierten Stereotyps

```
// set the CSharp 'delegate' stereotype
iClass.ApplyPredefinedStereotype(
 ENUMUMLPredefinedElement.ePredefined_CSharp_delegateStereotypeOfClass );
```

- Überprüfen, ob ein vordefinierter Stereotyp angewendet wird:

```
// check if package is a CSharp - namespace (if 'namespace' stereotype is applied)
bool bIsCSharpNamespace =
iPackage.IsPredefinedStereotypeApplied(
 ENUMUMLPredefinedElement.ePredefined_CSharp_namespaceStereotypeOfPackage );
```

- Setzen des Eigenschaftswerts einer vordefinierten Eigenschaftswertdefinition:

```
// set attribute-section "STAThread"
// ...
iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
 iSTAThread.Name);
```

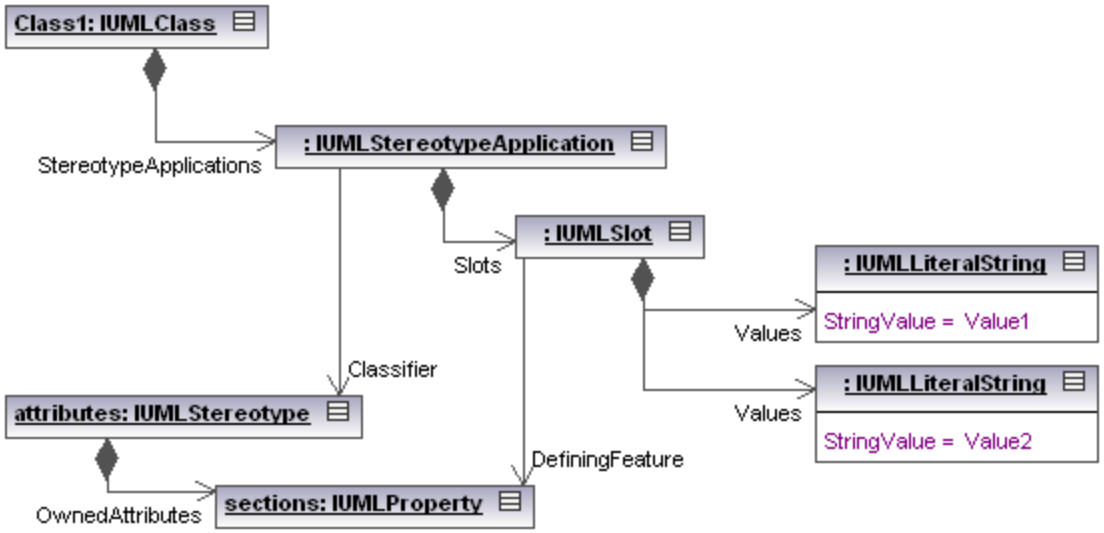
17.4.3.4 Anleitung zum Arbeiten mit Stereotypen und Eigenschaftswerten

Stereotype und Eigenschaftswerte sind, wie in der UML Superstructure Specification definiert, ziemlich komplex. UModel hat die Arbeit damit vereinfacht und behandelt sie ähnlich wie IUMLInstanceSpecification[1190]s und IUMLSlot[1266]s in UML. Im folgenden Beispiel wird das Stereotyp "attributes" auf "Class1" angewendet und die Tag-Definition "sections" hat die Eigenschaftswerte "Value1" und "Value2":



In der UModel API werden IUMLStereotypeApplication[1275]s vorgestellt und das Beispiel oben wird auf die folgende UMLData-Struktur gemappt:

© 2016-2022 Altova GmbH



Die Anwendung von Stereotypen und das Definieren von Eigenschaftswerten mit Hilfe der UModel API ist relativ einfach:

```
IUMLStereotype iStereotypeAttributes = ...;
IUMLProperty iTagDefSections = ...;
IUMLClass iClass = ...;

IUMLStereotypeApplication iStereotypeApp = iClass.ApplyStereotype(iStereotypeAttributes);
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value1");
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value2");
```

Informationen zur Behandlung vordefinierter Stereotypen, Tag-Definitionen und Eigenschaftswerte finden Sie im Abschnitt Vordefinierte UModel-Elemente 869.

17.4.3.5 Anleitung zur Verwendung von UMLData-Events und Event-Filtern

Event-Empfänger müssen die _IUMLDataEvents 1368 Schnittstelle implementieren, um eines oder mehrere der folgenden möglichen Events von IUMLData 1011 zu empfangen:

`OnBeforeErase`	Wird unmittelbar bevor die UML-Daten aus dem Modell gelöscht werden, gesendet. Wenn mehrere Daten gelöscht werden, wird dieses Event für jedes IUMLData 1011 Event gesendet (nicht nur für das oberste)
`OnAfterAddChild`	Wird gesendet, wenn die UML-Daten zur Modellstruktur hinzugefügt werden. Wenn mehrere Daten auf einmal hinzugefügt werden (z.B. wenn eine Klasse mit mehreren Attributen zu einem Paket hinzugefügt wird), wird nur das oberste IUMLData 1011 Event gesendet.

OnChanged	Wird gesendet, wenn die UML-Daten geändert wurden (z.B. wenn ein Klassenname geändert wurde)
OnMoveData	Wird gesendet, wenn UML-Daten in ein neues übergeordnetes Event verschoben wurden (z.B. wenn eine Klasse in der Modellstruktur in ein anderes Paket verschoben wird). Dieses Event kommt immer zwei Mal vor: einmal, beim Entfernen aus dem alten übergeordeten Event und einmal, wenn die UML-Daten an das neue übergeordnete Event angehängt werden.

Eventfilter können mit (Kombinationen von) ENUMUMLDataEventFilter[1370]n definiert werden, um festzulegen, welche Events von der UModel API[857] gesendet werden sollen. Um eine möglichst schnelle Verarbeitung bei möglichst geringem Speicherplatzbedarf zu gewährleisten, sollten Event-Empfänger nur für diejenigen Events registriert werden, die wirklich betroffen sind.

So registriert z.B. der folgende Code "OnAfterAddChild" Events, wenn genau das Root-Paket ein neues Child-Element erhält (es wird kein Event empfangen, wenn ein Child des Root-Pakets ein neues Child-Event erhält):

```
// ensure we get informed when m_RootPackage (and only itself; we do not care about its
children) gets a new child
m_RootPackage.EventFilter = (int)ENUMUMLDataEventFilter.eUMLDataEvent_AddChild;
```

UMLData Events arbeiten hierarchisch, daher kann der Event-Filter so eingestellt werden, dass nur Events vom angehängten IUMLData[1011] Event empfangen werden oder von angehängten IUMLData[1011] Event und allen seinen untergeordneten Events.

```
// ensure we get "OnBeforeErase" events also for *any* erased child (grandchild,...) of
the rootpackage
m_RootPackage.EventFilter |= (int)ENUMUMLDataEventFilter.eUMLDataEvent_EraseDataOrChild;
```

UMLData Events werden auch gesendet, wenn UML-Daten durch Rückgängig / Wiederholen geändert werden. Beachten Sie allerdings, dass während eines Rückgängig / Wiederholen-Befehls **keine** Änderungen an UML-Daten vorgenommen werden können:

```
public void OnAfterAddChild(IUMLData ipUMLParent, IUMLData ipUMLChild)
{
   // check if child was added by undo/redo
      // (we are not allowed to modify anything during Undo/Redo !!)
      IDocument iDoc = (IDocument)ipUMLChild.Parent;
      if (!iDoc.IsInUndoRedo)
      {
         // ...
      }
}
```

17.4.3.6 Anleitung zum Erstellen und Verwenden von Hyperlinks

Sie können in UModel Hyperlinks zwischen den meisten Modellierungselementen (mit Ausnahme von Linien) und folgenden Elementen erstellen:

© 2016-2022 Altova GmbH

- jedes Diagramm im aktuellen UMP-Projekt
- jedes Element in einem Diagramm
- jedes Element in der Modell-Struktur
- externe Dokumente, z.B. PDF, Excel oder Word-Dokumente
- Webseiten

Siehe auch Hyperlinking modeling elements 122.

Hyperlinks bilden nicht Teil der UML-Spezifikation und die UModel API stellt die folgenden Schnittstellen für Hyperlinks in

IUMLNamedElement 1222en bereit:

- IUMLHyperlink 1181 ist die allgemeine Basisschnittstelle und kann verwendet werden, um Links zu öffnen und den benutzerdefinierten und den Standard-Linknamen abzurufen
- IUMLHyperlink2File 1182 dient zum Behandeln externer Dokumente und Webseiten
- IUMLHyperlink2GuiElement 1183 dient zum Behandeln jedes Diagramms im aktuellen UMP-Projekte oder jedes Elemens in einem Diagramm
- IUMLHyperlink2Model 1184 dient zum Erstellen von Hyperlinks zu Modellelementen (in der Modell-Struktur)

Beispiele

Einfügen eines Links zur Altova-Homepage:

```
IUMLHyperlink2File iHyperlink = iMyClass.InsertOwnedHyperlink2FileAt(-1,
"http://www.altova.com");
```

Einfügen eines Hyperlink zu einem Diagramm im aktuellen UMP-Projekt:

```
IUMLGuiDiagram iDiagram = ...;
IUMLHyperlink2GuiElement iHyperlink = iMyClass.InsertOwnedHyperlink2GuiElementAt(-1,
iDiagram, null);
```

Einfügen eines Hyperlink zur Darstellung einer Klasse in einem Diagramm:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLHyperlink2GuiElement iHyperlink = iMyClass.InsertOwnedHyperlink2GuiElementAt(-1,
iNodeLink, null);
```

Einfügen eines Hyperlink zu einem Attribut einer Klasse in einem Diagramm:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLProperty iAttribute = ...;
IUMLHyperlink2GuiElement iHyperlink = iMyClass.InsertOwnedHyperlink2GuiElementAt(-1,
iNodeLink, iAttribute);
```

Einfügen eines Hyperlink zum selben (obigen) Attribut in der Modell-Struktur:

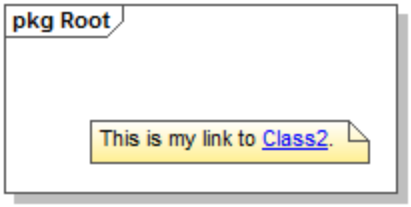
```
IUMLHyperlink2Model iHyperlink = iMyClass.InsertOwnedHyperlink2ModelAt(-1, iAttribute);
```

Öffnen aller Hyperlinks eines IUMLNamedElement[1222]:

```
foreach (IUMLHyperlink iHyperlink in iMyClass.OwnedHyperlinks)
   iHyperlink.OpenLink();
```

Sie können in UModel auch in Notizen (IUMLGuiNote[1333]) und Kommentare (IUMLComment[1132]) Hyperlinks einfügen:





Diese werden durch IUMLGuiTextHyperlink[1355]s (bzw. IUMLCommentTextHyperlinks[1133]) behandelt und die Position des ersten und letzten Zeichens des Hyperlink muss definiert werden, z.B.

```
IUMLGuiDiagram iDiagram = ...;
IUMLGuiNote iNote = iDiagram.AddUMLGuiNote(200, 100);

iNote.NoteText = "This is my Altova link";
int nStart = iNote.NoteText.IndexOf("Altova");
int nEnd = nStart + "Altova".Length;

IUMLGuiTextHyperlink iHyperlink = iNote.InsertOwnedGuiTextHyperlinkAt(nStart, nEnd,
"http://www.altova.com");
```

Ähnlich bei Hyperlinks in Kommentaren:

```
IUMLComment iComment = ...;
IUMLClass iClass2 = ...;

iComment.Body = "This is my link to Class2";
int nStart = iComment.Body.IndexOf("Class2");
int nEnd = nStart + "Class2".Length;

IUMLCommentTextHyperlink iHyperlink = iComment.InsertOwnedCommentTextHyperlinkAt(nStart,
nEnd, "");
iHyperlink.SetHyperlinkModelElementAddress( iClass2 );
```

17.4.3.7 Fehlerbehandlung

Die UModel API gibt Fehler auf zwei unterschiedliche Arten zurück. Jede API-Methode gibt ein `HRESULT` zurück. Dieser Rückgabewert informiert den Aufrufer über etwaige Fehler, die während der Ausführung der Methode aufgetreten sind. Wenn der Aufruf erfolgreich war, ist der Rückgabewert gleich `S_OK`. C/C++-Programmierer verwenden im Allgemeinen `HRESULT`, um Fehler ausfindig zu machen.

© 2016-2022 Altova GmbH

In VisualBasic, Scripting-Sprachen und anderen komplexen Entwicklungsumgebungen hat der Programmierer keinen Zugriff auf das zurückgegebene `HRESULT` eines COM-Aufrufs. Es wird der zweite Fehlermeldungsmechanismus der UModel API verwendet, die `IErrorInfo` Schnittstelle. Bei Auftreten eines Fehlers erstellt die API ein neues Objekt, das die `IErrorInfo` Schnittstelle implementiert. Die Entwicklungsumgebung nimmt diese Schnittstelle und setzt die bereitgestellten Informationen in ihre eigenen Fehlerbehandlungsmechanismen ein.

Im folgenden Beispielcodefragment sehen Sie, wie mit Fehlern verfahren wird, die von der UModel API in unterschiedlichen Umgebungen ausgelöst werden.

VisualBasic

Eine gängige Methode zur Fehlerbehandlung in VisualBasic ist die Definition eines Error Handler. Dieser Error Handler kann mit der `On Error` Anweisung definiert werden. Normalerweise zeigt der Handler eine Fehlermeldung an und führt Cleanup-Funktionen durch, um überzählige Referenzen und jede Art unnötiger Ressoucenbeanspruchung zu vermeiden.

VisualBasic füllt sein eigenes `Err` Objekt mit den Informationen aus der `IErrorInfo` Schnittstelle.

```
Sub Validate()
  'place variable declarations here

  'set error handler
  On Error GoTo ErrorHandler

  'if DoSomeWork fails, program execution continues at ErrorHandler:
  objUModel.ActiveDocument.DoSomeWork()

  'additional code comes here

  'exit
  Exit Sub

  ErrorHandler:
  MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
     "Description: " & Err.Description)
End Sub
```

JavaScript

Die Microsoft Implementierung von JavaScript (JScript) bietet einen try-catch-Mechanismus zur Behandlung von Fehlern, die durch COM-Aufrufe ausgelöst werden. Die Methode ist der VisualBasic-Methode insofern sehr ähnlich, als auch hier ein Fehlerobjekt deklariert wird, das die nötigen Informationen enthält.

```
   function Generate()
   {
      // please insert variable declarations here

      try
      {
         objUModel.ActiveDocument.DoSomeWork();
      }
      catch(Error)
      {
```

```
        sError = Error.description;
        nErrorCode = Error.number & 0xffff;
        return false;
    }

    return true;
}
```

C/C++

In C/C++ haben Sie einfachen Zugriff auf das `HRESULT` des COM-Aufrufs und die `IErrorInterface`.

```
HRESULT hr;

// Call DoSomeWork() from the UModel API
if(FAILED(hr = ipDocument->DoSomeWork()))
{
    IErrorInfo *ipErrorInfo = Null;

    if(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
    {
        BSTR   bstrDescr;
        ipErrorInfo->GetDescription(&bstrDescr);

        // handle Error information
        wprintf(L"Error message:\t%s\n",bstrDescr);
        ::SysFreeString(bstrDescr);

        // release Error info
        ipErrorInfo->Release();
    }
}
```

17.4.4 C# API-Beispiele

Als Einstiegshilfe enthält Ihr UModel-Paket unter dem folgenden Pfad ein C*-Beispielprojekt: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\API**.

Dieses Beispielprojekt enthält eine Referenz auf die UModel-Typbbibliothek, siehe Referenzieren der UModel-Typbibliothek 877. Eine solche Referenz ist in jedem Projekt, für das die UModel API benötigt wird, erforderlich, damit das Applikationshauptobjekt von Ihrem Code aus folgendermaßen referenziert werden kann:

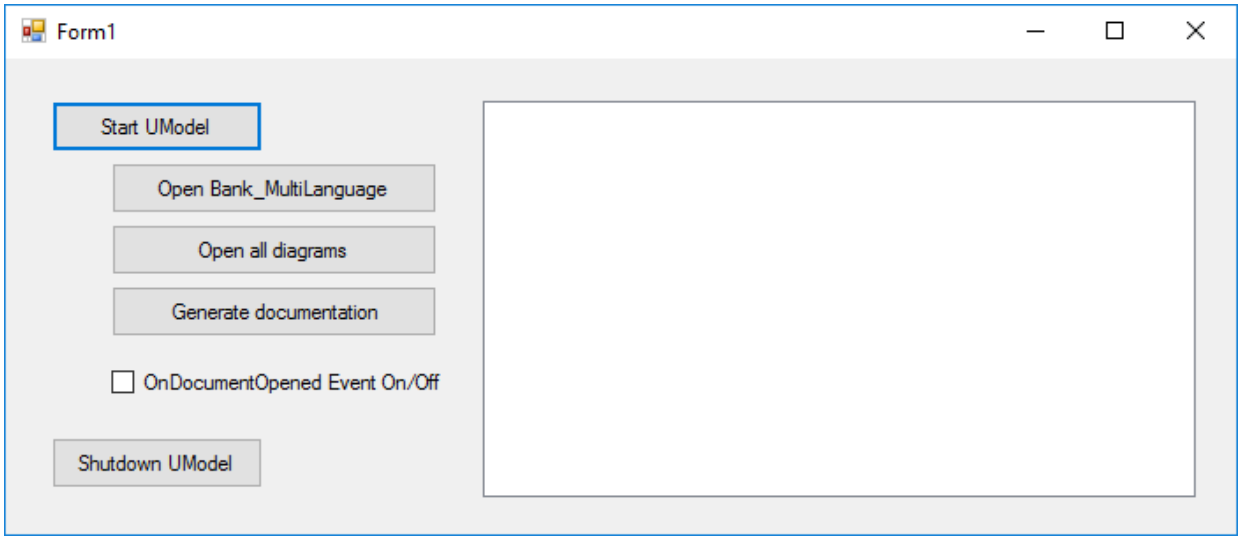
```
UModelLib.Application um = new UModelLib.Application();
MessageBox.Show(String.Format("Hello from UModel API version {0}.{1}",
um.APIMajorVersion, um.APIMinorVersion));
```

Wenn Sie ein 64-Bit-Betriebssystem haben und eine 32-Bit-Installation von UModel verwenden, fügen Sie die x86-Plattform im Konfigurationsmanager der Lösung hinzu und erstellen Sie das Beispiel mit dieser Konfiguration. Um den Konfigurationsmanager aufzurufen, klicken Sie auf den Menübefehl **Build | Configuration Manager**

© 2016-2022 Altova GmbH

In der Beispielapplikation sehen Sie ein Windows-Formular mit Schaltflächen, über die grundlegende MapForce-Operationen aufgerufen werden:

- Starten von UModel
- Öffnen von **Bank_MultiLanguage.ump**
- Öffnen aller Diagramme
- Generierung von Dokumentation für das gerade aktive Dokument
- Anzeige, wie auf Events gehört wird (`OnDocumentOpened` Event On/Off)
- Beenden von UModel

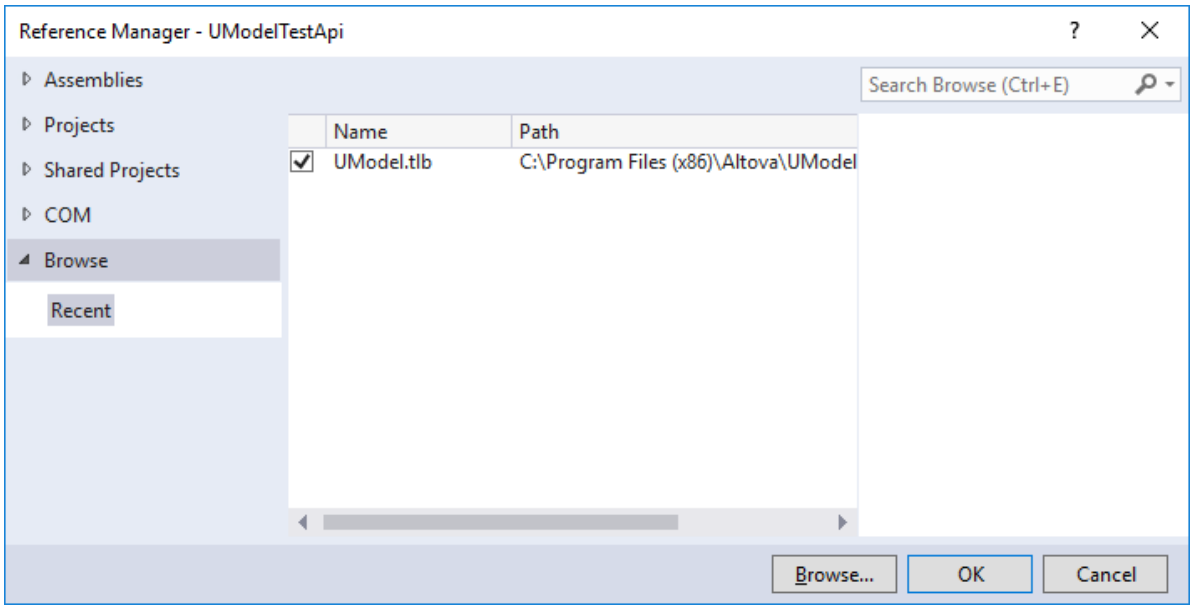


Der Code besteht im Grunde aus einer Reihe von Handlern für die Schaltflächen auf der oben gezeigten Benutzeroberfläche. Beachten Sie, dass Sie den über den Code referenzierten Pfad zum UModel-Beispielordner eventuell anpassen müssen.

17.4.4.1 Referenzieren der UModel-Typbibliothek

Um die API-Funktionalitäten von UModel über Ihr Visual Studio-Projekt nutzen zu können, fügen Sie zuerst in Visual Studio eine Referenz auf die UModel-Typbibliothek hinzu:

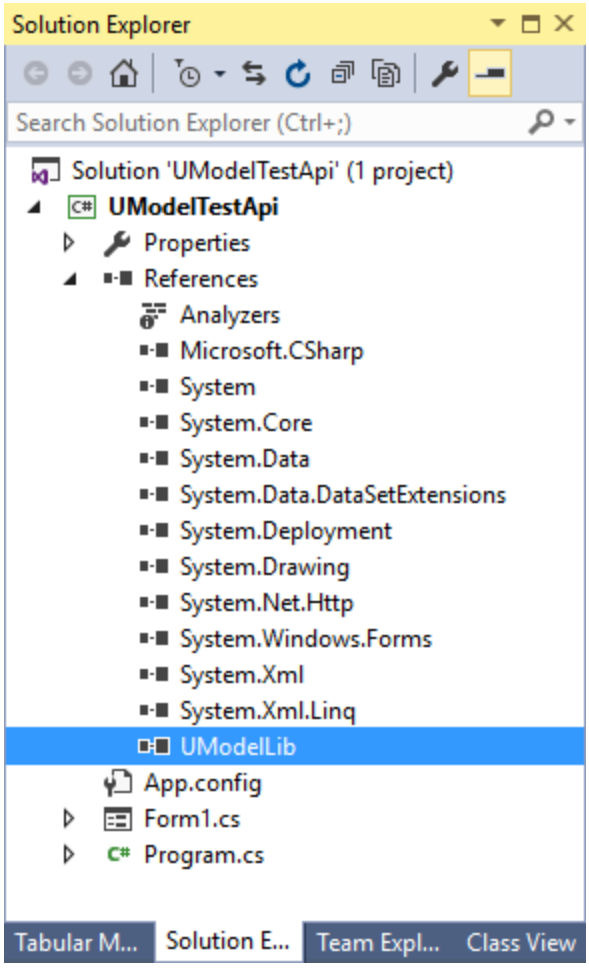
1. Erstellen Sie ein neues Visual Studio-Projekt oder öffnen Sie ein vorhandenes.
2. Klicken Sie im Menü **Project** auf **Add Reference**.
3. Wählen Sie im COM-Abschnitt **UModel Type Library** aus der Liste aus. Wenn dieser Eintrag im COM-Abschnitt nicht zur Verfügung steht, klicken Sie auf **Durchsuchen** und wählen Sie die Datei **UModel.tlb** aus dem UModel-Programmapplikationsordner aus.



Anmerkung: Die **UModel-Typbibliothek** ist nicht mit der **UModelPlugin-Typbibliothek** zu verwechseln. Mit letzterer können Sie Ihre eigenen Plug-ins erstellen und in UModel integrieren, siehe Hinzufügen einer Referenz zur UModel Plug-in-Bibliothek 838.

Nach Durchführung der oben beschriebenen Schritte sollte die UModel-Typbibliothek in der Liste der Referenzen Ihrer Visual Studio-Projektmappe zur Verfügung stehen, z.B:

© 2016-2022 Altova GmbH



17.4.4.2 Programmatischer Import von Binärtypen

Mit UModel können Sie Binärtypen aus .NET- .dll- oder Java .jar-Dateien entweder über die grafische Benutzeroberfläche oder programmatisch über die UModel API importieren. In diesem Beispiel wird gezeigt, wie Sie über die UModel API Binärtypen aus einer NET .dll-Datei in UModel importieren. Informationen zum Import von Binärtypen über die grafische Benutzeroberfläche finden Sie unter Importieren von Java-, C#- und VB.NET-Binärdateien[225].

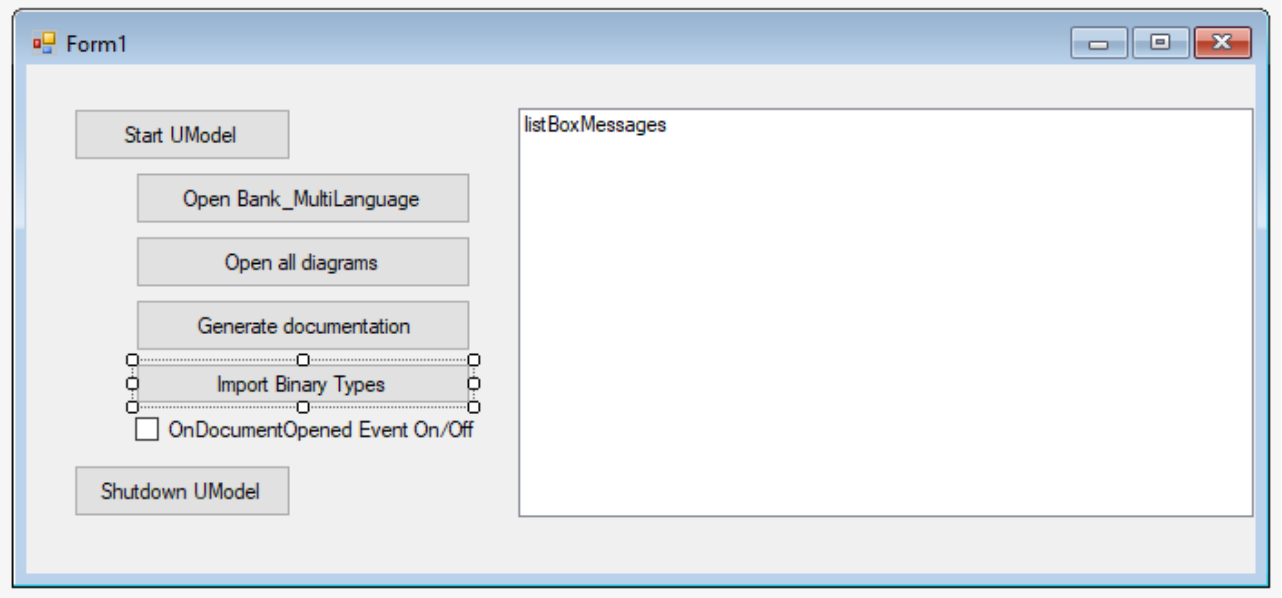
In diesem Beispiel wird Visual Studio 2019 und C# verwendet. Die unten stehende Anleitung ist (mit Ausnahme des Codefragments) ähnlich wie die für VB.NET. Um dieses Beispiel fertig stellen zu können, benötigen Sie eine .dll-Datei, die einige Typen (z.B. Klassen oder Schnittstellen) enthält, die Sie in UModel importieren möchten.

Zu diesem Zweck verwenden wir eine vorhandene C#-Demo-Applikation, die bereits in die UModel API integriert ist, anstatt ein Projekt von Grund auf neu zu erstellen. In dieser Demo wollen wir eine neue Schaltfläche hinzufügen. Wenn Sie darauf klicken, wird mit dieser Schaltfläche ein neues UModel-Projekt erstellt, in das Typen aus einer .dll-Datei importiert werden. Starten Sie zuerst Visual Studio und öffnen Sie die folgenden Projektmappe: **C:**

\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamplesAPI\C#\AutomateUModel_VS2010.sln.

Anmerkung: Die Demo-Applikation enthält bereits eine Referenz auf die UModel-Typbibliothek, daher muss nicht mehr explizit eine Referenz hinzugefügt werden. Wenn Sie jedoch ein neues Visual Studio-Projekt erstellen, müssen Sie die UModel-Typbibliothek von Ihrem Projekt aus referenzieren, siehe Referenzieren der UModel-Typbibliothek 877.

Öffnen Sie als nächstes `Form1.cs` im Design Editor und fügen Sie eine neue Schaltfläche hinzu. Nennen wir diese **Import Binary Types**.



Doppelklicken Sie auf die neue Schaltfläche und fügen Sie den folgenden Code in den Body der Handler-Methode ein. Stellen Sie sicher, dass der Pfad zur .dll-Datei korrekt ist und die .dll-Datei für den Import von Binärtypen geeignet ist (d.h. sie darf nicht verschleiert sein).

```
try
{
    // Create a new document
    UModelDocument = UModel.NewDocument();
    // Instantiate the Import Binary Types dialog
    UModelLib.ImportBinaryTypesDlg dlg = UModel.Dialogs.ImportBinaryTypesDlg;
    // Set the .NET runtime version according to your environment (must be greater than
v2.0) or use "any"
    dlg.Runtime = "any";
    // Set the import language (C# 8.0, in this case)
    dlg.Language = UModelLib.ENUMCodeLangVersion.eCodeLang_CSharp_8_0;
    // No need to show the dialog since we want to do this programmatically
    dlg.ShowDialog = false;
    // Add a new binary type entry to be imported
    UModelLib.IBinaryTypeEntry entry = dlg.CSharp_BinaryTypes.AddItem();
    // Specify the .dll to import (make sure to adjust the path)
    entry.Entry = "C:\\Path\\To\\My.dll";
```

 © 2016-2022 Altova GmbH

```
    // All types shall be imported from this .dll
    entry.ImportTypes = true;
    // The .dll is an executable
    entry.Executeable = true;
    // Perform the actual import
    UModelDocument.ImportBinaryTypes(dlg);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

Import aller Typen

Mit dem obigen Code wird ein neues UModel-Projekt erstellt, die Importoptionen im Dialogfeld "Binärtypen importieren" werden definiert und der Import der Binärtypen wird durchgeführt.

So führen Sie den C#-Code aus und importieren Binärtypen:

1. Drücken Sie **F5**, um die Visual Studio-Projektmappe zu erstellen und auszuführen.
2. Klicken Sie im daraufhin angezeigten Windows-Formular auf **UModel starten** und warten Sie, bis die UModel-Applikation geladen ist.
3. Klicken Sie erst, nachdem UModel fertig geladen wurde, auf **Import Binary Types** und überprüfen Sie das Ergebnis im Fenster "Meldungen" von UModel.

Wenn Sie nur bestimmte Typen importieren möchten, setzen Sie die Eigenschaft `ImportTypes` auf **false** und stellen Sie die zu importierenden Typen als Argumente für die Methode `TypesToImport` bereit. Die Liste der unterschiedlichen Typen kann, wie im Codefragment unten gezeigt, durch Kommas, Semikola oder Leerzeichen getrennt werden.

```
try
{
    UModelDocument = UModel.NewDocument();
    UModelLib.ImportBinaryTypesDlg dlg = UModel.Dialogs.ImportBinaryTypesDlg;
    dlg.ShowDialog = false;
    dlg.CSharp_BinaryTypes.RemoveAllItems();
    UModelLib.IBinaryTypeEntry entry = dlg.CSharp_BinaryTypes.AddItem();
    entry.Entry = "C:\\Path\\To\\My.dll";
    entry.ImportTypes = false;
    entry.Executeable = true;
    // import only specific types:
    entry.TypesToImport = "MyNamespace.Class1; MyNamespace.Class2";
    UModelDocument.ImportBinaryTypes(dlg);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

Import bestimmter Typen

17.4.4.3 Das Beispiel "Stile definieren"

Im folgenden Beispiel werden mehrere Stile für ausgewählte Diagrammelemente definiert (wenn der Stil verfügbar und nicht bereits definiert ist). Im Beispiel werden sowohl die UModel API als auch die UModel IDE Plug-In-Bibliothek verwendet. Es steht in der folgenden Datei zur Verfügung: **..\UModelExamples\IDEPlugIn\Styles\Styles.cs**.

Die Lösung enthält auch zwei Setup-Projekte (im Format .vdproj für 32-Bit und 64-Bit-Plattformen). Mit dem Setup werden alle erforderlichen Dateien installiert und das IDE Plug-in für COM sowie UModel werden auf dem Zielsystem registriert, sodass das Plug-in beim nächsten Start von UModel automatisch geladen wird.

Anmerkung:

- Für das Erstellen und Ausführen des Beispiels gelten dieselben Voraussetzungen wie für andere UModel IDE Plug-ins, siehe Erstellen und Ausführen des Plug-in 846.
- Visual Studio Setup-Projekte werden ab Visual Studio 2012 nicht unterstützt und benötigen eine separate Erweiterung, um geöffnet werden zu können. Nähere Informationen dazu finden Sie in den Informationsmeldungen des Visual Studio Migrationsassistenten.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using UModelLib;
using UModelPlugInLib;

/*
 * Styles sample
 * set following styles for selected diagram elements
 *      Fill Color
 *      Header Gradient Begin Color
 *      Header Gradient End Color
 * if style is available and not already set
 */

namespace Styles
{
    public class UModelStyles : UModelPlugInLib.IUModelPlugIn
    {
        bool m_bPlugInVersionOK = true; // verify if UModel-API has been changed in a way
that a recompile of this plug-in is recommended

        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        #region IUModelPlugIn Members

        public string GetDescription()
        {
```


© 2016-2022 Altova GmbH

```
            return "Styles sample Plug-in for UModel;This Plug-in demonstrates how to
change several styles of the selected diagram elements.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new System.IO.StreamReader(sPath + "\
\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "**path**" from the XML file with
                // the actual installation path of the plug-in to get the image file
                return sRet.Replace("**path**", sPath);
            }
            catch (System.Exception ex)
            {
                MessageBox.Show("Error in GetUIModifications:" + ex.Message);
                throw ex;
            }
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized

            // verify if UModel-API has been changed in a way that a recompile of this
plug-in is recommended:
            IApplication iApp = (IApplication)pUModel;
            if (iApp == null || iApp.APIMajorVersion != 5) // this plug-in was compiled
for API major version '5'!
            {
                MessageBox.Show("'Styles': This Plug-in has been made with a previous
version of the UModel-API and should be recompiled.\nDisabled Plug-in commands in the
meantime.");
                m_bPlugInVersionOK = false;
            }
        }

        public void OnShutdown(object pUModel)
        {
            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            UModelUpdateAction action = UModelUpdateAction.UModelUpdateAction_Disable;
            if (!m_bPlugInVersionOK)
                return action;
```

```
            // check for "fill red"
            if (nID == 3 || nID == 6)
                action = OnUpdateSetStyles((IApplication)pUModel);

            // check for "fill green"
            if (nID == 4 || nID == 7)
                action = OnUpdateSetStyles((IApplication)pUModel);

            // release unused objects
            GC.Collect();

            return action;
        }

        public void OnCommand(int nID, object pUModel)
        {
            if (!m_bPlugInVersionOK)
                return;

            // fill red
            if (nID == 3 || nID == 6)
                OnSetStyles((IApplication)pUModel, "red");

            // fill green
            if (nID == 4 || nID == 7)
                OnSetStyles((IApplication)pUModel, "green");

            // release unused objects
            GC.Collect();
        }

        #endregion

        #region SetStyles // set styles of selected diagram elements

        UModelUpdateAction OnUpdateSetStyles(IApplication pUModel)
        {
            if (pUModel == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if ( iActiveDiagram == null )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the selected elements on the active diagram
            IUMLDataList iSelection = iActiveDiagram.SelectedGuiElements;
            if ( iSelection == null )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // search all selected elements, if at least one has one of the styles to
change
            foreach ( IUMLGuiElement iSelGuiElement in iSelection )
```


© 2016-2022 Altova GmbH

```
            {
                // verify if it is a GuiVisibleElement (with Styles) and if it may be
modified
                if ( iSelGuiElement is IUMLGuiVisibleElement &&
iSelGuiElement.IsEditable )
                {
                    IUMLGuiVisibleElement iVisGuiElement = (IUMLGuiVisibleElement)
iSelGuiElement;

                    if
( iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_FillColor) != null ||
                        iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_
HeaderGradientBeginColor) != null ||
                        iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_
HeaderGradientEndColor) != null )
                    {
                        return UModelUpdateAction.UModelUpdateAction_Enable;
                    }
                }
            }

            // nothing found => disable command
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        public void OnSetStyles(IApplication pUModel, string sColor)
        {
            if (pUModel == null)
                return;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if (iActiveDiagram == null)
                return;

            // get the selected elements on the active diagram
            IUMLDataList iSelection = iActiveDiagram.SelectedGuiElements;
            if (iSelection == null)
                return;

            try
            {
                // make all modifications within one UndoStep; start modification here
                if (!iDoc.BeginModification())
                    return;

                // search all selected elements, and change the style if the wanted value
is not already used (directly applied or through style-chain)
                foreach (IUMLGuiElement iSelGuiElement in iSelection)
                {
                    // verify if it is a GuiVisibleElement (with Styles) and if it may be
modified
                     if (iSelGuiElement is IUMLGuiVisibleElement &&
iSelGuiElement.IsEditable )
```

```
                    {
                        IUMLGuiVisibleElement iVisGuiElement = (IUMLGuiVisibleElement)
iSelGuiElement;

                        // set Fill Color if possible and not already set
                        IUMLGuiStyle iStyle =
iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_FillColor);
                        if (iStyle != null && iStyle.UsedValue != sColor)
                            iStyle.Value = sColor;

                        // set Header Gradient Begin Color if possible and not already
set
                        iStyle =
iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientBeginColor)
;
                        if (iStyle != null && iStyle.UsedValue != sColor)
                            iStyle.Value = sColor;

                        // set Header Gradient End Color if possible and not already set
                        iStyle =
iVisGuiElement.Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientEndColor);
                        if (iStyle != null && iStyle.UsedValue != sColor)
                            iStyle.Value = sColor;
                    }
                }

                // do not forget to end modification and finish UndoStep
                iDoc.EndModification();
            }
            catch( System.Exception )
            {
                // rollback made changes
                iDoc.AbortModification();

                // add error handling
            }
        }

        #endregion
    }
}
```

17.4.4.4 Beispiel "C# Delegate"

Im folgenden C# IDE Plug-in Beispiel wird ein neuer C# Delegate in der oberen/linken Ecke des aktiven Diagrammfensters eingefügt (falls sich dieses Diagramm innerhalb einer C# Namespace Root befindet). Im Beispiel werden sowohl die UModel API als auch die UModel IDE Plug-In Library verwendet. Es steht in der folgenden Datei zur Verfügung: **..\UModelExamples\IDEPlugIn\CSharpDelegate\UModelCSharpDelegate.cs**.

Für das Erstellen und Ausführen des Beispiels gelten dieselben Voraussetzungen wie für andere UModel IDE Plug-ins, siehe Erstellen und Ausführen des Plug-in 846.

© 2016-2022 Altova GmbH

```
using System;
using System.Collections.Generic;
using System.Text;
using UModelLib;
using UModelPlugInLib;

/*
 * CSharp delegate sample
 * add a new CSharp delegate on the top/left corner of the active class diagram if
possible
 * (i.e. when diagram is inside a C# root namespace)
 */

namespace CSharpDelegate
{
    public class UModelCSharpDelegate : UModelPlugInLib.IUModelPlugIn
    {
        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "CSharpDelegate sample Plug-in for UModel;This Plug-in demonstrates
how to create a new CSharp delegate on a class diagram.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new System.IO.StreamReader(sPath + "\
\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "**path**" from the XML file with
                // the actual installation path of the plug-in to get the image file
                return sRet.Replace("**path**", sPath);
            }
            catch (System.Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("Error in GetUIModifications:" +
ex.Message);
                throw ex;
            }
        }
```

```
        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
        }

        public void OnShutdown(object pUModel)
        {
            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            UModelUpdateAction action = UModelUpdateAction.UModelUpdateAction_Disable;

            // check if we can add a new CSharpDelegate on the active diagram
            if (nID == 3 || nID == 4)
                action = OnUpdateAddNewCSharpDelegate((IApplication)pUModel);

            // release unused objects
            GC.Collect();

            return action;
        }

        public void OnCommand(int nID, object pUModel)
        {
            // add a new CSharpDelegate on the active diagram
            if (nID == 3 || nID == 4)
                OnAddNewCSharpDelegate((IApplication)pUModel);

            // release unused objects
            GC.Collect();
        }

        #endregion

        #region AddNewCSharpDelegate // add new CSharp delegate on active diagram

        UModelUpdateAction OnUpdateAddNewCSharpDelegate(IApplication pUModel)
        {
            if (pUModel == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if ( iActiveDiagram == null )
                return UModelUpdateAction.UModelUpdateAction_Disable;
```


© 2016-2022 Altova GmbH

```
            // get the UML diagram of the diagram window
            IUMLGuiDiagram iUMLDiagram = iActiveDiagram.Diagram;

            // check if it is a class diagram
            if ( !( iUMLDiagram is IUMLGuiClassDiagram) )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // verify if the diagram may be modified
            if ( !iUMLDiagram.IsEditable )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the UML element, which "owns" the class diagram
            IUMLElement iDiagramOwner = iUMLDiagram.LinkedOwner;
            if (iDiagramOwner == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // verify if we are inside a CSharp namespace root (otherwise adding a CSharp
delegate makes no sense)
            IUMLElement iFindNamespaceRoot = iDiagramOwner;
            while( iFindNamespaceRoot != null)
            {
                if ( iFindNamespaceRoot is IUMLPackage)
                {
                    IUMLPackage iPackage = (IUMLPackage) iFindNamespaceRoot;
                    if
( iPackage.IsCodeLangNamespaceRoot( ENUMCodeLang.eCodeLang_CSharp ) )
                        return UModelUpdateAction.UModelUpdateAction_Enable;
                }

                iFindNamespaceRoot = iFindNamespaceRoot.Owner;
            }

            // nothing found => disable command
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        public void OnAddNewCSharpDelegate(IApplication pUModel)
        {
            if (pUModel == null)
                return;

            // get the active document of the application
            IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return;

            // get the active diagram window
            IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if (iActiveDiagram == null)
                return;

            // get the UML diagram of the diagram window
            IUMLGuiDiagram iUMLDiagram = iActiveDiagram.Diagram;

            // get the CSharp profile
            IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc.RootPackage.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_P
rofile, false);
            if ( iCSharpProfile == null)
```

```
                return;

            try
            {
                // make all modifications within one UndoStep; start modification here
                if (!iDoc.BeginModification())
                    return;

                // get top left corner of the visible diagram area
                int nInsertPosX = iActiveDiagram.ScrollPosX;
                int nInsertPosY = iActiveDiagram.ScrollPosY;

                // add new class on diagram
                IUMLGuiNodeLink iClassNode = iUMLDiagram.AddUMLElement("Class",
nInsertPosX + 100, nInsertPosY + 100);

                IUMLClass iClass = (IUMLClass) iClassNode.Element;
                // use SetName (instead of Name) that UModel automatically generates a
valid, unique name starting with "NewDelegate"
                iClass.SetName("NewDelegate");

                // set the CSharp 'delegate' stereotype
                iClass.ApplyPredefinedStereotype(
 ENUMUMLPredefinedElement.ePredefined_CSharp_delegateStereotypeOfClass );

                // set attribute-section "STAThread"
                IUMLStereotypeApplication iStereotypeApp =
iClass.ApplyPredefinedStereotype(ENUMUMLPredefinedElement.ePredefined_CSharp_attributesSt
ereotypeOfClass);
                IUMLEnumerationLiteral iSTAThread = (IUMLEnumerationLiteral)
iCSharpProfile.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_Att
ributePresetsEnumeration_STAThreadEnumerationLiteral, true);
                iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
 iSTAThread.Name);

                // add delegate operation:
                IUMLOperation iOperation = iClass.InsertOwnedOperationAt(-1);
                iOperation.SetName( "delegate");

                // per default set operation-return type "void"
                IUMLPrimitiveType iTypeVoid = (IUMLPrimitiveType)
iCSharpProfile.FindPredefinedOwnedElement(ENUMUMLPredefinedElement.ePredefined_CSharp_voi
dPrimitiveType, true);
                iOperation.Type = iTypeVoid;

                // do not forget to end modification and finish UndoStep
                iDoc.EndModification();

                // at last focus newly inserted delegate on the diagram:
                iActiveDiagram.SelectGuiElement(iClassNode, true);
            }
            catch( System.Exception )
            {
                // rollback made changes
                iDoc.AbortModification();

                // add error handling
            }
```


© 2016-2022 Altova GmbH

```
        }

        #endregion
    }
}
```

17.4.4.5 Beispiel "Präfix definieren"

Im folgenden Beispiel wird automatisch ein Präfix definiert, wenn neue Attribute oder Enumerationliterale zu Ihrem UModel-Projekt hinzugefügt werden. Im Beispiel werden sowohl die UModel API als auch die UModel IDE Plug-In Library verwendet. Es steht in der folgenden Datei zur Verfügung: **....\UModelExamples\IDEPlugIn\DefaultPrefix\DefaultPrefix.cs**.

Für das Erstellen und Ausführen des Beispiels gelten dieselben Voraussetzungen wie für andere UModel IDE Plug-ins, siehe Erstellen und Ausführen des Plug-in 846.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Runtime.InteropServices.ComTypes;
using UModelLib;
using UModelPlugInLib;

/*
 * DefaultPrefix sample
 * listen for newly added UML data and
 * set the prefix of properties ('m_') and EnumerationLiterals ('k_')
 * if the corresponding option is turned on
 */

namespace DefaultPrefix
{
    /* UModelDefaultPrefix is the main class of this plugin and implements
UModelPlugInLib.IUModelPlugIn
     * it is also responsible for attaching/detaching UModelApplicationEvents to/from
UModels IApplication interface
     * and implements the handling of turning on/off the whole "SetPrefix" functionality
     */
    public class UModelDefaultPrefix : UModelPlugInLib.IUModelPlugIn
    {
        // variable which defines whether "SetPrefix" functionality is turned on or off
        bool m_bSetPrefix = true;

        // reference to UModelApplicationEvents; is only used when "SetPrefix"
functionality is turned on (to reduce overhead in the other case)
        UModelApplicationEvents m_AppEvents = null;

        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
```

```
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        // create UModelApplicationEvents and attach it to IApplication
        protected void AttachAppEvents( IApplication iUModelApp )
        {
            if (m_AppEvents == null && iUModelApp != null)
            {
                m_AppEvents = new UModelApplicationEvents();
                m_AppEvents.Attach(iUModelApp);
            }
        }

        // detach UModelApplicationEvents;
        protected void DetachAppEvents()
        {
            if (m_AppEvents != null)
            {
                m_AppEvents.Detach();
                m_AppEvents = null;
            }
        }

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "DefaultPrefix sample Plug-in for UModel;This Plug-in demonstrates how
to attach to several callback interfaces and how to add a prefix to newly inserted
elements.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new System.IO.StreamReader(sPath + "\
\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "**path**" from the XML file with
                // the actual installation path of the plug-in to get the image file
                return sRet.Replace("**path**", sPath);
            }
            catch (System.Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("Error in GetUIModifications:" +
ex.Message);
                throw ex;
            }
        }

        public void OnInitialize(object pUModel)
        {
```


© 2016-2022 Altova GmbH

```
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
            // and we can attach UModelApplicationEvents
            AttachAppEvents( (IApplication)pUModel );
        }

        public void OnShutdown(object pUModel)
        {
            // detach UModelApplicationEvents; stop receiving events
            DetachAppEvents();

            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            UModelUpdateAction action = UModelUpdateAction.UModelUpdateAction_Disable;

            // check if automatically setting the prefix is turned on:
            if (nID == 3 || nID == 4)
            {
                action = UModelUpdateAction.UModelUpdateAction_Enable;

                if (m_bSetPrefix)
                    action |= UModelUpdateAction.UModelUpdateAction_Check;
            }

            // release unused objects
            //GC.Collect(); not necessary since we do not access objects here

            return action;
        }

        public void OnCommand(int nID, object pUModel)
        {
            // toggle automatically setting the prefix:
            if (nID == 3 || nID == 4)
                m_bSetPrefix = !m_bSetPrefix;

            // attach UModelApplicationEvents when "SetPrefix" functionality is turned
on; detach otherwise
            if (m_bSetPrefix)
                AttachAppEvents((IApplication)pUModel);
            else
                DetachAppEvents();

            // release unused objects
            GC.Collect();
        }

        #endregion
    }
```

```
    /* UModelApplicationEvents is an eventhandler to receive _IApplicationEvents
     * that we know when UModel documents are opened or closed
     * and that we can Attach/Detach UModelDataEvents
     * We are interested in all _IApplicationEvents and use a connectionpoint to connect
to all these events
     */
    public class UModelApplicationEvents : UModelLib._IApplicationEvents
    {
        // connection point to _IApplicationEvents
        System.Runtime.InteropServices.ComTypes.IConnectionPoint m_cpApplicationEvents =
null;
        // connection cookie
        int m_nApplicationtEventsCookie = 0;
        // we always hold a reference to UModelDataEvents
        UModelDataEvents m_UMLDataEvents = new UModelDataEvents();

        public void Attach(IApplication iApp)
        {
            if (m_cpApplicationEvents == null && iApp != null)
            {
                // find connection point of _IApplicationEvents
                IConnectionPointContainer icpc = (IConnectionPointContainer)iApp;
                Guid IID = typeof(UModelLib._IApplicationEvents).GUID;
                icpc.FindConnectionPoint(ref IID, out m_cpApplicationEvents);

                // advise UModelApplicationEvents as sink for _IApplicationEvents
                m_cpApplicationEvents.Advise(this, out m_nApplicationtEventsCookie);

                // also attach UModelDataEvents to the current document and start
receiving events there
                m_UMLDataEvents.Attach(iApp.ActiveDocument);
            }
        }
        public void Detach()
        {
            if (m_cpApplicationEvents != null)
            {
                // also detach UModelDataEvents and stop receiving events there
                m_UMLDataEvents.Detach();

                // terminate established connection to _IApplicationEvents
                m_cpApplicationEvents.Unadvise(m_nApplicationtEventsCookie);
                m_cpApplicationEvents = null;
            }
        }

        #region _IApplicationEvents Members
        public void OnNewDocument(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnNewDocument " + ipDocument.Name);
            // a new document has been created in UModel => (re-)connect UModelDataEvents
            m_UMLDataEvents.Attach(ipDocument);
        }

        public void OnDocumentOpened(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnDocumentOpened " +
ipDocument.Name);
            // a document has been opened in UModel => (re-)connect UModelDataEvents
```


© 2016-2022 Altova GmbH

```
            m_UMLDataEvents.Attach(ipDocument);
        }

        public void OnDocumentClosed(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnDocumentClosed " +
ipDocument.Name);
            // document has been closed in UModel => disconnect UModelDataEvents
            m_UMLDataEvents.Detach();
        }

        public void OnShutdown()
        {
            Debug.WriteLine("UModelApplicationEvents.OnShutdown");
        }

        #endregion
    }


    /* UModelDataEvents is an eventhandler to receive _IUMLDataEvents
     * from the root-package and all its children.
     * We are only interested in 'OnAfterAddChild' events, so we use a delegate to
connect to this event.
     */
    public class UModelDataEvents : UModelLib._IUMLDataEvents
    {
        // hold a reference to the current UML Root package; this is safe as long as we
listen to when it is deleted
        protected UMLData m_RootPackage = null;

        // attach this eventhandler to the root-package of the (current) document
        public void Attach(IDocument iDoc)
        {
            if (m_RootPackage == null && iDoc != null && iDoc.RootPackage != null)
            {
                // hold a reference to the current UML Root package
                m_RootPackage = (UMLData)iDoc.RootPackage;

                // ensure we get 'OnAfterAddChild' events for *any* added child of the
rootpackage
                // (added to the root-package or one of its children)
                m_RootPackage.EventFilter = (int)
ENUMUMLDataEventFilter.eUMLDataEvent_AddChildOrGrandChild;
                // ensure we get informed when m_RootPackage (and only itself; we do not
care about its chidlren) is deleted
                m_RootPackage.EventFilter |= (int)
ENUMUMLDataEventFilter.eUMLDataEvent_EraseData;

                // we are only interested in 'OnAfterAddChild' and 'OnBeforeErase' events
so use and connect the delegates
                m_RootPackage.OnAfterAddChild += new
_IUMLDataEvents_OnAfterAddChildEventHandler(OnAfterAddChild);
                m_RootPackage.OnBeforeErase += new
_IUMLDataEvents_OnBeforeEraseEventHandler(OnBeforeErase);
            }
        }

        // detach eventhandler from the current UML Root package
```

```
        public void Detach()
        {
            if (m_RootPackage != null)
            {
                m_RootPackage.OnAfterAddChild -= OnAfterAddChild;
                m_RootPackage.OnBeforeErase -= OnBeforeErase;
                m_RootPackage = null;

                // release unused objects
                GC.Collect();
            }
        }

        #region _IUMLDataEvents Members

        public void OnAfterAddChild(IUMLData ipUMLParent, IUMLData ipUMLChild)
        {
            if (ipUMLParent == null || ipUMLChild == null)
                return;

            Debug.WriteLine("UModelDataEvents.OnAfterAddChild " + GetName(ipUMLChild) + "
to " + GetName(ipUMLParent));

           // verify if newly added child is of interesting kind:
            bool bIsEnumerationLiteral = (ipUMLChild is IUMLEnumerationLiteral);
            bool bIsProperty = (ipUMLChild is IUMLProperty);

            if (bIsProperty || bIsEnumerationLiteral)
            {
                try
                {
                    // check if child was added by undo/redo
                    // (we are not allowed to modify anything during Undo/Redo !!)
                    IDocument iDoc = (IDocument)ipUMLChild.Parent;
                    if (!iDoc.IsInUndoRedo)
                    {
                        // we only make one single modification here
                        // no need to use iDoc.BeginModification / iDoc.EndModification
in this case

                        // get the wanted prefix for the element kind
                        string sPrefix = null;

                        if (bIsProperty)
                            sPrefix = "m_";
                        if (bIsEnumerationLiteral)
                            sPrefix = "k_";

                        IUMLNamedElement iNamedChild = (IUMLNamedElement)ipUMLChild;

                        // set prefix only if not already set:
                        if (sPrefix != null && !iNamedChild.Name.StartsWith(sPrefix))
                        {
                            // use SetName (instead of Name) that UModel automatically
generates a valid, unique name starting with 'sPrefix + iNamedChild.Name'
                            iNamedChild.SetName(sPrefix + iNamedChild.Name);
                        }
                    }
                }
```


© 2016-2022 Altova GmbH

```
                catch (System.Exception e)
                {
                    Debug.WriteLine("EXCEPTION: " + e.Message);
                }
            }

            // release unused objects
            GC.Collect();
        }

        public void OnBeforeErase(IUMLData ipUMLData)
        {
            if (ipUMLData != null && m_RootPackage != null &&
ipUMLData.IsSameUMLData((IUMLData)m_RootPackage)) // should always be
            {
                // Detach ourself, since the UML data of m_RootPackage has been deleted
in UModel and we may not access it anymore
                Detach();
            }
        }

        public void OnChanged(IUMLData ipUMLData, string strHint)
        {
            // unused
        }

        public void OnMoveData(IUMLData ipUMLParent, IUMLData ipUMLChild, bool bAttach)
        {
            // unused
        }

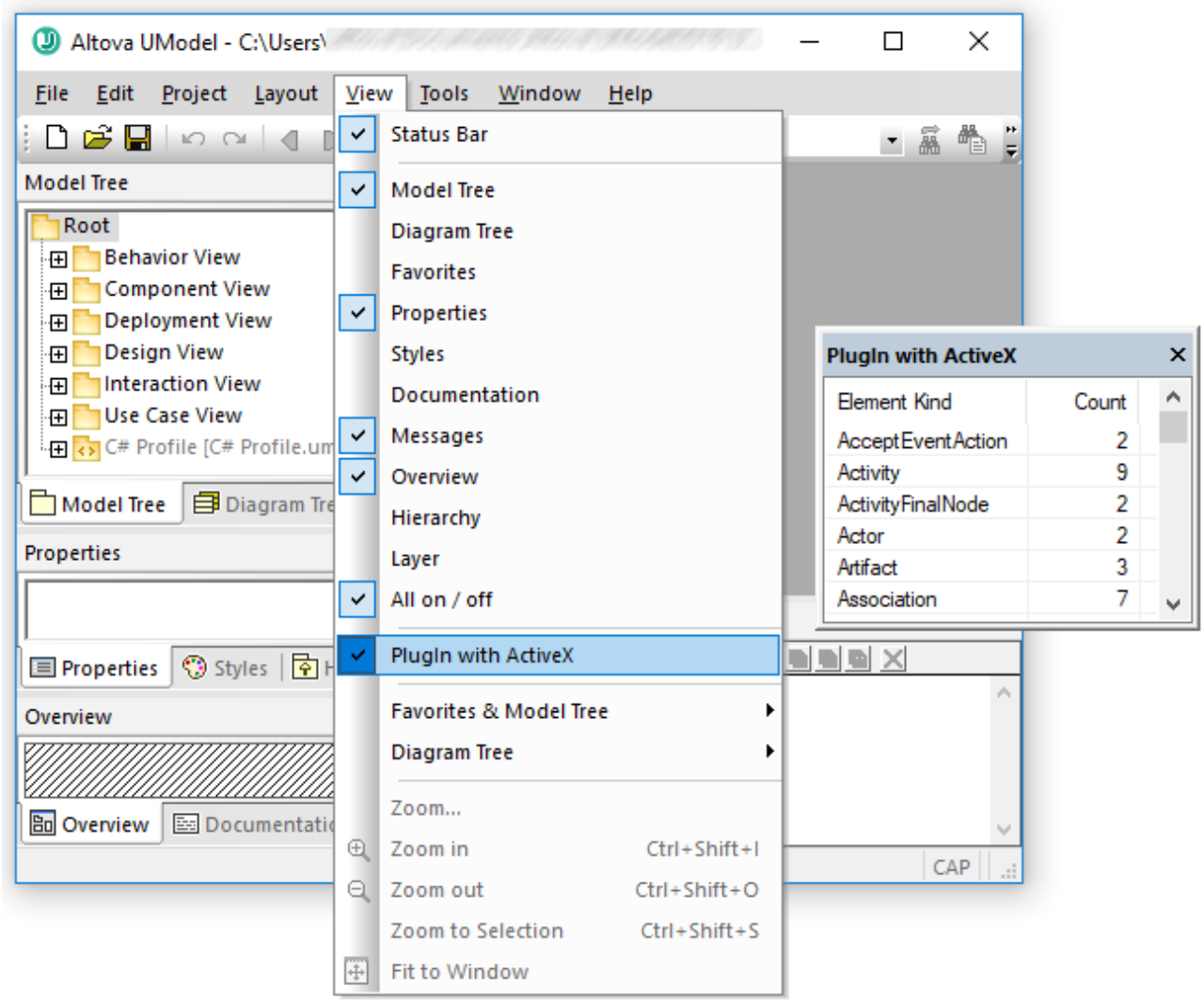
        #endregion

        protected string GetName(IUMLData iUMLData)
        {
            if (iUMLData is IUMLNamedElement)
                return ((IUMLNamedElement)iUMLData).Name;

            return "";
        }
    }
}
```

17.4.4.6 Beispiel "Statistik"

Das Beispiel "Statistik" sucht nach Datenänderungen und zählt Elemente verschiedener Elementarten. Im Beispiel werden sowoh die UModel API als auch die UModel IDE Plug-In-Bibliothek verwendet. Da das Plug-in von `System.Windows.Forms.UserControl` abgeleitet wird, fungiert es auch als ActiveX Control und die Ergebnisse können in UModel in einem benutzerdefinierten Fenster angezeigt werden:



Dieser Code steht in der folgenden Datei zur Verfügung: **..\UModelExamples\IDEPlugIn\StatisticsActiveX\StatisticsActiveX.cs**.

Für das Erstellen und Ausführen des Beispiels gelten dieselben Voraussetzungen wie für andere UModel IDE Plug-ins, siehe Erstellen und Ausführen des Plug-in 846.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Runtime.InteropServices.ComTypes;
using System.Windows.Forms;
using UModelLib;
using UModelPlugInLib;

/*
```

 © 2016-2022 Altova GmbH

```
 * StatisticsActiveX sample
 * listen for data modifications and count the elements of the different element kinds
 * show the result in a listview of an ActiveX control
 */
namespace StatisticsActiveX
{
    /* StatisticsActiveX is the main class of this plugin and implements
UModelPlugInLib.IUModelPlugIn
     * it is also responsible for attaching/detaching _IApplicationEvents and
_ITransactionEvents
     */
    public partial class StatisticsActiveX : UserControl,
                                             IUModelPlugIn,
                                            _IApplicationEvents,
                                            _ITransactionEvents
    {
        // a sorted dictionary to count the different element kinds
        private Statistics m_Statistics;
        // reference to the transaction notifier of a UModel document
        private TransactionNotifier m_TransactionNotifier;
        // connection point to _IApplicationEvents
        private IConnectionPoint m_cpApplicationEvents = null;
        // connection cookie
        int m_nApplicationtEventsCookie = 0;

        public StatisticsActiveX()
        {
            InitializeComponent();
        }

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "PlugIn with ActiveX;This Plug-in demonstrates how to show an ActiveX
control inside UModel.";
        }

        public string GetUIModifications()
        {
            // We don't add any menu or toolbar modifications.
            return "<ConfigurationData><Modifications/></ConfigurationData>";
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is fully initialized
            // and we can attach to get _IApplicationEvents

            IApplication iApp = (IApplication)pUModel;

            if (m_cpApplicationEvents == null && iApp != null)
            {
                // find connection point of _IApplicationEvents
```

```
                IConnectionPointContainer icpc = (IConnectionPointContainer)iApp;
                Guid IID = typeof(UModelLib._IApplicationEvents).GUID;
                icpc.FindConnectionPoint(ref IID, out m_cpApplicationEvents);

                // advise UModelApplicationEvents as sink for _IApplicationEvents
                m_cpApplicationEvents.Advise(this, out m_nApplicationtEventsCookie);
            }

            AttachTransactionEvents(iApp.ActiveDocument);
        }

        public void OnShutdown(object pUModel)
        {
            // detach application events; stop receiving events
            DetachTransactionEvents();

            if (m_cpApplicationEvents != null)
            {
                // terminate established connection to _IApplicationEvents
                m_cpApplicationEvents.Unadvise(m_nApplicationtEventsCookie);
                m_cpApplicationEvents = null;
            }

            // application will shutdown; release all unused objects
            GC.Collect();
        }

        public void OnCommand(int nID, object pUModel)
        {
            // unused; we did not add any menu- or toolbar-commands
        }

        public UModelUpdateAction OnUpdateCommand(int nID, object pUModel)
        {
            // unused; we did not add any menu- or toolbar-commands
            return UModelUpdateAction.UModelUpdateAction_Disable;
        }

        #endregion

        private void AttachTransactionEvents(IDocument iDoc)
        {
            if (iDoc != null)
            {
                m_TransactionNotifier = iDoc.TransactionNotifier;
                if (m_TransactionNotifier != null)
                {
                    // we are only interested in 'OnEndDataModification' events so use
and connect the delegate
                    m_TransactionNotifier.OnEndDataModification += new
_ITransactionEvents_OnEndDataModificationEventHandler(OnEndDataModification);
                }
            }

            UpdateStatistics(iDoc);
        }

        // detach eventhandler from the transaction notifier
        private void DetachTransactionEvents()
```


© 2016-2022 Altova GmbH

```
        {
            if (m_TransactionNotifier != null)
            {
                m_TransactionNotifier.OnEndDataModification -= OnEndDataModification;
                m_TransactionNotifier = null;
            }
            UpdateStatistics(null);
        }

        void UpdateStatistics(IDocument iDoc)
        {
            // count current elements
            Statistics statistics = new Statistics();

            if (iDoc != null && iDoc.RootPackage != null)
                CountElements(iDoc.RootPackage, ref statistics);

            // anything changed to last update ?
            if (!statistics.IsEqual(m_Statistics))
            {
                m_Statistics = statistics;
                PopulateListView(m_Statistics);
            }

            // release unused objects
            GC.Collect();
        }

        private void CountElements(IUMLElement iElem, ref Statistics statistics)
        {
            // we only count editable elements
            if (iElem == null || iElem.IsEditable == false)
                return;

            string sKindName = iElem.KindName;

            if (!statistics.ContainsKey(sKindName))
                statistics[sKindName] = 1;
            else
                statistics[sKindName]++;

            foreach (IUMLElement iChild in iElem.OwnedElements)
                CountElements(iChild, ref statistics);
        }

        private void PopulateListView(Statistics statistics)
        {
            listView1.BeginUpdate();

            listView1.Items.Clear();
            foreach (KeyValuePair<string, int> kvp in statistics)
            {
                ListViewItem item = new ListViewItem(kvp.Key);
                item.SubItems.Add(Convert.ToString(kvp.Value));

                listView1.Items.Add(item);
            }

            listView1.EndUpdate();
```

```
        }

        #region _ITransactionEvents Members

        public void OnBeginDataModification(Document ipDocument)
        {
            // begin of transaction
        }

        public void OnEndDataModification(Document ipDocument)
        {
            // end of transaction - update statistics
            if (ipDocument != null && ipDocument.TransactionNotifier ==
m_TransactionNotifier)
                UpdateStatistics(ipDocument);
        }

        #endregion

        #region _IApplicationEvents Members

        public void OnNewDocument(Document ipDocument)
        {
            // a new document has been created in UModel => (re-)connect transaction
events
            AttachTransactionEvents(ipDocument);
        }

        public void OnDocumentOpened(Document ipDocument)
        {
            // a document has been opened in UModel => (re-)connect transaction events
            AttachTransactionEvents(ipDocument);
        }

        public void OnDocumentClosed(Document ipDocument)
        {
            // document has been closed in UModel => disconnect transaction events
            if (ipDocument != null && ipDocument.TransactionNotifier ==
m_TransactionNotifier)
                DetachTransactionEvents();
        }

        public void OnShutdown()
        {
        }

        #endregion


        #region Statistics dictionary

        private class Statistics : SortedDictionary<string, int>
        {
            public bool IsEqual(Statistics other)
            {
                if (other == null)
                    return false;

                if (Count != other.Count)
```


© 2016-2022 Altova GmbH

```
                    return false;

                Enumerator e1 = GetEnumerator();
                Enumerator e2 = other.GetEnumerator();
                while (e1.MoveNext() && e2.MoveNext())
                {
                    if ((e1.Current.Key != e2.Current.Key) ||
                        (e1.Current.Value != e2.Current.Value))
                        return false;
                }

                return true;
            }
        }

        #endregion
    }
}
```

17.4.5 Java-Beispielprojekt

Das UModel-Installationspaket enthält ein Java-Beispielprojekt, das Sie unter dem folgenden Pfad finden: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\API**. Dieser Ordner enthält Java-Beispiele für die UModel API. Sie können das Beispielprojekt mit Hilfe der Batch-Datei `BuildAndRun.bat`, direkt über die Befehlszeile testen oder Sie können es in Eclipse kompilieren und ausführen. Anleitungen dafür finden Sie weiter unten.
Der Ordner für die Java-Beispiele enthält alle zum Ausführen des Beispielprojekts erforderlichen Dateien. Diese Dateien sind unten aufgelistet:

AltovaAutomation.dll	Java-COM Bridge: DLL-Teil
AltovaAutomation.jar	Java-COM Bridge: Java-Bibliotheksteil
UModelAPI.jar	Java-Klassen der MapForce API
RunUModel.java	Java-Beispielquellcode
BuildAndRun.bat	Batch-Datei zum Kompilieren und Ausführen des Beispielcodes über die Befehlszeile. Es wird ein Ordner benötigt, in dem sich die Java Virtual Machine als Parameter befindet.
.classpath	Hilfedatei Eclipse-Projekt
.project	Eclipse-Projektdatei
UModelAPI_JavaDoc.zip	`Javadoc` Datei, die die Hilfedokumentation für die Java API enthält
Readme.txt	Diese Datei

In diesem Beispielprojekt wird UModel gestartet und einige Operationen wie das Öffnen und Schließen von Dokumenten werden ausgeführt. UModel bleibt danach geöffnet. Sie müssen die Applikation manuell schließen.

Ausführen des Beispiels über die Befehlszeile

Um das Beispiel von der Befehlszeile aus auszuführen, öffnen Sie ein Eingabeaufforderungsfester, gehen Sie zum Ordner Java des Ordners API Examples (*Pfad siehe oben*) und geben Sie folgende Zeile ein:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

Der Java Binary-Ordner muss von einer JDK 1.5 oder höheren Version auf Ihrem Rechner sein.

Drücken Sie die **Eingabetaste.** Der Java-Quellcode in `RunUModel.java` wird kompiliert und anschließend ausgeführt.

Laden des Beispiels in Eclipse

Öffnen Sie Eclipse und wählen Sie den Befehl **Import | Existing Projects into Workspace** um die Eclipse-Projektdatei (`.project`) im Ordner Java des Ordners API Examples (*Pfad siehe oben*) zu Eclipse hinzuzufügen. Daraufhin wird das Projekt `RunUModel` in Ihrem Package Explorer oder Navigator angezeigt.

Wählen Sie das Projekt aus und klicken Sie anschließend auf **Run as | Java Application** um das Beispiel auszuführen.

Anmerkung: Sie können einen Klassennamen oder eine Methode der Java API auswählen und F1 drücken, um Hilfe zu dieser Klasse oder Methode zu erhalten.

17.4.6 JScript-Beispiele

Dieser Abschnitt enthält eine Liste von JScript-Code, anhand dessen die folgenden grundlegenden Funktionalitäten erläutert werden:

- Starten der Applikation 905
- Dokumentaufruf 905
- Dokumentation generieren 907
- Code generieren 908
- Dokumentation aktualisieren 912

Beispieldateien

Sie finden den Code in den Beispieldateien, die Sie je nach Bedarf testen und modifizieren können. Die JScript-Beispieldateien befinden sich unter: **C:\Benutzer\<Benutzername>\Dokumente\Altova\UModel2023\UModelExamples\API**.

Die Beispieldateien können auf zwei Arten ausgeführt werden:

- *Über die Befehlszeile:* Öffnen Sie ein Eingabeaufforderungsfenster ändern Sie den Verzeichnispfad zum oben angegebenen und geben Sie den Namen eines der Beispiel-Skripts ein (z.B. `Start.js`).
- *Über den Windows Explorer:* Navigieren Sie im Windows Explorer zur JScript-Datei und doppelklicken Sie darauf.

© 2016-2022 Altova GmbH

Das Skript wird vom mit Windows-Betriebssystemen bereitgestellten Windows Script Host ausgeführt. Nähere Informationen zum Windows Script Host finden Sie in der MSDN-Dokumentation (https://msdn.microsoft.com).

17.4.6.1 Applikation starten

Mit dem unten aufgeführten JScript-Code wird die Applikation gestartet und beendet. Wenn bereits eine Instanz der Applikation ausgeführt wird, so wird diese Instanz aufgerufen.

Anmerkung: Für 32-Bit-Versionen von UModel ist der registrierte Name oder programmatische Identifier (ProgId) des COM-Objekts `UModel.Application`. Für 64-Bit-Versionen von UModel ist der Name `UModel_x64.Application`.

Dieser Code steht in der Beispieldatei `..\UModelExamples\API\JScript\Start.js` zur Verfügung (siehe auch Beispieldateien 904).

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objUModel = WScript.GetObject("", "UModel.Application");   }
catch(err) {}

if( typeof( objUModel ) == "undefined" )
{
   try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create UModel.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objUModel.Visible = true;

WScript.Echo(objUModel.Edition + " has successfully started. ");

objUModel.Visible = false; // will shutdown application if it has no more COM connections
//objUModel.Visible = true;   // will keep application running with UI visible
```

17.4.6.2 Dokumentaufruf

Im unten aufgelisteten JScript-Code wird gezeigt, wie man Dokumente öffnet, ein Dokument zum aktiven Dokument macht, durch die offenen Dokumente iteriert und Dokumente schließt.

Dieser Code steht in der Beispieldatei `..\UModelExamples\API\JScript\DocumentAccess.js` zur Verfügung (siehe auch Beispieldateien 904).

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objUModel = WScript.GetObject("", "UModel.Application");   }
catch(err) {}

if( typeof( objUModel ) == "undefined" )
{
   try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create UModel.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objUModel.Visible = true;

// *************************** code snippet for "Simple Document Access"
***********************

// Locate examples using property PersonalDataDirectory
objDoc = objUModel.OpenDocument(objUModel.PersonalDataDirectory + "\\UModelExamples\
\Bank_MultiLanguage.ump");
// open all diagrams
objDoc.OpenAllDiagrams();

// *************************** code snippet for "Simple Document Access"
***********************

// *************************** code snippet for "Iteration"
***********************************

objName = "";
count = 0;
// go through all open diagrams using a JScript Enumerator
for (var iterDiagrams = new Enumerator(objDoc.DiagramWindows); !iterDiagrams.atEnd();
iterDiagrams.moveNext())
{
   objName += "\t" + ++count + " " + iterDiagrams.item().Name + "\n";
}

WScript.Echo("Opened diagrams: \n" + objName);

// go through all open diagrams using index-based access to the document collection
for (i = objDoc.DiagramWindows.Count; i > 0; i--)
   objDoc.DiagramWindows.Item(i).Close();


// *************************** code snippet for "Iteration"
***********************************

//objUModel.Visible = false;      // will shutdown application if it has no more COM
connections
objUModel.Visible = true;   // will keep application running with UI visible
```


© 2016-2022 Altova GmbH

17.4.6.3 Dokumentation generieren

Im unten aufgelisteten JScript-Code wird gezeigt, wie man Dokumentation für die Datei **Bank_MultiLanguage.ump** im Ordner UModelExamples generiert.

Dieser Code steht in der Beispieldatei `..\UModelExamples\API\JScript\GenerateDoc.js` zur Verfügung (siehe auch Beispieldateien 904).

```
// Initialize application's COM object. This will start a new instance of the application
and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try {   objUModel = WScript.GetObject("", "UModel.Application");   }
catch(err) {}

if( typeof( objUModel ) == "undefined" )
{
   try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
   catch(err)
   {
      WScript.Echo( "Can't access or create UModel.Application" );
      WScript.Quit();
   }
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objUModel.Visible = true;

// Locate examples via USERPROFILE shell variable.
objWshShell = WScript.CreateObject("WScript.Shell");
majorVersionYear = objUModel.MajorVersion + 1998
strExamplesFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\Documents\
\Altova\\UModel" + majorVersionYear + "\\UModelExamples\\";

objDoc = objUModel.OpenDocument(strExamplesFolder + "Bank_MultiLanguage.ump");

// generate documentation
dlgs = objUModel.Dialogs;
docDlg = dlgs.GenerateDocumentationDlg;
docDlg.OutputFormat = 0; // ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML

var myObject = new ActiveXObject("Scripting.FileSystemObject");
strDocOutputFolder = strExamplesFolder + "GeneraredDocFromJScriptExample\\";

if (!myObject.FolderExists(strDocOutputFolder))
   myObject.CreateFolder(strDocOutputFolder);

strResultFile = strDocOutputFolder + "Bank_MultiLanguage.html";
objDoc.generateDocumentation(docDlg, strResultFile);
```

```
//objUModel.Visible = false;      // will shutdown application if it has no more COM
connections
objUModel.Visible = true;   // will keep application running with UI visible
```

17.4.6.4 Code generieren

Im unten aufgelisteten JScript-Code wird ein neues UModel-Projekt erstellt, einige Klassen erstellt und Code generiert.

Dieser Code steht in der Beispieldatei `..\UModelExamples\API\JScript\UModelCreateCode.js` zur Verfügung (siehe auch Beispieldateien 904).

```
// ##########################################
// access runing UModel.Application or
// launch new one and access it
// ##########################################


// ##########################################
// CreateCode sample
// shows forward engineering from scratch
// it creates some coding elements in a new UModel project and generates code (saving the
project afterwards)
// ##########################################



// ############ global variables ################
var objUModel = null;
var objWshShell = null;
var objFSO = null;

// ###################### Helpers ############################

function Exit(strErrorText)
{
   WScript.Echo(strErrorText);

   if (objUModel != null)
      objUModel.Quit();

   WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
   // the Shell and FileSystemObject of the windows scripting host often always useful
   try
   {
      objWshShell = WScript.CreateObject("WScript.Shell");
      objFSO = WScript.CreateObject("Scripting.FileSystemObject");
   }
   catch(err)
   {
```


© 2016-2022 Altova GmbH

```
      Exit("Can't create WScript.Shell object");
   }

   // create the UModel connection
   // if there is a running instance of UModel (that never had a connection) - use it
   // otherwise, we automatically create a new instance
   try {   objUModel = WScript.GetObject("", "UModel.Application");   }
   catch(err) {}

   if( typeof( objUModel ) == "undefined" )
   {
      try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
      catch(err)
      {
         objUModel = null;
         Exit( "Can't access or create UModel.Application" );
      }
   }
}

function GetSourceCodeDirectory()
{
   // get directory for source code
   var path =  objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\
\CreateCode";
   var codeDirectory = objFSO.BuildPath( path, "SampleCode" );
   return codeDirectory;
}

function GetUMPFilePath()
{
   // get file path to save UModel projectfile
   return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\\CreateCode\
\CreateCode.ump";
}

function IncludeCSharpProfile( objDocument )
{
   try
   {
      // get dialog for including subprojects:
      var objIncludeSubProjectDialog = objUModel.Dialogs.IncludeSubprojectDlg;

      objIncludeSubProjectDialog.ProjectFile = objUModel.InstallationDirectory + "\
\UModelInclude\\c# Profile.ump";

      return objDocument.IncludeSubproject( objIncludeSubProjectDialog );
   }
   catch(err)
   {
      Exit("Can't include CSharp profile");
   }
}

// ////////////////////// MAIN ////////////////////////////

CreateGlobalObjects();

objUModel.Visible = true;
```

```
// open a new, empty document
var objDocument = objUModel.NewDocument();
// get the root-package
var objRootPackage = objDocument.RootPackage;

if ( objDocument    != null &&
     objRootPackage != null &&
    IncludeCSharpProfile( objDocument ) )
{
   // create coding elements
    try
    {
        // make all modifications within one UndoStep; start modification here
      if ( !objDocument.BeginModification() )
         Exit("No modifications allowed");

      // create a namespace root package
      var objCSharpRootNamespace = objRootPackage.InsertPackagedElementAt( -1,
"Package" );
      objCSharpRootNamespace.SetName( "CSharp" );

      // find C# Profile...
      var objCSharpProfile = objRootPackage.FindPredefinedOwnedElement( 159, false );//
ePredefined_CSharp_Profile = 159,
      // ...and apply it to the package, which is now a CSharp namespace root
      objCSharpRootNamespace.InsertProfileApplicationAt( -1, objCSharpProfile );


      // create a C# namespace package...
      var objCSharpNamespace = objCSharpRootNamespace.InsertPackagedElementAt( -1,
"Package" );
      objCSharpNamespace.SetName( "Namespace1" );
      // ... and apply the predefined C# namespace stereotype
      objCSharpNamespace.ApplyPredefinedStereotype( 223 ); //
ePredefined_CSharp_namespaceStereotypeOfPackage = 223,

      // create new class within the C# namespace
      var objClass     = objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
      var objClass2    = objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
      var objBaseClass = objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
      objClass    .SetName( "MyClass"     );
      objClass2   .SetName( "MyClass2"    );
      objBaseClass.SetName( "MyBaseClass" );

      // set attribute-section "STAThread"
      var objAttributesStereotypeApplication =
objClass.ApplyPredefinedStereotype( 191 );//
ePredefined_CSharp_attributesStereotypeOfClass = 191
      var objSTAThread = objCSharpProfile.FindPredefinedOwnedElement( 185, true ); //
ePredefined_CSharp_AttributePresetsEnumeration_STAThreadEnumerationLiteral = 185
      objAttributesStereotypeApplication.SetPredefinedTaggedValueAt(-1, 192,
objSTAThread.Name); // ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty =
192

      // insert new attribute
      var objProperty = objClass.InsertOwnedAttributeAt( -1 );
      objProperty.SetName( "m_Att" );
      objProperty.Visibility = 2;      // eVisibility_Private = 2
```


© 2016-2022 Altova GmbH

```
      objProperty.Type = objClass2;

      // insert new operation
      var objOperation = objClass.InsertOwnedOperationAt( -1 );
      objOperation.SetName( "GetAtt" );
      objOperation.Type = objClass2;

      // derive MyClass from MyBaseClass
      objClass.InsertGeneralizationAt( -1, objBaseClass );


      // find the component view package
      var objComponentView = objRootPackage.FindPredefinedOwnedElement( 1, false );//
ePredefined_ComponentViewPackage = 1

      // create a new component for C# 3.0 and set the source code directory, where we
want to generate the source code
      var objComponent = objComponentView.InsertPackagedElementAt( -1, "Component" );
      objComponent.CodeLangVersion= 5; //    eCodeLang_CSharp_3_0   = 5,
      objComponent.CodeProjectFileOrDirectory = GetSourceCodeDirectory();
      objComponent.IsCodeProjectFile = false;

      // this component should realize our classes:
      objComponent.InsertRealizationAt( -1, objClass     );
      objComponent.InsertRealizationAt( -1, objClass2    );
      objComponent.InsertRealizationAt( -1, objBaseClass );

        // do not forget to end modification and finish UndoStep
      objDocument.EndModification();
    }
   catch( err )
   {
        // rollback made changes
      objDocument.AbortModification();
      Exit("Error when creating UML model elements");
   }

   // update code from model
   try
   {
      // explicitely run a syntax check
      if ( objDocument.CheckProjectSyntax() )
      {
         // get dialog for code <=> model synchonizations and set the wanted options:
         var objSynchronizationSettingsDlg =
objUModel.Dialogs.SynchronizationSettingsDlg;

         objSynchronizationSettingsDlg.CodeFromModel_Synchronization = 0; //
eSynchronization_Merge = 0
         objSynchronizationSettingsDlg.CodeFromModel_UserDefinedSPLTemplatesOverrideDefau
lt = true;

         // update code from model
         if ( !objDocument.SynchronizeCodeFromModel( objSynchronizationSettingsDlg ) )
            Exit("Update code from model failed");
      }
      else
         Exit("Syntax check failed");
   }
```

```
   catch( err )
   {
      Exit("Error when updating code from model");
   }

   // save project
   objDocument.SaveAs( GetUMPFilePath() );

   WScript.Echo("Finished successfully");
}

// if something went wrong (and we did not save the project),
// we also do not want get asked for saving => set ModifiedFlag to false
if ( objDocument != null )
   objDocument.ModifiedFlag = false;

objUModel.Visible = false;   // will shutdown application if it was started by this
script
```

17.4.6.5 Dokumentation aktualisieren

Bei Ausführung des folgenden JScript-Beispiels wird bei der ersten Ausführung des Codes ein Reverse Engineering aller C#-UModel API-Beispiele aus dem Verzeichnis **..\UModelExamples\IDEPlugIn** durchgeführt und es werden HTML- und RTF-Dokumentation sowie ein XMI-Bericht des UModel-Projekts erstellt. Die erzeugten UMP-Dateien sowie die generierte Dokumentationsausgabe werden im Verzeichnis **.. \UModelExamples\API\JScript\UpdateDocumentation** gespeichert. Bei späteren Ausführungen werden die zuvor generierten Projektdateien geöffnet und HTML- und RTF-Dokumentation erstellt. Außerdem wird ein XMI-Export durchgeführt, vorausgesetzt, es wurden Änderungen am UML-Modell vorgenommen.

Dieser Code steht in der Beispieldatei `..\UModelExamples\API\JScript\UModelUpdateDocumentation.js` zur Verfügung (siehe Beispieldateien 904).

```
// ##########################################
// access runing UModel.Application or
// launch new one and access it
// ##########################################


// ##########################################
// UpdateDocumentation sample
// *) When running the first time (= when no UMP file exists), reverse engineer all C#
UModelAPI samples
//    and create HTML and RTF documentation, make XMI export and save UMP file
// *) when UMP file already exists, open it and synchronize model from code
//    create HTML and RTF documentation and XMI export only if something has been changed
(listen to all different UML data events)
// ##########################################

var bRunVisible = true;
var bShowDialogs = bRunVisible && false;

// ////////////// global variables /////////////////
var objUModel   = null;
```


© 2016-2022 Altova GmbH

```
var objWshShell = null;
var objFSO      = null;

var bChangedAnything = false;
var nAddedClasses    = 0;
var nAddedInterfaces= 0;
var nAddedProperties= 0;
var nAddedOperations= 0;

// ////////////////////// Helpers ///////////////////////////////

function Exit(strErrorText)
{
   WScript.Echo(strErrorText);

   if (objUModel != null)
      objUModel.Quit();

   WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
   // the Shell and FileSystemObject of the windows scripting host often always useful
   try
   {
      objWshShell = WScript.CreateObject("WScript.Shell");
      objFSO = WScript.CreateObject("Scripting.FileSystemObject");
   }
   catch(err)
   {
      Exit("Can't create WScript.Shell object");
   }

   // create the UModel connection
   // if there is a running instance of UModel (that never had a connection) - use it
   // otherwise, we automatically create a new instance
   try {   objUModel = WScript.GetObject("", "UModel.Application");   }
   catch(err) {}

   if( typeof( objUModel ) == "undefined" )
   {
      try   {   objUModel = WScript.GetObject("", "UModel_x64.Application")   }
      catch(err)
      {
         objUModel = null;
         Exit( "Can't access or create UModel.Application" );
      }
   }
}

// ////////////////////// get different filepathes / ensure folders are
created  ////////////////////////////
function GetScriptPath()
{
   var path = objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\
\UpdateDocumentation";

   if ( !objFSO.FolderExists( path ) )
```

```
      objFSO.CreateFolder( path );

    return path;
}

function GetFilePath( subdir, filename )
{
   var path = objFSO.BuildPath( GetScriptPath(), subdir );

   if ( !objFSO.FolderExists( path ) )
      objFSO.CreateFolder( path );

   return path + "\\" + filename;
}

function GetUMPFilePath ()    { return GetFilePath( "UMP",       "UModelAPI.ump" ); }
function GetXMIFilePath ()    { return GetFilePath( "Output_XMI", "UModelAPI.xmi" ); }
function GetHTMLFilePath()    { return GetFilePath( "Output_HTML","UModelAPI.html"); }
function GetRTFFilePath ()    { return GetFilePath( "Output_RTF", "UModelAPI.rtf" ); }

// ////////////////////// UML data event handlers ////////////////////////////
function objRootPackage_OnChanged( objData, strHint )
{
   bChangedAnything = true;
}

// recursively count newly added classes, interfaces, properties and operations
function CountAddedElements( objNewChild )
{
   if ( objNewChild != null )
   {
      if ( objNewChild.KindName == "Class"     ) ++nAddedClasses;
      if ( objNewChild.KindName == "Interface" ) ++nAddedInterfaces;
      if ( objNewChild.KindName == "Property"  ) ++nAddedProperties;
      if ( objNewChild.KindName == "Operation" ) ++nAddedOperations;

      var ownedElements = objNewChild.OwnedElements;
      var itr = new Enumerator( ownedElements );
      for ( ; !itr.atEnd(); itr.moveNext() )
         CountAddedElements( itr.item() );
   }
}

function objRootPackage_OnAfterAddChild( objParent, objNewChild )
{
    bChangedAnything = true;

   // recursively count newly added classes, interfaces, properties and operations
   CountAddedElements( objNewChild );
}

function objRootPackage_OnBeforeErase( objData )
{
    bChangedAnything = true;
}

function objRootPackage_OnMoveData( objParent, objChild, bAttach )
{
    bChangedAnything = true;
```


© 2016-2022 Altova GmbH

```
}

// ////////////////////// MAIN //////////////////////////////

CreateGlobalObjects();

if ( bRunVisible )
   objUModel.Visible = true;

var objDocument = null;

try
{
   // open document if it exists; create new one otherwise
   var bDocumentExisted = false;

   if ( objFSO.FileExists( GetUMPFilePath() ) )
   {
      objDocument = objUModel.OpenDocument( GetUMPFilePath() );
      bDocumentExisted = true;
   }
   else
   {
      objDocument = objUModel.NewDocument();
      objDocument.SaveAs( GetUMPFilePath() );
   }

   if ( objDocument == null )
      Exit( "Cannot create or open UModel projectfile" );

   // connect to receive _IUMLDataEvents from the root-package and all its children:
   var objRootPackage = objDocument.RootPackage;
    WScript.ConnectObject (objRootPackage, "objRootPackage_" );

   // ensure we get *all* events from root-package and *all* children:
   objRootPackage.EventFilter = 2 + // eUMLDataEvent_EraseDataOrChild      = 2,
                          8 + // eUMLDataEvent_AddChildOrGrandChild   = 8,
                         32 + // eUMLDataEvent_ChangeDataOrChild      = 32,
                         128; // eUMLDataEvent_MoveChildOrGrandChild   = 128
   if ( bDocumentExisted )
   {
      // UModel projectfile already exists => update model from code

      // get dialog for code <=> model synchonizations and set the wanted options:
      var objSynchronizationSettingsDlg = objUModel.Dialogs.SynchronizationSettingsDlg;
      objSynchronizationSettingsDlg.ShowDialog = bShowDialogs;

      objSynchronizationSettingsDlg.ModelFromCode_Synchronization = 0; //
eSynchronization_Merge = 0

      // update model from code
      if ( !objDocument.SynchronizeModelFromCode( objSynchronizationSettingsDlg ) )
         Exit("Update model from code failed");
   }
   else
   {
      // UModel projectfile did not exist => newly import code into model

      var objImportSourceDirectoryDlg = objUModel.Dialogs.ImportSourceDirectoryDlg;
```

```
      objImportSourceDirectoryDlg.ShowDialog = bShowDialogs;

      // set source code directory to import
      objImportSourceDirectoryDlg.Directory = objUModel.PersonalDataDirectory + "\
\UModelExamples\\IDEPlugIn";
      objImportSourceDirectoryDlg.ProcessSubdirectories = true;
      // set source code language to import (C# 3.0)
      objImportSourceDirectoryDlg.Language      = 5; // eCodeLang_CSharp_3_0      = 5
      objImportSourceDirectoryDlg.Synchronization = 0; // eSynchronization_Merge = 0
      // import in a new package
      objImportSourceDirectoryDlg.ImportInNewPackage = true;

      objImportSourceDirectoryDlg.DiagramGeneration = true;

      // content diagram generation settings
      objImportSourceDirectoryDlg.Content_GenerateSingleDiagram            = true;
      objImportSourceDirectoryDlg.Content_GenerateDiagramPerPackage         = true;
      objImportSourceDirectoryDlg.Content_ShowNestedClassifiersSeparately      = false;
      objImportSourceDirectoryDlg.Content_ShowAnonymousBoundElements         = false;
      objImportSourceDirectoryDlg.Content_HyperlinkPackagesToDiagrams         = true;
      objImportSourceDirectoryDlg.Content_ShowAttributesCompartment         = true;
      objImportSourceDirectoryDlg.Content_ShowOperationsCompartment         = true;
      objImportSourceDirectoryDlg.Content_ShowNestedClassifiersCompartment   = false;
      objImportSourceDirectoryDlg.Content_ShowEnumerationLiteralsCompartment   = true;
      objImportSourceDirectoryDlg.Content_ShowTaggedValues               = true;
      objImportSourceDirectoryDlg.Content_Autolayout                     = 1; //
eDiagramLayout_Hierarchic = 1
      // open diagrams that autolayout is done:
      objImportSourceDirectoryDlg.Content_OpenDiagrams                  = true;

      // package dependency diagram generation settings (disabled)
      objImportSourceDirectoryDlg.PackageDependency_GenerateDiagram = false;

      // import source directory
      if ( !objDocument.ImportSourceDirectory( objImportSourceDirectoryDlg) )
      {
         // also delete newly created (empty) UMP file that source code directory import
is retried the next time
         objFSO.DeleteFile( GetUMPFilePath() );
         Exit( "Error on importing source directory" );
      }
   }

   // disconnect from getting root-package events
   WScript.DisconnectObject( objRootPackage );
}
catch( err )
{
   // also delete newly created (empty) UMP file that source code directory import is
retried the next time
   objFSO.DeleteFile( GetUMPFilePath() );
   Exit( "Error on importing source directory" );
}

//if something has changed, update the outputs:
if ( bChangedAnything )
{
   try
   {
```


© 2016-2022 Altova GmbH

```
      // make XMI export for UML2.1.2
      var objIExportXMIFileDlg = objUModel.Dialogs.ExportXMIFileDlg;
      objIExportXMIFileDlg.ShowDialog = bShowDialogs;
      objIExportXMIFileDlg.XMIFile            = GetXMIFilePath();
      objIExportXMIFileDlg.PrettyPrintXMIOutput   = true;
      objIExportXMIFileDlg.ExportUUIDs         = true;
      objIExportXMIFileDlg.ExportExtensions      = true;
      objIExportXMIFileDlg.ExportDiagrams         = true;
      objIExportXMIFileDlg.XMIType            = 1; // eXMI21ForUML212 = 1

      // export to XMI file:
      if ( !objDocument.ExportToXMIFile( objIExportXMIFileDlg ) )
      {
         // error on XMI generation
      }
   }
   catch( err )
   {
      // error on XMI generation
   }

   try
   {
      var objIDocumentationGenerationDlg = objUModel.Dialogs.GenerateDocumentationDlg;
      objIDocumentationGenerationDlg.ShowDialog = bShowDialogs;

      objIDocumentationGenerationDlg.GenerateLinksToLocalFiles = 1; //
eDocumentationFilePath_RelativeToResultFile   = 1
      objIDocumentationGenerationDlg.SplitOutputToMultipleFiles = true;
      objIDocumentationGenerationDlg.ShowResultFileAfterGeneration = true;

      objIDocumentationGenerationDlg.Details_SelectAll();
      // show up to 10 base class/interface hierarchies
      objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthUp = 10;
      // only show directly derived classes/interfaces
      objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthDown = 1;
      // keep hierarchy diagram as small as possible => expand each element only once
      objIDocumentationGenerationDlg.Details_HierarchyDiagramExpandItemsOnlyOnce = true;

      objIDocumentationGenerationDlg.Include_SelectAllDiagrams();
      objIDocumentationGenerationDlg.Include_SelectNoElements();
      objIDocumentationGenerationDlg.Include_Index = true;
      objIDocumentationGenerationDlg.Include_IncludedSubprojects = false;
      objIDocumentationGenerationDlg.Include_NamedElementsOnly = true;
      objIDocumentationGenerationDlg.Include_UnknownExternals = false;

      var objIncludeElements = objIDocumentationGenerationDlg.Include_Elements;
      var itrIncludeElements = new Enumerator( objIncludeElements );
      for ( ; !itrIncludeElements.atEnd(); itrIncludeElements.moveNext() )
      {
         var objElemSel = itrIncludeElements.item();

         if ( objElemSel.KindName == "Class"         ||
              objElemSel.KindName == "Interface"     ||
              objElemSel.KindName == "Enumeration"   ||
              objElemSel.KindName == "Operation"     ||
              objElemSel.KindName == "Package"      )
         {
            objElemSel.Selection = true;
```

```
         }
      }

      // generate HTML documentation (with PNG pictures)
      objIDocumentationGenerationDlg.OutputFormat = 0; // eDocumentationOutputFormat_HTML
  = 0
      objIDocumentationGenerationDlg.DiagramImageFormat = 0; // eOutputImageFormat_PNG =
0
      objIDocumentationGenerationDlg.EmbedDiagrams = false;
      if ( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetHTMLFilePath() ) )
      {
         // error on HTML documentation generation
      }

      // generate RTF documentation (with embeded EMF pictures)
      objIDocumentationGenerationDlg.ShowDialog = false; // don't show dialog again
      objIDocumentationGenerationDlg.OutputFormat = 2; // eDocumentationOutputFormat_RTF
  = 2
      objIDocumentationGenerationDlg.DiagramImageFormat = 1; // eOutputImageFormat_EMF =
1
      objIDocumentationGenerationDlg.EmbedDiagrams = true;
      if ( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetRTFFilePath() ) )
      {
         // error on RTF documentation generation
      }
   }
   catch( err )
   {
      // error on documentation generation
   }

   // show the number of newly added classes, interfaces, properties and operations
   if ( bRunVisible )
   {
      WScript.Echo( "Added classes: "       + nAddedClasses   +
                    "\nAdded interfaces: " + nAddedInterfaces +
                    "\nAdded properties: " + nAddedProperties +
                    "\nAdded operations: " + nAddedOperations );
   }
}
else
{
   if ( bRunVisible )
      WScript.Echo( "Nothing has changed" );
}

// always save document (although it's not really necessary when nothing has been
changed)
objDocument.Save();

if ( bRunVisible )
   objUModel.Visible = false;   // will shutdown application if it was started by this
script
```


© 2016-2022 Altova GmbH

17.5 UModel API-Referenz

In dieser Dokumentation werden die Schnittstellen, Operationen, Enumerationen und Events der UModel API[921] beschrieben. Der Inhalt ist in die folgenden Unterabschnitte gegliedert:

- UModel Plug-ins[919]: Hier finden Sie Informationen zu den erforderlichen Schnittstellen, um Ihre eigenen Plug-ins in UModel zu integrieren.
- UModel API-Schnittstellen[921]: Hier finden Sie Informationen über alle Schnittstellen der UModel API mit Ausnahme der UML Data-Schnittstellen (siehe nächster Punkt)
- UMLData-Schnittstellen[1010]: Hier finden Sie Informationen zu den Schnittstellen auf der UML-Daten-Ebene. Diese Schnittstellen bilden auch Teil der UModel API, werden aber separat beschrieben. Sie bieten Zugriff auf UML-Elemente in einem UModel-Dokument.

17.5.1 UModel Plug-Ins

In diesem Abschnitt finden Sie Informationen zu den erforderlichen API-Schnittstellen, um Ihre eigenen Plug-ins in UModel zu integrieren. Informationen zu Begriffen und Anleitungen zum Erstellen von UModel IDE-Plug-ins finden Sie unter UModel IDE Plug-Ins[837].

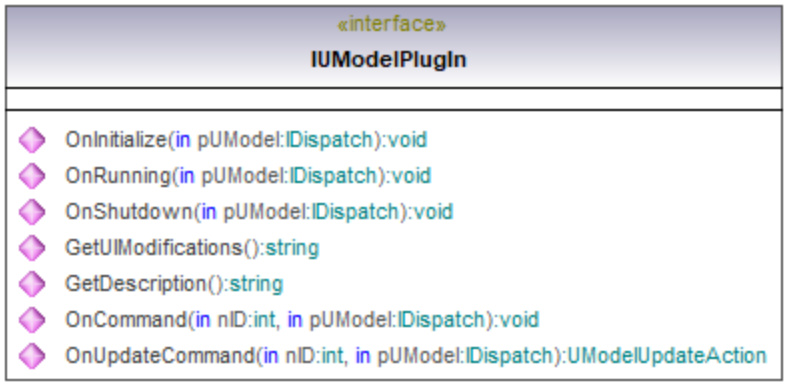
C#-Codebeispiele für in UModel integrierte Plug-ins finden Sie in den folgenden Kapiteln:

- Beispiel "Stile definieren"[882]
- Beispiel "C# Delegate"[886]
- Beispiel "Präfix definieren"[891]
- Beispiel "Statistik"[897]

17.5.1.1 UModelAPI - IUModelPlugIn

Interface **IUModelPlugIn**

diagram



Operation **IUModelPlugIn::GetDescription**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUModelPlugIn::GetUIModifications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUModelPlugIn::OnCommand**

parameter	name	direction	type	type modifier	multiplicity	default
	nID	**in**	**int**			
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnInitialize**

parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnRunning**

parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

Operation **IUModelPlugIn::OnShutdown**

parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	**in**	**IDispatch**			
	return	**return**	**void**			

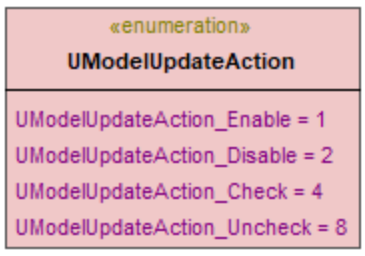
Operation **IUModelPlugIn::OnUpdateCommand**

parameter	name	direction	type	type modifier	multiplicity	default
	nID	**in**	**int**			
	pUModel	**in**	**IDispatch**			
	return	**return**	**UModelUpdateAction** 920			

17.5.1.2 UModelAPI - UModelUpdateAction

Enumeration **UModelUpdateAction**

diagram	



 © 2016-2022 Altova GmbH

typedElements	Interface IUModelPlugIn 919	Operation OnUpdateCommand 920

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2 UModel API-Schnittstellen

Dieser Abschnitt enthält Informationen zu den Objekten der UModel COM API. Die Objekte sind allgemein beschrieben, da die API mit praktisch jeder Sprache, die den Aufruf eines COM-Objekts unterstützt, verwendet werden kann. Sprachspezifische Beispiele finden Sie unter:

- C#-Beispielprojekt 876
- Java-Beispielprojekt 903
- JScript-Beispiele 904

Die API besteht aus zwei Hauptabschnitten, in denen die in der jeweiligen API verwendeten Schnittstellen und Enumerationstypen beschrieben sind. Die Enumerationswerte enthalten sowohl den String-Namen als auch einen numerischen Wert. Wenn Ihre Skripting-Umgebung Enumerationen nicht unterstützt, verwenden Sie stattdessen die numerischen Werte.

In .NET gibt es für jede Schnittstelle der MapForce COM Automation Interface eine .NET-Klasse mit demselben Namen. Auch COM-Typen werden in den entsprechenden .NET-Typ konvertiert. So wird etwa ein Typ wie `Long` aus der COM API in .NET als `System.Int32` angezeigt.

Beachten Sie in Java die folgenden Syntaxvarianten:

- **Klassen und Klassennamen** Für jede Schnittstelle des MapForce Automation Interface gibt es eine Java-Klasse mit dem Namen der Schnittstelle.
- **Methodennamen** Die Methodennamen im Java Interface sind dieselben wie die in den COM Interfaces, beginnen aber aufgrund der Java-Namenskonventionen mit einem Kleinbuchstaben. Zum Aufrufen von COM-Eigenschaften können Java-Methoden verwendet werden, deren Eigenschaftsname das Präfix get und set erhalten. Wenn eine Eigenschaft keinen Schreibzugriff ermöglicht, steht keine Setter-Methode zur Verfügung. So stehen z.B. für die Eigenschaft `Name` des `Document` Interface stehen die Java-Methoden `getName` und `setName` zur Verfügung.
- **Enumerationen** Für jede im Automation Interface definierte Enumeration ist eine Java-Enumeration desselben Namens und mit denselben Werten definiert.
- **Events und Event Handler** Für jedes Interface im Automation Interface, das Events unterstützt, steht ein Java-Interface desselben Namens plus 'Event' zur Verfügung. Um das Überladen von Einzel-Events zu vereinfachen, gibt es eine Java-Klasse mit Standardimplementierungen für alle Events. Der Name dieser Java-Klasse ist der Name des Event Interface plus 'DefaultHandler'. Beispiel:

```
Application // Java class to access the application
ApplicationEvents // Events interface for the application
ApplicationEventsDefaultHandler // Default handler for "ApplicationEvents"
```

UModel API-Fehler

Die UModel API kann die unten aufgelisteten Fehlercodes zurückgeben.

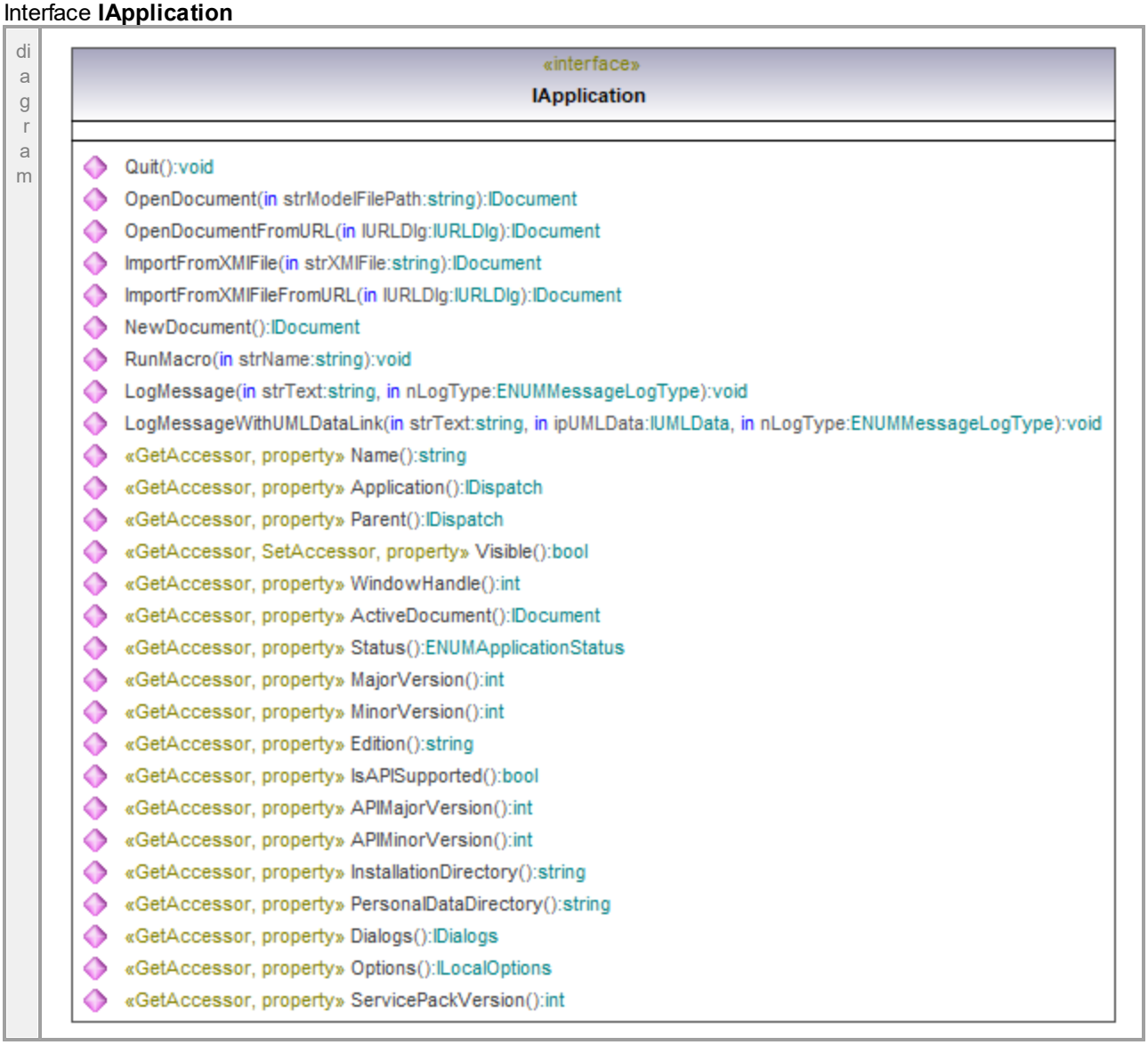
1000	Das application-Objekt ist nicht mehr gültig.
1001	Es wurde ein ungültiger Parameter oder eine ungültige Adresse für den Rückgabeparameter definiert.
1002	Die UModel API steht in der aktuellen Version nicht zur Verfügung.
1003	Modelltransformationen werden in der aktuellen Edition nicht unterstützt
1050	Makro nicht gefunden
1051	Ungültige (verschachtelte) Makroausführung
1100	Fehler beim Speichern der Datei. Wahrscheinlich war der Dateiname ungültig.
1101	Ungültiger (doppelter) Aufruf von BeginModification.
1102	EndModification wurde ohne BeginModification aufgerufen
1200	Fehler beim Löschen der Datei an der URL.
1201	Fehler beim Erstellen des Verzeichnisses an der URL.

Die `UMLData`-Schnittstellen haben spezifische Fehler, siehe UMLData-Schnittstellen 1010.

 © 2016-2022 Altova GmbH

17.5.2.1 UModelAPI - IApplication

Interface **IApplication**



Operation **IApplication::ActiveDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDocument** 938			

Operation **IApplication::APIMajorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	A change in the **APIMajorVersion** of the type library (e.g. 1.0 => 2.0) means that non-scripting clients (e.g. IDE PlugIns written in C#, VB.NET, C++,...) should be recompiled.					

Operation **IApplication::APIMinorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IApplication::Dialogs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IDialogs 936			

Operation **IApplication::Edition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::ImportFromXMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	strXMIFile	**in**	**string**			
	return	**return**	IDocument 938			

Operation **IApplication::ImportFromXMIFileFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	IURLDlg 997			
	return	**return**	IDocument 938			

Operation **IApplication::InstallationDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::IsAPISupported**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IApplication::LogMessage**

parameter	name	direction	type	type modifier	multiplicity	default
	strText	**in**	**string**			
	nLogType	**in**	**ENUMMessageLogType**			
	return	**return**	**void**			

Operation **IApplication::LogMessageWithUMLDataLink**

parameter	name	direction	type	type modifier	multiplicity	default
	strText	**in**	**string**			
	ipUMLData	**in**	IUMLData 1011			
	nLogType	**in**	**ENUMMessageLogType**			

© 2016-2022 Altova GmbH

	return	**return**	**void**

Operation **IApplication::MajorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::MinorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::NewDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDocument** 938			

Operation **IApplication::OpenDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	strModelFilePath	**in**	**string**			
	return	**return**	**IDocument** 938			

Operation **IApplication::OpenDocumentFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg** 997			
	return	**return**	**IDocument** 938			

Operation **IApplication::Options**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptions** 973			

Operation **IApplication::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IApplication::PersonalDataDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IApplication::Quit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IApplication::RunMacro**

parameter	name	direction	type	type modifier	multiplicity	default
	strName	**in**	**string**			
	return	**return**	**void**			

Operation **IApplication::ServicePackVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IApplication::Status**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMApplicationStatus 1003			

Operation **IApplication::Visible**

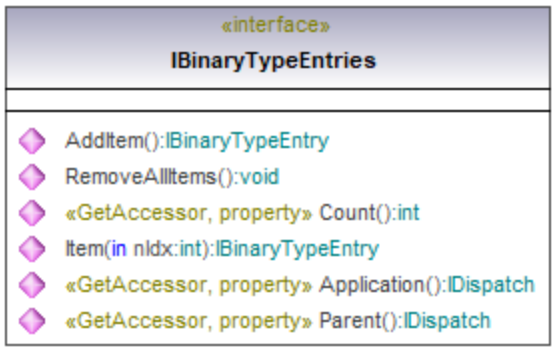
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IApplication::WindowHandle**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.2.2 UModelAPI - IBinaryTypeEntries

Interface **IBinaryTypeEntries**

diagram	
typedElements	Interface IImportBinaryTypesDlg 958 Operation CSharp_BinaryTypes 958 Java_BinaryTypes 959 VBasic_BinaryTypes 960

Operation **IBinaryTypeEntries::AddItem**

 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IBinaryTypeEntry** 927			

Operation **IBinaryTypeEntries::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IBinaryTypeEntries::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IBinaryTypeEntries::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IBinaryTypeEntry** 927			

Operation **IBinaryTypeEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

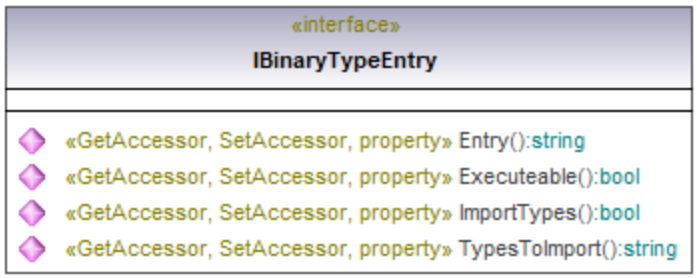
Operation **IBinaryTypeEntries::RemoveAllItems**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.3 UModelAPI - IBinaryTypeEntry

Interface **IBinaryTypeEntry**

diagram	
typedElements	Interface **IBinaryTypeEntries** 926 Operation **AddItem** 926 **Item** 927

Operation **IBinaryTypeEntry::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IBinaryTypeEntry::Executeable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IBinaryTypeEntry::ImportTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

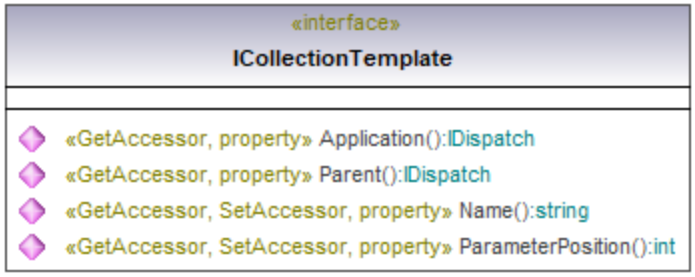
Operation **IBinaryTypeEntry::TypesToImport**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.4 UModelAPI - ICollectionTemplate

Interface **ICollectionTemplate**

diagram	
typedElements	Interface **ICollectionTemplates** 929 Operation **InsertItemAt** 929 **Item** 930

Operation **ICollectionTemplate::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ICollectionTemplate::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **ICollectionTemplate::ParameterPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

© 2016-2022 Altova GmbH

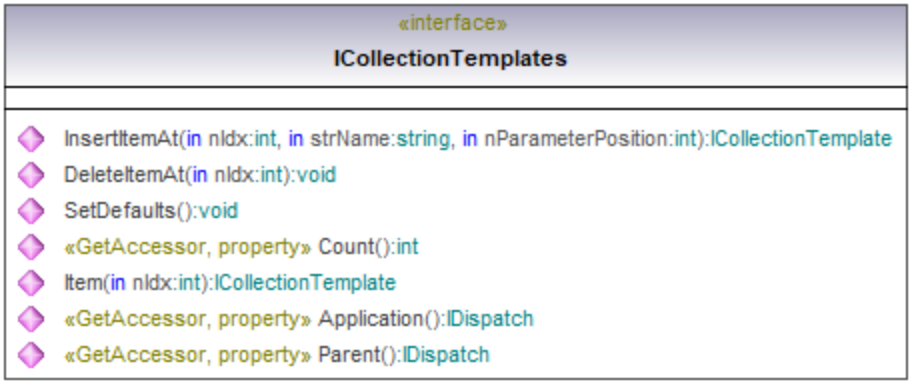
Operation **ICollectionTemplate::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.5 UModelAPI - ICollectionTemplates

Interface **ICollectionTemplates**

diagram



typedElements

Interface
ILocalOptionsDiagramEditing 977

Operation
CollectionTemplates_CSharp 978
CollectionTemplates_Java 978
CollectionTemplates_UML 978
CollectionTemplates_VBasic 978

Operation **ICollectionTemplates::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ICollectionTemplates::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ICollectionTemplates::DeleteItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **ICollectionTemplates::InsertItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strName	**in**	**string**			

	nParameterPosition	**in**	**int**			
	return	**return**	**ICollectionTemplate** 928			

Operation **ICollectionTemplates::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**ICollectionTemplate** 928			

Operation **ICollectionTemplates::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

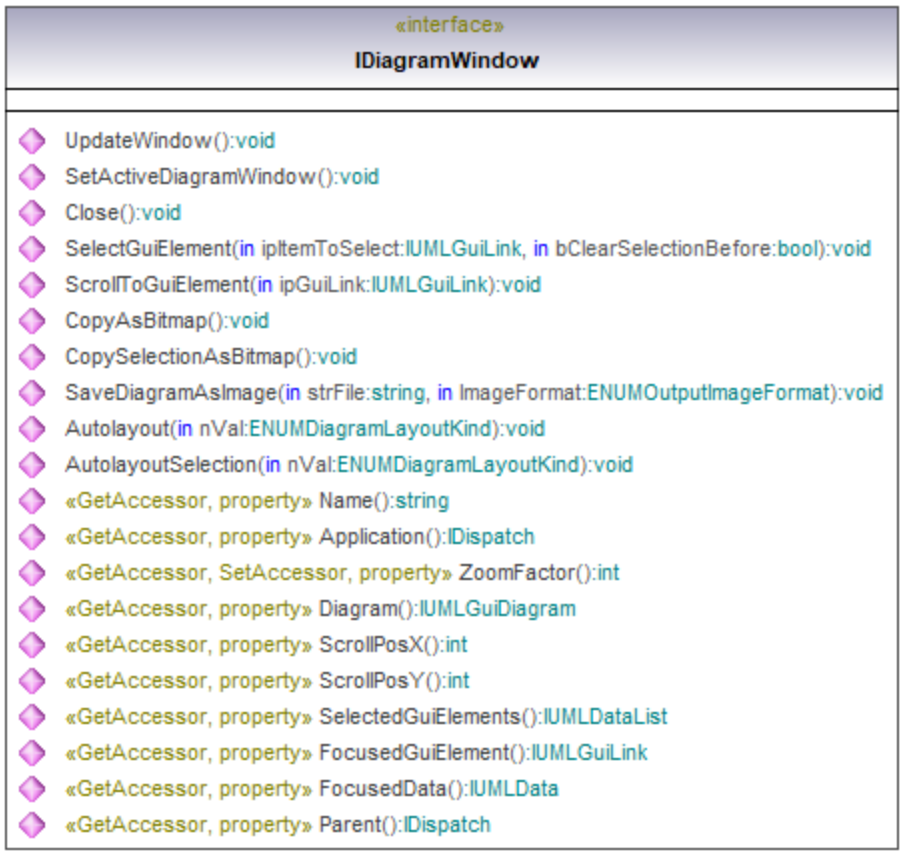
Operation **ICollectionTemplates::SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.2.6 UModelAPI - IDiagramWindow

Interface **IDiagramWindow**

diagram	
typedElements	Interface _IDiagramWindowEvents 999 Interface _IDocumentEvents 1000 Interface IDiagramWindows 934 Interface IDocument 938 Operation OnDiagramWindowClosed 999 Operation OnActivateDiagramWindow 1000 OnDiagramWindowClosed 1000 OnDiagramWindowOpened 1000 Operation Item 934 Operation ActiveDiagramWindow 939 OpenDiagram 943

Operation **IDiagramWindow::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDiagramWindow::Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	nVal	**in**	ENUMDiagramLayoutKind 1005			
	return	**return**	**void**			

Operation **IDiagramWindow::AutolayoutSelection**

parameter	name	direction	type	type modifier	multiplicity	default
	nVal	**in**	**ENUMDiagramLayoutKind** 1005			
	return	**return**	**void**			

Operation **IDiagramWindow::Close**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::CopyAsBitmap**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::CopySelectionAsBitmap**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::Diagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagram** 1316			

Operation **IDiagramWindow::FocusedData**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData** 1011			

Operation **IDiagramWindow::FocusedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink** 1328			

Operation **IDiagramWindow::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDiagramWindow::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDiagramWindow::SaveDiagramAsImage**

parameter	name	direction	type	type modifier	multiplicity	default
	strFile	**in**	**string**			
	ImageFormat	**in**	**ENUMOutputImageFormat** 1008			
	return	**return**	**void**			

Operation **IDiagramWindow::ScrollPosX**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IDiagramWindow::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IDiagramWindow::ScrollToGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	ipGuiLink	**in**	**IUMLGuiLink** 1328			
	return	**return**	**void**			

Operation **IDiagramWindow::SelectedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLGuiElement 1321.					

Operation **IDiagramWindow::SelectGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	ipItemToSelect	**in**	**IUMLGuiLink** 1328			
	bClearSelectionBefore	**in**	**bool**			
	return	**return**	**void**			

Operation **IDiagramWindow::SetActiveDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDiagramWindow::UpdateWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

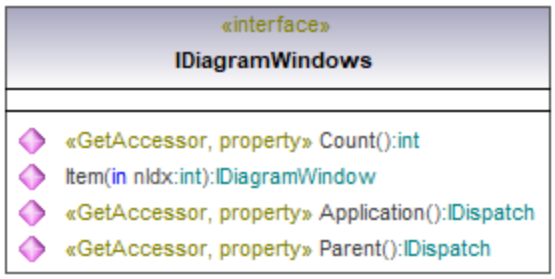
Operation **IDiagramWindow::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

© 2016-2022 Altova GmbH

17.5.2.7 UModelAPI - IDiagramWindows

Interface **IDiagramWindows**

diagram	
typedElements	Interface IDocument 938 Operation DiagramWindows 939

Operation **IDiagramWindows::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDiagramWindows::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IDiagramWindows::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IDiagramWindow 931			

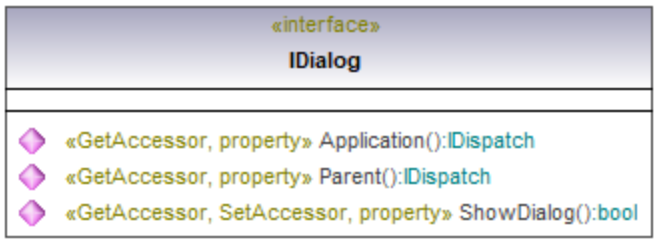
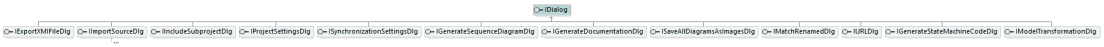
Operation **IDiagramWindows::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

© 2016-2022 Altova GmbH

17.5.2.8 UModelAPI - IDialog

Interface **IDialog**

diagram	
hierarchy	

Operation **IDialog::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialog::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

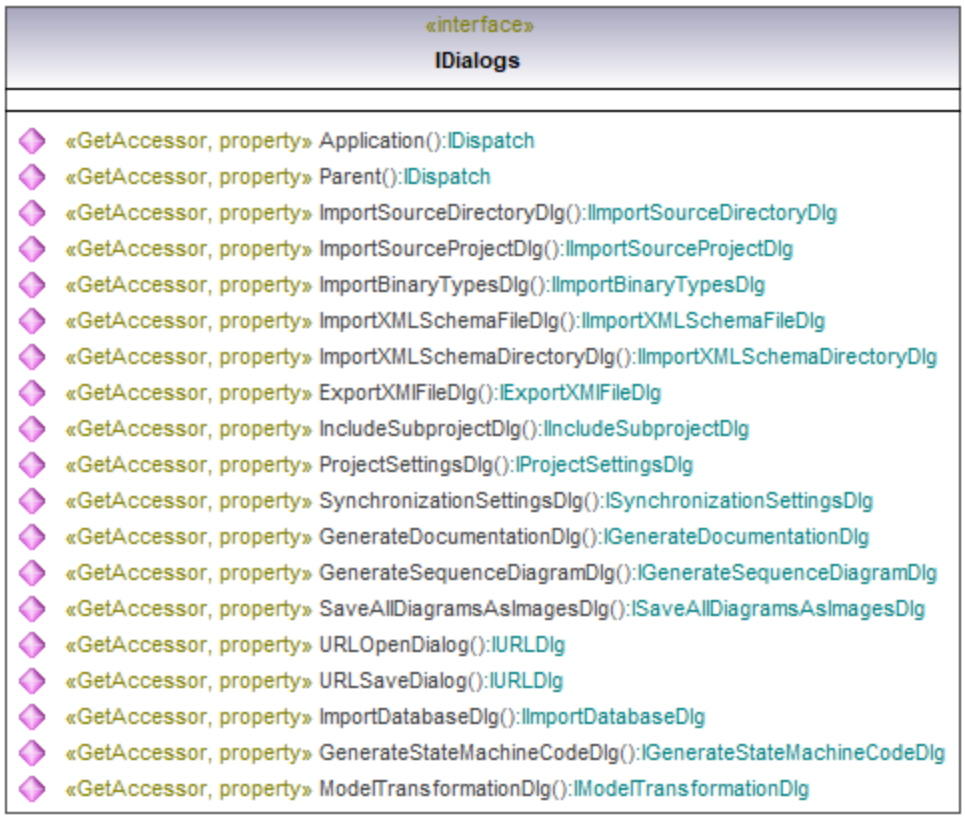
Operation **IDialog::ShowDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

17.5.2.9 UModelAPI - IDialogs

Interface **IDialogs**

diagram	
typedElements	Interface **IApplication** 923 Operation **Dialogs** 924

Operation **IDialogs::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialogs::ExportXMIFileDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IExportXMIFileDlg** 945			

Operation **IDialogs::GenerateDocumentationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IGenerateDocumentationDlg** 947			

Operation **IDialogs::GenerateSequenceDiagramDlg**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IGenerateSequenceDiagramDlg 954			

Operation **IDialogs::GenerateStateMachineCodeDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IGenerateStateMachineCodeDlg 956			

Operation **IDialogs::ImportBinaryTypesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportBinaryTypesDlg 958			

Operation **IDialogs::ImportDatabaseDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportDatabaseDlg 962			

Operation **IDialogs::ImportSourceDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportSourceDirectoryDlg 963			

Operation **IDialogs::ImportSourceProjectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportSourceProjectDlg 967			

Operation **IDialogs::ImportXMLSchemaDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportXMLSchemaDirectoryDlg 969			

Operation **IDialogs::ImportXMLSchemaFileDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IImportXMLSchemaFileDlg 970			

Operation **IDialogs::IncludeSubprojectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IIncludeSubprojectDlg 971			

Operation **IDialogs::ModelTransformationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IModelTransformationDlg** 988			

Operation **IDialogs::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDialogs::ProjectSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IProjectSettingsDlg** 992			

Operation **IDialogs::SaveAllDiagramsAsImagesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ISaveAllDiagramsAsImagesDlg** 995			

Operation **IDialogs::SynchronizationSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ISynchronizationSettingsDlg** 995			

Operation **IDialogs::URLOpenDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg** 997			

Operation **IDialogs::URLSaveDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg** 997			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.10 UModelAPI - IDocument

Interface **IDocument**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **_IApplicationEvents** 998 — Operation **OnDocumentClosed** 998, **OnDocumentOpened** 999, **OnNewDocument** 999

© 2016-2022 Altova GmbH

	Interface _IDocumentEvents 1000	Operation OnAfterReloadDocument 1000 OnBeforeReloadDocument 1000 OnDocumentClosed 1000 OnDocumentSaved 1001 OnDocumentSavedAs 1001 OnModifiedFlagChanged 1001
	Interface _ITransactionEvents 1002	Operation OnBeginDataModification 1002 OnEndDataModification 1002
	Interface IApplication 923	Operation ActiveDocument 923 ImportFromXMIFile 924 ImportFromXMIFileFromURL 924 NewDocument 925 OpenDocument 925 OpenDocumentFromURL 925

Operation **IDocument::AbortModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::ActiveDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IDiagramWindow 931			

Operation **IDocument::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDocument::BeginModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::CanFocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	IUMLData 1011			
	return	**return**	**bool**			

Operation **IDocument::CheckProjectSyntax**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::DiagramWindows**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IDiagramWindows 934			

Operation **IDocument::ElementFamilyStyles**

parameter	name	direction	type	type modifier	multiplicity	default

	strKind	**in**	**string**
	return	**return**	**IUMLGuiStyles** 1346

Operation **IDocument::EndModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::ExportToXMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IExportXMIFileDlg** 945			
	return	**return**	**bool**			

Operation **IDocument::FocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData** 1011			
documentation	Get the focused UML data of the docoument. Normally this is the one which is shown in the "Properties" window.					

Operation **IDocument::FocusedUMLDataNotifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IFocusedUMLDataNotifier** 947			

Operation **IDocument::FocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData** 1011			
	bFocusModelTree	**in**	**bool**			
	bEnsureModelTreeVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IDocument::FullName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDocument::GenerateDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IGenerateDocumentationDlg** 947			
	strResultFile	**in**	**string**			
	return	**return**	**bool**			

Operation **IDocument::GenerateSequenceDiagram**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	ipDlg	**in**	IGenerateSequenceDiagramDlg 954
	ipOp	**in**	IUMLOperation 1236
	return	**return**	**bool**

Operation **IDocument::GenerateSequenceDiagramsForAllOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	bAllPublicOnly	**in**	**bool**			
	bIncludeGettersAndSetters	**in**	**bool**			
	return	**return**	**void**			

Operation **IDocument::GenerateStateMachineCode**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	IGenerateStateMachineCodeDlg 956			
	ipStateMachine	**in**	IUMLStateMachine 1272			
	return	**return**	**bool**			

Operation **IDocument::GuiRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiRootElement 1338			

Operation **IDocument::ImportBinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	IImportBinaryTypesDlg 958			
	return	**return**	**bool**			

Operation **IDocument::ImportDatabase**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	IImportDatabaseDlg 962			
	return	**return**	**bool**			

Operation **IDocument::ImportSourceDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	IImportSourceDirectoryDlg 963			
	return	**return**	**bool**			

Operation **IDocument::ImportSourceProject**

parameter	name	direction	type	type modifier	multiplicity	default

	ipDlg	**in**	**IImportSourceProjectDlg** 967
	return	**return**	**bool**

Operation **IDocument::ImportXMLSchemaDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportXMLSchemaDirectoryDlg** 969			
	return	**return**	**bool**			

Operation **IDocument::ImportXMLSchemaFile**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportXMLSchemaFileDlg** 970			
	return	**return**	**bool**			

Operation **IDocument::IncludeSubproject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IIncludeSubprojectDlg** 971			
	return	**return**	**bool**			

Operation **IDocument::IsInUndoRedo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::IsLoadedFromPreviousFileFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	True, when the document has been loaded from a project file with a previous file format version. When saving the document, previous versions of UModel will not be able to load this file anymore.					

Operation **IDocument::LineStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles** 1346			

Operation **IDocument::MergeProject**

parameter	name	direction	type	type modifier	multiplicity	default
	strProjectFile	**in**	**string**			
	return	**return**	**bool**			

Operation **IDocument::MergeProject3Way**

parameter	name	direction	type	type modifier	multiplicity	default
	strProjectFile	**in**	**string**			
	strCommonAncestorFile	**in**	**string**			

© 2016-2022 Altova GmbH

	return	**return**	**bool**			

Operation **IDocument::MergeProjectFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	lURLDlg	**in**	IURLDlg 997			
	return	**return**	**bool**			

Operation **IDocument::ModelTransformation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	IModelTransformationDlg 988			
	return	**return**	**bool**			

Operation **IDocument::ModifiedFlag**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDocument::NodeStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiStyles 1346			

Operation **IDocument::OpenAllDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::OpenDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLDiagram	**in**	IUMLGuiDiagram 1316			
	return	**return**	IDiagramWindow 931			

Operation **IDocument::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IDocument::Path**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IDocument::ProjectSettings**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IProjectSettingsDlg** 992			
	return	**return**	**void**			

Operation **IDocument::ProjectStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles** 1346			

Operation **IDocument::Reload**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::RootPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

Operation **IDocument::Save**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IDocument::SaveAllDiagramsAsImages**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISaveAllDiagramsAsImagesDlg** 995			
	return	**return**	**bool**			

Operation **IDocument::SaveAs**

parameter	name	direction	type	type modifier	multiplicity	default
	strFileName	**in**	**string**			
	return	**return**	**void**			

Operation **IDocument::SaveCopyAs**

parameter	name	direction	type	type modifier	multiplicity	default
	strFileName	**in**	**string**			
	return	**return**	**void**			

Operation **IDocument::Saved**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IDocument::SaveToURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg** 997			

© 2016-2022 Altova GmbH

	return	**return**	**void**

Operation **IDocument::SynchronizationSettings**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg** 995			
	return	**return**	**void**			

Operation **IDocument::SynchronizeCodeFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg** 995			
	return	**return**	**bool**			

Operation **IDocument::SynchronizeModelFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg** 995			
	return	**return**	**bool**			

Operation **IDocument::TransactionNotifier**

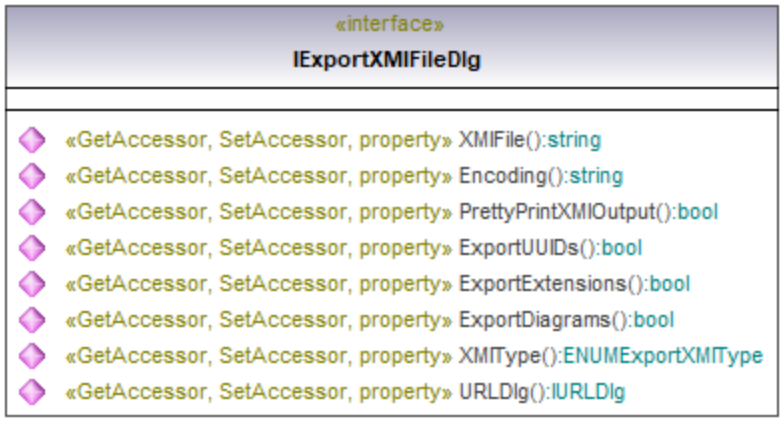
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ITransactionNotifier** 996			

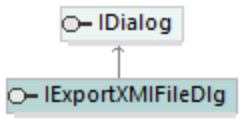
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.11 UModelAPI - IExportXMIFileDlg

Interface **IExportXMIFileDlg**

diagram



hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation ExportXMIFileDlg 936 Operation ExportToXMIFile 940

Operation **IExportXMIFileDlg::Encoding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IExportXMIFileDlg::ExportDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::ExportExtensions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::ExportUUIDs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::PrettyPrintXMIOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IExportXMIFileDlg::URLDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IURLDlg 997			

Operation **IExportXMIFileDlg::XMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

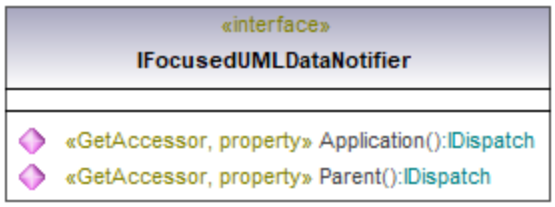
Operation **IExportXMIFileDlg::XMIType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMExportXMIType 1007			

© 2016-2022 Altova GmbH

17.5.2.12 UModelAPI - IFocusedUMLDataNotifier

Interface **IFocusedUMLDataNotifier**

diagram	
typedElements	Interface IDocument 938 Operation FocusedUMLDataNotifier 940

Operation **IFocusedUMLDataNotifier::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

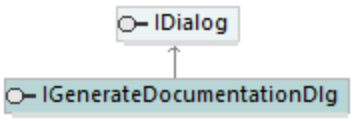
Operation **IFocusedUMLDataNotifier::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.13 UModelAPI - IGenerateDocumentationDlg

Interface **IGenerateDocumentationDlg**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation GenerateDocumentationDlg 936 Operation GenerateDocumentation 940

Operation **IGenerateDocumentationDlg::CreateFolderForDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Associations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_BoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Constraints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Diagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Documentation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_DocumentationAsHTML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramExpandItemsOnlyOnce**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramNestingDepthDown**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramNestingDepthUp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Details_Hyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**bool**

Operation **IGenerateDocumentationDlg::Details_ImplementedInterfaces**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_OperationParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Properties**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_SelectAll**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Details_SelectNone**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Details_ShownOnDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_SourceTargetOfRelations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_TemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_TemplateParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Details_TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::DiagramImageFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMOutputImageFormat** 1008			

Operation **IGenerateDocumentationDlg::EmbedCSSinHTML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::EmbedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_GetFace**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	return	**return**	**string**			

Operation **IGenerateDocumentationDlg::Fonts_GetSize**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Fonts_GetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Fonts_IsBold**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_IsItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_IsUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_SetBold**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetFace**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetSize**

parameter	name	direction	type	type modifier	multiplicity	default

	nSetting	**in**	**ENUMDocumentationFontSetting** 1006
	nNewVal	**in**	**int**
	return	**return**	**void**

Operation **IGenerateDocumentationDlg::Fonts_SetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting** 1006			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::GenerateLinksToLocalFiles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDocumentationFilePathKind** 1006			

Operation **IGenerateDocumentationDlg::Include_Diagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IKindSelectionList** 972			

Operation **IGenerateDocumentationDlg::Include_Elements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IKindSelectionList** 972			

Operation **IGenerateDocumentationDlg::Include_IncludedPredefinedSubprojects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_IncludedSubprojects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_Index**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_NamedElementsOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Include_SelectAllDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectAllElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectAllKindsOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	**in**	**string**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	**in**	**string**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectNoDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_SelectNoElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Include_UnknownExternals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::OutputFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDocumentationOutputForma**			

	t 1007

Operation **IGenerateDocumentationDlg::ShowResultFileAfterGeneration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::SplitOutputToMultipleFiles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::SPSFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateDocumentationDlg::UseFixedDesign**

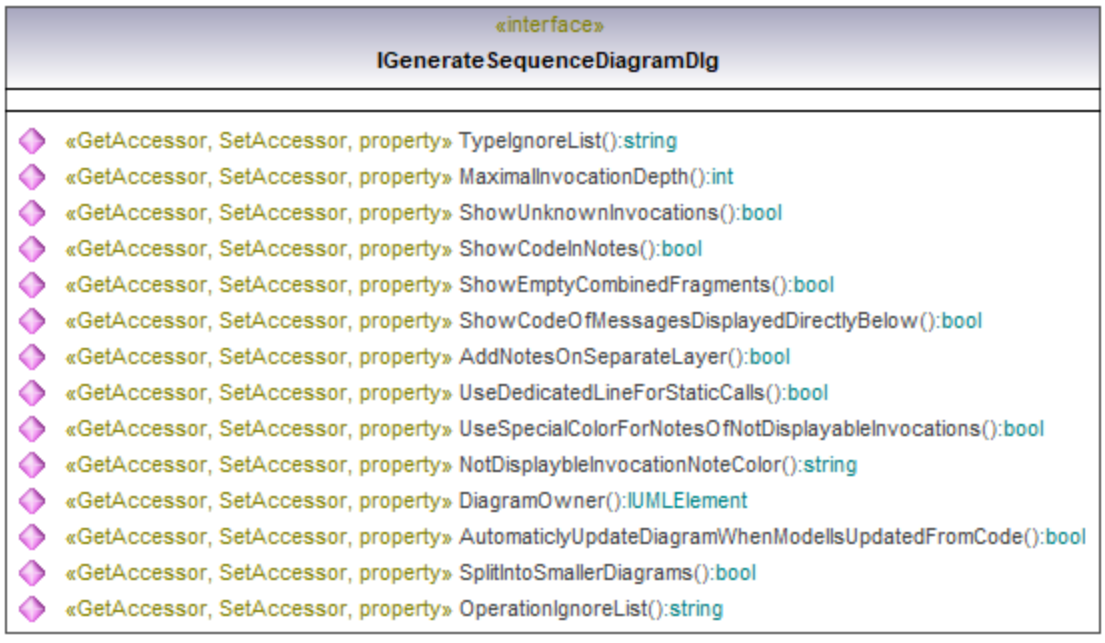
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

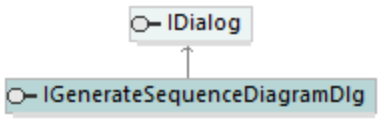
17.5.2.14 UModelAPI - IGenerateSequenceDiagramDlg

Interface **IGenerateSequenceDiagramDlg**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation GenerateSequenceDiagramDlg 936 Operation GenerateSequenceDiagram 940

Operation **IGenerateSequenceDiagramDlg::AddNotesOnSeparateLayer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::AutomaticlyUpdateDiagramWhenModelIsUpdatedFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::DiagramOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement 1157			

Operation **IGenerateSequenceDiagramDlg::MaximalInvocationDepth**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IGenerateSequenceDiagramDlg::NotDisplaybleInvocationNoteColor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateSequenceDiagramDlg::OperationIgnoreList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateSequenceDiagramDlg::ShowCodeInNotes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::ShowCodeOfMessagesDisplayedDirectlyBelow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::ShowEmptyCombinedFragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::ShowUnknownInvocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::SplitIntoSmallerDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateSequenceDiagramDlg::TypeIgnoreList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateSequenceDiagramDlg::UseDedicatedLineForStaticCalls**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

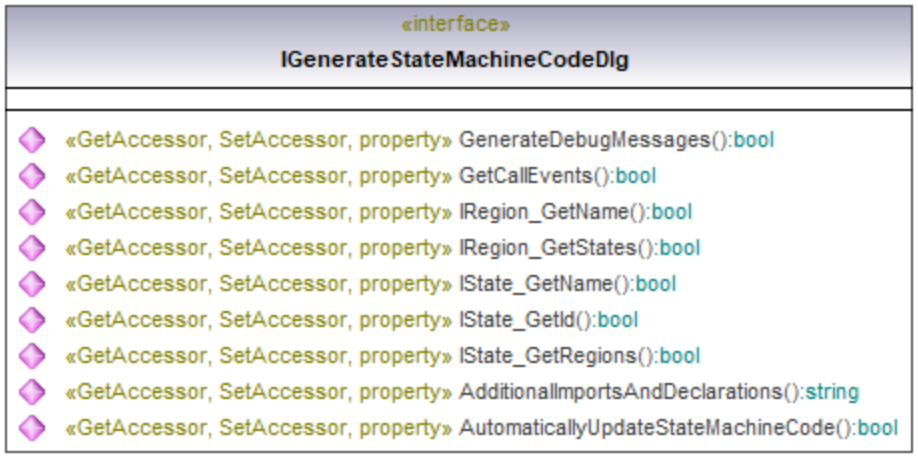
Operation **IGenerateSequenceDiagramDlg::UseSpecialColorForNotesOfNotDisplayableInvocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

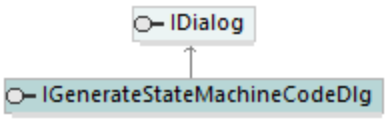
17.5.2.15 UModelAPI - IGenerateStateMachineCodeDlg

Interface **IGenerateStateMachineCodeDlg**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation GenerateStateMachineCodeDlg 937 Operation GenerateStateMachineCode 941

Operation **IGenerateStateMachineCodeDlg::AdditionalImportsAndDeclarations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IGenerateStateMachineCodeDlg::AutomaticallyUpdateStateMachineCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::GenerateDebugMessages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::GetCallEvents**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IRegion_GetName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IRegion_GetStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IState_GetId**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IGenerateStateMachineCodeDlg::IState_GetName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

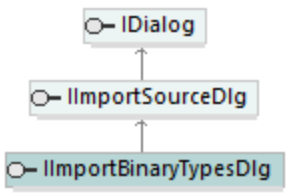
Operation **IGenerateStateMachineCodeDlg::IState_GetRegions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2.16 UModelAPI - IImportBinaryTypesDlg

Interface **IImportBinaryTypesDlg**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation ImportBinaryTypesDlg 937 Operation ImportBinaryTypes 941

Operation **IImportBinaryTypesDlg::Content_GenerateDiagramPerPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_GenerateSingleDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_HyperlinkPackagesToDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_ShowAnonymousBoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Content_ShowNestedClassifiersSeparately**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IBinaryTypeEntries 926			

Operation **IImportBinaryTypesDlg::CSharp_ImportReferencedTypes**

 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_ImportTypesOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_OneAttributePerAttributeSection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_ReflectionOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_SuppressAttributeSections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_SuppressAttributeSuffix**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IBinaryTypeEntries 926			

Operation **IImportBinaryTypesDlg::Java_ImportReferencedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_ImportTypesOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_SuppressAnnotationModifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Runtime**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IBinaryTypeEntries 926			

Operation **IImportBinaryTypesDlg::VBasic_ImportReferencedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_ImportTypesOnly**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_OneAttributePerAttributeSection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportBinaryTypesDlg::VBasic_ReflectionOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportBinaryTypesDlg::VBasic_SuppressAttributeSections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

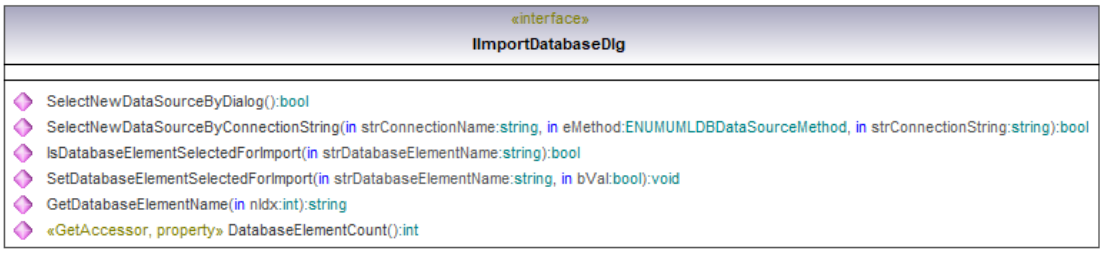
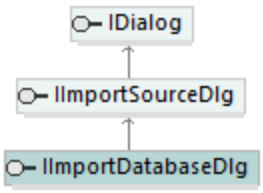
Operation **IImportBinaryTypesDlg::VBasic_SuppressAttributeSuffix**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

17.5.2.17 UModelAPI - IImportDatabaseDlg

Interface **IImportDatabaseDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation ImportDatabaseDlg 937 Operation ImportDatabase 941

Operation **IImportDatabaseDlg::DatabaseElementCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IImportDatabaseDlg::GetDatabaseElementName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IImportDatabaseDlg::IsDatabaseElementSelectedForImport**

parameter	name	direction	type	type modifier	multiplicity	default
	strDatabaseElementName	**in**	**string**			
	return	**return**	**bool**			

Operation **IImportDatabaseDlg::SelectNewDataSourceByConnectionString**

parameter	name	direction	type	type modifier	multiplicity	default
	strConnectionName	**in**	**string**			
	eMethod	**in**	**ENUMUMLDBDataSourceMethod** 1371			
	strConnectionString	**in**	**string**			
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

Operation **IImportDatabaseDlg::SelectNewDataSourceByDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

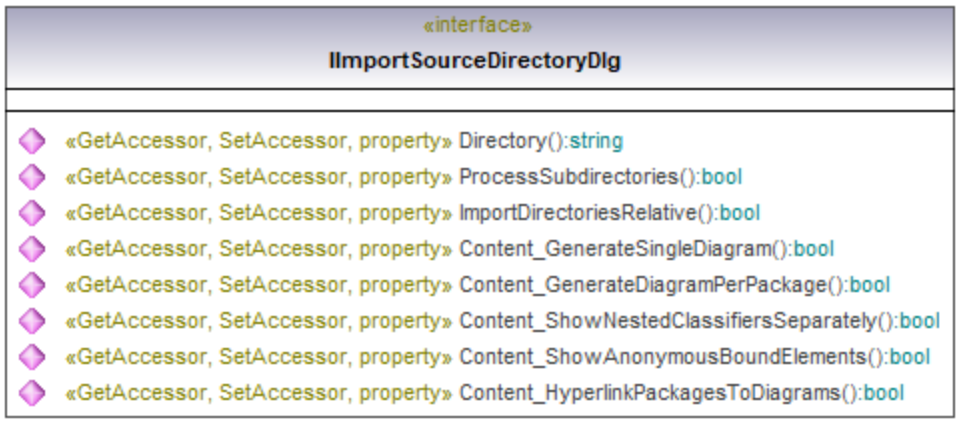
Operation **IImportDatabaseDlg::SetDatabaseElementSelectedForImport**

parameter	name	direction	type	type modifier	multiplicity	default
	strDatabaseElementName	**in**	**string**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

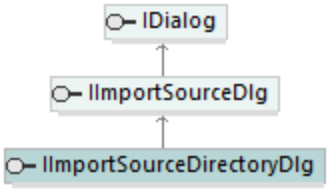
17.5.2.18 UModelAPI - IImportSourceDirectoryDlg

Interface **IImportSourceDirectoryDlg**

diagram



hierarchy



typedElements

Interface **IDialogs** 936
Interface **IDocument** 938
Operation **ImportSourceDirectoryDlg** 937
Operation **ImportSourceDirectory** 941

Operation **IImportSourceDirectoryDlg::Content_GenerateDiagramPerPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_GenerateSingleDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_HyperlinkPackagesToDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_ShowAnonymousBoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Content_ShowNestedClassifiersSeparately**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::Directory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportSourceDirectoryDlg::ImportDirectoriesRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDirectoryDlg::ProcessSubdirectories**

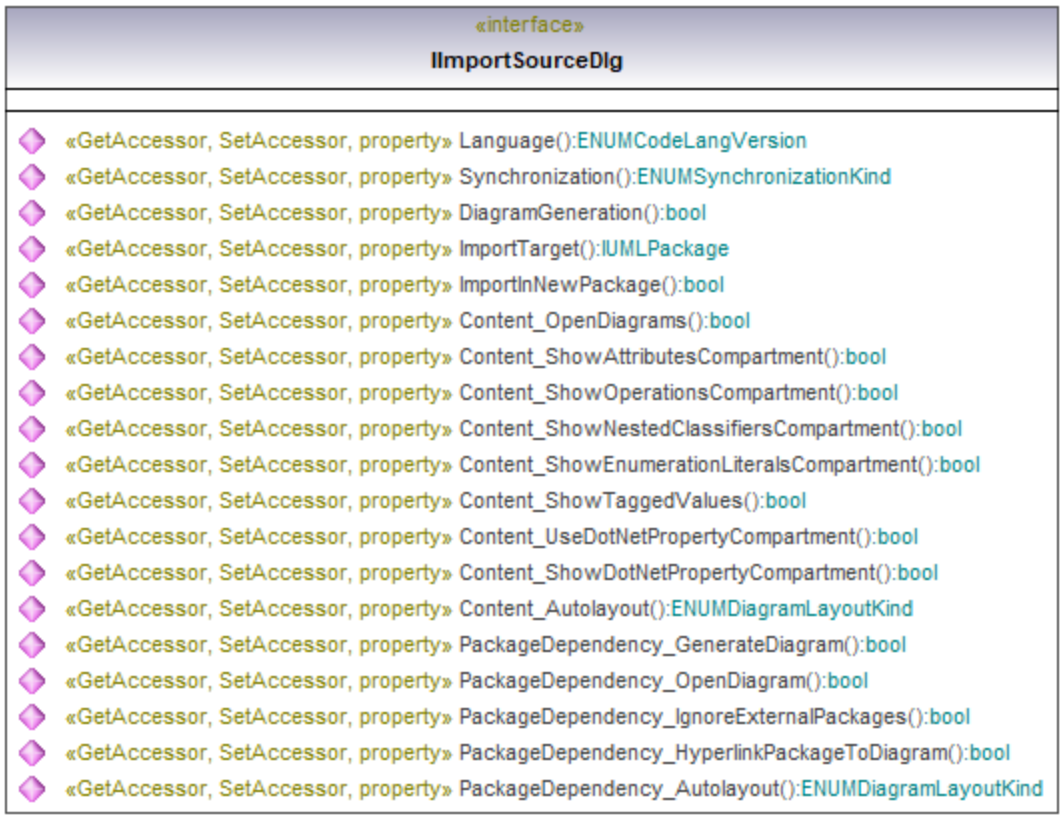
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

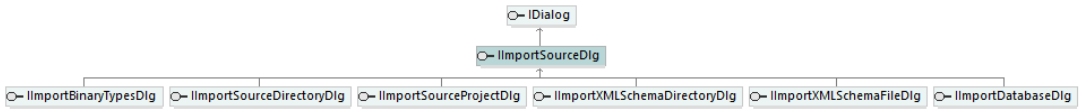
17.5.2.19 UModelAPI - IImportSourceDlg

Interface **IImportSourceDlg**

diagram



hierarchy



Operation **IImportSourceDlg::Content_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMDiagramLayoutKind 1005			

Operation **IImportSourceDlg::Content_OpenDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowAttributesCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowEnumerationLiteralsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowNestedClassifiersCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowOperationsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_ShowTaggedValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::Content_UseDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::DiagramGeneration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::ImportInNewPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::ImportTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

Operation **IImportSourceDlg::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion** 1005			

Operation **IImportSourceDlg::PackageDependency_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDiagramLayoutKind** 1005			

© 2016-2022 Altova GmbH

Operation **IImportSourceDlg::PackageDependency_GenerateDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::PackageDependency_HyperlinkPackageToDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::PackageDependency_IgnoreExternalPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceDlg::PackageDependency_OpenDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

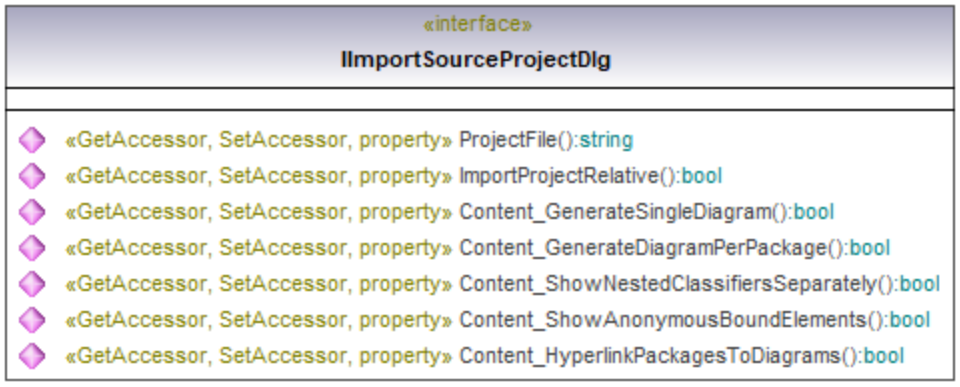
Operation **IImportSourceDlg::Synchronization**

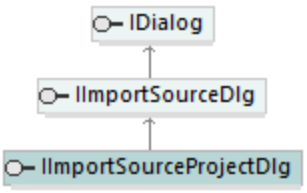
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMSynchronizationKind 1009			

17.5.2.20 UModelAPI - IImportSourceProjectDlg

Interface **IImportSourceProjectDlg**

diagram



hierarchy	
typed Elements	Interface **IDialogs** 936 Interface **IDocument** 938 Operation **ImportSourceProjectDlg** 937 Operation **ImportSourceProject** 941

Operation **IImportSourceProjectDlg::Content_GenerateDiagramPerPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_GenerateSingleDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_HyperlinkPackagesToDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_ShowAnonymousBoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::Content_ShowNestedClassifiersSeparately**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportSourceProjectDlg::ImportProjectRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

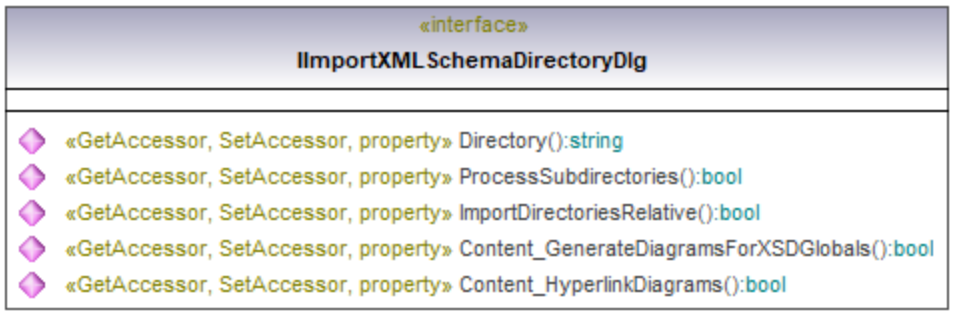
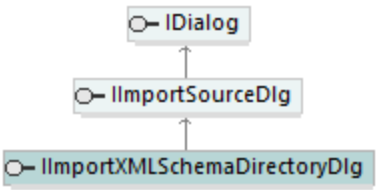
Operation **IImportSourceProjectDlg::ProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

© 2016-2022 Altova GmbH

17.5.2.21 UModelAPI - IImportXMLSchemaDirectoryDlg

Interface **IImportXMLSchemaDirectoryDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation ImportXMLSchemaDirectoryDlg 937 Operation ImportXMLSchemaDirectory 942

Operation **IImportXMLSchemaDirectoryDlg::Content_GenerateDiagramsForXSDGlobals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaDirectoryDlg::Content_HyperlinkDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaDirectoryDlg::Directory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IImportXMLSchemaDirectoryDlg::ImportDirectoriesRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

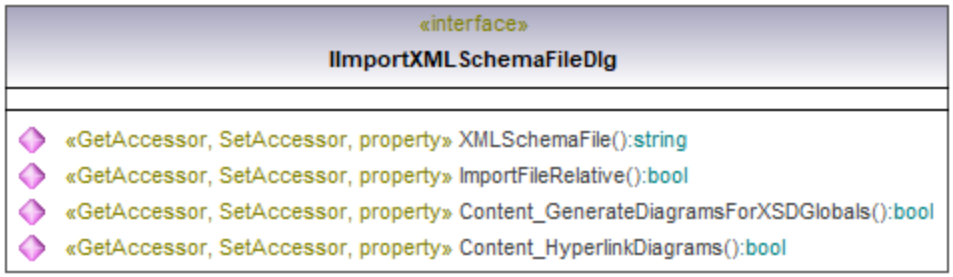
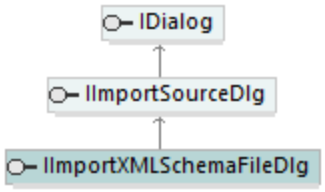
Operation **IImportXMLSchemaDirectoryDlg::ProcessSubdirectories**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2.22 UModelAPI - IImportXMLSchemaFileDlg

Interface **IImportXMLSchemaFileDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation ImportXMLSchemaFileDlg 937 Operation ImportXMLSchemaFile 942

Operation **IImportXMLSchemaFileDlg::Content_GenerateDiagramsForXSDGlobals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaFileDlg::Content_HyperlinkDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaFileDlg::ImportFileRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IImportXMLSchemaFileDlg::XMLSchemaFile**

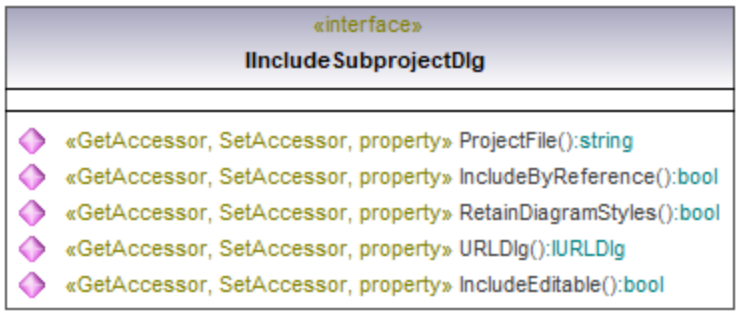
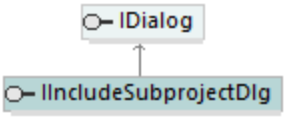
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

 © 2016-2022 Altova GmbH

17.5.2.23 UModelAPI - IIncludeSubprojectDlg

Interface **IIncludeSubprojectDlg**

diagram	
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation IncludeSubprojectDlg 937 Operation IncludeSubproject 942

Operation **IIncludeSubprojectDlg::IncludeByReference**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IIncludeSubprojectDlg::IncludeEditable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IIncludeSubprojectDlg::ProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IIncludeSubprojectDlg::RetainDiagramStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

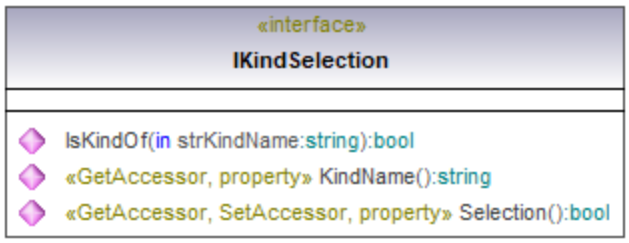
Operation **IIncludeSubprojectDlg::URLDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IURLDlg 997			

© 2016-2022 Altova GmbH

17.5.2.24 UModelAPI - IKindSelection

Interface **IKindSelection**

diagram	
typedElements	Interface IKindSelectionList 972 Operation Item 973

Operation **IKindSelection::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	**in**	**string**			
	return	**return**	**bool**			

Operation **IKindSelection::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IKindSelection::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.25 UModelAPI - IKindSelectionList

Interface **IKindSelectionList**

diagram	
typedElements	Interface IGenerateDocumentationDlg 947 Operation Include_Diagrams 952 Include_Elements 952

 © 2016-2022 Altova GmbH

Operation **IKindSelectionList::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IKindSelectionList::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IKindSelectionList::Item**

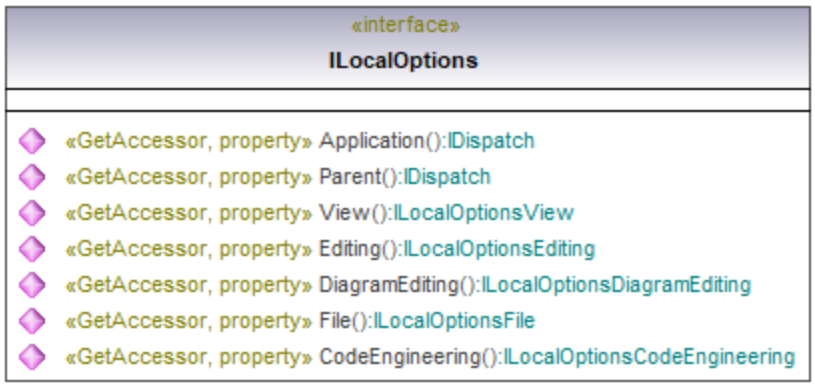
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IKindSelection** 972			

Operation **IKindSelectionList::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

17.5.2.26 UModelAPI - ILocalOptions

Interface **ILocalOptions**

diagram	
typedElements	Interface **IApplication** 923 Operation **Options** 925

Operation **ILocalOptions::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptions::CodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsCodeEngineering** 975			

Operation **ILocalOptions::DiagramEditing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsDiagramEditing** 977			

Operation **ILocalOptions::Editing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsEditing** 980			

Operation **ILocalOptions::File**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsFile** 981			

Operation **ILocalOptions::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

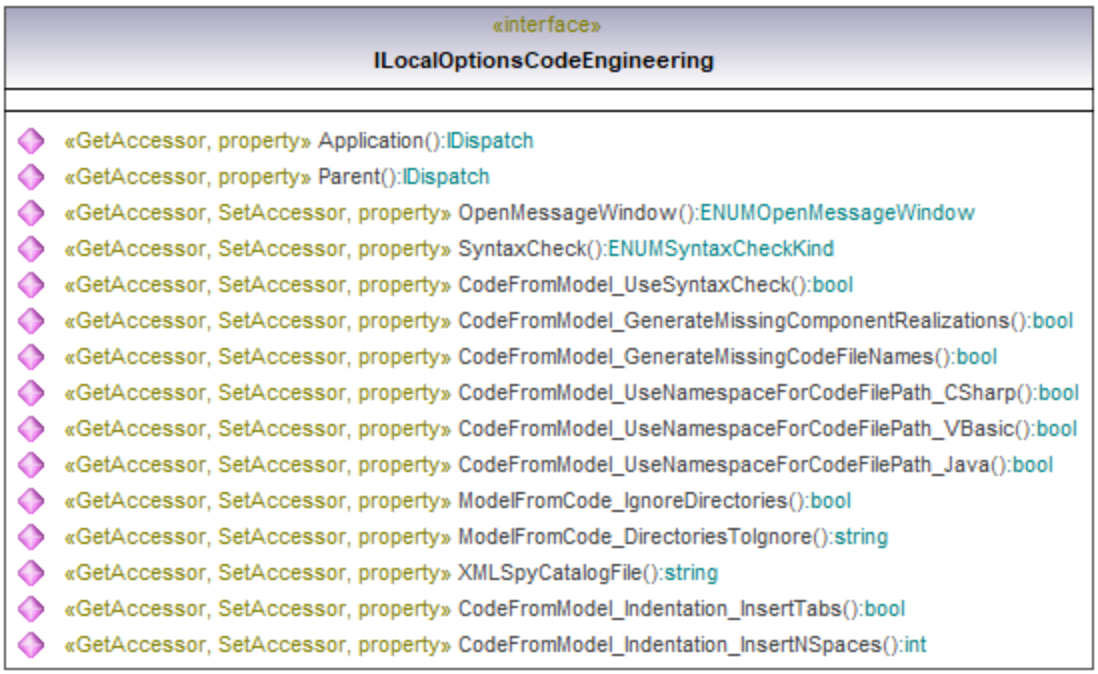
Operation **ILocalOptions::View**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsView** 983			

© 2016-2022 Altova GmbH

17.5.2.27 UModelAPI - ILocalOptionsCodeEngineering

Interface **ILocalOptionsCodeEngineering**

diagram	
typedElements	Interface ILocalOptions 973 Operation CodeEngineering 973

Operation **ILocalOptionsCodeEngineering::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_GenerateMissingCodeFileNames**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_GenerateMissingComponentRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_Indentation_InsertNSpaces**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_Indentation_InsertTabs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_CSharp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_Java**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_VBasic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseSyntaxCheck**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::ModelFromCode_DirectoriesToIgnore**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **ILocalOptionsCodeEngineering::ModelFromCode_IgnoreDirectories**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsCodeEngineering::OpenMessageWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMOpenMessageWindow** 1008			

Operation **ILocalOptionsCodeEngineering::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsCodeEngineering::SyntaxCheck**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSyntaxCheckKind** 1009			

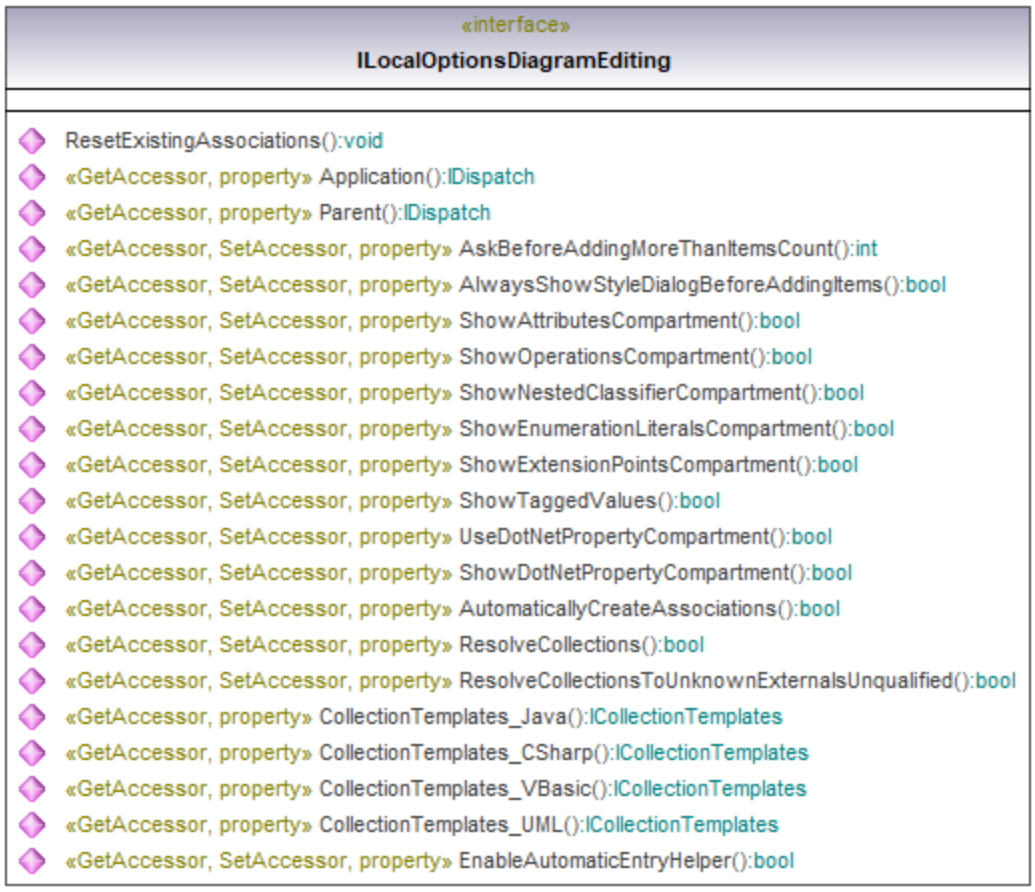
Operation **ILocalOptionsCodeEngineering::XMLSpyCatalogFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

© 2016-2022 Altova GmbH

17.5.2.28 UModelAPI - ILocalOptionsDiagramEditing

Interface **ILocalOptionsDiagramEditing**

diagram	
typedElements	Interface ILocalOptions 973 Operation DiagramEditing 974

Operation **ILocalOptionsDiagramEditing::AlwaysShowStyleDialogBeforeAddingItems**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsDiagramEditing::AskBeforeAddingMoreThanItemsCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsDiagramEditing::AutomaticallyCreateAssociations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_CSharp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates** 929			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_Java**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates** 929			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_UML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates** 929			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_VBasic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates** 929			

Operation **ILocalOptionsDiagramEditing::EnableAutomaticEntryHelper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsDiagramEditing::ResetExistingAssociations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **ILocalOptionsDiagramEditing::ResolveCollections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ResolveCollectionsToUnknownExternalsUnqualified**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowAttributesCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

Operation **ILocalOptionsDiagramEditing::ShowDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowEnumerationLiteralsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowExtensionPointsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowNestedClassifierCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowOperationsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsDiagramEditing::ShowTaggedValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

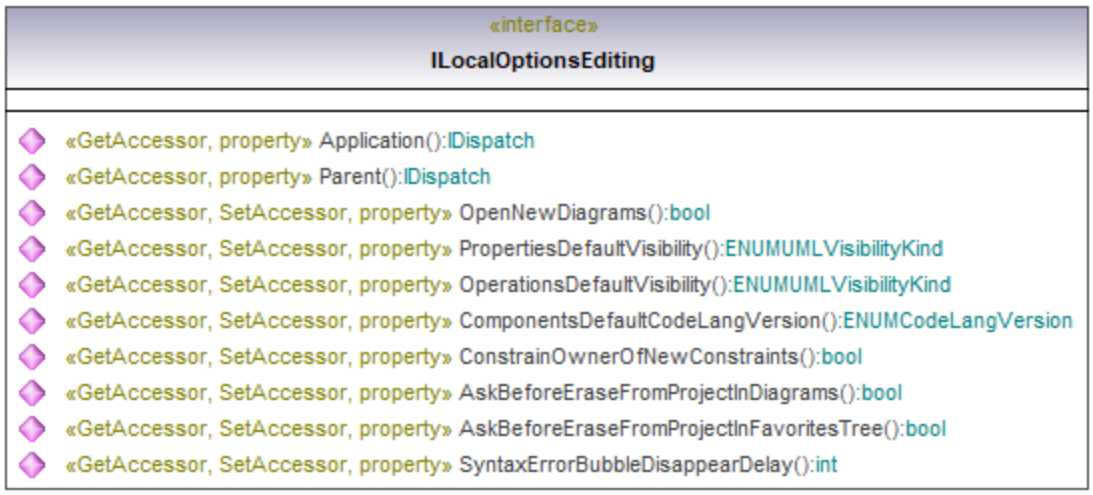
Operation **ILocalOptionsDiagramEditing::UseDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

17.5.2.29 UModelAPI - ILocalOptionsEditing

Interface **ILocalOptionsEditing**

diagram	
typedElements	Interface **ILocalOptions** 973 Operation **Editing** 974

Operation **ILocalOptionsEditing::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsEditing::AskBeforeEraseFromProjectInDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::AskBeforeEraseFromProjectInFavoritesTree**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::ComponentsDefaultCodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion** 1005			

Operation **ILocalOptionsEditing::ConstrainOwnerOfNewConstraints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsEditing::OpenNewDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

Operation **ILocalOptionsEditing::OperationsDefaultVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLVisibilityKind 1378			

Operation **ILocalOptionsEditing::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsEditing::PropertiesDefaultVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLVisibilityKind 1378			

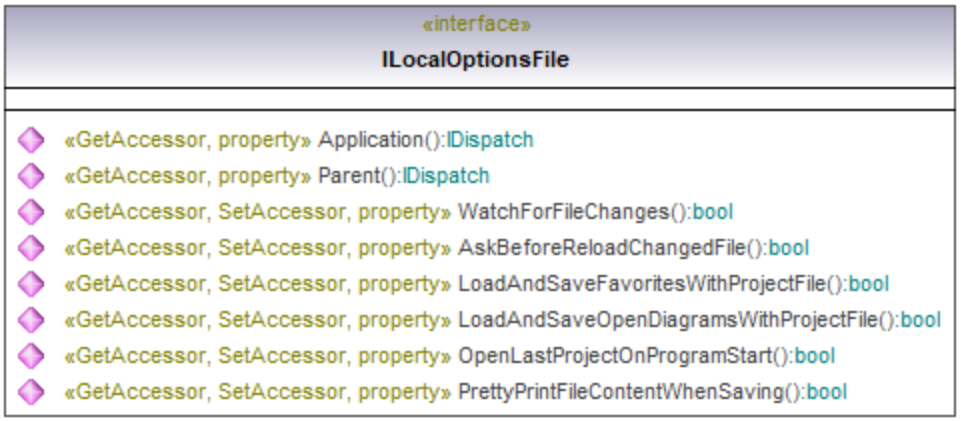
Operation **ILocalOptionsEditing::SyntaxErrorBubbleDisappearDelay**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.2.30 UModelAPI - ILocalOptionsFile

Interface **ILocalOptionsFile**

diagram



typedElements: Interface ILocalOptions 973 — Operation File 974

Operation **ILocalOptionsFile::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsFile::AskBeforeReloadChangedFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::LoadAndSaveFavoritesWithProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::LoadAndSaveOpenDiagramsWithProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::OpenLastProjectOnProgramStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsFile::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsFile::PrettyPrintFileContentWhenSaving**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

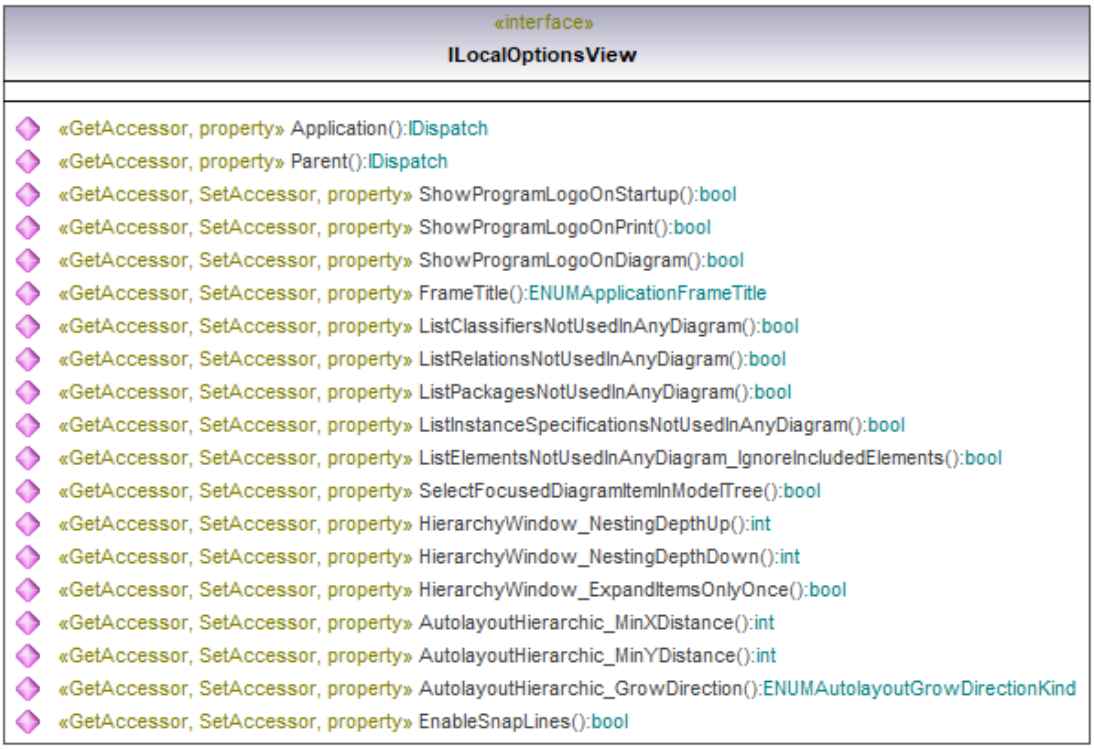
Operation **ILocalOptionsFile::WatchForFileChanges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

17.5.2.31 UModelAPI - ILocalOptionsView

Interface **ILocalOptionsView**

diagram	
typedElements	Interface ILocalOptions 973 Operation View 974

Operation **ILocalOptionsView::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsView::AutolayoutHierarchic_GrowDirection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMAutolayoutGrowDirectionKind 1004			

Operation **ILocalOptionsView::AutolayoutHierarchic_MinXDistance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::AutolayoutHierarchic_MinYDistance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::EnableSnapLines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::FrameTitle**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMApplicationFrameTitle** 1003			

Operation **ILocalOptionsView::HierarchyWindow_ExpandItemsOnlyOnce**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::HierarchyWindow_NestingDepthDown**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::HierarchyWindow_NestingDepthUp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **ILocalOptionsView::ListClassifiersNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListElementsNotUsedInAnyDiagram_IgnoreIncludedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListInstanceSpecificationsNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListPackagesNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ListRelationsNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsView::SelectFocusedDiagramItemInModelTree**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ShowProgramLogoOnDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ShowProgramLogoOnPrint**

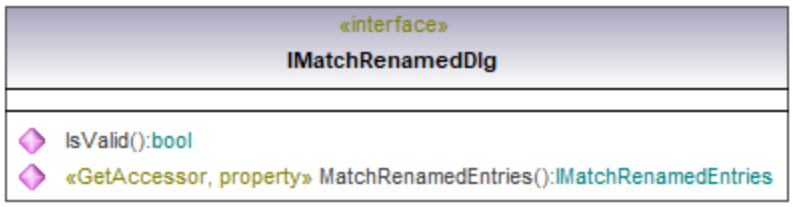
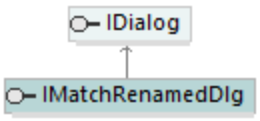
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **ILocalOptionsView::ShowProgramLogoOnStartup**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.2.32 UModelAPI - IMatchRenamedDlg

Interface **IMatchRenamedDlg**

diagram	
hierarchy	
typedElements	Interface _ISynchronizationEvents 1002 Operation OnMatchRenamed 1002

Operation **IMatchRenamedDlg::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

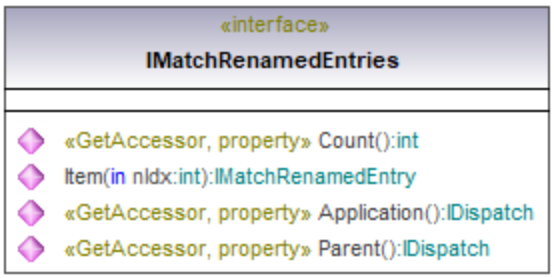
Operation **IMatchRenamedDlg::MatchRenamedEntries**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IMatchRenamedEntries** 986

17.5.2.33 UModelAPI - IMatchRenamedEntries

Interface **IMatchRenamedEntries**

diagram	
typedElements	Interface **IMatchRenamedDlg** 985 Operation **MatchRenamedEntries** 985

Operation **IMatchRenamedEntries::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IMatchRenamedEntries::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IMatchRenamedEntries::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IMatchRenamedEntry** 987			

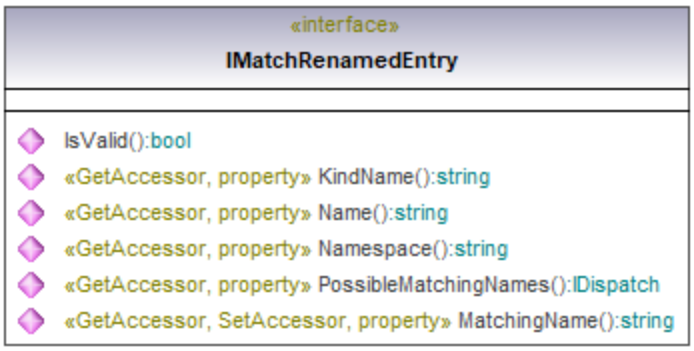
Operation **IMatchRenamedEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

 © 2016-2022 Altova GmbH

17.5.2.34 UModelAPI - IMatchRenamedEntry

Interface **IMatchRenamedEntry**

diagram	
typedElements	Interface IMatchRenamedEntries 986 Operation Item 986

Operation **IMatchRenamedEntry::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IMatchRenamedEntry::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::MatchingName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IMatchRenamedEntry::PossibleMatchingNames**

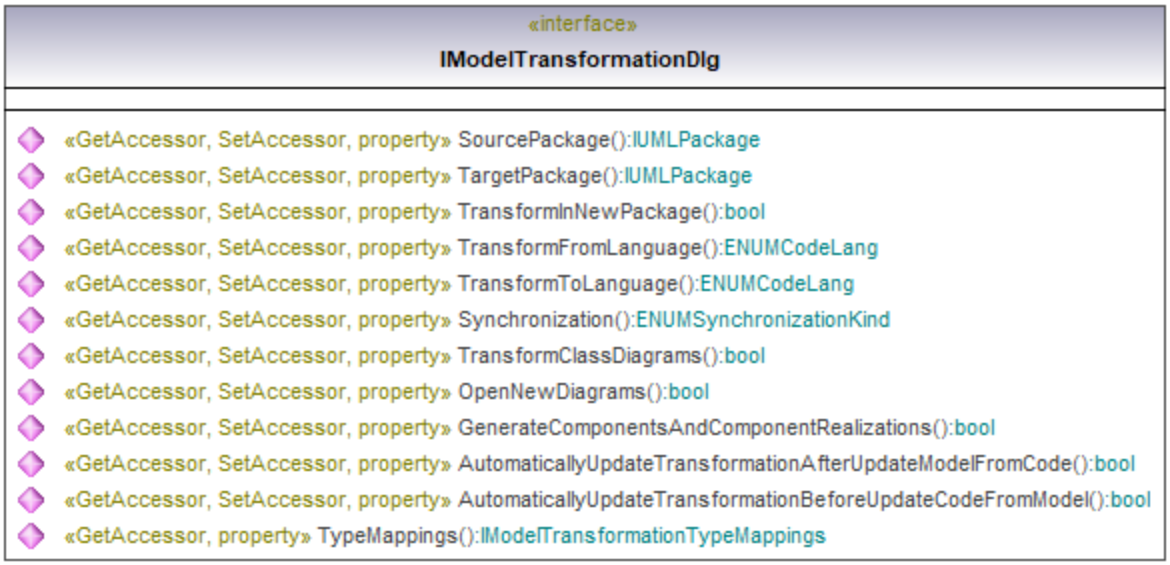
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			
documentation	Returns an array of values of type **string**.					

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

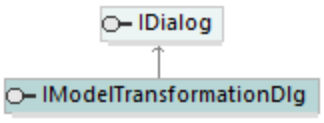
17.5.2.35 UModelAPI - IModelTransformationDlg

Interface **IModelTransformationDlg**

diagram



hierarchy



typedElements

Interface IDialogs 936
Interface IDocument 938
Operation ModelTransformationDlg 937
Operation ModelTransformation 943

Operation **IModelTransformationDlg::AutomaticallyUpdateTransformationAfterUpdateModelFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::AutomaticallyUpdateTransformationBeforeUpdateCodeFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::GenerateComponentsAndComponentRealizations**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2016-2022 Altova GmbH

	return	**return**	**bool**

Operation **IModelTransformationDlg::OpenNewDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::SourcePackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IModelTransformationDlg::Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMSynchronizationKind 1009			

Operation **IModelTransformationDlg::TargetPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IModelTransformationDlg::TransformClassDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::TransformFromLanguage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMCodeLang 1004			

Operation **IModelTransformationDlg::TransformInNewPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IModelTransformationDlg::TransformToLanguage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMCodeLang 1004			

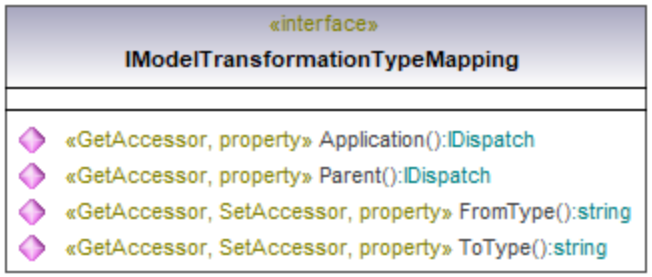
Operation **IModelTransformationDlg::TypeMappings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IModelTransformationTypeMappings 991			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2.36 UModelAPI - IModelTransformationTypeMapping

Interface **IModelTransformationTypeMapping**

diagram	
typedElements	Interface IModelTransformationTypeMappings 991 Operation InsertItemAt 991 Item 991

Operation **IModelTransformationTypeMapping::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IModelTransformationTypeMapping::FromType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IModelTransformationTypeMapping::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IModelTransformationTypeMapping::ToType**

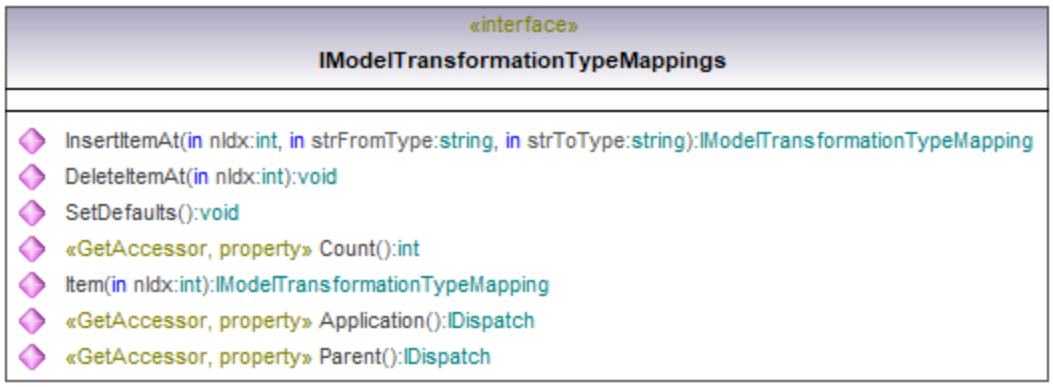
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

 © 2016-2022 Altova GmbH

17.5.2.37 UModelAPI - IModelTransformationTypeMappings

Interface **IModelTransformationTypeMappings**

diagram	
typedElements	Interface IModelTransformationDlg 988 Operation TypeMappings 989

Operation **IModelTransformationTypeMappings::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IModelTransformationTypeMappings::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IModelTransformationTypeMappings::DeleteItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IModelTransformationTypeMappings::InsertItemAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFromType	**in**	**string**			
	strToType	**in**	**string**			
	return	**return**	IModelTransformationTypeMapping 990			

Operation **IModelTransformationTypeMappings::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IModelTransformationTypeMapping 990			

Operation **IModelTransformationTypeMappings::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

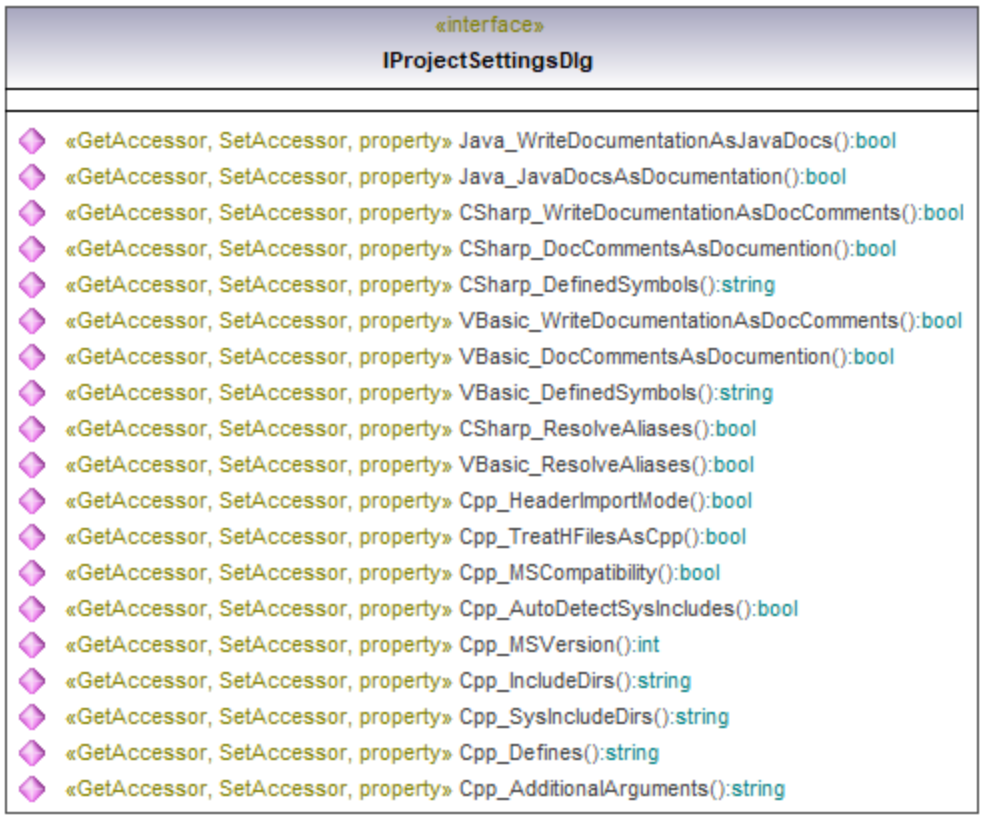
Operation **IModelTransformationTypeMappings::SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

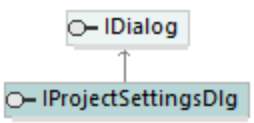
17.5.2.38 UModelAPI - IProjectSettingsDlg

Interface **IProjectSettingsDlg**

diagram



hierarchy



 © 2016-2022 Altova GmbH

typedElements	Interface IDialogs 936 Interface IDocument 938	Operation ProjectSettingsDlg 938 Operation ProjectSettings 943

Operation **IProjectSettingsDlg::Cpp_AdditionalArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_AutoDetectSysIncludes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Cpp_Defines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_HeaderImportMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Cpp_IncludeDirs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_MSCompatibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Cpp_MSVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IProjectSettingsDlg::Cpp_SysIncludeDirs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::Cpp_TreatHFilesAsCpp**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::CSharp_DefinedSymbols**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::CSharp_DocCommentsAsDocumention**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**bool**

Operation **IProjectSettingsDlg::CSharp_ResolveAliases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::CSharp_WriteDocumentationAsDocComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Java_JavaDocsAsDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::Java_WriteDocumentationAsJavaDocs**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::VBasic_DefinedSymbols**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IProjectSettingsDlg::VBasic_DocCommentsAsDocumention**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IProjectSettingsDlg::VBasic_ResolveAliases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

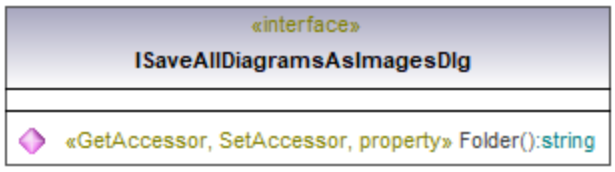
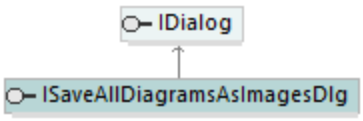
Operation **IProjectSettingsDlg::VBasic_WriteDocumentationAsDocComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

17.5.2.39 UModelAPI - ISaveAllDiagramsAsImagesDlg

Interface **ISaveAllDiagramsAsImagesDlg**

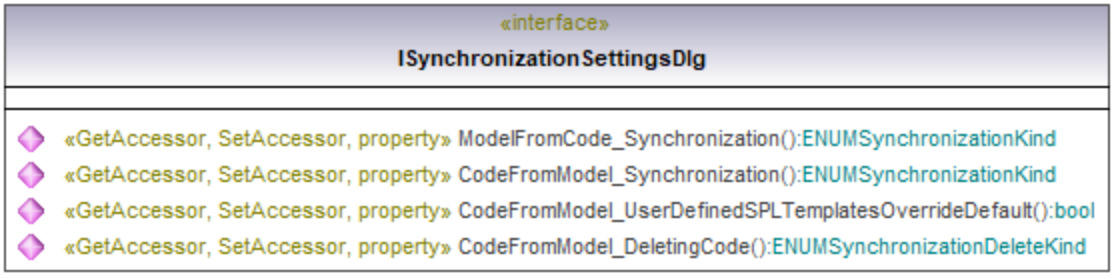
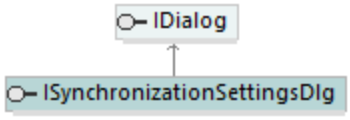
diagram	
hierarchy	
typedElements	Interface IDialogs 936 Interface IDocument 938 Operation SaveAllDiagramsAsImagesDlg 938 Operation SaveAllDiagramsAsImages 944

Operation **ISaveAllDiagramsAsImagesDlg::Folder**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.2.40 UModelAPI - ISynchronizationSettingsDlg

Interface **ISynchronizationSettingsDlg**

diagram	
hierarchy	

typedElements	Interface IDialogs 936 Interface IDocument 938	Operation SynchronizationSettingsDlg 938 Operation SynchronizationSettings 945 SynchronizeCodeFromModel 945 SynchronizeModelFromCode 945

Operation **ISynchronizationSettingsDlg::CodeFromModel_DeletingCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMSynchronizationDeleteKind 1008			

Operation **ISynchronizationSettingsDlg::CodeFromModel_Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMSynchronizationKind 1009			

Operation **ISynchronizationSettingsDlg::CodeFromModel_UserDefinedSPLTemplatesOverrideDefault**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

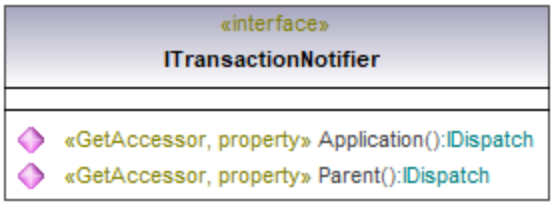
Operation **ISynchronizationSettingsDlg::ModelFromCode_Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMSynchronizationKind 1009			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2.41 UModelAPI - ITransactionNotifier

Interface **ITransactionNotifier**

diagram	
typedElements	Interface IDocument 938 Operation TransactionNotifier 945
documentation	Use this interface to register for _ITransactionEvents 1002.

Operation **ITransactionNotifier::Application**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**IDispatch**

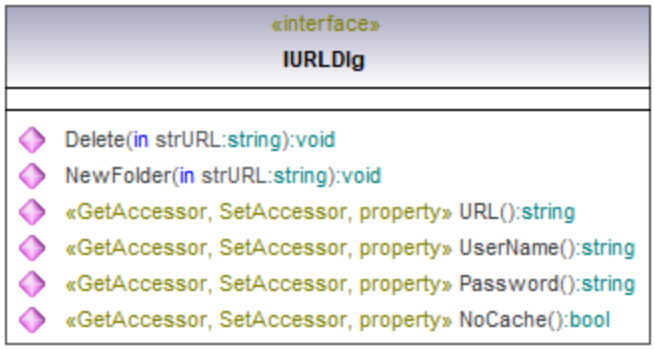
Operation **ITransactionNotifier::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

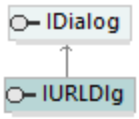
17.5.2.42 UModelAPI - IURLDlg

Interface **IURLDlg**

diagram



hierarchy



typedElements

Interface **IApplication** 923
Interface **IDialogs** 936
Interface **IDocument** 938
Interface **IExportXMIFileDlg** 945
Interface **IIncludeSubprojectDlg** 971
Operation **ImportFromXMIFileFromURL** 924
OpenDocumentFromURL 925
Operation **URLOpenDialog** 936
URLSaveDialog 938
Operation **MergeProjectFromURL** 943
SaveToURL 944
Operation **URLDlg** 946
Operation **URLDlg** 971

Operation **IURLDlg::Delete**

parameter	name	direction	type	type modifier	multiplicity	default
	strURL	**in**	**string**			
	return	**return**	**void**			

Operation **IURLDlg::NewFolder**

parameter	name	direction	type	type modifier	multiplicity	default
	strURL	**in**	**string**			
	return	**return**	**void**			

Operation **IURLDlg::NoCache**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IURLDlg::Password**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IURLDlg::URL**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IURLDlg::UserName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.2.43 Events

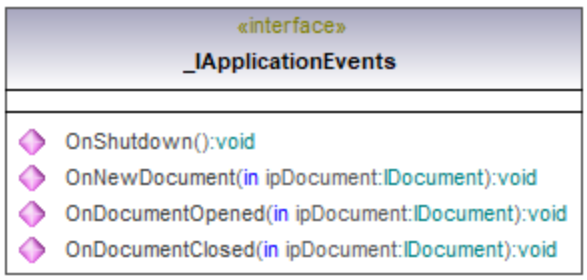
Hier finden Sie eine Liste aller Events, die von der UModel API gesendet werden.

Hier[1367] finden Sie eine Liste der Events, die von der UMLData-Ebene gesendet werden.

17.5.2.43.1 UModelAPI - _IApplicationEvents

Interface **_IApplicationEvents**

diagram	



Operation **_IApplicationEvents::OnDocumentClosed**

 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	IDocument 938			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnDocumentOpened**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	IDocument 938			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnNewDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	IDocument 938			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnShutdown**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

17.5.2.43.2 UModelAPI - _IDiagramWindowEvents

Interface **_IDiagramWindowEvents**

diagram



Operation **_IDiagramWindowEvents::OnDiagramWindowClosed**

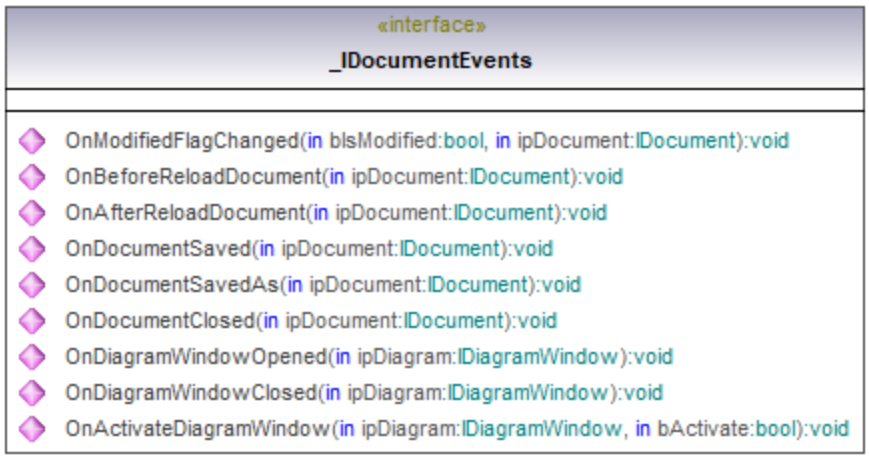
parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	IDiagramWindow 931			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.2.43.3 UModelAPI - _IDocumentEvents

Interface **_IDocumentEvents**

diagram



Operation **_IDocumentEvents::OnActivateDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	IDiagramWindow 931			
	bActivate	**in**	**bool**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnAfterReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	IDocument 938			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnBeforeReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	IDocument 938			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	IDiagramWindow 931			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDiagramWindowOpened**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	IDiagramWindow 931			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentClosed**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument** 938			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentSaved**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument** 938			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentSavedAs**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument** 938			
	return	**return**	**void**			

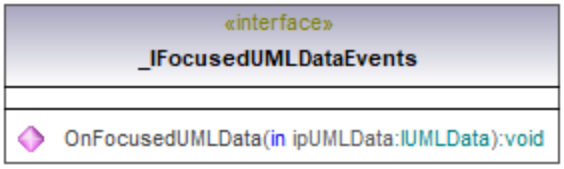
Operation **_IDocumentEvents::OnModifiedFlagChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	bIsModified	**in**	**bool**			
	ipDocument	**in**	**IDocument** 938			
	return	**return**	**void**			

17.5.2.43.4 UModelAPI - _IFocusedUMLDataEvents

Interface **_IFocusedUMLDataEvents**

diagram	



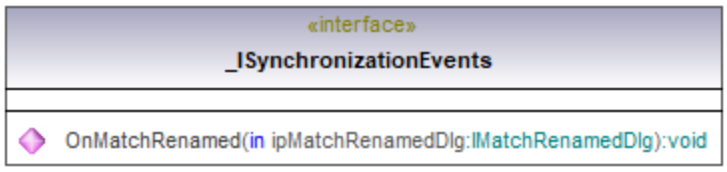
Operation **_IFocusedUMLDataEvents::OnFocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData** 1011			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.2.43.5 UModelAPI - _ISynchronizationEvents

Interface **_ISynchronizationEvents**

diagram	

Operation **_ISynchronizationEvents::OnMatchRenamed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipMatchRenamedDlg	**in**	**IMatchRenamedDlg** 985			
	return	**return**	**void**			

17.5.2.43.6 UModelAPI - _ITransactionEvents

Interface **_ITransactionEvents**

diagram	
documentation	Use ITransactionNotifier 996 to register for these events

Operation **_ITransactionEvents::OnBeginDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument** 938			
	return	**return**	**void**			

Operation **_ITransactionEvents::OnEndDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument** 938			
	return	**return**	**void**			

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.44 Enumerations

Dies ist einer Liste aller von der UModel API verwendeten Enumerationen. Wenn Ihre Skripting-Umgebung Enumerationen nicht unterstützt, verwenden Sie statt dessen die Zahlenwerte.

Eine Liste der auf der UMLData-Ebene definierten Enumerationen finden Sie hier[1369].

17.5.2.44.1 UModelAPI - ENUMApplicationFrameTitle

Enumeration **ENUMApplicationFrameTitle**

diagram	«enumeration» **ENUMApplicationFrameTitle** eApplicationFrameTitle_FileNameOnly = 0 eApplicationFrameTitle_FullPathName = 1
typedElements	Interface **ILocalOptionsView**[983] Operation **FrameTitle**[984]

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.44.2 UModelAPI - ENUMApplicationStatus

Enumeration **ENUMApplicationStatus**

diagram	«enumeration» **ENUMApplicationStatus** eApplicationRunning = 0 eApplicationAfterLicenseCheck = 1 eApplicationBeforeLicenseCheck = 2 eApplicationConcurrentLicenseCheckFailed = 3 eApplicationProcessingCommandLine = 4
typedElements	Interface **IApplication**[923] Operation **Status**[926]

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2.44.3 UModelAPI - ENUMAutolayoutGrowDirectionKind

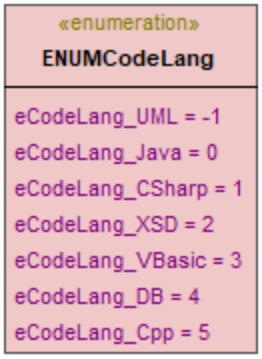
Enumeration **ENUMAutolayoutGrowDirectionKind**

diagram	
typedElements	Interface ILocalOptionsView 983 Operation AutolayoutHierarchic_GrowDirection 983

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.2.44.4 UModelAPI - ENUMCodeLang

Enumeration **ENUMCodeLang**

diagram	
typedElements	Interface IModelTransformationDlg 988 Interface IUMLComponent 1135 Interface IUMLDataAll 1019 Interface IUMLPackage 1238 Operation TransformFromLanguage 989 TransformToLanguage 989 Operation CodeLang 1135 Operation CodeLang 1025 IsCodeLangNamespace 1054 IsCodeLangNamespaceRoot 1054 Operation IsCodeLangNamespace 1240 IsCodeLangNamespaceRoot 1240

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.44.5 UModelAPI - ENUMCodeLangVersion

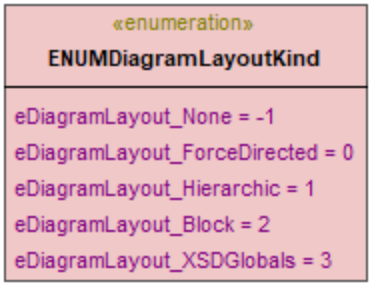
Enumeration **ENUMCodeLangVersion**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **IImportSourceDlg** 965 Interface **ILocalOptionsEditing** 980 Interface **IUMLComponent** 1135 Interface **IUMLDataAll** 1019 Operation **Language** 966 Operation **ComponentsDefaultCodeLangVersion** 980 Operation **CodeLangVersion** 1135 Operation **CodeLangVersion** 1025

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.44.6 UModelAPI - ENUMDiagramLayoutKind

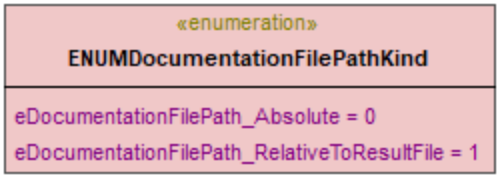
Enumeration **ENUMDiagramLayoutKind**

diagram	
typedElements	Interface **IDiagramWindow** 931 Interface **IImportSourceDlg** 965 Operation **Autolayout** 931 **AutolayoutSelection** 932 Operation **Content_Autolayout** 965 **PackageDependency_Autolayout** 966

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

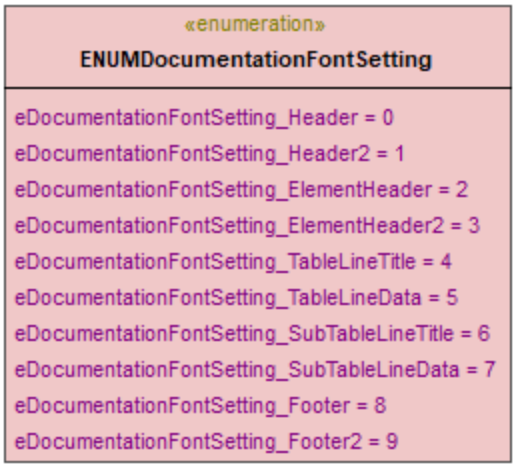
17.5.2.44.7 UModelAPI - ENUMDocumentationFilePathKind

Enumeration **ENUMDocumentationFilePathKind**

diagram	
typedElements	Interface IGenerateDocumentationDlg 947 Operation GenerateLinksToLocalFiles 952

17.5.2.44.8 UModelAPI - ENUMDocumentationFontSetting

Enumeration **ENUMDocumentationFontSetting**

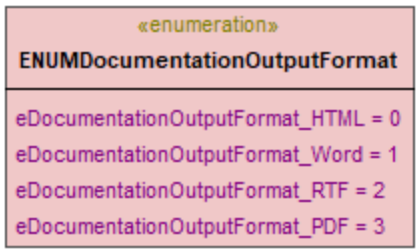
diagram	
typedElements	Interface IGenerateDocumentationDlg 947 Operation Fonts_GetFace 950 Fonts_GetSize 950 Fonts_GetTextColor 950 Fonts_IsBold 950 Fonts_IsItalic 951 Fonts_IsUnderline 951 Fonts_SetBold 951 Fonts_SetFace 951 Fonts_SetItalic 951 Fonts_SetSize 951 Fonts_SetTextColor 952 Fonts_SetUnderline 952

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.44.9 UModelAPI - ENUMDocumentationOutputFormat

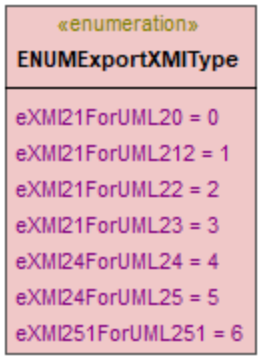
Enumeration **ENUMDocumentationOutputFormat**

diagram	
typedElements	Interface **IGenerateDocumentationDlg** 947 Operation **OutputFormat** 953

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.2.44.10 UModelAPI - ENUMExportXMIType

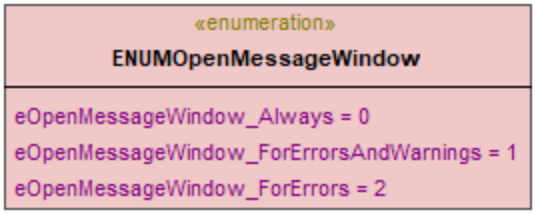
Enumeration **ENUMExportXMIType**

diagram	
typedElements	Interface **IExportXMIFileDlg** 945 Operation **XMIType** 946

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

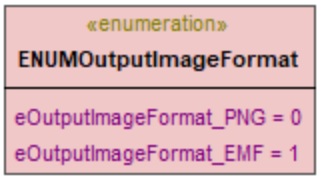
17.5.2.44.11 UModelAPI - ENUMOpenMessageWindow

Enumeration **ENUMOpenMessageWindow**

diagram	
typedElements	Interface **ILocalOptionsCodeEngineering** 975 Operation **OpenMessageWindow** 976

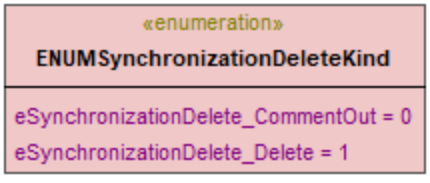
17.5.2.44.12 UModelAPI - ENUMOutputImageFormat

Enumeration **ENUMOutputImageFormat**

diagram	
typedElements	Interface **IDiagramWindow** 931 Operation **SaveDiagramAsImage** 932 Interface **IGenerateDocumentationDlg** 947 Operation **DiagramImageFormat** 950

17.5.2.44.13 UModelAPI - ENUMSynchronizationDeleteKind

Enumeration **ENUMSynchronizationDeleteKind**

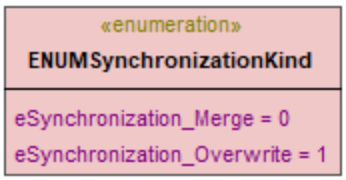
diagram	

 © 2016-2022 Altova GmbH

typedElements	Interface ISynchronizationSettingsDlg 995	Operation CodeFromModel_DeletingCode 996

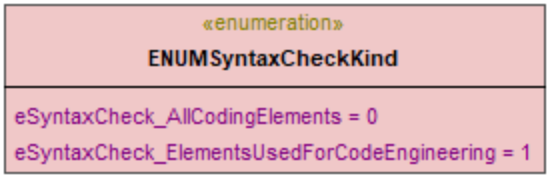
17.5.2.44.14 UModelAPI - ENUMSynchronizationKind

Enumeration **ENUMSynchronizationKind**

diagram	
typedElements	Interface IImportSourceDlg 965 Operation Synchronization 967 Interface IModelTransformationDlg 988 Operation Synchronization 989 Interface ISynchronizationSettingsDlg 995 Operation CodeFromModel_Synchronization 996 ModelFromCode_Synchronization 996

17.5.2.44.15 UModelAPI - ENUMSyntaxCheckKind

Enumeration **ENUMSyntaxCheckKind**

diagram	
typedElements	Interface ILocalOptionsCodeEngineering 975 Operation SyntaxCheck 976

© 2016-2022 Altova GmbH

17.5.3 UMLData-Schnittstellen

Über die UMLData-Schnittstellen haben Sie direkten Zugriff auf ein Dokument auf UML-Ebene. Sie können die UML-Darstellung des Dokuments lesen und direkt ändern.

IUMLData[1010] ist die allgemeine Basisschnittstelle von IUMLElement[1090] und IUMLGuiElement[1304].

IUMLElements[1090] enthält Elemente, die in der von der OMG definierten "UML Speciification" definiert sind (siehe auch http://www.uml.org)

IUMLGuiElements[1304] enthält Elemente von Altova, die für Diagramme und zum Anzeigen von IUMLElements[1090] in Diagrammen verwendet werden.

Informationen zum Lesen und Ändern von UML- und GUI-Elementen finden Sie auch im Kapitel Objektmodell UMLData[860].

Fehler

Die `IUMLData`-Schnittstelle kann die unten aufgelisteten API-Fehlercodes zurückgeben.

1000	Das application-Objekt ist nicht mehr gültig.
1001	Für den Rückgabeparameter wurde ein ungültiger Parameter oder eine ungültige Adresse definiert.
1002	Die UModel API steht in der aktuellen Version nicht zur Verfügung.
1400	Ungültige UMLData-Änderung.
1401	Ungültige Waypoint-Änderung.
1402	Es sind keine Änderungen zulässig.
1403	Während Rückgängig/Wiederholen sind keine Änderungen zulässig.
1404	Das Element wird durch einen Elementstil (Sichtbarkeit) ausgeblendet.
1405	Das vordefinierte Element wurde nicht gefunden.
1406	Das vordefinierte Element ist von einer ungültigen Art.

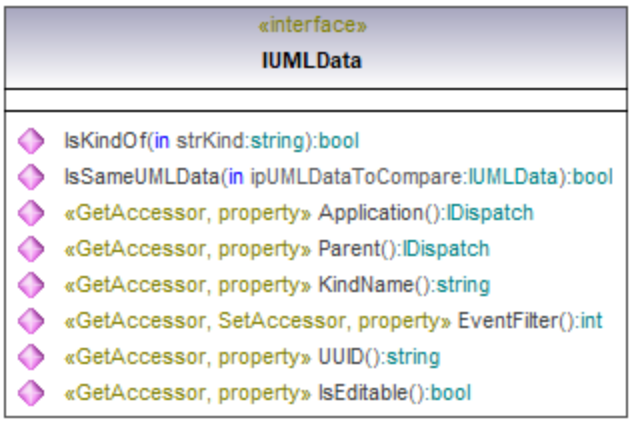
Fehler der UModel API finden Sie hier[921].

© 2016-2022 Altova GmbH

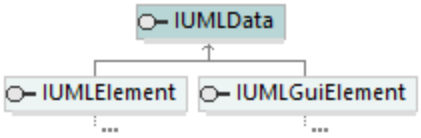
17.5.3.1 UModelAPI - IUMLData

Interface **IUMLData**

diagram



hierarchy



typedElements

Interface _IFocusedUMLDataEvents 1001
Interface _IUMLDataEvents 1368
Interface IApplication 923
Interface IDiagramWindow 931
Interface IDocument 938
Interface IUMLCommentTextHyperlink 1133
Interface IUMLData 1011
Interface IUMLDataAll 1019
Interface IUMLDataList 1013
Interface IUMLElement 1157

Operation OnFocusedUMLData 1001
Operation OnAfterAddChild 1368
OnBeforeErase 1368 OnChanged 1368
OnMoveData 1368
Operation LogMessageWithUMLDataLink 924
Operation FocusedData 932
Operation CanFocusUMLDataInModelTree 939
FocusedUMLData 940
FocusUMLDataInModelTree 940
Operation SetHyperlinkModelElementAddress 1134
Operation IsSameUMLData 1012
Operation AddUMLGuiNodeLink 1021
FindPredefinedOwnedElement 1034
InsertOwnedDiagramAt 1048
InsertOwnedHyperlink2ModelAt 1048 IsSameUMLData 1057
LinkedModelElement 1060
SetHyperlinkModelElementAddress 1075
Operation ContainsUMLData 1018 Item 1018
Operation FindPredefinedOwnedElement 1158

	Interface IUMLGuiDiagram 1316 Interface IUMLGuiRootElement 1338 Interface IUMLGuiTextHyperlink 1355 Interface IUMLHyperlink2Model 1184 Interface IUMLNamedElement 1222	Operation AddUMLGuiNodeLink 1317 Operation InsertOwnedDiagramAt 1339 Operation SetHyperlinkModelElementAddress 1356 Operation LinkedModelElement 1184 Operation InsertOwnedHyperlink2ModelAt 1223

Operation **IUMLData::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLData::EventFilter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLData::IsEditable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLData::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**bool**			

Operation **IUMLData::IsSameUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLDataToCompare	**in**	IUMLData 1011			
	return	**return**	**bool**			

Operation **IUMLData::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLData::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLData::UUID**

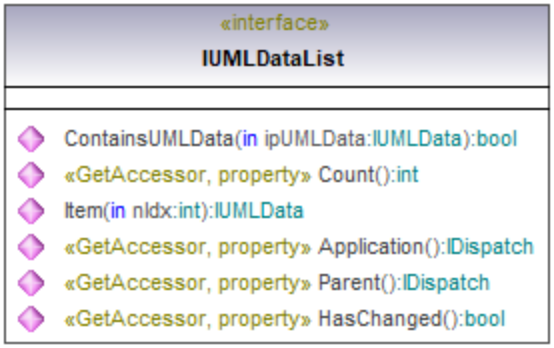
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

© 2016-2022 Altova GmbH

17.5.3.2 UModelAPI - IUMLDataList

Interface **IUMLDataList**

diagram



typedElements

Interface IDiagramWindow 931
Operation SelectedGuiElements 933

Interface IUMLAcceptEventAction 1091
Operation ActionTriggers 1091
EventActionResults 1091

Interface IUMLAction 1092
Operation InputPins 1093
LocalPostConditions 1093
LocalPreConditions 1093
OutputPins 1093

Interface IUMLActivity 1095
Operation ActivityEdges 1096
ActivityGroups 1096
ActivityNodes 1096

Interface IUMLActivityEdge 1097
Operation ActivityPartitions 1098
InterruptibleActivityRegions 1098
StructuredActivityNodes 1098

Interface IUMLActivityGroup 1099
Operation ContainedEdges 1100
ContainedNodes 1100
SubGroups 1100

Interface IUMLActivityNode 1101
Operation IncomingEdges 1101
OutgoingEdges 1102

Interface IUMLActivityPartition 1103
Operation Edges 1103 Nodes 1104
SubPartitions 1105

Interface IUMLArtifact 1106
Operation Manifestations 1107
NestedArtifacts 1107
OwnedAttributes 1108
OwnedOperations 1108

Interface IUMLAssociation 1108
Operation EndTypes 1109 MemberEnds 1109
NavigableOwnedEnds 1109
OwnedEnds 1109

Interface IUMLBehavior 1110
Operation OwnedParameters 1111
Postconditions 1111
Preconditions 1112

Interface IUMLBehavioralFeature 1112
Operation Methods 1113
OwnedParameters 1113
RaisedExceptions 1113

Interface IUMLBehavioredClassifier 1114
Operation InterfaceRealizations 1115
OwnedBehaviors 1115

Interface IUMLCallAction 1116
Operation Results 1117

Interface IUMLClass 1121
Operation NestedClassifiers 1123
OwnedOperations 1123

OwnedReceptions 1123
SuperClasses 1123
Interface IUMLClassifier 1124 Operation Attributes 1125
CollaborationUses 1125
Features 1125 Generalizations 1125
Generals 1126
InheritedMembers 1126
OwnedUseCases 1127 Specifics 1127
UseCases 1127
Interface IUMLClassifierTemplateParameter 1127 Operation ConstrainingClassifiers 1128
Interface IUMLCollaboration 1129 Operation CollaborationRoles 1129
Interface IUMLCollaborationUse 1130 Operation RoleBindings 1130
Interface IUMLCombinedFragment 1131 Operation Operands 1131
Interface IUMLComment 1132 Operation AnnotatedElements 1132
OwnedHyperlinks 1133
Interface IUMLComponent 1135 Operation Realizations 1136
Interface IUMLConnectionPointReference 1138 Operation Entries 1139 Exits 1139
Interface IUMLConstraint 1142 Operation ConstrainedElements 1143
Interface IUMLDataAll 1019 Operation ActionTriggers 1019
ActivityEdges 1020
ActivityGroups 1020
ActivityNodes 1020
ActivityPartitions 1020
ActualGates 1020
AllAppliableStereotypes 1022
AllWaypoints 1022
AnnotatedElements 1022
AppliedStereotypes 1022
Arguments 1023 AttachedNodes 1023
Attributes 1023
ClientDependencies 1025
Clients 1025 CollaborationRoles 1025
CollaborationUses 1026
ConnectionPoints 1026
Connections 1026
ConstrainedElements 1026
ConstrainingClassifiers 1026
ContainedEdges 1027
ContainedNodes 1027 Conveyed 1027
DeployedElements 1028
Deployments 1028 Edges 1029
ElementImports 1029 EndTypes 1029
Entries 1029 EventActionResults 1032
ExceptionHandlers 1033
ExceptionTypes 1033 Exits 1033
Extends 1034
ExtensionLocations 1034
ExtensionPoints 1034 Features 1034
FeaturingClassifiers 1034
FormalGates 1035 Fragments 1035
Generalizations 1035 Generals 1035

 © 2016-2022 Altova GmbH

GetOwnedElementsOfKind 1036
GuiLinks 1037 Handlers 1037
ImportedMembers 1038
Includes 1038 IncomingEdges 1038
Incomings 1039
InformationFlowRealizations 1039
InformationSources 1039
InformationTargets 1039
InheritedMembers 1039
InputElements 1039 InputPins 1039
InputValues 1039 InStates 1053
InterfaceRealizations 1053
InterruptibleActivityRegions 1053
InterruptingEdges 1054 Layers 1059
Lifelines 1060
LineConnectionWaypoints 1060
LineLinks 1060
LocalPostConditions 1061
LocalPreConditions 1061
LowerValues 1061
Manifestations 1061
MemberEnds 1061 Members 1061
Messages 1062 Methods 1062
NavigableOwnedEnds 1063
NestedArtifacts 1063
NestedClassifiers 1063
NestedNodes 1064
NestedPackages 1064 Nodes 1064
Observations 1064 Operands 1065
OutgoingEdges 1065 Outgoings 1065
OutputElements 1065
OutputPins 1065 OutputValues 1065
OwnedArguments 1066
OwnedAttributes 1066
OwnedBehaviors 1066
OwnedComments 1066
OwnedConnectors 1066
OwnedDiagrams 1066
OwnedElements 1066
OwnedEnds 1067
OwnedGuiElements 1067
OwnedGuiNodeLinks 1067
OwnedHyperlinks 1067
OwnedLiterals 1067
OwnedMembers 1067
OwnedOperations 1067
OwnedParameters 1067
OwnedPorts 1067
OwnedReceptions 1067
OwnedRules 1068
OwnedStereotypes 1068
OwnedTemplateBindings 1068
OwnedTemplateParameters 1068
OwnedTypes 1068
OwnedUseCases 1068

	PackagedElements 1070 PackageImports 1070 PackageMerges 1070 ParameterSubstitutions 1071 Postconditions 1071 Preconditions 1071 ProfileApplications 1072 Qualifiers 1072 RaisedExceptions 1072 Realizations 1072 RealizingConnectors 1072 Referred 1073 Regions 1073 RelatedElements 1073 RelativeNodes 1073 Results 1074 RoleBindings 1074 Slots 1082 Sources 1082 Specifics 1082 StereotypeApplications 1083 StructuredActivityNodes 1083 SubGroups 1084 Subjects 1084 SubmachineStates 1084 SubPartitions 1084 SubVertices 1084 SuperClasses 1084 SupplierDependencies 1084 Suppliers 1085 Targets 1085 TextLabels 1086 Transitions 1086 Triggers 1087 TypedElements 1087 UpperValues 1087 UseCases 1088 Values 1088 Waypoints 1089
Interface IUMLDataType 1146	Operation OwnedAttributes 1147 OwnedOperations 1147
Interface IUMLDependency 1148	Operation Clients 1149 Suppliers 1149
Interface IUMLDeploymentTarget 1151	Operation DeployedElements 1151 Deployments 1151
Interface IUMLDirectedRelationship 1153	Operation Sources 1153 Targets 1153
Interface IUMLDuration 1154	Operation Observations 1154
Interface IUMLElement 1157	Operation AllAppliableStereotypes 1158 AppliedStereotypes 1158 GetOwnedElementsOfKind 1158 OwnedComments 1159 OwnedElements 1160 StereotypeApplications 1160
Interface IUMLEncapsulatedClassifier 1162	Operation OwnedPorts 1162
Interface IUMLEnumeration 1162	Operation OwnedLiterals 1163
Interface IUMLExceptionHandler 1166	Operation ExceptionTypes 1166
Interface IUMLExecutableNode 1167	Operation Handlers 1168
Interface IUMLExpansionRegion 1171	Operation InputElements 1171 OutputElements 1172
Interface IUMLExpression 1172	Operation Operands 1173
Interface IUMLExtend 1173	Operation ExtensionLocations 1174
Interface IUMLFeature 1175	Operation FeaturingClassifiers 1176
Interface IUMLGuiDiagram 1316	Operation GuiLinks 1318 Layers 1319 OwnedHyperlinks 1319
Interface IUMLGuiElement 1321	Operation OwnedGuiElements 1321

© 2016-2022 Altova GmbH

Interface IUMLGuiLineLink 1326
Operation AllWaypoints 1327
LineConnectionWaypoints 1328
Waypoints 1328

Interface IUMLGuiLink 1328
Operation AttachedNodes 1329
RelativeNodes 1329

Interface IUMLGuiNodeLink 1331
Operation OwnedGuiNodeLinks 1332

Interface IUMLGuiNote 1333
Operation OwnedHyperlinks 1334

Interface IUMLGuiRootElement 1338
Operation OwnedDiagrams 1339

Interface IUMLGuiTextLabelWaypoint 1358
Operation TextLabels 1359

Interface IUMLGuiWaypoint 1366
Operation LineLinks 1366

Interface IUMLInformationFlow 1186
Operation Conveyed 1186
InformationFlowRealizations 1187
InformationSources 1187
InformationTargets 1187
RealizingConnectors 1188

Interface IUMLInstanceSpecification 1190
Operation Slots 1192

Interface IUMLInteraction 1194
Operation FormalGates 1194 Fragments 1194
Lifelines 1195 Messages 1195

Interface IUMLInteractionUse 1198
Operation ActualGates 1198

Interface IUMLInterface 1199
Operation NestedClassifiers 1201
OwnedAttributes 1201
OwnedOperations 1201
OwnedReceptions 1201

Interface IUMLInterruptibleActivityRegion 1203
Operation InterruptingEdges 1204 Nodes 1204

Interface IUMLInvocationAction 1207
Operation Arguments 1207

Interface IUMLMessage 1216
Operation OwnedArguments 1217

Interface IUMLMultiplicityElement 1221
Operation LowerValues 1222 UpperValues 1222

Interface IUMLNamedElement 1222
Operation ClientDependencies 1223
OwnedHyperlinks 1224
SupplierDependencies 1224

Interface IUMLNamespace 1225
Operation ElementImports 1225
ImportedMembers 1225
Members 1226 OwnedMembers 1226
OwnedRules 1226
PackageImports 1227
PackageMerges 1227

Interface IUMLNode 1227
Operation NestedNodes 1228

Interface IUMLObjectNode 1229
Operation ExceptionHandlers 1230 InStates 1230

Interface IUMLOpaqueAction 1233
Operation InputValues 1233 OutputValues 1234

Interface IUMLPackage 1238
Operation NestedPackages 1240
OwnedStereotypes 1240
OwnedTypes 1240
PackagedElements 1241
ProfileApplications 1241

Interface IUMLProperty 1251
Operation Qualifiers 1254

Interface IUMLProtocolTransition 1256
Operation Referred 1256

Interface IUMLRegion 1261
Operation SubVertices 1262 Transitions 1262

Interface IUMLRelationship 1262
Operation RelatedElements 1263

Interface IUMLSignal 1264
Operation OwnedAttributes 1265

Interface IUMLSlot 1266
Operation Values 1267

	Interface IUMLState 1268	Operation ConnectionPoints 1269 Connections 1269 Regions 1270
	Interface IUMLStateMachine 1272	Operation ConnectionPoints 1272 Regions 1273 SubmachineStates 1273
	Interface IUMLStructuredActivityNode 1277	Operation Edges 1278 Nodes 1278
	Interface IUMLStructuredClassifier 1279	Operation OwnedAttributes 1279 OwnedConnectors 1280
	Interface IUMLTemplateableElement 1280	Operation OwnedTemplateBindings 1281
	Interface IUMLTemplateBinding 1281	Operation ParameterSubstitutions 1282
	Interface IUMLTemplateSignature 1285	Operation OwnedTemplateParameters 1286
	Interface IUMLTimeExpression 1288	Operation Observations 1289
	Interface IUMLTransition 1291	Operation Triggers 1293
	Interface IUMLType 1294	Operation TypedElements 1295
	Interface IUMLUseCase 1296	Operation Extends 1297 ExtensionPoints 1297 Includes 1297 Subjects 1298
	Interface IUMLVertex 1303	Operation Incomings 1304 Outgoings 1304

Operation **IUMLDataList::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLDataList::ContainsUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	IUMLData 1011			
	return	**return**	**bool**			

Operation **IUMLDataList::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataList::HasChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataList::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLData 1011			

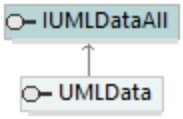
Operation **IUMLDataList::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

© 2016-2022 Altova GmbH

17.5.3.3 UModelAPI - IUMLDataAll

Interface **IUMLDataAll**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
hierarchy	

Operation **IUMLDataAll::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComponent** 1135			

Operation **IUMLDataAll::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLDataAll::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActionInputPin** 1094			

Operation **IUMLDataAll::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer** 1320			

Operation **IUMLDataAll::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity** 1095			

Operation **IUMLDataAll::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameterableElement 1246			

Operation **IUMLDataAll::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

Operation **IUMLDataAll::AddOwnedGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	IUMLGuiNodeLink 1331			
	return	**return**	**void**			

Operation **IUMLDataAll::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	IUMLGuiNodeLink 1331			

Operation **IUMLDataAll::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	ipFromLink	**in**	IUMLGuiLink 1328
	ipToLink	**in**	IUMLGuiLink 1328
	return	**return**	IUMLGuiContainmentLink 1314

Operation **IUMLDataAll::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	IUMLData 1011			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	IUMLGuiNodeLink 1331			

Operation **IUMLDataAll::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	IUMLGuiNote 1333			

Operation **IUMLDataAll::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	IUMLGuiNote 1333			
	ipToLink	**in**	IUMLGuiNodeLink 1331			
	return	**return**	IUMLGuiNoteLink 1334			

Operation **IUMLDataAll::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	IUMLGuiNote 1333			
	ipToLink	**in**	IUMLGuiLineLink 1326			
	nDistanceFromLineBegin	**in**	**int**			
	return	**return**	IUMLGuiNoteLink 1334			

Operation **IUMLDataAll::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	ipFromNode	**in**	IUMLGuiNodeLink 1331			
	ipToNode	**in**	IUMLGuiNodeLink 1331			
	return	**return**	IUMLGuiLineLink 1326			

Operation **IUMLDataAll::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**ENUMUMLAggregationKind** 1369

Operation **IUMLDataAll::Alias**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::AllAppliableStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::AllowSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::AllWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLDataAll::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

Operation **IUMLDataAll::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProfile** 1249			

Operation **IUMLDataAll::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::ApplyingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	ENUMUMLPredefinedElement 1376			
	return	**return**	IUMLStereotypeApplication 1275			

Operation **IUMLDataAll::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	IUMLStereotype 1273			
	return	**return**	IUMLStereotypeApplication 1275			

Operation **IUMLDataAll::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Association**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLAssociation 1108			

Operation **IUMLDataAll::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProperty 1251			

Operation **IUMLDataAll::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiLink 1328			

Operation **IUMLDataAll::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::BeginOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::BehaviorExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavioralFeature 1112			

Operation **IUMLDataAll::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::BooleanValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Bottom**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateableElement 1280			

Operation **IUMLDataAll::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOperation 1236			

Operation **IUMLDataAll::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInputPin 1139			

Operation **IUMLDataAll::ChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification 1299			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass** 1121			

Operation **IUMLDataAll::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLDataAll::ClientDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Clients**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::CodeLang**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang** 1004			

Operation **IUMLDataAll::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion** 1005			

Operation **IUMLDataAll::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLDataAll::CodeProjectFileOrDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLCollaboration** 1129			

Operation **IUMLDataAll::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLCallConcurrencyKind** 1369			

Operation **IUMLDataAll::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLConnectorKind** 1370			

Operation **IUMLDataAll::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation** 1108			

Operation **IUMLDataAll::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::ConstrainingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::ConstrainingPointX**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**int**

Operation **IUMLDataAll::ConstrainingPointY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLRegion 1261			

Operation **IUMLDataAll::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamespace 1225			

Operation **IUMLDataAll::Contract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInterface 1199			

Operation **IUMLDataAll::ContrainingAreaIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLLifeline 1209			

Operation **IUMLDataAll::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataType 1146			

Operation **IUMLDataAll::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLDataAll::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::DefaultLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::DefaultParamValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStructuralFeature** 1276			

Operation **IUMLDataAll::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Deployments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLParameterDirectionKind** 1376			

Operation **IUMLDataAll::DistanceFromLineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::DoActivity**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::Element**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement 1157			

Operation **IUMLDataAll::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::EndOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::EndTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::Enumeration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLEnumeration 1162			

Operation **IUMLDataAll::EraseAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**void**

Operation **IUMLDataAll::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::EraseExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseFromDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	IUMLGuiElement 1321			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**void**

Operation **IUMLDataAll::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::EraseWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::Event**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLEvent 1165			

Operation **IUMLDataAll::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::EventFilter**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ExceptionHandlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLObjectNode 1229			

Operation **IUMLDataAll::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExecutionSpecification 1169			

Operation **IUMLDataAll::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExecutionSpecification 1169			

Operation **IUMLDataAll::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::Exits**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification 1299			

Operation **IUMLDataAll::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExpression 1172			

Operation **IUMLDataAll::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

Operation **IUMLDataAll::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

Operation **IUMLDataAll::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::FeaturingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::FileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	strName	**in**	**string**			
	return	**return**	IUMLNamedElement 1222			

Operation **IUMLDataAll::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	ENUMUMLPredefinedElement 1376			
	bRecursive	**in**	**bool**			
	return	**return**	IUMLData 1011			

Operation **IUMLDataAll::Finish**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification** 1231			

Operation **IUMLDataAll::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter** 1283			

Operation **IUMLDataAll::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLDataAll::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::GeneralValueLifelineNameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLDataAll::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLDataAll::GetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**int**

Operation **IUMLDataAll::GetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	bWithBrackets	**in**	**bool**			
	return	**return**	**string**			

Operation **IUMLDataAll::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOperation 1236			

Operation **IUMLDataAll::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	bRecursive	**in**	**bool**			
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::GetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLLifeline 1209			

Operation **IUMLDataAll::GetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	ENUMUMLPredefinedElement 1376			
	return	**return**	IUMLStereotypeApplication 1275			

Operation **IUMLDataAll::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	IUMLStereotype 1273			
	return	**return**	IUMLStereotypeApplication 1275			

Operation **IUMLDataAll::GetTargetLifeline**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline** 1209			

Operation **IUMLDataAll::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel** 1356			
	return	**return**	**string**			

Operation **IUMLDataAll::GetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::GetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLDataAll::Guard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::GuiLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement** 1321			

Operation **IUMLDataAll::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode** 1167			

Operation **IUMLDataAll::Handlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::HSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::ImplementingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavioredClassifier** 1114			

Operation **IUMLDataAll::ImportedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement** 1241			

Operation **IUMLDataAll::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

Operation **IUMLDataAll::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace** 1225			

Operation **IUMLDataAll::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity** 1095			

Operation **IUMLDataAll::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase** 1296			

Operation **IUMLDataAll::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLTrigger 1293			

Operation **IUMLDataAll::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	IUMLActivityNode 1101			

	ipTo	**in**	IUMLActivityNode 1101
	return	**return**	IUMLActivityEdge 1097

Operation **IUMLDataAll::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	IUMLActivityGroup 1099			

Operation **IUMLDataAll::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	IUMLActivityNode 1101			

Operation **IUMLDataAll::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLGate 1180			

Operation **IUMLDataAll::InsertAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	IUMLElement 1157			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLInputPin 1189			

Operation **IUMLDataAll::InsertArgumentOfKindAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	IUMLInputPin 1189			

Operation **IUMLDataAll::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::InsertCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnectableElement** 1137			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertCollaborationUseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLCollaborationUse** 1130			

Operation **IUMLDataAll::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConnectionPointReference** 1138			

Operation **IUMLDataAll::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate** 1257			

Operation **IUMLDataAll::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLElement** 1157			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	ipVal	**in**	**IUMLInteractionFragment** 1197
	return	**return**	**void**

Operation **IUMLDataAll::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDeployedArtifact	**in**	**IUMLDeployedArtifact** 1149			
	return	**return**	**IUMLDeployment** 1150			

Operation **IUMLDataAll::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge** 1097			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedElement	**in**	**IUMLPackageableElement** 1241			
	return	**return**	**IUMLElementImport** 1161			

Operation **IUMLDataAll::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate** 1257			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin** 1237			

Operation **IUMLDataAll::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	nIdx	**in**	**int**
	ipVal	**in**	**IUMLPseudostate** 1257
	return	**return**	**void**

Operation **IUMLDataAll::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipExtendedCase	**in**	**IUMLUseCase** 1296			
	return	**return**	**IUMLExtend** 1173			

Operation **IUMLDataAll::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pExtensionLocation	**in**	**IUMLExtensionPoint** 1175			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExtensionPoint** 1175			

Operation **IUMLDataAll::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate** 1180			

Operation **IUMLDataAll::InsertFragmentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInteractionFragment** 1197			

Operation **IUMLDataAll::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipGeneral	**in**	**IUMLClassifier** 1124			
	return	**return**	**IUMLGeneralization** 1180			

Operation **IUMLDataAll::InsertHandlerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	return	**return**	**IUMLExceptionHandler** 1166

Operation **IUMLDataAll::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipIncludingCase	**in**	**IUMLUseCase** 1296			
	return	**return**	**IUMLInclude** 1185			

Operation **IUMLDataAll::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLRelationship** 1262			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement** 1222			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement** 1222			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode** 1170			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin** 1189			

Operation **IUMLDataAll::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLState** 1268			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipContract	**in**	**IUMLInterface** 1199			
	return	**return**	**IUMLInterfaceRealization** 1202			

Operation **IUMLDataAll::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge** 1097			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiDiagramLayer** 1320			

Operation **IUMLDataAll::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLLifeline** 1209			

Operation **IUMLDataAll::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::InsertLowerUpperValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strLower	**in**	**string**			
	strUpper	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertManifestationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	ipUtilizedElement	**in**	**IUMLPackageableElement** 1241			
	return	**return**	**IUMLManifestation** 1214			

Operation **IUMLDataAll::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLMessage** 1216			

Operation **IUMLDataAll::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLArtifact** 1106			

Operation **IUMLDataAll::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLDataAll::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLNode** 1227			

Operation **IUMLDataAll::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLActivityNode** 1101			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation** 1231			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInteractionOperand** 1197			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode** 1170			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin** 1237			

Operation **IUMLDataAll::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLDataAll::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLDataAll::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLComment** 1132			

Operation **IUMLDataAll::InsertOwnedCommentTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLCommentTextHyperlink** 1133			

Operation **IUMLDataAll::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	ipFrom	**in**	**IUMLConnectableElement** 1137
	ipTo	**in**	**IUMLConnectableElement** 1137
	return	**return**	**IUMLConnector** 1140

Operation **IUMLDataAll::InsertOwnedDiagramAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUMLParent	**in**	**IUMLData** 1011			
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiDiagram** 1316			

Operation **IUMLDataAll::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLGuiTextHyperlink** 1355			

Operation **IUMLDataAll::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**IUMLHyperlink2File** 1182			

Operation **IUMLDataAll::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement** 1365			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement** 1222			
	return	**return**	**IUMLHyperlink2GuiElement** 1183			

Operation **IUMLDataAll::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedData	**in**	**IUMLData** 1011			
	return	**return**	**IUMLHyperlink2Model** 1184			

Operation **IUMLDataAll::InsertOwnedLiteralAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

© 2016-2022 Altova GmbH

	return	**return**	**IUMLEnumerationLiteral** 1164			

Operation **IUMLDataAll::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLDataAll::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLParameter** 1244			

Operation **IUMLDataAll::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPort** 1248			

Operation **IUMLDataAll::InsertOwnedReceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLReception** 1259			

Operation **IUMLDataAll::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSignature	**in**	**IUMLTemplateSignature** 1285			
	return	**return**	**IUMLTemplateBinding** 1281			

Operation **IUMLDataAll::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLClassifierTemplateParameter** 1127			

Operation **IUMLDataAll::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**IUMLUseCase** 1296

Operation **IUMLDataAll::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLPackageableElement** 1241			

Operation **IUMLDataAll::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLElement** 1157			
	ipTo	**in**	**IUMLElement** 1157			
	return	**return**	**IUMLPackageableElement** 1241			

Operation **IUMLDataAll::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedPackage	**in**	**IUMLPackage** 1238			
	return	**return**	**IUMLPackageImport** 1242			

Operation **IUMLDataAll::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipMergedPackage	**in**	**IUMLPackage** 1238			
	return	**return**	**IUMLPackageMerge** 1243			

Operation **IUMLDataAll::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFormalParameter	**in**	**IUMLTemplateParameter** 1283			
	ipActualParameter	**in**	**IUMLParameterableElement** 1246			
	return	**return**	**IUMLTemplateParameterSubstitution** 1284			

Operation **IUMLDataAll::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	nIdx	**in**	**int**
	return	**return**	**IUMLConstraint** 1142

Operation **IUMLDataAll::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipAppliedProfile	**in**	**IUMLProfile** 1249			
	return	**return**	**IUMLProfileApplication** 1250			

Operation **IUMLDataAll::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLDataAll::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLType** 1294			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipRealizingClassifier	**in**	**IUMLClassifier** 1124			
	return	**return**	**IUMLComponentRealization** 1136			

Operation **IUMLDataAll::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnector** 1140			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion** 1261			

Operation **IUMLDataAll::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin** 1237			

Operation **IUMLDataAll::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	return	**return**	**IUMLSlot** 1266			

Operation **IUMLDataAll::InsertSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLDataAll::InsertSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pSubject	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityPartition** 1103			

Operation **IUMLDataAll::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLVertex** 1303			

Operation **IUMLDataAll::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSource	**in**	**IUMLVertex** 1303			
	ipTarget	**in**	**IUMLVertex** 1303			
	return	**return**	**IUMLTransition** 1291			

Operation **IUMLDataAll::InsertTriggerAt**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger** 1293			

Operation **IUMLDataAll::InsertValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::InsertWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiWaypoint** 1366			

Operation **IUMLDataAll::Instance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification** 1190			

Operation **IUMLDataAll::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::IntegerValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::InteractionOperator**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLInteractionOperatorKind** 1374			

Operation **IUMLDataAll::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

Operation **IUMLDataAll::InterfaceRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::InterruptibleActivityRegions**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLDataAll::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsActivityReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsBehavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	ENUMCodeLang 1004			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	ENUMCodeLang 1004			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCombineDuplicate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsComputable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsConjugated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsControl**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsControlType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsDerived**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsDerivedUnion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsDimension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsEditable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	IUMLElement 1157			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsExternal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsFinalSpecialization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsHorizontal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsIndirectlyInstantiated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsLocallyReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsLocked**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsMultiCast**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsMultiReceive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsNavigable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::IsNull**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsOrthogonal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsOwnedEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsPositioned**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement** 1376			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsQuery**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSameUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLDataToCompare	**in**	**IUMLData** 1011			

	return	**return**	**bool**

Operation **IUMLDataAll::IsService**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsShared**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsShowAsGeneralValueLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSimple**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsStatic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype** 1273			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSubmachineState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsSynchronous**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel** 1356			
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsUnmarshall**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsUseForCodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsVarArgList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::IsVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::JoinSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Label**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Layer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiDiagramLayer 1320			

Operation **IUMLDataAll::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Left**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiElement 1321			

Operation **IUMLDataAll::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiElement 1321			

Operation **IUMLDataAll::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiVisibleElement 1365			

Operation **IUMLDataAll::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamedElement 1222			

Operation **IUMLDataAll::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLData 1011			

Operation **IUMLDataAll::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IUMLElement 1157

Operation **IUMLDataAll::LocalPostConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Location**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDeploymentTarget 1151			

Operation **IUMLDataAll::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOpaqueExpression 1235			

Operation **IUMLDataAll::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification 1299			

Operation **IUMLDataAll::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification 1299			

Operation **IUMLDataAll::MemberEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Members**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLDataAll::MergeLayersAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromIdx	**in**	**int**			
	nToIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::Message**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLMessage 1216			

Operation **IUMLDataAll::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLMessageKind 1374			

Operation **IUMLDataAll::Messages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLMessageSort 1375			

Operation **IUMLDataAll::MetaClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Methods**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiMiddleWaypoint 1330			

Operation **IUMLDataAll::Min**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLExpansionKind** 1371			

Operation **IUMLDataAll::MoveTo**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::NameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace** 1225			

Operation **IUMLDataAll::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::NestedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::NestedPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::NestingInterface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

Operation **IUMLDataAll::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

Operation **IUMLDataAll::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::NoteText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::NoteTextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::NoteTextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OccurringEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent** 1165			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLDataAll::OperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint** 1195			

Operation **IUMLDataAll::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLDataAll::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLDataAll::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLObjectNodeOrderingKind** 1375			

Operation **IUMLDataAll::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OutputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OutputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OutputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedActual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement** 1246			

Operation **IUMLDataAll::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComment** 1132			

Operation **IUMLDataAll::OwnedDocCommentBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::OwnedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::OwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameterableElement 1246			

Operation **IUMLDataAll::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedPorts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedReceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateSignature 1285			

Operation **IUMLDataAll::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement 1157			

Operation **IUMLDataAll::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLAssociation 1108			

Operation **IUMLDataAll::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLDataAll::OwningElement**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IUMLElement 1157			

Operation **IUMLDataAll::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiNodeLink 1331			

Operation **IUMLDataAll::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInstanceSpecification 1190			

Operation **IUMLDataAll::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInstanceSpecification 1190			

Operation **IUMLDataAll::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLMultiplicityElement 1221			

Operation **IUMLDataAll::OwningPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLDataAll::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameter 1244			

Operation **IUMLDataAll::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProperty 1251			

Operation **IUMLDataAll::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLSignal 1264			

Operation **IUMLDataAll::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLSlot 1266			

Operation **IUMLDataAll::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLState 1268			

Operation **IUMLDataAll::OwningTemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateParameter 1283			

Operation **IUMLDataAll::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProtocolTransition 1256			

Operation **IUMLDataAll::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLMultiplicityElement 1221			

Operation **IUMLDataAll::Package**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLDataAll::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::PackageMerges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameter 1244			

Operation **IUMLDataAll::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameterableElement 1246			

Operation **IUMLDataAll::ParameterSignature**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateSignature 1285			

Operation **IUMLDataAll::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLDataAll::PinValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::PostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLDataAll::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::PostTypeModifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::PreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLDataAll::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExecutableNode 1167			

Operation **IUMLDataAll::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProtocolStateMachine 1255			

Operation **IUMLDataAll::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLPseudostateKind 1377			

Operation **IUMLDataAll::QualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::RaisedExceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClassifier 1124			

Operation **IUMLDataAll::RealizingConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IUMLMessageEnd 1218			

Operation **IUMLDataAll::ReceivingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLDataAll::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiDiagram 1316			

Operation **IUMLDataAll::Referred**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::RefersTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInteraction 1194			

Operation **IUMLDataAll::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExpansionRegion 1171			

Operation **IUMLDataAll::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExpansionRegion 1171			

Operation **IUMLDataAll::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::RelatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Represents**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLConnectableElement 1137			

Operation **IUMLDataAll::Result**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOutputPin 1237			

Operation **IUMLDataAll::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Right**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Role**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConnectableElement 1137			

Operation **IUMLDataAll::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::Selector**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification 1299			

Operation **IUMLDataAll::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IUMLMessageEnd 1218			

Operation **IUMLDataAll::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLSignal 1264			

Operation **IUMLDataAll::SeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	IUMLElement 1157			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement	**in**	IUMLGuiVisibleElement 1365			
	ipLinkedGuiElementCell	**in**	IUMLNamedElement 1222			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	IUMLData 1011			
	return	**return**	**void**			

Operation **IUMLDataAll::SetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

Operation **IUMLDataAll::SetNewActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin** 1189			

Operation **IUMLDataAll::SetNewChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLDataAll::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString** 1213			

Operation **IUMLDataAll::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	strKind	**in**	**string**
	return	**return**	**IUMLBehavior** 1110

Operation **IUMLDataAll::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLDataAll::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLDataAll::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLDataAll::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::SetNewMapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression** 1235			

Operation **IUMLDataAll::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewMaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			

	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewMin**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewOperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint** 1195			

Operation **IUMLDataAll::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLParameterableElement** 1246			

Operation **IUMLDataAll::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::SetNewProtocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine** 1255			

Operation **IUMLDataAll::SetNewSelector**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin** 1189			

Operation **IUMLDataAll::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLDataAll::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString** 1213			

Operation **IUMLDataAll::SetNewStateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::SetNewTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature** 1285			

Operation **IUMLDataAll::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLDataAll::SetNewWhen**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression** 1288			

Operation **IUMLDataAll::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLOperation** 1236			
	return	**return**	**void**			

Operation **IUMLDataAll::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	x	**in**	**int**			
	y	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nProperty	**in**	**ENUMUMLPredefinedElement** 1376			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetRect**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	nRight	**in**	**int**			
	nBottom	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetScrollPos**

parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLDataAll::SetSlotValueAt**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetStateIndexErased**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDataAll::SetTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel** 1356			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	nPosition	**in**	**int**
	return	**return**	**void**

Operation **IUMLDataAll::Signal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLSignal 1264			

Operation **IUMLDataAll::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInputPin 1189			

Operation **IUMLDataAll::Signature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateSignature 1285			

Operation **IUMLDataAll::SingleExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Source**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityNode 1101			

Operation **IUMLDataAll::Sources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Specific**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClassifier 1124			

Operation **IUMLDataAll::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification 1299			

Operation **IUMLDataAll::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Start**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOccurrenceSpecification 1231			

Operation **IUMLDataAll::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLState 1268			

Operation **IUMLDataAll::StateCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::StateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLDataAll::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLStateMachine 1272			

Operation **IUMLDataAll::Stereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLStereotype 1273			

Operation **IUMLDataAll::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiStyles 1346			

Operation **IUMLDataAll::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiStyles 1346			

Operation **IUMLDataAll::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLStateMachine 1272			

Operation **IUMLDataAll::SubmachineStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::SubVertices**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::SuperGroup**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityGroup 1099			

Operation **IUMLDataAll::SuperPartition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityPartition 1103			

Operation **IUMLDataAll::SupplierDependencies**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Symbol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Target**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityNode 1101			

Operation **IUMLDataAll::Targets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Template**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateableElement 1280			

Operation **IUMLDataAll::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateBinding 1281			

Operation **IUMLDataAll::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateParameter 1283			

Operation **IUMLDataAll::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::TextLabelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement 1157			

Operation **IUMLDataAll::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	ENUMUMLGuiTextLabelKind 1373

Operation **IUMLDataAll::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::TextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::TimeObservationEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamedElement 1222			

Operation **IUMLDataAll::TimeTickLengthCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Top**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::Transformation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLDataAll::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLDataAll::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLTransitionKind 1377			

Operation **IUMLDataAll::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLVertex 1303			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLVertex 1303			

Operation **IUMLDataAll::Triggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLType 1294			

Operation **IUMLDataAll::TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	ENUMUMLPredefinedElement 1376			
	return	**return**	**void**			

Operation **IUMLDataAll::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	IUMLStereotype 1273			
	return	**return**	**void**			

Operation **IUMLDataAll::UnlimitedValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::UpperBound**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::UpperValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::UseCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

Operation **IUMLDataAll::UseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::UseForForwardEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::UserDefinedLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::UtilizedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackageableElement 1241			

Operation **IUMLDataAll::UUID**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Values**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLDataAll::Viewpoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLVisibilityKind 1378			

Operation **IUMLDataAll::VisualStatePositionCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::VSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

© 2016-2022 Altova GmbH

Operation **IUMLDataAll::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLDataAll::Waypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLDataAll::Weight**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLDataAll::When**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression** 1288			

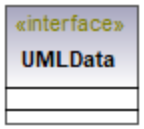
Operation **IUMLDataAll::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

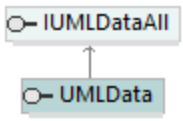
17.5.3.4 UModelAPI - UMLData

Interface **UMLData**

diagram



hierarchy



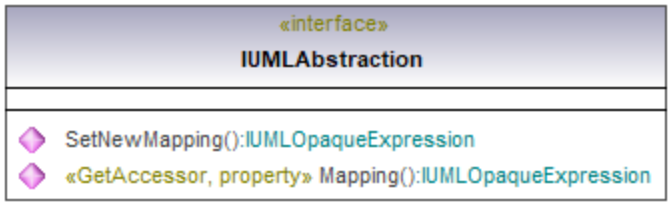
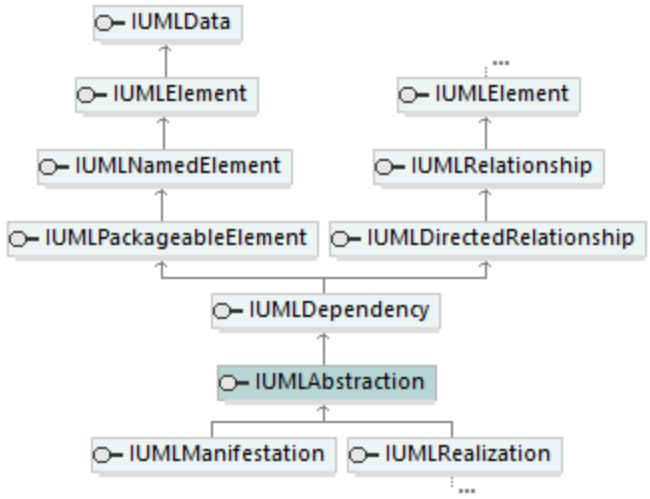
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5 IUMLElement

Dies ist eine Liste von Elementen, die in der UML-Spezifikation der OMG definiert sind (siehe auch http://www.uml.org).

17.5.3.5.1 UModelAPI - IUMLAbstraction

Interface **IUMLAbstraction**

diagram	
hierarchy	

Operation **IUMLAbstraction::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression** 1235			

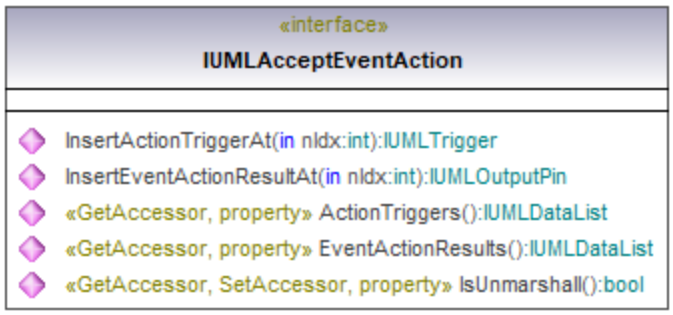
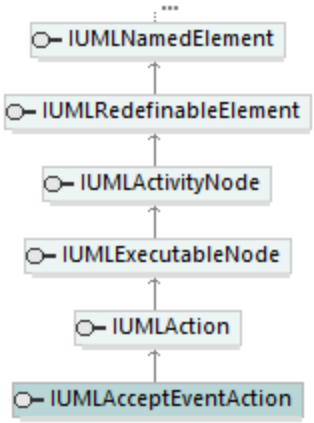
Operation **IUMLAbstraction::SetNewMapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression** 1235			

© 2016-2022 Altova GmbH

17.5.3.5.2 UModelAPI - IUMLAcceptEventAction

Interface **IUMLAcceptEventAction**

diagram	
hierarchy	

Operation **IUMLAcceptEventAction::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLTrigger 1293.					

Operation **IUMLAcceptEventAction::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLOutputPin 1237.					

Operation **IUMLAcceptEventAction::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger** 1293			

Operation **IUMLAcceptEventAction::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin** 1237			

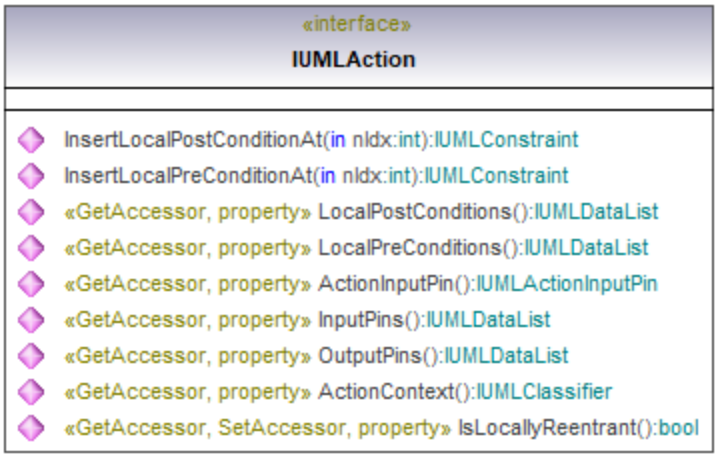
Operation **IUMLAcceptEventAction::IsUnmarshall**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

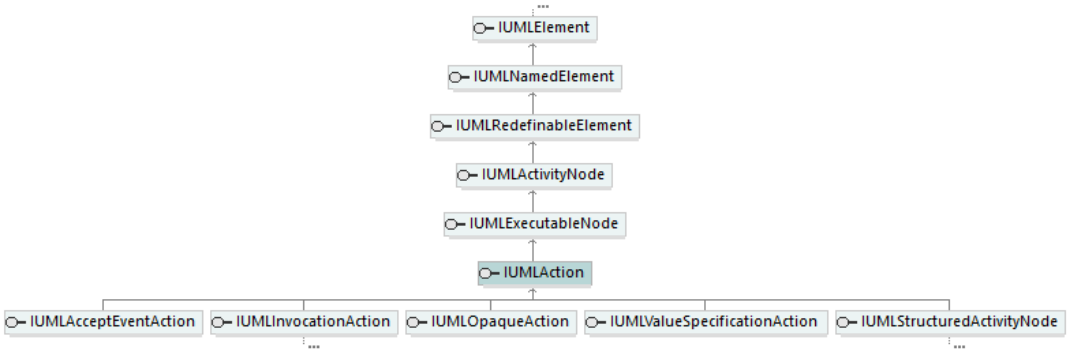
17.5.3.5.3 UModelAPI - IUMLAction

Interface **IUMLAction**

diagram



hierarchy



Operation **IUMLAction::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2016-2022 Altova GmbH

	return	**return**	**IUMLClassifier** 1124

Operation **IUMLAction::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActionInputPin** 1094			

Operation **IUMLAction::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLInputPin 1189.					

Operation **IUMLAction::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLAction::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLAction::IsLocallyReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLAction::LocalPostConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLConstraint 1142.					

Operation **IUMLAction::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLConstraint 1142.					

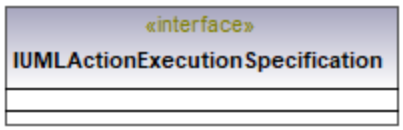
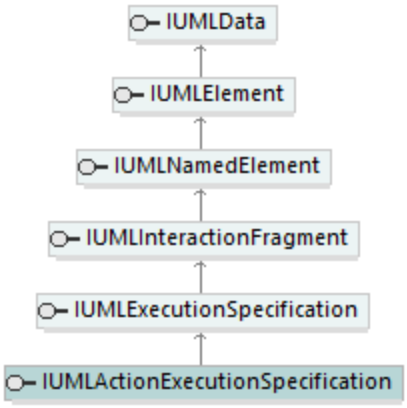
Operation **IUMLAction::OutputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

documentation	A list of elements of type IUMLOutputPin 1237.

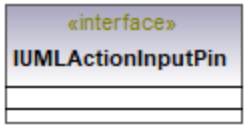
17.5.3.5.4 UModelAPI - IUMLActionExecutionSpecification

Interface **IUMLActionExecutionSpecification**

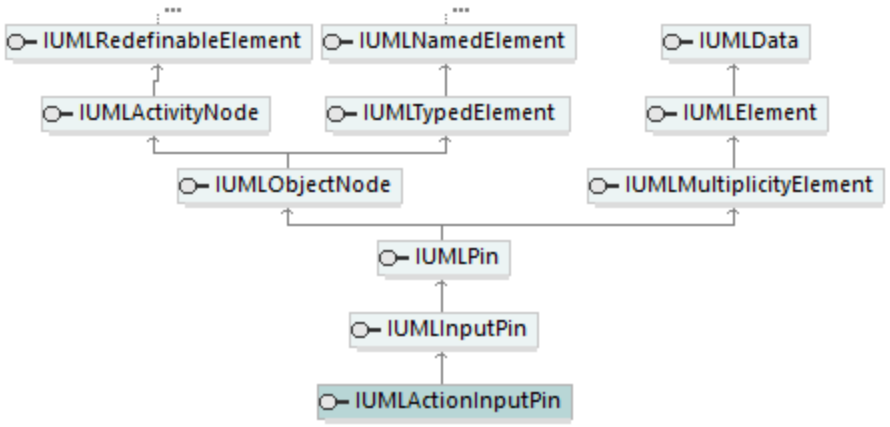
diagram	
hierarchy	

17.5.3.5.5 UModelAPI - IUMLActionInputPin

Interface **IUMLActionInputPin**

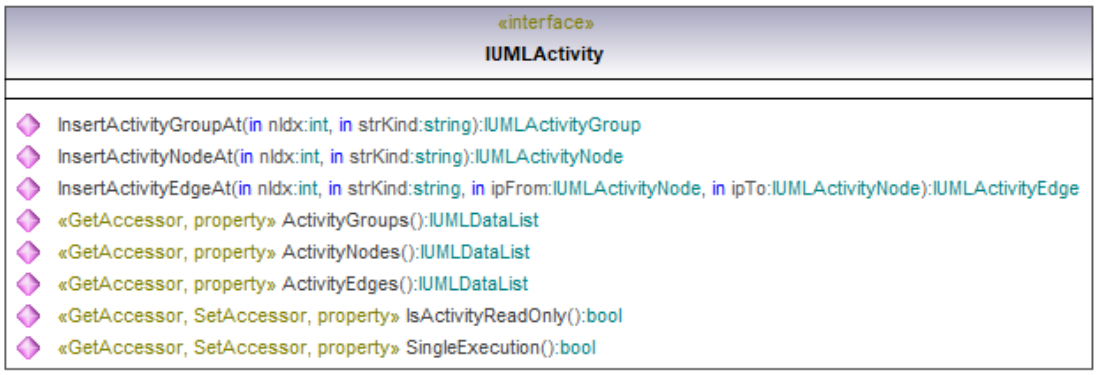
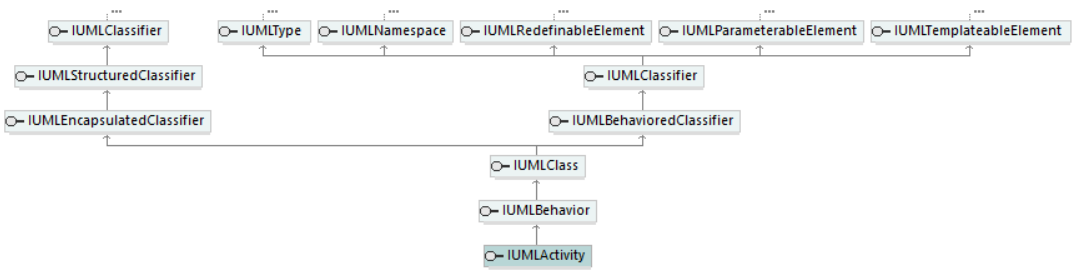
diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLAction** 1092 Interface **IUMLDataAll** 1019 Operation **ActionInputPin** 1093 Operation **ActionInputPin** 1019

17.5.3.5.6 UModelAPI - IUMLActivity

Interface **IUMLActivity**

diagram	
hierarchy	

typedElements	Interface IUMLActivityEdge 1097 Interface IUMLActivityGroup 1099 Interface IUMLDataAll 1019	Operation Activity 1098 Operation InActivity 1100 Operation Activity 1019 InActivity 1038

Operation **IUMLActivity::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityEdge 1097.					

Operation **IUMLActivity::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityGroup 1099.					

Operation **IUMLActivity::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityNode 1101.					

Operation **IUMLActivity::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	IUMLActivityNode 1101			
	ipTo	**in**	IUMLActivityNode 1101			
	return	**return**	IUMLActivityEdge 1097			

Operation **IUMLActivity::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	IUMLActivityGroup 1099			

Operation **IUMLActivity::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	IUMLActivityNode 1101			

Operation **IUMLActivity::IsActivityReadOnly**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

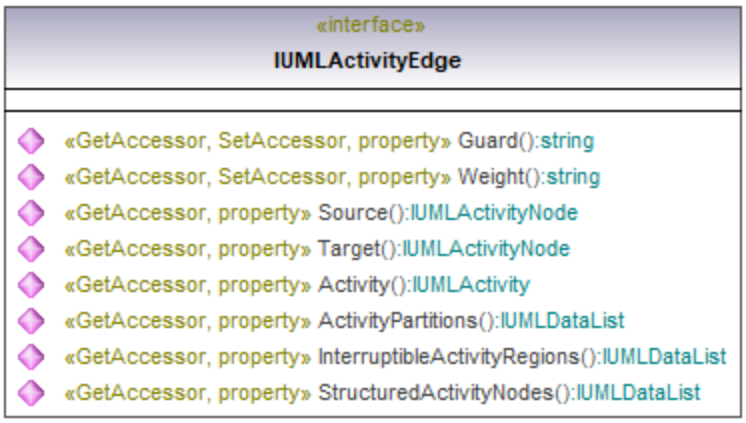
Operation **IUMLActivity::SingleExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

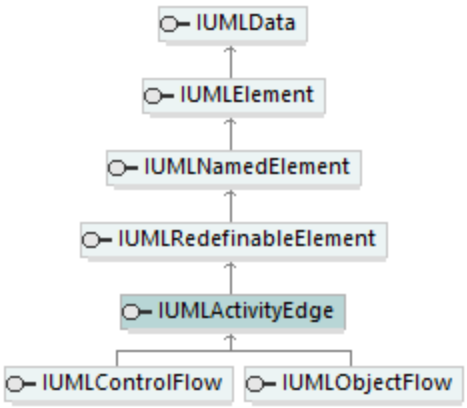
17.5.3.5.7 UModelAPI - IUMLActivityEdge

Interface **IUMLActivityEdge**

diagram



hierarchy



typedElements

Interface **IUMLActivity** 1095
Interface **IUMLActivityPartition** 1103
Interface **IUMLDataAll** 1019

Operation **InsertActivityEdgeAt** 1096
Operation **InsertEdgeAt** 1104
Operation **InsertActivityEdgeAt** 1039
InsertEdgeAt 1042
InsertInterruptingEdgeAt 1045

	Interface IUMLInterruptibleActivityRegion 1203 Interface IUMLStructuredActivityNode 1277	Operation InsertInterruptingEdgeAt 1204 Operation InsertEdgeAt 1278

Operation **IUMLActivityEdge::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivity 1095			

Operation **IUMLActivityEdge::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityPartition 1103.					

Operation **IUMLActivityEdge::Guard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLActivityEdge::InterruptibleActivityRegions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLInterruptibleActivityRegion 1203.					

Operation **IUMLActivityEdge::Source**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityNode 1101			

Operation **IUMLActivityEdge::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLStructuredActivityNode 1277.					

Operation **IUMLActivityEdge::Target**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityNode 1101			

Operation **IUMLActivityEdge::Weight**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

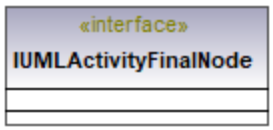
© 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

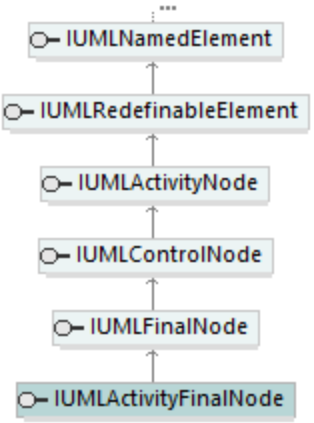
17.5.3.5.8 UModelAPI - IUMLActivityFinalNode

Interface **IUMLActivityFinalNode**

diagram



hierarchy

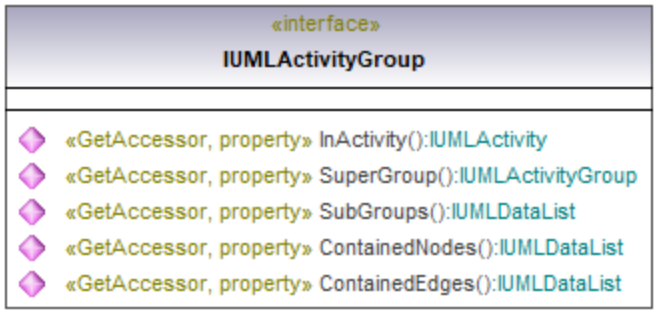


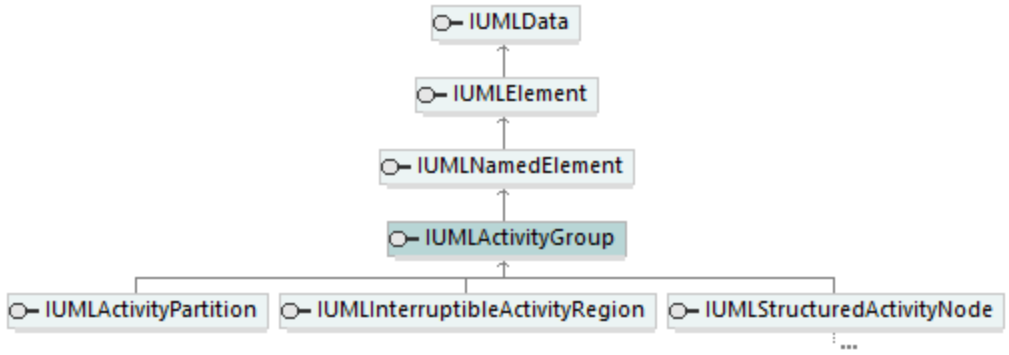
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.9 UModelAPI - IUMLActivityGroup

Interface **IUMLActivityGroup**

diagram



hierarchy	
typedElements	Interface IUMLActivity 1095 Interface IUMLActivityGroup 1099 Interface IUMLDataAll 1019 Operation InsertActivityGroupAt 1096 Operation SuperGroup 1100 Operation InsertActivityGroupAt 1040 SuperGroup 1084

Operation **IUMLActivityGroup::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityEdge 1097.					

Operation **IUMLActivityGroup::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityNode 1101.					

Operation **IUMLActivityGroup::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivity 1095			

Operation **IUMLActivityGroup::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityGroup 1099.					

Operation **IUMLActivityGroup::SuperGroup**

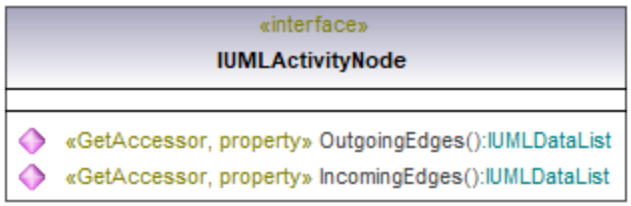
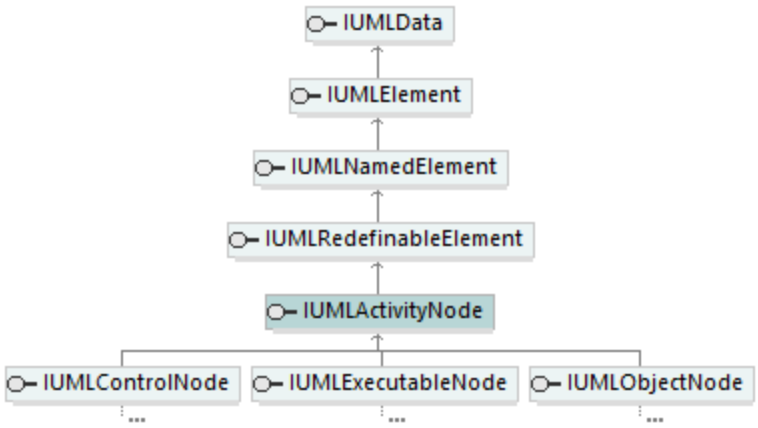
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLActivityGroup 1099			

© 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.10 UModelAPI - IUMLActivityNode

Interface **IUMLActivityNode**

diagram	
hierarchy	
typedElements	Interface **IUMLActivity** 1095 Interface **IUMLActivityEdge** 1097 Interface **IUMLActivityPartition** 1103 Interface **IUMLDataAll** 1019 Interface **IUMLInterruptibleActivityRegion** 1203 Interface **IUMLStructuredActivityNode** 1277 Operation **InsertActivityEdgeAt** 1096 **InsertActivityNodeAt** 1096 Operation **Source** 1098 **Target** 1098 Operation **InsertNodeAt** 1104 Operation **InsertActivityEdgeAt** 1039 **InsertActivityNodeAt** 1040 **InsertNodeAt** 1046 **Source** 1082 **Target** 1085 Operation **InsertNodeAt** 1204 Operation **InsertNodeAt** 1278

Operation **IUMLActivityNode::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLActivityEdge 1097.					

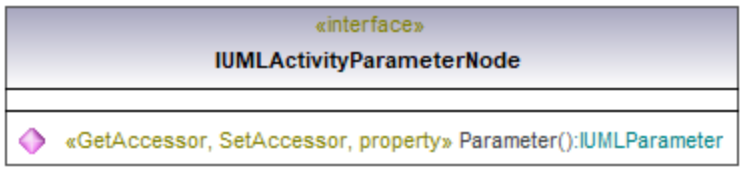
Operation **IUMLActivityNode::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLActivityEdge 1097.					

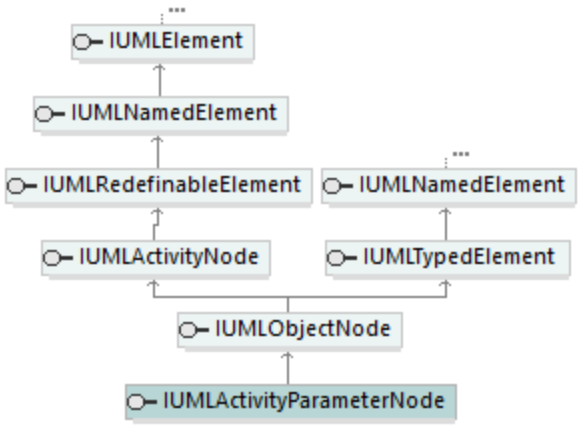
17.5.3.5.11 UModelAPI - IUMLActivityParameterNode

Interface **IUMLActivityParameterNode**

diagram



hierarchy



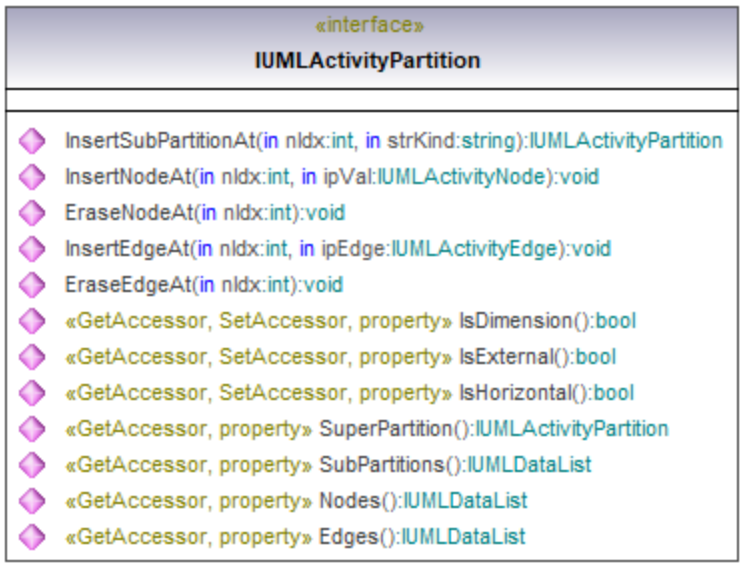
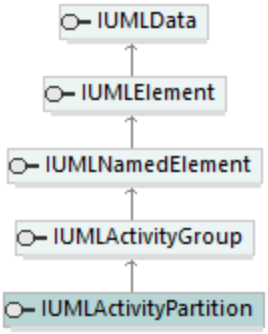
Operation **IUMLActivityParameterNode::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameter 1244			

 © 2016-2022 Altova GmbH

17.5.3.5.12 UModelAPI - IUMLActivityPartition

Interface **IUMLActivityPartition**

diagram	
hierarchy	
typedElements	Interface **IUMLActivityPartition** 1103 Interface **IUMLDataAll** 1019 Operation **InsertSubPartitionAt** 1104 **SuperPartition** 1105 Operation **InsertSubPartitionAt** 1052 **SuperPartition** 1084

Operation **IUMLActivityPartition::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLActivityEdge 1097.					

Operation **IUMLActivityPartition::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	return	**return**	**void**

Operation **IUMLActivityPartition::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge** 1097			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLActivityNode** 1101			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityPartition** 1103			

Operation **IUMLActivityPartition::IsDimension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivityPartition::IsExternal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivityPartition::IsHorizontal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivityPartition::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLActivityNode 1101.					

© 2016-2022 Altova GmbH

Operation **IUMLActivityPartition::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLActivityPartition 1103.					

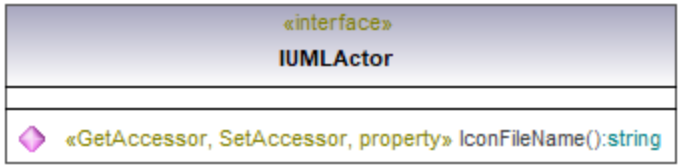
Operation **IUMLActivityPartition::SuperPartition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityPartition** 1103			

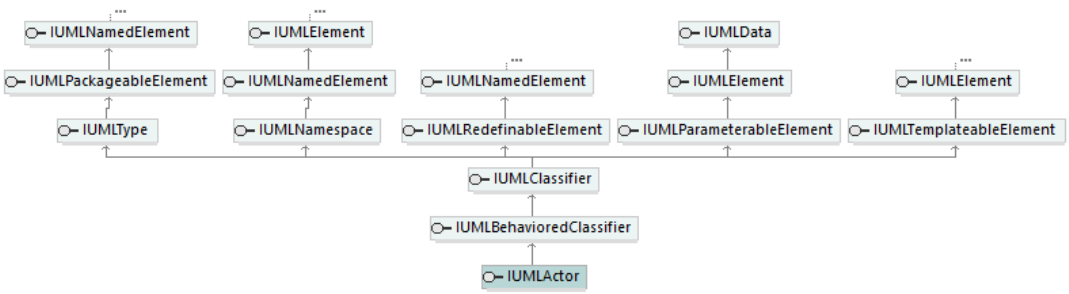
17.5.3.5.13 UModelAPI - IUMLActor

Interface **IUMLActor**

diagram



hierarchy



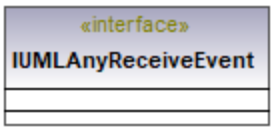
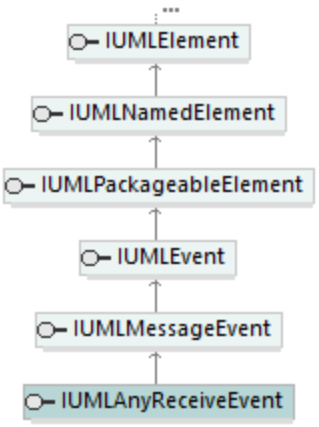
Operation **IUMLActor::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

© 2016-2022 Altova GmbH

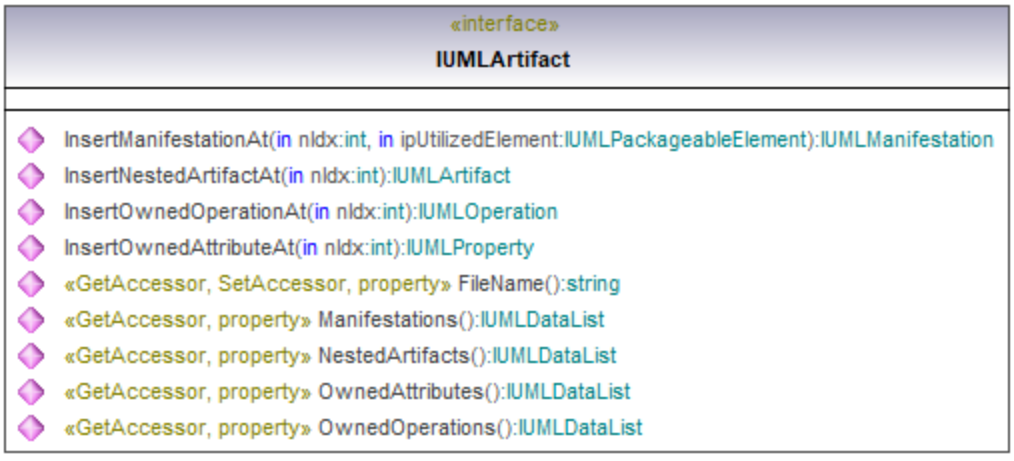
17.5.3.5.14 UModelAPI - IUMLAnyReceiveEvent

Interface **IUMLAnyReceiveEvent**

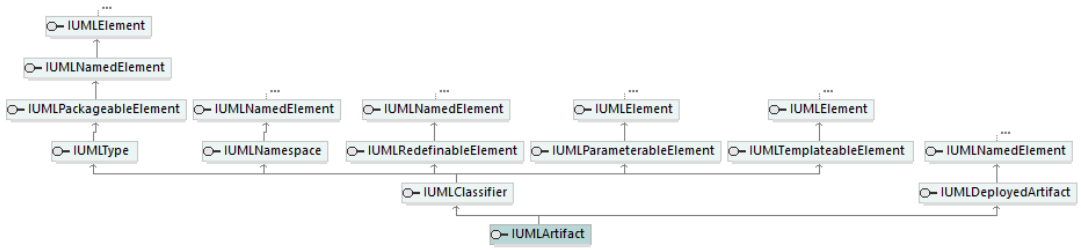
diagram	
hierarchy	

17.5.3.5.15 UModelAPI - IUMLArtifact

Interface **IUMLArtifact**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLArtifact** 1106 Interface **IUMLDataAll** 1019 Operation **InsertNestedArtifactAt** 1107 Operation **InsertNestedArtifactAt** 1046

Operation **IUMLArtifact::FileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLArtifact::InsertManifestationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUtilizedElement	**in**	**IUMLPackageableElement** 1241			
	return	**return**	**IUMLManifestation** 1214			

Operation **IUMLArtifact::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLArtifact** 1106			

Operation **IUMLArtifact::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLArtifact::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLArtifact::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLManifestation 1214.					

Operation **IUMLArtifact::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLArtifact 1106.					

Operation **IUMLArtifact::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

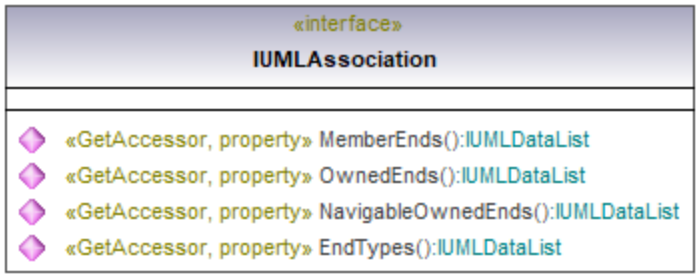
Operation **IUMLArtifact::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLOperation 1236.					

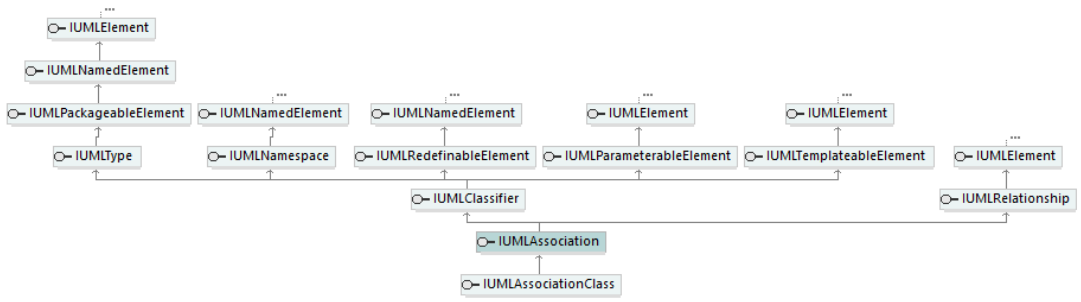
17.5.3.5.16 UModelAPI - IUMLAssociation

Interface **IUMLAssociation**

diagram



hierarchy



typedElements	Interface **IUMLConnector** 1140	Operation **ConnectorType** 1141

 © 2016-2022 Altova GmbH

	Interface **IUMLDataAll** 1019 Interface **IUMLProperty** 1251	Operation **Association** 1023 **ConnectorType** 1026 **OwningAssociation** 1068 Operation **Association** 1252 **OwningAssociation** 1254

Operation **IUMLAssociation::EndTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLType 1294.					

Operation **IUMLAssociation::MemberEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

Operation **IUMLAssociation::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

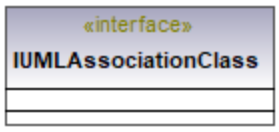
Operation **IUMLAssociation::OwnedEnds**

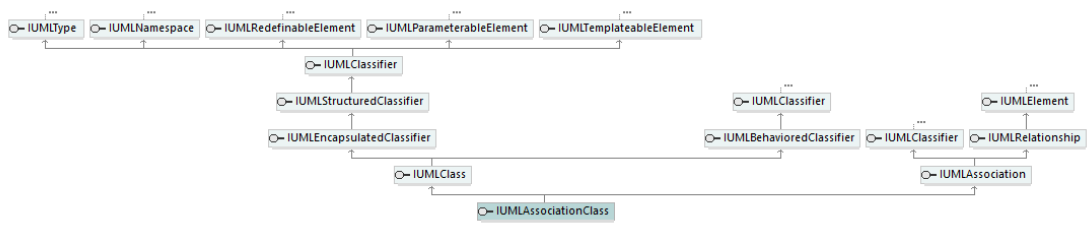
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.17 UModelAPI - IUMLAssociationClass

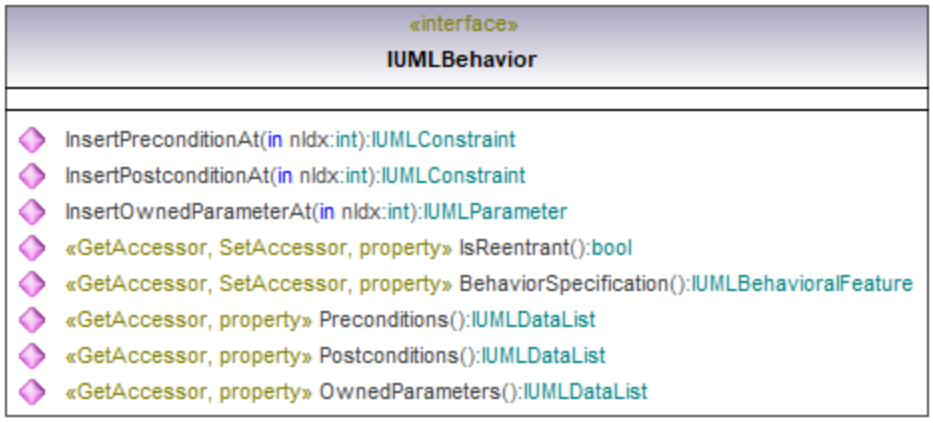
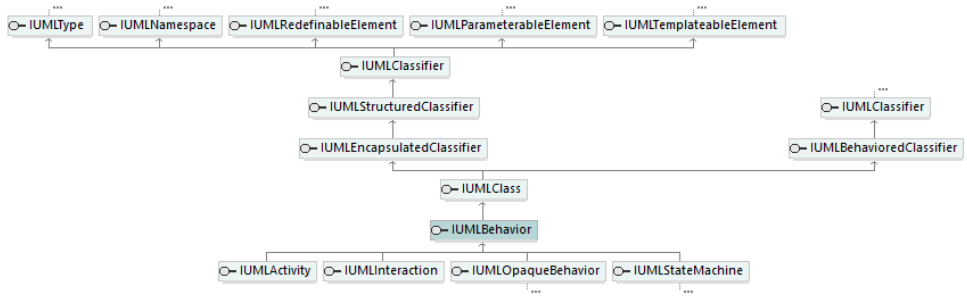
Interface **IUMLAssociationClass**

diagram	

hierarchy	

17.5.3.5.18 UModelAPI - IUMLBehavior

Interface **IUMLBehavior**

diagram	
hierarchy	
typedElements	Interface **IUMLBehavioredClassifier** 1114 Interface **IUMLBehaviorExecutionSpecification** 1115 Interface **IUMLCallBehaviorAction** 1117 Interface **IUMLDataAll** 1019 Operation **InsertOwnedBehaviorAt** 1114 Operation **BehaviorExecution** 1115 Operation **Behavior** 1117 Operation **Behavior** 1024 **BehaviorExecution** 1024 **DecisionInput** 1028 **DoActivity** 1028

 © 2016-2022 Altova GmbH

	Interface IUMLDecisionNode 1148 Interface IUMLObjectFlow 1228 Interface IUMLObjectNode 1229 Interface IUMLState 1268 Interface IUMLTransition 1291	Effect 1029 Entry 1029 Exit 1033 InsertOwnedBehaviorAt 1047 Selection 1074 SetNewDoActivity 1076 SetNewEffect 1077 SetNewEntry 1077 SetNewExit 1077 Transformation 1086 Operation DecisionInput 1148 Operation Transformation 1228 Operation Selection 1230 Operation DoActivity 1269 Entry 1269 Exit 1269 SetNewDoActivity 1270 SetNewEntry 1270 SetNewExit 1270 Operation Effect 1292 SetNewEffect 1292

Operation **IUMLBehavior::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavioralFeature 1112			

Operation **IUMLBehavior::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLParameter 1244			

Operation **IUMLBehavior::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLConstraint 1142			

Operation **IUMLBehavior::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLConstraint 1142			

Operation **IUMLBehavior::IsReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLBehavior::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLParameter 1244.					

Operation **IUMLBehavior::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLDataList 1013
documentation	A list of elements of type IUMLConstraint 1142.		

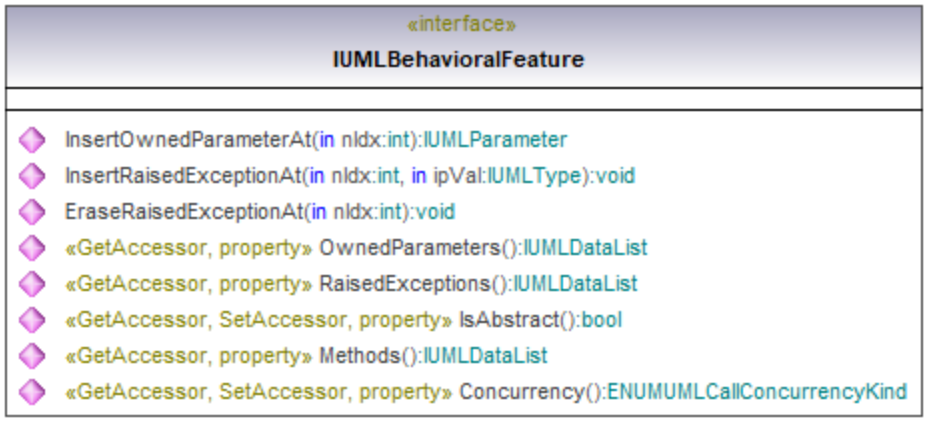
Operation **IUMLBehavior::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLConstraint 1142.					

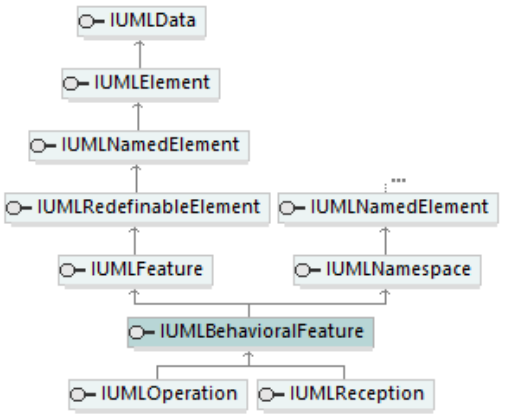
17.5.3.5.19 UModelAPI - IUMLBehavioralFeature

Interface **IUMLBehavioralFeature**

diagram



hierarchy



 © 2016-2022 Altova GmbH

typedElements	Interface IUMLBehavior 1110 Interface IUMLDataAll 1019	Operation BehaviorSpecification 1111 Operation BehaviorSpecification 1024

Operation **IUMLBehavioralFeature::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLCallConcurrencyKind 1369			

Operation **IUMLBehavioralFeature::EraseRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLBehavioralFeature::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLParameter 1244			

Operation **IUMLBehavioralFeature::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	IUMLType 1294			
	return	**return**	**void**			

Operation **IUMLBehavioralFeature::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLBehavioralFeature::Methods**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLBehavior 1110.					

Operation **IUMLBehavioralFeature::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLParameter 1244.					

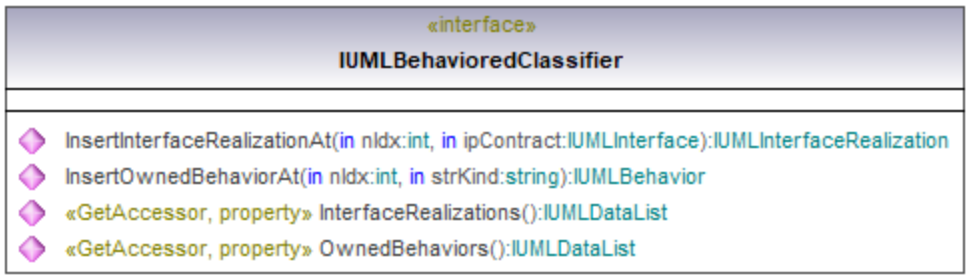
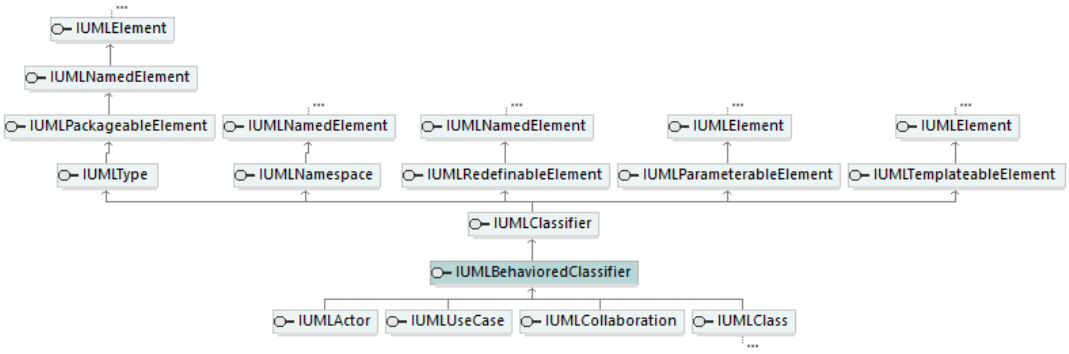
Operation **IUMLBehavioralFeature::RaisedExceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

documentation	A list of elements of type IUMLType 1294.

17.5.3.5.20 UModelAPI - IUMLBehavioredClassifier

Interface **IUMLBehavioredClassifier**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLInterfaceRealization 1202 Operation ImplementingClassifier 1038 Operation ImplementingClassifier 1203

Operation **IUMLBehavioredClassifier::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipContract	**in**	IUMLInterface 1199			
	return	**return**	IUMLInterfaceRealization 1202			

Operation **IUMLBehavioredClassifier::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			

 © 2016-2022 Altova GmbH

	return	**return**	IUMLBehavior 1110

Operation **IUMLBehavioredClassifier::InterfaceRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLInterfaceRealizations 1202.					

Operation **IUMLBehavioredClassifier::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLBehavior 1110.					

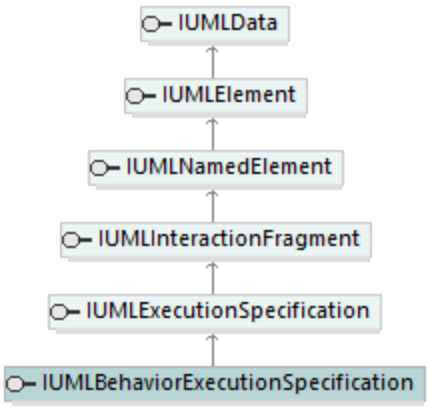
17.5.3.5.21 UModelAPI - IUMLBehaviorExecutionSpecification

Interface **IUMLBehaviorExecutionSpecification**

diagram



hierarchy



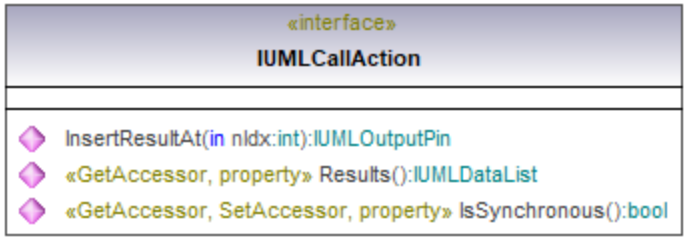
Operation **IUMLBehaviorExecutionSpecification::BehaviorExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior** 1110			

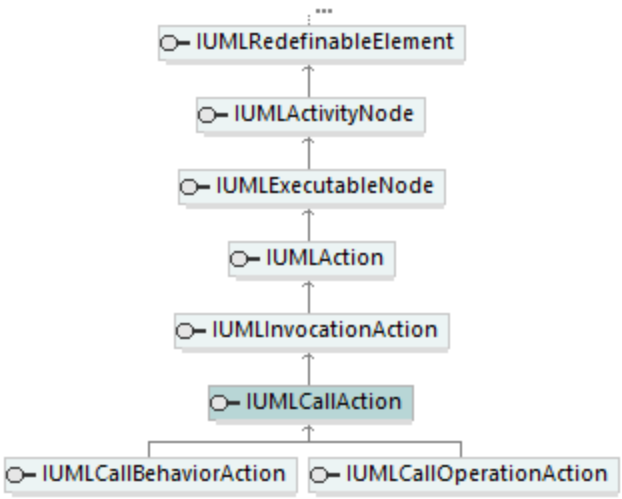
17.5.3.5.22 UModelAPI - IUMLCallAction

Interface **IUMLCallAction**

diagram



hierarchy



Operation **IUMLCallAction::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin** 1237			

Operation **IUMLCallAction::IsSynchronous**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

 © 2016-2022 Altova GmbH

Operation **IUMLCallAction::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLOutputPin 1237.					

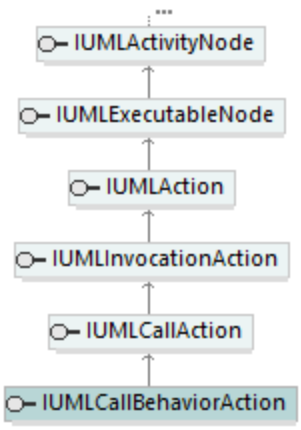
17.5.3.5.23 UModelAPI - IUMLCallBehaviorAction

Interface **IUMLCallBehaviorAction**

diagram



hierarchy



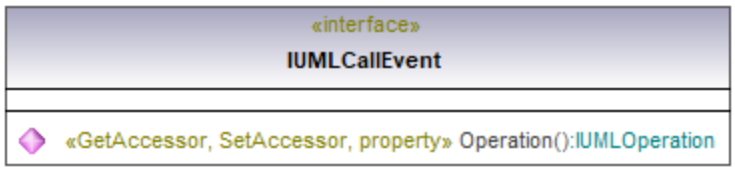
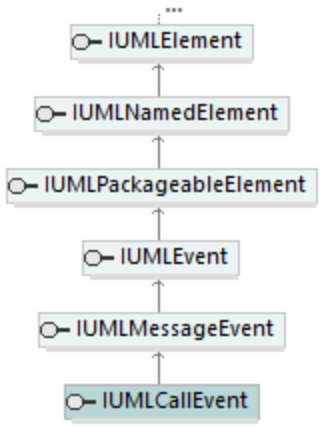
Operation **IUMLCallBehaviorAction::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

© 2016-2022 Altova GmbH

17.5.3.5.24 UModelAPI - IUMLCallEvent

Interface **IUMLCallEvent**

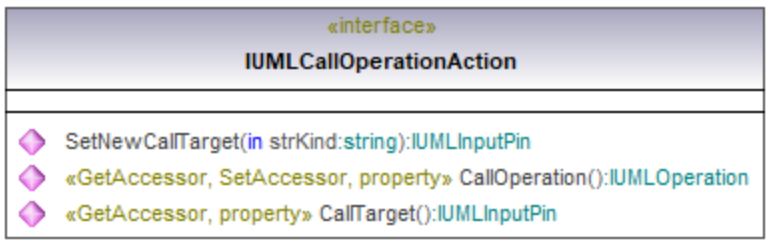
diagram	
hierarchy	

Operation **IUMLCallEvent::Operation**

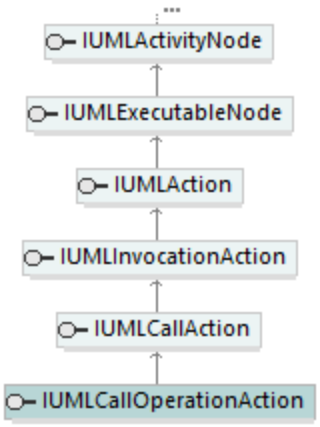
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOperation 1236			

17.5.3.5.25 UModelAPI - IUMLCallOperationAction

Interface **IUMLCallOperationAction**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	

Operation **IUMLCallOperationAction::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLCallOperationAction::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin** 1189			

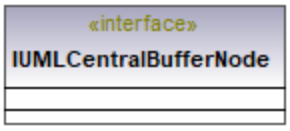
Operation **IUMLCallOperationAction::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin** 1189			

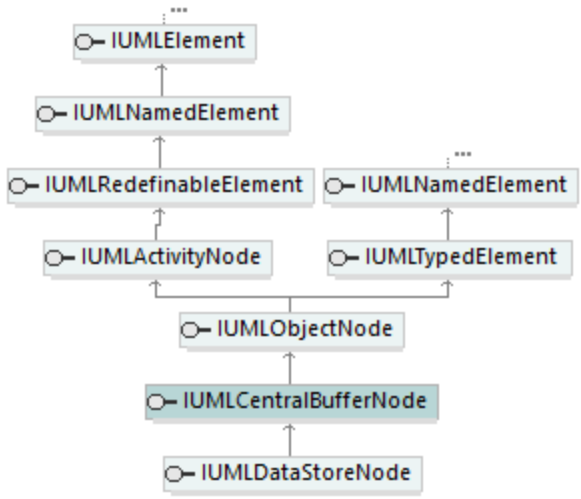
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.26 UModelAPI - IUMLCentralBufferNode

Interface **IUMLCentralBufferNode**

diagram	

hierarchy



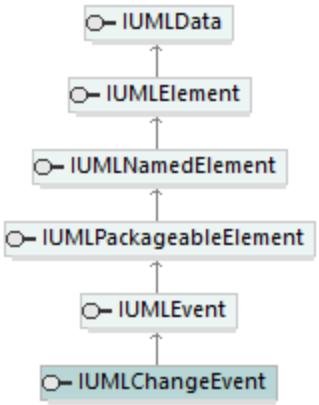
17.5.3.5.27 UModelAPI - IUMLChangeEvent

Interface **IUMLChangeEvent**

diagram



hierarchy



 © 2016-2022 Altova GmbH

Operation **IUMLChangeEvent::ChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

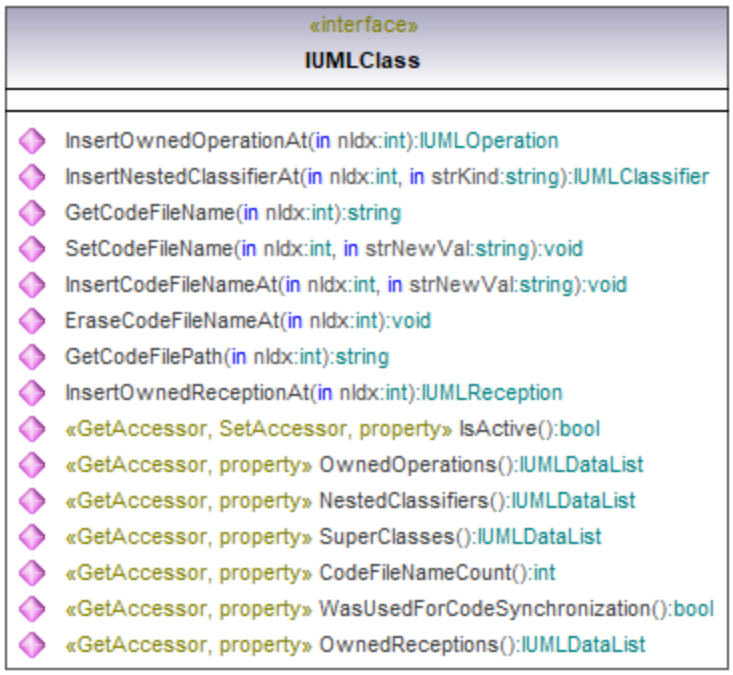
Operation **IUMLChangeEvent::SetNewChangeExpression**

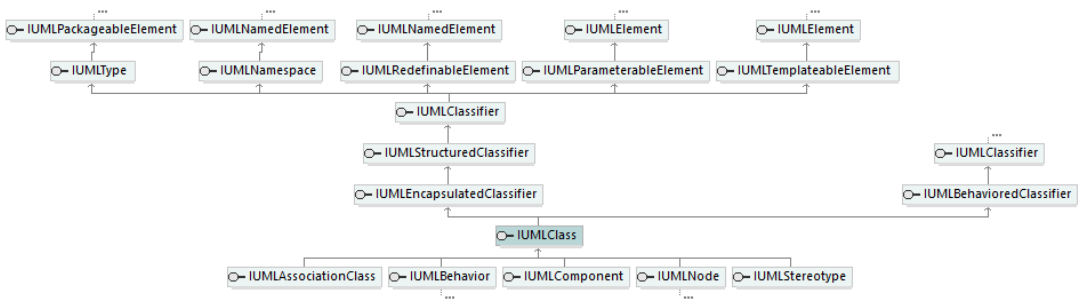
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

17.5.3.5.28 UModelAPI - IUMLClass

Interface **IUMLClass**

diagram



hierarchy	
typedElements	Interface **IUMLClassifier** 1124 Interface **IUMLDataAll** 1019 Interface **IUMLOperation** 1236 Interface **IUMLProperty** 1251 Interface **IUMLReception** 1259 Operation **Class** 1125 Operation **Class** 1025 Operation **Class** 1237 Operation **Class** 1252 Operation **Class** 1259

Operation **IUMLClass::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLClass::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLClass::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLClass::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			
documentation	get the full code file path					

Operation **IUMLClass::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLClass::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			

© 2016-2022 Altova GmbH

	return	**return**	**IUMLClassifier** 1124

Operation **IUMLClass::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLClass::InsertOwnedReceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLReception** 1259			

Operation **IUMLClass::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClass::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLClass::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLOperation 1236.					

Operation **IUMLClass::OwnedReceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLClass::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLClass::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClass 1121.					

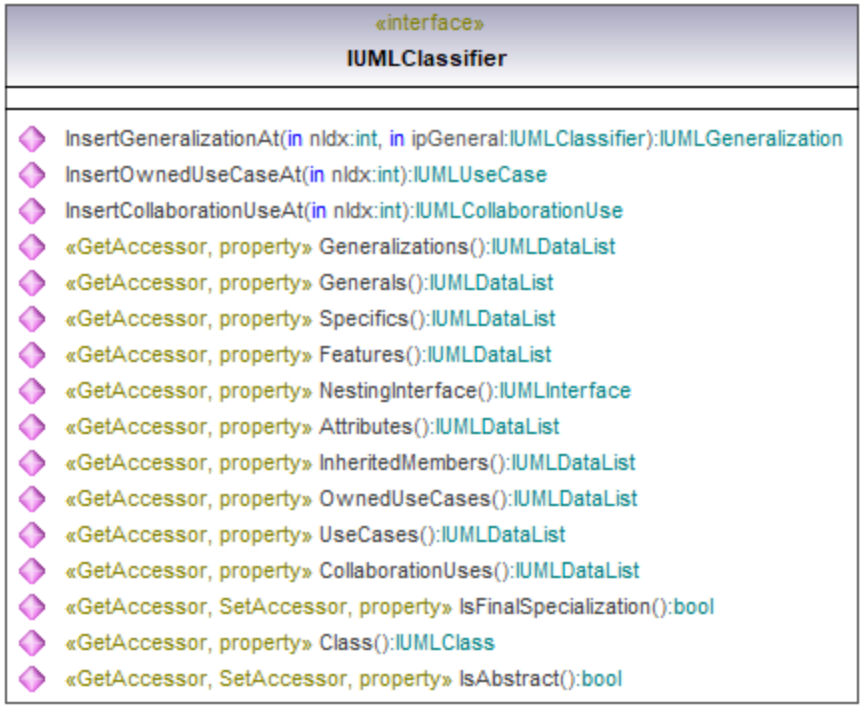
Operation **IUMLClass::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

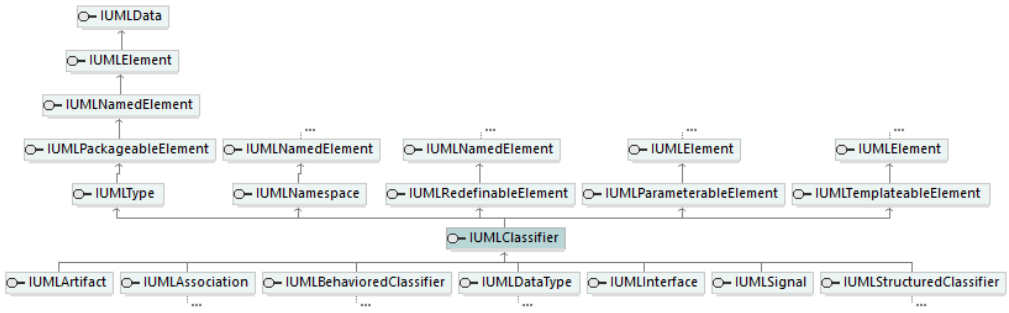
17.5.3.5.29 UModelAPI - IUMLClassifier

Interface **IUMLClassifier**

diagram



hierarchy



typedElements

Interface **IUMLAction** 1092
Interface **IUMLClass** 1121
Operation **ActionContext** 1092
Operation **InsertNestedClassifierAt** 1122

 © 2016-2022 Altova GmbH

	Interface **IUMLClassifier** 1124	Operation **InsertGeneralizationAt** 1126
	Interface **IUMLClassifierTemplateParameter** 1127	Operation **InsertConstrainingClassifierAt** 1128
	Interface **IUMLComponent** 1135	Operation **InsertRealizationAt** 1135
	Interface **IUMLComponentRealization** 1136	Operation **RealizingClassifier** 1137
	Interface **IUMLDataAll** 1019	Operation **ActionContext** 1019 **Classifier** 1025 **General** 1035 **InsertConstrainingClassifierAt** 1041 **InsertConveyedAt** 1041 **InsertExceptionTypeAt** 1042 **InsertGeneralizationAt** 1043 **InsertNestedClassifierAt** 1046 **InsertRealizationAt** 1051 **InsertSubjectAt** 1052 **RealizingClassifier** 1072 **Specific** 1082
	Interface **IUMLExceptionHandler** 1166	Operation **InsertExceptionTypeAt** 1167
	Interface **IUMLGeneralization** 1180	Operation **General** 1181 **Specific** 1181
	Interface **IUMLInformationFlow** 1186	Operation **InsertConveyedAt** 1187
	Interface **IUMLInstanceSpecification** 1190	Operation **Classifier** 1191
	Interface **IUMLInterface** 1199	Operation **InsertNestedClassifierAt** 1201
	Interface **IUMLProperty** 1251	Operation **Classifier** 1252
	Interface **IUMLUseCase** 1296	Operation **InsertSubjectAt** 1298

Operation **IUMLClassifier::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

Operation **IUMLClassifier::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass** 1121			

Operation **IUMLClassifier::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLCollaborationUse 1130.					

Operation **IUMLClassifier::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLFeature 1175.					

Operation **IUMLClassifier::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLGeneralization 1180.					

Operation **IUMLClassifier::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLClassifier::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLNamedElement 1222.					

Operation **IUMLClassifier::InsertCollaborationUseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLCollaborationUse** 1130			

Operation **IUMLClassifier::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipGeneral	**in**	**IUMLClassifier** 1124			
	return	**return**	**IUMLGeneralization** 1180			

Operation **IUMLClassifier::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLUseCase** 1296			

Operation **IUMLClassifier::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClassifier::IsFinalSpecialization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClassifier::NestingInterface**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

Operation **IUMLClassifier::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLUseCase 1296.					

Operation **IUMLClassifier::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLClassifier::UseCases**

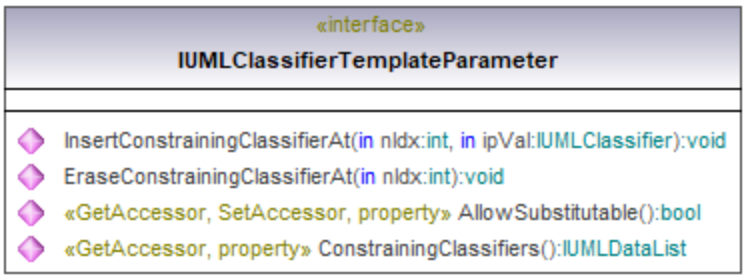
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLUseCase 1296.					

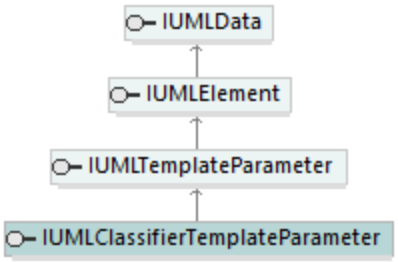
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.30 UModelAPI - IUMLClassifierTemplateParameter

Interface **IUMLClassifierTemplateParameter**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLTemplateSignature** 1285 Operation **InsertOwnedTemplateParameterAt** 1049 Operation **InsertOwnedTemplateParameterAt** 1286

Operation **IUMLClassifierTemplateParameter::AllowSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLClassifierTemplateParameter::ConstrainingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLClassifierTemplateParameter::EraseConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

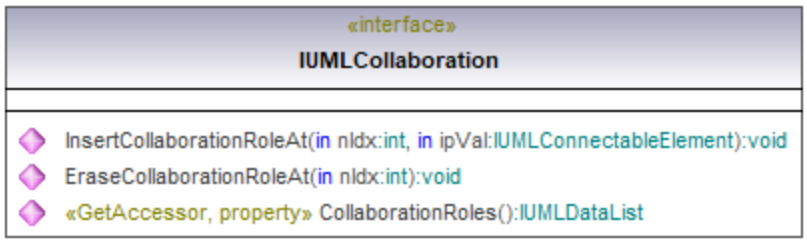
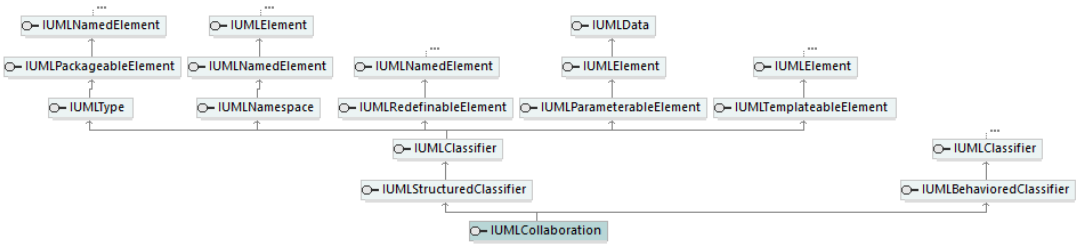
Operation **IUMLClassifierTemplateParameter::InsertConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.3.5.31 UModelAPI - IUMLCollaboration

Interface **IUMLCollaboration**

diagram	
hierarchy	
typedElements	Interface **IUMLCollaborationUse** 1130 Interface **IUMLDataAll** 1019 Operation **CollaborationType** 1130 Operation **CollaborationType** 1025

Operation **IUMLCollaboration::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLCollaboration 1129.					

Operation **IUMLCollaboration::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

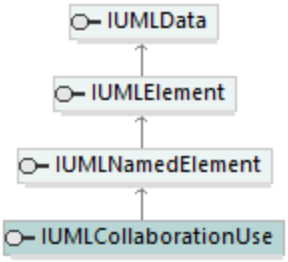
Operation **IUMLCollaboration::InsertCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnectableElement** 1137			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.3.5.32 UModelAPI - IUMLCollaborationUse

Interface **IUMLCollaborationUse**

diagram	
hierarchy	
typedElements	Interface IUMLClassifier 1124 Interface IUMLDataAll 1019 Operation InsertCollaborationUseAt 1126 Operation InsertCollaborationUseAt 1041

Operation **IUMLCollaborationUse::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLCollaboration 1129			

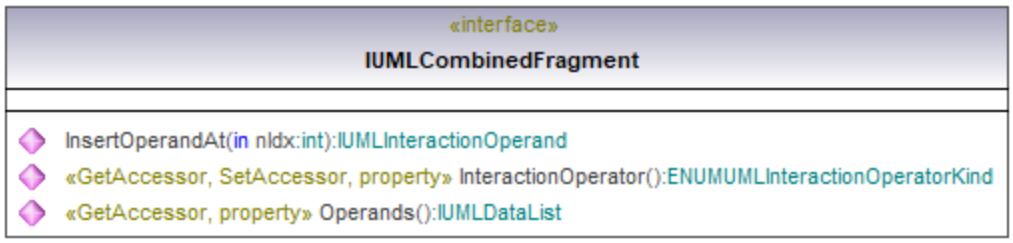
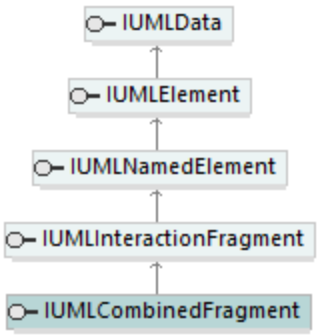
Operation **IUMLCollaborationUse::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLDependency 1148.					

© 2016-2022 Altova GmbH

17.5.3.5.33 UModelAPI - IUMLCombinedFragment

Interface **IUMLCombinedFragment**

diagram	
hierarchy	

Operation **IUMLCombinedFragment::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLInteractionOperand 1197			

Operation **IUMLCombinedFragment::InteractionOperator**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLInteractionOperatorKind 1374			

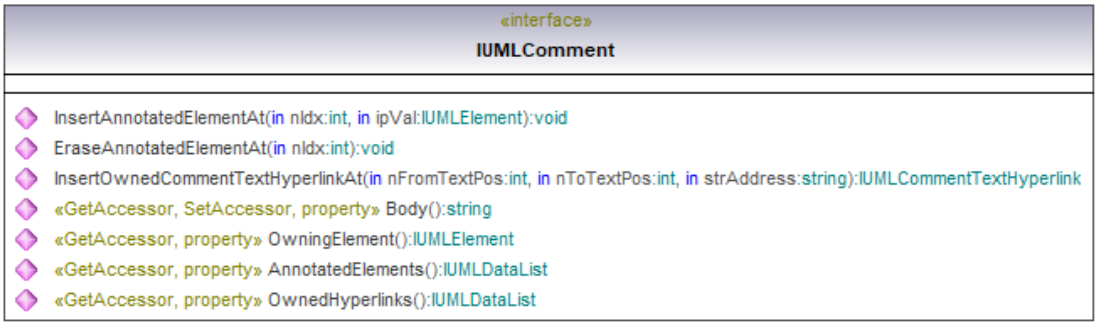
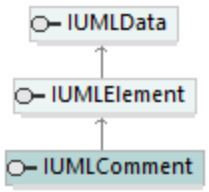
Operation **IUMLCombinedFragment::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLInteractionOperand 1197.					

© 2016-2022 Altova GmbH

17.5.3.5.34 UModelAPI - IUMLComment

Interface **IUMLComment**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLElement 1157 Operation InsertOwnedCommentAt 1047 OwnedDocComment 1066 Operation InsertOwnedCommentAt 1159 OwnedDocComment 1159

Operation **IUMLComment::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLElement 1157.					

Operation **IUMLComment::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLComment::EraseAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLComment::InsertAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

© 2016-2022 Altova GmbH

	ipVal	**in**	**IUMLElement** 1157
	return	**return**	**void**

Operation **IUMLComment::InsertOwnedCommentTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLCommentTextHyperlink** 1133			

Operation **IUMLComment::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLComment::OwningElement**

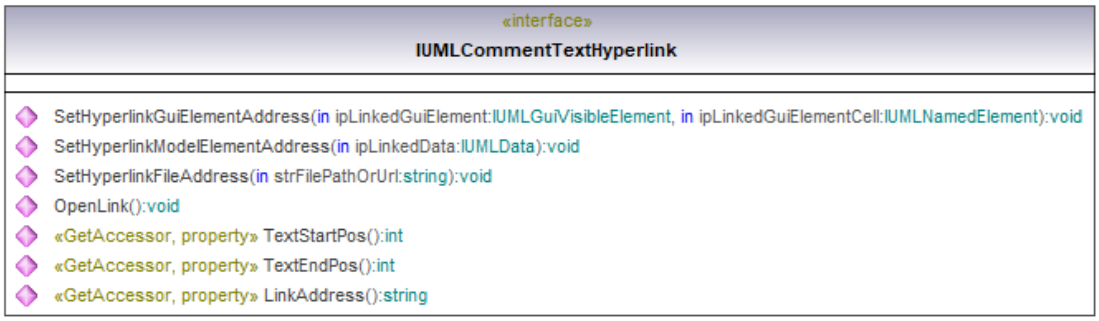
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

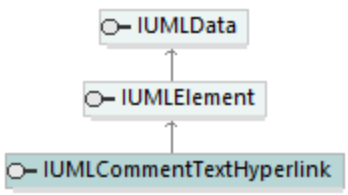
17.5.3.5.35 UModelAPI - IUMLCommentTextHyperlink

Interface **IUMLCommentTextHyperlink**

diagram



hierarchy



typedElements	Interface IUMLComment 1132 Interface IUMLDataAll 1019	Operation InsertOwnedCommentTextHyperlinkAt 1133 Operation InsertOwnedCommentTextHyperlinkAt 1047

Operation **IUMLCommentTextHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLCommentTextHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement** 1365			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement** 1222			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData** 1011			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

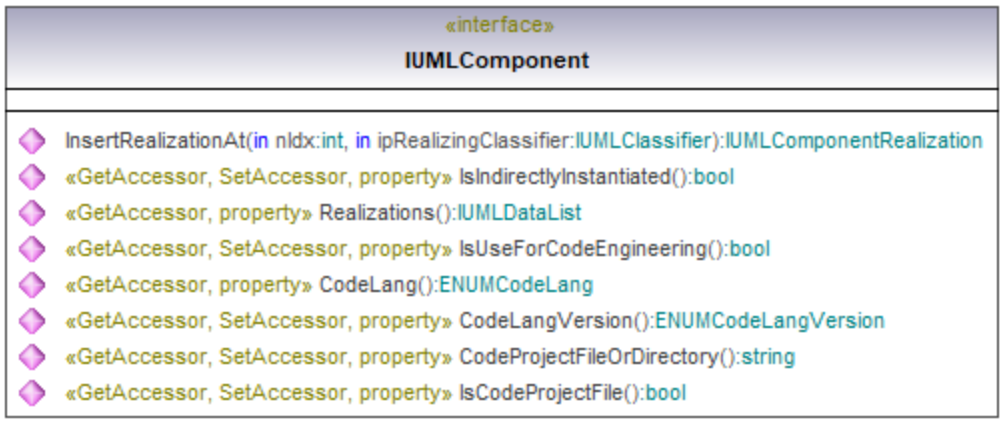
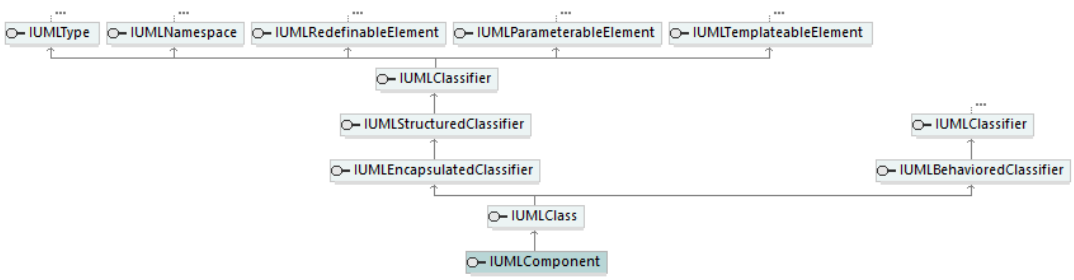
Operation **IUMLCommentTextHyperlink::TextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

© 2016-2022 Altova GmbH

17.5.3.5.36 UModelAPI - IUMLComponent

Interface **IUMLComponent**

diagram	
hierarchy	
typedElements	Interface **IUMLComponentRealization** 1136 Interface **IUMLDataAll** 1019 Operation **Abstraction** 1137 Operation **Abstraction** 1019

Operation **IUMLComponent::CodeLang**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang** 1004			

Operation **IUMLComponent::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion** 1005			

Operation **IUMLComponent::CodeProjectFileOrDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLComponent::InsertRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	ipRealizingClassifier	in	IUMLClassifier 1124			
	return	return	IUMLComponentRealization 1136			

Operation **IUMLComponent::IsCodeProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLComponent::IsIndirectlyInstantiated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLComponent::IsUseForCodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLComponent::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList 1013			
documentation	A list of elements of type IUMLComponentRealization 1136.					

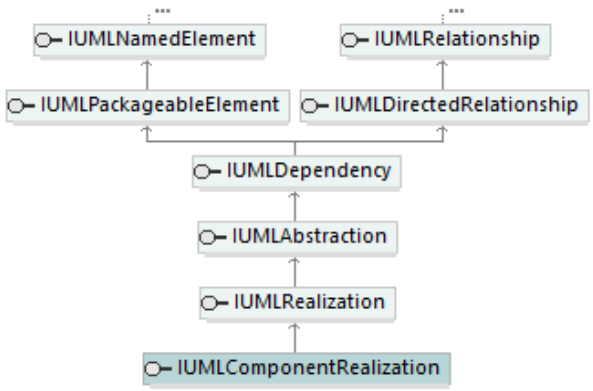
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.37 UModelAPI - IUMLComponentRealization

Interface **IUMLComponentRealization**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLComponent** 1135 Interface **IUMLDataAll** 1019 Operation **InsertRealizationAt** 1135 Operation **InsertRealizationAt** 1051

Operation **IUMLComponentRealization::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComponent** 1135			

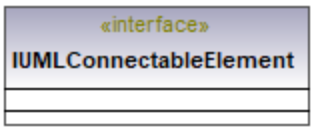
Operation **IUMLComponentRealization::RealizingClassifier**

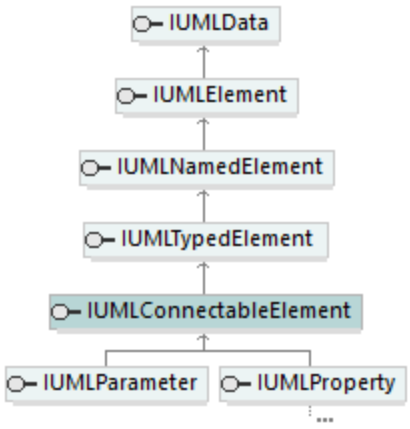
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.38 UModelAPI - IUMLConnectableElement

Interface **IUMLConnectableElement**

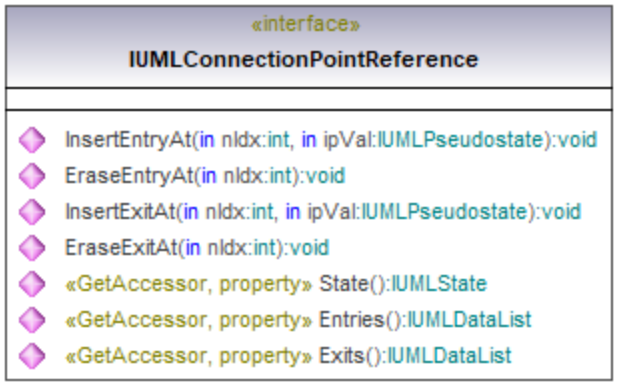
diagram	

hierarchy	
typedElements	Interface IUMLCollaboration 1129 Interface IUMLConnectorEnd 1141 Interface IUMLDataAll 1019 Interface IUMLLifeline 1209 Interface IUMLStructuredClassifier 1279 Operation InsertCollaborationRoleAt 1129 Operation Role 1141 Operation InsertCollaborationRoleAt 1041 InsertOwnedConnectorAt 1047 Represents 1073 Role 1074 Operation Represents 1209 Operation InsertOwnedConnectorAt 1279

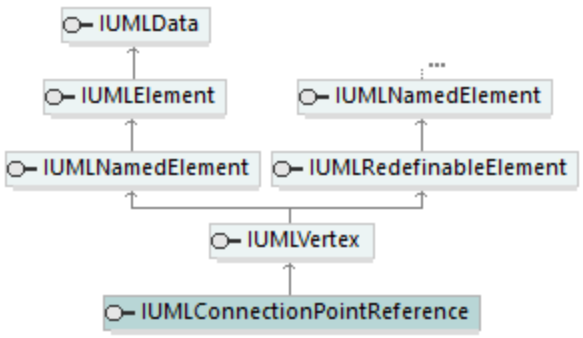
UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.5.39 UModelAPI - IUMLConnectionPointReference

Interface **IUMLConnectionPointReference**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLState** 1268 Operation **InsertConnectionAt** 1041 Operation **InsertConnectionAt** 1269

Operation **IUMLConnectionPointReference::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPseudostate 1257.					

Operation **IUMLConnectionPointReference::EraseEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::EraseExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::Exits**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPseudostate 1257.					

Operation **IUMLConnectionPointReference::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate** 1257			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	ipVal	**in**	**IUMLPseudostate** 1257
	return	**return**	**void**

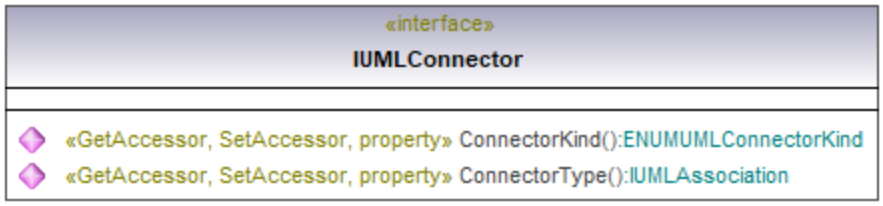
Operation **IUMLConnectionPointReference::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState** 1268			

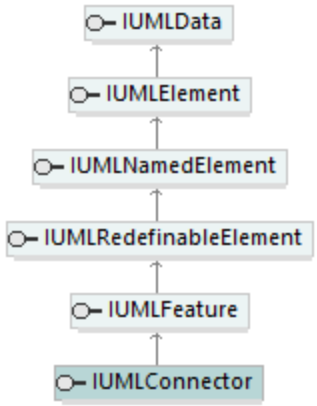
17.5.3.5.40 UModelAPI - IUMLConnector

Interface **IUMLConnector**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019

Interface **IUMLInformationFlow** 1186

Interface **IUMLStructuredClassifier** 1279

Operation **InsertOwnedConnectorAt** 1047

InsertRealizingConnectorAt 1051

Operation **InsertRealizingConnectorAt** 1188

Operation **InsertOwnedConnectorAt** 1279

Operation **IUMLConnector::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2016-2022 Altova GmbH

	return	**return**	**ENUMUMLConnectorKind** 1370
documentation	**Deprecated:** Since UML2.3 (UModel2010r2) 'ConnectorKind' is derived and cannot be set anymore.		

Operation **IUMLConnector::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation** 1108			

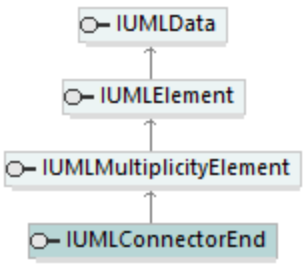
17.5.3.5.41 UModelAPI - IUMLConnectorEnd

Interface **IUMLConnectorEnd**

diagram



hierarchy



Operation **IUMLConnectorEnd::Role**

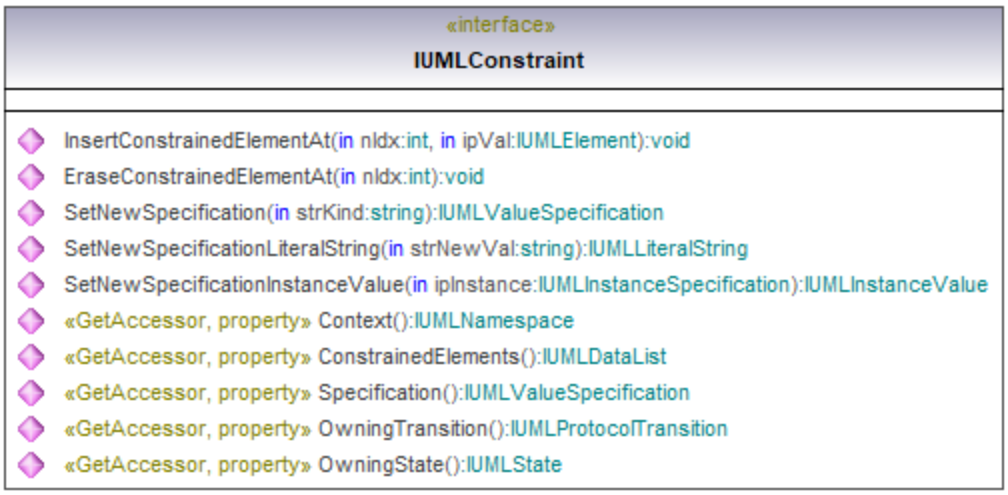
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement** 1137			

© 2016-2022 Altova GmbH

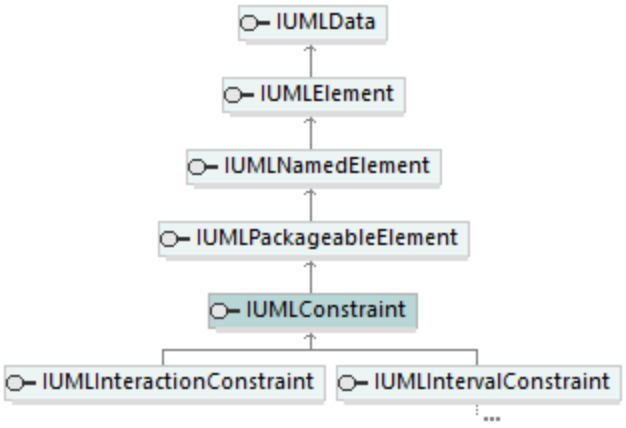
17.5.3.5.42 UModelAPI - IUMLConstraint

Interface **IUMLConstraint**

diagram



hierarchy



typedElements

Interface **IUMLAction** 1092

Interface **IUMLBehavior** 1110

Interface **IUMLDataAll** 1019

Operation **InsertLocalPostConditionAt** 1093
InsertLocalPreConditionAt 1093
Operation **InsertPostconditionAt** 1111
InsertPreconditionAt 1111
Operation **InsertLocalPostConditionAt** 1045
InsertLocalPreConditionAt 1045
InsertOwnedRuleAt 1049
InsertPostconditionAt 1050
InsertPreconditionAt 1051
Invariant 1054 **OwningConstraint** 1068
PostCondition 1071
PreCondition 1071
SetNewInvariant 1077
SetNewPostCondition 1078
SetNewPreCondition 1078
SetNewStateInvariant 1079

© 2016-2022 Altova GmbH

	SetNewTransitionGuard 1079 StateInvariant 1083 TransitionGuard 1086
Interface IUMLNamespace 1225	Operation InsertOwnedRuleAt 1226
Interface IUMLProtocolTransition 1256	Operation PostCondition 1256 PreCondition 1256 SetNewPostCondition 1256 SetNewPreCondition 1257
Interface IUMLState 1268	Operation SetNewStateInvariant 1270 StateInvariant 1271
Interface IUMLStateInvariant 1271	Operation Invariant 1272 SetNewInvariant 1272
Interface IUMLTransition 1291	Operation SetNewTransitionGuard 1292 TransitionGuard 1292
Interface IUMLValueSpecification 1299	Operation OwningConstraint 1301

Operation **IUMLConstraint::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLElement 1157.					

Operation **IUMLConstraint::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamespace 1225			

Operation **IUMLConstraint::EraseConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLConstraint::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	IUMLElement 1157			
	return	**return**	**void**			

Operation **IUMLConstraint::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLState 1268			

Operation **IUMLConstraint::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProtocolTransition 1256			

Operation **IUMLConstraint::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLConstraint::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLConstraint::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString** 1213			

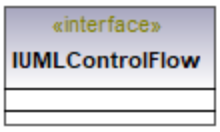
Operation **IUMLConstraint::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

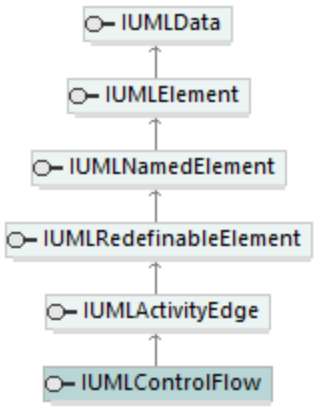
17.5.3.5.43 UModelAPI - IUMLControlFlow

Interface **IUMLControlFlow**

diagram	
	

 © 2016-2022 Altova GmbH

hierarchy



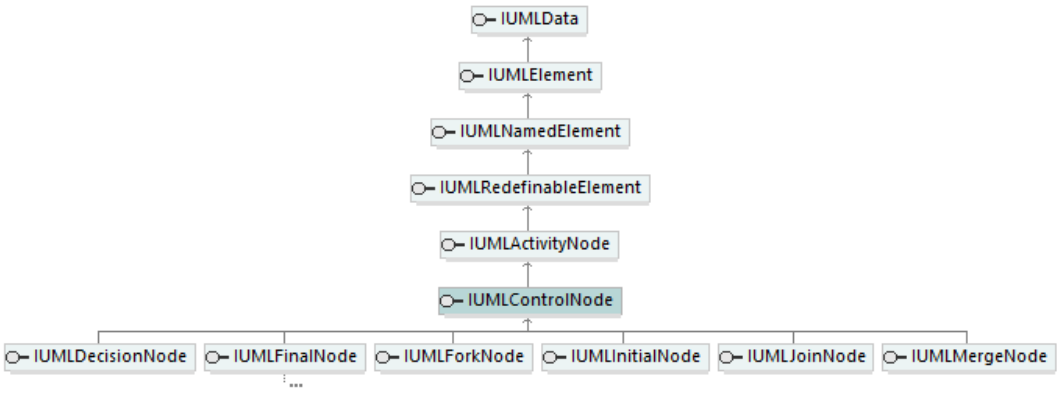
17.5.3.5.44 UModelAPI - IUMLControlNode

Interface **IUMLControlNode**

diagram



hierarchy

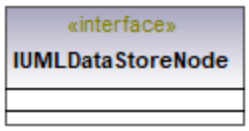


© 2016-2022 Altova GmbH

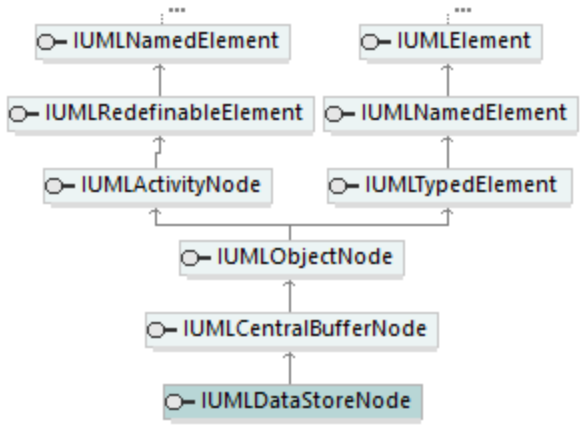
17.5.3.5.45 UModelAPI - IUMLDataStoreNode

Interface **IUMLDataStoreNode**

diagram



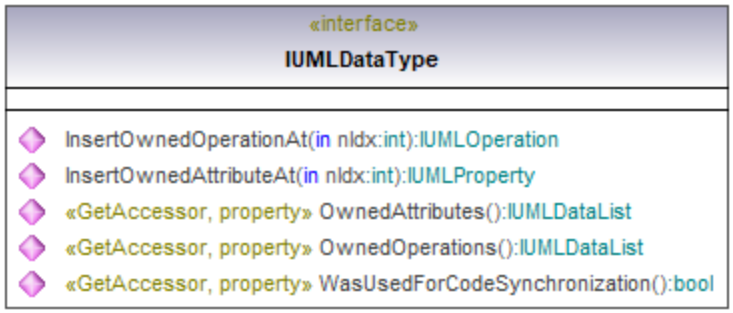
hierarchy



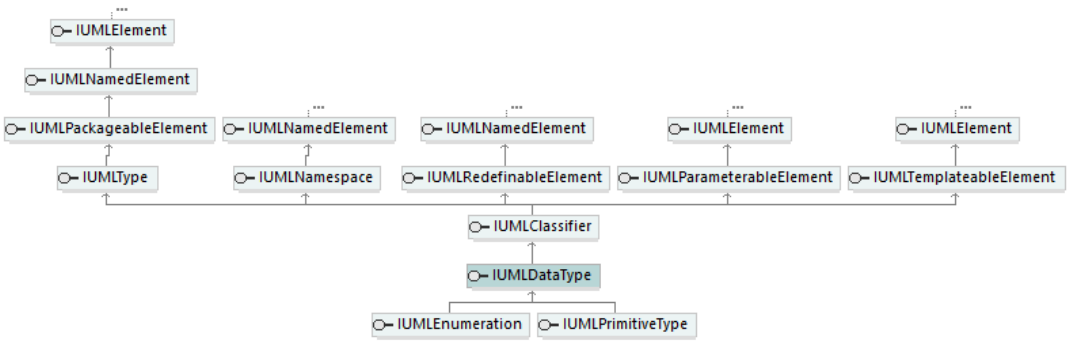
17.5.3.5.46 UModelAPI - IUMLDataType

Interface **IUMLDataType**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLOperation** 1236 Interface **IUMLProperty** 1251 Operation **Datatype** 1027 Operation **Datatype** 1237 Operation **Datatype** 1253

Operation **IUMLDataType::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLDataType::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLDataType::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

Operation **IUMLDataType::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLOperation 1236.					

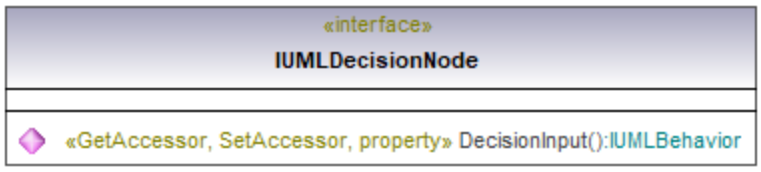
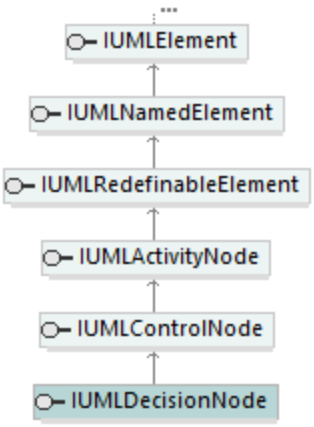
Operation **IUMLDataType::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

© 2016-2022 Altova GmbH

17.5.3.5.47 UModelAPI - IUMLDecisionNode

Interface **IUMLDecisionNode**

diagram	
hierarchy	

Operation **IUMLDecisionNode::DecisionInput**

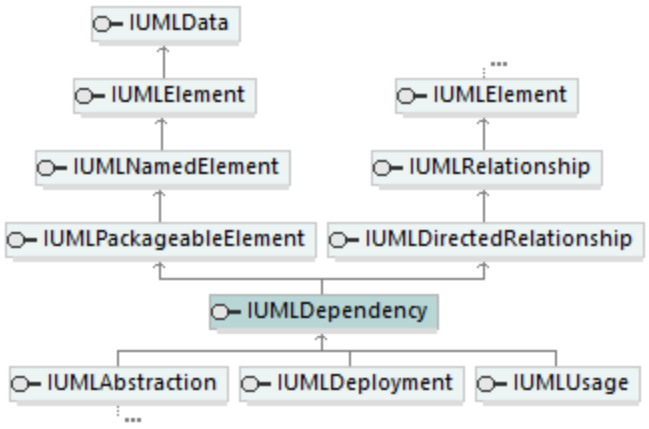
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

17.5.3.5.48 UModelAPI - IUMLDependency

Interface **IUMLDependency**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	

Operation **IUMLDependency::Clients**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLNamedElement 1222.					

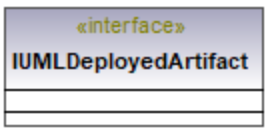
Operation **IUMLDependency::Suppliers**

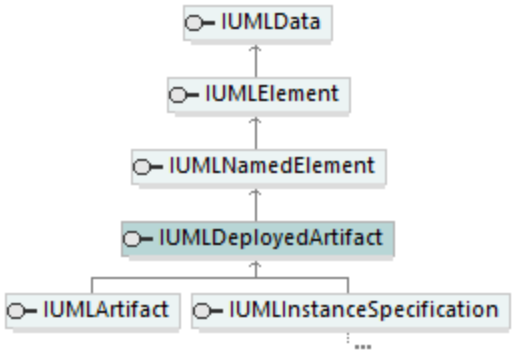
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLNamedElement 1222.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.49 UModelAPI - IUMLDeployedArtifact

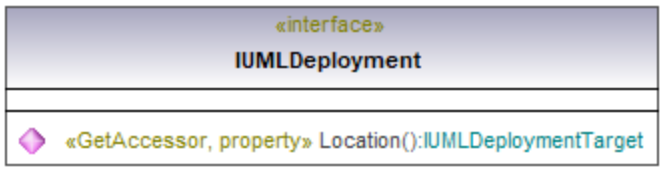
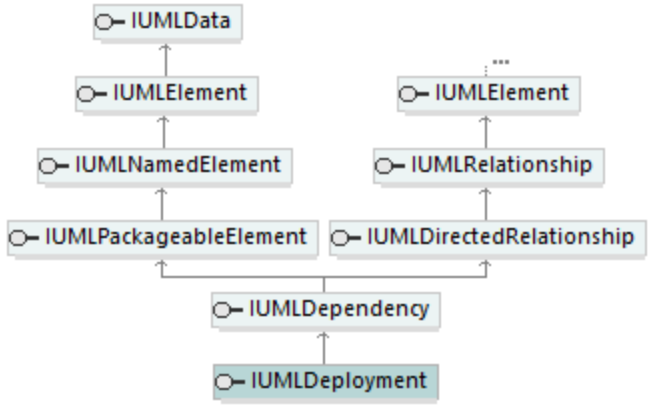
Interface **IUMLDeployedArtifact**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLDeploymentTarget** 1151 Operation **InsertDeploymentAt** 1042 Operation **InsertDeploymentAt** 1152

17.5.3.5.50 UModelAPI - IUMLDeployment

Interface **IUMLDeployment**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Operation **InsertDeploymentAt** 1042

 © 2016-2022 Altova GmbH

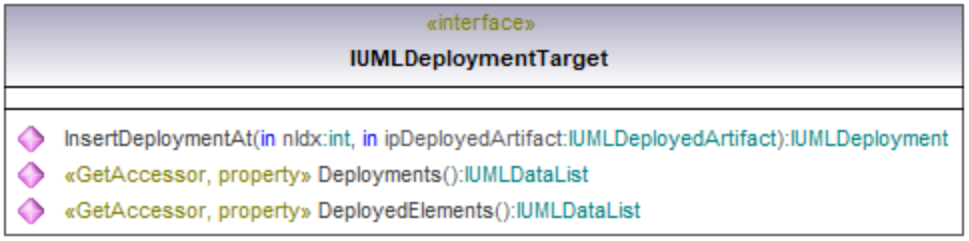
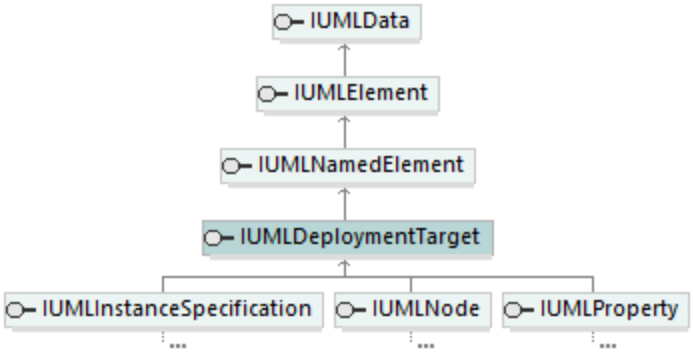
	Interface **IUMLDeploymentTarget** 1151 Operation **InsertDeploymentAt** 1152

Operation **IUMLDeployment::Location**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDeploymentTarget** 1151			

17.5.3.5.51 UModelAPI - IUMLDeploymentTarget

Interface **IUMLDeploymentTarget**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLDeployment** 1150 Operation **Location** 1061 Operation **Location** 1151

Operation **IUMLDeploymentTarget::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPackageableElement 1241.					

Operation **IUMLDeploymentTarget::Deployments**

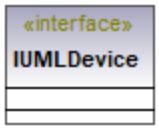
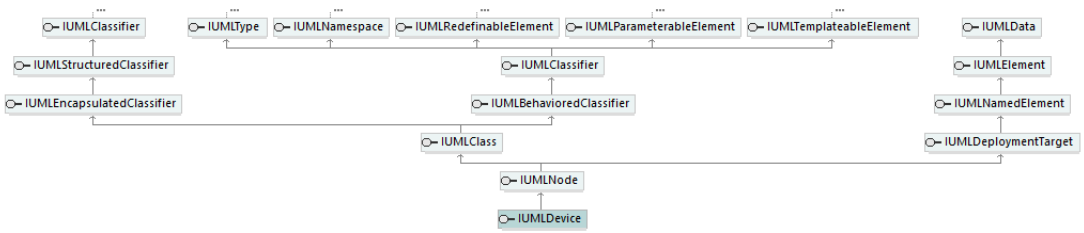
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLDeployment 1150.					

Operation **IUMLDeploymentTarget::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDeployedArtifact	**in**	**IUMLDeployedArtifact** 1149			
	return	**return**	**IUMLDeployment** 1150			

17.5.3.5.52 UModelAPI - IUMLDevice

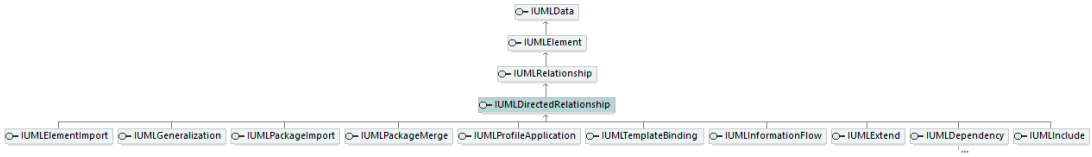
Interface **IUMLDevice**

diagram	
hierarchy	

 © 2016-2022 Altova GmbH

17.5.3.5.53 UModelAPI - IUMLDirectedRelationship

Interface **IUMLDirectedRelationship**

diagram	
hierarchy	

Operation **IUMLDirectedRelationship::Sources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLElement 1157.					

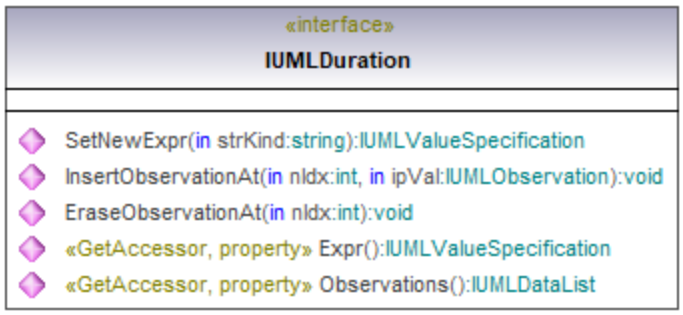
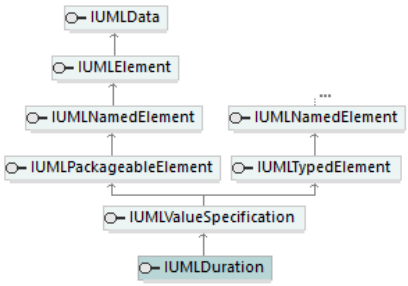
Operation **IUMLDirectedRelationship::Targets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLElement 1157.					

© 2016-2022 Altova GmbH

17.5.3.5.54 UModelAPI - IUMLDuration

Interface **IUMLDuration**

diagram	
hierarchy	

Operation **IUMLDuration::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLDuration::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLDuration::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation** 1231			
	return	**return**	**void**			

Operation **IUMLDuration::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLObservation 1231.					

© 2016-2022 Altova GmbH

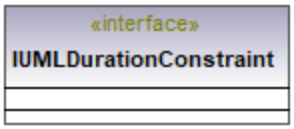
Operation **IUMLDuration::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

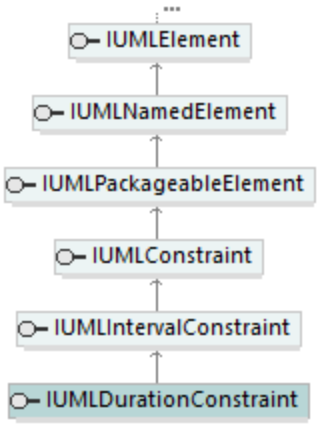
17.5.3.5.55 UModelAPI - IUMLDurationConstraint

Interface **IUMLDurationConstraint**

diagram



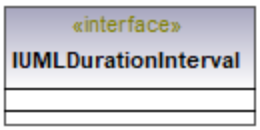
hierarchy



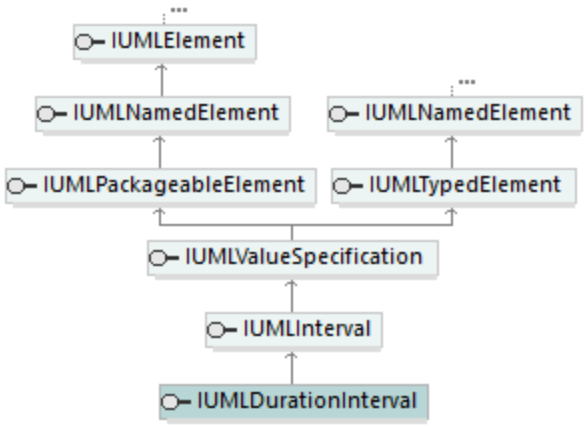
17.5.3.5.56 UModelAPI - IUMLDurationInterval

Interface **IUMLDurationInterval**

diagram



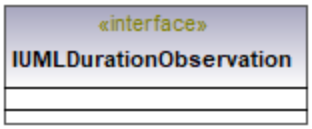
hierarchy



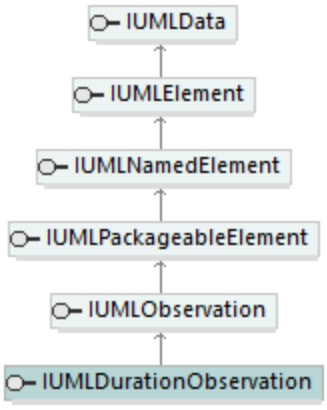
17.5.3.5.57 UModelAPI - IUMLDurationObservation

Interface **IUMLDurationObservation**

diagram



hierarchy

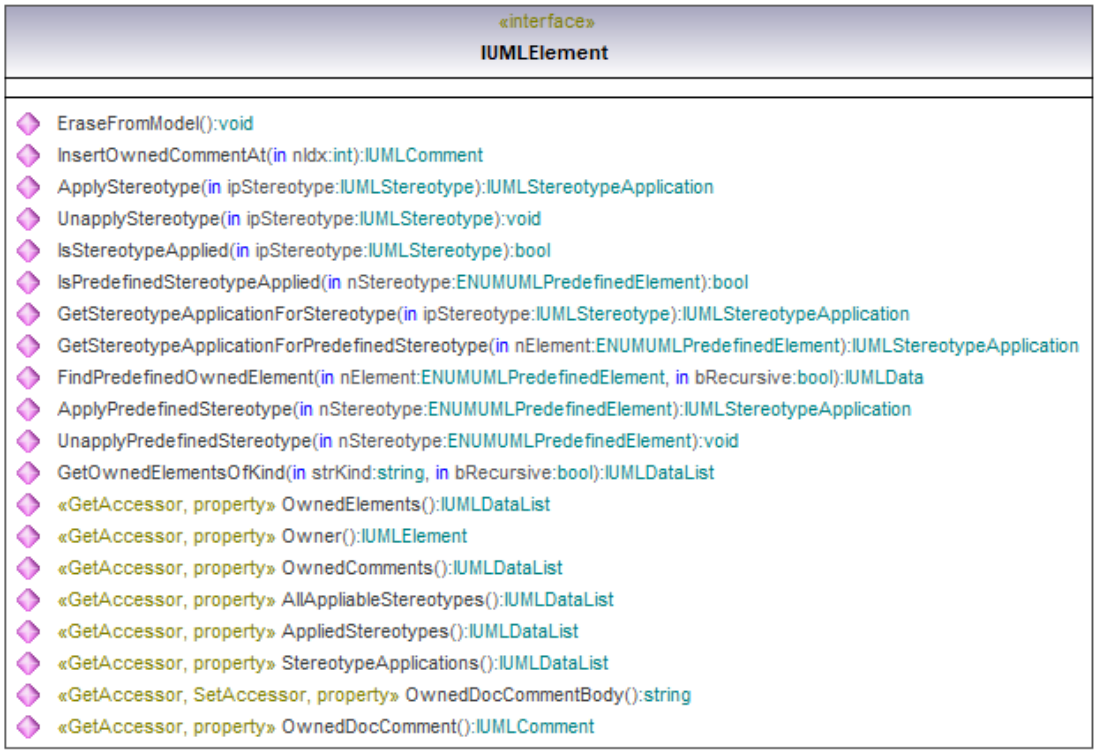


 © 2016-2022 Altova GmbH

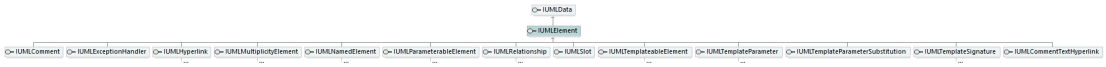
17.5.3.5.58 UModelAPI - IUMLElement

Interface **IUMLElement**

diagram



hierarchy



typedElements

Interface IGenerateSequenceDiagramDlg 954

Interface IUMLComment 1132
Interface IUMLConstraint 1142
Interface IUMLDataAll 1019
Interface IUMLElement 1157
Interface IUMLGuiDiagram 1316
Interface IUMLGuiLink 1328
Interface IUMLGuiNodeLink 1331

Operation DiagramOwner 955

Operation InsertAnnotatedElementAt 1132
OwningElement 1133
Operation InsertConstrainedElementAt 1143
Operation AppliedElement 1022 Element 1029
InsertAnnotatedElementAt 1040
InsertConstrainedElementAt 1041
InsertPackagedElementRelationship 1050 IsElementVisible 1055
LinkedOwner 1060 Owner 1068
OwningElement 1068
SetElementVisible 1075
TextLabelElement 1085
Operation Owner 1160
Operation LinkedOwner 1319
Operation Element 1323
Operation IsElementVisible 1332
SetElementVisible 1332

	Interface IUMLGuiTextLabel 1356 Interface IUMLPackage 1238 Interface IUMLStereotypeApplication 1275	Operation TextLabelElement 1357 Operation InsertPackagedElementRelationshipAt 1239 Operation AppliedElement 1275

Operation **IUMLElement::AllAppliableStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLStereotype 1273.					

Operation **IUMLElement::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLStereotype 1273.					

Operation **IUMLElement::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	ENUMUMLPredefinedElement 1376			
	return	**return**	IUMLStereotypeApplication 1275			

Operation **IUMLElement::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	IUMLStereotype 1273			
	return	**return**	IUMLStereotypeApplication 1275			

Operation **IUMLElement::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			
documentation	Use this function to erase the element from the model and all diagrams. Use IUMLGuiDiagram 1316::EraseFromDiagram 1318 to erase from diagram only.					

Operation **IUMLElement::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	ENUMUMLPredefinedElement 1376			
	bRecursive	**in**	**bool**			
	return	**return**	IUMLData 1011			

Operation **IUMLElement::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	strKind	**in**	**string**
	bRecursive	**in**	**bool**
	return	**return**	**IUMLDataList** 1013
documentation	get all owned elements of the specified kind (*strKind*)		

Operation **IUMLElement::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement** 1376			
	return	**return**	**IUMLStereotypeApplication** 1275			

Operation **IUMLElement::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype** 1273			
	return	**return**	**IUMLStereotypeApplication** 1275			

Operation **IUMLElement::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLComment** 1132			

Operation **IUMLElement::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement** 1376			
	return	**return**	**bool**			

Operation **IUMLElement::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype** 1273			
	return	**return**	**bool**			

Operation **IUMLElement::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLComment 1132.					

Operation **IUMLElement::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComment** 1132			

Operation **IUMLElement::OwnedDocCommentBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLElement::OwnedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLElement 1157.					

Operation **IUMLElement::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

Operation **IUMLElement::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLStereotypeApplication 1275.					

Operation **IUMLElement::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement** 1376			
	return	**return**	**void**			

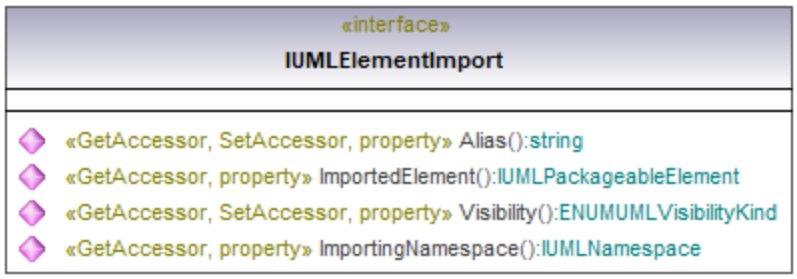
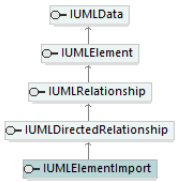
Operation **IUMLElement::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype** 1273			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.3.5.59 UModelAPI - IUMLElementImport

Interface **IUMLElementImport**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLNamespace 1225 Operation InsertElementImportAt 1042 Operation InsertElementImportAt 1226

Operation **IUMLElementImport::Alias**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLElementImport::ImportedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackageableElement 1241			

Operation **IUMLElementImport::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamespace 1225			

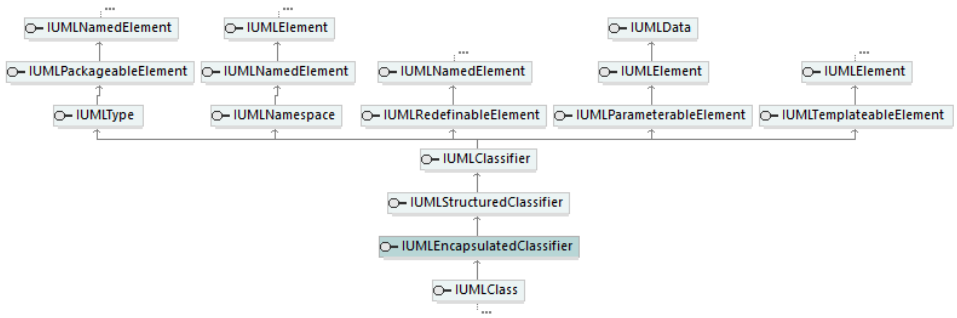
Operation **IUMLElementImport::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLVisibilityKind 1378			

© 2016-2022 Altova GmbH

17.5.3.5.60 UModelAPI - IUMLEncapsulatedClassifier

Interface **IUMLEncapsulatedClassifier**

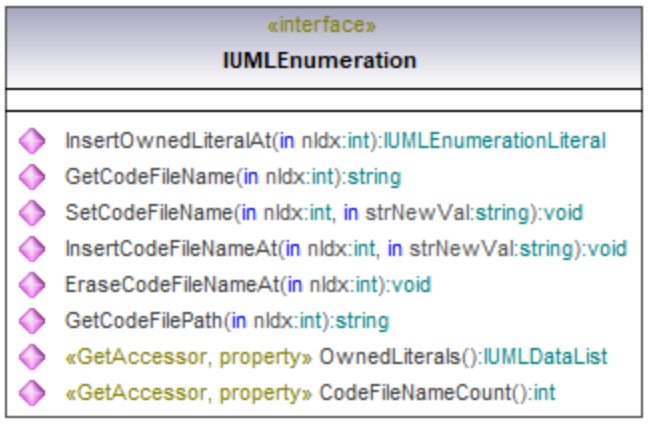
diagram	
hierarchy	

Operation **IUMLEncapsulatedClassifier::OwnedPorts**

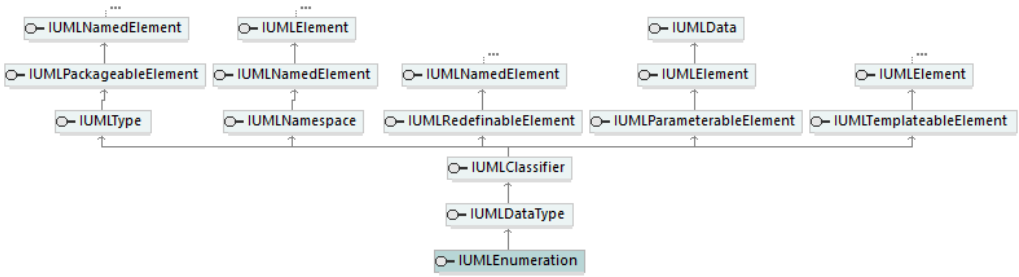
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

17.5.3.5.61 UModelAPI - IUMLEnumeration

Interface **IUMLEnumeration**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLEnumerationLiteral** 1164 Operation **Enumeration** 1029 Operation **Enumeration** 1164

Operation **IUMLEnumeration::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLEnumeration::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLEnumeration::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLEnumeration::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			
documentation	get the full code file path					

Operation **IUMLEnumeration::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLEnumeration::InsertOwnedLiteralAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLEnumerationLiteral** 1164			

Operation **IUMLEnumeration::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLEnumerationLiteral 1164.					

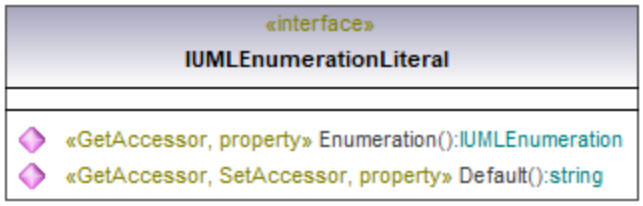
Operation **IUMLEnumeration::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

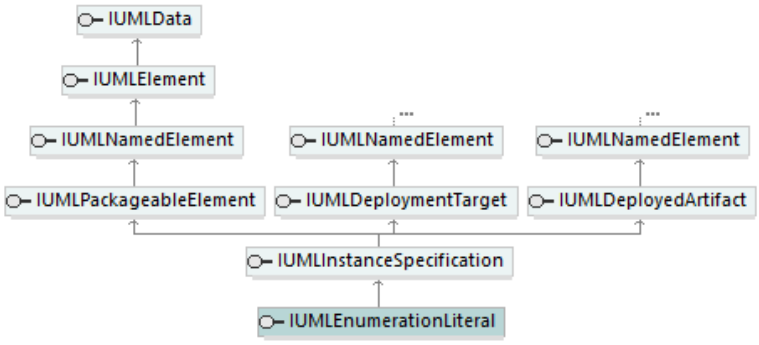
17.5.3.5.62 UModelAPI - IUMLEnumerationLiteral

Interface **IUMLEnumerationLiteral**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019
Interface **IUMLEnumeration** 1162
Operation **InsertOwnedLiteralAt** 1048
Operation **InsertOwnedLiteralAt** 1163

Operation **IUMLEnumerationLiteral::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLEnumerationLiteral::Enumeration**

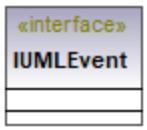
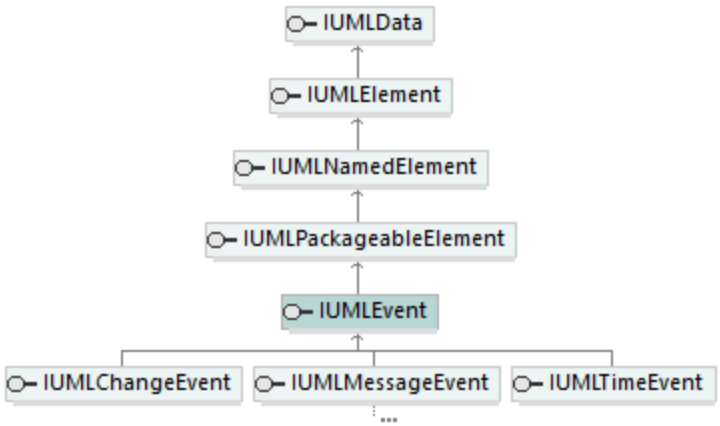
parameter	name	direction	type	type modifier	multiplicity	default

 © 2016-2022 Altova GmbH

	return	**return**	**IUMLEnumeration** 1162

17.5.3.5.63 UModelAPI - IUMLEvent

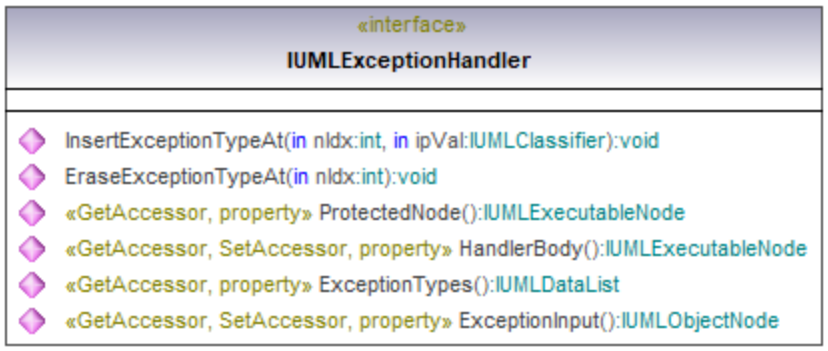
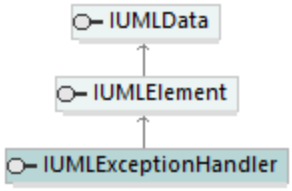
Interface **IUMLEvent**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLOccurrenceSpecification** 1231 Interface **IUMLTrigger** 1293 Operation **Event** 1032 **OccurringEvent** 1064 Operation **OccurringEvent** 1232 Operation **Event** 1294

© 2016-2022 Altova GmbH

17.5.3.5.64 UModelAPI - IUMLExceptionHandler

Interface **IUMLExceptionHandler**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLExecutableNode 1167 Operation InsertHandlerAt 1043 Operation InsertHandlerAt 1168

Operation **IUMLExceptionHandler::EraseExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExceptionHandler::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLObjectNode 1229			

Operation **IUMLExceptionHandler::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLExceptionHandler::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**IUMLExecutableNode** 1167

Operation **IUMLExceptionHandler::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

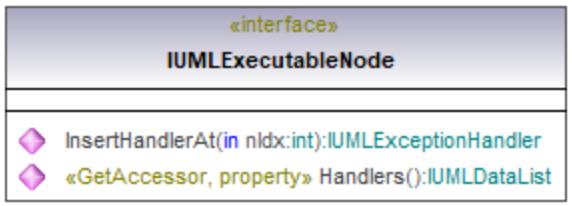
Operation **IUMLExceptionHandler::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode** 1167			

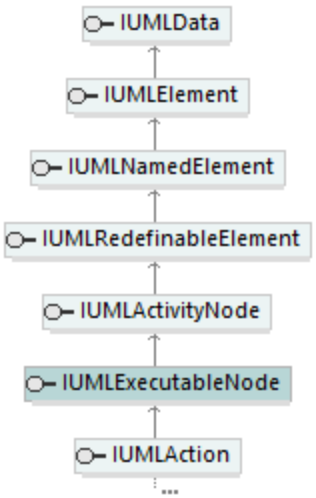
17.5.3.5.65 UModelAPI - IUMLExecutableNode

Interface **IUMLExecutableNode**

diagram



hierarchy



typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLExceptionHandler** 1166	Operation **HandlerBody** 1037 **ProtectedNode** 1072 Operation **HandlerBody** 1166 **ProtectedNode** 1167

Operation **IUMLExecutableNode::Handlers**

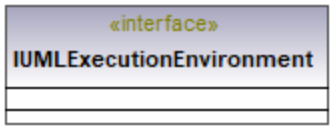
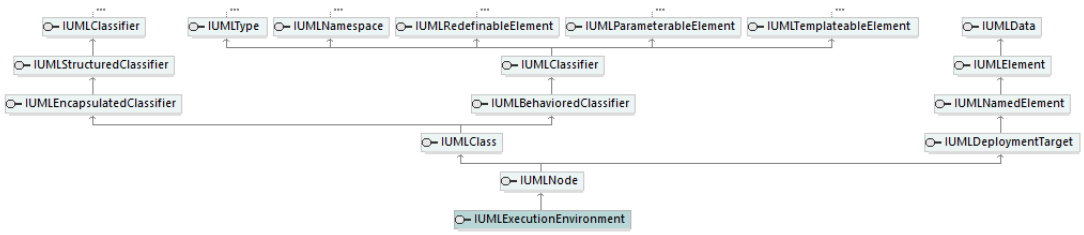
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLExceptionHandler 1166.					

Operation **IUMLExecutableNode::InsertHandlerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExceptionHandler** 1166			

17.5.3.5.66 UModelAPI - IUMLExecutionEnvironment

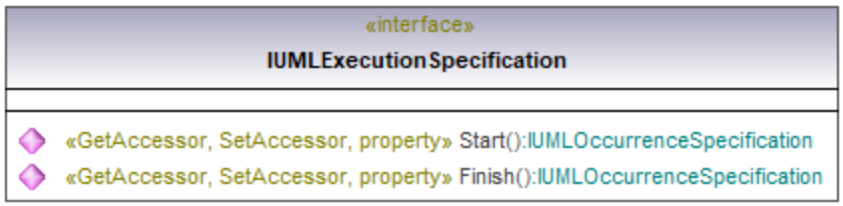
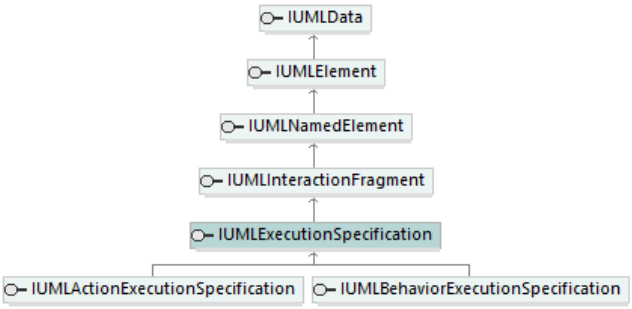
Interface **IUMLExecutionEnvironment**

diagram	
hierarchy	

 © 2016-2022 Altova GmbH

17.5.3.5.67 UModelAPI - IUMLExecutionSpecification

Interface **IUMLExecutionSpecification**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLOccurrenceSpecification 1231 Operation ExecutionSpecificationFinish 1033 ExecutionSpecificationStart 1033 Operation ExecutionSpecificationFinish 1232 ExecutionSpecificationStart 1232

Operation **IUMLExecutionSpecification::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOccurrenceSpecification 1231			

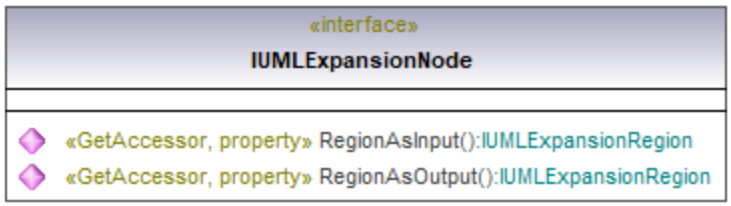
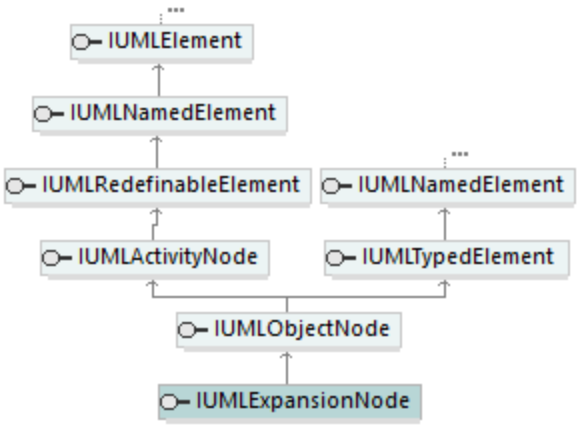
Operation **IUMLExecutionSpecification::Start**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOccurrenceSpecification 1231			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.5.68 UModelAPI - IUMLExpansionNode

Interface **IUMLExpansionNode**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLExpansionRegion 1171 Operation InsertInputElementAt 1044 InsertOutputElementAt 1047 Operation InsertInputElementAt 1172 InsertOutputElementAt 1172

Operation **IUMLExpansionNode::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExpansionRegion 1171			

Operation **IUMLExpansionNode::RegionAsOutput**

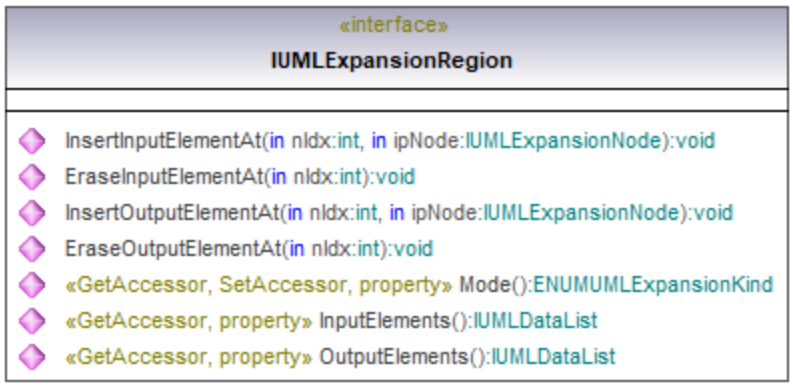
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExpansionRegion 1171			

© 2016-2022 Altova GmbH

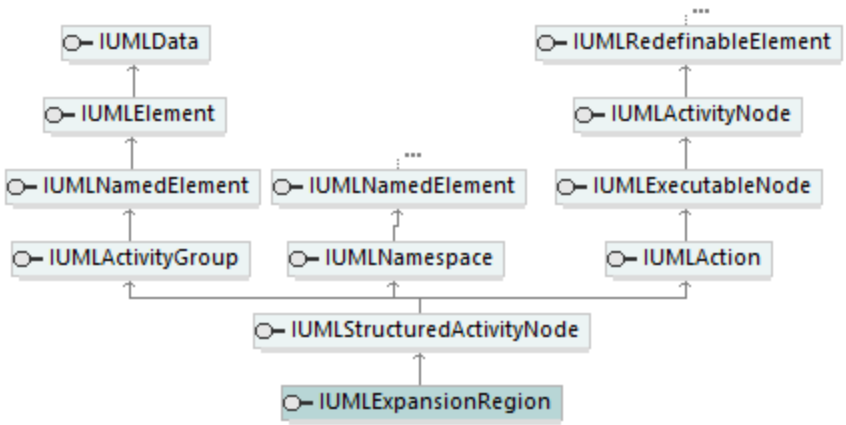
17.5.3.5.69 UModelAPI - IUMLExpansionRegion

Interface **IUMLExpansionRegion**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019

Interface **IUMLExpansionNode** 1170

Operation **RegionAsInput** 1073

RegionAsOutput 1073

Operation **RegionAsInput** 1170

RegionAsOutput 1170

Operation **IUMLExpansionRegion::EraseInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::EraseOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLDataList** 1013
documentation	A list of elements of type IUMLExpansionNode 1170.		

Operation **IUMLExpansionRegion::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode** 1170			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode** 1170			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLExpansionKind** 1371			

Operation **IUMLExpansionRegion::OutputElements**

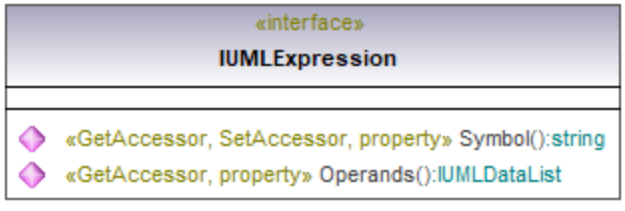
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLExpansionNode 1170.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

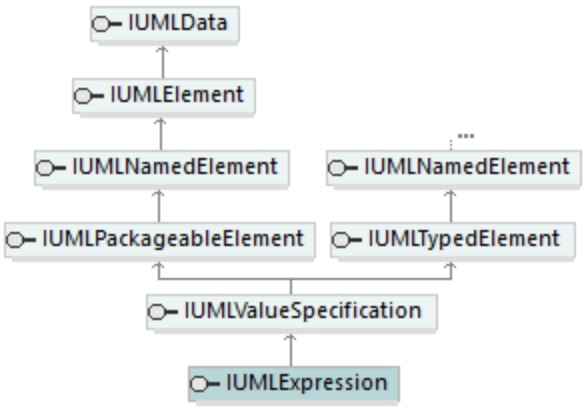
17.5.3.5.70 UModelAPI - IUMLExpression

Interface **IUMLExpression**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLValueSpecification** 1299 Operation **Expression** 1033 Operation **Expression** 1301

Operation **IUMLExpression::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLValueSpecification 1299.					

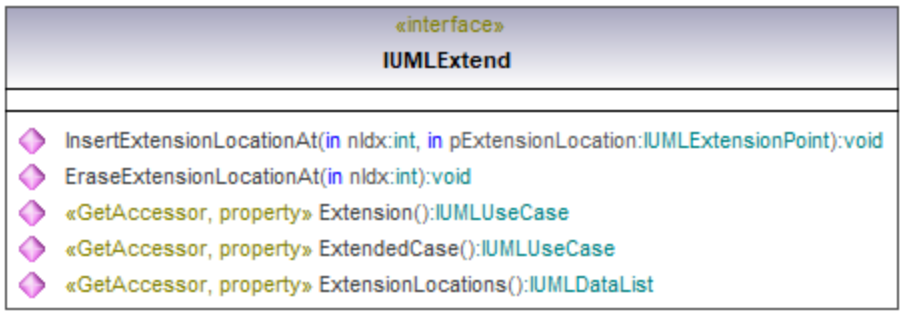
Operation **IUMLExpression::Symbol**

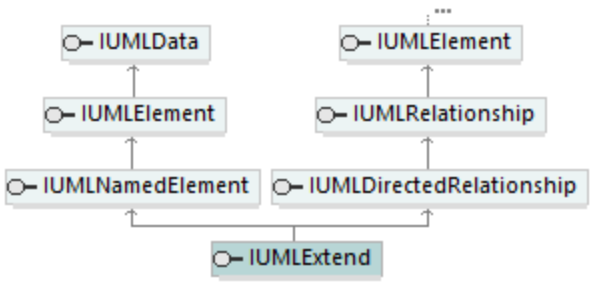
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.71 UModelAPI - IUMLExtend

Interface **IUMLExtend**

diagram	

hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLUseCase 1296 Operation InsertExtendAt 1043 Operation InsertExtendAt 1297

Operation **IUMLExtend::EraseExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLExtend::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

Operation **IUMLExtend::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

Operation **IUMLExtend::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLExtensionPoint 1175.					

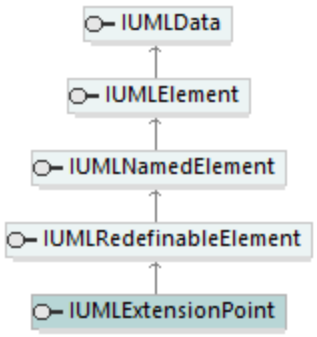
Operation **IUMLExtend::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pExtensionLocatiion		**IUMLExtensionPoint** 1175			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.3.5.72 UModelAPI - IUMLExtensionPoint

Interface **IUMLExtensionPoint**

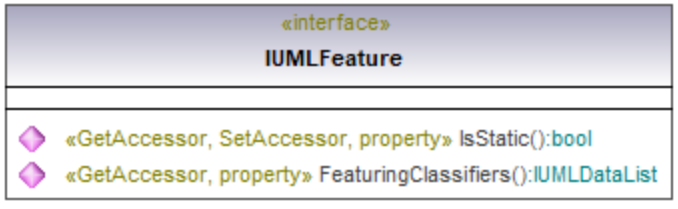
diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLExtend 1173 Interface IUMLUseCase 1296 Operation InsertExtensionLocationAt 1043 InsertExtensionPointAt 1043 Operation InsertExtensionLocationAt 1174 Operation InsertExtensionPointAt 1298

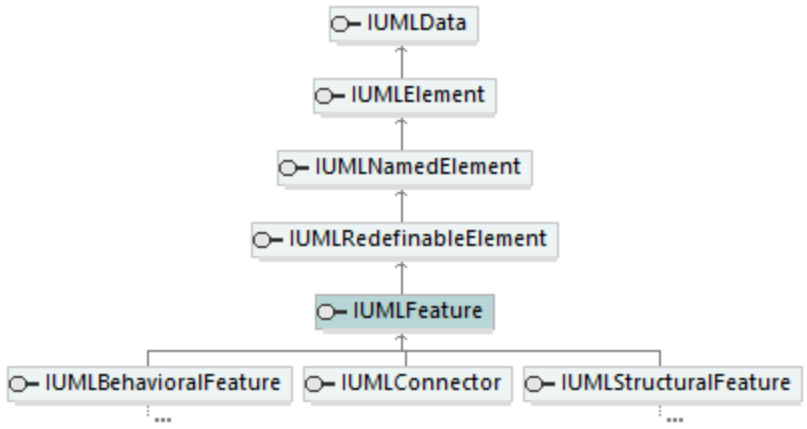
Operation **IUMLExtensionPoint::UseCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLUseCase 1296			

17.5.3.5.73 UModelAPI - IUMLFeature

Interface **IUMLFeature**

diagram	

hierarchy	

Operation **IUMLFeature::FeaturingClassifiers**

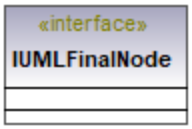
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLFeature::IsStatic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

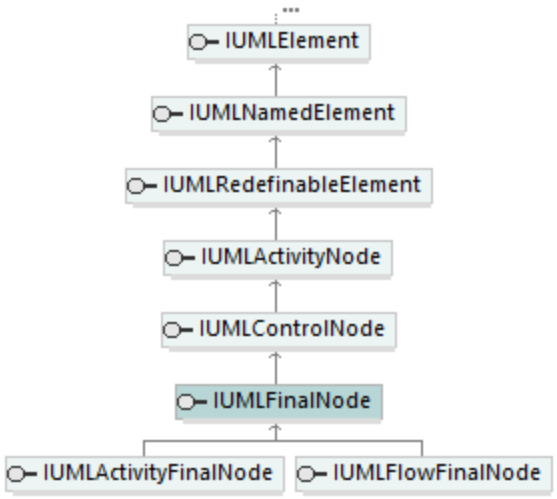
17.5.3.5.74 UModelAPI - IUMLFinalNode

Interface **IUMLFinalNode**

diagram	

 © 2016-2022 Altova GmbH

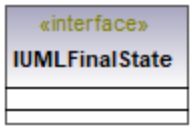
hierarchy



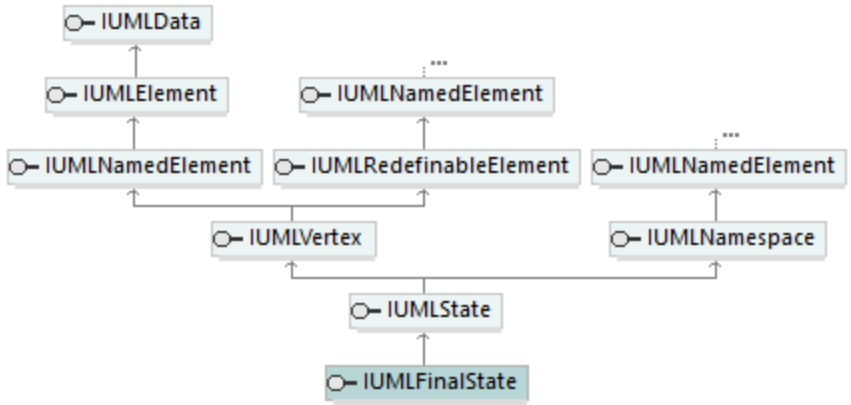
17.5.3.5.75 UModelAPI - IUMLFinalState

Interface **IUMLFinalState**

diagram



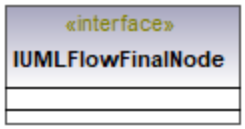
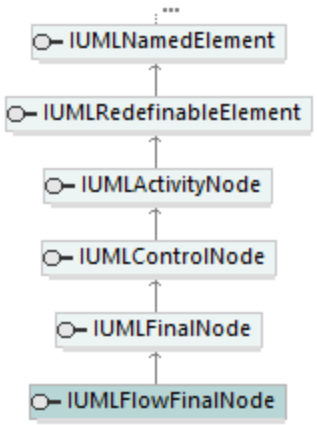
hierarchy



UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.5.76 UModelAPI - IUMLFlowFinalNode

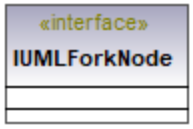
Interface **IUMLFlowFinalNode**

diagram	
hierarchy	

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

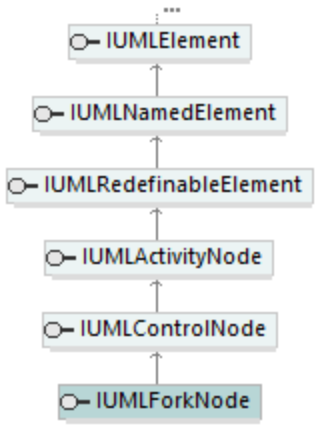
17.5.3.5.77 UModelAPI - IUMLForkNode

Interface **IUMLForkNode**

diagram	

 © 2016-2022 Altova GmbH

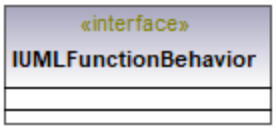
hierarchy



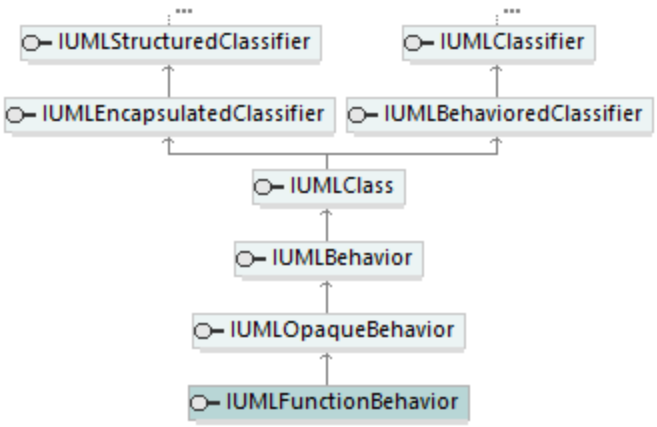
17.5.3.5.78 UModelAPI - IUMLFunctionBehavior

Interface **IUMLFunctionBehavior**

diagram



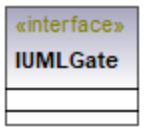
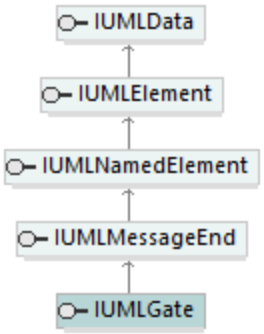
hierarchy



© 2016-2022 Altova GmbH

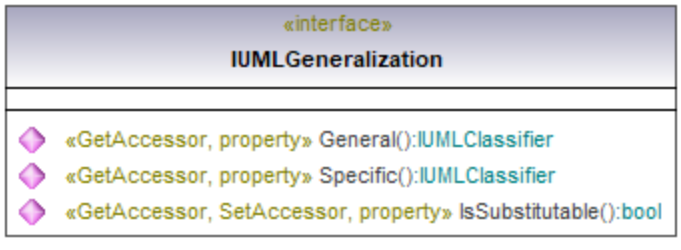
17.5.3.5.79 UModelAPI - IUMLGate

Interface **IUMLGate**

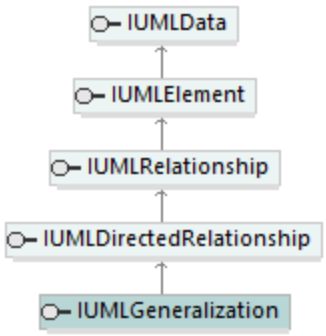
diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLInteraction 1194 Interface IUMLInteractionUse 1198 Operation InsertActualGateAt 1040 InsertFormalGateAt 1043 Operation InsertFormalGateAt 1194 Operation InsertActualGateAt 1199

17.5.3.5.80 UModelAPI - IUMLGeneralization

Interface **IUMLGeneralization**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLClassifier** 1124 Interface **IUMLDataAll** 1019 Operation **InsertGeneralizationAt** 1126 Operation **InsertGeneralizationAt** 1043

Operation **IUMLGeneralization::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLGeneralization::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

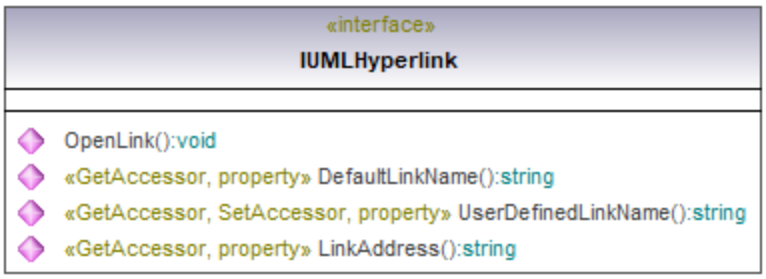
Operation **IUMLGeneralization::Specific**

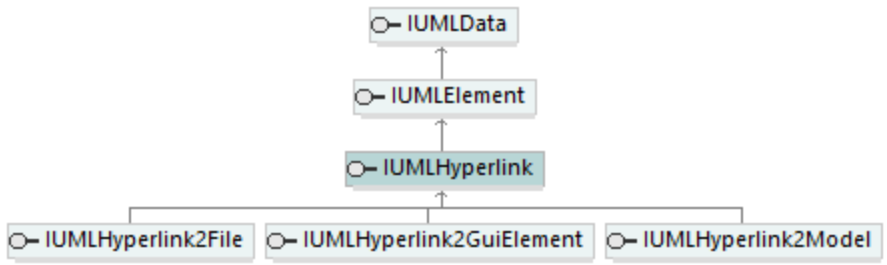
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.81 UModelAPI - IUMLHyperlink

Interface **IUMLHyperlink**

diagram	

hierarchy	

Operation **IUMLHyperlink::DefaultLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLHyperlink::UserDefinedLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

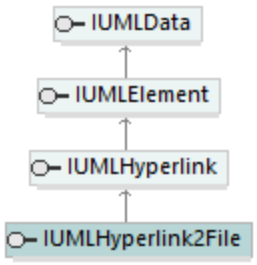
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.82 UModelAPI - IUMLHyperlink2File

Interface **IUMLHyperlink2File**

diagram	

 © 2016-2022 Altova GmbH

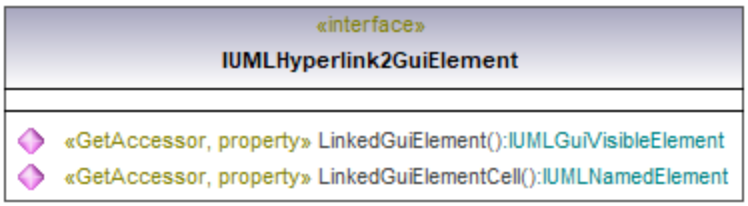
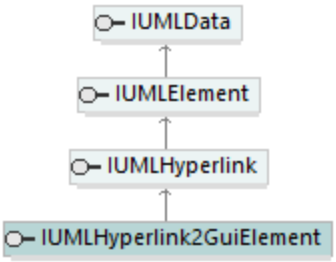
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLNamedElement 1222 Operation InsertOwnedHyperlink2FileAt 1048 Operation InsertOwnedHyperlink2FileAt 1223

Operation **IUMLHyperlink2File::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

17.5.3.5.83 UModelAPI - IUMLHyperlink2GuiElement

Interface **IUMLHyperlink2GuiElement**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLNamedElement 1222 Operation InsertOwnedHyperlink2GuiElement At 1048 Operation InsertOwnedHyperlink2GuiElement At 1223

Operation **IUMLHyperlink2GuiElement::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiVisibleElement 1365			

Operation **IUMLHyperlink2GuiElement::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamedElement 1222			

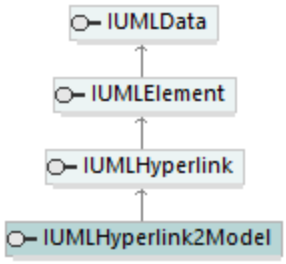
17.5.3.5.84 UModelAPI - IUMLHyperlink2Model

Interface **IUMLHyperlink2Model**

diagram



hierarchy



typedElements

Interface IUMLDataAll 1019
Interface IUMLNamedElement 1222
Operation InsertOwnedHyperlink2ModelAt 1048
Operation InsertOwnedHyperlink2ModelAt 1223

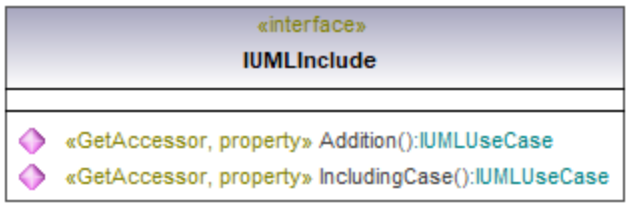
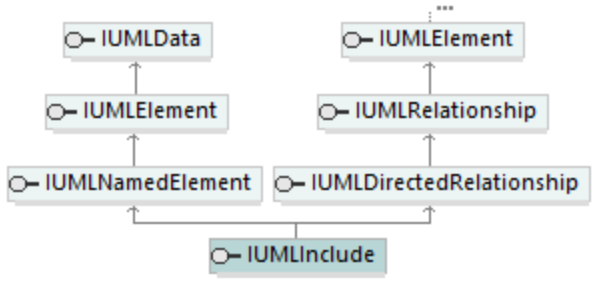
Operation **IUMLHyperlink2Model::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLData 1011			

 © 2016-2022 Altova GmbH

17.5.3.5.85 UModelAPI - IUMLInclude

Interface **IUMLInclude**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLUseCase** 1296 Operation **InsertIncludeAt** 1044 Operation **InsertIncludeAt** 1298

Operation **IUMLInclude::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase** 1296			

Operation **IUMLInclude::IncludingCase**

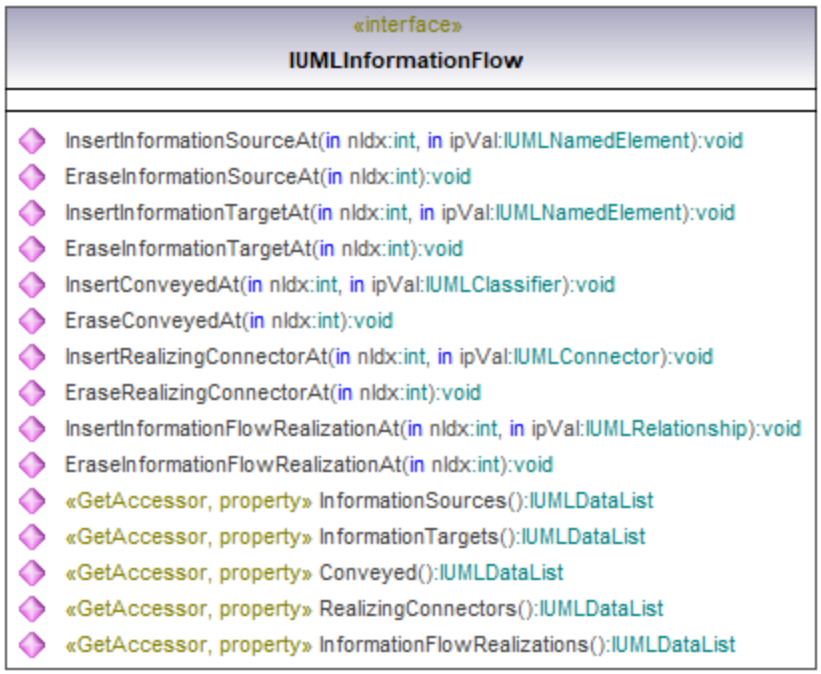
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase** 1296			

© 2016-2022 Altova GmbH

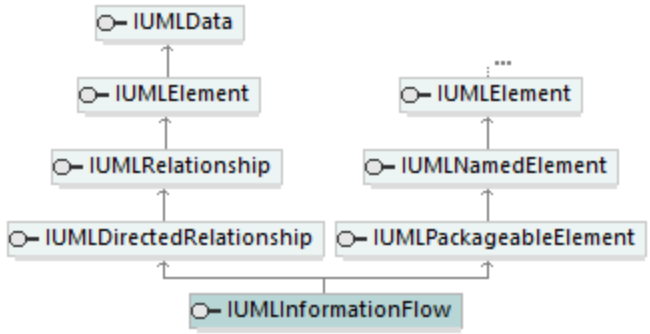
17.5.3.5.86 UModelAPI - IUMLInformationFlow

Interface **IUMLInformationFlow**

diagram



hierarchy



Operation **IUMLInformationFlow::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLInformationFlow::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationFlowRealizationAt**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLInformationFlow::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLInformationFlow::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLInformationFlow::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	IUMLClassifier 1124			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	IUMLRelationship 1262			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default

	nIdx	**in**	**int**
	ipVal	**in**	**IUMLNamedElement** 1222
	return	**return**	**void**

Operation **IUMLInformationFlow::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement** 1222			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnector** 1140			
	return	**return**	**void**			

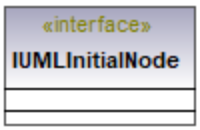
Operation **IUMLInformationFlow::RealizingConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

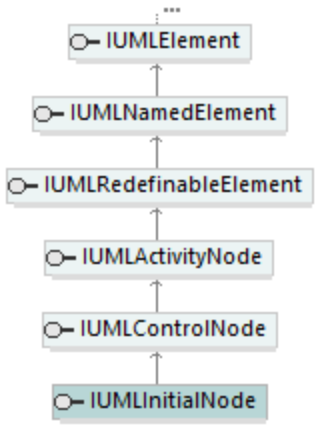
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.87 UModelAPI - IUMLInitialNode

Interface **IUMLInitialNode**

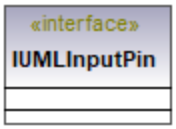
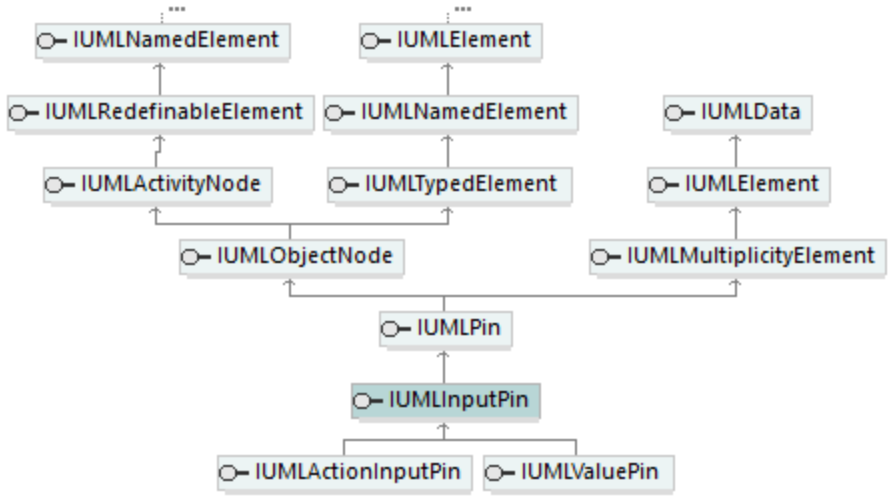
diagram	
	

 © 2016-2022 Altova GmbH

hierarchy	

17.5.3.5.88 UModelAPI - IUMLInputPin

Interface **IUMLInputPin**

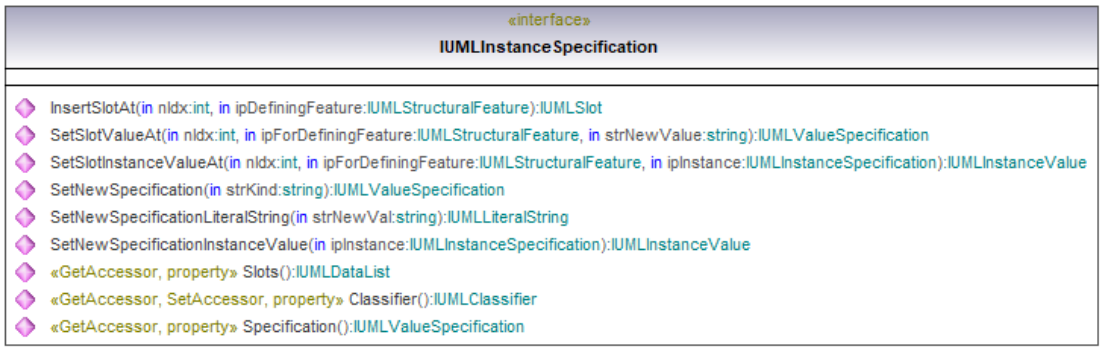
diagram	
hierarchy	
typedElements	Interface **IUMLCallOperationAction** 1118 Operation **CallTarget** 1119, **SetNewCallTarget** 1119

	Interface IUMLDataAll 1019	Operation CallTarget 1024 InsertArgumentAt 1040 InsertArgumentOfKindAt 1040 InsertInputValueAt 1044 SetNewCallTarget 1076 SetNewSignalTarget 1079 SignalTarget 1082
	Interface IUMLInvocationAction 1207	Operation InsertArgumentAt 1207 InsertArgumentOfKindAt 1207
	Interface IUMLOpaqueAction 1233	Operation InsertInputValueAt 1233
	Interface IUMLSendSignalAction 1263	Operation SetNewSignalTarget 1264 SignalTarget 1264

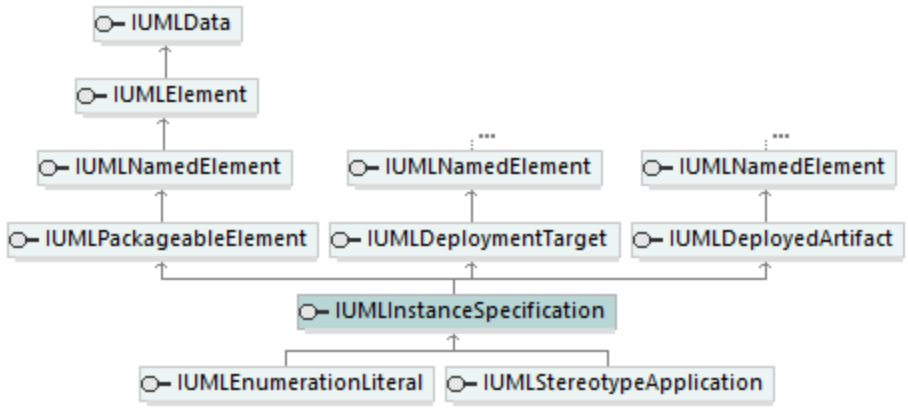
17.5.3.5.89 UModelAPI - IUMLInstanceSpecification

Interface **IUMLInstanceSpecification**

diagram



hierarchy



typedElements	Interface IUMLConstraint 1142	Operation SetNewSpecificationInstanceValue 1144

 © 2016-2022 Altova GmbH

	Interface **IUMLDataAll** 1019	Operation **InsertSlotInstanceValueAt** 1052 **Instance** 1053 **OwningInstance** 1069 **OwningInstanceSpec** 1069 **SetNewDefaultValueInstanceValue** 1076 **SetNewSpecificationInstanceValue** 1079 **SetSlotInstanceValueAt** 1080
	Interface **IUMLInstanceSpecification** 1190	Operation **SetNewSpecificationInstanceValue** 1191 **SetSlotInstanceValueAt** 1191
	Interface **IUMLInstanceValue** 1192	Operation **Instance** 1193
	Interface **IUMLParameter** 1244	Operation **SetNewDefaultValueInstanceValue** 1245
	Interface **IUMLProperty** 1251	Operation **SetNewDefaultValueInstanceValue** 1254
	Interface **IUMLSlot** 1266	Operation **InsertSlotInstanceValueAt** 1267 **OwningInstance** 1267
	Interface **IUMLValueSpecification** 1299	Operation **OwningInstanceSpec** 1301

Operation **IUMLInstanceSpecification::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLInstanceSpecification::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	return	**return**	**IUMLSlot** 1266			

Operation **IUMLInstanceSpecification::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLInstanceSpecification::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLInstanceSpecification::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString** 1213			

Operation **IUMLInstanceSpecification::SetSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLInstanceSpecification::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLInstanceSpecification::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLSlot 1266.					

Operation **IUMLInstanceSpecification::Specification**

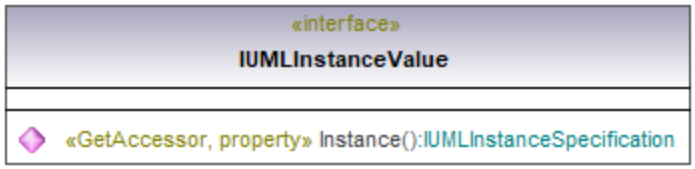
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

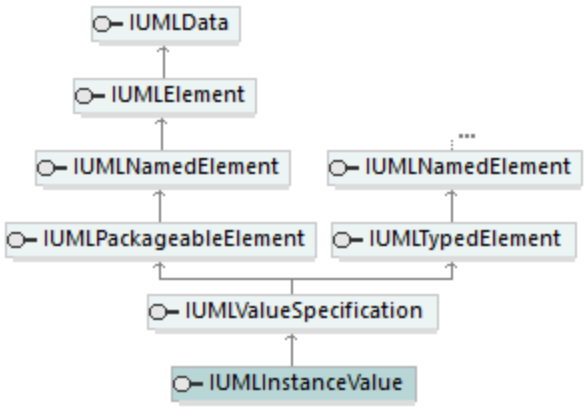
17.5.3.5.90 UModelAPI - IUMLInstanceValue

Interface **IUMLInstanceValue**

diagram	



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLConstraint** 1142 Interface **IUMLDataAll** 1019 Interface **IUMLInstanceSpecification** 1190 Interface **IUMLParameter** 1244 Interface **IUMLProperty** 1251 Interface **IUMLSlot** 1266 Operation **SetNewSpecificationInstanceValue** 1144 Operation **InsertSlotInstanceValueAt** 1052 **SetNewDefaultValueInstanceValue** 1076 **SetNewSpecificationInstanceValue** 1079 **SetSlotInstanceValueAt** 1080 Operation **SetNewSpecificationInstanceValue** 1191 **SetSlotInstanceValueAt** 1191 Operation **SetNewDefaultValueInstanceValue** 1245 Operation **SetNewDefaultValueInstanceValue** 1254 Operation **InsertSlotInstanceValueAt** 1267

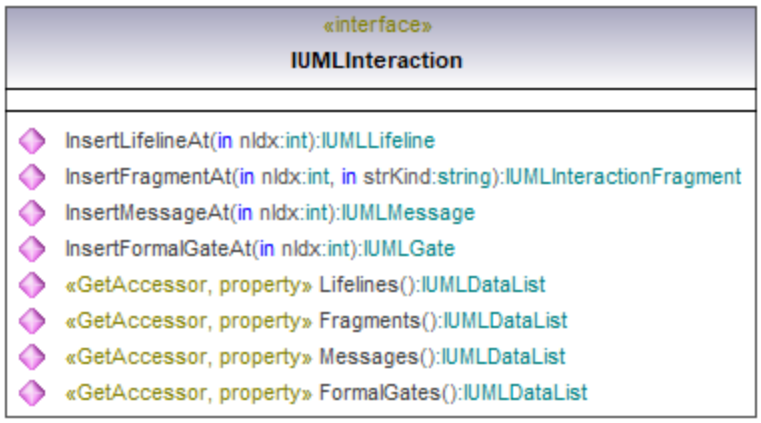
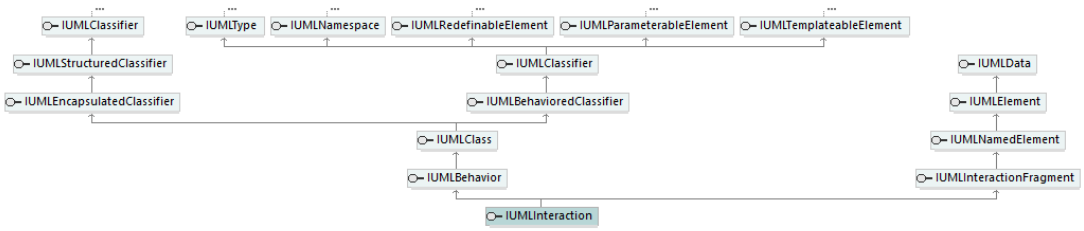
Operation **IUMLInstanceValue::Instance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification** 1190			

© 2016-2022 Altova GmbH

17.5.3.5.91 UModelAPI - IUMLInteraction

Interface **IUMLInteraction**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInteractionUse** 1198 Operation **RefersTo** 1073 Operation **RefersTo** 1199

Operation **IUMLInteraction::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLGate 1180.					

Operation **IUMLInteraction::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLInteractionFragment 1197.					

Operation **IUMLInteraction::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate** 1180			

Operation **IUMLInteraction::InsertFragmentAt**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInteractionFragment** 1197			

Operation **IUMLInteraction::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLLifeline** 1209			

Operation **IUMLInteraction::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLMessage** 1216			

Operation **IUMLInteraction::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLLifeline 1209.					

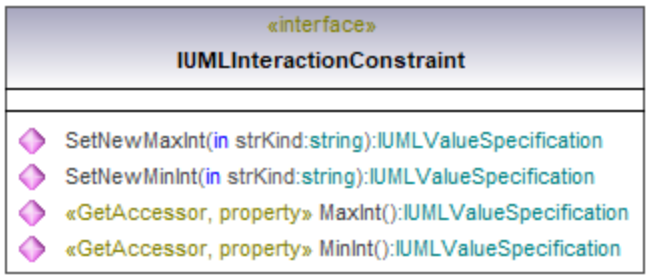
Operation **IUMLInteraction::Messages**

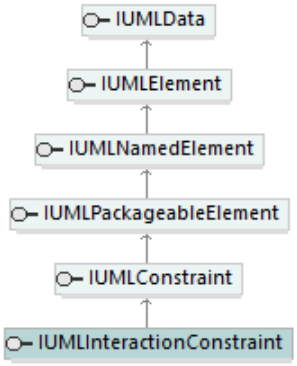
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLMessage 1216.					

17.5.3.5.92 UModelAPI - IUMLInteractionConstraint

Interface **IUMLInteractionConstraint**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInteractionOperand** 1197 Operation **OperandGuard** 1065 **SetNewOperandGuard** 1078 Operation **OperandGuard** 1198 **SetNewOperandGuard** 1198

Operation **IUMLInteractionConstraint::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLInteractionConstraint::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLInteractionConstraint::SetNewMaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

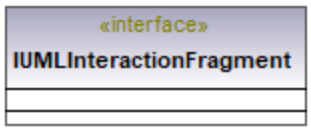
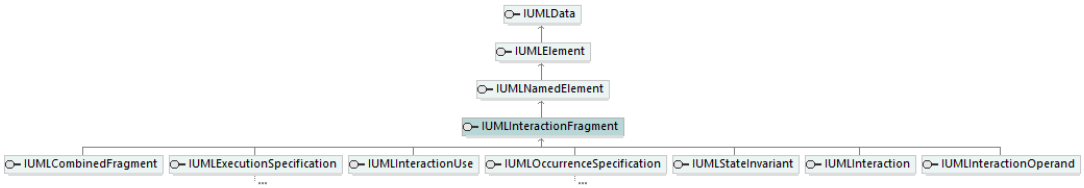
Operation **IUMLInteractionConstraint::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

© 2016-2022 Altova GmbH

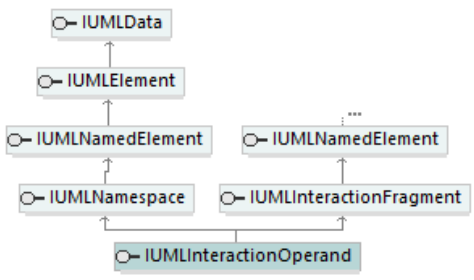
17.5.3.5.93 UModelAPI - IUMLInteractionFragment

Interface **IUMLInteractionFragment**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInteraction** 1194 Interface **IUMLLifeline** 1209 Operation **InsertCoveredByAt** 1041 **InsertFragmentAt** 1043 Operation **InsertFragmentAt** 1194 Operation **InsertCoveredByAt** 1209

17.5.3.5.94 UModelAPI - IUMLInteractionOperand

Interface **IUMLInteractionOperand**

diagram	
hierarchy	
typedElements	Interface **IUMLCombinedFragment** 1131 Operation **InsertOperandAt** 1131

	Interface IUMLDataAll[1019]	Operation InsertOperandAt[1046]

Operation **IUMLInteractionOperand::OperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInteractionConstraint[1195]			

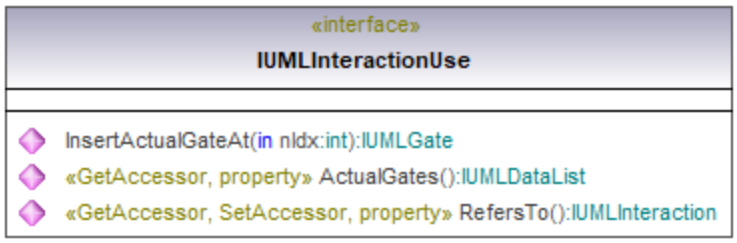
Operation **IUMLInteractionOperand::SetNewOperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInteractionConstraint[1195]			

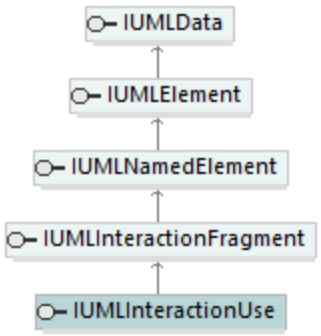
17.5.3.5.95 UModelAPI - IUMLInteractionUse

Interface **IUMLInteractionUse**

diagram



hierarchy



Operation **IUMLInteractionUse::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList[1013]			
documentation	A list of elements of type IUMLGate[1180].					

 © 2016-2022 Altova GmbH

Operation **IUMLInteractionUse::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLGate 1180			

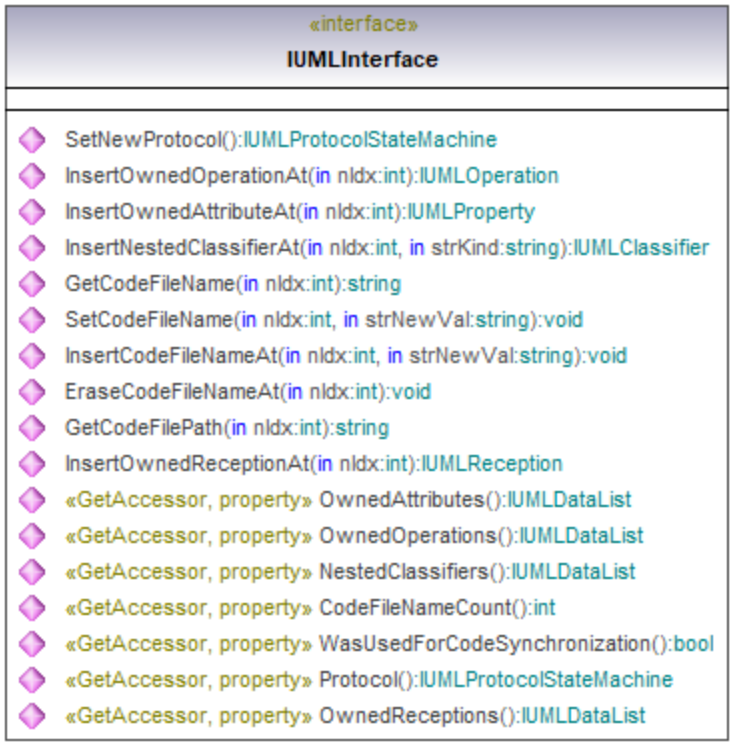
Operation **IUMLInteractionUse::RefersTo**

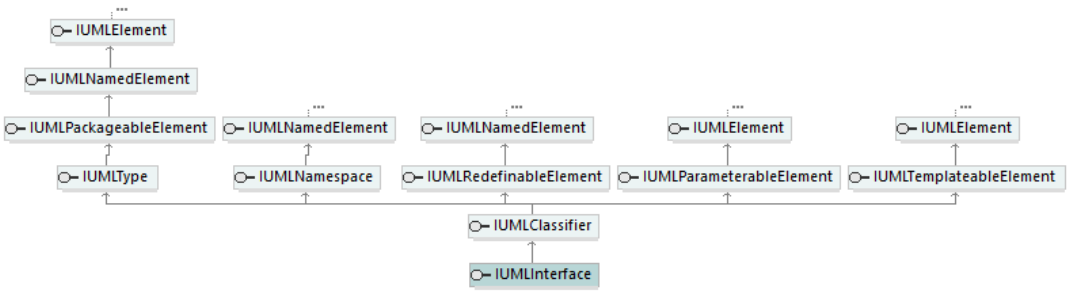
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInteraction 1194			

17.5.3.5.96 UModelAPI - IUMLInterface

Interface **IUMLInterface**

diagram



hierarchy	
typedElements	Interface IUMLBehavioredClassifier 1114 Interface IUMLClassifier 1124 Interface IUMLDataAll 1019 Interface IUMLInterfaceRealization 1202 Interface IUMLOperation 1236 Interface IUMLProperty 1251 Interface IUMLProtocolStateMachine 1255 Interface IUMLReception 1259 Operation InsertInterfaceRealizationAt 1114 Operation NestingInterface 1126 Operation Contract 1027 Operation InsertInterfaceRealizationAt 1045 Operation Interface 1053 NestingInterface 1064 Operation Contract 1203 Operation Interface 1237 Operation Interface 1253 Operation Interface 1255 Operation Interface 1259

Operation **IUMLInterface::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLInterface::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInterface::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			

Operation **IUMLInterface::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**string**			
documentation	get the full code file path					

Operation **IUMLInterface::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			

© 2016-2022 Altova GmbH

	return	**return**	**void**

Operation **IUMLInterface::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLClassifier** 1124			

Operation **IUMLInterface::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLInterface::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLInterface::InsertOwnedReceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLReception** 1259			

Operation **IUMLInterface::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

Operation **IUMLInterface::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

Operation **IUMLInterface::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLOperation 1236.					

Operation **IUMLInterface::OwnedReceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLInterface::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine** 1255			

Operation **IUMLInterface::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLInterface::SetNewProtocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine** 1255			

Operation **IUMLInterface::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

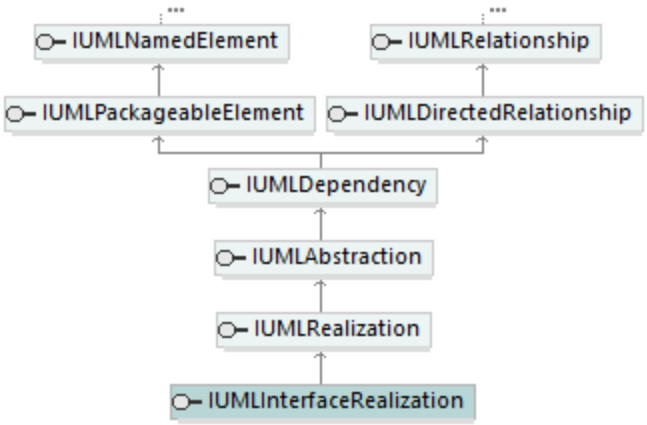
17.5.3.5.97 UModelAPI - IUMLInterfaceRealization

Interface **IUMLInterfaceRealization**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typed Elements	Interface IUMLBehavioredClassifier 1114 Interface IUMLDataAll 1019 Operation InsertInterfaceRealizationAt 1114 Operation InsertInterfaceRealizationAt 1045

Operation **IUMLInterfaceRealization::Contract**

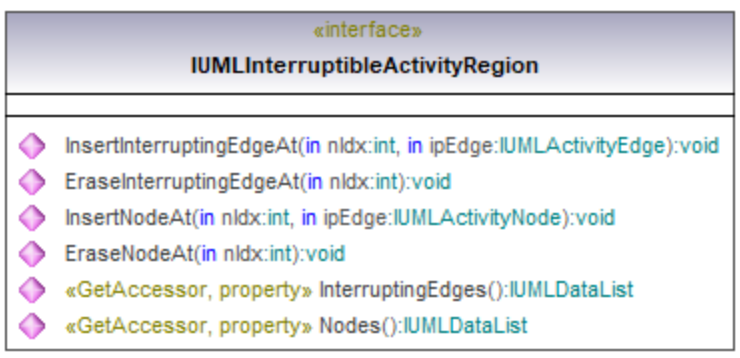
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInterface 1199			

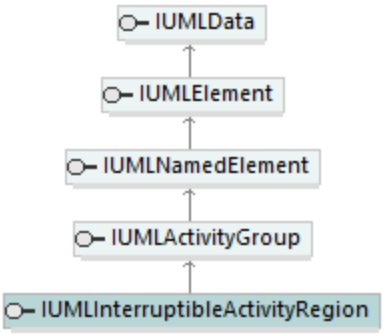
Operation **IUMLInterfaceRealization::ImplementingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavioredClassifier 1114			

17.5.3.5.98 UModelAPI - IUMLInterruptibleActivityRegion

Interface **IUMLInterruptibleActivityRegion**

diagram	

hierarchy	

Operation **IUMLInterruptibleActivityRegion::EraseInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge** 1097			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityNode** 1101			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLActivityEdge 1097.					

Operation **IUMLInterruptibleActivityRegion::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

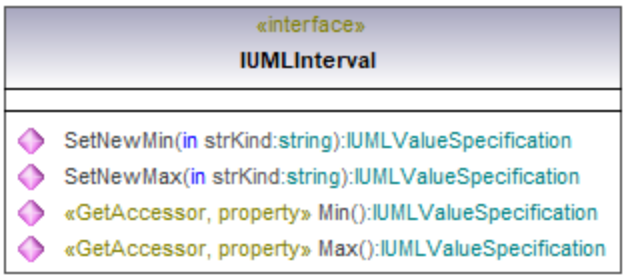
© 2016-2022 Altova GmbH

documentation	A list of elements of type IUMLActivityNode[1101].

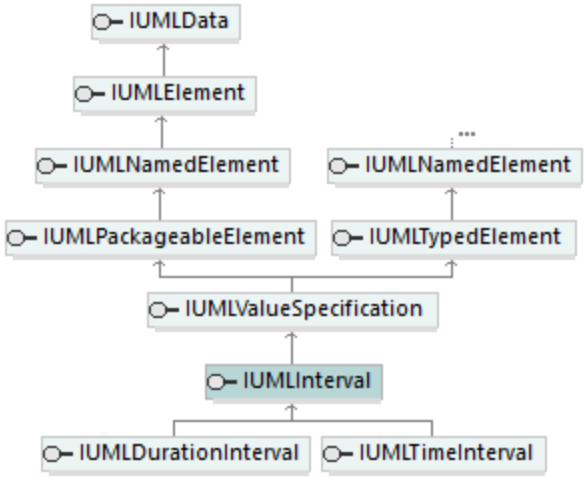
17.5.3.5.99 UModelAPI - IUMLInterval

Interface **IUMLInterval**

diagram



hierarchy



Operation **IUMLInterval::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification[1299]			

Operation **IUMLInterval::Min**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification[1299]			

Operation **IUMLInterval::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

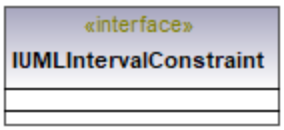
Operation **IUMLInterval::SetNewMin**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

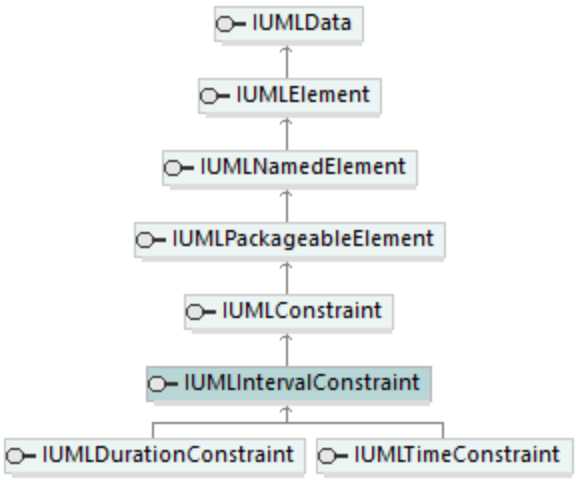
17.5.3.5.100 UModelAPI - IUMLIntervalConstraint

Interface **IUMLIntervalConstraint**

diagram



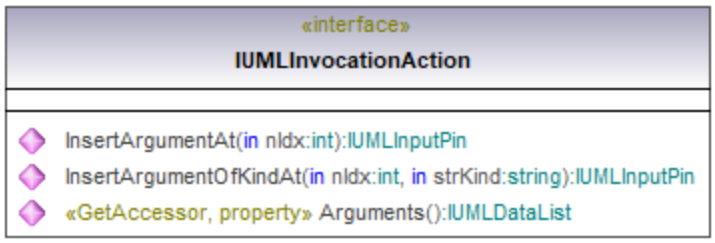
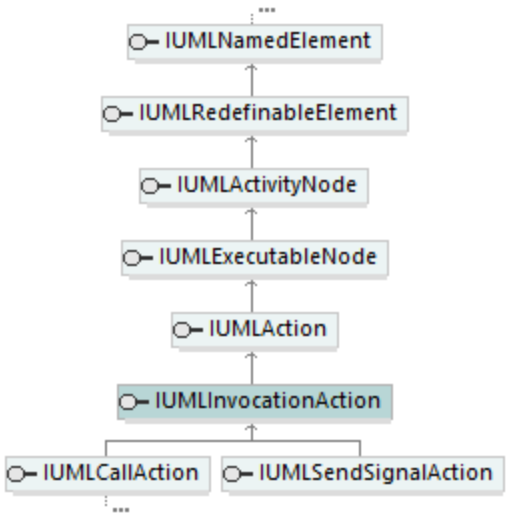
hierarchy



 © 2016-2022 Altova GmbH

17.5.3.5.101 UModelAPI - IUMLInvocationAction

Interface **IUMLInvocationAction**

diagram	
hierarchy	

Operation **IUMLInvocationAction::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLInputPin 1189.					

Operation **IUMLInvocationAction::InsertArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin** 1189			

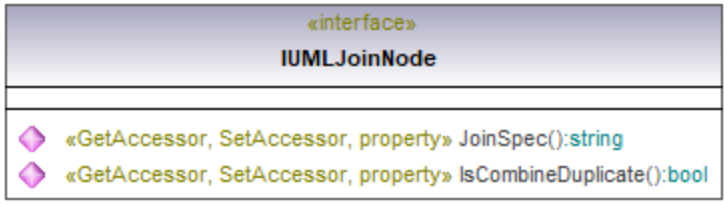
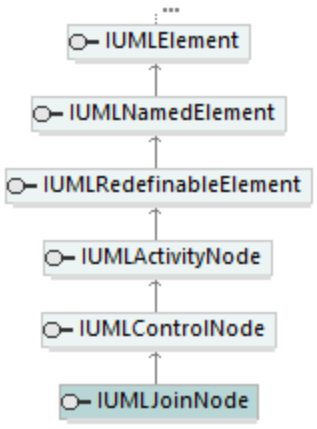
Operation **IUMLInvocationAction::InsertArgumentOfKindAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin** 1189			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.5.102 UModelAPI - IUMLJoinNode

Interface **IUMLJoinNode**

diagram	
hierarchy	

Operation **IUMLJoinNode::IsCombineDuplicate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLJoinNode::JoinSpec**

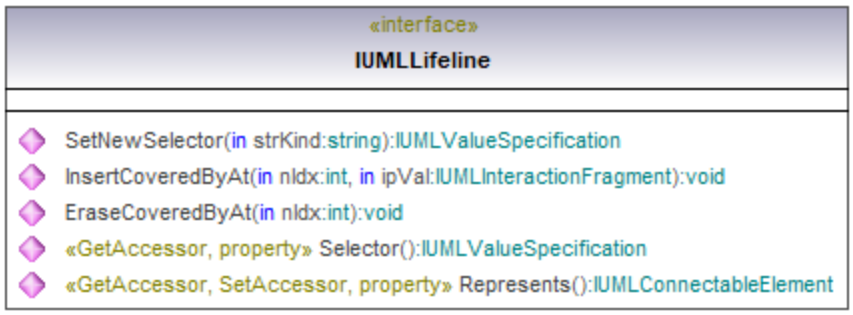
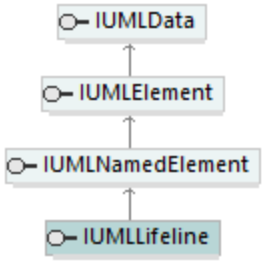
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

 © 2016-2022 Altova GmbH

17.5.3.5.103 UModelAPI - IUMLLifeline

Interface **IUMLLifeline**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLInteraction 1194 Interface IUMLMessage 1216 Interface IUMLOccurrenceSpecification 1231 Interface IUMLStateInvariant 1271 Operation Covered 1027 GetSourceLifeline 1036 GetTargetLifeline 1036 InsertLifelineAt 1045 Operation InsertLifelineAt 1195 Operation GetSourceLifeline 1216 GetTargetLifeline 1216 Operation Covered 1232 Operation Covered 1271

Operation **IUMLLifeline::EraseCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLLifeline::InsertCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLInteractionFragment** 1197			
	return	**return**	**void**			

Operation **IUMLLifeline::Represents**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement** 1137			

Operation **IUMLLifeline::Selector**

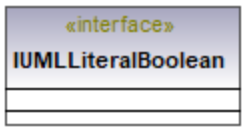
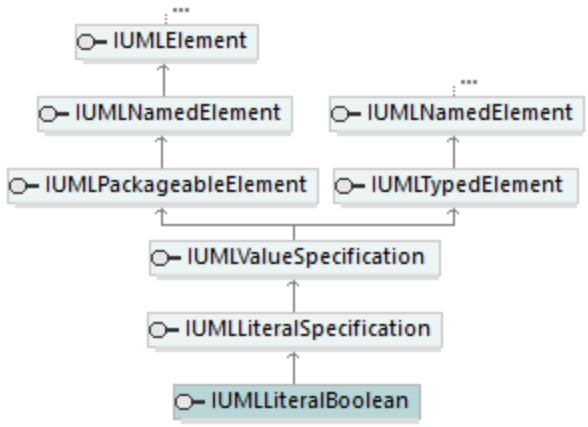
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLLifeline::SetNewSelector**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

17.5.3.5.104 UModelAPI - IUMLLiteralBoolean

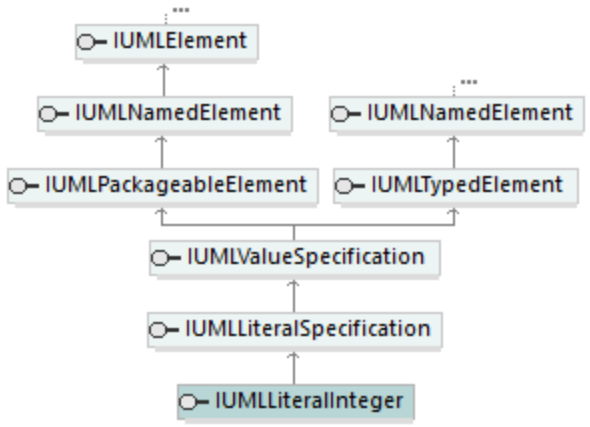
Interface **IUMLLiteralBoolean**

diagram	
hierarchy	

 © 2016-2022 Altova GmbH

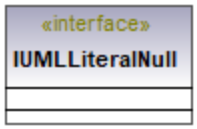
17.5.3.5.105 UModelAPI - IUMLLiteralInteger

Interface **IUMLLiteralInteger**

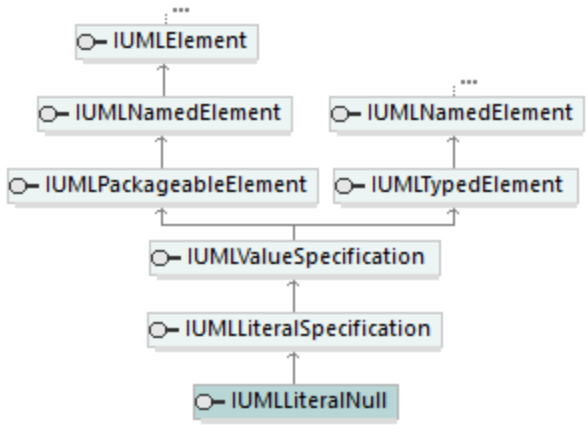
diagram	
hierarchy	

17.5.3.5.106 UModelAPI - IUMLLiteralNull

Interface **IUMLLiteralNull**

diagram	

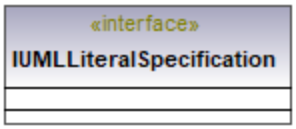
hierarchy



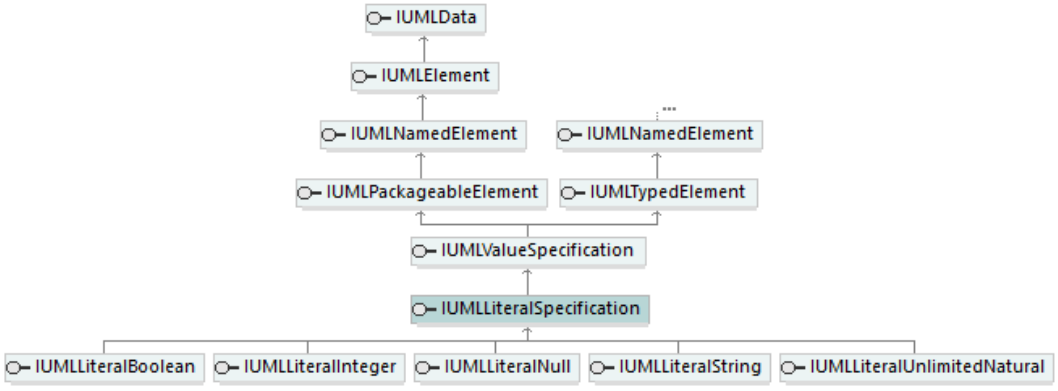
17.5.3.5.107 UModelAPI - IUMLLiteralSpecification

Interface **IUMLLiteralSpecification**

diagram



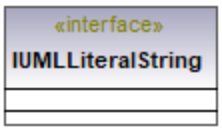
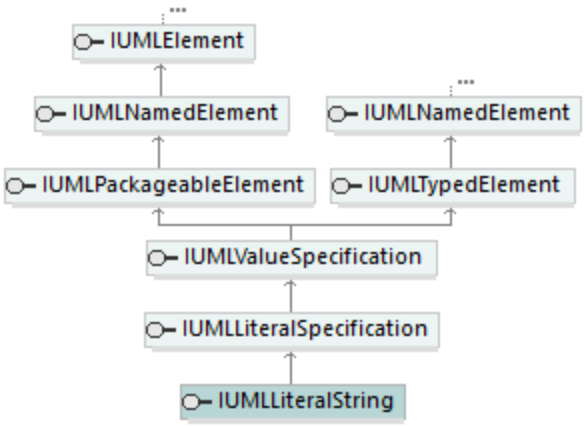
hierarchy



 © 2016-2022 Altova GmbH

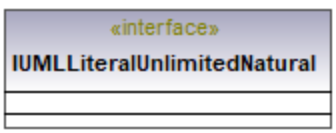
17.5.3.5.108 UModelAPI - IUMLLiteralString

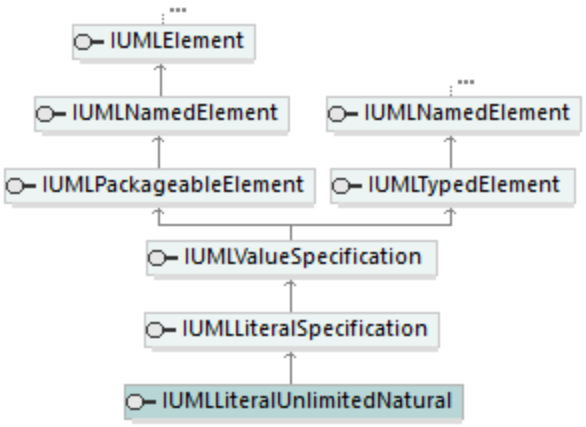
Interface **IUMLLiteralString**

diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** 1142 Interface **IUMLDataAll** 1019 Interface **IUMLInstanceSpecification** 1190 Interface **IUMLParameter** 1244 Interface **IUMLProperty** 1251 Operation **SetNewSpecificationLiteralString** 1144 Operation **SetNewDefaultValueLiteralString** 1076 **SetNewSpecificationLiteralString** 1079 Operation **SetNewSpecificationLiteralString** 1191 Operation **SetNewDefaultValueLiteralString** 1245 Operation **SetNewDefaultValueLiteralString** 1254

17.5.3.5.109 UModelAPI - IUMLLiteralUnlimitedNatural

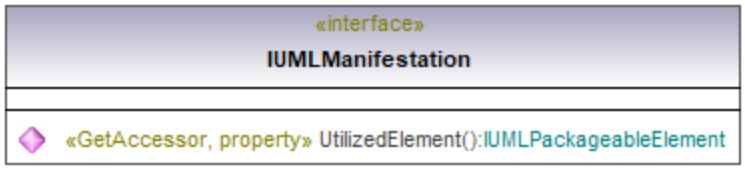
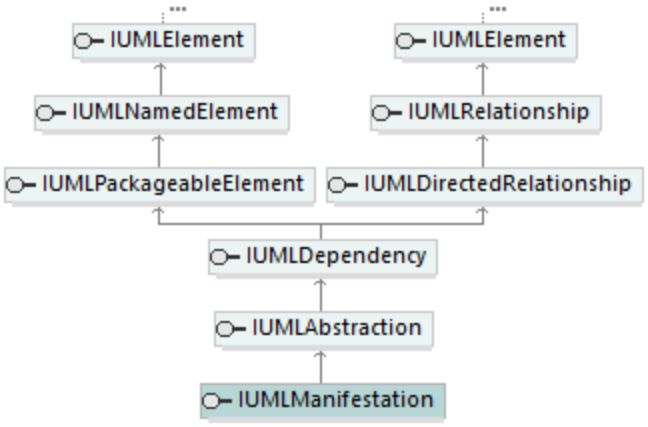
Interface **IUMLLiteralUnlimitedNatural**

diagram	

hierarchy	

17.5.3.5.110 UModelAPI - IUMLManifestation

Interface **IUMLManifestation**

diagram	
hierarchy	
typedElements	Interface **IUMLArtifact** 1108 Interface **IUMLDataAll** 1019 Operation **InsertManifestationAt** 1107 Operation **InsertManifestationAt** 1045

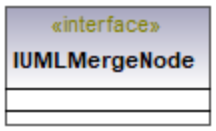
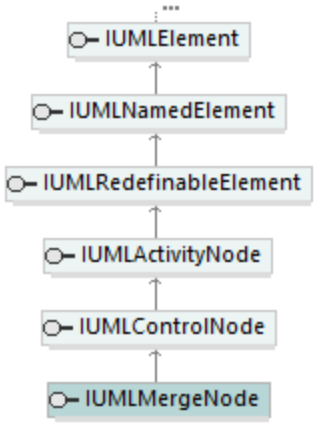
Operation **IUMLManifestation::UtilizedElement**

 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement** 1241			

17.5.3.5.111 UModelAPI - IUMLMergeNode

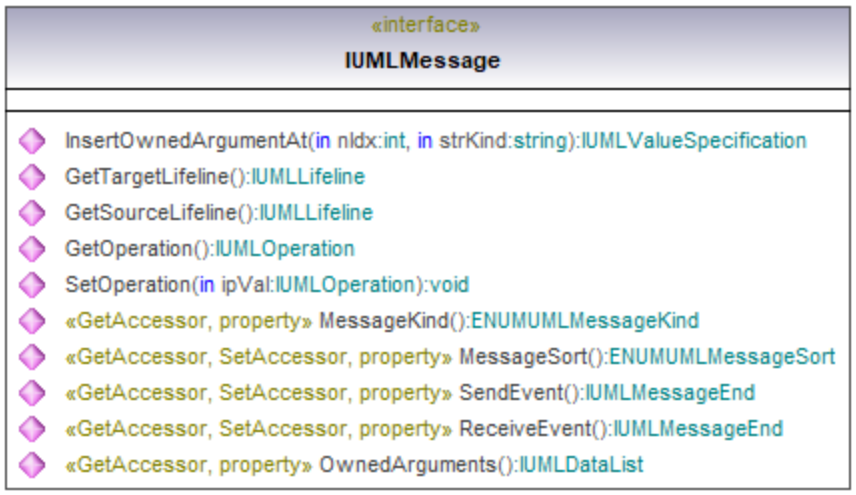
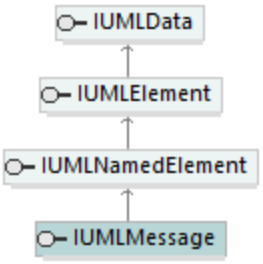
Interface **IUMLMergeNode**

diagram	
hierarchy	

© 2016-2022 Altova GmbH

17.5.3.5.112 UModelAPI - IUMLMessage

Interface **IUMLMessage**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInteraction** 1194 Interface **IUMLMessageEnd** 1218 Operation **InsertMessageAt** 1046 **Message** 1062 Operation **InsertMessageAt** 1195 Operation **Message** 1218

Operation **IUMLMessage::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLMessage::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline** 1209			

Operation **IUMLMessage::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline** 1209			

© 2016-2022 Altova GmbH

Operation **IUMLMessage::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLMessage::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageKind** 1374			

Operation **IUMLMessage::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageSort** 1375			

Operation **IUMLMessage::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLValueSpecification 1299.					

Operation **IUMLMessage::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd** 1218			

Operation **IUMLMessage::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd** 1218			

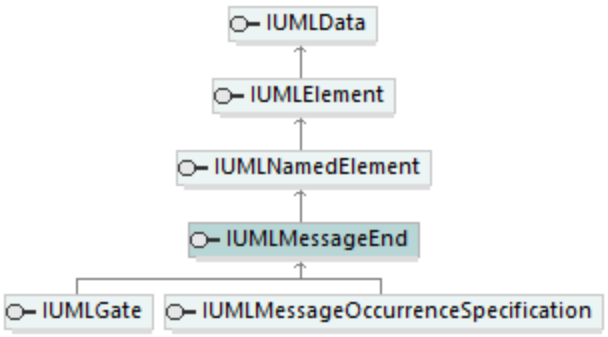
Operation **IUMLMessage::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLOperation** 1236			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.3.5.113 UModelAPI - IUMLMessageEnd

Interface **IUMLMessageEnd**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLMessage 1216 Operation ReceiveEvent 1072 SendEvent 1074 Operation ReceiveEvent 1217 SendEvent 1217

Operation **IUMLMessageEnd::Message**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLMessage 1216			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

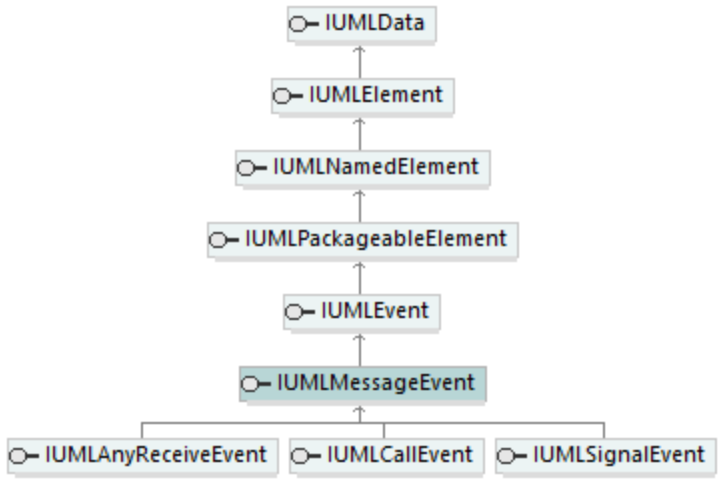
17.5.3.5.114 UModelAPI - IUMLMessageEvent

Interface **IUMLMessageEvent**

diagram	

© 2016-2022 Altova GmbH

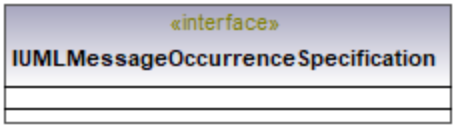
hierarchy



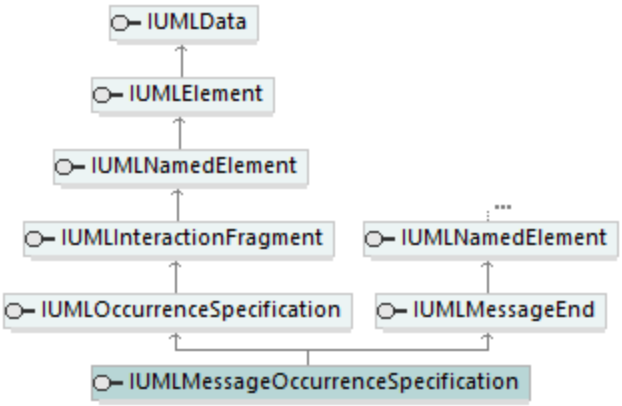
17.5.3.5.115 UModelAPI - IUMLMessageOccurrenceSpecification

Interface **IUMLMessageOccurrenceSpecification**

diagram

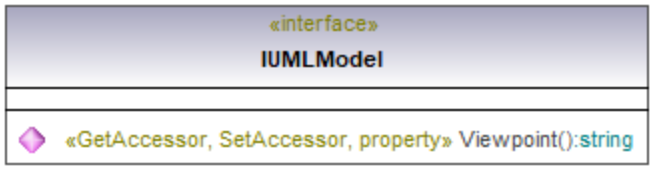
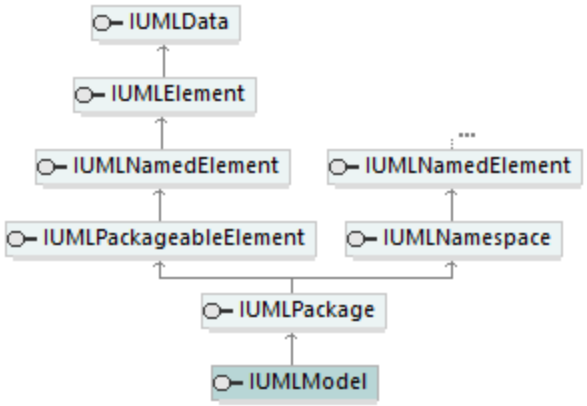


hierarchy



17.5.3.5.116 UModelAPI - IUMLModel

Interface **IUMLModel**

diagram	
hierarchy	

Operation **IUMLModel::Viewpoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

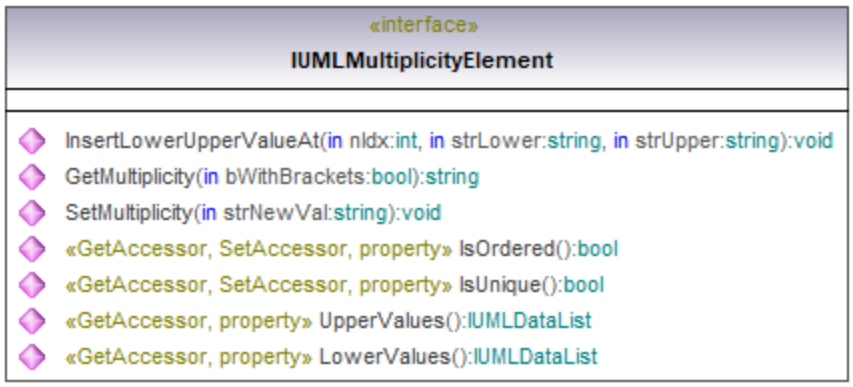
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

 © 2016-2022 Altova GmbH

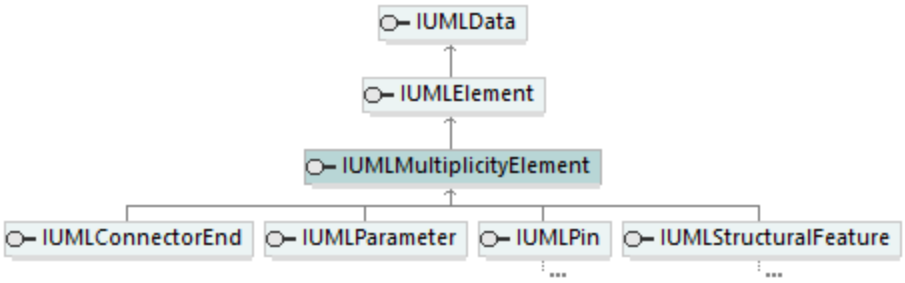
17.5.3.5.117 UModelAPI - IUMLMultiplicityElement

Interface **IUMLMultiplicityElement**

diagram



hierarchy



typedElements

Interface IUMLDataAll 1019

Interface IUMLValueSpecification 1299

Operation OwningLower 1069
OwningUpper 1070

Operation OwningLower 1301
OwningUpper 1302

Operation **IUMLMultiplicityElement::GetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	bWithBrackets	**in**	**bool**			
	return	**return**	**string**			

Operation **IUMLMultiplicityElement::InsertLowerUpperValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strLower	**in**	**string**			
	strUpper	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLMultiplicityElement::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLMultiplicityElement::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLMultiplicityElement::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLValueSpecification 1299.					

Operation **IUMLMultiplicityElement::SetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

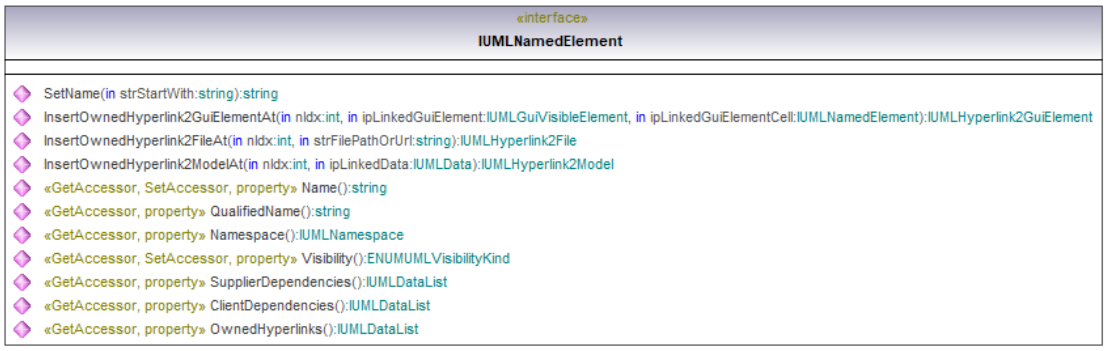
Operation **IUMLMultiplicityElement::UpperValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLValueSpecification 1299.					

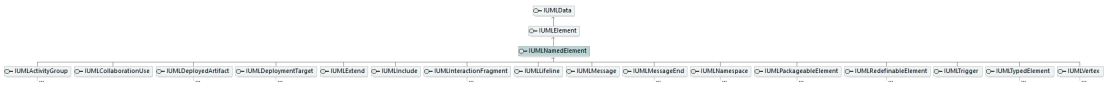
17.5.3.5.118 UModelAPI - IUMLNamedElement

Interface **IUMLNamedElement**

diagram



hierarchy



typedElements

Interface **IUMLCommentTextHyperlink** 1133

Operation **SetHyperlinkGuiElementAddress** 1134

 © 2016-2022 Altova GmbH

	Interface IUMLDataAll 1019	Operation FindOwnedMemberWithQualifiedName 1034 InsertInformationSourceAt 1044 InsertInformationTargetAt 1044 InsertOwnedHyperlink2GuiElementAt 1048 LinkedGuiElementCell 1060 SetHyperlinkGuiElementAddress 1075 TimeObservationEvent 1086
	Interface IUMLGuiTextHyperlink 1355	Operation SetHyperlinkGuiElementAddress 1356
	Interface IUMLHyperlink2GuiElement 1183	Operation LinkedGuiElementCell 1184
	Interface IUMLInformationFlow 1186	Operation InsertInformationSourceAt 1187 InsertInformationTargetAt 1188
	Interface IUMLNamedElement 1222	Operation InsertOwnedHyperlink2GuiElementAt 1223
	Interface IUMLNamespace 1225	Operation FindOwnedMemberWithQualifiedName 1225
	Interface IUMLTimeObservation 1290	Operation TimeObservationEvent 1291

Operation **IUMLNamedElement::ClientDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLDependency 1143.					

Operation **IUMLNamedElement::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFilePathOrUrl	**in**	**string**			
	return	**return**	IUMLHyperlink2File 1182			

Operation **IUMLNamedElement::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedGuiElement	**in**	IUMLGuiVisibleElement 1365			
	ipLinkedGuiElementCell	**in**	IUMLNamedElement 1222			
	return	**return**	IUMLHyperlink2GuiElement 1183			

Operation **IUMLNamedElement::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedData	**in**	IUMLData 1011			
	return	**return**	IUMLHyperlink2Model 1184			

Operation **IUMLNamedElement::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLNamedElement::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace** 1225			

Operation **IUMLNamedElement::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLHyperlink 1181.					

Operation **IUMLNamedElement::QualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLNamedElement::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			
documentation	This function w ill find and set a unique name (starting w ith 'strStartWith') that the element is distinguishable in its parent namespace.					

Operation **IUMLNamedElement::SupplierDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLDependency 1148.					

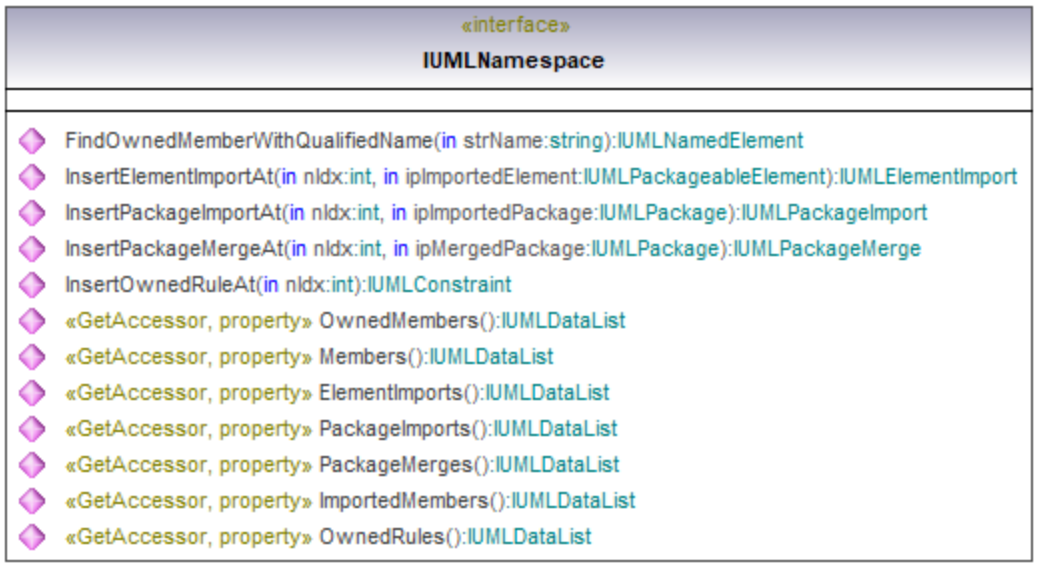
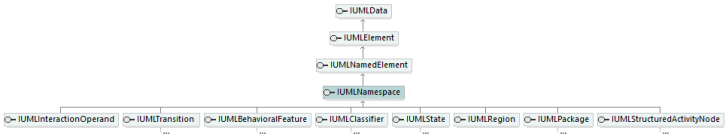
Operation **IUMLNamedElement::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind** 1378			

© 2016-2022 Altova GmbH

17.5.3.5.119 UModelAPI - IUMLNamespace

Interface **IUMLNamespace**

diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** 1142 Interface **IUMLDataAll** 1019 Interface **IUMLElementImport** 1161 Interface **IUMLNamedElement** 1222 Interface **IUMLPackageImport** 1242 Operation **Context** 1143 Operation **Context** 1027 **ImportingNamespace** 1038 **Namespace** 1063 Operation **ImportingNamespace** 1161 Operation **Namespace** 1224 Operation **ImportingNamespace** 1242

Operation **IUMLNamespace::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLElementImport 1161.					

Operation **IUMLNamespace::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	strName	**in**	**string**			
	return	**return**	**IUMLNamedElement** 1222			

Operation **IUMLNamespace::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPackageableElement 1241.					

Operation **IUMLNamespace::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedElement	**in**	**IUMLPackageableElement** 1241			
	return	**return**	**IUMLElementImport** 1161			

Operation **IUMLNamespace::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLNamespace::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedPackage	**in**	**IUMLPackage** 1238			
	return	**return**	**IUMLPackageImport** 1242			

Operation **IUMLNamespace::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipMergedPackage	**in**	**IUMLPackage** 1238			
	return	**return**	**IUMLPackageMerge** 1243			

Operation **IUMLNamespace::Members**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLNamedElement 1222.					

Operation **IUMLNamespace::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLNamedElement 1222.					

Operation **IUMLNamespace::OwnedRules**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLConstraint 1142.					

Operation **IUMLNamespace::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPackageImport 1242.					

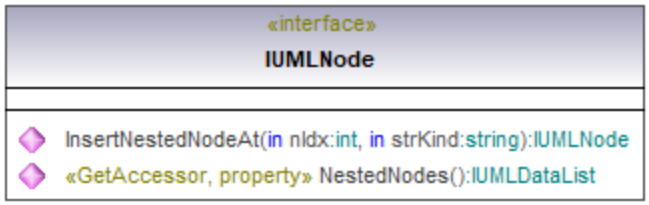
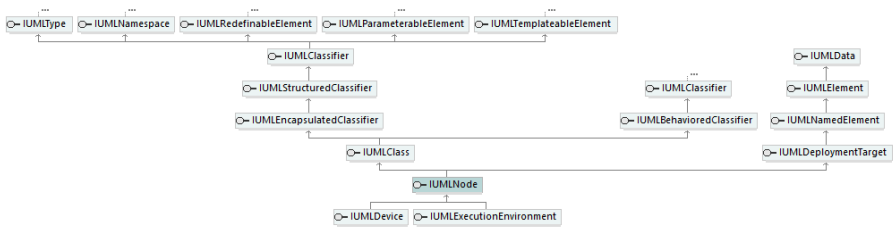
Operation **IUMLNamespace::PackageMerges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPackageMerge 1243.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.120 UModelAPI - IUMLNode

Interface **IUMLNode**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLNode** 1227			Operation **InsertNestedNodeAt** 1046 Operation **InsertNestedNodeAt** 1227

Operation **IUMLNode::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	strKind	**in**	**string**
	return	**return**	**IUMLNode** 1227

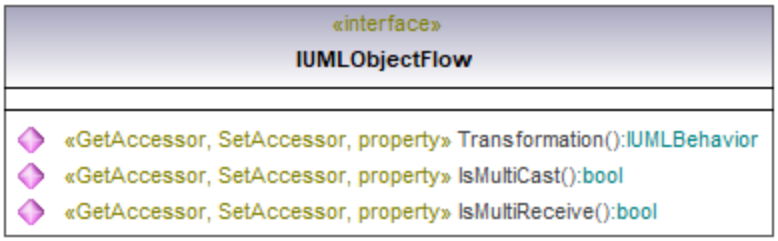
Operation **IUMLNode::NestedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLNode 1227.					

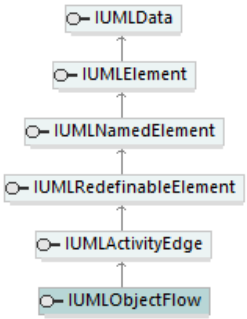
17.5.3.5.121 UModelAPI - IUMLObjectFlow

Interface **IUMLObjectFlow**

diagram



hierarchy



Operation **IUMLObjectFlow::IsMultiCast**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLObjectFlow::IsMultiReceive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLObjectFlow::Transformation**

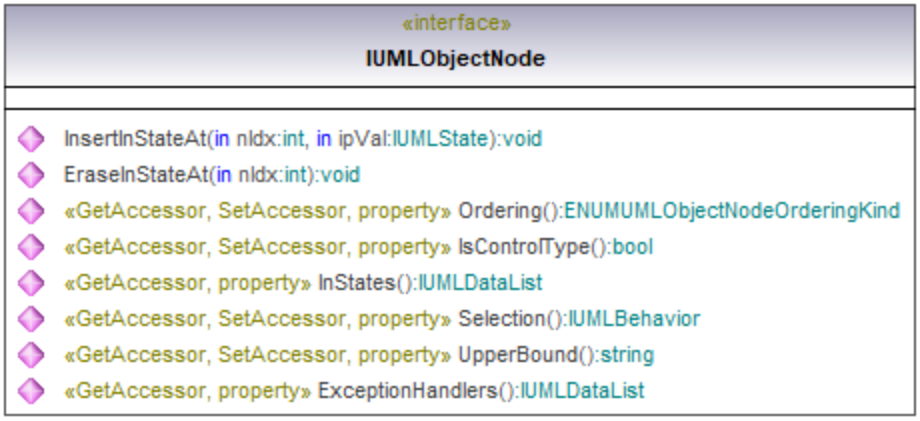
 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

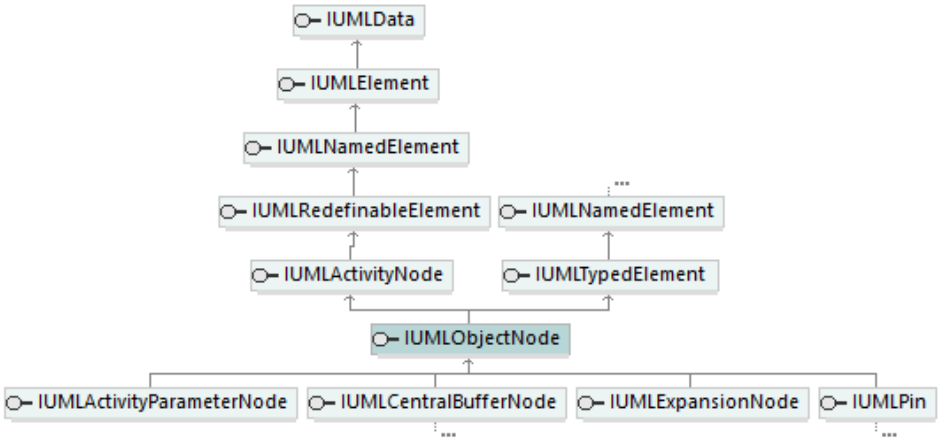
17.5.3.5.122 UModelAPI - IUMLObjectNode

Interface **IUMLObjectNode**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019
Interface **IUMLExceptionHandler** 1166
Operation **ExceptionInput** 1033
Operation **ExceptionInput** 1166

Operation **IUMLObjectNode::EraseInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLObjectNode::ExceptionHandlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLExceptionHandler 1166.					

Operation **IUMLObjectNode::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLState** 1268			
	return	**return**	**void**			

Operation **IUMLObjectNode::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLState 1268.					

Operation **IUMLObjectNode::IsControlType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLObjectNode::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLObjectNodeOrderingKind** 1375			

Operation **IUMLObjectNode::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior** 1110			

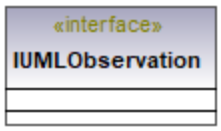
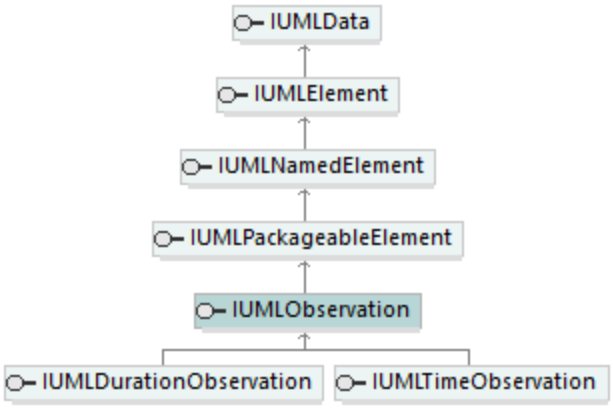
Operation **IUMLObjectNode::UpperBound**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

© 2016-2022 Altova GmbH

17.5.3.5.123 UModelAPI - IUMLObservation

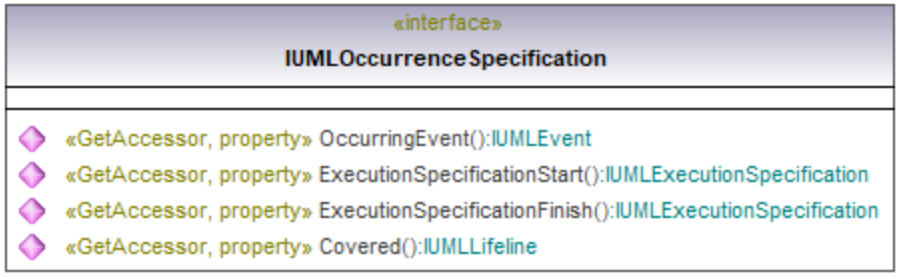
Interface **IUMLObservation**

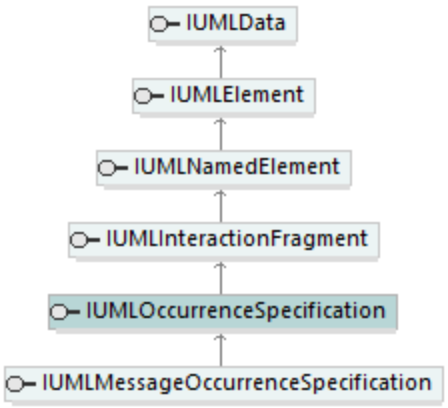
diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLDuration** 1154 Interface **IUMLTimeExpression** 1288 Operation **InsertObservationAt** 1046 Operation **InsertObservationAt** 1154 Operation **InsertObservationAt** 1289

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.124 UModelAPI - IUMLOccurrenceSpecification

Interface **IUMLOccurrenceSpecification**

diagram	

hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLExecutionSpecification 1169 Operation Finish 1034 Start 1083 Operation Finish 1169 Start 1169

Operation **IUMLOccurrenceSpecification::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLLifeline 1209			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExecutionSpecification 1169			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLExecutionSpecification 1169			

Operation **IUMLOccurrenceSpecification::OccurringEvent**

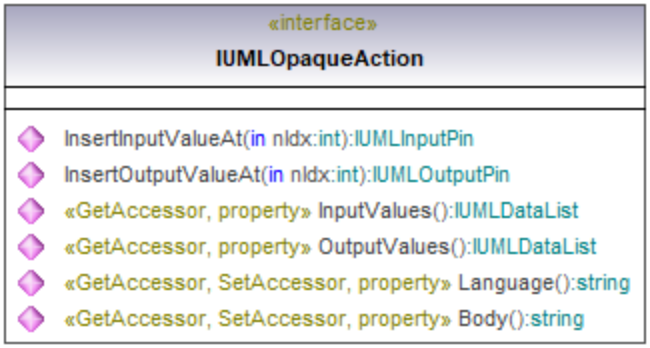
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLEvent 1165			

© 2016-2022 Altova GmbH

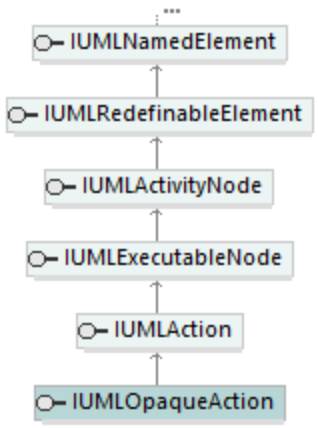
17.5.3.5.125 UModelAPI - IUMLOpaqueAction

Interface **IUMLOpaqueAction**

diagram



hierarchy



Operation **IUMLOpaqueAction::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLOpaqueAction::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLOpaqueAction::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin** 1189			

Operation **IUMLOpaqueAction::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	return	**return**	IUMLOutputPin 1237

Operation **IUMLOpaqueAction::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

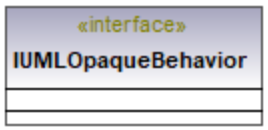
Operation **IUMLOpaqueAction::OutputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

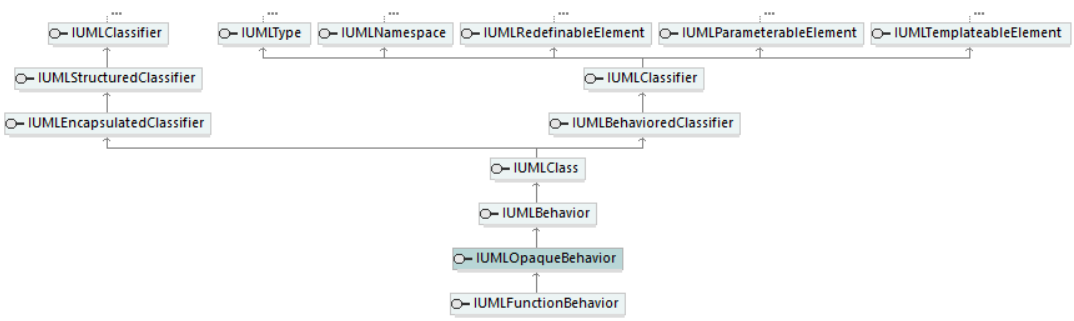
17.5.3.5.126 UModelAPI - IUMLOpaqueBehavior

Interface **IUMLOpaqueBehavior**

diagram



hierarchy



 © 2016-2022 Altova GmbH

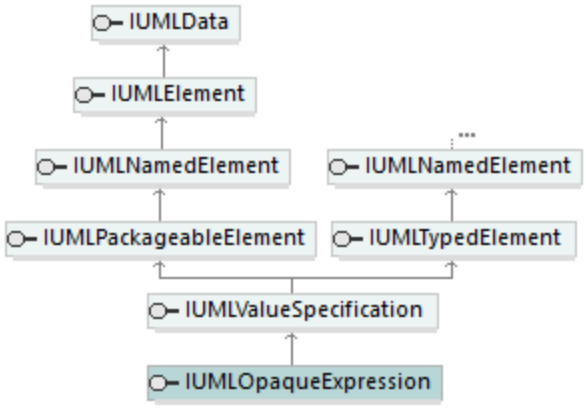
17.5.3.5.127 UModelAPI - IUMLOpaqueExpression

Interface **IUMLOpaqueExpression**

diagram



hierarchy



typedElements

Interface IUMLAbstraction 1090
Interface IUMLDataAll 1019

Operation Mapping 1090 SetNewMapping 1090
Operation Mapping 1061 SetNewMapping 1077

Operation **IUMLOpaqueExpression::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

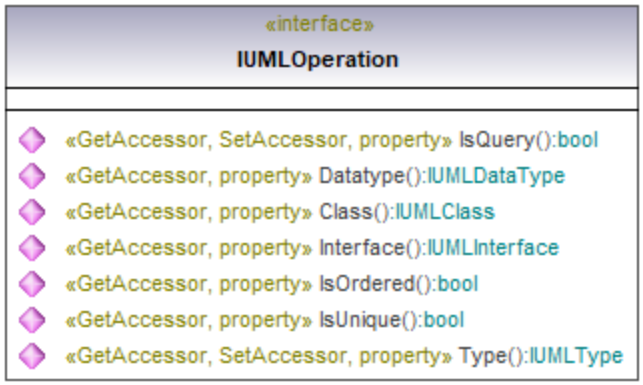
Operation **IUMLOpaqueExpression::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

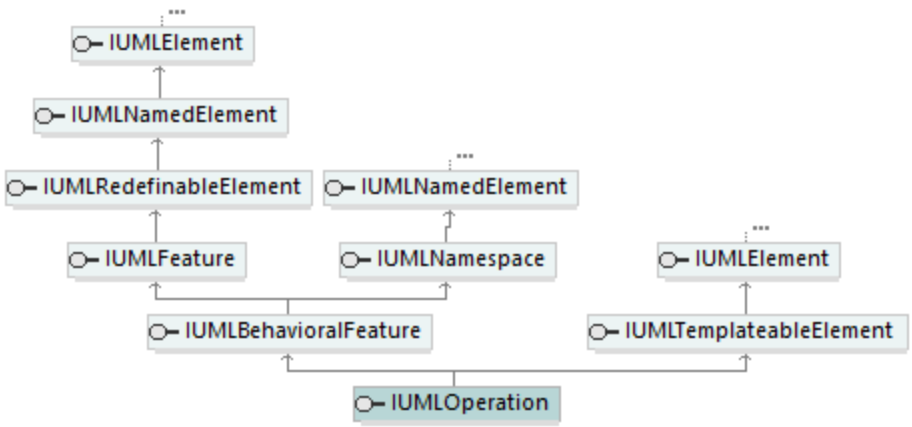
17.5.3.5.128 UModelAPI - IUMLOperation

Interface **IUMLOperation**

diagram



hierarchy



typedElements

Interface IDocument 938
Interface IUMLArtifact 1106
Interface IUMLCallEvent 1118
Interface IUMLCallOperationAction 1118
Interface IUMLClass 1121
Interface IUMLDataAll 1019
Interface IUMLDataType 1146
Interface IUMLGuiSequenceDiagram 1342
Interface IUMLInterface 1199
Interface IUMLMessage 1216
Interface IUMLParameter 1244

Operation GenerateSequenceDiagram 940
Operation InsertOwnedOperationAt 1107
Operation Operation 1118
Operation CallOperation 1119
Operation InsertOwnedOperationAt 1123
Operation CallOperation 1024
CodeOperation 1025
GetOperation 1036
InsertOwnedOperationAt 1049
Operation 1065 SetOperation 1079
Operation InsertOwnedOperationAt 1147
Operation CodeOperation 1343
Operation InsertOwnedOperationAt 1201
Operation GetOperation 1216 SetOperation 1217
Operation Operation 1245

 © 2016-2022 Altova GmbH

Operation **IUMLOperation::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass** 1121			

Operation **IUMLOperation::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType** 1146			

Operation **IUMLOperation::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

Operation **IUMLOperation::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLOperation::IsQuery**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLOperation::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

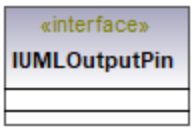
Operation **IUMLOperation::Type**

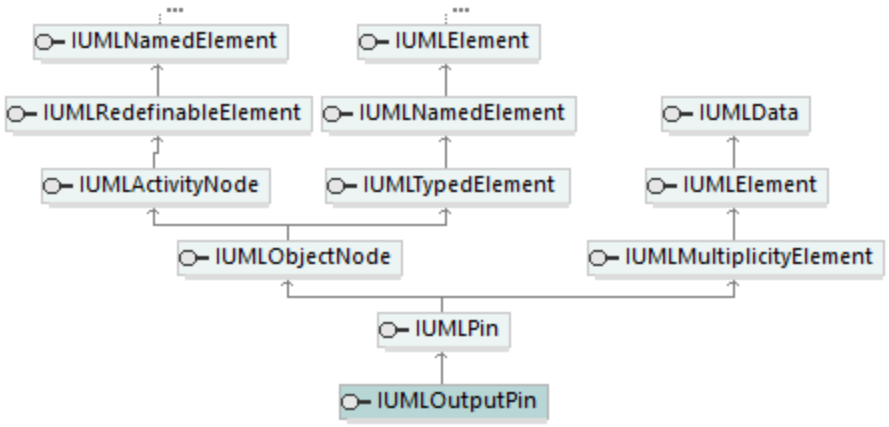
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType** 1294			

17.5.3.5.129 UModelAPI - IUMLOutputPin

Interface **IUMLOutputPin**

diagram	



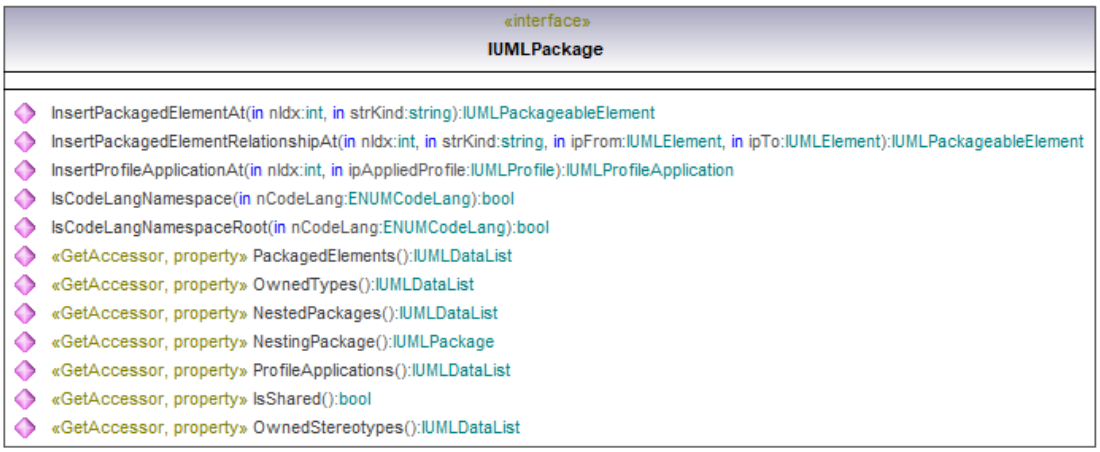
hierarchy	
typedElements	Interface **IUMLAcceptEventAction** 1091 Interface **IUMLCallAction** 1116 Interface **IUMLDataAll** 1019 Interface **IUMLOpaqueAction** 1233 Interface **IUMLValueSpecificationAction** 1302 Operation **InsertEventActionResultAt** 1092 Operation **InsertResultAt** 1116 Operation **InsertEventActionResultAt** 1042 **InsertOutputValueAt** 1047 **InsertResultAt** 1052 **Result** 1074 Operation **InsertOutputValueAt** 1233 Operation **Result** 1303

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

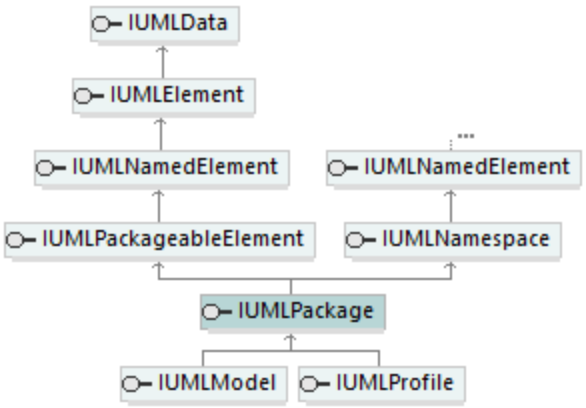
17.5.3.5.130 UModelAPI - IUMLPackage

Interface **IUMLPackage**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IDocument** 938 Interface **IImportSourceDlg** 965 Interface **IModelTransformationDlg** 988 Interface **IUMLDataAll** 1019 Interface **IUMLNamespace** 1225 Interface **IUMLPackage** 1238 Interface **IUMLPackageableElement** 1241 Interface **IUMLPackageImport** 1242 Interface **IUMLPackageMerge** 1243 Interface **IUMLProfileApplication** 1250 Interface **IUMLType** 1294 Operation **RootPackage** 944 Operation **ImportTarget** 966 Operation **SourcePackage** 989 **TargetPackage** 989 Operation **ApplyingPackage** 1022 **ImportedPackage** 1038 **InsertPackageImportAt** 1050 **InsertPackageMergeAt** 1050 **MergedPackage** 1062 **NestingPackage** 1064 **OwningPackage** 1069 **Package** 1070 **ReceivingPackage** 1073 Operation **InsertPackageImportAt** 1226 **InsertPackageMergeAt** 1226 Operation **NestingPackage** 1240 Operation **OwningPackage** 1242 Operation **ImportedPackage** 1242 Operation **MergedPackage** 1243 **ReceivingPackage** 1243 Operation **ApplyingPackage** 1251 Operation **Package** 1294

Operation **IUMLPackage::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLPackageableElement** 1241			

Operation **IUMLPackage::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLElement** 1157			

	ipTo	**in**	IUMLElement 1157			
	return	**return**	IUMLPackageableElement 1241			

Operation **IUMLPackage::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipAppliedProfile	**in**	IUMLProfile 1249			
	return	**return**	IUMLProfileApplication 1250			

Operation **IUMLPackage::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	ENUMCodeLang 1004			
	return	**return**	**bool**			

Operation **IUMLPackage::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	ENUMCodeLang 1004			
	return	**return**	**bool**			

Operation **IUMLPackage::IsShared**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPackage::NestedPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLPackage 1238.					

Operation **IUMLPackage::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLPackage::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLPackage::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

© 2016-2022 Altova GmbH

documentation	A list of elements of type IUMLType 1294.

Operation **IUMLPackage::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPackageableElement 1241.					

Operation **IUMLPackage::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProfileApplication 1250.					

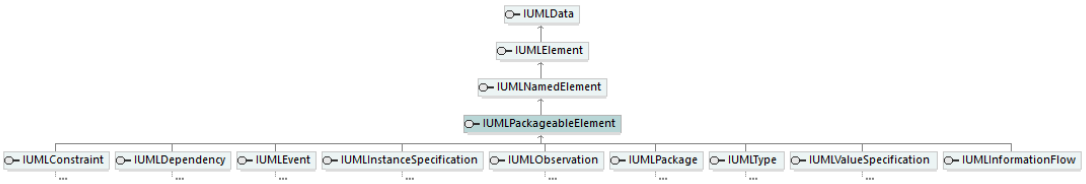
17.5.3.5.131 UModelAPI - IUMLPackageableElement

Interface **IUMLPackageableElement**

diagram



hierarchy



typedElements

Interface **IUMLArtifact** 1106
Interface **IUMLDataAll** 1019
Interface **IUMLElementImport** 1161
Interface **IUMLManifestation** 1214
Interface **IUMLNamespace** 1225
Interface **IUMLPackage** 1238

Operation **InsertManifestationAt** 1107
Operation **ImportedElement** 1038
InsertElementImportAt 1042
InsertManifestationAt 1045
InsertPackagedElementAt 1050
InsertPackagedElementRelationship 1050 **UtilizedElement** 1088
Operation **ImportedElement** 1161
Operation **UtilizedElement** 1214
Operation **InsertElementImportAt** 1226
Operation **InsertPackagedElementAt** 1239
InsertPackagedElementRelationsh

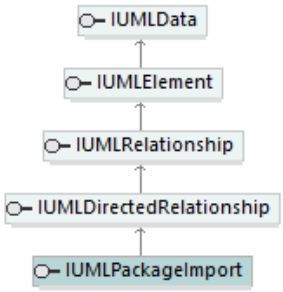
	ipAt 1239

Operation **IUMLPackageableElement::OwningPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

17.5.3.5.132 UModelAPI - IUMLPackageImport

Interface **IUMLPackageImport**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLNamespace 1225 Operation InsertPackageImportAt 1050 Operation InsertPackageImportAt 1226

Operation **IUMLPackageImport::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLPackageImport::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamespace 1225			

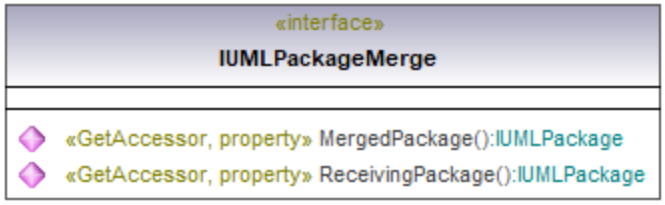
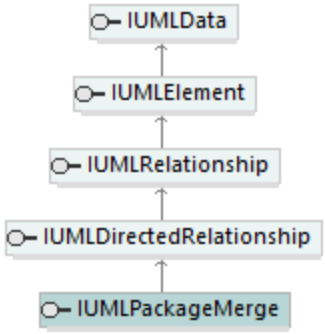
 © 2016-2022 Altova GmbH

Operation **IUMLPackageImport::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLVisibilityKind 1378			

17.5.3.5.133 UModelAPI - IUMLPackageMerge

Interface **IUMLPackageMerge**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLNamespace 1225 Operation InsertPackageMergeAt 1050 Operation InsertPackageMergeAt 1226

Operation **IUMLPackageMerge::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

Operation **IUMLPackageMerge::ReceivingPackage**

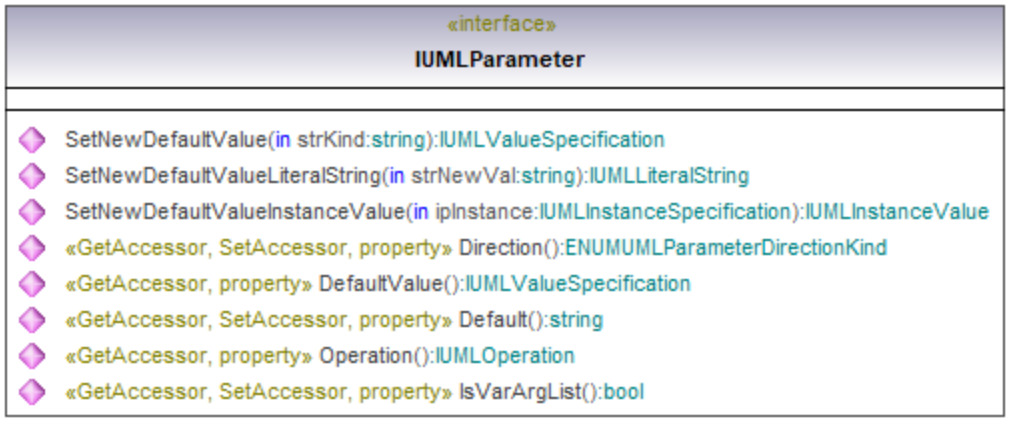
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage 1238			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

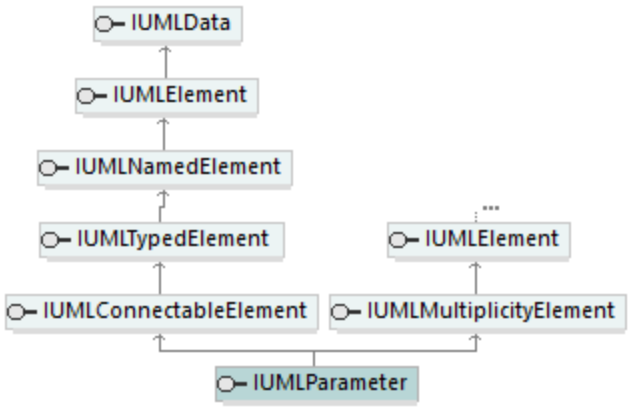
17.5.3.5.134 UModelAPI - IUMLParameter

Interface **IUMLParameter**

diagram



hierarchy



typedElements

Interface **IUMLActivityParameterNode** 1102
Interface **IUMLBehavior** 1110
Interface **IUMLBehavioralFeature** 1112
Interface **IUMLDataAll** 1019
Interface **IUMLValueSpecification** 1299

Operation **Parameter** 1102
Operation **InsertOwnedParameterAt** 1111
Operation **InsertOwnedParameterAt** 1113
Operation **InsertOwnedParameterAt** 1049
OwningParameter 1069
Parameter 1070
Operation **OwningParameter** 1301

Operation **IUMLParameter::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

 © 2016-2022 Altova GmbH

Operation **IUMLParameter::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLParameter::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLParameterDirectionKind** 1376			

Operation **IUMLParameter::IsVarArgList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLParameter::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation** 1236			

Operation **IUMLParameter::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLParameter::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

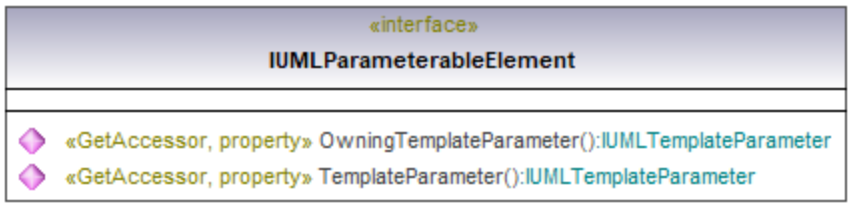
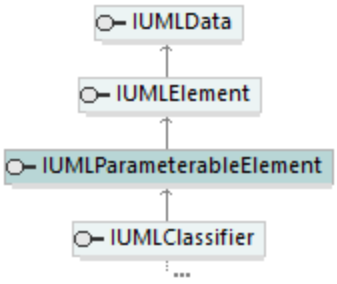
Operation **IUMLParameter::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString** 1213			

© 2016-2022 Altova GmbH

17.5.3.5.135 UModelAPI - IUMLParameterableElement

Interface **IUMLParameterableElement**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLTemplateBinding 1281 Interface IUMLTemplateParameter 1283 Interface IUMLTemplateParameterSubstitution 1284 Operation Actual 1020, InsertParameterSubstitutionAt 1050, OwnedActual 1066, OwnedParameteredElement 1067, ParameteredElement 1070, SetNewOwnedParameteredElement 1078 Operation InsertParameterSubstitutionAt 1282 Operation OwnedParameteredElement 1283, ParameteredElement 1283, SetNewOwnedParameteredElement 1284 Operation Actual 1284, OwnedActual 1285

Operation **IUMLParameterableElement::OwningTemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateParameter 1283			

Operation **IUMLParameterableElement::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateParameter 1283			

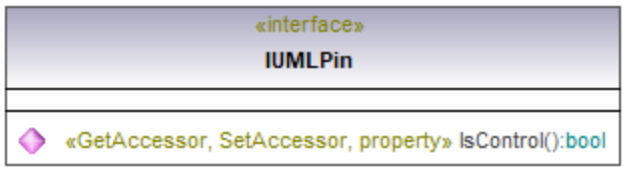
© 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

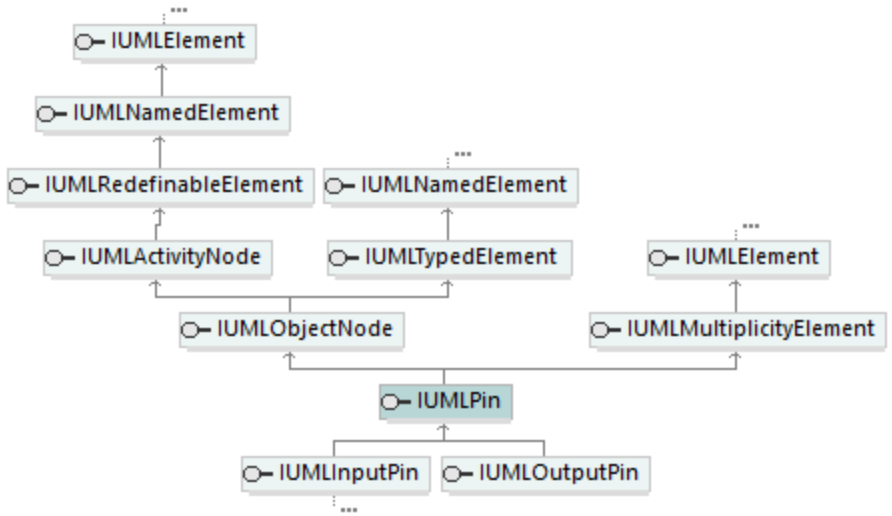
17.5.3.5.136 UModelAPI - IUMLPin

Interface **IUMLPin**

diagram



hierarchy



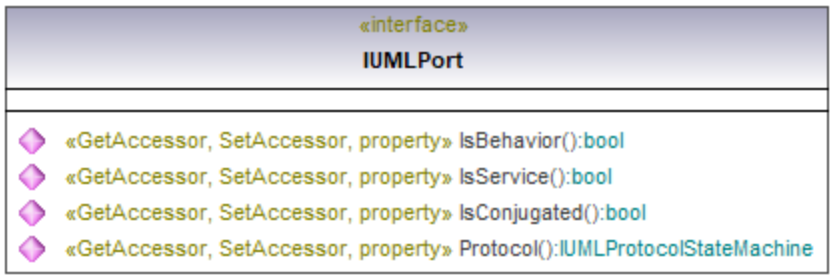
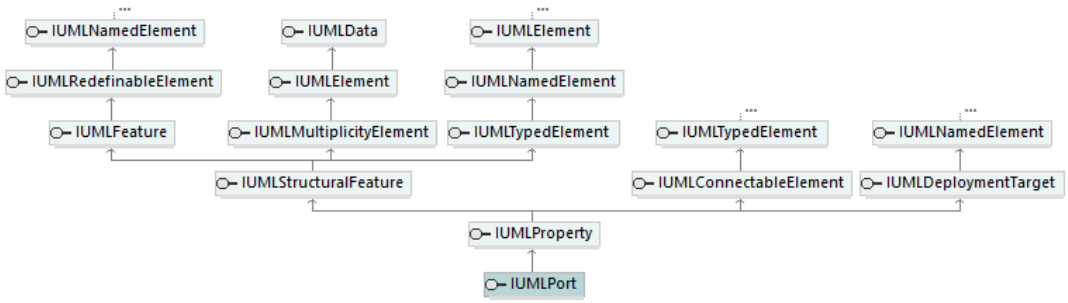
Operation **IUMLPin::IsControl**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.137 UModelAPI - IUMLPort

Interface **IUMLPort**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLStructuredClassifier 1279 Operation InsertOwnedPortAt 1049 Operation InsertOwnedPortAt 1279

Operation **IUMLPort::IsBehavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPort::IsConjugated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPort::IsService**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLPort::Protocol**

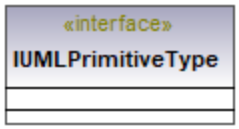
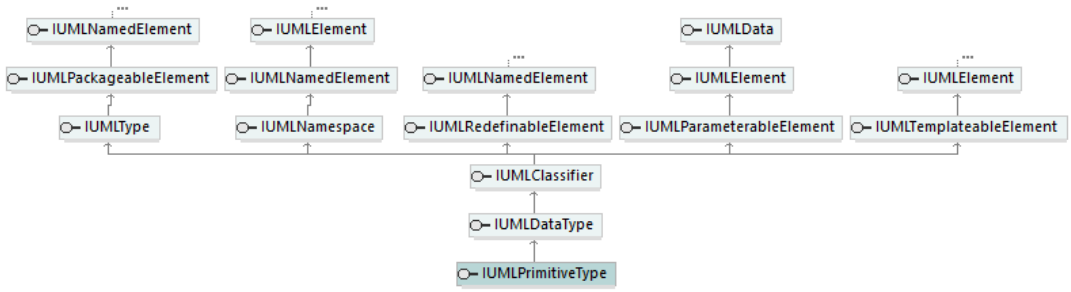
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProtocolStateMachine 1255			

© 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.138 UModelAPI - IUMLPrimitiveType

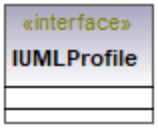
Interface **IUMLPrimitiveType**

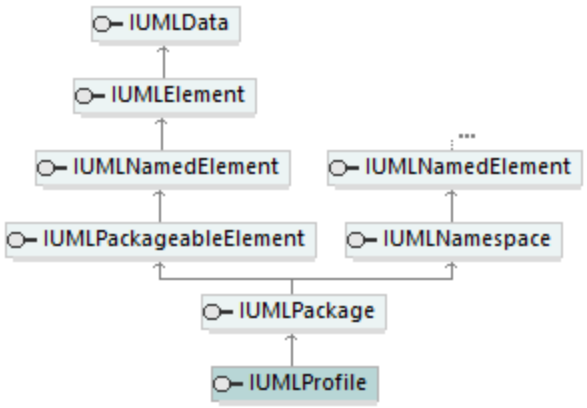
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.139 UModelAPI - IUMLProfile

Interface **IUMLProfile**

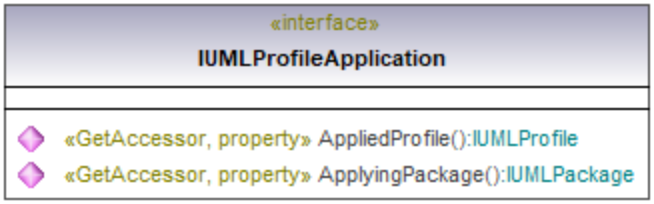
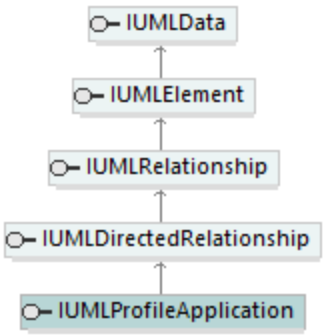
diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLPackage** 1238 Interface **IUMLProfileApplication** 1250 Operation **AppliedProfile** 1022 **InsertProfileApplicationAt** 1051 Operation **InsertProfileApplicationAt** 1240 Operation **AppliedProfile** 1251

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.140 UModelAPI - IUMLProfileApplication

Interface **IUMLProfileApplication**

diagram	
hierarchy	

 © 2016-2022 Altova GmbH

typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLPackage** 1238	Operation **InsertProfileApplicationAt** 1051 Operation **InsertProfileApplicationAt** 1240

Operation **IUMLProfileApplication::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProfile** 1249			

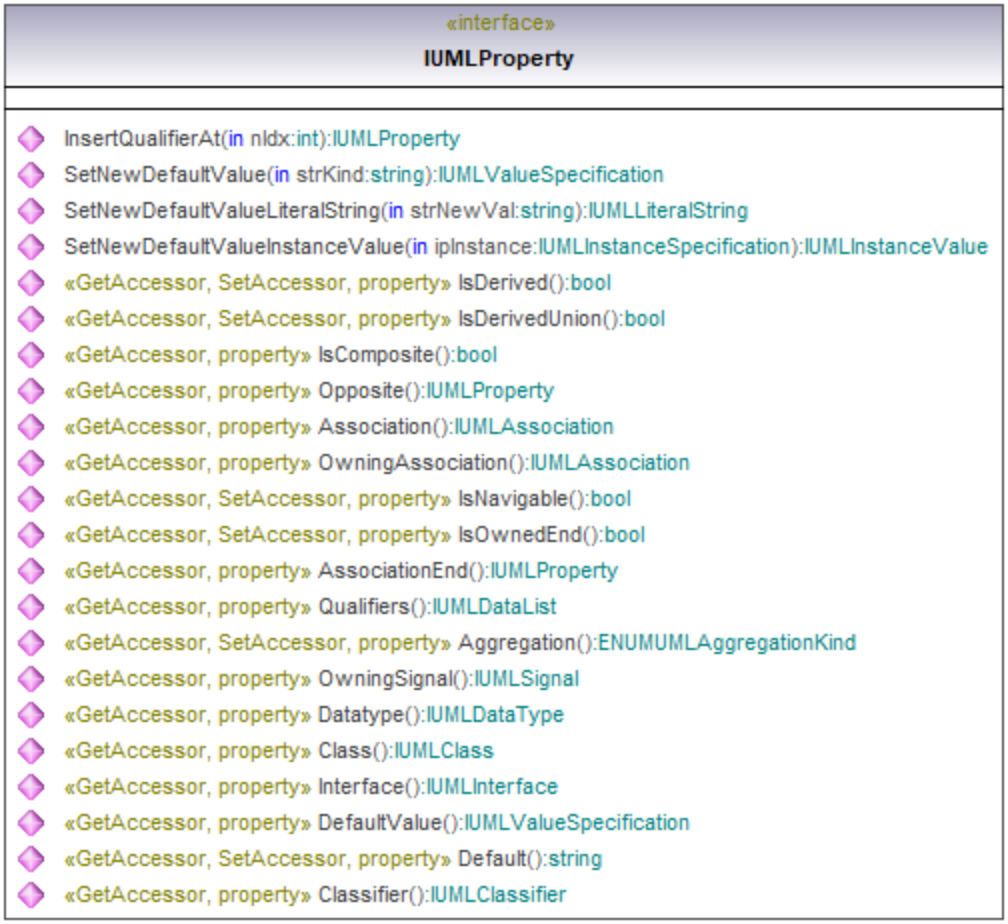
Operation **IUMLProfileApplication::ApplyingPackage**

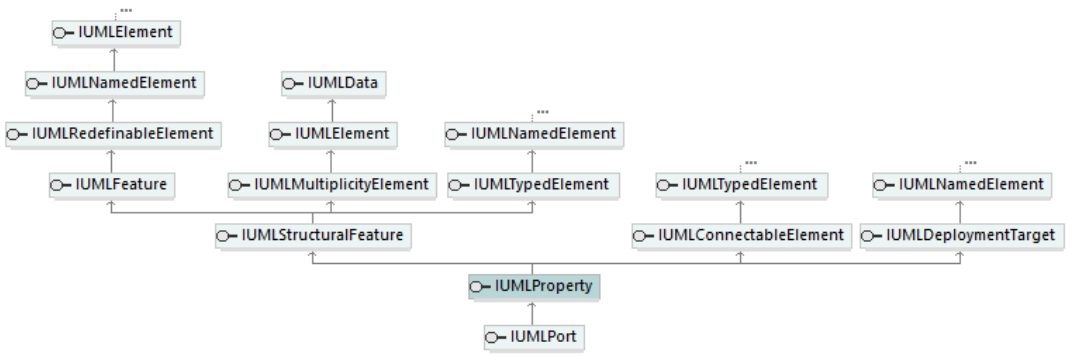
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

17.5.3.5.141 UModelAPI - IUMLProperty

Interface **IUMLProperty**

diagram



hierarchy	
typedElements	Interface IUMLArtifact 1106 Interface IUMLDataAll 1019 Interface IUMLDataType 1146 Interface IUMLInterface 1199 Interface IUMLProperty 1251 Interface IUMLSignal 1264 Interface IUMLStructuredClassifier 1279 Interface IUMLValueSpecification 1299 Operation InsertOwnedAttributeAt 1107 Operation AssociationEnd 1023 Operation InsertOwnedAttributeAt 1047 Operation InsertQualifierAt 1051 Operation Opposite 1065 Operation OwningProperty 1069 Operation InsertOwnedAttributeAt 1147 Operation InsertOwnedAttributeAt 1201 Operation AssociationEnd 1252 Operation InsertQualifierAt 1253 Operation Opposite 1254 Operation InsertOwnedAttributeAt 1265 Operation InsertOwnedAttributeAt 1279 Operation OwningProperty 1301

Operation **IUMLProperty::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLAggregationKind 1369			

Operation **IUMLProperty::Association**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLAssociation 1108			

Operation **IUMLProperty::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLProperty 1251			

Operation **IUMLProperty::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLClass 1121			

Operation **IUMLProperty::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**IUMLClassifier** 1124

Operation **IUMLProperty::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType** 1146			

Operation **IUMLProperty::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLProperty::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLProperty::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLProperty::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

Operation **IUMLProperty::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsDerived**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsDerivedUnion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsNavigable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLProperty::IsOwnedEnd**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**bool**

Operation **IUMLProperty::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLProperty::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation** 1108			

Operation **IUMLProperty::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal** 1264			

Operation **IUMLProperty::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

Operation **IUMLProperty::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLProperty::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

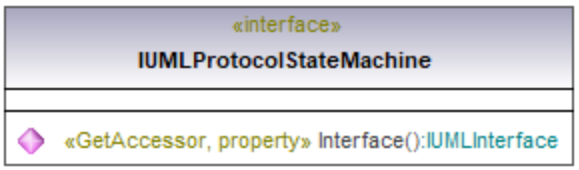
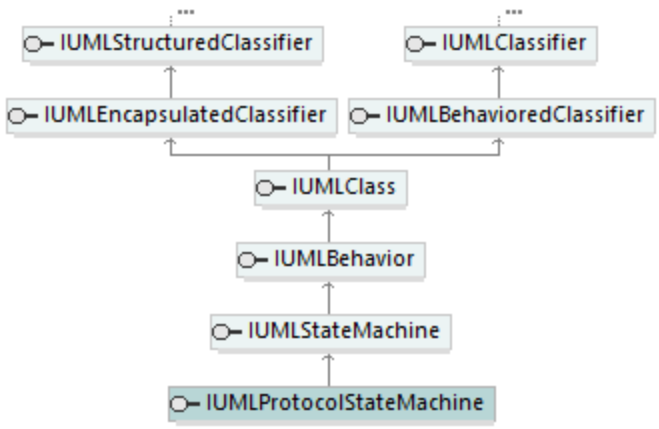
Operation **IUMLProperty::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString** 1213			

© 2016-2022 Altova GmbH

17.5.3.5.142 UModelAPI - IUMLProtocolStateMachine

Interface **IUMLProtocolStateMachine**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInterface** 1199 Interface **IUMLPort** 1248 Operation **Protocol** 1072 **SetNewProtocol** 1078 Operation **Protocol** 1202 **SetNewProtocol** 1202 Operation **Protocol** 1248

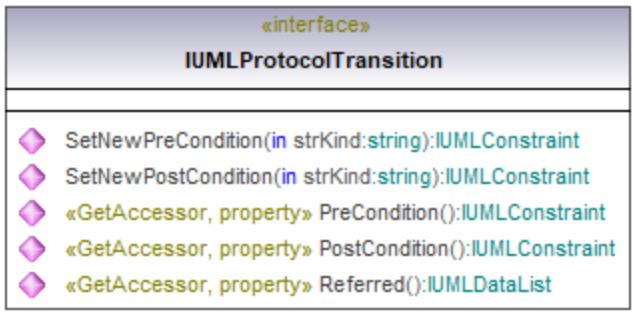
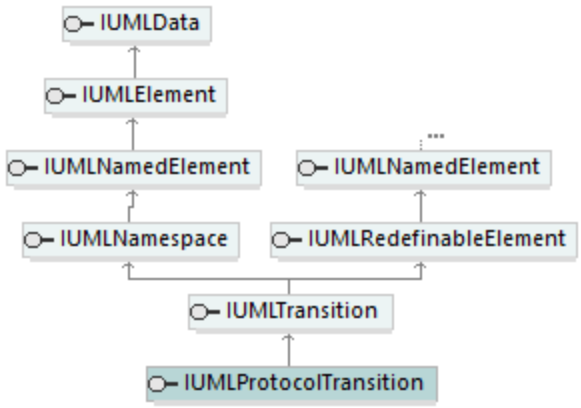
Operation **IUMLProtocolStateMachine::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

© 2016-2022 Altova GmbH

17.5.3.5.143 UModelAPI - IUMLProtocolTransition

Interface **IUMLProtocolTransition**

diagram	
hierarchy	
typedElements	Interface IUMLConstraint 1142 Interface IUMLDataAll 1019 Operation OwningTransition 1143 Operation OwningTransition 1070

Operation **IUMLProtocolTransition::PostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLProtocolTransition::PreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLProtocolTransition::Referred**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLProtocolTransition::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	strKind	**in**	**string**
	return	**return**	**IUMLConstraint** 1142

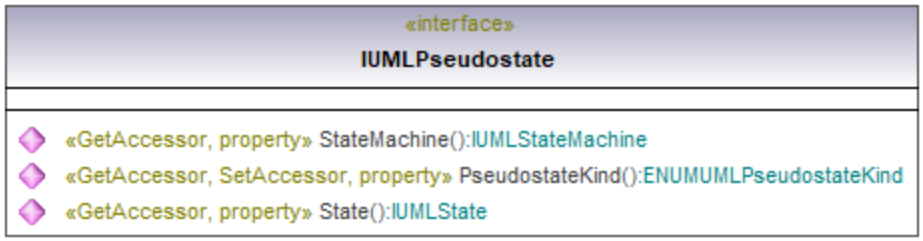
Operation **IUMLProtocolTransition::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

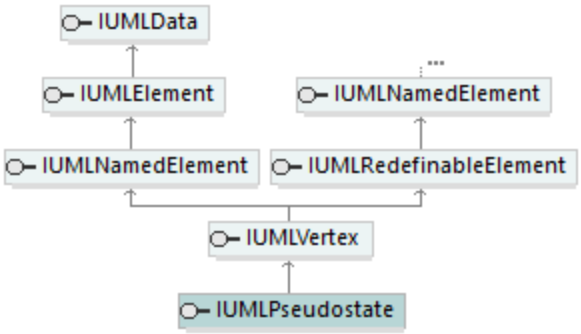
17.5.3.5.144 UModelAPI - IUMLPseudostate

Interface **IUMLPseudostate**

diagram



hierarchy



typedElements

Interface **IUMLConnectionPointReference** 1138	Operation **InsertEntryAt** 1139 **InsertExitAt** 1139
Interface **IUMLDataAll** 1019	Operation **InsertConnectionPointAt** 1041 **InsertEntryAt** 1042 **InsertExitAt** 1042
Interface **IUMLState** 1268	Operation **InsertConnectionPointAt** 1269
Interface **IUMLStateMachine** 1272	Operation **InsertConnectionPointAt** 1273

Operation **IUMLPseudostate::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	ENUMUMLPseudostateKind 1377

Operation **IUMLPseudostate::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLState 1268			

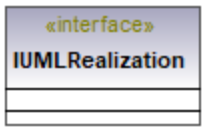
Operation **IUMLPseudostate::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLStateMachine 1272			

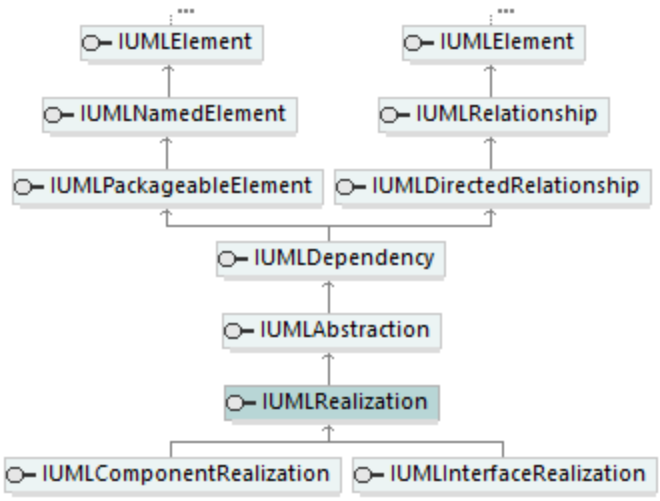
17.5.3.5.145 UModelAPI - IUMLRealization

Interface **IUMLRealization**

diagram



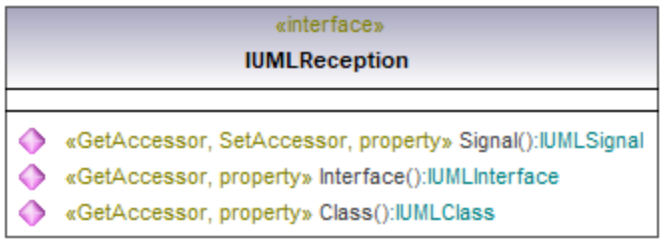
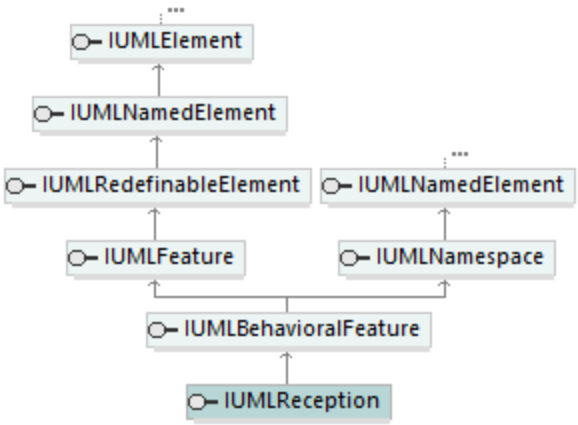
hierarchy



 © 2016-2022 Altova GmbH

17.5.3.5.146 UModelAPI - IUMLReception

Interface **IUMLReception**

diagram	
hierarchy	
typedElements	Interface **IUMLClass** 1121 Interface **IUMLDataAll** 1019 Interface **IUMLInterface** 1199 Operation **InsertOwnedReceptionAt** 1123 Operation **InsertOwnedReceptionAt** 1049 Operation **InsertOwnedReceptionAt** 1201

Operation **IUMLReception::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass** 1121			

Operation **IUMLReception::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface** 1199			

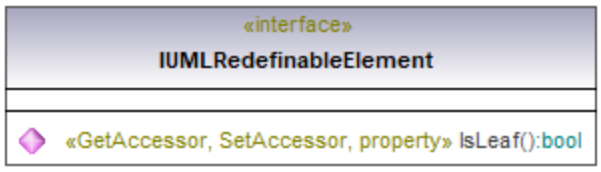
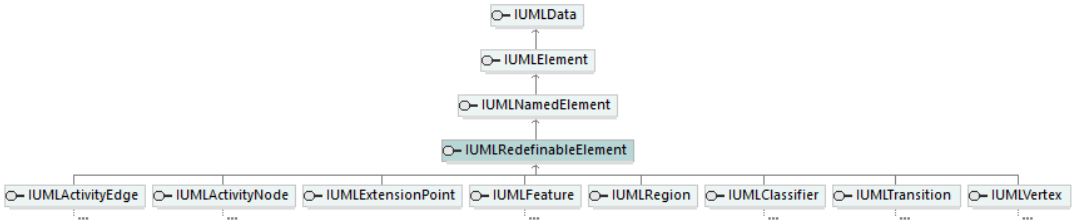
Operation **IUMLReception::Signal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal** 1264			

© 2016-2022 Altova GmbH

17.5.3.5.147 UModelAPI - IUMLRedefinableElement

Interface **IUMLRedefinableElement**

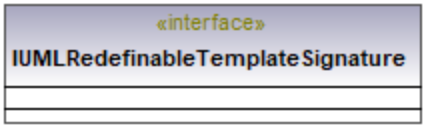
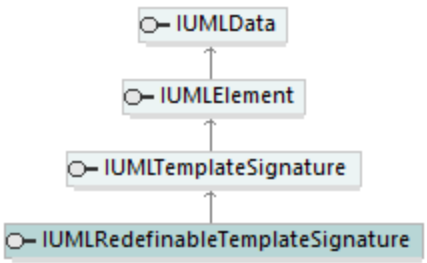
diagram	
hierarchy	

Operation **IUMLRedefinableElement::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.148 UModelAPI - IUMLRedefinableTemplateSignature

Interface **IUMLRedefinableTemplateSignature**

diagram	
hierarchy	

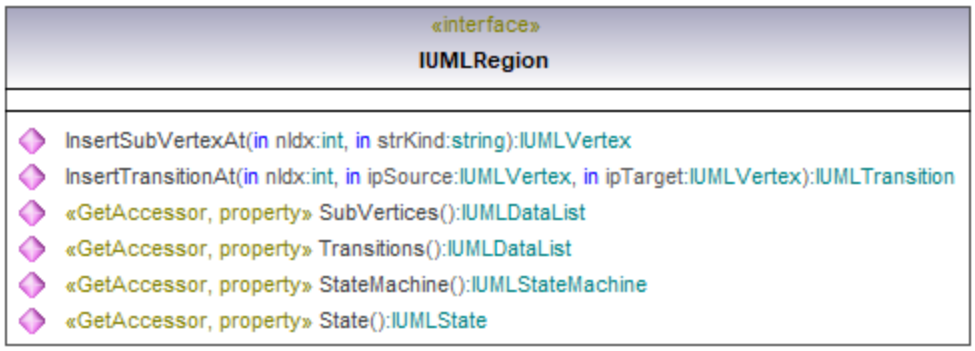
 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

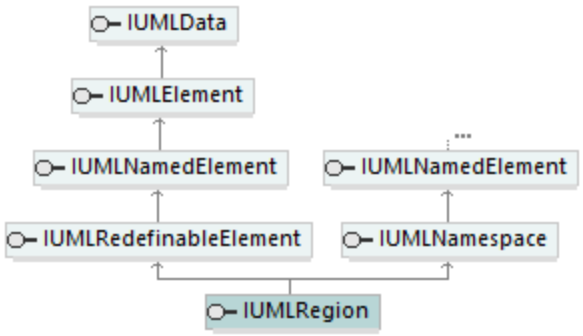
17.5.3.5.149 UModelAPI - IUMLRegion

Interface **IUMLRegion**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019
Interface **IUMLState** 1268
Interface **IUMLStateMachine** 1272
Interface **IUMLVertex** 1303
Operation **Container** 1027 **InsertRegionAt** 1051
Operation **InsertRegionAt** 1270
Operation **InsertRegionAt** 1273
Operation **Container** 1304

Operation **IUMLRegion::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLVertex** 1303			

Operation **IUMLRegion::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSource	**in**	**IUMLVertex** 1303			
	ipTarget	**in**	**IUMLVertex** 1303			

© 2016-2022 Altova GmbH

	return	**return**	IUMLTransition 1291			

Operation **IUMLRegion::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLState 1268			

Operation **IUMLRegion::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLStateMachine 1272			

Operation **IUMLRegion::SubVertices**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLVertex 1303.					

Operation **IUMLRegion::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLTransition 1291.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

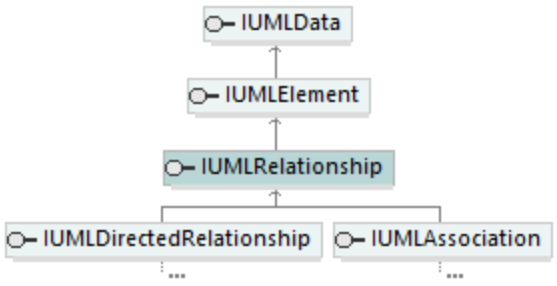
17.5.3.5.150 UModelAPI - IUMLRelationship

Interface **IUMLRelationship**

diagram	



 © 2016-2022 Altova GmbH

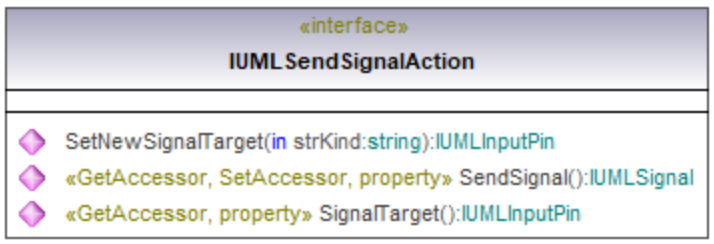
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInformationFlow** 1136 Operation **InsertInformationFlowRealizationAt** 1044 Operation **InsertInformationFlowRealizationAt** 1187

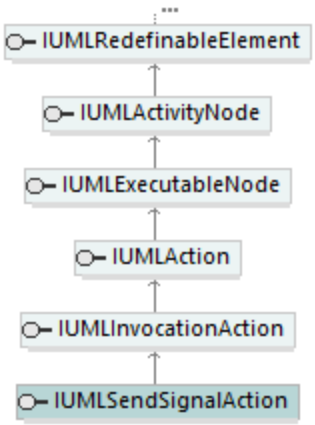
Operation **IUMLRelationship::RelatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLElement 1157.					

17.5.3.5.151 UModelAPI - IUMLSendSignalAction

Interface **IUMLSendSignalAction**

diagram	

hierarchy	

Operation **IUMLSendSignalAction::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal** 1264			

Operation **IUMLSendSignalAction::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin** 1189			

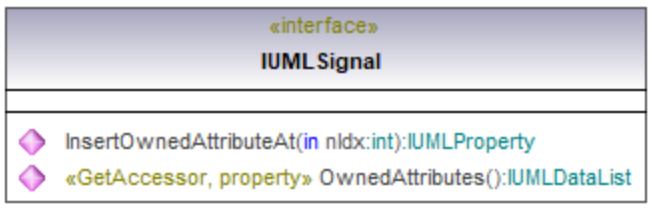
Operation **IUMLSendSignalAction::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin** 1189			

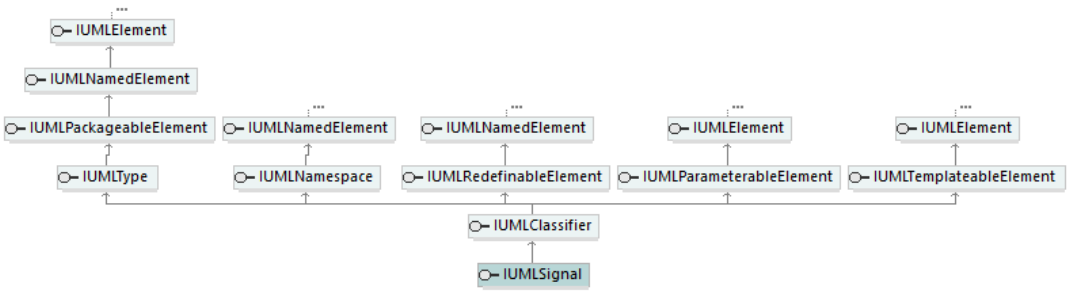
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.152 UModelAPI - IUMLSignal

Interface **IUMLSignal**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLProperty** 1251 Interface **IUMLReception** 1259 Interface **IUMLSendSignalAction** 1263 Interface **IUMLSignalEvent** 1265 Operation **OwningSignal** 1069 **SendSignal** 1075 **Signal** 1082 Operation **OwningSignal** 1254 Operation **Signal** 1259 Operation **SendSignal** 1264 Operation **Signal** 1266

Operation **IUMLSignal::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

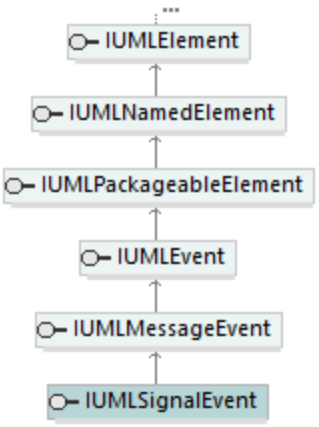
Operation **IUMLSignal::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLProperty 1251.					

17.5.3.5.153 UModelAPI - IUMLSignalEvent

Interface **IUMLSignalEvent**

diagram	

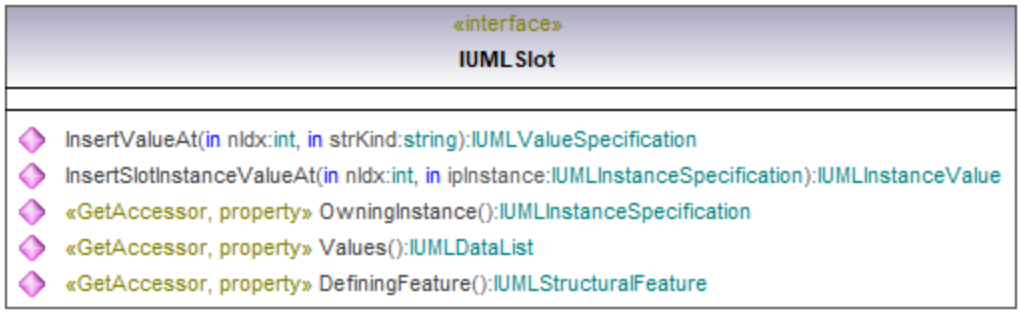
hierarchy	

Operation **IUMLSignalEvent::Signal**

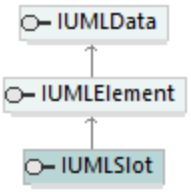
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLSignal 1264			

17.5.3.5.154 UModelAPI - IUMLSlot

Interface **IUMLSlot**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLInstanceSpecification** 1190 Interface **IUMLValueSpecification** 1299 Operation **InsertSlotAt** 1052 **OwningSlot** 1069 Operation **InsertSlotAt** 1191 Operation **OwningSlot** 1302

Operation **IUMLSlot::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStructuralFeature** 1276			

Operation **IUMLSlot::InsertSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipInstance	**in**	**IUMLInstanceSpecification** 1190			
	return	**return**	**IUMLInstanceValue** 1192			

Operation **IUMLSlot::InsertValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLSlot::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification** 1190			

Operation **IUMLSlot::Values**

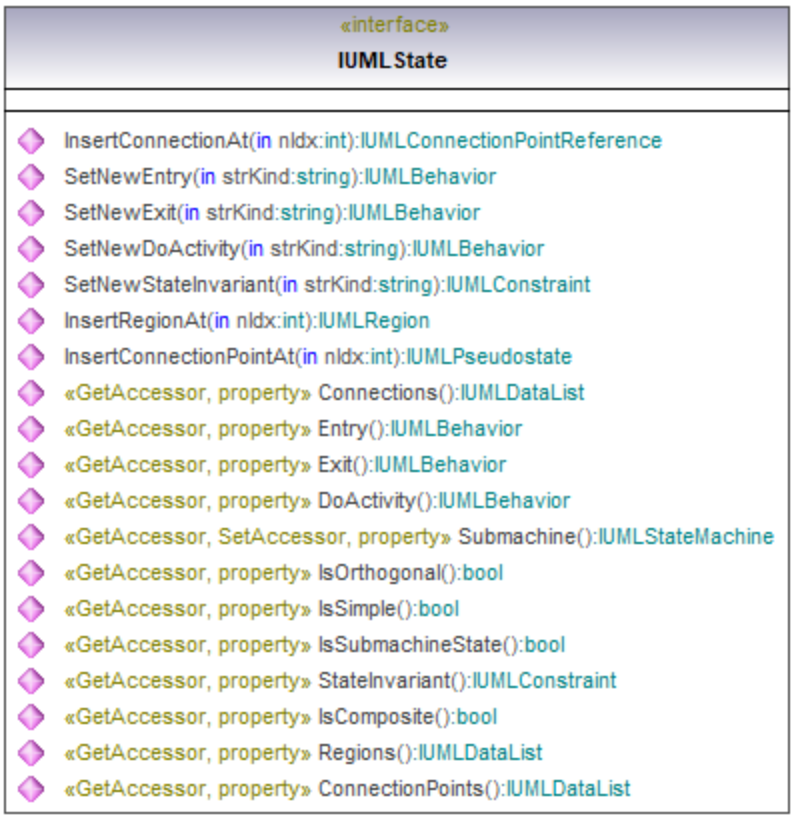
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLValueSpecification 1299.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

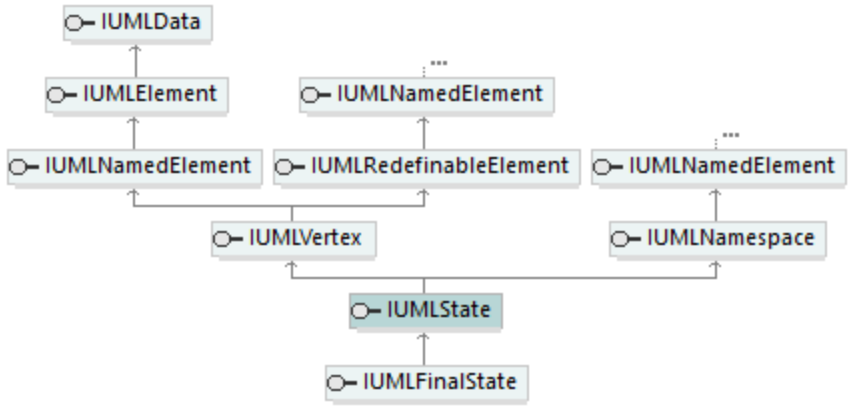
17.5.3.5.155 UModelAPI - IUMLState

Interface **IUMLState**

diagram



hierarchy



typedElements

Interface **IUMLConnectionPointReference**

Operation **State** 1140

 © 2016-2022 Altova GmbH

	Interface **IUMLConstraint** 1142 Interface **IUMLDataAll** 1019 Interface **IUMLObjectNode** 1229 Interface **IUMLPseudostate** 1257 Interface **IUMLRegion** 1261	Operation **OwningState** 1143 Operation **InsertInStateAt** 1044 **OwningState** 1069 **State** 1083 Operation **InsertInStateAt** 1230 Operation **State** 1258 Operation **State** 1262

Operation **IUMLState::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPseudostate 1257.					

Operation **IUMLState::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLConnectionPointReference 1138.					

Operation **IUMLState::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLState::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLState::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLState::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConnectionPointReference** 1138			

Operation **IUMLState::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate** 1257			

Operation **IUMLState::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLRegion 1261			

Operation **IUMLState::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::IsOrthogonal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::IsSimple**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::IsSubmachineState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLState::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLRegion 1261.					

Operation **IUMLState::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLState::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLState::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior** 1110			

Operation **IUMLState::SetNewStateInvariant**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLState::StateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint** 1142			

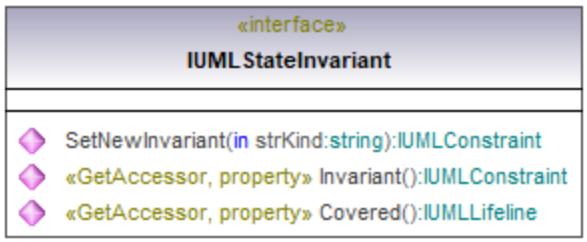
Operation **IUMLState::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine** 1272			

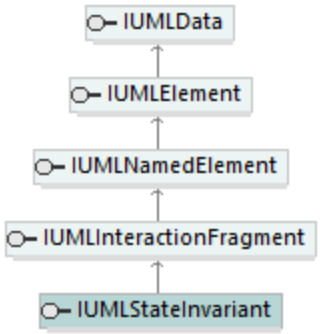
17.5.3.5.156 UModelAPI - IUMLStateInvariant

Interface **IUMLStateInvariant**

diagram



hierarchy



Operation **IUMLStateInvariant::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline** 1209			

Operation **IUMLStateInvariant::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

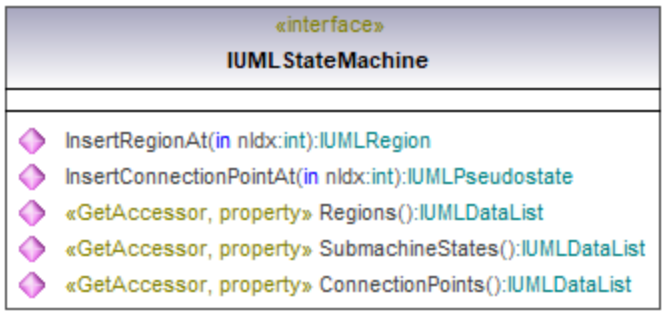
Operation **IUMLStateInvariant::SetNewInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	IUMLConstraint 1142			

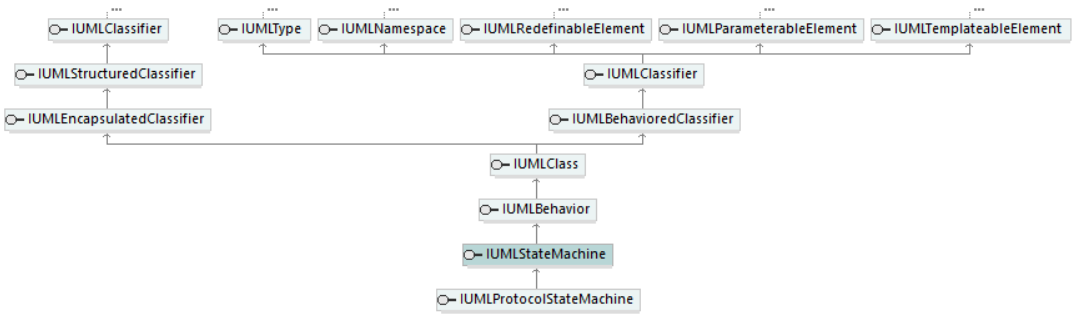
17.5.3.5.157 UModelAPI - IUMLStateMachine

Interface **IUMLStateMachine**

diagram



hierarchy



typedElements

Interface IDocument 938
Interface IUMLDataAll 1019
Interface IUMLPseudostate 1257
Interface IUMLRegion 1261
Interface IUMLState 1268

Operation GenerateStateMachineCode 941
Operation StateMachine 1083 Submachine 1084
Operation StateMachine 1258
Operation StateMachine 1262
Operation Submachine 1271

Operation **IUMLStateMachine::ConnectionPoints**

 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLPseudostate 1257.					

Operation **IUMLStateMachine::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate** 1257			

Operation **IUMLStateMachine::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion** 1261			

Operation **IUMLStateMachine::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLRegion 1261.					

Operation **IUMLStateMachine::SubmachineStates**

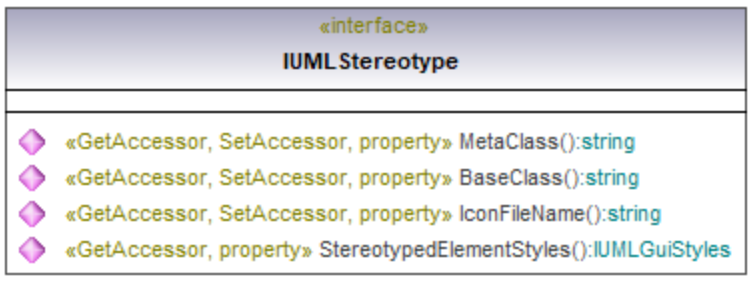
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLState 1268.					

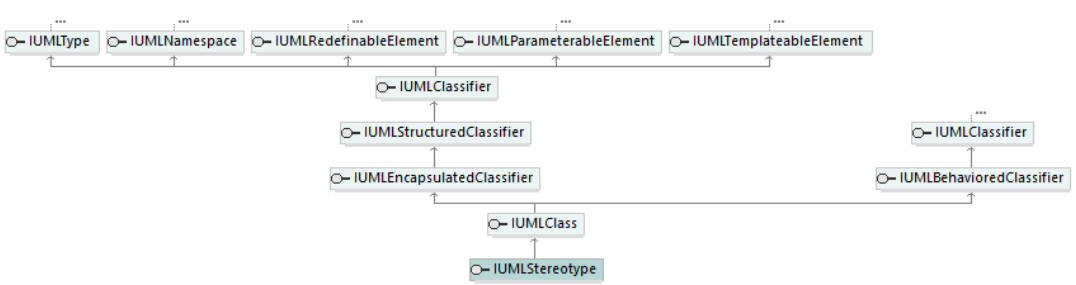
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.158 UModelAPI - IUMLStereotype

Interface **IUMLStereotype**

diagram



hierarchy	
typedElements	Interface IUMLDataAll 1019 Operation ApplyStereotype 1023 GetStereotypeApplicationForStereotype 1036 IsStereotypeApplied 1058 Stereotype 1083 UnapplyStereotype 1087 Interface IUMLElement 1157 Operation ApplyStereotype 1158 GetStereotypeApplicationForStereotype 1159 IsStereotypeApplied 1159 UnapplyStereotype 1160 Interface IUMLStereotypeApplication 1275 Operation Stereotype 1276

Operation **IUMLStereotype::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLStereotype::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLStereotype::MetaClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

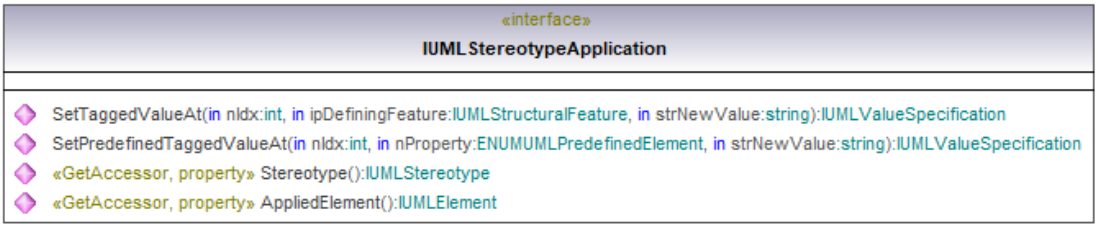
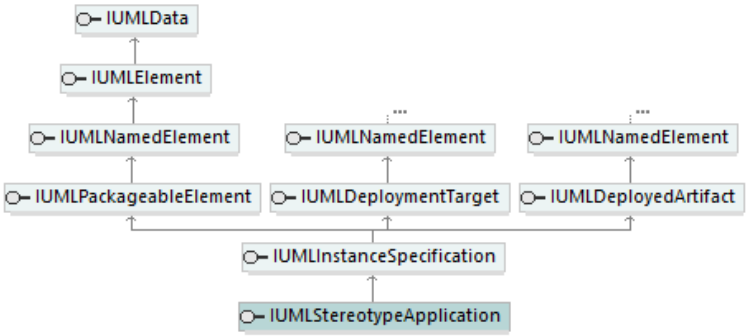
Operation **IUMLStereotype::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiStyles 1346			

© 2016-2022 Altova GmbH

17.5.3.5.159 UModelAPI - IUMLStereotypeApplication

Interface **IUMLStereotypeApplication**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLElement** 1157 Operation **ApplyPredefinedStereotype** 1023 **ApplyStereotype** 1023 **GetStereotypeApplicationForPredefinedStereotype** 1036 **GetStereotypeApplicationForStereotype** 1036 Operation **ApplyPredefinedStereotype** 1158 **ApplyStereotype** 1158 **GetStereotypeApplicationForPredefinedStereotype** 1159 **GetStereotypeApplicationForStereotype** 1159

Operation **IUMLStereotypeApplication::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

Operation **IUMLStereotypeApplication::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nProperty	**in**	**ENUMUMLPredefinedElement** 1376			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLStereotypeApplication::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature** 1276			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

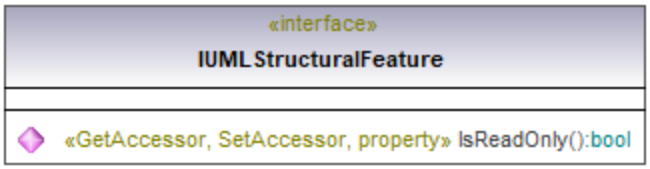
Operation **IUMLStereotypeApplication::Stereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStereotype** 1273			

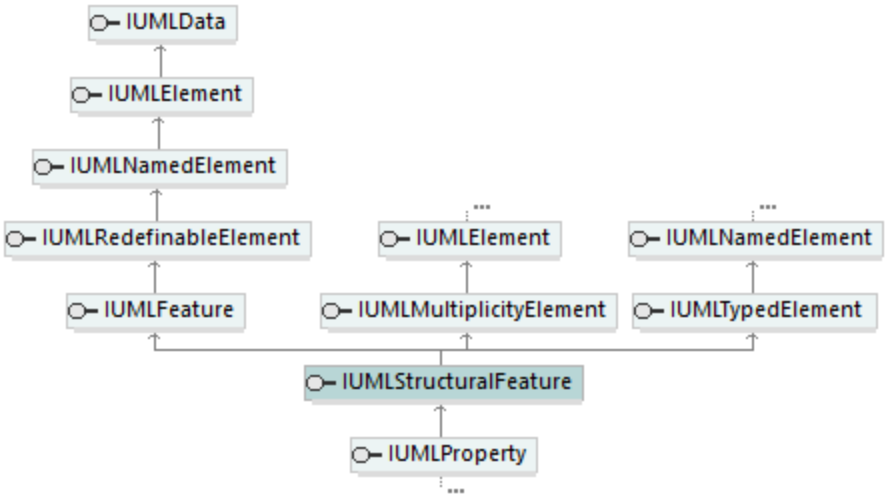
17.5.3.5.160 UModelAPI - IUMLStructuralFeature

Interface **IUMLStructuralFeature**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019

Operation **DefiningFeature** 1028 **InsertSlotAt** 1052 **SetSlotInstanceValueAt** 1080 **SetSlotValueAt** 1080 **SetTaggedValueAt** 1081

 © 2016-2022 Altova GmbH

	Interface **IUMLInstanceSpecification** 1190 Interface **IUMLSlot** 1266 Interface **IUMLStereotypeApplication** 1275	Operation **InsertSlotAt** 1191 **SetSlotInstanceValueAt** 1191 **SetSlotValueAt** 1192 Operation **DefiningFeature** 1267 Operation **SetTaggedValueAt** 1276

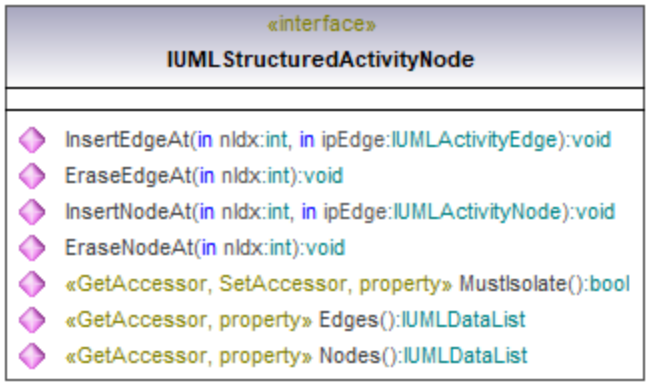
Operation **IUMLStructuralFeature::IsReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

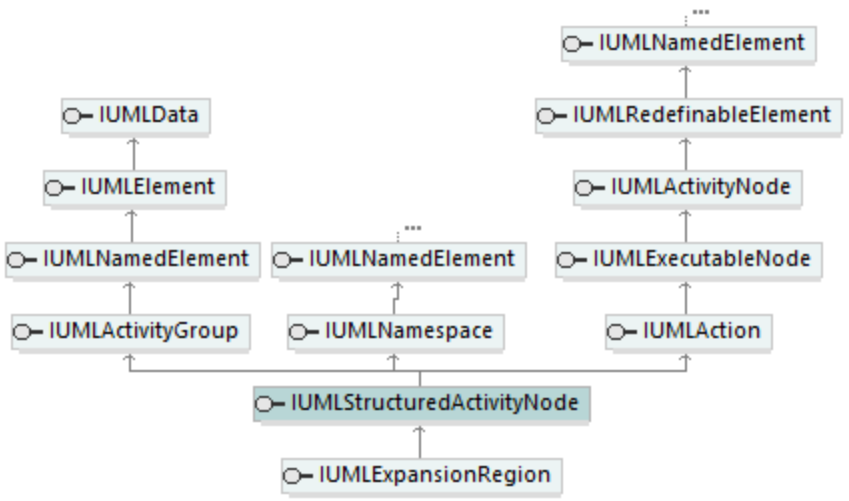
17.5.3.5.161 UModelAPI - IUMLStructuredActivityNode

Interface **IUMLStructuredActivityNode**

diagram



hierarchy



Operation **IUMLStructuredActivityNode::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLStructuredActivityNode::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge** 1097			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityNode** 1101			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLStructuredActivityNode::Nodes**

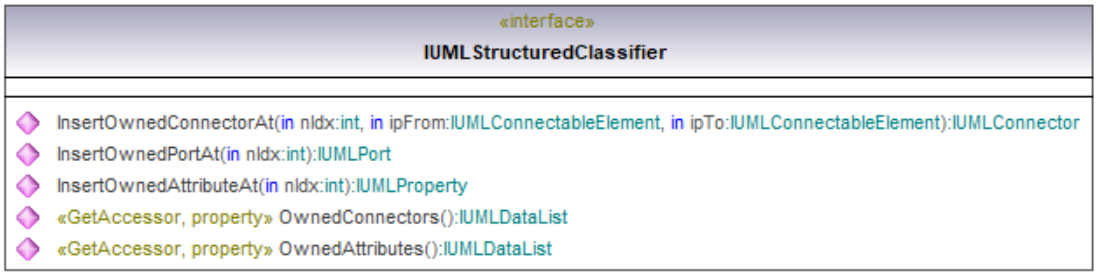
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

© 2016-2022 Altova GmbH

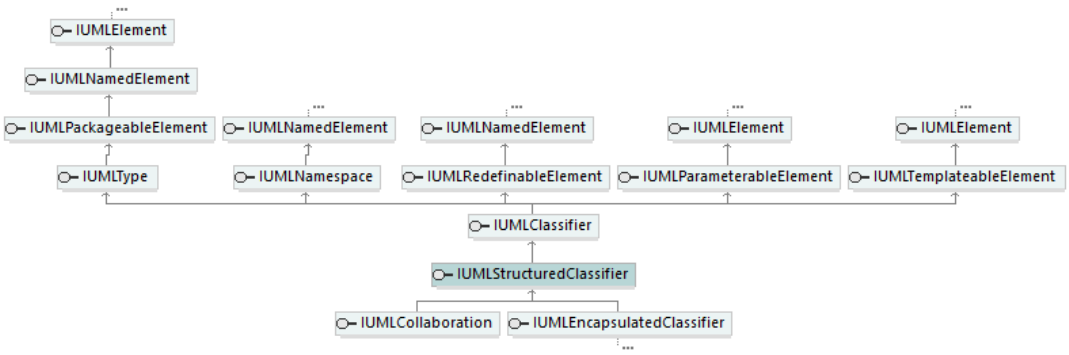
17.5.3.5.162 UModelAPI - IUMLStructuredClassifier

Interface **IUMLStructuredClassifier**

diagram



hierarchy



Operation **IUMLStructuredClassifier::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLStructuredClassifier::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFrom	**in**	**IUMLConnectableElement** 1137			
	ipTo	**in**	**IUMLConnectableElement** 1137			
	return	**return**	**IUMLConnector** 1140			

Operation **IUMLStructuredClassifier::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPort** 1248			

Operation **IUMLStructuredClassifier::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	IUMLDataList 1013
documentation	A list of elements of type IUMLProperty 1251.		

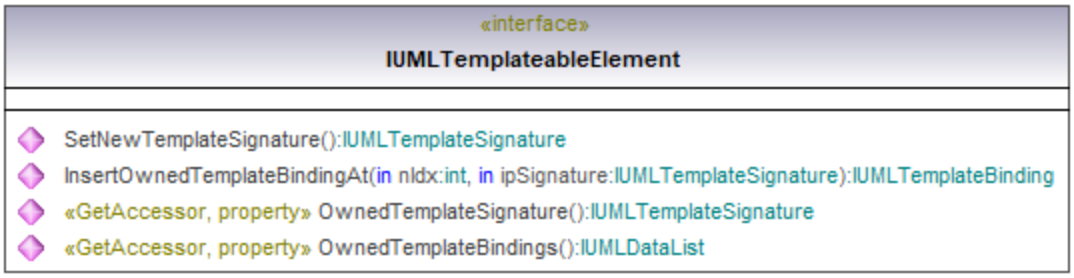
Operation **IUMLStructuredClassifier::OwnedConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLConnector 1140.					

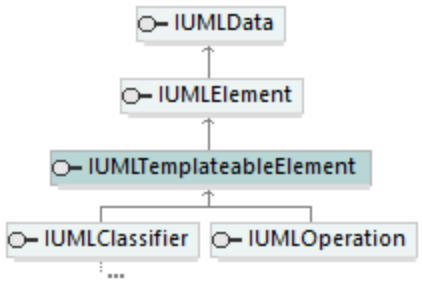
17.5.3.5.163 UModelAPI - IUMLTemplateableElement

Interface **IUMLTemplateableElement**

diagram



hierarchy



typedElements

Interface IUMLDataAll 1019
Interface IUMLTemplateBinding 1281
Interface IUMLTemplateSignature 1285
Operation BoundElement 1024 Template 1035
Operation BoundElement 1282
Operation Template 1286

Operation **IUMLTemplateableElement::InsertOwnedTemplateBindingAt**

 © 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSignature	**in**	**IUMLTemplateSignature** 1285			
	return	**return**	**IUMLTemplateBinding** 1281			

Operation **IUMLTemplateableElement::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLTemplateBinding 1281.					

Operation **IUMLTemplateableElement::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature** 1285			

Operation **IUMLTemplateableElement::SetNewTemplateSignature**

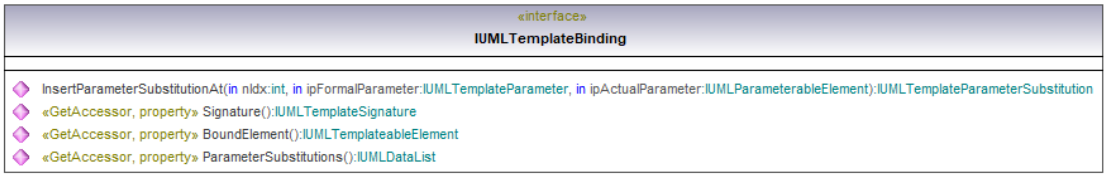
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature** 1285			

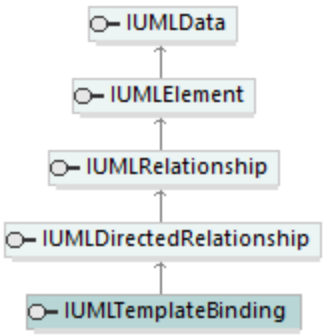
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.164 UModelAPI - IUMLTemplateBinding

Interface **IUMLTemplateBinding**

diagram



hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLTemplateableElement 1280 Interface IUMLTemplateParameterSubstitution 1284 Operation InsertOwnedTemplateBindingAt 1049 TemplateBinding 1085 Operation InsertOwnedTemplateBindingAt 1280 Operation TemplateBinding 1285

Operation **IUMLTemplateBinding::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateableElement 1280			

Operation **IUMLTemplateBinding::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFormalParameter	**in**	IUMLTemplateParameter 1283			
	ipActualParameter	**in**	IUMLParameterableElement 1246			
	return	**return**	IUMLTemplateParameterSubstitution 1284			

Operation **IUMLTemplateBinding::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLTemplateParameterSubstitution 1284.					

Operation **IUMLTemplateBinding::Signature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateSignature 1285			

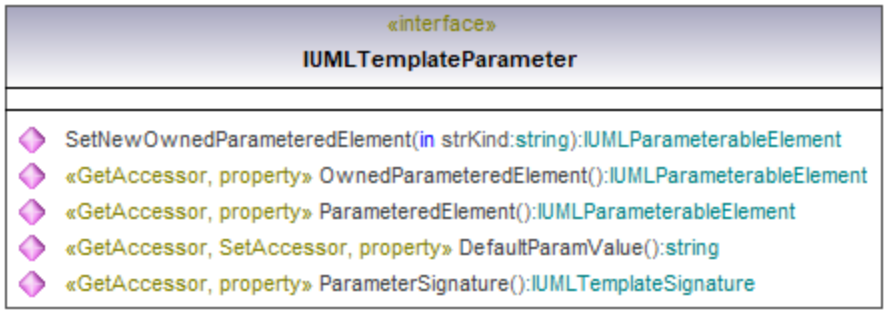
© 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

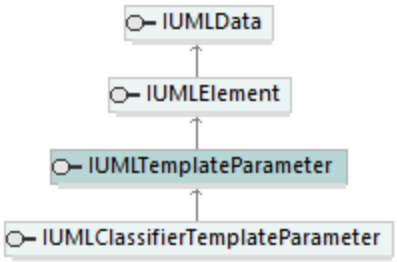
17.5.3.5.165 UModelAPI - IUMLTemplateParameter

Interface **IUMLTemplateParameter**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019

Interface **IUMLParameterableElement** 1246

Interface **IUMLTemplateBinding** 1281

Interface **IUMLTemplateParameterSubstitution** 1284

Operation **Formal** 1035

Operation **InsertParameterSubstitutionAt** 1050

Operation **OwningTemplateParameter** 1070

Operation **TemplateParameter** 1085

Operation **OwningTemplateParameter** 1246

Operation **TemplateParameter** 1246

Operation **InsertParameterSubstitutionAt** 1282

Operation **Formal** 1285

Operation **IUMLTemplateParameter::DefaultParamValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLTemplateParameter::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement** 1246			

Operation **IUMLTemplateParameter::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**IUMLParameterableElement** 1246

Operation **IUMLTemplateParameter::ParameterSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature** 1285			

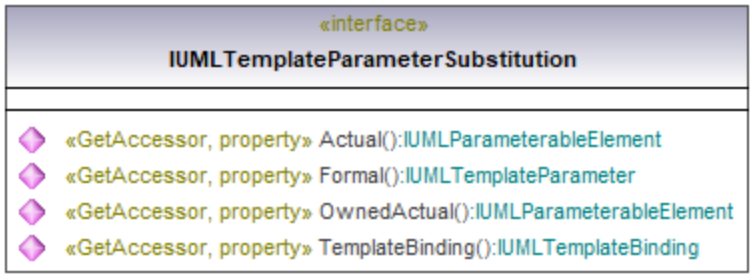
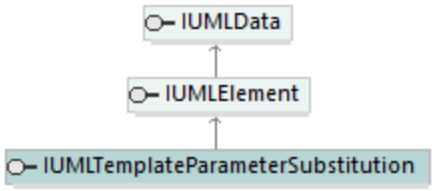
Operation **IUMLTemplateParameter::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLParameterableElement** 1246			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.166 UModelAPI - IUMLTemplateParameterSubstitution

Interface **IUMLTemplateParameterSubstitution**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLTemplateBinding** 1281 Operation **InsertParameterSubstitutionAt** 1050 Operation **InsertParameterSubstitutionAt** 1282

Operation **IUMLTemplateParameterSubstitution::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement** 1246			

© 2016-2022 Altova GmbH

Operation **IUMLTemplateParameterSubstitution::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateParameter 1283			

Operation **IUMLTemplateParameterSubstitution::OwnedActual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLParameterableElement 1246			

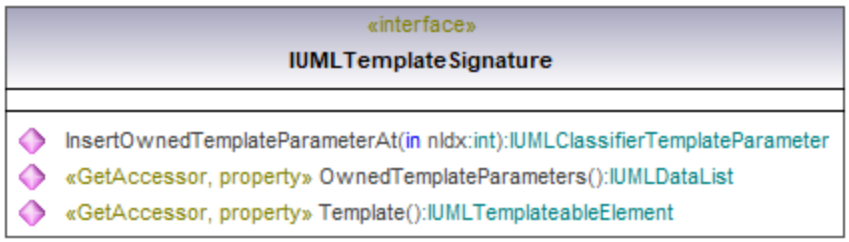
Operation **IUMLTemplateParameterSubstitution::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateBinding 1281			

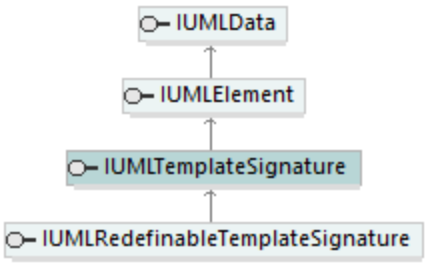
17.5.3.5.167 UModelAPI - IUMLTemplateSignature

Interface **IUMLTemplateSignature**

diagram



hierarchy



typedElements

Interface IUMLDataAll 1019

Operation InsertOwnedTemplateBindingAt 1049 OwnedTemplateSignature 1068 ParameterSignature 1070 SetNewTemplateSignature 1079 Signature 1082

	Interface IUMLTemplateableElement 1280 Interface IUMLTemplateBinding 1281 Interface IUMLTemplateParameter 1283	Operation InsertOwnedTemplateBindingAt 1280 OwnedTemplateSignature 1281 SetNewTemplateSignature 1281 Operation Signature 1282 Operation ParameterSignature 1284

Operation **IUMLTemplateSignature::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLClassifierTemplateParameter 1127			

Operation **IUMLTemplateSignature::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of elements of type IUMLTemplateParameter 1283.					

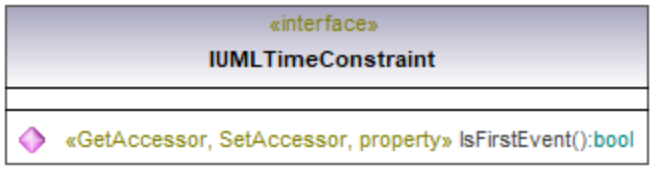
Operation **IUMLTemplateSignature::Template**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLTemplateableElement 1280			

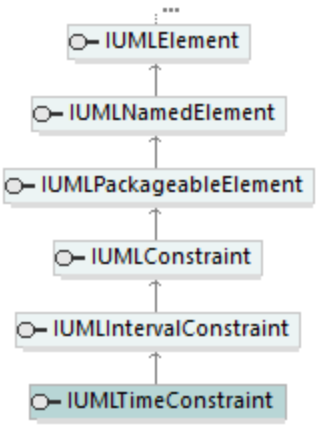
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.168 UModelAPI - IUMLTimeConstraint

Interface **IUMLTimeConstraint**

diagram	

 © 2016-2022 Altova GmbH

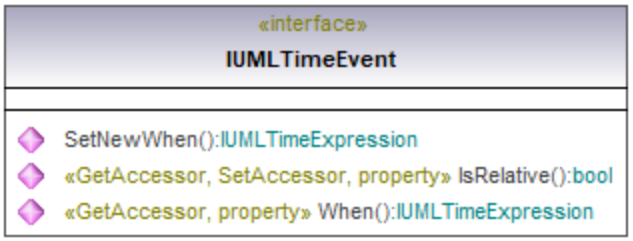
hierarchy	

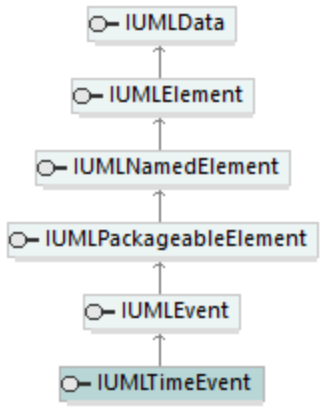
Operation **IUMLTimeConstraint::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

17.5.3.5.169 UModelAPI - IUMLTimeEvent

Interface **IUMLTimeEvent**

diagram	

hierarchy	

Operation **IUMLTimeEvent::IsRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLTimeEvent::SetNewWhen**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression** 1288			

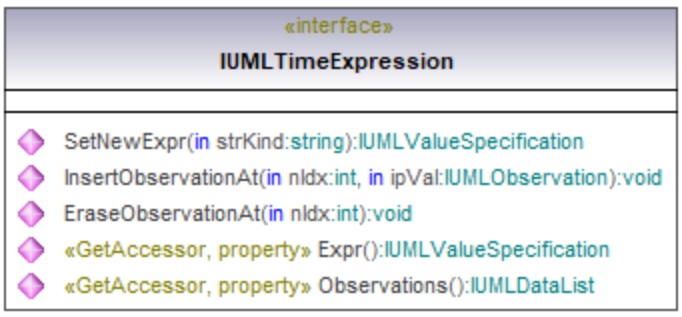
Operation **IUMLTimeEvent::When**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression** 1288			

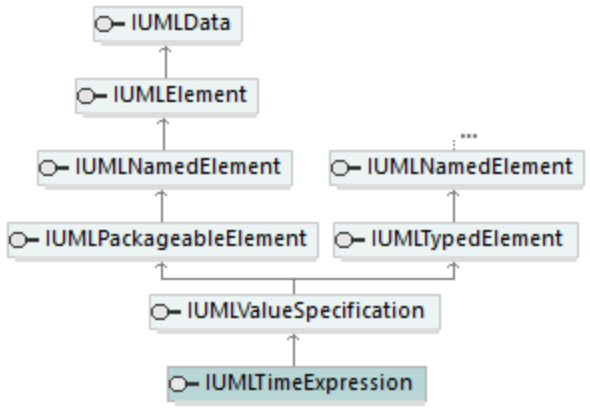
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.170 UModelAPI - IUMLTimeExpression

Interface **IUMLTimeExpression**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLTimeEvent 1287 Operation SetNewWhen 1079 When 1089 Operation SetNewWhen 1288 When 1288

Operation **IUMLTimeExpression::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLTimeExpression::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLTimeExpression::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation** 1231			
	return	**return**	**void**			

Operation **IUMLTimeExpression::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLObservation 1231.					

Operation **IUMLTimeExpression::SetNewExpr**

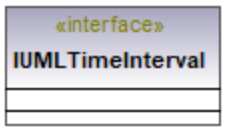
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

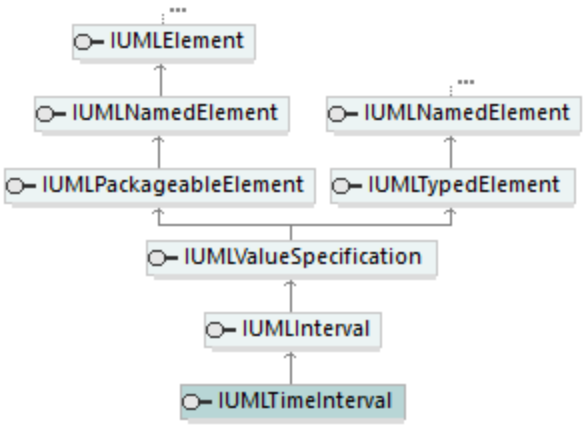
17.5.3.5.171 UModelAPI - IUMLTimeInterval

Interface **IUMLTimeInterval**

diagram



hierarchy

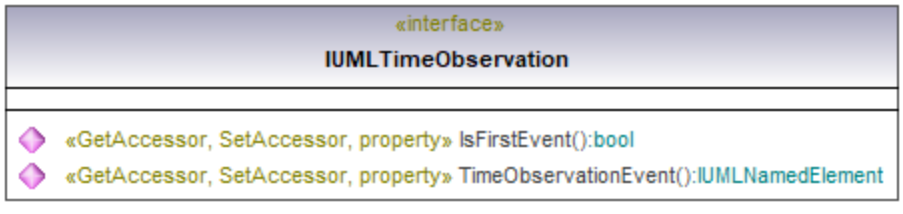


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

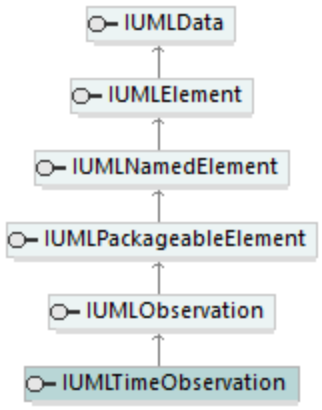
17.5.3.5.172 UModelAPI - IUMLTimeObservation

Interface **IUMLTimeObservation**

diagram



 © 2016-2022 Altova GmbH

hierarchy	

Operation **IUMLTimeObservation::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

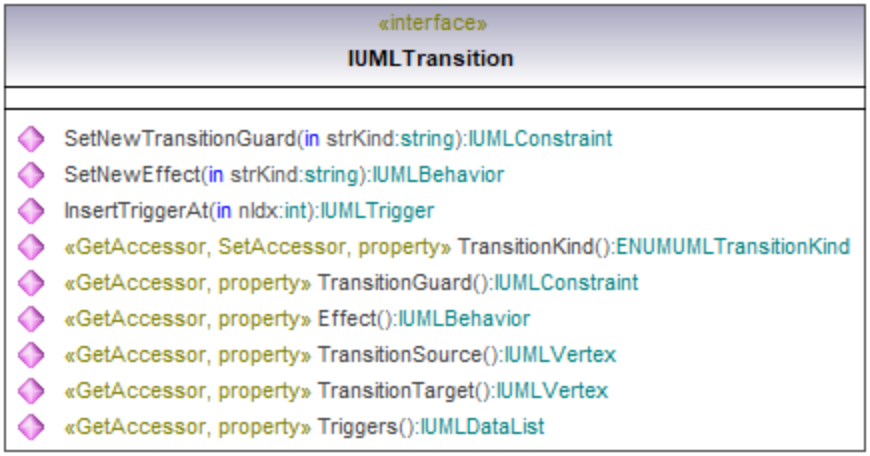
Operation **IUMLTimeObservation::TimeObservationEvent**

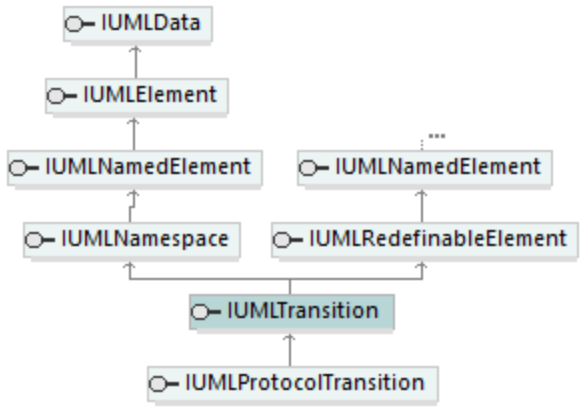
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLNamedElement 1222			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.173 UModelAPI - IUMLTransition

Interface **IUMLTransition**

diagram	

hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLRegion 1261 Operation InsertTransitionAt 1052 Operation InsertTransitionAt 1261

Operation **IUMLTransition::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLBehavior 1110			

Operation **IUMLTransition::InsertTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLTrigger 1293			

Operation **IUMLTransition::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	IUMLBehavior 1110			

Operation **IUMLTransition::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	IUMLConstraint 1142			

Operation **IUMLTransition::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLConstraint 1142			

Operation **IUMLTransition::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**ENUMUMLTransitionKind** 1377

Operation **IUMLTransition::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex** 1303			

Operation **IUMLTransition::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex** 1303			

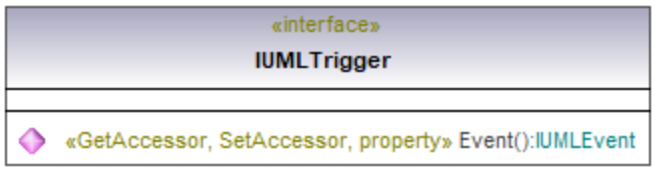
Operation **IUMLTransition::Triggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLTrigger 1293.					

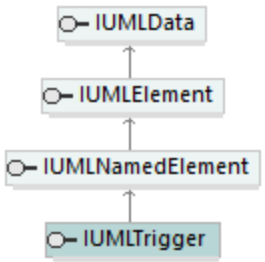
17.5.3.5.174 UModelAPI - IUMLTrigger

Interface **IUMLTrigger**

diagram



hierarchy



typedElements

Interface **IUMLAcceptEventAction** 1091
Interface **IUMLDataAll** 1019
Operation **InsertActionTriggerAt** 1091
Operation **InsertActionTriggerAt** 1039
InsertTriggerAt 1052

	Interface **IUMLTransition** 1291	Operation **InsertTriggerAt** 1292

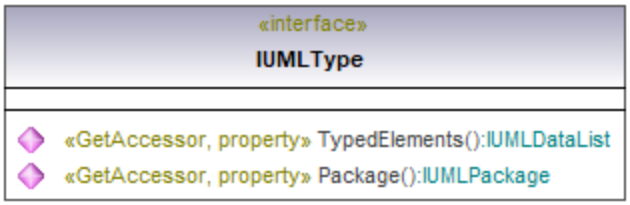
Operation **IUMLTrigger::Event**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent** 1165			

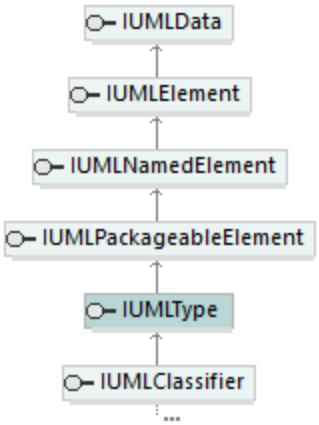
17.5.3.5.175 UModelAPI - IUMLType

Interface **IUMLType**

diagram



hierarchy



typedElements

Interface **IUMLBehavioralFeature** 1112	Operation **InsertRaisedExceptionAt** 1113
Interface **IUMLDataAll** 1019	Operation **InsertRaisedExceptionAt** 1051
	Type 1087
Interface **IUMLOperation** 1236	Operation **Type** 1237
Interface **IUMLTypedElement** 1295	Operation **Type** 1295

Operation **IUMLType::Package**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage** 1238			

 © 2016-2022 Altova GmbH

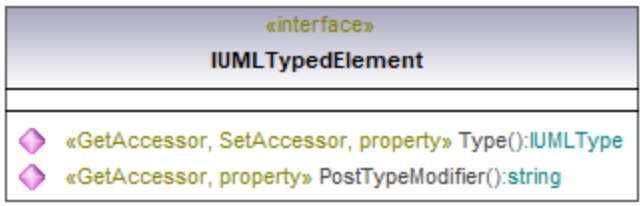
Operation **IUMLType::TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLTypedElement 1295.					

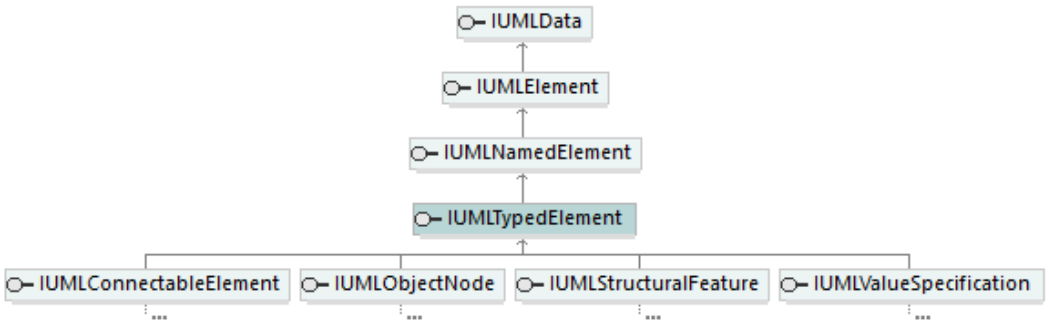
17.5.3.5.176 UModelAPI - IUMLTypedElement

Interface **IUMLTypedElement**

diagram



hierarchy



Operation **IUMLTypedElement::PostTypeModifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

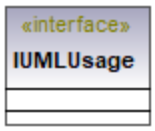
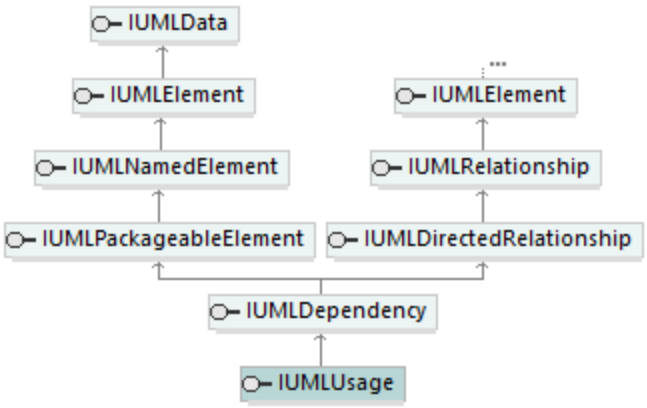
Operation **IUMLTypedElement::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType** 1294			

© 2016-2022 Altova GmbH

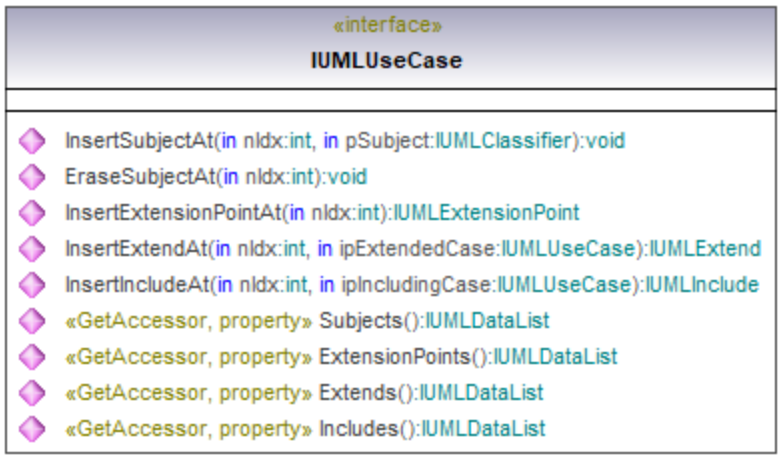
17.5.3.5.177 UModelAPI - IUMLUsage

Interface **IUMLUsage**

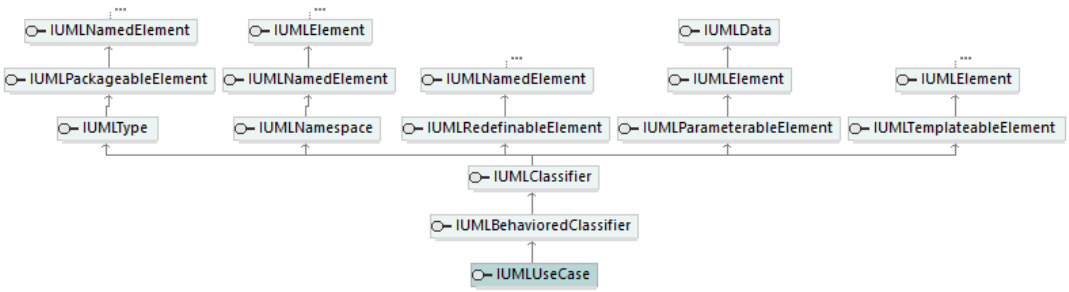
diagram	
hierarchy	

17.5.3.5.178 UModelAPI - IUMLUseCase

Interface **IUMLUseCase**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLClassifier** 1124 Interface **IUMLDataAll** 1019 Interface **IUMLExtend** 1173 Interface **IUMLExtensionPoint** 1175 Interface **IUMLInclude** 1185 Interface **IUMLUseCase** 1296 Operation **InsertOwnedUseCaseAt** 1126 Operation **Addition** 1020 **ExtendedCase** 1033 **Extension** 1034 **IncludingCase** 1038 **InsertExtendAt** 1043 **InsertIncludeAt** 1044 **InsertOwnedUseCaseAt** 1049 **UseCase** 1087 Operation **ExtendedCase** 1174 **Extension** 1174 Operation **UseCase** 1175 Operation **Addition** 1185 **IncludingCase** 1185 Operation **InsertExtendAt** 1297 **InsertIncludeAt** 1298

Operation **IUMLUseCase::EraseSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLUseCase::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLExtend 1173.					

Operation **IUMLUseCase::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLExtensionPoint 1175.					

Operation **IUMLUseCase::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLInclude 1185.					

Operation **IUMLUseCase::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipExtendedCase	**in**	**IUMLUseCase** 1296			
	return	**return**	**IUMLExtend** 1173			

Operation **IUMLUseCase::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExtensionPoint** 1175			

Operation **IUMLUseCase::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipIncludingCase	**in**	**IUMLUseCase** 1296			
	return	**return**	**IUMLInclude** 1185			

Operation **IUMLUseCase::InsertSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pSubject	**in**	**IUMLClassifier** 1124			
	return	**return**	**void**			

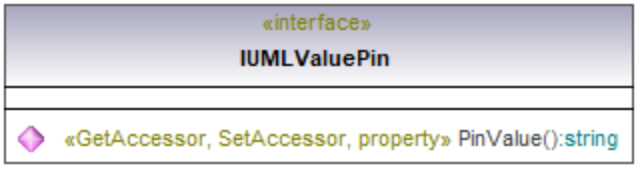
Operation **IUMLUseCase::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLClassifier 1124.					

17.5.3.5.179 UModelAPI - IUMLValuePin

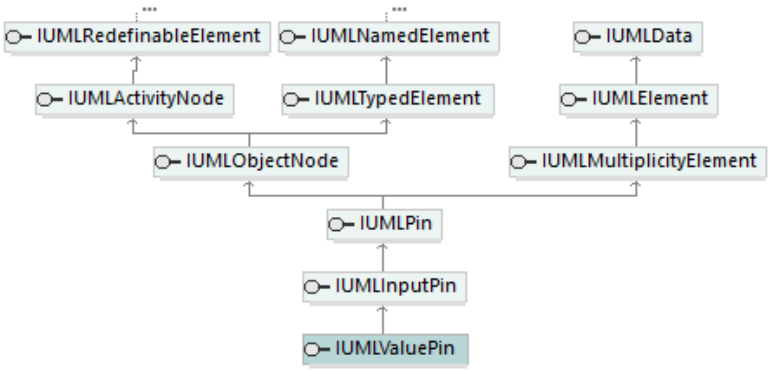
Interface **IUMLValuePin**

diagram	



 © 2016-2022 Altova GmbH

hierarchy



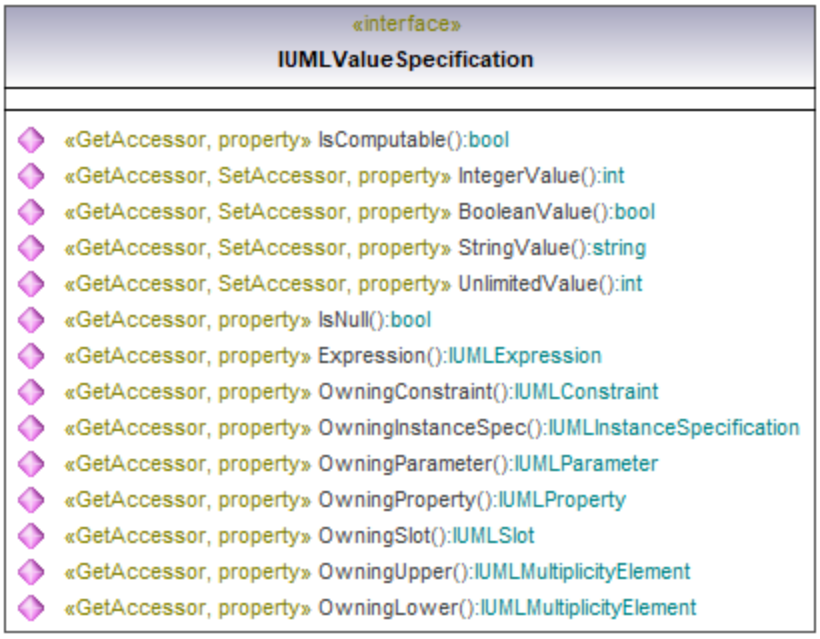
Operation **IUMLValuePin::PinValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

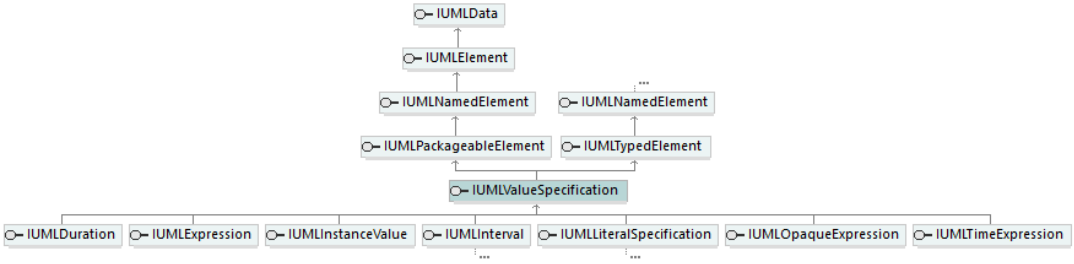
17.5.3.5.180 UModelAPI - IUMLValueSpecification

Interface **IUMLValueSpecification**

diagram



hierarchy



typedElements

Interface **IUMLChangeEvent** 1120
Operation **ChangeExpression** 1121
SetNewChangeExpression 1121

Interface **IUMLConstraint** 1142
Operation **SetNewSpecification** 1143
Specification 1144

Interface **IUMLDataAll** 1019
Operation **ActionValue** 1019
ChangeExpression 1024
DefaultValue 1028 **Expr** 1033
InsertOwnedArgumentAt 1047
InsertValueAt 1053 **Max** 1061
MaxInt 1061 **Min** 1062 **MinInt** 1063
Selector 1074
SetNewActionValue 1076
SetNewChangeExpression 1076
SetNewDefaultValue 1076
SetNewExpr 1077 **SetNewMax** 1077
SetNewMaxInt 1077 **SetNewMin** 1078
SetNewMinInt 1078
SetNewSelector 1078
SetNewSpecification 1079
SetPredefinedTaggedValueAt 1080
SetSlotValueAt 1080
SetTaggedValueAt 1081
Specification 1082

Interface **IUMLDuration** 1154
Operation **Expr** 1154 **SetNewExpr** 1155

Interface **IUMLInstanceSpecification** 1190
Operation **SetNewSpecification** 1191
SetSlotValueAt 1192
Specification 1192

Interface **IUMLInteractionConstraint** 1195
Operation **MaxInt** 1196 **MinInt** 1196
SetNewMaxInt 1196
SetNewMinInt 1196

Interface **IUMLInterval** 1205
Operation **Max** 1205 **Min** 1205 **SetNewMax** 1206
SetNewMin 1206

Interface **IUMLLifeline** 1209
Operation **Selector** 1210 **SetNewSelector** 1210

Interface **IUMLMessage** 1216
Operation **InsertOwnedArgumentAt** 1217

Interface **IUMLParameter** 1244
Operation **DefaultValue** 1245
SetNewDefaultValue 1245

Interface **IUMLProperty** 1251
Operation **DefaultValue** 1253
SetNewDefaultValue 1254

Interface **IUMLSlot** 1266
Operation **InsertValueAt** 1267

Interface **IUMLStereotypeApplication** 1275
Operation **SetPredefinedTaggedValueAt** 1275
SetTaggedValueAt 1276

Interface **IUMLTimeExpression** 1288
Operation **Expr** 1289 **SetNewExpr** 1289

Interface **IUMLValueSpecificationAction** 1302
Operation **ActionValue** 1303
SetNewActionValue 1303

© 2016-2022 Altova GmbH

Operation **IUMLValueSpecification::BooleanValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLValueSpecification::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpression** 1172			

Operation **IUMLValueSpecification::IntegerValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLValueSpecification::IsComputable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLValueSpecification::IsNull**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLValueSpecification::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint** 1142			

Operation **IUMLValueSpecification::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification** 1190			

Operation **IUMLValueSpecification::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement** 1221			

Operation **IUMLValueSpecification::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter** 1244			

Operation **IUMLValueSpecification::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty** 1251			

Operation **IUMLValueSpecification::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSlot** 1266			

Operation **IUMLValueSpecification::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement** 1221			

Operation **IUMLValueSpecification::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLValueSpecification::UnlimitedValue**

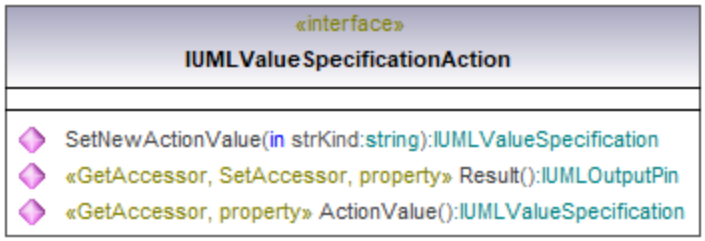
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

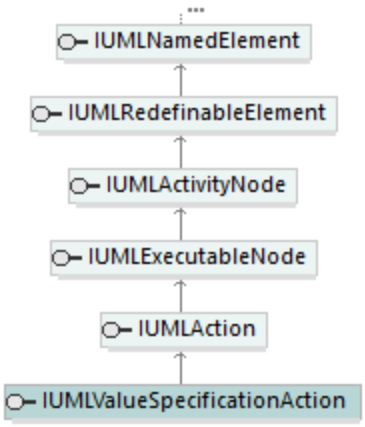
17.5.3.5.181 UModelAPI - IUMLValueSpecificationAction

Interface **IUMLValueSpecificationAction**

diagram



 © 2016-2022 Altova GmbH

hierarchy	

Operation **IUMLValueSpecificationAction::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification** 1299			

Operation **IUMLValueSpecificationAction::Result**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOutputPin** 1237			

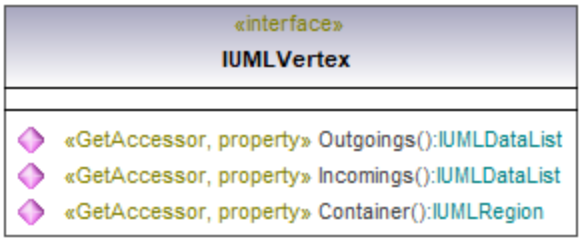
Operation **IUMLValueSpecificationAction::SetNewActionValue**

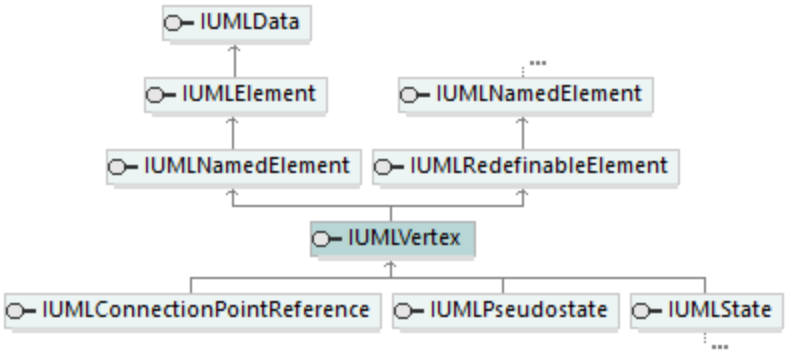
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification** 1299			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.5.182 UModelAPI - IUMLVertex

Interface **IUMLVertex**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLRegion** 1261 Interface **IUMLTransition** 1291 Operation **InsertSubVertexAt** 1052 **InsertTransitionAt** 1052 **TransitionSource** 1086 **TransitionTarget** 1087 Operation **InsertSubVertexAt** 1261 **InsertTransitionAt** 1261 Operation **TransitionSource** 1293 **TransitionTarget** 1293

Operation **IUMLVertex::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLRegion** 1261			

Operation **IUMLVertex::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLTransition 1291.					

Operation **IUMLVertex::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLTransition 1291.					

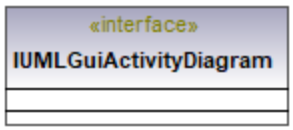
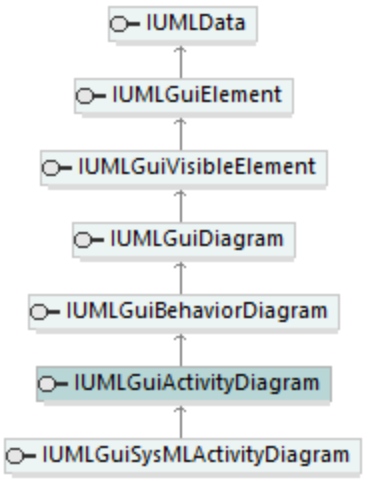
17.5.3.6 IUMLGuiElement

Dies ist eine Liste von Altova-Elementen für Diagramme und zur Anzeige von IUMLElements 1090 in Diagrammen.

 © 2016-2022 Altova GmbH

17.5.3.6.1 UModelAPI - IUMLGuiActivityDiagram

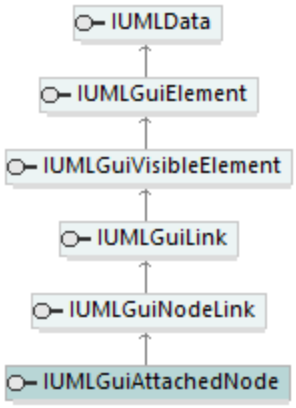
Interface **IUMLGuiActivityDiagram**

diagram	
hierarchy	

17.5.3.6.2 UModelAPI - IUMLGuiAttachedNode

Interface **IUMLGuiAttachedNode**

diagram	

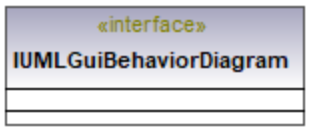
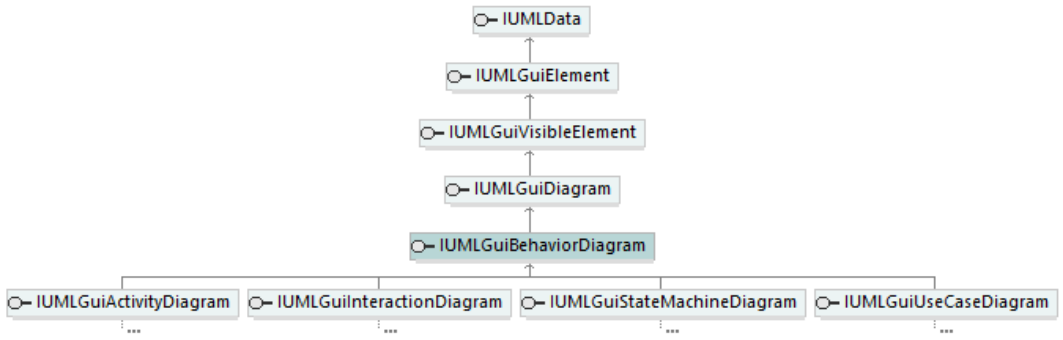
hierarchy	
documentation	This GUI element is a node (possibly without a linked IUMLElement[1157]) which is directly attached to a IUMLGuiNodeLink[1331]. It disappears and pops up based on data set in the element of the IUMLGuiNodeLink[1331] it is attached to. The user usually only has control of this element via styles or the node it is attached to. This node is used as graphical object on diagrams to represent Tagged Values for example.

Operation **IUMLGuiAttachedNode::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**[1328]			

17.5.3.6.3 UModelAPI - IUMLGuiBehaviorDiagram

Interface **IUMLGuiBehaviorDiagram**

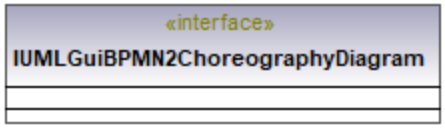
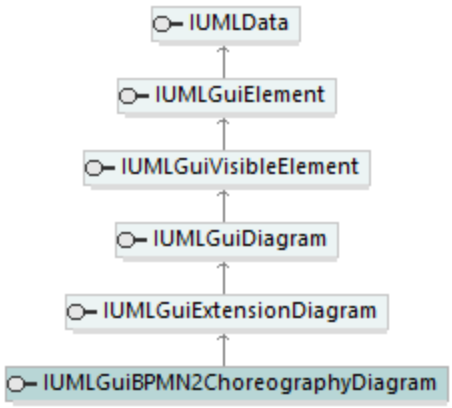
diagram	
hierarchy	

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.4 UModelAPI - IUMLGuiBPMN2ChoreographyDiagram

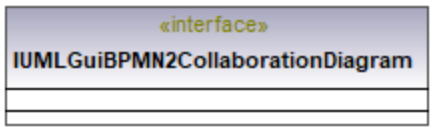
Interface **IUMLGuiBPMN2ChoreographyDiagram**

diagram	
hierarchy	

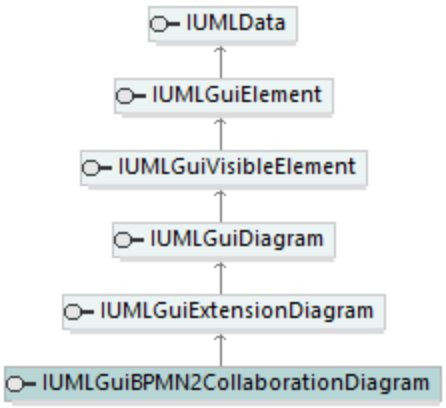
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.5 UModelAPI - IUMLGuiBPMN2CollaborationDiagram

Interface **IUMLGuiBPMN2CollaborationDiagram**

diagram	

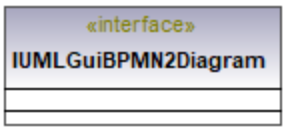
hierarchy



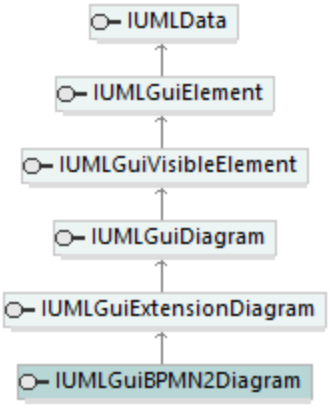
17.5.3.6.6 UModelAPI - IUMLGuiBPMN2Diagram

Interface **IUMLGuiBPMN2Diagram**

diagram



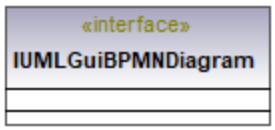
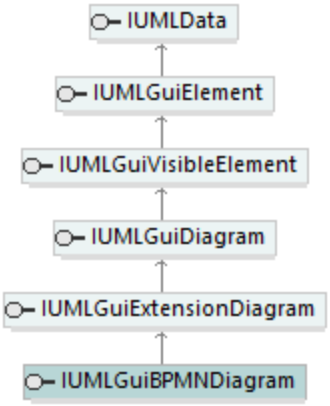
hierarchy



 © 2016-2022 Altova GmbH

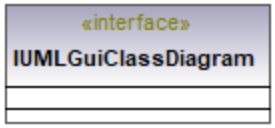
17.5.3.6.7 UModelAPI - IUMLGuiBPMNDiagram

Interface **IUMLGuiBPMNDiagram**

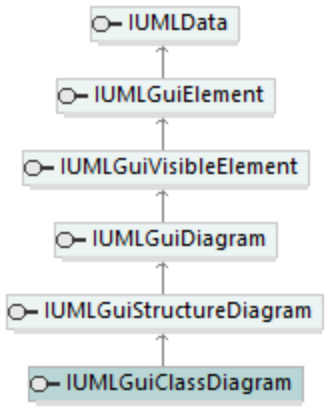
diagram	
hierarchy	

17.5.3.6.8 UModelAPI - IUMLGuiClassDiagram

Interface **IUMLGuiClassDiagram**

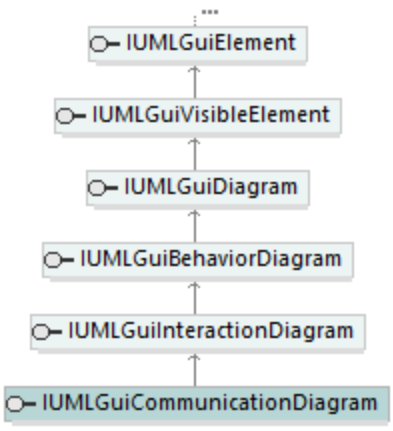
diagram	

© 2016-2022 Altova GmbH

hierarchy	

17.5.3.6.9 UModelAPI - IUMLGuiCommunicationDiagram

Interface **IUMLGuiCommunicationDiagram**

diagram	
hierarchy	

Operation **IUMLGuiCommunicationDiagram::AddUMLGuiCommunicationLink**

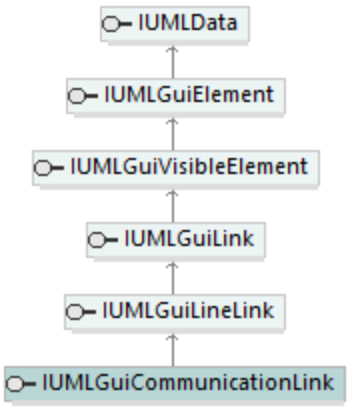
parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	IUMLGuiLink 1328			

 © 2016-2022 Altova GmbH

	ipToLink **return**	**in** **return**	IUMLGuiLink 1328 IUMLGuiCommunicationLink 1311

17.5.3.6.10 UModelAPI - IUMLGuiCommunicationLink

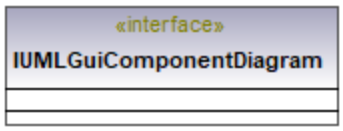
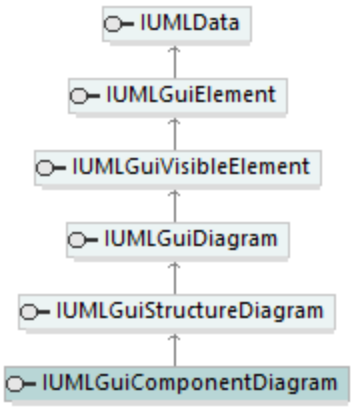
Interface **IUMLGuiCommunicationLink**

diagram	
hierarchy	
typedElements	Interface IUMLGuiCommunicationDiagram 1310 Operation AddUMLGuiCommunicationLink 1310
documentation	This line link is used on communication diagrams to provide a connection link for messages between lifelines.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.11 UModelAPI - IUMLGuiComponentDiagram

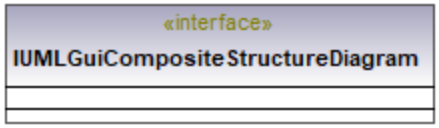
Interface **IUMLGuiComponentDiagram**

diagram	
hierarchy	

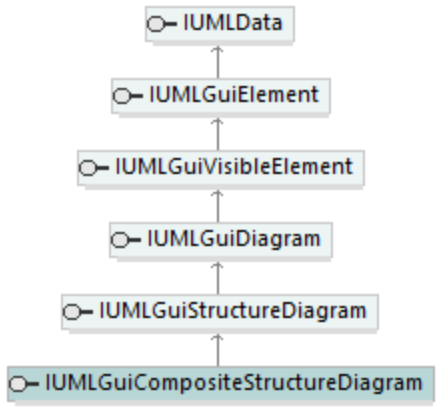
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.12 UModelAPI - IUMLGuiCompositeStructureDiagram

Interface **IUMLGuiCompositeStructureDiagram**

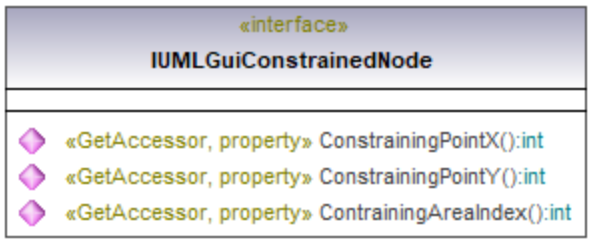
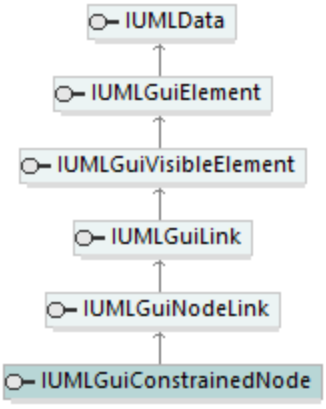
diagram	

© 2016-2022 Altova GmbH

hierarchy	

17.5.3.6.13 UModelAPI - IUMLGuiConstrainedNode

Interface **IUMLGuiConstrainedNode**

diagram	
hierarchy	
documentation	This node link is used to represent objects on diagrams w hich can be directly attached to a parent node link and placed relatively to and in special constraining areas of the parent. Used for example for Pins on Activtiy Diagrams.

Operation **IUMLGuiConstrainedNode::ConstrainingPointX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	X coordinate relative to the uppor left position of the contraining area.					

Operation **IUMLGuiConstrainedNode::ConstrainingPointY**

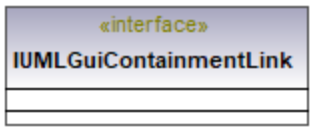
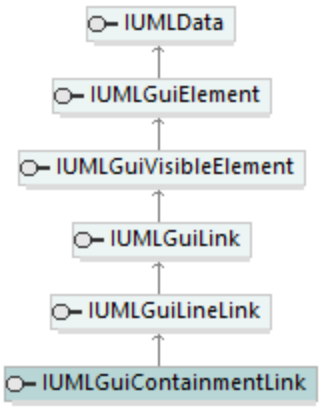
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	Y coordinate relative to the uppor left position of the contraining area.					

Operation **IUMLGuiConstrainedNode::ContrainingAreaIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	Defines the index of the area w here this node is currently in and to w hich its relative position has its origin.					

17.5.3.6.14 UModelAPI - IUMLGuiContainmentLink

Interface **IUMLGuiContainmentLink**

diagram	
hierarchy	

 © 2016-2022 Altova GmbH

typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiDiagram** 1316	Operation **AddUMLGuiContainmentLink** 1020 Operation **AddUMLGuiContainmentLink** 1317

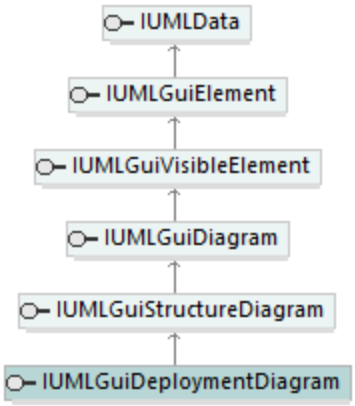
17.5.3.6.15 UModelAPI - IUMLGuiDeploymentDiagram

Interface **IUMLGuiDeploymentDiagram**

diagram



hierarchy

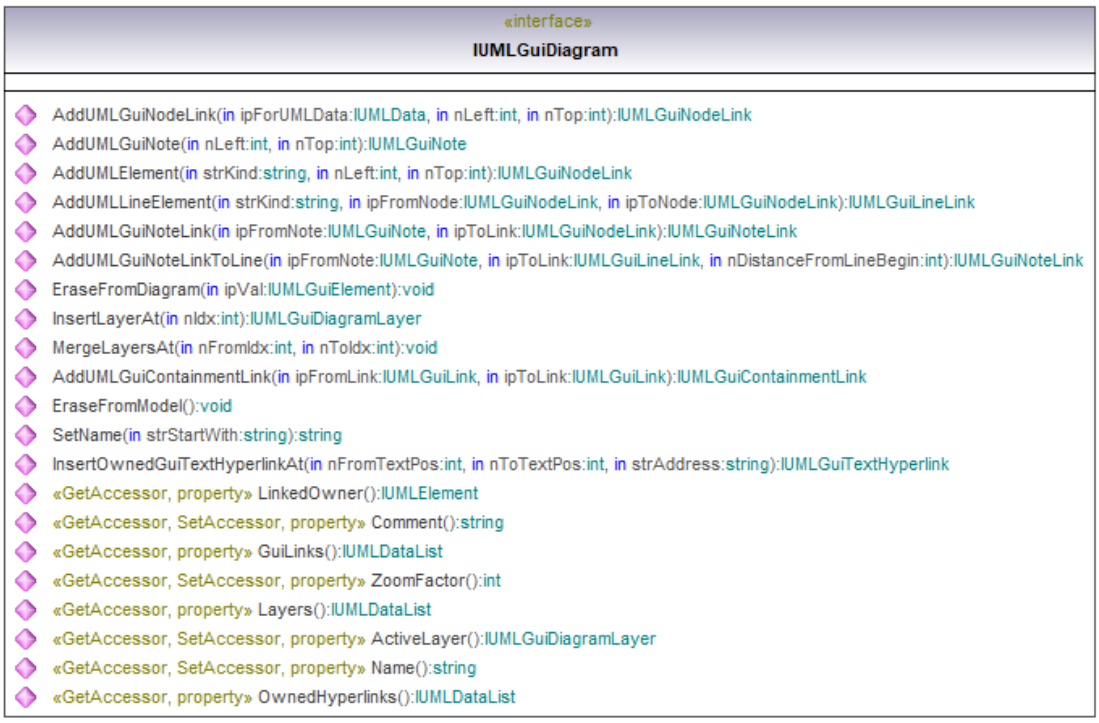


© 2016-2022 Altova GmbH

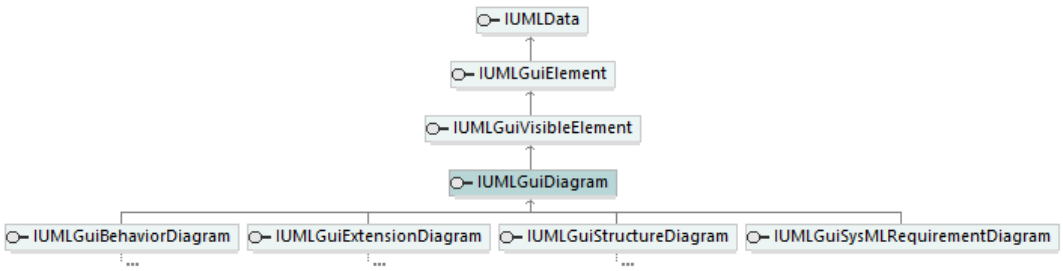
17.5.3.6.16 UModelAPI - IUMLGuiDiagram

Interface **IUMLGuiDiagram**

diagram



hierarchy



typedElements

Interface **IDiagramWindow** 931
Interface **IDocument** 938
Interface **IUMLDataAll** 1019
Interface **IUMLGuiRootElement** 1338
Interface **IUMLGuiSubDiagramNode** 1347
Operation **Diagram** 932
Operation **OpenDiagram** 943
Operation **InsertOwnedDiagramAt** 1048
ReferencedDiagram 1073
Operation **InsertOwnedDiagramAt** 1339
Operation **ReferencedDiagram** 1348

documentation

Represents an UML diagram and contains all layers, nodes (represented as IUMLGuiNodeLink 1331) and lines (represented as IUMLGuiLineLink 1326).Use the property GuiLinks 1318 to access these.

Operation **IUMLGuiDiagram::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	return	**return**	**IUMLGuiDiagramLayer** 1320

Operation **IUMLGuiDiagram::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink** 1331			
documentation	Adds a new UML element (e.g. IUMLClass 1121, IUMLPackage 1238,...) to the model and shows it with a new IUMLGuiNodeLink 1331 on the diagram.					

Operation **IUMLGuiDiagram::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink** 1328			
	ipToLink	**in**	**IUMLGuiLink** 1328			
	return	**return**	**IUMLGuiContainmentLink** 1314			

Operation **IUMLGuiDiagram::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLData** 1011			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink** 1331			
documentation	Adds a new IUMLGuiNodeLink 1331 for an existing UML element (e.g. IUMLClass 1121, IUMLPackage 1238,...) on the diagram.					

Operation **IUMLGuiDiagram::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNote** 1333			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote** 1333			
	ipToLink	**in**	**IUMLGuiNodeLink** 1331			
	return	**return**	**IUMLGuiNoteLink** 1334			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote** 1333			
	ipToLink	**in**	**IUMLGuiLineLink** 1326			

	nDistanceFromLineBegin		**int**
	return	**return**	**IUMLGuiNoteLink** 1334

Operation **IUMLGuiDiagram::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	ipFromNode	**in**	**IUMLGuiNodeLink** 1331			
	ipToNode	**in**	**IUMLGuiNodeLink** 1331			
	return	**return**	**IUMLGuiLineLink** 1326			
documentation	Adds a new UML line element (e.g. IUMLGeneralization 1180, IUMLAssociation 1108,...) to the model and shows it with a new IUMLGuiLineLink 1326 on the diagram.					

Operation **IUMLGuiDiagram::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiDiagram::EraseFromDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLGuiElement** 1321			
	return	**return**	**void**			
documentation	Use this function to erase the element from the diagram only. Use IUMLElement 1157::EraseFromModel 1158 to erase from the model and all diagrams.					

Operation **IUMLGuiDiagram::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLGuiDiagram::GuiLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of elements of type IUMLGuiLink 1328 which are displayed directly on this diagram. Usually, these are IUMLGuiNodeLink 1331s and IUMLGuiLineLink 1326s.					

Operation **IUMLGuiDiagram::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiDiagramLayer** 1320			

Operation **IUMLGuiDiagram::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default

© 2016-2022 Altova GmbH

	nFromTextPos	**in**	**int**
	nToTextPos	**in**	**int**
	strAddress	**in**	**string**
	return	**return**	**IUMLGuiTextHyperlink** 1355

Operation **IUMLGuiDiagram::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A list of all layers in the diagam. The list contains elements of type IUMLGuiDiagramLayer 1320.					

Operation **IUMLGuiDiagram::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

Operation **IUMLGuiDiagram::MergeLayersAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromIdx	**in**	**int**			
	nToIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiDiagram::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiDiagram::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

Operation **IUMLGuiDiagram::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

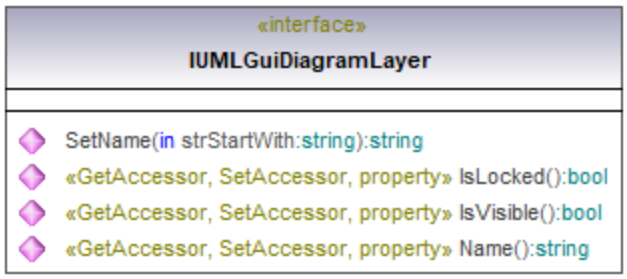
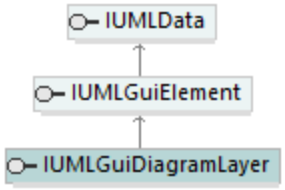
Operation **IUMLGuiDiagram::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

© 2016-2022 Altova GmbH

17.5.3.6.17 UModelAPI - IUMLGuiDiagramLayer

Interface **IUMLGuiDiagramLayer**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLGuiDiagram 1316 Interface IUMLGuiLink 1328 Operation ActiveLayer 1019 InsertLayerAt 1045 Layer 1059 Operation ActiveLayer 1316 InsertLayerAt 1318 Operation Layer 1329
documentation	Represents a layer on an IUMLGuiDiagram 1316. Makes it possible to group elements on a diagram into categories, to lock/unlock them and to make them visible or invisible.

Operation **IUMLGuiDiagramLayer::IsLocked**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiDiagramLayer::IsVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiDiagramLayer::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiDiagramLayer::SetName**

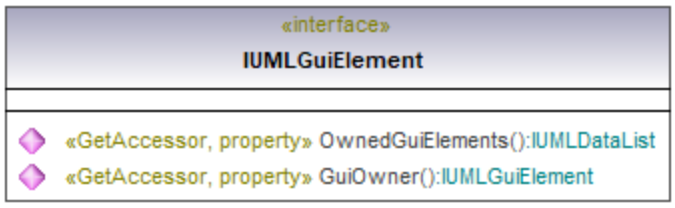
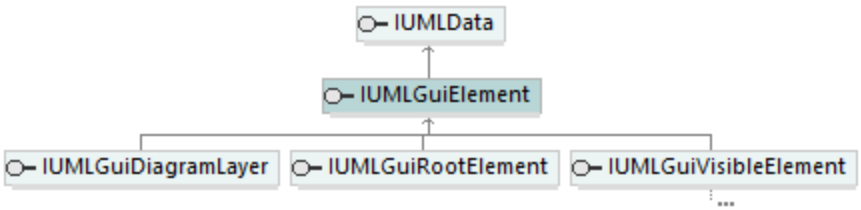
parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

© 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.18 UModelAPI - IUMLGuiElement

Interface **IUMLGuiElement**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiDiagram** 1316 Interface **IUMLGuiElement** 1321 Interface **IUMLGuiLineLink** 1326 Operation **EraseFromDiagram** 1031 **GuiOwner** 1037 **LineBegin** 1060 **LineEnd** 1060 Operation **EraseFromDiagram** 1318 Operation **GuiOwner** 1321 Operation **LineBegin** 1327 **LineEnd** 1328
documentation	The base class for all graphical objects.

Operation **IUMLGuiElement::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement** 1321			

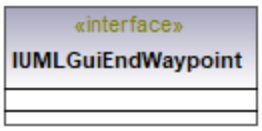
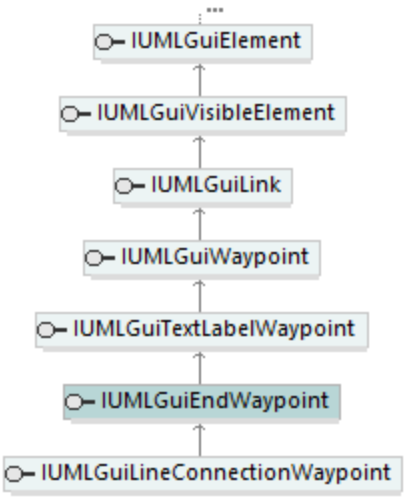
Operation **IUMLGuiElement::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	Returns a derived list of all owned Gui elements. All elements in this list are a subtype if IUMLGuiElement 1321.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

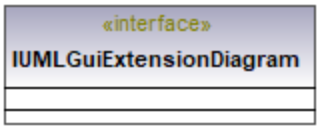
17.5.3.6.19 UModelAPI - IUMLGuiEndWaypoint

Interface **IUMLGuiEndWaypoint**

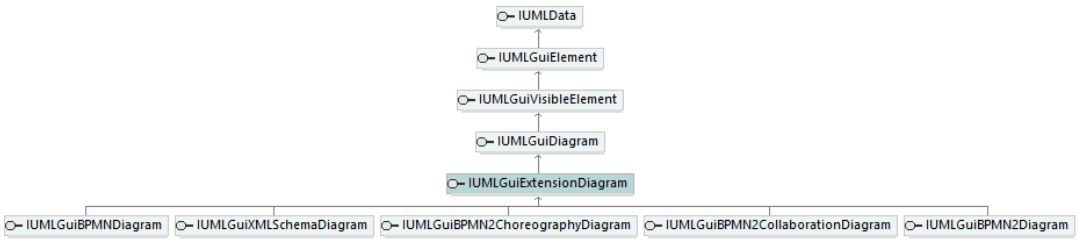
diagram	
hierarchy	
documentation	A special waypoint which only occurrs at the end or the begin of a line represented by a IUMLGuiLineLink 1326.

17.5.3.6.20 UModelAPI - IUMLGuiExtensionDiagram

Interface **IUMLGuiExtensionDiagram**

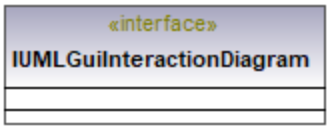
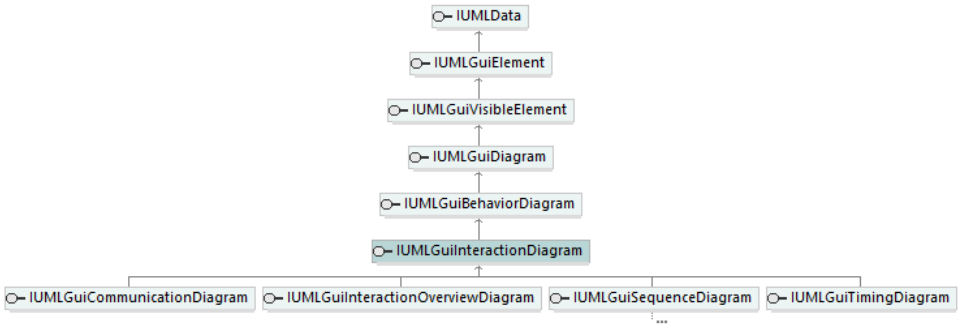
diagram	

 © 2016-2022 Altova GmbH

hierarchy	
documentation	This diagram type is the base for all UModel specific extension diagrams (for example BPMN diagrams, XML Schema Diagrams) and not a diagram type for itself.

17.5.3.6.21 UModelAPI - IUMLGuiInteractionDiagram

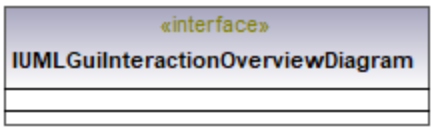
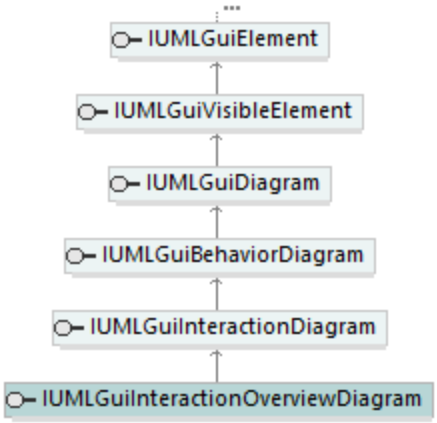
Interface **IUMLGuiInteractionDiagram**

diagram	
hierarchy	

© 2016-2022 Altova GmbH

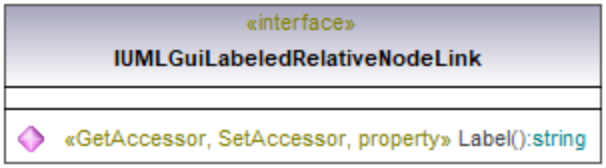
17.5.3.6.22 UModelAPI - IUMLGuiInteractionOverviewDiagram

Interface **IUMLGuiInteractionOverviewDiagram**

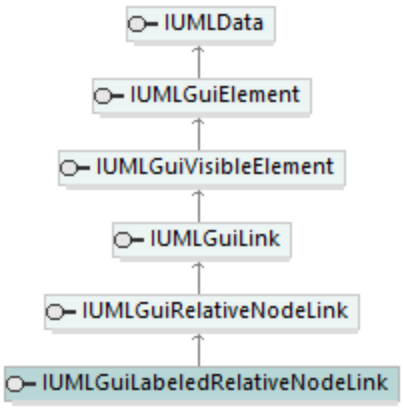
diagram	
hierarchy	

17.5.3.6.23 UModelAPI - IUMLGuiLabeledRelativeNodeLink

Interface **IUMLGuiLabeledRelativeNodeLink**

diagram	

 © 2016-2022 Altova GmbH

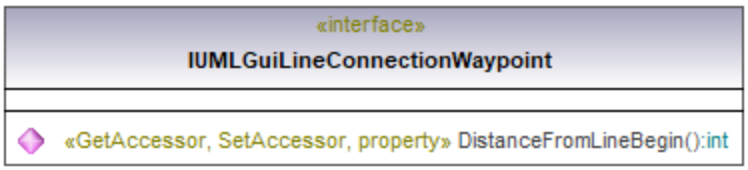
hierarchy	
documentation	This special gui link is used for elements which are relative to another node and have a label, for example for the names of Messages on Communication diagrams.

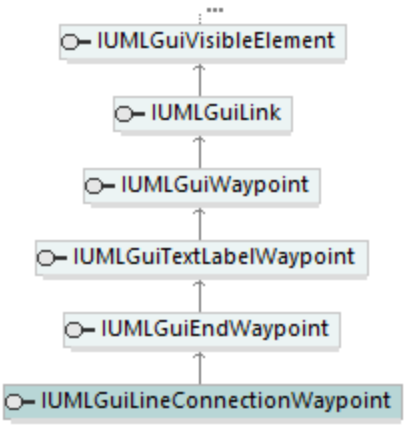
Operation **IUMLGuiLabeledRelativeNodeLink::Label**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

17.5.3.6.24 UModelAPI - IUMLGuiLineConnectionWaypoint

Interface **IUMLGuiLineConnectionWaypoint**

diagram	

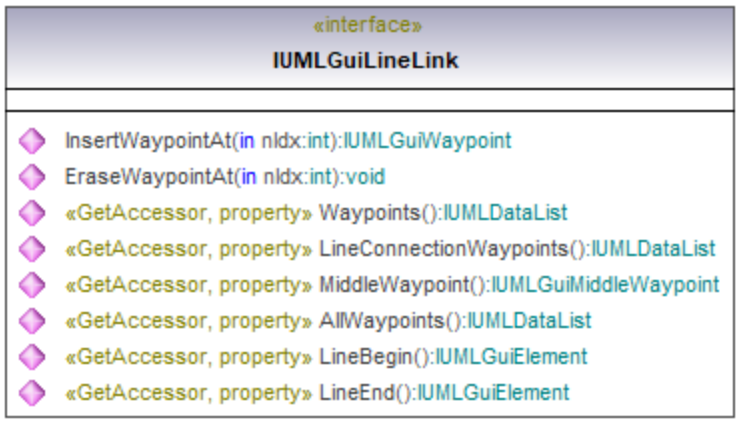
hierarchy	
documentation	This special waypoint marks the part of a line where it is connected to another line. For example, when drawing a noteLink from a note to a line on a diagram in UModel, a waypoint of this type is created where the noteLink connects to the target line. Using the DistanceFromLineBegin 1326 property, the waypoint sets a floating fixed position for itself on the line.

Operation **IUMLGuiLineConnectionWaypoint::DistanceFromLineBegin**

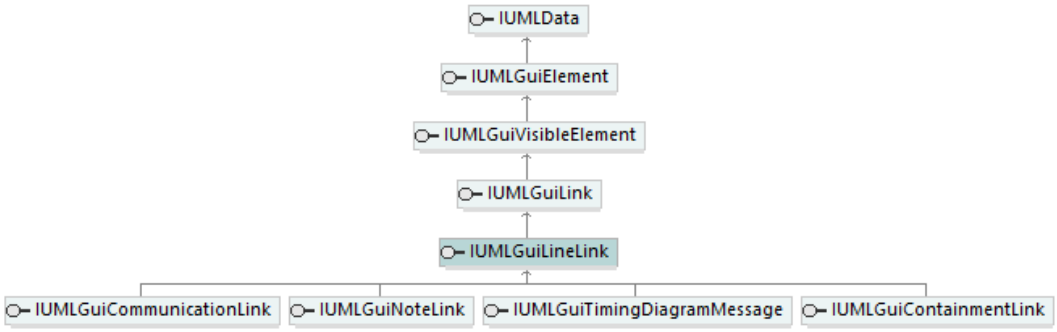
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

17.5.3.6.25 UModelAPI - IUMLGuiLineLink

Interface **IUMLGuiLineLink**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiDiagram** 1316 Operation **AddUMLGuiNoteLinkToLine** 1021 **AddUMLLineElement** 1021 Operation **AddUMLGuiNoteLinkToLine** 1317 **AddUMLLineElement** 1318
documentation	This interface represents a line on a diagram. There are some special lines deriving from this interface available as well. A line is composed of multiple but at least 2 waypoints which are connected to each other, which can be accessed using the AllWaypoints 1327 property. Two of these waypoints are usually of type IUMLGuiEndWaypoint 1322. There may be also a Middlewaypoint accessible with the property MiddleWaypoint 1328 which can be used for example to access textlabels and a LineConnectionWaypoint 1328 when the line is connected to another line, like to a IUMLGuiNoteLink 1334. The LineBegin 1327 property refers the first object and LineEnd 1328 property refers the second graphical object which the line connects.

Operation **IUMLGuiLineLink::AllWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	A derived list of all waypoints which are part of this line. All elements in this list are of type (or subtype of) IUMLGuiWaypoint 1366.					

Operation **IUMLGuiLineLink::EraseWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiLineLink::InsertWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiWaypoint** 1366			

Operation **IUMLGuiLineLink::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement** 1321			

documentation	A reference to the first object, where the line starts.

Operation **IUMLGuiLineLink::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of all waypoints which connect the line with other lines. All elements in this list are of type (or subtype of) IUMLGuiWaypoint 1366.					

Operation **IUMLGuiLineLink::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiElement 1321			
documentation	A reference to the second object, where the line ends.					

Operation **IUMLGuiLineLink::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiMiddleWaypoint 1330			

Operation **IUMLGuiLineLink::Waypoints**

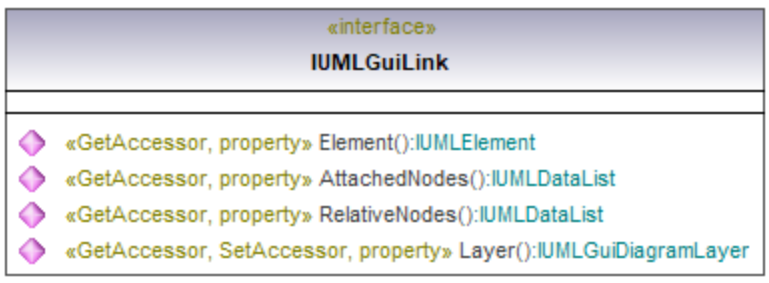
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	A list of all waypoints which form the vertices of this line. All elements in this list are of type (or subtype of) IUMLGuiWaypoint 1366.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

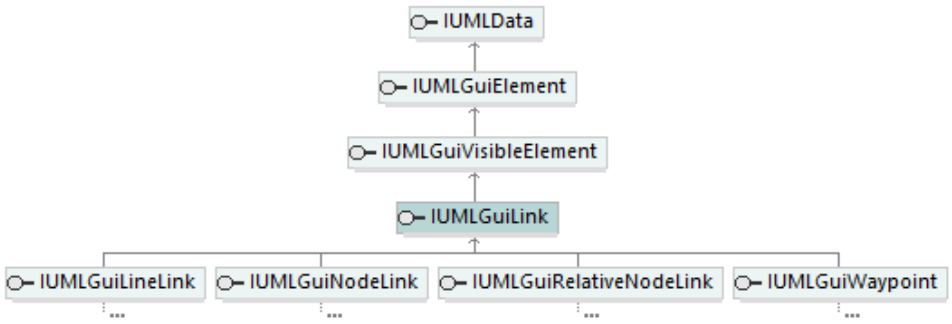
17.5.3.6.26 UModelAPI - IUMLGuiLink

Interface **IUMLGuiLink**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IDiagramWindow** 931 Interface **IUMLDataAll** 1019 Interface **IUMLGuiAttachedNode** 1305 Interface **IUMLGuiCommunicationDiagram** 1310 Interface **IUMLGuiDiagram** 1316 Operation **FocusedGuiElement** 932 **ScrollToGuiElement** 933 **SelectGuiElement** 933 Operation **AddUMLGuiContainmentLink** 1020 **AttachedTo** 1023 Operation **AttachedTo** 1306 Operation **AddUMLGuiCommunicationLink** 1310 Operation **AddUMLGuiContainmentLink** 1317
documentation	A GuiLink represents a graphical object on a diagram which is connected to an element from the UML (like a Class, an Interface or a Lifeline). This connected object can be accessed using the Element 1329 property. IUMLGuiLinks further can have attached nodes (IUMLGuiAttachedNode 1305) which appear when necessary, like Tagged Values and are associated with a Layer on the diagram.

Operation **IUMLGuiLink::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	Returns a derived list of all attached nodes of this element. All elements in this list are of type (or subtype) of IUMLGuiAttachedNode 1305.					

Operation **IUMLGuiLink::Element**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

Operation **IUMLGuiLink::Layer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer** 1320			

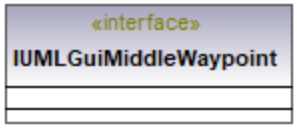
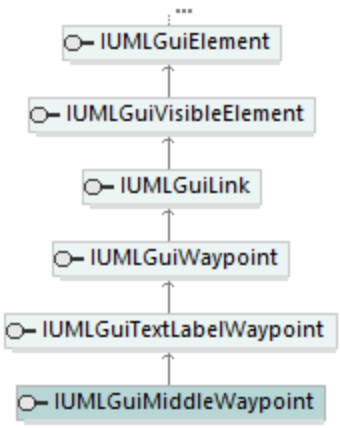
Operation **IUMLGuiLink::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	Returns a list of relative nodes to this gui link. The list contains only elements of type (or subtype of) IUMLGuiRelativeNodeLink 1337.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.27 UModelAPI - IUMLGuiMiddleWaypoint

Interface **IUMLGuiMiddleWaypoint**

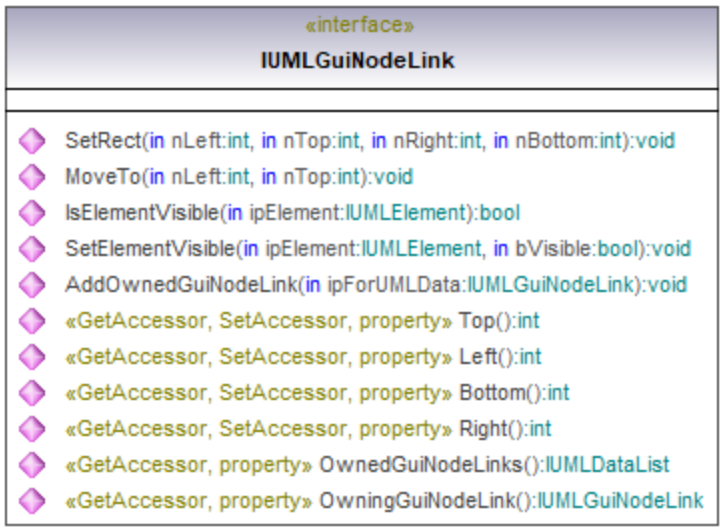
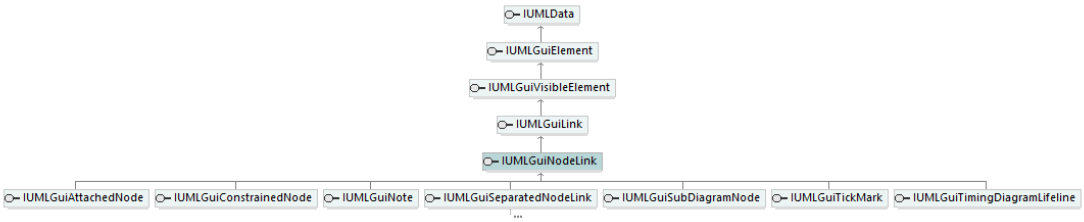
diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiLineLink** 1326 Operation **MiddleWaypoint** 1062 Operation **MiddleWaypoint** 1328
documentation	A middle waypoint is a special waypoint on a line (IUMLGuiLineLink 1326) which appears in the center of the line and can have text labels attached to it.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

 © 2016-2022 Altova GmbH

17.5.3.6.28 UModelAPI - IUMLGuiNodeLink

Interface **IUMLGuiNodeLink**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiDiagram** 1316 Interface **IUMLGuiNodeLink** 1331 Operation **AddOwnedGuiNodeLink** 1020 **AddUMLElement** 1020 **AddUMLGuiNodeLink** 1021 **AddUMLGuiNoteLink** 1021 **AddUMLLineElement** 1021 **OwningGuiNodeLink** 1069 Operation **AddUMLElement** 1317 **AddUMLGuiNodeLink** 1317 **AddUMLGuiNoteLink** 1317 **AddUMLLineElement** 1318 Operation **AddOwnedGuiNodeLink** 1331 **OwningGuiNodeLink** 1332
documentation	A GuiNodeLink represents a graphical object on a diagram which usually represents an element from the UML (for example a Class, an Interface or a Lifeline). It has a position defined by a rectangle which can be positioned freely on the diagram. A GuiNodeLink can itself contain other GuiNodeLinks, for example when displaying a big state in a state machine diagam which contains other, smaller substates. Additionally, if the GuiNodeLink displays cells on it, like for example operations and properties on a class, it can store for each element shown if the element should be visible or not. Use the SetElementVisible 1332 and IsElementVisible 1332 functions for this.

Operation **IUMLGuiNodeLink::AddOwnedGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLGuiNodeLink** 1331			
	return	**return**	**void**			

Operation **IUMLGuiNodeLink::Bottom**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiNodeLink::IsElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	**IUMLElement** 1157			
	return	**return**	**bool**			

Operation **IUMLGuiNodeLink::Left**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiNodeLink::MoveTo**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiNodeLink::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	Returns a list of all owned gui node links, all nodes which are directly contained in this node. All elements in this list are of type (or subtype of) IUMLGuiLink 1328.					

Operation **IUMLGuiNodeLink::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiNodeLink** 1331			

Operation **IUMLGuiNodeLink::Right**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiNodeLink::SetElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipElement	**in**	**IUMLElement** 1157			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

Operation **IUMLGuiNodeLink::SetRect**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	nRight	**in**	**int**			
	nBottom	**in**	**int**			
	return	**return**	**void**			

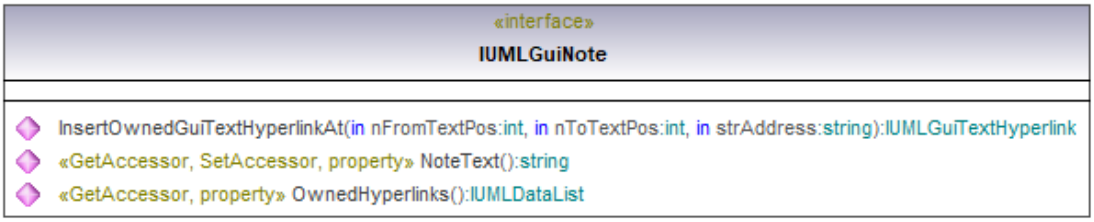
Operation **IUMLGuiNodeLink::Top**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

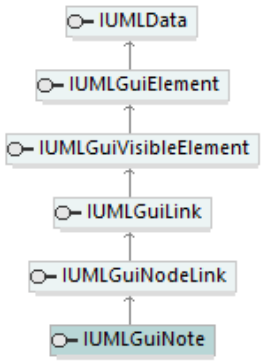
17.5.3.6.29 UModelAPI - IUMLGuiNote

Interface **IUMLGuiNote**

diagram



hierarchy



typedElements

Interface **IUMLDataAll** 1019

Interface **IUMLGuiDiagram** 1316

Operation **AddUMLGuiNote** 1021
AddUMLGuiNoteLink 1021
AddUMLGuiNoteLinkToLine 1021

Operation **AddUMLGuiNote** 1317
AddUMLGuiNoteLink 1317
AddUMLGuiNoteLinkToLine 1317

documentation

A IUMLGuiNote 1333 is the graphical object resembling a note on UModel diagrams displaying a text comment.

	It provides access to the note text and a list of hyperlinks in this text. These hyperlinks are nothing more than a list of URLs together with an begin and end number referencing positions in the text. Any text between a such a begin and end position is displayed as hyperlink and triggers UModel to open the URL when clicked.

Operation **IUMLGuiNote::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLGuiTextHyperlink** 1355			

Operation **IUMLGuiNote::NoteText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

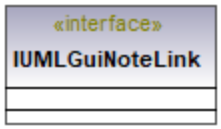
Operation **IUMLGuiNote::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			

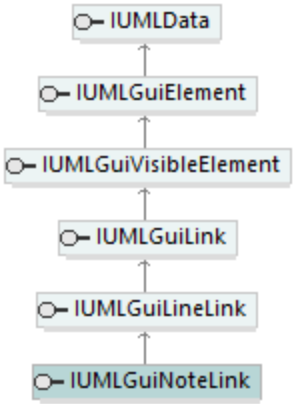
17.5.3.6.30 UModelAPI - IUMLGuiNoteLink

Interface **IUMLGuiNoteLink**

diagram



hierarchy

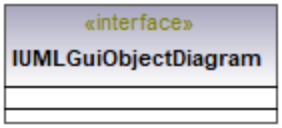
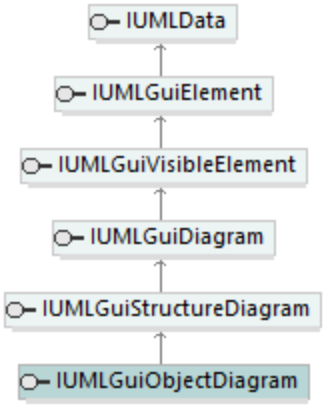


 © 2016-2022 Altova GmbH

typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiDiagram** 1316 Operation **AddUMLGuiNoteLink** 1021 **AddUMLGuiNoteLinkToLine** 1021 Operation **AddUMLGuiNoteLink** 1317 **AddUMLGuiNoteLinkToLine** 1317
documentation	A notelink is a special IUMLGuiLineLink 1326 which connects a IUMLGuiNote 1333 with another IUMLGuiLink 1328 element. It is displayed as a dotted line.

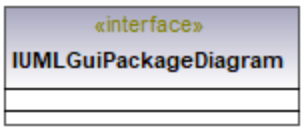
17.5.3.6.31 UModelAPI - IUMLGuiObjectDiagram

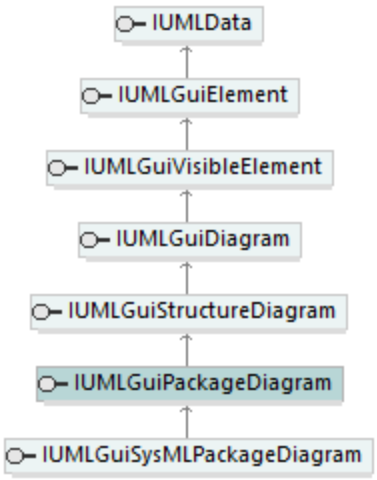
Interface **IUMLGuiObjectDiagram**

diagram	
hierarchy	

17.5.3.6.32 UModelAPI - IUMLGuiPackageDiagram

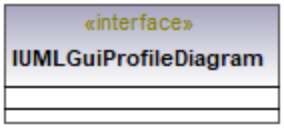
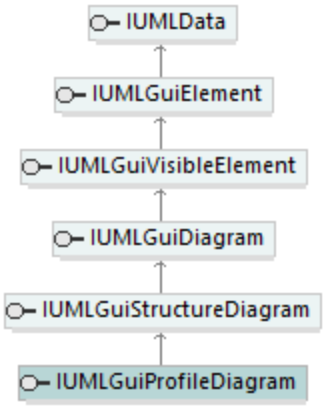
Interface **IUMLGuiPackageDiagram**

diagram	

hierarchy	

17.5.3.6.33 UModelAPI - IUMLGuiProfileDiagram

Interface **IUMLGuiProfileDiagram**

diagram	
hierarchy	

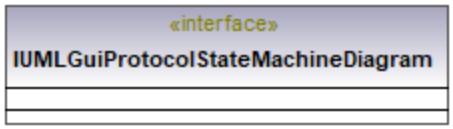
 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

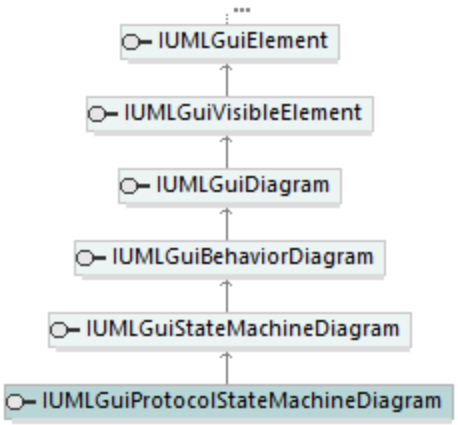
17.5.3.6.34 UModelAPI - IUMLGuiProtocolStateMachineDiagram

Interface **IUMLGuiProtocolStateMachineDiagram**

diagram



hierarchy

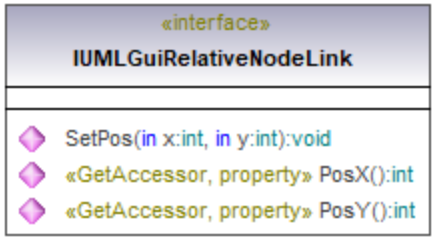


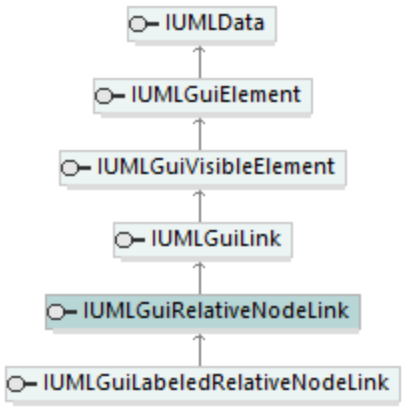
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.35 UModelAPI - IUMLGuiRelativeNodeLink

Interface **IUMLGuiRelativeNodeLink**

diagram



hierarchy	
documentation	This gui link is used for elements which are positioned relative to another node. For example the names of Messages on Communication diagrams use a specialization of this interface.

Operation **IUMLGuiRelativeNodeLink::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiRelativeNodeLink::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

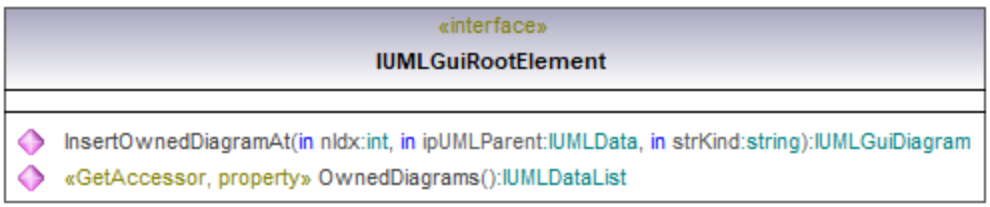
Operation **IUMLGuiRelativeNodeLink::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	x	**in**	**int**			
	y	**in**	**int**			
	return	**return**	**void**			

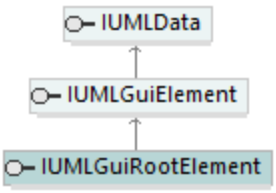
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.36 UModelAPI - IUMLGuiRootElement

Interface **IUMLGuiRootElement**

diagram	

 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IDocument** 938 Operation **GuiRoot** 941
documentation	This is the root interface for all graphical objects and contains all diagrams which exists in the UModel project.

Operation **IUMLGuiRootElement::InsertOwnedDiagramAt**

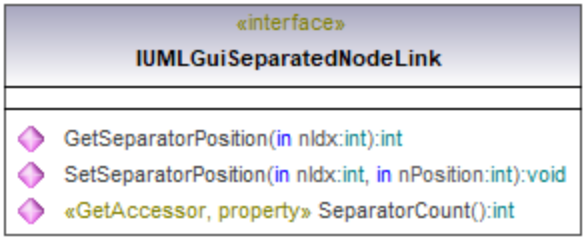
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUMLParent	**in**	IUMLData 1011			
	strKind	**in**	**string**			
	return	**return**	IUMLGuiDiagram 1316			

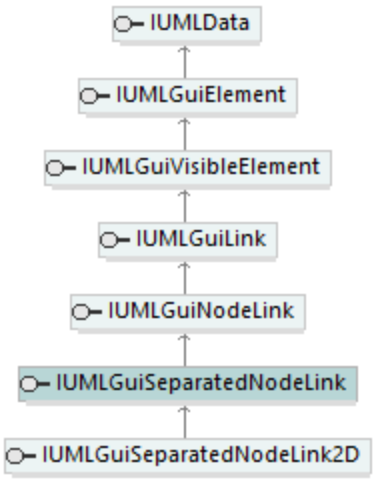
Operation **IUMLGuiRootElement::OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			
documentation	Returns a list of all diagrams in this UModel project. All elements in this list are of type (or subtype of) IUMLGuiDiagram 1316.					

17.5.3.6.37 UModelAPI - IUMLGuiSeparatedNodeLink

Interface **IUMLGuiSeparatedNodeLink**

diagram	

hierarchy	
documentation	This node link represents a graphical object on a UModel diagram which can be separated into two or more parts by one or more either horizontal or vertical lines. For each line, the position of the separator is stored in this node. This node type is used for example by CombinedFragments, ActivityPartitions and States with regions.

Operation **IUMLGuiSeparatedNodeLink::GetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink::SeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

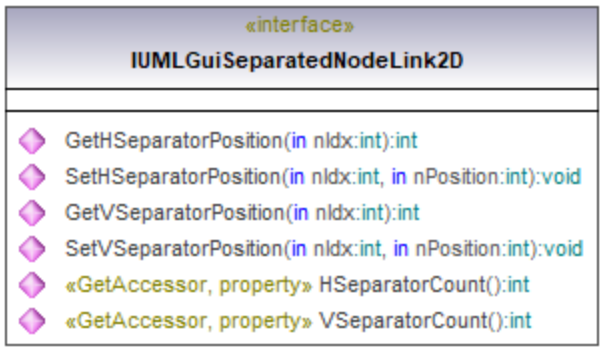
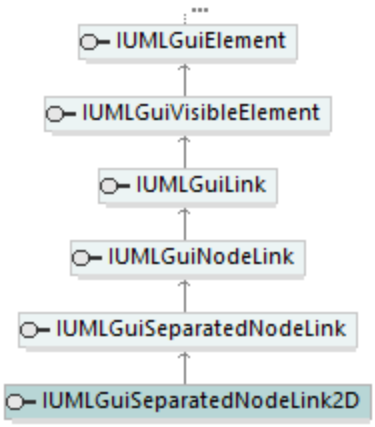
Operation **IUMLGuiSeparatedNodeLink::SetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

17.5.3.6.38 UModelAPI - IUMLGuiSeparatedNodeLink2D

Interface **IUMLGuiSeparatedNodeLink2D**

diagram	
hierarchy	
documentation	This node link represents a graphical object on a UModel diagram which can be separated into parts by one or more horizontal or vertical lines, but in contrast to IUMLGuiSeparatedNodeLink[1339], the node can be subdivided vertically and horizontally at the same time. For each vertical or horizontal separation line, the position of the separator is stored in this node. This node type is used for example by ActivityPartitions.

Operation **IUMLGuiSeparatedNodeLink2D::GetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink2D::GetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiSeparatedNodeLink2D::HSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default

	return	**return**	**int**

Operation **IUMLGuiSeparatedNodeLink2D::SetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiSeparatedNodeLink2D::SetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiSeparatedNodeLink2D::VSeparatorCount**

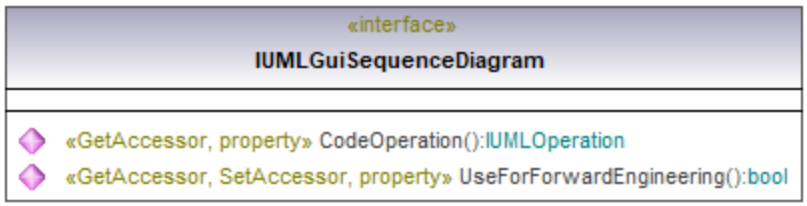
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.39 UModelAPI - IUMLGuiSequenceDiagram

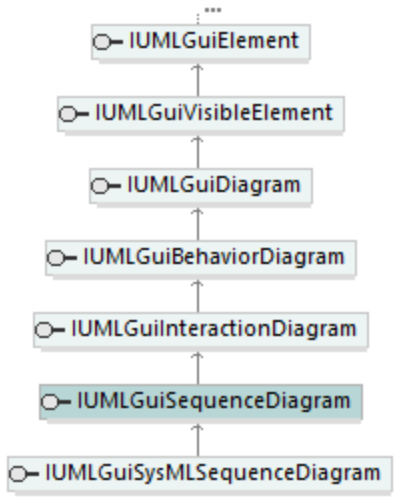
Interface **IUMLGuiSequenceDiagram**

diagram



© 2016-2022 Altova GmbH

hierarchy	



Operation **IUMLGuiSequenceDiagram::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLOperation 1236			

Operation **IUMLGuiSequenceDiagram::UseForForwardEngineering**

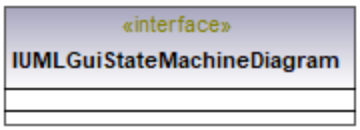
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

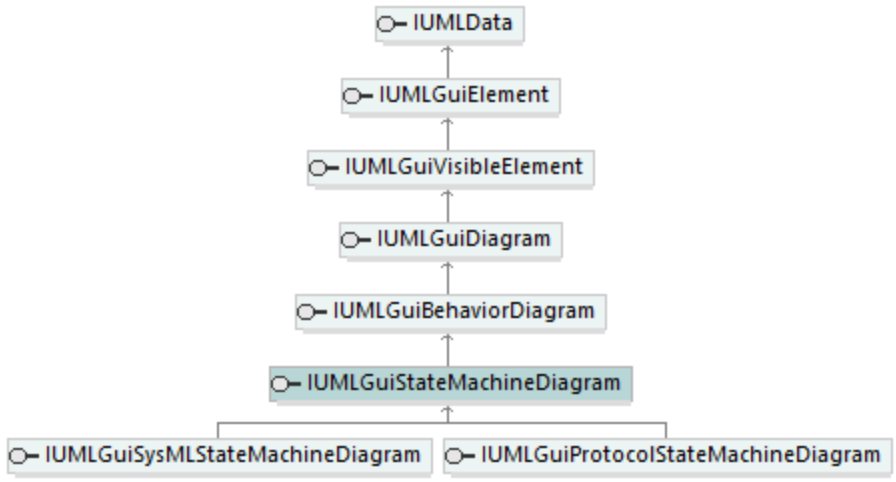
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.40 UModelAPI - IUMLGuiStateMachineDiagram

Interface **IUMLGuiStateMachineDiagram**

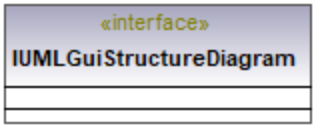
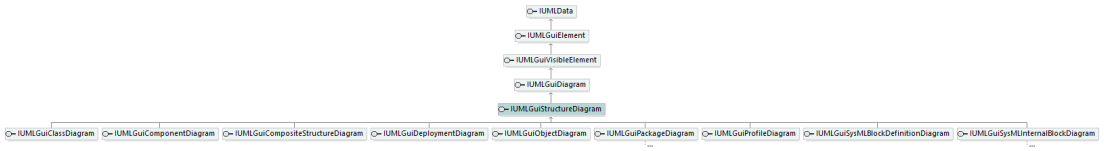
diagram	



hierarchy	

17.5.3.6.41 UModelAPI - IUMLGuiStructureDiagram

Interface **IUMLGuiStructureDiagram**

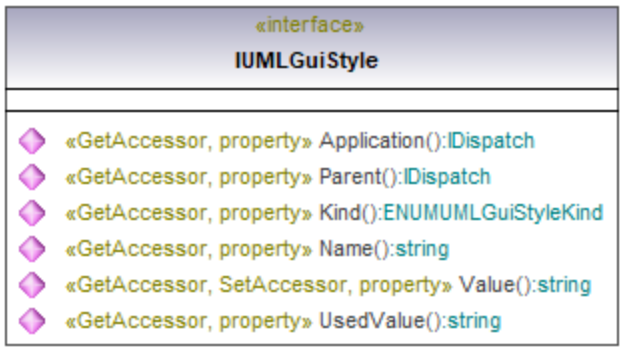
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

 © 2016-2022 Altova GmbH

17.5.3.6.42 UModelAPI - IUMLGuiStyle

Interface **IUMLGuiStyle**

diagram	
typedElements	Interface IUMLGuiStyles 1346 Operation GetStyle 1346 Item 1347

Operation **IUMLGuiStyle::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyle::Kind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLGuiStyleKind 1371			

Operation **IUMLGuiStyle::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiStyle::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyle::UsedValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

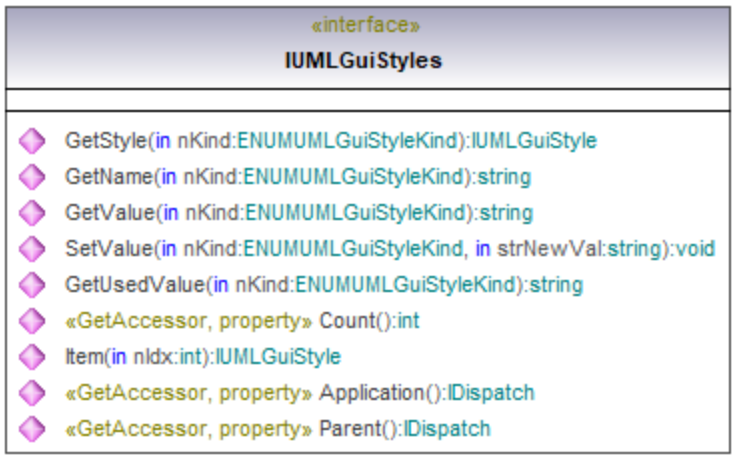
Operation **IUMLGuiStyle::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

© 2016-2022 Altova GmbH

17.5.3.6.43 UModelAPI - IUMLGuiStyles

Interface **IUMLGuiStyles**

diagram	
typedElements	Interface IDocument 938 Interface IUMLDataAll 1019 Interface IUMLGuiVisibleElement 1365 Interface IUMLStereotype 1273 Operation ElementFamilyStyles 939 Operation LineStyles 942 Operation NodeStyles 943 Operation ProjectStyles 944 Operation StereotypedElementStyles 1083 Operation Styles 1084 Operation Styles 1365 Operation StereotypedElementStyles 1274

Operation **IUMLGuiStyles::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyles::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiStyles::GetName**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	ENUMUMLGuiStyleKind 1371			
	return	**return**	**string**			

Operation **IUMLGuiStyles::GetStyle**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	ENUMUMLGuiStyleKind 1371			
	return	**return**	IUMLGuiStyle 1345			

Operation **IUMLGuiStyles::GetUsedValue**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind** 1371			
	return	**return**	**string**			

Operation **IUMLGuiStyles::GetValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind** 1371			
	return	**return**	**string**			

Operation **IUMLGuiStyles::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiStyle** 1345			

Operation **IUMLGuiStyles::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

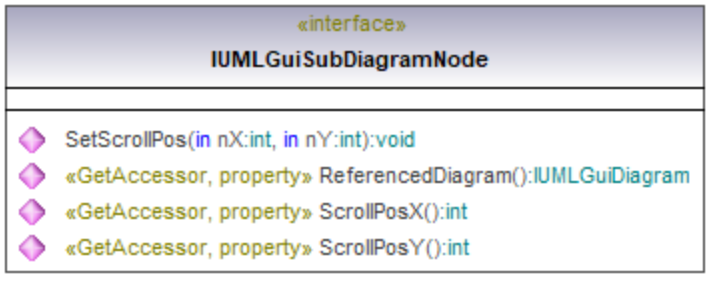
Operation **IUMLGuiStyles::SetValue**

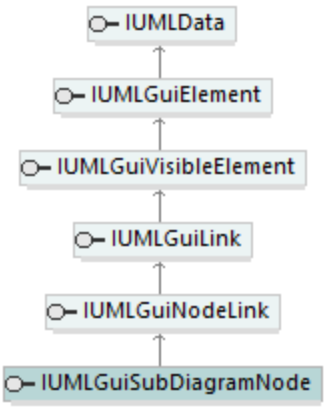
parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind** 1371			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

17.5.3.6.44 UModelAPI - IUMLGuiSubDiagramNode

Interface **IUMLGuiSubDiagramNode**

diagram	



hierarchy	
documentation	The sub diagram node represents a node link on a diagram which again includes another diagram. This is used for example on interaction overview diagrams to display sequence, communication and timing diagrams inside nodes. The property ReferencedDiagram[1348] controls the diagrams which is shown inside the node.

Operation **IUMLGuiSubDiagramNode::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagram**[1316]			

Operation **IUMLGuiSubDiagramNode::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiSubDiagramNode::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

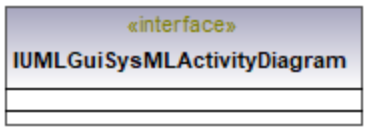
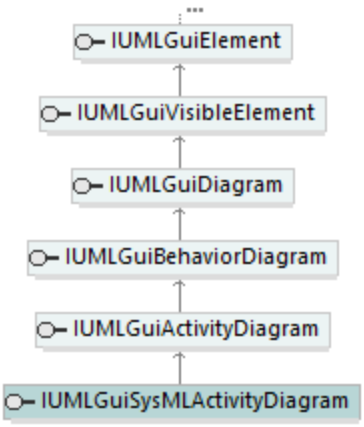
Operation **IUMLGuiSubDiagramNode::SetScrollPos**

parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

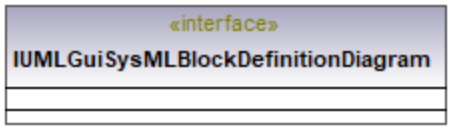
17.5.3.6.45 UModelAPI - IUMLGuiSysMLActivityDiagram

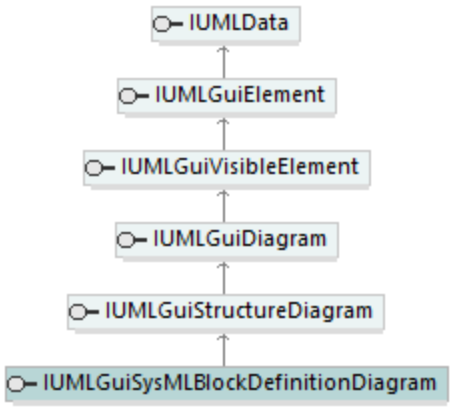
Interface **IUMLGuiSysMLActivityDiagram**

diagram	
hierarchy	

17.5.3.6.46 UModelAPI - IUMLGuiSysMLBlockDefinitionDiagram

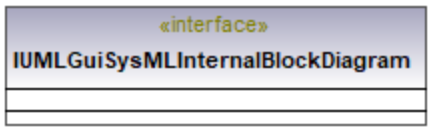
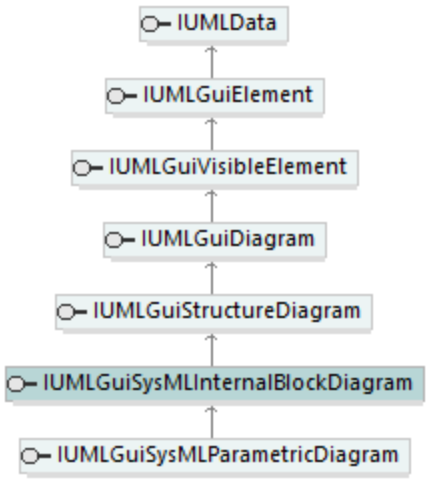
Interface **IUMLGuiSysMLBlockDefinitionDiagram**

diagram	

hierarchy	

17.5.3.6.47 UModelAPI - IUMLGuiSysMLInternalBlockDiagram

Interface **IUMLGuiSysMLInternalBlockDiagram**

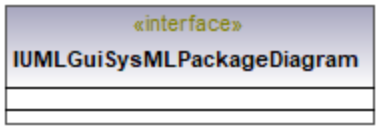
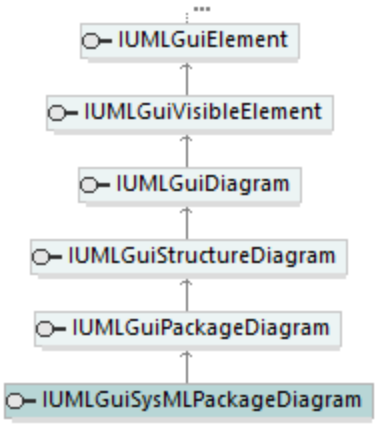
diagram	
hierarchy	

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.48 UModelAPI - IUMLGuiSysMLPackageDiagram

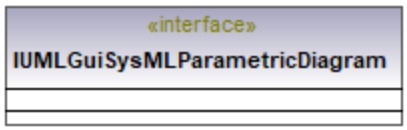
Interface **IUMLGuiSysMLPackageDiagram**

diagram	
hierarchy	

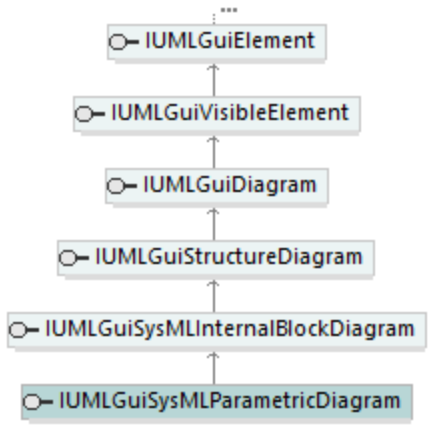
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.6.49 UModelAPI - IUMLGuiSysMLParametricDiagram

Interface **IUMLGuiSysMLParametricDiagram**

diagram	

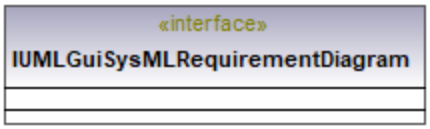
hierarchy



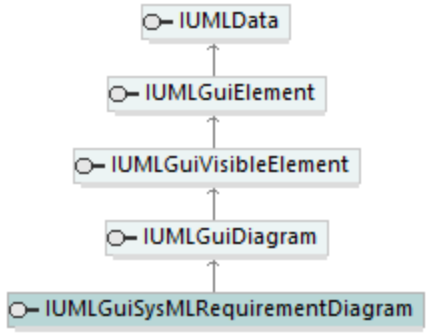
17.5.3.6.50 UModelAPI - IUMLGuiSysMLRequirementDiagram

Interface **IUMLGuiSysMLRequirementDiagram**

diagram



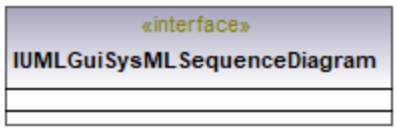
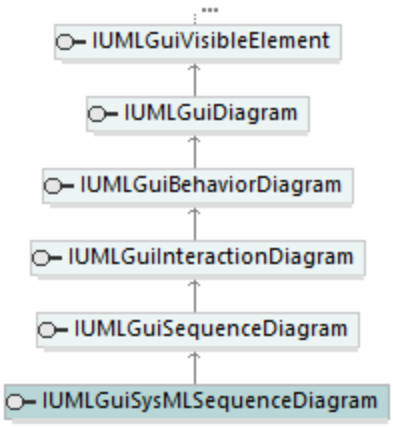
hierarchy



 © 2016-2022 Altova GmbH

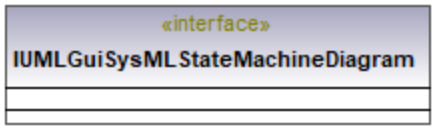
17.5.3.6.51 UModelAPI - IUMLGuiSysMLSequenceDiagram

Interface **IUMLGuiSysMLSequenceDiagram**

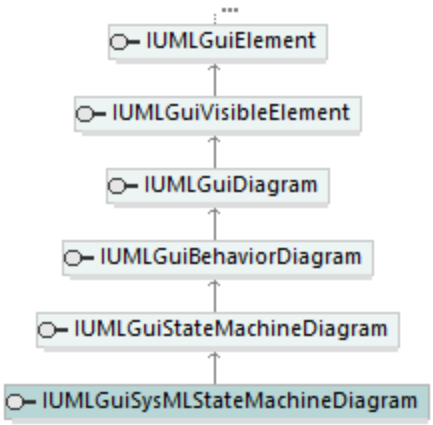
diagram	
hierarchy	

17.5.3.6.52 UModelAPI - IUMLGuiSysMLStateMachineDiagram

Interface **IUMLGuiSysMLStateMachineDiagram**

diagram	

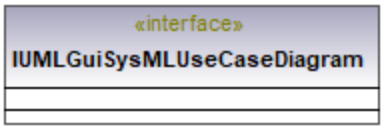
hierarchy



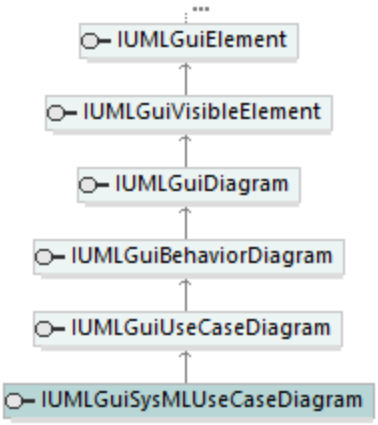
17.5.3.6.53 UModelAPI - IUMLGuiSysMLUseCaseDiagram

Interface **IUMLGuiSysMLUseCaseDiagram**

diagram



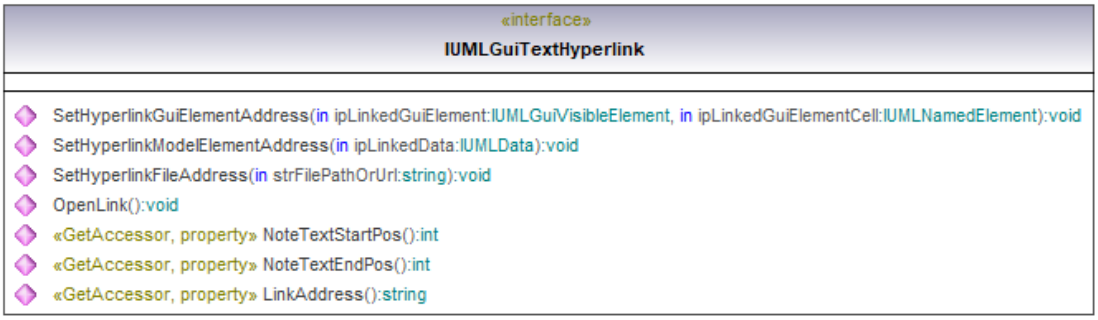
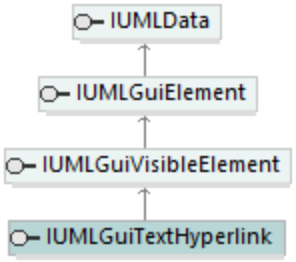
hierarchy



 © 2016-2022 Altova GmbH

17.5.3.6.54 UModelAPI - IUMLGuiTextHyperlink

Interface **IUMLGuiTextHyperlink**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLGuiDiagram** 1316 Interface **IUMLGuiNote** 1333 Operation **InsertOwnedGuiTextHyperlinkAt** 1048 Operation **InsertOwnedGuiTextHyperlinkAt** 1318 Operation **InsertOwnedGuiTextHyperlinkAt** 1334
documentation	Text Hyperlinks store an URL together w ith a begin and an end number referencing positions in some text. Any text betw een a such a begin and end position is displayed as hyperlink and triggers UModel to open the URL w hen clicked. This is used in IUMLGuiNote 1333 to create hyperlinks inside of the text comment for example.

Operation **IUMLGuiTextHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLGuiTextHyperlink::NoteTextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextHyperlink::NoteTextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

Operation **IUMLGuiTextHyperlink::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**void**			

Operation **IUMLGuiTextHyperlink::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement** 1365			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement** 1222			
	return	**return**	**void**			

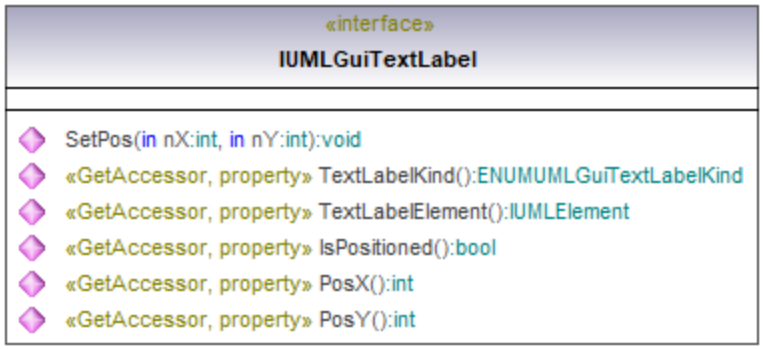
Operation **IUMLGuiTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData** 1011			
	return	**return**	**void**			

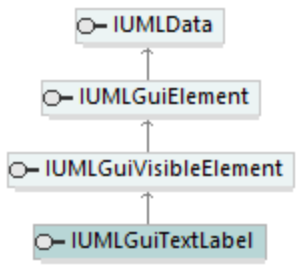
17.5.3.6.55 UModelAPI - IUMLGuiTextLabel

Interface **IUMLGuiTextLabel**

diagram



 © 2016-2022 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** 1019 Operation **GetTextLabelText** 1037 **IsTextLabelVisible** 1058 **SetTextLabelVisible** 1081 Interface **IUMLGuiTextLabelWaypoint** 1358 Operation **GetTextLabelText** 1358 **IsTextLabelVisible** 1358 **SetTextLabelVisible** 1359
documentation	A text label is an graphical object displaying additional data at the begin, end or at the center of a line. The IUMLGuiTextLabel 1356 interface provides access to this element. The text label can reference an UML Element using the TextLabelElement 1357 property and has a TextLabelKind 1357 which affects what text is shown in the text label.

Operation **IUMLGuiTextLabel::IsPositioned**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiTextLabel::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextLabel::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTextLabel::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTextLabel::TextLabelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement** 1157			

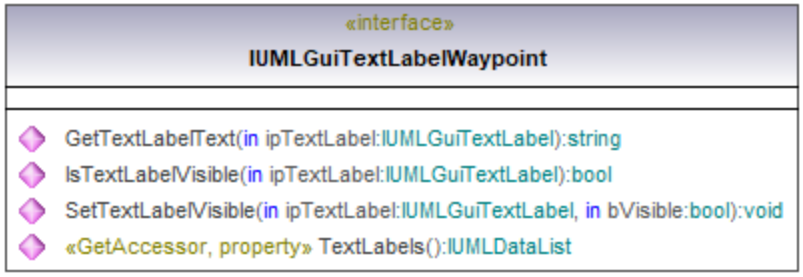
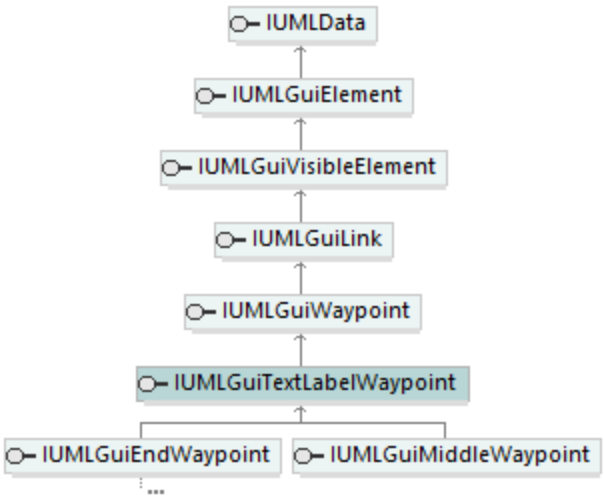
Operation **IUMLGuiTextLabel::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLGuiTextLabelKind** 1373			

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.6.56 UModelAPI - IUMLGuiTextLabelWaypoint

Interface **IUMLGuiTextLabelWaypoint**

diagram	
hierarchy	
documentation	A text label waypoint is a special waypoint as part of a IUMLGuiLineLink 1326 which can have one or more text labels associated with it. Text label waypoints can usually appear at the begin, end or in the middle of a line. The waypoint stores not only the text labels it shows, but also the visibility of each text label.

Operation **IUMLGuiTextLabelWaypoint::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel** 1356			
	return	**return**	**string**			

Operation **IUMLGuiTextLabelWaypoint::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2016-2022 Altova GmbH

	ipTextLabel	**in**	**IUMLGuiTextLabel** 1356
	return	**return**	**bool**

Operation **IUMLGuiTextLabelWaypoint::SetTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel** 1356			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLGuiTextLabelWaypoint::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList** 1013			
documentation	Returns a list of all text labels of this waypoint. Contains only elements of type (or subtype of) IUMLGuiTextLabel 1356 .					

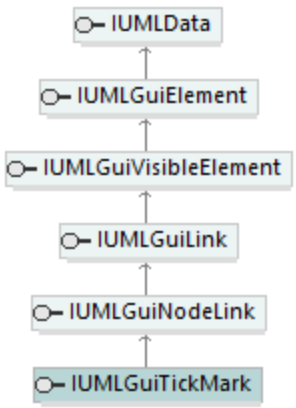
17.5.3.6.57 UModelAPI - IUMLGuiTickMark

Interface **IUMLGuiTickMark**

diagram



hierarchy



typedElements	Interface **IUMLGuiTimingDiagram** 1360 Operation **AddUMLGuiTickMark** 1360
documentation	A tick mark is a special graphical item appearing on the border of lifelines on timing diagrams. It represents a certain point in time and is displayed as short vertical line. It has a Value 1360 property which is displayed as text below the vertical line.

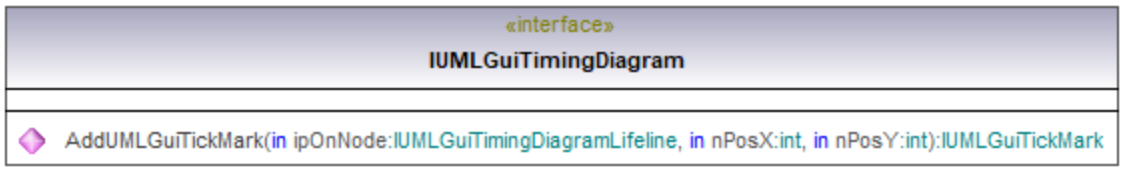
Operation **IUMLGuiTickMark::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

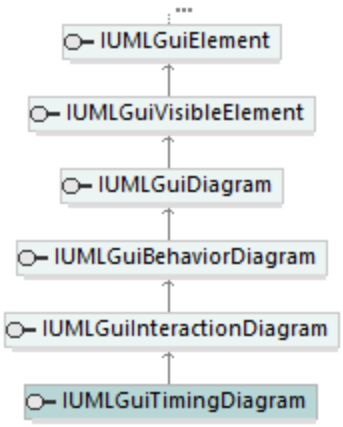
17.5.3.6.58 UModelAPI - IUMLGuiTimingDiagram

Interface **IUMLGuiTimingDiagram**

diagram



hierarchy



Operation **IUMLGuiTimingDiagram::AddUMLGuiTickMark**

parameter	name	direction	type	type modifier	multiplicity	default
	ipOnNode	**in**	**IUMLGuiTimingDiagramLifeline** 1361			
	nPosX	**in**	**int**			
	nPosY	**in**	**int**			

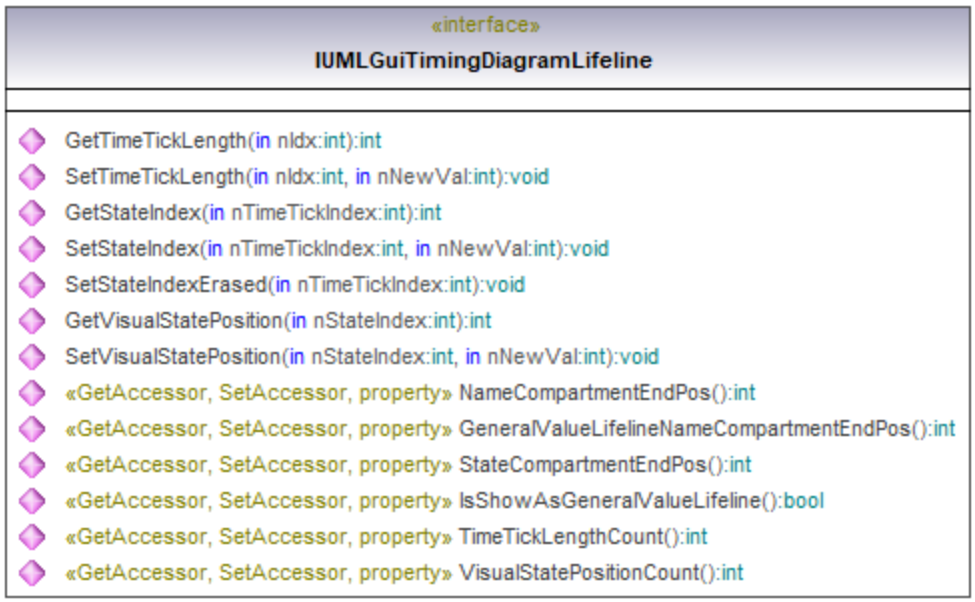
 © 2016-2022 Altova GmbH

	return	**return**	IUMLGuiTickMark 1359

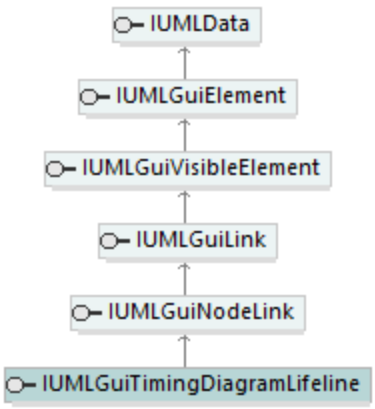
17.5.3.6.59 UModelAPI - IUMLGuiTimingDiagramLifeline

Interface **IUMLGuiTimingDiagramLifeline**

diagram



hierarchy



typedElements

Interface IUMLGuiTimingDiagram 1360

Operation AddUMLGuiTickMark 1360

documentation	A IUMLGuiTimingDiagramLifeline[1361] is the graphical representation of a lifeline on a timing diagram. This type of lifeline has several options to display its data and provides access to these through its numerous properties.

Operation **IUMLGuiTimingDiagramLifeline::GeneralValueLifelineNameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::GetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::GetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::GetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::IsShowAsGeneralValueLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLGuiTimingDiagramLifeline::NameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::SetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::SetStateIndexErased**

parameter	name	direction	type	type modifier	multiplicity	default
	nTimeTickIndex	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::SetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

© 2016-2022 Altova GmbH

Operation **IUMLGuiTimingDiagramLifeline::SetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nStateIndex	**in**	**int**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLGuiTimingDiagramLifeline::StateCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiTimingDiagramLifeline::TimeTickLengthCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

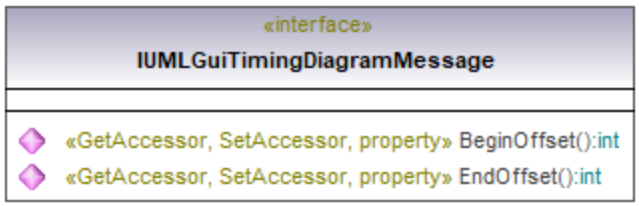
Operation **IUMLGuiTimingDiagramLifeline::VisualStatePositionCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

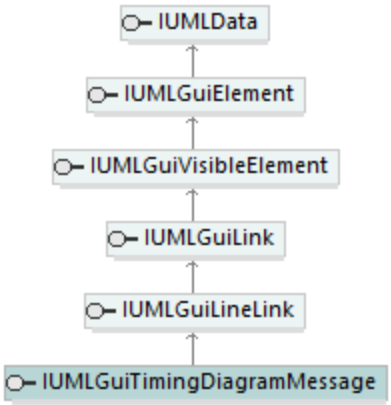
17.5.3.6.60 UModelAPI - IUMLGuiTimingDiagramMessage

Interface **IUMLGuiTimingDiagramMessage**

diagram



hierarchy



documentation	A IUMLGuiTimingDiagramMessage 1363 is a line usually connecting two IUMLGuiTimingDiagramLifeline 1361 s. For each lifeline on one of its ends, it stores its position from the start of the state or general value compartement.

Operation **IUMLGuiTimingDiagramMessage::BeginOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

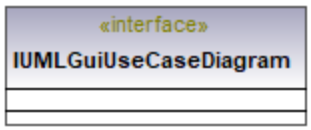
Operation **IUMLGuiTimingDiagramMessage::EndOffset**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

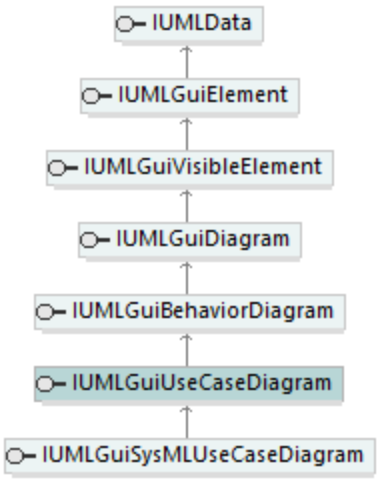
17.5.3.6.61 UModelAPI - IUMLGuiUseCaseDiagram

Interface **IUMLGuiUseCaseDiagram**

diagram



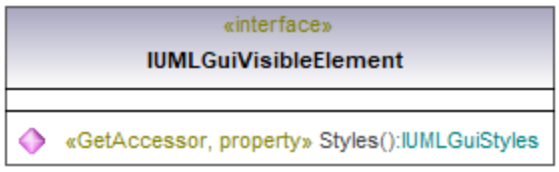
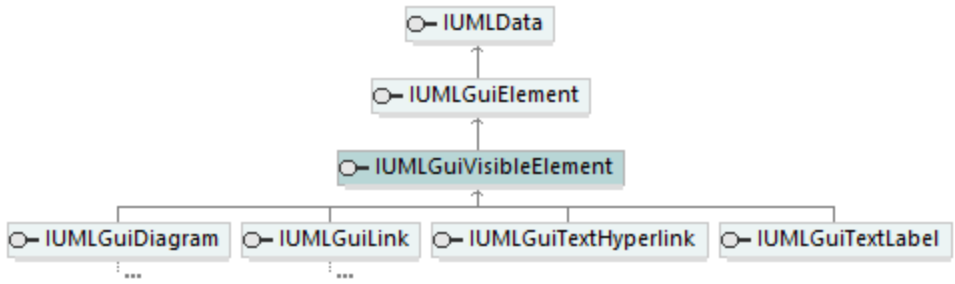
hierarchy



 © 2016-2022 Altova GmbH

17.5.3.6.62 UModelAPI - IUMLGuiVisibleElement

Interface **IUMLGuiVisibleElement**

diagram	
hierarchy	
typedElements	Interface IUMLCommentTextHyperlink 1133 Interface IUMLDataAll 1019 Interface IUMLGuiTextHyperlink 1355 Interface IUMLHyperlink2GuiElement 1183 Interface IUMLNamedElement 1222 Operation SetHyperlinkGuiElementAddress 1134 Operation InsertOwnedHyperlink2GuiElementAt 1048 LinkedGuiElement 1060 SetHyperlinkGuiElementAddress 1075 Operation SetHyperlinkGuiElementAddress 1356 Operation LinkedGuiElement 1184 Operation InsertOwnedHyperlink2GuiElementAt 1223
documentation	The IUMLGuiVisibleElement 1365 is the base interface for most visible elements which can be placed on UModel diagrams. Visible elements have a style with which it is possible to infliuence its color and/or shape, depending on the actual type of the visible element.

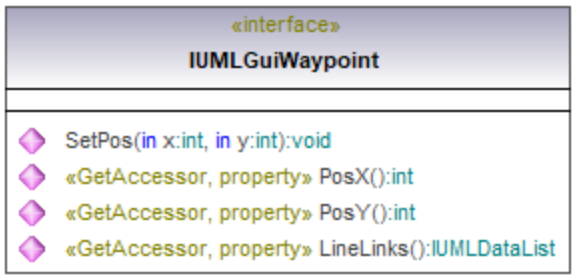
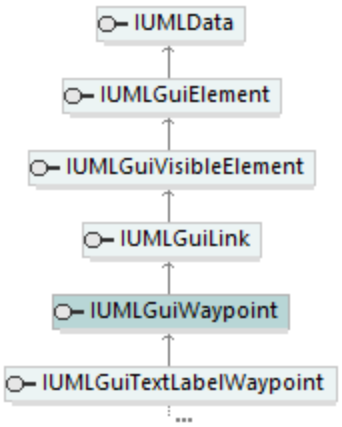
Operation **IUMLGuiVisibleElement::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLGuiStyles 1346			

© 2016-2022 Altova GmbH

17.5.3.6.63 UModelAPI - IUMLGuiWaypoint

Interface **IUMLGuiWaypoint**

diagram	
hierarchy	
typedElements	Interface IUMLDataAll 1019 Interface IUMLGuiLineLink 1326 Operation InsertWaypointAt 1053 Operation InsertWaypointAt 1327
documentation	A IUMLGuiWaypoint 1366 is a part of a IUMLGuiLineLink 1326 which defines the position of one point of a line. There are several subtypes of this interface which for example make it possible to attach text labels or lines to a waypoint or line.

Operation **IUMLGuiWaypoint::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList 1013			

Operation **IUMLGuiWaypoint::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLGuiWaypoint::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

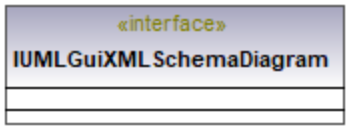
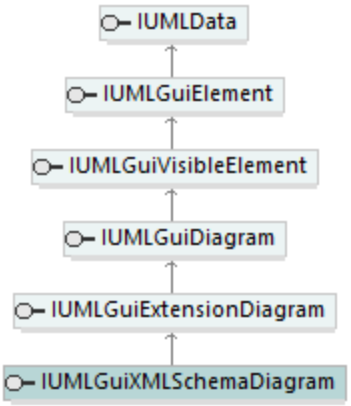
Operation **IUMLGuiWaypoint::SetPos**

© 2016-2022 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	x	**in**	**int**			
	y	**in**	**int**			
	return	**return**	**void**			

17.5.3.6.64 UModelAPI - IUMLGuiXMLSchemaDiagram

Interface **IUMLGuiXMLSchemaDiagram**

diagram	
hierarchy	

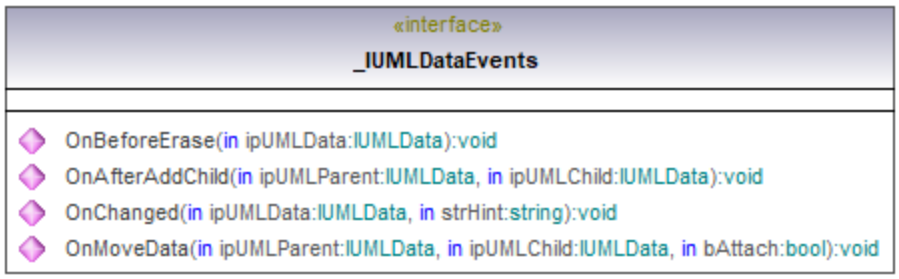
17.5.3.7 Events

Dies ist eine Liste aller von der UModel API auf `UMLData`-Ebene gesendeten Events.

Siehe auch UMLData Events und Eventfilters 871.

17.5.3.7.1 UModelAPI - _IUMLDataEvents

Interface **_IUMLDataEvents**

diagram	

Operation **_IUMLDataEvents::OnAfterAddChild**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLParent	**in**	**IUMLData** 1011			
	ipUMLChild	**in**	**IUMLData** 1011			
	return	**return**	**void**			

Operation **_IUMLDataEvents::OnBeforeErase**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData** 1011			
	return	**return**	**void**			

Operation **_IUMLDataEvents::OnChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData** 1011			
	strHint	**in**	**string**			
	return	**return**	**void**			
documentation	**strHint** is for future use only!					

Operation **_IUMLDataEvents::OnMoveData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLParent	**in**	**IUMLData** 1011			
	ipUMLChild	**in**	**IUMLData** 1011			
	bAttach	**in**	**bool**			
	return	**return**	**void**			

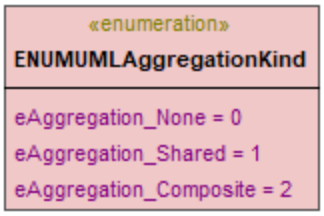
© 2016-2022 Altova GmbH

17.5.3.8 Enumerations

Dies ist eine Liste aller von der UModel API auf `UMLData`-Ebene verwendeten Enumerationen. Falls Ihre Skripting-Umgebung Enumerationen nicht unterstützt, verwenden Sie statt dessen die Zahlenwerte.

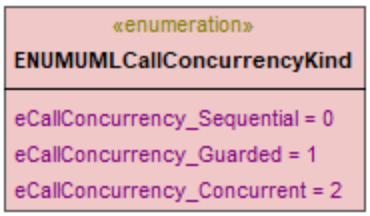
17.5.3.8.1 UModelAPI - ENUMUMLAggregationKind

Enumeration **ENUMUMLAggregationKind**

diagram	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLProperty** 1251 Operation **Aggregation** 1021 Operation **Aggregation** 1252

17.5.3.8.2 UModelAPI - ENUMUMLCallConcurrencyKind

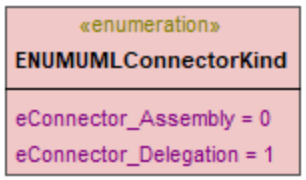
Enumeration **ENUMUMLCallConcurrencyKind**

diagram	
typedElements	Interface **IUMLBehavioralFeature** 1112 Interface **IUMLDataAll** 1019 Operation **Concurrency** 1113 Operation **Concurrency** 1026

© 2016-2022 Altova GmbH

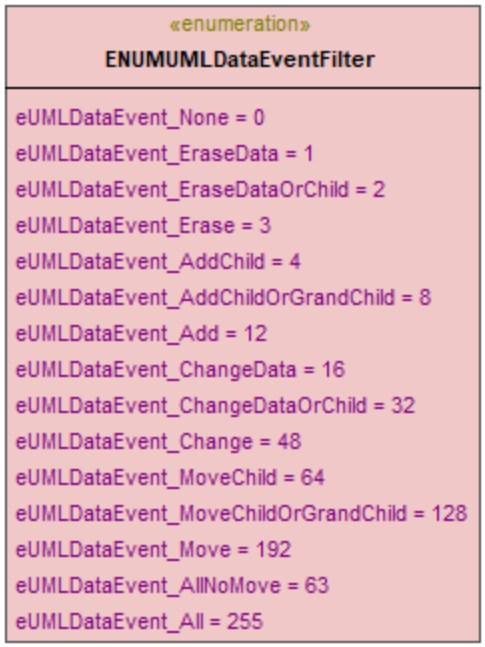
17.5.3.8.3 UModelAPI - ENUMUMLConnectorKind

Enumeration **ENUMUMLConnectorKind**

diagram	
typedElements	Interface IUMLConnector 1140 Interface IUMLDataAll 1019 Operation ConnectorKind 1140 Operation ConnectorKind 1026

17.5.3.8.4 UModelAPI - ENUMUMLDataEventFilter

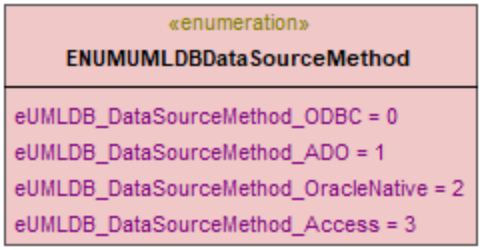
Enumeration **ENUMUMLDataEventFilter**

diagram	

 © 2016-2022 Altova GmbH

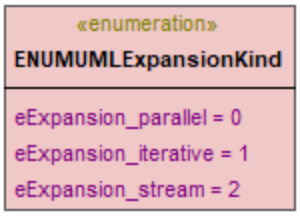
17.5.3.8.5 UModelAPI - ENUMUMLDBDataSourceMethod

Enumeration **ENUMUMLDBDataSourceMethod**

diagram	
typedElements	Interface **IImportDatabaseDlg** 962 — Operation **SelectNewDataSourceByConnectionString** 962

17.5.3.8.6 UModelAPI - ENUMUMLExpansionKind

Enumeration **ENUMUMLExpansionKind**

diagram	
typedElements	Interface **IUMLDataAll** 1019 — Operation **Mode** 1063 Interface **IUMLExpansionRegion** 1171 — Operation **Mode** 1172

17.5.3.8.7 UModelAPI - ENUMUMLGuiStyleKind

Enumeration **ENUMUMLGuiStyleKind**

diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **IUMLGuiStyle** 1345 — Operation **Kind** 1345

	Interface IUMLGuiStyles 1346	Operation GetName 1346 GetStyle 1346 GetUsedValue 1346 GetValue 1347 SetValue 1347

© 2016-2022 Altova GmbH

17.5.3.8.8 UModelAPI - ENUMUMLGuiTextLabelKind

Enumeration **ENUMUMLGuiTextLabelKind**

diagram



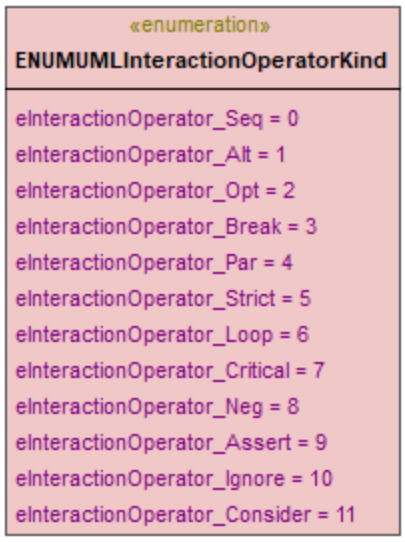
typedElements

Interface **IUMLDataAll** 1019
Interface **IUMLGuiTextLabel** 1356
Operation **TextLabelKind** 1035
Operation **TextLabelKind** 1357

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.8.9 UModelAPI - ENUMUMLInteractionOperatorKind

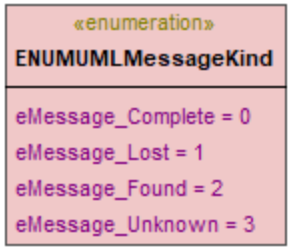
Enumeration **ENUMUMLInteractionOperatorKind**

diagram	
typedElements	Interface IUMLCombinedFragment 1131 Interface IUMLDataAll 1019 Operation InteractionOperator 1131 Operation InteractionOperator 1053

UML documentation generated by UModel UML Editor http://www.altova.com/umodel Wed Jan 27 07:46:44 2021

17.5.3.8.10 UModelAPI - ENUMUMLMessageKind

Enumeration **ENUMUMLMessageKind**

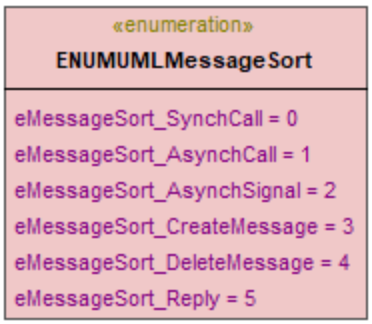
diagram	
typedElements	Interface IUMLDataAll 1019 Interface IUMLMessage 1216 Operation MessageKind 1062 Operation MessageKind 1217

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.11 UModelAPI - ENUMUMLMessageSort

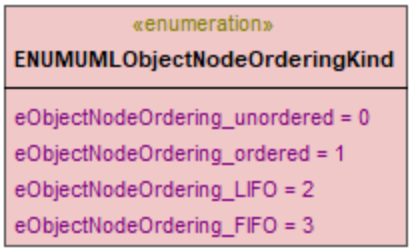
Enumeration **ENUMUMLMessageSort**

diagram	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLMessage** 1216 Operation **MessageSort** 1062 Operation **MessageSort** 1217

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.12 UModelAPI - ENUMUMLObjectNodeOrderingKind

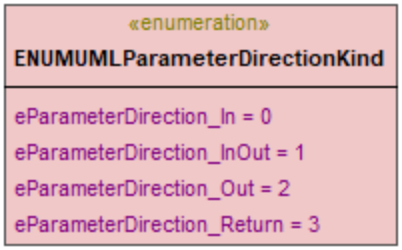
Enumeration **ENUMUMLObjectNodeOrderingKind**

diagram	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLObjectNode** 1229 Operation **Ordering** 1035 Operation **Ordering** 1230

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.13 UModelAPI - ENUMUMLParameterDirectionKind

Enumeration **ENUMUMLParameterDirectionKind**

diagram	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLParameter** 1244 Operation **Direction** 1028 Operation **Direction** 1245

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.14 UModelAPI - ENUMUMLPredefinedElement

Enumeration **ENUMUMLPredefinedElement**

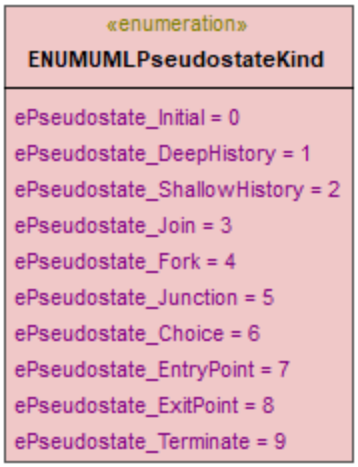
diagram	*The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (https://www.altova.com/manual/UModel/umodelenterprise/).*
typedElements	Interface **IUMLDataAll** 1019 Operation **ApplyPredefinedStereotype** 1023 **FindPredefinedOwnedElement** 1034 **GetStereotypeApplicationForPredefinedStereotype** 1036 **IsPredefinedStereotypeApplied** 1057 **SetPredefinedTaggedValueAt** 1080 **UnapplyPredefinedStereotype** 1087 Interface **IUMLElement** 1157 Operation **ApplyPredefinedStereotype** 1158 **FindPredefinedOwnedElement** 1158 **GetStereotypeApplicationForPredefinedStereotype** 1159 **IsPredefinedStereotypeApplied** 1159 **UnapplyPredefinedStereotype** 1160 Interface **IUMLStereotypeApplication** 1275 Operation **SetPredefinedTaggedValueAt** 1275
documentation	**Deprecated:** ePredefined_Java_finalStereotypeOfProperty ePredefined_Java_finalStereotypeOfOperation ePredefined_Java_finalStereotypeOfClass

 © 2016-2022 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.15 UModelAPI - ENUMUMLPseudostateKind

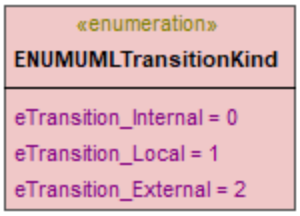
Enumeration **ENUMUMLPseudostateKind**

diagram	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLPseudostate** 1257 Operation **PseudostateKind** 1072 Operation **PseudostateKind** 1257

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.16 UModelAPI - ENUMUMLTransitionKind

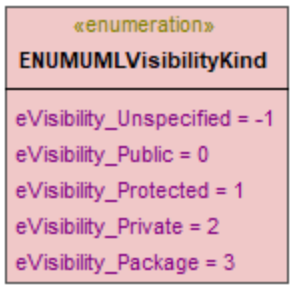
Enumeration **ENUMUMLTransitionKind**

diagram	
typedElements	Interface **IUMLDataAll** 1019 Interface **IUMLTransition** 1291 Operation **TransitionKind** 1086 Operation **TransitionKind** 1292

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** Wed Jan 27 07:46:44 2021

17.5.3.8.17 UModelAPI - ENUMUMLVisibilityKind

Enumeration **ENUMUMLVisibilityKind**

diagram	
typedElements	Interface **ILocalOptionsEditing** 980 Interface **IUMLDataAll** 1019 Interface **IUMLElementImport** 1161 Interface **IUMLNamedElement** 1222 Interface **IUMLPackageImport** 1242 Operation **OperationsDefaultVisibility** 981 Operation **PropertiesDefaultVisibility** 981 Operation **Visibility** 1088 Operation **Visibility** 1161 Operation **Visibility** 1224 Operation **Visibility** 1243

© 2016-2022 Altova GmbH

18 SPL-Referenz

In diesem Abschnitt erhalten Sie eine Übersicht über Spy Programming Language (SPL), die Template-Sprache von Code Generator.

Es wird vorausgesetzt, dass Sie bereits über Programmiererfahrung verfügen und mit Operatoren, Funktionen, Variablen und Klassen sowie den Grundzügen von in SPl häufig verwendeter objektorientierter Programmierung vertraut sind.

Im Ordner \UModelspl finden Sie die von UModel verwendeten Vorlagen**.** Anhand dieser Dateien können Sie Ihre eigenen Vorlagen entwickeln.

Funktionsweise von Code Generator

Die Basis von Code Generator bilden die Vorlagendateien (.spl) und das von UModel bereitgestellte Objektmodell. Die Vorlagendateien enthalten SPL-Anweisungen zum Erstellen von Dateien, Lesen von Informationen aus dem Objektmodell und Ausführen von Berechnungen, die in der Zielprogrammiersprache auch Literalcodefragmente enthalten.

Die Vorlagendatei wird von Code Generator interpretiert und je nach Vorlage wird eine **.java, .cs-**Quellcodedatei oder eine andere Dateiart ausgegeben.

18.1 Grundlegende SPL-Struktur

Eine SPL-Datei enthält Literaltext, der ausgegeben werden soll, der zwischendurch Code Generator-Anweisungen enthält.

Code Generator-Anweisungen sind in eckige Klammern eingeschlossen '**[**' und '**]**'. Innerhalb einer Klammer können mehrere Anweisungen stehen. Zusätzliche Anweisungen müssen durch eine neue Zeile oder einen Doppelpunkt ':' getrennt werden.

Gültige Beispiele sind:

```
[$x = 42
$x = $x + 1]
```

oder

```
[$x = 42: $x = $x + 1]
```

Hinzufügen von Text zu Dateien

Text, der nicht innerhalb von **[** und **]** steht, wird direkt in die Ausgabedatei geschrieben.
Um eckige Klammern als Literale auszugeben, setzen Sie davor als Excape-Zeichen einen umgekehrten Schrägstrich: **\[** und **\]**. Um einen umgekehrten Schrägstrich auszugeben, verwenden Sie \\.

Kommentare

Kommentare innerhalb eines Anweisungsblocks beginnen immer mit einem **'** Zeichen und enden an der nächsten Zeile oder an einem Zeichen, zum Beenden eines Blocks **]**.

© 2016-2022 Altova GmbH

18.2 Variablen

Jede etwas komplexere SPL-Datei enthält Variablen. Einige Variablen sind vom Code Generator vordefiniert. Sie können jederzeit neue Variablen erstellen, indem Sie diesen Werte zuweisen.

Das **$**-Zeichen wird zur **Deklarierung** oder **Verwendung** einer Variablen verwendet. Vor der Variable muss immer ein **$** stehen. Bei Variablennamen muss die **Groß- und Kleinschreibung beachtet werden.**

Variablentypen:
- Integer - wird auch als Boolescher Wert verwendet, wobei 0 false ist und alle anderen Werte true.
- String
- Objekt - wird von UModel bereitgestellt
- Iterator (siehe foreach[1392]-Anweisung)

Variablentypen werden durch die erste Zuweisung deklariert.

```
[$x = 0]
```

x ist nun ein Integer.

```
[$x = "teststring"]
```

x wird nun als String behandelt.

Strings

String-Konstanten müssen wie im oben gezeigten Beispiel immer innerhalb von doppelten Anführungszeichen stehen. **\n** und **\t** innerhalb von doppelten Anführungszeichen werden als neue Zeile und Tabulator interpretiert, \" ist ein literales doppeltes Anführungszeichen und \\ ist ein umgekehrter Schrägstrich. String-Konstanten können auch mehrere Zeilen einnehmen.

Bei der Verkettung von Strings wird das **&** Zeichen verwendet.

```
[$BasePath = $outputpath & "/" & $JavaPackageDir]
```

Objekte

Objekte stehen für die Informationen, die im UModel-Projekt enthalten sind. Objekte haben **Eigenschaften**, die über den **.** Operator aufgerufen werden können. In SPL können keine neuen Objekte erstellt werden (sie werden durch den Code Generator vordefiniert, aus der Input-Komponente abgeleitet). Sie können Objekte jedoch Variablen zuweisen.

Beispiel:

```
class [=$class.Name]
```

In diesem Beispiel wird das Wort "class" ausgegeben, gefolgt von einem Leerzeichen und dem Wert der Eigenschaft **Name** des Objekts **$class**.

In der folgenden Tabelle sehen Sie die Beziehung zwischen UML-Elementen und ihren SPL-Entsprechungen sowie eine kurze Beschreibung dazu.

Vordefinierte Variablen

UML-Element	SPL-Eigenschaft	Multiplizität	UML-Attribut / Assoziation	UModel-Attribut / Assoziation	Beschreibung
BehavioralFeature	isAbstract		isAbstract:Boolean		
BehavioralFeature	raisedException	*	raisedException:Type		
BehavioralFeature	ownedParameter	*	ownedParameter:Parameter		
BehavioredClassifier	interfaceRealization	*	interfaceRealization:InterfaceRealization		
Class	ownedOperation	*	ownedOperation:Operation		
Class	nestedClassifier	*	nestedClassifier:Classifier		
Classifier	namespace	*		namespace:Package	packages with code language <<namespace>> set
Classifier	rootNamespace	*		project root namespace:String	VB only - root namespace
Classifier	generalization	*	generalization:Generalization		
Classifier	isAbstract		isAbstract:Boolean		
ClassifierTemplateParameter	constrainingClassifier	*	constrainingClassifier		
Comment	body		body:String		
DataType	ownedAttribute	*	ownedAttribute:Property		
DataType	ownedOperation	*	ownedOperation:Operation		
Element	kind			kind:String	
Element	owner	0..1	owner:Element		

© 2016-2022 Altova GmbH

UML-Element	SPL-Eigenschaft	Multiplizität	UML-Attribut / Assoziation	UModel-Attribut / Assoziation	Beschreibung
Element	appliedStereotype	*		appliedStereotype: StereotypeApplication	applied stereotypes
Element	ownedComment	*	ownedComment:Comment		
ElementImport	importedElement	1	importedElement:PackageableElement		
Enumeration	ownedLiteral	*	ownedLiteral:EnumerationLiteral		
Enumeration	nestedClassifier	*		nestedClassifier::Classifier	
Enumeration	interfaceRealization	*		interfaceRealization:Interface	
EnumerationLiteral	ownedAttribute	*		ownedAttribute:Property	
EnumerationLiteral	ownedOperation	*		ownedOperation:Operation	
EnumerationLiteral	nestedClassifier	*		nestedClassifier:Classifier	
Feature	isStatic		isStatic:Boolean		
Generalization	general	1	general:Classifier		
Interface	ownedAttribute	*	ownedAttribute:Property		
Interface	ownedOperation	*	ownedOperation:Operation		
Interface	nestedClassifier	*	nestedClassifier:Classifier		
InterfaceRealization	contract	1	contract:Interface		
MultiplicityElement	lowerValue	0..1	lowerValue:ValueSpecification		
MultiplicityElement	upperValue	0..1	upperValue:ValueSpecification		
NamedElement	name		name:String		
NamedElement	visibility		visibility:VisibilityKind		
NamedElement	isPublic			isPublic:Boolean	visibility <public>

UML-Element	SPL-Eigenschaft	Multiplizität	UML-Attribut / Assoziation	UModel-Attribut / Assoziation	Beschreibung
NamedElement	isProtected			isProtected:Boolean	visibility <protected>
NamedElement	isPrivate			isPrivate:Boolean	visibility <private>
NamedElement	isPackage			isPackage:Boolean	visibility <package>
NamedElement	namespacePrefix			namespacePrefix:String	XSD only - namespace prefix when exists
NamedElement	parseableName			parseableName:String	CSharp, VB only - name with escaped keywords (@)
Namespace	elementImport	*	elementImport:ElementImport		
Operation	ownedReturnParameter	0..1		ownedReturnParameter:Parameter	parameter with direction return set
Operation	type	0..1		type	type of parameter with direction return set
Operation	ownedOperationParameter	*		ownedOperationParameter:Parameter	all parameters excluding parameter with direction return set
Operation	implementedInterface	1		implementedInterface:Interface	CSharp only - the implemented interface
Operation	ownedOperationImplementations	*		implementedOperation:OperationImplementation	VB only - the implemented interfaces/operations
OperationImplementation	implementedOperationOwner	1		implementedOperationOwner:Interface	interface implemented by the operation
OperationImplementation	implementedOperationName			name:String	name of the implemented operation
OperationImplementation	implementedOperationParseableName			parseableName:String	name of the implemented operation with escaped keywords
Package	namespace	*		namespace:Package	packages with code language <<namespace>> set

© 2016-2022 Altova GmbH

UML-Element	SPL-Eigenschaft	Multiplizität	UML-Attribut / Assoziation	UModel-Attribut / Assoziation	Beschreibung
PackageableElement	owningPackage	0..1		owningPackage	set if owner is a package
PackageableElement	owningNamespacePackage	0..1		owningNamespacePackage:Package	owning package with code language <<namespace>> set
Parameter	direction		direction:ParameterDirectionKind		
Parameter	isIn			isIn:Boolean	direction <in>
Parameter	isInOut			isInOut:Boolean	direction <inout>
Parameter	isOut			isOut:Boolean	direction <out>
Parameter	isReturn			isReturn:Boolean	direction <return>
Parameter	isVarArgList			isVarArgList:Boolean	true if parameter is a variable argument list
Parameter	defaultValue	0..1	defaultValue:ValueSpecification		
Property	defaultValue	0..1	defaultValue:ValueSpecification		
RedefinableElement	isLeaf		isLeaf:Boolean		
Slot	name			name:String	name of the defining feature
Slot	values	*	value:ValueSpecification		
Slot	value			value:String	value of the first value specification
StereotypeApplication	name			name:String	name of applied stereotype
StereotypeApplication	taggedValue	*		taggedValue:Slot	first slot of the instance specification
StructuralFeature	isReadOnly		isReadOnly		
StructuredClassifier	ownedAttribute	*	ownedAttribute:Property		
TemplateBinding	signature	1	signature:TemplateSignature		

UML-Element	SPL-Eigenschaft	Multiplizität	UML-Attribut / Assoziation	UModel-Attribut / Assoziation	Beschreibung
TemplateBinding	parameterSubstitution	*	parameterSubstitution:TemplateParameterSubstitution		
TemplateParameter	paramDefault			paramDefault:String	template parameter default value
TemplateParameter	ownedParameteredElement	1	ownedParameteredElement:ParameterableElement		
TemplateParameterSubstitution	parameterSubstitution			parameterSubstitution:String	Java only - code wildcard handling
TemplateParameterSubstitution	parameterDimensionCount			parameterDimensionCount:Integer	code dimension count of the actual parameter
TemplateParameterSubstitution	actual	1	OwnedActual:ParameterableElement		
TemplateParameterSubstitution	formal	1	formal:TemplateParameter		
TemplateSignature	template	1	template:TemplateableElement		
TemplateSignature	ownedParameter	*	ownedParameter:TemplateParameter		
TemplateableElement	isTemplate			isTemplate:Boolean	true if template signature set
TemplateableElement	ownedTemplateSignature	0..1	ownedTemplateSignature:TemplateSignature		
TemplateableElement	templateBinding	*	templateBinding:TemplateBinding		
Type	typeName	*		typeName:PackageableElement	qualified code type names
TypedElement	type	0..1	type:Type		
TypedElement	postTypeModifier			postTypeModifier:String	postfix code modifiers
ValueSpecification	value			value:String	string value of the value specification

Hinzufügen eines Präfixes zu Attributen einer Klasse bei der Codegenerierung

Eventuell müssen Sie allen neuen Attributen in Ihrem Projekt die Zeichen "m_" voranstellen.

Alle neuen Kodierungselemente werden unter Verwendung der SPL-Vorlagen geschrieben:

© 2016-2022 Altova GmbH

Wenn Sie einen Blick in UModelSPL\C#[Java]**Default\Attribute.spl** werfen, können Sie die Schreibweise des Namens ändern, z.B. können Sie

```
write $Property.name
```

durch

```
write "m_" & $Property.name
```

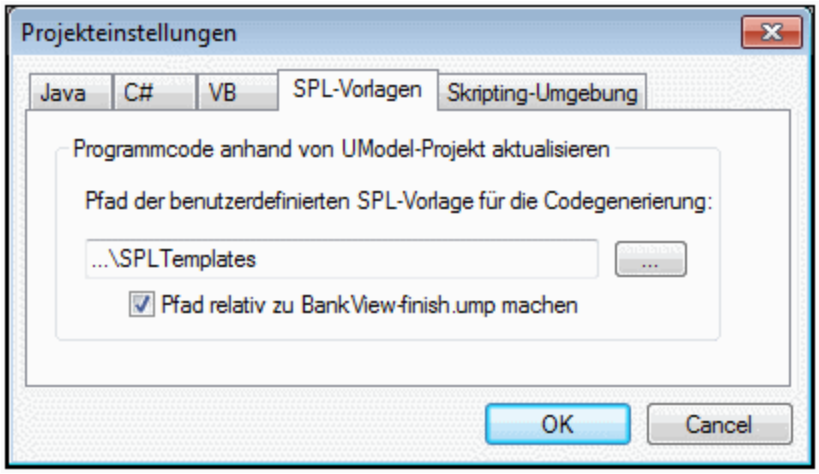
ersetzen.

Es wird unbedingt empfohlen, Ihr Modell sofort nach der Codegenerierung anhand des Codes aktualisieren, um sicherzustellen, dass Code und Modell synchron sind.

Bitte beachten Sie:
Kopieren Sie, wie bereits erwähnt, die SPL-Vorlagen ein Verzeichnis höher hinein (d.h. oberhalb des **Standardverzeichnisses** für **UModelSPL\C#**) bevor Sie sie ändern. Damit stellen Sie sicher, dass sie bei Installation einer neuen Version von UModel nicht überschrieben werden. Stellen Sie bitte sicher, dass Sie im Dialogfeld "Synchronisierungseinstellungen" auf dem Register **Code von Modell** das Kontrollkästchen "Benutzerdefinierte setzt Standardvorlage außer Kraft" aktiviert haben.

SPL-Vorlagen

SPL-Vorlagen können über die Menüoption **Projekt | Projekteinstellungen** (siehe Abbildung unten) pro UModel-Projekt definiert werden. Auch relative Pfade werden unterstützt. Vorlagen, die im angegebenen Verzeichnis nicht gefunden werden, werden im lokalen Standard-Verzeichnis gesucht.



Globale Objekte

$Options	ein Objekt, das globale Optionen enthält:
	generateComments:bool doc-Kommentare generieren (true/false)

$Indent	ein String zum Einrücken von generiertem Code und zur Anzeige der aktuellen Verschachtelungsebene.
$IndentStep	ein String zum Einrücken von generiertem Code und zur Darstellung einer Verschachtelungsebene.
$NamespacePrefix	nur XSD – das Target Namespace Präfix, falls vorhanden

Routinen zur Behandlung von Strings

```
integer Compare(s)
```

Der Rückgabewert gibt die lexikographische Beziehung des String zu s an (unter Berücksichtigung der Groß- und Kleinschreibung):

<0:	der String ist kleiner als s
0:	der String ist identisch mit s
>0:	der String ist größer als s

```
integer CompareNoCase(s)
```

Der Rückgabewert gibt die lexikographische Beziehung des String zu s an (ohne Berücksichtigung der Groß- und Kleinschreibung):

<0:	der String ist kleiner als s
0:	der String ist identisch mit s
>0:	der String ist größer als s

```
integer Find(s)
```

Durchsucht den String nach der ersten Instanz eines Substring s.
Gibt den nullbasierten Index des ersten Zeichens von s oder -1 zurück, wenn s nicht gefunden wird.

```
string Left(n)
```

Gibt die ersten n Zeichen des String zurück.

```
integer Length()
```

Gibt die Länge des String zurück.

© 2016-2022 Altova GmbH

```
string MakeUpper()
```

Gibt einen String zurück, der in Großbuchstaben konvertiert wurde.

```
string MakeUpper(n)
```

Gibt einen String zurück, dessen erste n Zeichen in Großbuchstaben konvertiert wurden.

```
string MakeLower()
```

Gibt einen String zurück, der in Kleinbuchstaben konvertiert wurde.

```
string MakeLower(n))
```

Gibt einen String zurück, dessen erste n Zeichen in Kleinbuchstaben konvertiert wurden.

```
string Mid(n)
```

Gibt einen String zurück, der mit der nullbasierten Indexposition n beginnt

```
string Mid(n,m)
```

Gibt einen String zurück, der mit der nullbasierten Indexposition n beginnt und die Länge m hat

```
string RemoveLeft(s)
```

Gibt einen String zurück, der den Substring s ausnimmt, wenn Left(s.Length()) dem Substring s entspricht.

```
string RemoveLeftNoCase(s)
```

Gibt einen String zurück, der den Substring s ausnimmt, wenn Left(s.Length()) dem Substring s entspricht (Groß- und Kleinschreibung wird ignoriert).

```
string RemoveRight(s)
```

Gibt einen String zurück, der den Substring s ausnimmt, wenn Right(s.Length() dem Substring s entspricht.

```
string RemoveRightNoCase(s)
```

Gibt einen String zurück, der den Substring s ausnimmt, wenn Right(s.Length() dem Substring s entspricht (Groß- und Kleinschreibung wird ignoriert).

```
string Repeat(s,n)
```

Gibt einen String zurück, der den Substring s n mal enthält.

```
string Right(n)
```

Gibt die letzten n Zeichen des String zurück.

18.3 Operatoren

Operatoren in SPL funktionieren wie in den meisten anderen Programmiersprachen.

Liste von SPL-Operatoren absteigend nach Vorrangigkeit gereiht:

.	Objekteigenschaft aufrufen
( )	Gruppierung eines Ausdrucks
true	Boolesche Konstante "true"
false	Boolesche Konstante "false"
&	String-Verkettung
-	Zeichen für negative Zahl
not	Logische Negation
*	Multiplizieren
/	Dividieren
%	Modulo
+	Addieren
-	Subtrahieren
<=	Kleiner oder gleich
<	Kleiner als
>=	Größer oder gleich
>	Größer als
=	Gleich
<>	Nicht gleich
and	Logische Konjunktion (mit short circuit-Auswertung)
or	Logische Disjunktion (mit short circuit-Auswertung)
=	Zuweisung

© 2016-2022 Altova GmbH

18.4 Bedingungen

SPL erlaubt die Verwendung von Standard-"if"-Anweisungen. Die Syntax lautet wie folgt:

```
if condition
	statements
else
	statements
endif
```

oder ohne else:

```
if condition
	statements
endif
```

Beachten Sie bitte, dass für die Bedingung keine runden Klammern verwendet werden!

Wenn in allen anderen Programmiersprachen werden Bedingungen mit logischen und Vergleichsoperatoren[1390] konstruiert.

Beispiel:

```
[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
	whatever you want ['inserts whatever you want, in the resulting file]
[endif]
```

Switch

SPL enthält auch eine Multiple Choice-Anweisung.

Syntax:

```
switch $variable
	case X:
		statements
	case Y:
	case Z:
		statements
	default:
		statements
endswitch
```

Die Case-Bezeichnungen müssen Konstanten oder Variablen sein.

Die switch-Anweisung in SPL fällt nicht durch die Cases (wie in C), daher wird auch keine "break"-Anweisung benötigt.

18.5 Collections und foreach

Collections und Iteratoren

Eine Collection enthält mehrere Objekte - wie ein gewöhnliches Array. Iteratoren lösen das Problem des Speicherns und Inkrementierens von Array Indizes beim Aufrufen von Objekten.

Syntax:

```
foreach iterator in collection
       statements
next
```

Beispiel:

```
[foreach $class in $classes
       if not $class.IsInternal
              ]      class [=$class.Name];
[      endif
next]
```

Beispiel 2:

```
[foreach $i in 1 To 3
      Write "// Step " & $i & "\n"
      ' Do some work
next]
```

Foreach iteriert durch alle Datenelemente in $classes und führt für die Anweisung den Code der auf die Anweisung folgt, bis zur **nächsten** Anweisung aus.

In jeder Iteration wird **$class** dem nächsten Klassenobjekt zugewiesen. Sie arbeiten einfach mit dem Klassenobjekt, anstelle classes[i]->class->Name(), zu verwenden, wie das in C++ der Fall wäre.

Alle Collection-Iteratoren haben die folgenden zusätzlichen Eigenschaften:

Index	der aktuelle Index beginnend mit 0
IsFirst	true, wenn das aktuelle Objekt das erste der Collection ist (Index ist 0)
IsLast	true, wenn das aktuelle Objekt das letzte einer Collection ist

Beispiel:

```
[foreach $enum in $facet.Enumeration
       if not $enum.IsFirst
```


© 2016-2022 Altova GmbH

```
            ], [
      endif
      ]"[=$enum.Value]"[
next]
```

Routinen zur Bearbeitung von Collections

collection **SortByName**(bAscending)

gibt eine Collection zurück, deren Elemente in aufsteigender oder absteigender Reihenfolge (unter Berücksichtigung der Groß- und Kleinschreibung) nach Namen sortiert sind .

collection **SortByNameNoCase**(bAscending)

gibt eine Collection zurück, deren Elemente in aufsteigender oder absteigender Reihenfolge (ohne Berücksichtigung der Groß- und Kleinschreibung) nach Namen sortiert sind .

Beispiel:

```
$SortedNestedClassifier = $Class.nestedClassifier.SortByNameNoCase( true )
```

collection **SortByKind**(bAscending)

gibt eine Collection zurück, deren Elemente in aufsteigender oder absteigender Reihenfolge nach kind-Namen (z.B. “Class”, “Interface”,…) sortiert sind.

collection **SortByKindAndName**(bAscendingKind, bAscendingName)

gibt eine Collection zurück, deren Elemente in aufsteigender oder absteigender Reihenfolge nach "kind" (z.B. “Class”, “Interface”,…) sortiert sind, wobei berücksichtigt wird, ob die "kinds" sortiert in aufsteigender oder absteigender Reihenfolge (unter Berücksichtigung der Groß- und Kleinschreibung) im Namen identisch sind.

collection **SortByKindAndNameNoCase**(bAscending)

gibt eine Collection zurück, deren Elemente in aufsteigender oder absteigender Reihenfolge nach "kind" (z.B. “Class”, “Interface”,…) sortiert sind, wobei berücksichtigt wird, ob die "kinds" sortiert in aufsteigender oder absteigender Reihenfolge (ohne Berücksichtigung der Groß- und Kleinschreibung) im Namen identisch sind.

18.6 Subroutinen

Code Generator unterstützt nun Subroutinen in der Form von Prozeduren oder Funktionen.

Funktionen:
- Übergabe von Werten nach Wert und nach Referenz
- Lokale/globale Parameter (lokal innerhalb von Subroutinen)
- Lokale Variablen
- Rekursiver Aufruf (Subroutinen können sich selbst aufrufen)

18.6.1 Deklaration einer Subroutine

Subroutinen

Syntax-Beispiel:

```
Sub SimpleSub()

       ... lines of code
 EndSub
```

- **Sub** ist das Schlüsselwort, das die Prozedur notiert.
- **SimpleSub** ist der Name, der der Subroutine zugewiesen wurde.
- Runde **Klammern** können eine Parameterliste enthalten.
- Der Code-Block einer Subroutine beginnt direkt nach der geschlossenen Klammer nach dem Parameter.
- **EndSub** notiert das Ende des Code-Blocks.

Bitte beachten Sie:
Eine rekursive oder kaskadierende Subroutine-**Deklaration** ist nicht zulässig, d.h. eine Subroutine darf keine weiteren Subroutinen enthalten.

Parameter

Parameter können auch durch Prozeduren unter Verwendung der folgenden Syntax übergeben werden:

- Alle Parameter müssen Variablen sein
- Variablen muss das **$-**Zeichen vorangestellt werden
- Lokale Variablen werden in einer Subroutine definiert
- Globale Variablen werden explizit außerhalb von Subroutinen deklariert
- Mehrere Parameter werden innerhalb runder Klammern durch Kommas "**,**" voneinander getrennt
- Parameter können Werte übergeben

Parameter - Übergabe von Werten

Parameter können auf zwei Arten übergeben werden: nach Wert oder nach Referenz, wobei die Schlüsselwörter ByVal bzw. ByRef verwendet werden.

© 2016-2022 Altova GmbH

Syntax:

```
' define sub CompleteSub()
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] ...
```

- **ByVal** definiert, dass der Parameter nach Wert übergeben wird. Beachten Sie, dass die meisten Objekte nur nach Referenz übergeben werden können.
- **ByRef** definiert, dass der Parameter nach Referenz übergeben wird. Dies ist die Standardeinstellung, wenn weder ByVal noch ByRef definiert ist.

Funktionsrückgabewerte

Für die Rückgabe eines Werts von einer Subroutine verwenden Sie ein **Return**-Statement. Eine solche Funktion kann von innerhalb eines Ausdrucks aufgerufen werden.

Beispiel:

```
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
  return $localName
else
  return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

18.6.2 Subroutinenaufruf

Verwenden Sie zum Aufrufen einer Subroutine **call**, gefolgt vom Namen der Prozedur und ggf. den Parametern.

```
Call SimpleSub()
```

oder

```
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

Funktionsaufruf

Um eine Funktion (jede Subroutine, die ein **Return**-Statement enthält) aufzurufen, verwenden Sie einfach deren Namen in einem Ausdruck. Verwenden Sie zum Aufrufen von Funktionen nicht das **Call**-Statement.
Beispiel:

```
$QName = MakeQualifiedName($namespace, "entry")
```

19 Lizenzinformationen

Dieser Anhang enthält die folgenden Informationen:

- Informationen über den Vertrieb dieses Software-Produkts
- Informationen zur Software-Aktivierung und Lizenzüberwachung
- die Lizenzvereinbarung zu diesem Software-Produkt

Lesen Sie die Informationen bitte sorgfältig - sie sind rechtlich bindend, da Sie sich bei der Installation dieses Software-Produkts damit einverstanden erklärt haben.

Den Inhalt aller Altova-Lizenzenvereinbarungen finden Sie auf der Altova Website unter Rechtliches.

© 2016-2022 Altova GmbH

19.1 Electronic Software Distribution

Dieses Produkt ist über EDS (Electronic Software Distribution), also auf elektronischem Weg erhältlich, eine Methode, die die folgenden einzigartigen Vorteile bietet:

- Sie können die Software kostenlos 30 Tage lang testen, bevor Sie sich zu einem Kauf entscheiden. (*Anmerkung: Die Lizenz für MobileTogether Designer ist kostenlos*.)
- Wenn Sie sich entschieden haben, die Software zu kaufen, können Sie Ihre Bestellung online auf der Altova Website tätigen. Sie erhalten dann innerhalb weniger Minuten ein vollständig lizenziertes Produkt.
- Sie erhalten immer die neueste Version unserer Software
- Die Software enthält ein umfassendes Hilfesystem, das Sie von der Benutzeroberfläche der Appikation aus aufrufen können. Die neueste Version des Benutzerhandbuchs steht auf unserer Website www.altova.com (i) im HTML-Format zum Aufrufen online und (ii) im PDF-Format zum Download und Ausdrucken zur Verfügung.

30-Tage-Evaluierungszeitraum

Nachdem Sie dieses Software-Produkt heruntergeladen haben, können Sie es 30 Tage lang kostenlos testen. Während dieses Zeitraums werden Sie nach etwa 20 Tagen in regelmäßigen Abständen daran erinnert, dass die Software noch nicht lizenziert wurde. Diese Erinnerungsmeldung wird allerdings nur einmal, nämlich bei jedem Start des Programms, angezeigt. Wenn Sie das Programm nach Ablauf des 30-tägigen Evaluierungszeitraums weiterhin verwenden möchten, müssen Sie eine Produktlizenz erwerben, die Sie in Form einer Lizenzdatei mit einem Keycode erhalten. Laden Sie die Lizenzdatei über das Dialogfeld "Software-Aktivierung" Ihres Produkts hoch, um das Produkt freizuschalten.

Sie können Ihre Produktlizenz über https://shop.altova.com/ erwerben.

Weitergabe der Software an andere Mitarbeiter in Ihrem Unternehmen zu Testzwecken

Wenn Sie die Evaluierungsversion der Software auch anderen Personen in Ihrem Unternehmen über das Netzwerk zur Verfügung stellen möchten oder wenn Sie sie auf einem PC installieren möchten, der nicht mit dem Internet verbunden ist, dürfen Sie nur das Installationsprogramm weitergeben, vorausgesetzt es wurde nicht modifiziert. Jeder, der das von Ihnen zur Verfügung gestellte Installationsprogramm aufruft, muss einen eigenen Evaluierungs-Keycode für 30 Tage anfordern. Nach Ablauf des Testzeitraums, muss eine Lizenz erworben werden, damit das Produkt weiter verwendet werden kann.

19.2 Software-Aktivierung und Lizenzüberwachung

Im Rahmen der Aktivierung der Software durch Altova, verwendet die Software unter Umständen Ihr internes Netzwerk und Ihre Internetverbindung, um die Lizenzdaten während der Installation, Registrierung, der Verwendung oder der Aktualisierung an einen von Altova betriebenen Lizenzserver zu übertragen und die Authentizität der Lizenzdaten zu überprüfen, damit Altova-Software nicht ohne Lizenz oder auf unzulässige Art und Weise verwendet werden kann und um den Kundenservice gleichzeitig zu verbessern. Bei der Aktivierung werden zwischen Ihrem Computer und dem Altova-Lizenzserver für die Lizenzierung erforderliche Daten wie Informationen über Betriebssystem, IP-Adresse, Datum/Uhrzeit, Software-Version und Computername sowie andere Informationen ausgetauscht.

Ihr Altova-Produkt verfügt über ein integriertes Lizenzüberwachungsmodul, das ebenfalls dazu beiträgt, unbeabsichtigte Verletzungen der Lizenzvereinbarung zu vermeiden. Ihr Produkt kann entweder mit einer Einzelplatzlizenz oder einer Mehrfachlizenz erworben werden. Je nach Lizenz stellt das Lizenzüberwachungsmodul sicher, dass nicht mehr als die lizenzierte Anzahl an Benutzern die Applikation gleichzeitig verwendet.

Bei dieser Lizenzüberwachungsmethode wird Ihr LAN-Netzwerk verwendet, um die Kommunikation zwischen Instanzen der Applikation, die auf verschiedenen Computern laufen, zu überwachen.

Einzelplatzlizenz

Beim Start der Applikation wird im Rahmen der Lizenzüberprüfung ein kurzes Broadcast-Datagramm abgesendet, um andere Instanzen des Produkts, die auf anderen Computern im selben Netzwerk laufen, zu finden. Wenn keine Antwort einlangt, wird ein Port geöffnet, der Informationen von anderen Instanzen der Applikation empfangen kann.

Mehrplatzlizenz

Wenn Sie im selben LAN mehrere Instanzen der Applikation verwenden, kommunizieren diese beim Start kurz miteinander, um Keycode-Informationen auszutauschen, damit Sie sicher sein können, dass nicht mehr als die lizenzierte Anzahl an Lizenzen gleichzeitig in Verwendung ist. Dieselbe Lizenzüberwachungstechnologie wird auch bei Unix und vielen anderen Datenbankentwicklungstools verwendet. Sie gestattet Benutzern den Erwerb von Parallellizenzen für mehrere Benutzer zu vernünftigen Preisen.

Wir sind außerdem bestrebt, nur wenige, kleine Netzwerkpakete zu versenden, um Ihr Netzwerk nicht zu überlasten. Die von Ihrem Altova Produkt verwendeten TCP/IP Ports (2799) sind offiziell bei IANA registriert, (*nähere Informationen siehe IANA Service Name Registry*) und unser Lizenzüberwachungsmodul basiert auf einer bewährten und erprobten Technologie.

Wenn Sie eine Firewall verwenden, werden Sie unter Umständen feststellen, dass die Computer, auf denen Altova-Produkte laufen, über Port 2799 miteinander kommunizieren. Sie können diesen Netzwerkverkehr zwischen verschiedenen Gruppen in Ihrem Unternehmen natürlich blockieren, solange Sie mit anderen Mitteln sicherstellen können, dass Ihre Lizenzvereinbarung eingehalten wird.

Anmerkung zu Zertifikaten

Ihre Altova Applikation kontaktiert den Altova Lizenzierungsserver über HTTPS (`link.altova.com`). Für diese Kommunikation verwendet Altova ein registriertes SSL-Zertifikat. Wenn dieses Zertifikat ersetzt wird (z.B. von Ihrer IT-Abteilung oder einer externen Agentur), werden Sie von Ihrer Altova Applikation gewarnt, dass die Verbindung nicht sicher ist. Sie könnten Ihre Altova Applikation mit dem Ersetzungszertifikat starten. Dies

© 2016-2022 Altova GmbH

würde jedoch auf Ihr eigenes Risiko geschehen. Wenn Sie eine Warnung sehen, dass die *Verbindung nicht sicher* ist, überprüfen Sie den Ursprung des Zertifikats und wenden Sie sich an Ihr IT-Team (die in der Lage sein sollten, zu entscheiden, ob das Abfangen und die Ersetzung des Altova-Zertifkats fortgesetzt werden soll).

Wenn Ihr Unternehmen sein eigenes Zertifikat verwenden muss (z.B. um die Kommunikation zu und von Client-Rechnern zu überwachen), empfehlen wir Ihnen, Altova LicenseServer, die kostenlose Lizenzverwaltungssoftware von Altova in Ihrem Netzwerk zu installieren. Client-Rechner verwenden mit dieser Konfiguration weiterhin die Zertifikate Ihres Unternehmens, während der Altova LicenseServer für die Kommunikation mit Altova das Altova-Zertifikat das Altova-Zertifikat verwenden kann.

19.3 Altova Endbenutzer-Lizenzvereinbarung

- Die Altova-Endbenutzer-Lizenzvereinbarung kann unter https://www.altova.com/de/legal/eula eingesehen werden.
- Die Altova-Datenschutzbestimmungen finden Sie unter https://www.altova.com/de/privacy.

© 2016-2022 Altova GmbH

Index

.

.NET 5,
als UModel-Profil, 174
Typen aus Binärdateien importieren, 104
Unterstützung, 13
.NET Core, 13
Assemblys importieren, 230
.NET Framework,
Assemblys importieren, 230
UModel Projekt inkludieren, 174

A

Abgeleitete,
Klasse, 40
Abhängigkeit,
inkludieren, 22
Verwendung, 54
Abhängigkeiten,
anzeigen, 94
Abstrakte,
Klasse, 31
ADO,
als Datenverbindungsschnittstelle, 577
Verbindung einrichten, 583
ADO.NET,
Verbindung einrichten, 589
Akteur,
benutzerdefiniert, 22
Aktiviere Versionskontrolle, 710
Aktivierungsfeld,
Ausführungsspezifikation, 414
Aktivität, 375
Diagramm zu Transition hinzufügen, 375
Diagrammelemente, 363
Operation hinzufügen, 375
Verzweigung / Merge erstellen, 361
zum Zustand hinzufügen, 375
Aktivitätsdiagramm, 357
BPMN, 510
Diagramm SysML, 550
Elemente einfügen, 358
Symbole, 734
Aktualisieren,
Projektdatei, 240
Aktuellste Version holen, 711
Alle,
erweitern/reduzieren, 449
Anbieter wechseln,
Versionskontrolle, 726
Anforderungsdiagramm,
SysML, 549
Annotation,
Text - BPMN, 523
Anpassen, 771
Befehle der Symbolleiste/des Menüs, 771
Ansicht, 764
zu mehreren Instanzen eines Elements, 449
Antwort,
Nachricht automatisch generieren, 420
Anwendungsfall,
hinzufügen, 22
Anzeigen,
Eigenschaft als Assziation, 315
Regionsnamen ausblenden, 383
Applikationsobjekt, 858
Arbeitsablauf,
Projekt, 162
Arbeitsverzeichnis,
Versionskontrolle, 707
Artefakt,
BPMN, 523
Manifest, 60
zu Node hinzufügen, 60
Assoziation,
aggregrieren/zusammensetzen, 31
als Beziehung, 143
anzeigen, 147
BPMN, 520
Eigenschaften ändern, 147
erstellen, 143, 147
Objektverknüpfungen, 46
reflexive Assoziationen, 147
typisierte Eigenschaft anzeigen, 315
Use Case, 22
zwischen Klassen, 31
Assoziationen,

© 2016-2022 Altova GmbH

Assoziationen,
anzeigen, 94
Assoziationsqualifier,
erstellen, 147
Attribut,
Autokomplettierungsfenster, 781
ein-/ausblenden, 449
Farbe, 455
Aufruf,
Nachricht, 420
Aufrufoperation,
einfügen, 358
Aufrufverhalten,
einfügen, 358
Ausblenden,
einblenden - Slot, 449
Auschecken, 714
Auschecken rückgängig, 716
Ausführungsspezifikation,
Lebenslinie, 414
Ausgliedern,
aus Versionskontrolle, 720
Auslösendes Ereignis,
Zeitverlaufsdiagramm, 445
Ausnahme,
Ausnahmeereignis hinzufügen, 449
Ausnahmeereignis,
hinzufügen, 449
Ausrichten,
beim Ziehen an Hilfslinien ausrichten, 781
Elemente beim Ziehen, 22
Ausrichtungshilfslinien, 22
Außer Kraft setzen,
SPL-Standardvorlagen, 240
Austrittspunkt,
zu Unterautomat hinzufügen, 383
Autokomplettierung,
Fenster zum Bearbeiten von Klassen, 781
Funktion, 31
Autokomplettierung von Datentypen,
auslösen, 140
deaktiviere, 140
Automatisch generieren,
Antwortnachricht, 420

B

Ball and socket,
Interface notation, 449
Basis,
Klasse, 40
Basisklasse,
überschreiben, 449
Batch-Modus,
Projekte erstellen, 109
Projekte laden, 109
Projekte speichern, 109
Bearbeiten (Menü),
Befehle, 757
Bearbeitungen rückgängig machen, 716
Bedingter Fluss, 520
Befehle,
zur Symbolleiste/zum Menü hinzufügen, 771
Befehlszeile,
Binärtypen importieren, 104
Code und Modell synchronisieren, 104
Programmcode generieren, 104
Projekte erstellen, 109
Projekte laden, 109
Projekte speichern, 109
Quellcode importieren, 104
Referenz, 104
Benutzerdefinierte,
SPL-Vorlagen, 240
Benutzerdefinierter,
Akteur, 22
Benutzer-DSN,
einrichten, 596
Benutzeroberfläche,
mittels Plug-in konfigurieren, 849
Bereich,
einzelne/mehrere erweitern, 449
Beziehungen,
Abhängigkeit, 143
Aggregation, 143
anzeigen, 146
Assoziation, 114, 143
Generalisierung, 114, 143
Komposition, 143
Realisierung, 143

© 2016-2022 Altova GmbH

Beziehungen,
Stil ändern, 144
Bilder,
als Elementhintergrund verwenden, 126
Binärdateien,
ins Modell importieren, 225
BPMN, 509
1.0 in 2.0 konvertieren, 509
Artefakte, 523
Assoziation, 520
BPMN 2.0,
Ereignisse, 510
Flussobjekte, 510
Tasks, 510
Business Process Modeling Notation, 509
icons, 750

C

C#,
Attribute importieren, 227
automatisch implementierte Eigenschaften, 187
Binärdateien importieren, 225, 230
Code generieren, 180, 187
Codegenerierungsoptionen, 186
Fehlerbehandlung, 874
Optionen für den Code-Import, 212
Quellcode importieren, 209
C#-Modell,
in Java konvertieren, 329
C++,
Code generieren, 202
Fehlerbehandlung, 874
Quellcode importieren, 209, 211
Reverse Engineering, 211
Call-Nachricht,
gehe zu Operation, 420
Classifier,
einschränken, 313
neu, 241
umbenennen, 241
Code, 243
Code zu Sequenzdiagramm hinzufügen, 437
Java-Code und Klassendateinamen, 243
Refactoring, 243
Sequenzdiagramm generieren, 426, 864
Sequenzdiagramm manuell generieren, 865
SPL, 1379
Standard, 781
Synchronisierung, 240
von Sequenzdiagramm generieren, 434
Code Engineering,
Assoziatonen auflösen, 150
Code zu Modell, 75
Fehler, 99
Informationsmeldungen, 99
Komponentenrealisierungen generieren, 241
Modell zu Code, 65
Projektdatei in neuen Ordner verschieben, 162
Tutorial-Beispiele, 18
Warnungen, 99
Collection-Assoziation,
erstellen, 150
in Collection-Vorlagen auflösen, 150
Voraussetzungen, 150
COM API,
im Skript-Editor, 820
Combined Fragment, 416
Continuous Flow,
Modellieren / Streaming der SysML-Aktivität, 550
Copyright-Informationen, 1396
CR/LF,
für ump-Datei beim Speichern, 162

D

Datei,
Projektdateien zusammenführen, 307
ump, 162
von URL öffnen, 755
Datei (Menü),
Befehle, 755
Datei abrufen,
Versionskontrolle, 711
Datei holen,
Versionskontrolle, 711
Datei-DSN,
einrichten, 596
Datenbank,
anhand des Modells aktualisieren, 572
für das Round-Trip Engineering konfigurieren, 570
in UModel importieren, 555, 558

© 2016-2022 Altova GmbH

Datenbank,
mit UModel modellieren, 557
Datenbankmodell,
in eine andere Datenbankart konvertieren, 335
Datenbanktreiber,
Übersicht, 580
Datenbankverbindung,
Assistenten starten, 578
Beispiele einrichten, 607
einrichten, 577
Datenobjekt,
BPMN, 523
Deaktiviere Versionskontrolle, 710
Deployment,
Diagramm, 60
Symbole, 739
Deployment-Diagramm, 468
Diagram,
- Use Case-Diagramm, 402
Diagramm, 469, 781
- Aktivitätsdiagramm, 357
- Interaktionsübersichtsdiagramm, 406
- Klassendiagramm, 449
Aktivität zu Transition hinzufügen, 375
als png speichern, 755
BPMN, 509
Code von Sequenzdiagramm generieren, 434
Code zu Sequenzdiagramm hinzufügen, 437
Datenbank - importieren, 555
Deployment-Diagramm, 468
Elemente aus inkludierten Dateien igrnoieren, 781
Elemente einfügen, 114
Kommunikationsdiagramm, 402
Komponentendiagramm, 468
Kompositionsstrukturdiagramm, 465
mehrere Klasseninstanzen, 449
nicht verwendete Elemente suchen, 120
Objektdiagramm, 469
offene Diagramme mit Projekt speichern, 781
Paketabhängigkeiten erstellen, 469
Paktediagramm, 469
Schnellbildlauf, 96
Sequenzdiagramm, 411
Sequenzdigarmme SysML, 551
Stile, 93
Symbole, 733
Symbolreferenz, 86
Übersicht anzeigen, 96
Use Case SysML, 553
XML-Schema, 490
Zeitverlaufsdiagramm, 440
zu Favoriten hinzufügen, 91
Zusätzliches - XML-Schema, 490
Zustandsdiagramm, 374
Zustandsdiagramm SysML, 552
Diagramm - Sequenz,
manuell anhand von Code generieren, 865
Sequenzdiagramm anhand von Code generieren, 865
von Code generieren, 864
Diagramme, 356
an Fenstergröße anpassen, 142
aus Projekt löschen, 134
Aussehen ändern, 134
Ebenen hinzufügen, 138
erstellen, 101, 129
generieren, 130
Größe ändern, 134
in einem Projekt anzeigen, 90
öffnen, 133
Strukturdiagramme, 449
über das Fenster "Hierarchie" generieren, 94
vergrößern/verkleinern, 142
Verhaltensdiagramme, 357
Diagramm-Struktur (Fenster), 90
Diagrammtyp,
identifizieren, 101
Dokumentation,
anhand von UML-Projekt generieren, 345
aus Quellcode importieren, 125
Quellcode mit Dokumenation generieren, 125
zu Elementen hinzufügen, 125
Dokumentation (Fenster), 97
Download Versionskontroll-Projekt, 707
Druckvorschau,
Optionen, 755
Durchsuchen,
Diagramme, 118
Elemente, 118
Text, 118

E

Ebene (Fenster), 98
Ebenen,

© 2016-2022 Altova GmbH

Ebenen,
anzeigen, 138
ausblenden, 138
löschen, 138
sperren, 138
zu Diagrammen hinzufügen, 138
Eigenschaft,
als Lebenslinie typisiert, 414
Farbe, 455
typisierte - anzeigen, 315
wiederverwenden, 40
Eigenschaften,
hinzufügen, 31
Versionskontrolle, 725
Eigenschaften (Fenster),
benutzerdefinierte Eigenschaften hinzufügen, 92
Eigenschaftswerte,
als Enumerationen, 479, 482
anzeigen oder ausblenden, 159
Beispiel, 482
Beispiele, 156
Definition, 156
erstellen, 157, 479
Ein-/ausblenden,
Attribute, Operationen, 449
Einblenden,
ausblenden - Slot, 449
Einchecken, 715
Einfügen, 358
Aktion (Aufrufoperation), 358
Aktion (Aufrufverhalten), 358
einfachen Zustand, 375
Interaktionsübersichtselemente, 407
Kompositionsstrukturelemente, 466
Paketdiagrammelemente, 471
Zeitverlaufsdiagrammelemente, 441
Einführung, 12
Einschränken,
Classifier, 313
Einstelllungen,
Synchronisierung, 240
Einstellungen,
Versionskontrolle, 781
Eintrittspunkt,
zu Unterautomat hinzufügen, 383
Element,
Stile, 93
zu Favoriten hinzufügen, 91
Elemente,
aus Diagramm löschen, 117
aus Include-Dateien ignorieren, 781
aus Projekt löschen, 117
Aussehen ändern, 126
automatisches Layout, 136
benutzerdefinierte Bilder anwenden auf, 126
dokumentieren, 97, 125
Eigenschaften ändern, 92
einschränken, 121
ersetzen, 118
Größe anpassen, 136
Hyperlinks erstellen, 122
in einem Diagramm ausrichten, 136
in einem Diagramm suchen, 120
kopieren, 116
suchen, 118
umbenennen, 116
verschieben, 116
zu einem Diagramm hinzufügen, 114
zum Modell hinzufügen, 86, 113
Zustandsdiagramm einfügen, 375
zwischen Ebenen verschieben, 138
Elementimport,
anzeigen, 94
Endbenutzer-Lizenzvereinbarung, 1396, 1400
Enthältbeziehung,
in einem Diagramm zeichnen, 153
Ereignis,
BPMN, 510
Erstellen,
getter / setter Methoden, 449
Erweitern,
alle Klassenbereiche, 449
Erweiterter,
Subprozess, 517
Evaluierungszeitraum,
für Altova-Software-Produkte, 1397
von Altova Software-Produkten, 1396
Exportieren,
UModel-Projekte in XMI, 663
Externe Applikationen,
von UModel aus öffnen, 774
Extras, 765
Optionen, 781
Extras (Menü),
benutzerdefinierte Befehle hinzufügen, 774

© 2016-2022 Altova GmbH

F

Farbe,
Syntaxfärbung - aktivieren /deaktivieren, 455
Favoriten (Fenster),
entfernen aus, 91
hinzufügen zu, 91
Fehler,
beim Code Engineering, 99
Fehlerbehandlung,
allgemeine Beschreibung, 874
Fenster,
Standardeinstellungen wiederherstellen, 765
Firebird,
über ODBC verbinden, 608
Verbindung über JDBC, 607
Fluss, 520
bedingter, 520
Nachrichtenfluss, 520
Sequenzfluss, 520
Standard, 520
Flussobjekte, 510
Forward Engineering, 65
Freigeben,
aus Versionskontrolle, 721

G

Gate,
Sequenzdiagramm, 419
Gateways,
BPMN 2.0, 510
Datenbasierter exklusiver Gateway (XOR), 510
Ereignisbasierter exklusiver Gateway (XOR), 510
Inklusiver Gateway (OR), 510
Komplexer Gateway (Entscheidung/Zusammenfluss), 510
Paralleler Gateway (AND), 510
Gehe zu,
Lebenslinie, 414
Generalisieren,
spezialisieren, 40
Generalisierung,
als Beziehung, 114, 143
erstellen, 143
Generalisierungen,
anzeigen, 94
Generated documentation,
options, 349
Generics,
Unterstützung für Java und C#, 313
Generieren,
Antwortnachricht automatisch generieren, 420
Komponentenrealisierungen automatisch generieren, 241
Sequenzdiagramm anhand von Code, 864
Sequenzdiagramm von Kommunikationsdiagramm, 403
UML-Projektdokumentation, 345
Geschwindigkeit,
Verbesserung, 179
Get,
getter / setter Methoden, 449
Gruppe,
BPMN, 523

H

Hierarchie (Fenster), 94
Hierarchiediagramm,
in der Dokumentation angezeige Ebenen, 345
Hilfe (Menü),
Befehle, 795
Hilfslinie,
beim Ziehen mit der Maus, 781
Hinzufügen, 717
Diagramm zu Paket, 22
neues Projekt, 162
Paket zu Projekt, 22
Projekt in Versionskontrolle, 717
Versionskontrolle, 717
HRESULT,
und Fehlerbehandlung, 874
Hyperlinks,
im Dokumentationstext, 125

I

IBM DB2,
über JDBC verbinden, 611

© 2016-2022 Altova GmbH

IBM DB2,
über ODBC verbinden, 613
IBM DB2 für i,
über ODBC verbinden, 620
Verbinden über JDBC, 619
IBM Informix,
über JDBC verbinden, 623
Icon,
Business Process Modeling Notation, 750
Ignorieren, 781
Elemente in Liste, 781
Verzeichnisse, 781
Import,
SQL-Datenbank, 555
Importieren,
XMI in UModel, 663
Inkludieren,
.NET Framework, 174
Abhängigkeit, 22
UModel Projekt, 174
Instanz,
Diagramm, 46
mehrere Klassen, Anzeige, 449
Objekt, 46
Intelligentes,
Autokomplettieren, 31
Interaction Use, 419
Interaktinosoperand, 416
Interaktionsoperand,
mehrzeiliger, 416
Interaktionsoperator,
definieren, 416
Interaktionsübersicht,
Elemente einfügen, 407
Interaktionsübersichtsdiagramm, 406
Symbole, 740
Interface,
ball and socket, 449

J

Java,
Annotationen importieren, 227
Binärdateien importieren, 233
Code generieren, 180, 193
Code und Klassendateinamen, 243
Codegenerierungsoptionen, 186
Optionen für den Code-Import, 212
Quellcode importieren, 209
Java-Modell,
in C++ konvertieren, 322
JavaScript,
Fehlerbehandlung, 874
JDBC,
als Datenverbindungsschnittstelle, 577
mit Teradata verbinden, 656
Verbindung einrichten (Windows), 599
JScript,
Skripterstellung mit UModel, 809

K

Katalog,
Datei - XMLSpy Katalogdatei, 781
Klasse, 40
abgeleitete, 40
abstrakte und konkrete, 31
Assoziationen, 31
Autokomplettierungsfenster aktivieren, 781
ball and socket interface, 449
Basis, 40
Basisklasse überschreiben, 449
Diagramme, 31
Eigenschaften hinzufügen, 31
erweitern, Bereiche reduzieren, 449
in Komponentendiagramm, 54
mehrere Instanzen in einem Diagramm, 449
Namensänderungen - Synchronisierung, 243
Operation - überschreiben, 449
Operationen hinzufügen, 31
Symbole, 735
Synchronisierung, 240
Syntaxfärbung, 455
Klassendiagramm, 449
neues hinzufügen, 31
Klassennamen ändern,
Auswirkung auf Codedateinamen, 243
Knoten,
Stile, 93
Kollaboration,
Kompositionsstrukturdiagramm, 466
Kommunikationsdiagramm, 402

© 2016-2022 Altova GmbH

Kommunikationsdiagramm, 402
Symbole, 736
von Sequenzdiagramm generieren, 403
Kompatibilität,
Projekte aktualisieren, 240
Komponente,
Diagramm, 54
Klasse einfügen, 54
Realisierung, 54
Symbole, 738
Komponentenansicht,
als Paket, 116
Komponentendiagramm, 468
Komponentenrealisierung,
automatische Generierung, 241
Komposition,
Assoziation - erstellen, 31
Kompositionsstruktur,
Elemente einfügen, 466
Kompositionsstrukturdiagramm, 465
Symbole, 737
Konkrete,
Klasse, 31
Konvertieren,
BPMN 1.0 in 2.0, 509

L

Layout (Menü),
Befehle, 762
Lebenslinie, 414
allgemeiner Wert, 441
Attribute, 414
Eigenschaft vom Typ Lebenslinie, 414
Gehe zu, 414
Linie,
orthogonal, 54
Linien,
benutzerdefinierte, 144
formatieren, 46
gerade, 144
Hilfslinien, 781
orthogonale, 144
Stil ändern, 144
verschieben, 144
Links,
in generated documentation, 349
Lizenz, 1400
Informationen, 1396
Lizenzüberwachung,
in Altova-Produkten, 1398
Lokales Projekt, 707
Löschen, 771
Befehl aus der Symbolleiste löschen, 771
Symbol aus der Symbolleiste, 771
Symbolleiste, 773

M

Mail,
Projekt senden, 755
Makros,
aktivieren, 822, 834
ausführen, 835
entwickeln, 809, 816
Manifest,
Artekfakt, 60
Manuell generieren,
Sequenzdiagramm von Code, 865
MariaDB,
über ODBC verbinden, 625
Mehrzeilig,
Use Case, 22
Mehrzeiliger,
Akteurtext, 22
Interaktionsoperand, 416
Meldungen (Fenster),
Referenz, 99
Menü,
Ansicht, 764
Befehl hinzufügen/löschen, 771
Extras, 765
Merge,
in Aktivität erstellen, 361
Methode,
Ausnahmeereignis hinzufügen, 449
Methoden,
getter / setter, 449
Microsoft Access,
über ADO verbinden, 583, 627
Microsoft SQL Server,
über ADO verbinden, 629

© 2016-2022 Altova GmbH

Microsoft SQL Server,
über ODBC verbinden, 632
Modell,
Elemente hinzufügen, 86, 113
in eine andere Sprache transformieren, 317
Klassennamen ändern - Auswirkung in Java, 243
Modellieren,
Performance verbessern, 179
Modell-Struktur (Fenster),
Einträge erweitern oder reduzieren, 86
Modellelemente anzeigen oder ausblenden, 86
Modellelemente sortieren, 86
Projekt anzeigen, 86
Symbolreferenz, 86
MySQL,
über ODBC verbinden, 638

N

Nachricht, 420
Aufruf, 420
einfügen, 420
gehe zu Operation, 420
nummerieren, 420
Objekt erstellen, 420
Pfeile, 420
verschieben, 420
Zeitverlaufsdiagramm, 447
Nachrichtenfluss, 520
Nachrichtenpfeile verschieben, 420
Name,
Regionsnamen - ein- /ausblenden, 383
Native Oberfläche ausführen, 726
Neu,
Classifier, 241
Neue Zeile,
in Lebenslinie, 403
Interaktionsoperand, 416
Node,
Artefakt hinzufügen, 60
hinzufügen, 60
Nummerieren,
Nachrichten, 420

O

Objekt,
Diagramm, 46
Nachricht erstellen, 420
Symbole, 741
Verknüpfungen - Assoziationen, 46
Objektdiagramm, 469
Objektmodell,
Übersicht, 858
ODBC,
als Datenverbindungsschnittstelle, 577
mit MariaDB verbinden, 625
mit Teradate verbinden, 657
Verbindung einrichten, 596
ODBC-Treiber,
Verfügbarkeit überprüfen, 596
Öffnen,
Projekt-Versionskontrolle, 707
OLE DB,
als Datenverbindungsschnittstelle, 577
OpenJDK,
als Java Virtual Machine, 599
Binärdateien importieren, 226
Operand,
Interaktion, 416
Operation,
Autokomplettierungsfenster, 781
automatisch zu Aktivität hinzufügen, 375
ein-/ausblenden, 449
Farbe, 455
gehe zu von Call-Nachricht, 420
überschreiben, 449
Vorlage, 315
wiederverwenden, 40
Operation automatisch hinzufügen, 375
Operationen,
hinzufügen, 31
Operator,
Interaktion, 416
Optionen, 781
Extras, 781
Versionskontrolle, 781
Options,
when generating documentation, 349

© 2016-2022 Altova GmbH

Oracle-Datenbank,
über JDBC verbinden, 640
über ODBC verbinden, 642
Ordner,
in Versionskontrolle abrufen, 712
Ordner abrufen,
Versionskontrolle, 712
Orthogonale,
Linie, 54
Orthogonaler,
Zustand, 383

P

Paket,
Standardpakete, 86
Symbolreferenz, 86
Paketdiagramm, 469
Abhängigkeiten erstellen, 469
Elemente einfügen, 471
Symbole, 742
SysML, 547
Paketimport,
anzeigen, 94
Paketmerge, 471
Paketzusammenführung,
anzeigen, 94
Pakteimport, 471
Parameter,
Vorlage, 315
Per E-Mail senden,
Projekt, 755
Performance,
Verbesserung, 179
Pfad,
Projekt verschieben, 162
Projektordner ändern, 162
SPL-Vorlagenpfad, 1381
Pool,
Swimlane, 522
PostgreSQL,
nativ verbinden, 604
über ODBC verbinden, 647
Pretty Print,
in exportierten XMI-Dateien, 663
Projekt mit Pretty Print speichern, 162
Profile,
auf ein Paket anwenden, 170, 477
Definition, 476
erstellen, 477
vordefinierte, 477
Progress OpenEdge-Datenbank,
über ODBC verbinden, 651
verbinden über JDBC, 649
Projekt, 781
3-Weg-Zusammenführung, 308, 310
anzeigen, 86
Arbeitsablauf, 162
Datei - aktualisieren, 240
Dokumentation generieren, 345
entfernen aus Versionskontrolle, 720
erstellen, 162
Hinzufügen in Versionskontrolle, 717
in Module aufteilen, 170
in Unterprojekte aufteilen, 170
letzten beim Starten öffnen, 781
Modellelemente hinzufügen oder entfernen, 86
offene Diagramme speichern, 781
Paket einfügen, 162
per E-Mail senden, 755
speichern - Pretty Print, 162
Standardcode, 781
Stile, 93
UModel Projekt inkludieren, 174
verschieben, 162
zusammenführen, 307
Projekt (Menü),
Befehle, 759
Projekt öffnen,
Versionskontrolle, 707
Projektsyntax,
überprüfen, 99
Projektzusammenführung,
3-Weg, 308
Provider,
auswählen, 707
Prozess,
erweiterter Subprozess, 517
reduzierter Subprozess, 518

© 2016-2022 Altova GmbH

R

Raster,
Elemente an Hilfslinien ausrichten, 22
Hilfslinien, 781
Realisierung,
Komponente, 54
Komponentenrealisierungen generieren, 241
Rechtliches, 1396
Rechtschreibung,
prüfen, 97
Reduzieren,
Klassenbereiche, 449
Reduzierter,
Subprozess, 518
Refactoring,
Klassennamen - Synchronisierung, 243
Referenz, 754
Region,
zu zusammengesetztem Zustand hinzufügen, 383
Regionsname,
ein- / ausblenden, 383
Reverse Engineering, 75
C++, 211
Root,
als Paket, 116
Katalog - XMLSpy, 781
Paket/Klasse synchronisieren, 240

S

SC,
Syntaxfärbung, 455
Schaltflächen,
Sichtbarkeitsschaltflächen, 449
Schnittstelle,
implementieren, 449
Sequenz,
Diagramm SysML, 551
Symbole, 745
Sequenzdiagramm, 411
Code generieren, 434
Code hinzufügen, 437
Combined Fragment, 416
Elemente einfügen, 412
Gate, 419
Interaction Use, 419
Lebenslinie, 414
manuell anhand von Code generieren, 865
Nachrichten, 420
von Code generieren, 864
von Kommunikationsdiagarmm generieren, 403
Zustandsinvariante, 420
Sequenzdiagramme,
anhand von Getter / Setter generieren, 432
anhand von Quellcode generieren, 426
mehrere generieren, 432
Sequenzfluss, 520
Set,
getter / setter Methoden, 449
Shortcuts,
löschen, 777
zuweisen, 777
Sichtbarkeit,
Schaltflächen - auswählen, 449
Signatur,
Vorlage, 313, 314
Skript-Editor,
Übersicht, 809, 811
Slot,
ein-/ausblenden, 449
Software-Produktlizenz, 1400
Speichern,
Diagramm als Bild, 755
Spezialisieren,
generalisieren, 40
SPL, 1379
benutzerdefinierte, 240
Codeblöcke, 1380
foreach, 1392
SPL-Vorlagen,
Vorlagenpfad, 1381
SQL,
in UModel importieren, 555
SQL Server,
über ADO verbinden, 583
über ADO.NET verbinden, 589
über JDBC verbinden, 599
Standard,
Projektcode, 781
SPL-Vorlagen, 240

© 2016-2022 Altova GmbH

Standardfluss, 520
Standpunkte,
Paketdiagramm SysML, 547
Starten,
mit letzten Projekt, 781
Status aktualisieren,
Versionskontrolle, 726
Stereotype,
auf Elemente anwenden, 157, 482
Beispiel, 482
Beispiele, 155, 476
benutzerdefinierte Stile hinzufügen, 487
benutzerdefinierte Symbole hinzufügen, 487
Definition, 155
erstellen, 479, 482
zum Fenster "Eigenschaften" hinzufügen, 92
Stile,
auf Diagramme anwenden, 134
auf Elemente anwenden, 126
auf Linien anwenden, 144
kaskadierend, 134
kaskadierende, 126, 144
Vorrangigkeit, 126, 134, 144
Stile (Fenster), 93
STL-Datentypen,
zum Diagramm hinzufügen, 202
Strukturdiagramme,
Diagramme, 449
StyleVision,
generierte Dokumentation anpassen mit, 345, 354
Subprozess,
erweiterter, 517
reduziert, 518
Suchen,
Diagramme, 118
Elemente, 118
Text, 118
Swimlane,
Pool, 522
Sybase,
über JDBC verbinden, 654
Symbol, 739
Aktivitätsdiagramm, 734
Deployment, 739
groß anzeigen, 781
Klasse, 735
Komponente, 738
Objekt, 741
Sequenz, 745
Verteilung, 739
zur Symbolleiste/zum Menü hinzufügen, 771
Symbole,
Interaktionsübersichtsdiagramm, 740
Kommunikationsdiagramm, 736
Kompositionsstrukturdiagramm, 737
Paketdiagramm, 742
Use Case, 748
XML-Schemadiagramm, 749
Zeitverlaufsdiagramm, 747
Zustandsdiagramm, 746
Symbolleiste,
aktivieren/deaktivieren, 773
Befehl hinzufügen, 771
große Symbole anzeigen, 781
neue erstellen, 773
Standardeinstellungen wiederherstellen, 765
Symbolleiste & Menübefehle zurücksetzen, 773
Synchronisieren,
mit neuem Ordner, 162
Root/Paket/Klasse, 240
Synchronisierung, 243
Änderung von Klassennamen, 243
Einstellungen, 240
Klasse und Codedateinamen, 243
Syntaxfärbung, 455
SysML,
Aktivitätsdiagramm, 550
Anforderungsdiagramm, 549
Blockdefinitionsdiagramm, 538
Diagramme erstellen, 536
Einführung, 536
Internes Blockdiagramm, 541
Paketdiagramm, 547
Sequenzdiagramm, 551
Use Case-Diagramm, 553
Zusicherungsdiagramm, 546
Zustandsdiagramm, 552
System-DSN,
einrichten, 596
Systemhierarchie,
Paketdiagramm SysML, 547

© 2016-2022 Altova GmbH

T

Tastaturkürzel,
in Tooltipp anzeigen, 781
löschen, 777
zuweisen, 777
Teradata,
über JDBC verbinden, 656
über ODBC verbinden, 657
Textannotation,
BPMN, 523
Tick-Symbol,
Zeitverlaufsdiagramm, 444
Tooltipp,
anzeigen, 781
Tastaturkürzel anzeigen, 781
Transformation,
Einstellungen, 320
Transition, 375
Aktivitätsdiagramm hinzufügen, 375
Trigger definieren, 375
zwischen Zuständen definieren, 375
Trigger,
Transitions-Trigger definieren, 375
Tutorial,
Beispieldateien, 18
Typ,
Eigenschaft - anzeigen, 315
Typisieren,
Eigenschaft - als Lebenslinie, 414

U

Überschreiben,
Klassenoperationen, 449
Übersicht (Fenster),
scrollen, 96
Umbenennen,
Classifier, 241
UML,
Diagramme, 356
Sichtbarkeitsschaltflächen, 449
Variablen, 1381
Vorlagen, 313
UModel,
Einführung, 13
Hauptfunktionen, 13
UModel API,
Übersicht, 857
UModel Diagrammsymbole, 733
UModel Einführung, 12
UModel Plug-in für Visual Studio,
installieren, 667
UModel Plug-In-Bibliothek,
Referenz darauf hinzufügen, 838
UModel-Projekte,
öffnen, speichern, erstellen, 19
UModel-Typbibliothek,
Referenz darauf hinzufügen, 840
Ump,
Dateierweiterung, 162
Projektverzeichnis ändern, 162
Unterautomatenzustand,
neuen Eintrtts-/Austrittspunkt hinzufügen, 383
Unterprojekt,
anhand eines Hauptprojekts erstellen, 170
wieder in das Hauptprojekt integrieren, 170
Unterschiede anzeigen, 724
URL,
Datei öffnen von, 755
Use Case,
Assoziation, 22
Bereiche, 22
Diagramm SysML, 553
hinzufügen, 22
mehrzeilig, 22
Symbole, 748
Use Case diagram, 402

V

Variablen,
UML, 1381
VB.NET,
Binärdateien importieren, 225
Code generieren, 180
Codegenerierungsoptionen, 186
Optionen für den Code-Import, 212
Quellcode importieren, 209

© 2016-2022 Altova GmbH

VBScript,
Skripterstellung mit UModel, 809
Verbessern,
Performance, 179
Verbindungsobjekte, 520
Verhaltensdiagramme, 357
Verlauf,
anzeigen, 722
Verlauf anzeigen, 722
Verschieben,
Projekt, 162
Versionsdateien vergleichen, 724
Versionskontrolle,
aktivieren /deaktivieren, 710
aktuellste Version holen, 711
auschecken, 714
Auschecken rückgängig, 716
ausgliedern, 720
Befehle, 707
Datei abrufen, 711
Eigenschaften, 725
Einchecken, 715
Hinzufügen in Versionskontrolle, 717
installieren eines Versionskontroll-Plug-in, 702
native Oberfläche ausführen, 726
Optionen / Einstellungen, 781
Projekt öffnen, 707
Status aktualisieren, 726
Unterschiede anzeigen, 724
Verlauf anzeigen, 722
Versionskontrollsystem wechseln, 726
Verteilung,
Symbole, 739
Vertrieb,
von Altova Software-Produkten, 1396
von Altova-Software-Produkten, 1397
Verwendung,
Abhängigkeit, 54
Vorlage, 315
Verzeichnis,
beim Zusammenführen igrnoieren, 781
Projektverzeichnis ändern, 162
Verzweigung,
in Aktivität erstellen, 361
Visual Basic,
Fehlerbehandlung, 874
Visual Studio,
addin UModel support to solutions, 668
automatische Synchronisierung von Code und Modell, 673
Code mit Modell synchronisieren, 673
UModel als Plug-in integrieren, 665
UModel-Projekte laden/entladen, 672
Vorlage,
Signatur, 313, 314
Verwendung, 315
Vorlagen,
benutzerdefinierte SPL, 240
Operation/Parameter, 315
SPL-Vorlagen, 1381

W

Warnungen,
beim Code Engineering, 99
Wertverlaufslinie,
Zeitverlaufsdiagramm, 441
Wiederherstellen,
Symbolleisten und Fenster, 765

X

XMI,
importieren und exportieren, 663
XML-Schema,
anhand von Modell generieren, 500
Diagramme, 490
Diagramme erstellen, 497
ins Modell importieren, 491
modellieren, 497, 500
Namespace deklarieren, 497
XML-Schemadiagramm,
Symbole, 749

Z

Zeilenumbruch,
im Akteurtext, 22
Zeitbedingung,
Zeitverlaufsdiagramm, 447
Zeitdauerbedingung,
Zeitverlaufsdiagramm, 446

© 2016-2022 Altova GmbH

Zeitlinie,
Zustandsänderungen definieren, 441
Zeitverlaufsdiagramm, 440, 441
auslösendes Ereignis, 445
Elemente einfügen, 441
Lebenslinie, 441
Nachricht, 447
Symbole, 747
Tick-Symbol, 444
wechseln zwischen Typen, 441
Wertverlaufslinie, 441
Zeitbedingung, 447
Zeitdauerbedingung, 446
Zeitlinie, 441
Zurücksetzen,
Symbolleiste & Menübefehle, 773
Zusammenführen,
Projekte, 307
Verzeichnis ignorieren, 781
Zusammenführung,
3-Weg-Projektzusammenführung, 308, 310
Zusammengesetzter Zustand,
Region hinzufügen, 383
Zustand, 375
Aktivität hinzufügen, 375
einfachen einfügen, 375
orthogonaler, 383
Transition definieren, 375
Unterautomatenzustand, 383
zusammengesetzter, 383
Zustandsänderungen,
auf einer Zeitlinie definieren, 441
Zustandsdiagramm,
Diagramm SysML, 552
Diagrammelemente, 395
Elemente einfügen, 375
Symbole, 746
zusammengesetzte Zustände - Regionen, 383
Zustände, Aktivitäten, Transitionen, 375
Zustandsdiragramm, 374
Zustandsinvariante, 420

© 2016-2022 Altova GmbH