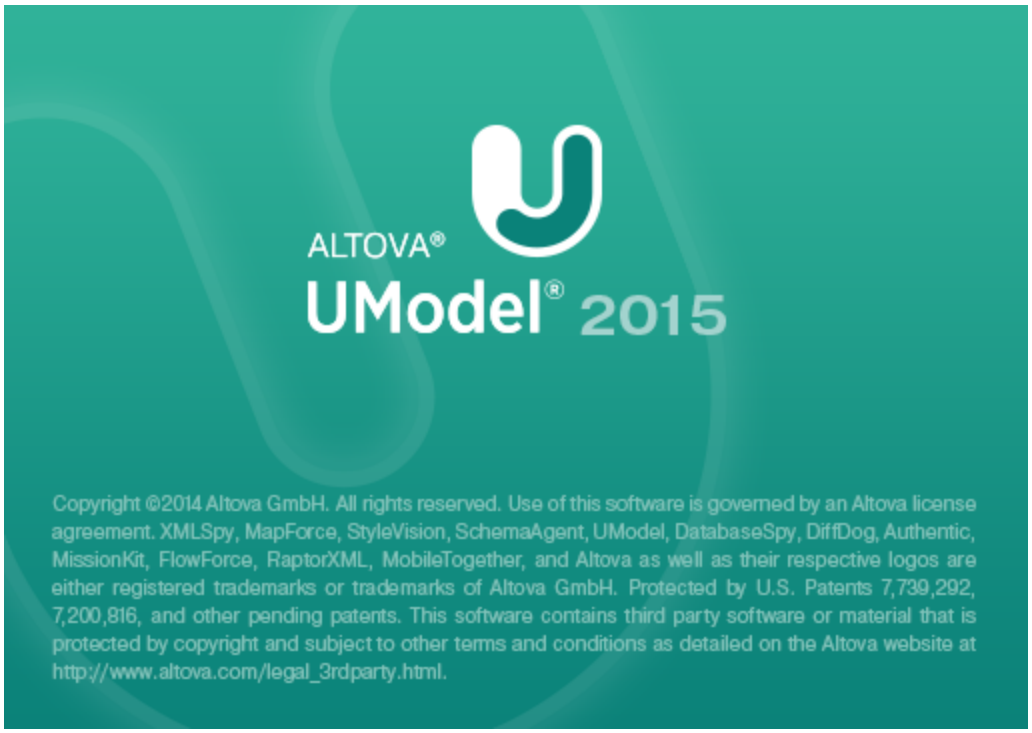
Manual y referencia del usuario



Copyright ©2014 Altova GmbH. All rights reserved. Use of this software is governed by an Altova license agreement. XMLSpy, MapForce, StyleVision, SchemaAgent, UModel, DatabaseSpy, DiffDog, Authentic, MissionKit, FlowForce, RaptorXML, MobileTogether, and Altova as well as their respective logos are either registered trademarks or trademarks of Altova GmbH. Protected by U.S. Patents 7,739,292, 7,200,816, and other pending patents. This software contains third party software or material that is protected by copyright and subject to other terms and conditions as detailed on the Altova website at http://www.altova.com/legal_3rdparty.html.

Manual y referencia del usuario de Altova UModel® 2015

Todos los derechos reservados. Ningún fragmento de esta publicación podrá ser reproducido de manera alguna (ya sea de forma gráfica, electrónica o mecánica, fotocopiado, grabado o reproducido en sistemas de almacenamiento y recuperación de información) sin el consentimiento expreso por escrito de su autor/editor.

Los productos a los que se hace referencia en este documento pueden ser marcas registradas de sus respectivos propietarios. El autor y editor no afirman ser propietarios de dichas marcas registradas.

Durante la elaboración de este documento se tomaron todas las precauciones necesarias para prevenir errores. Sin embargo, el autor y editor no se responsabilizan de los errores u omisiones que pudiese contener el documento ni de los posibles daños o perjuicios derivados del uso del contenido de este documento o de los programas y código fuente que vengan con el documento. Bajo ninguna circunstancia se podrá considerar al autor y editor responsables de la pérdida de beneficios ni de cualquier otro daño y perjuicio derivado directa o indirectamente del uso de este documento.

Fecha de publicación: 2014

© 2014 Altova GmbH

Tabla de contenido

1 Altova UModel® 2015 3

2 Introducción a UModel 6

3 Novedades de UModel 8

4 Tutorial de UModel 14

4.1 Iniciar UModel ... 16
4.2 Casos de uso ... 20
4.3 Diagramas de clases ... 30
4.3.1 Crear clases derivadas ... 37
4.4 Diagramas de objetos ... 43
4.5 Diagramas de componentes ... 49
4.6 Diagramas de implementación ... 55
4.7 Ingeniería de ida y vuelta (modelo - código - modelo) ... 60
4.8 Ingeniería de ida y vuelta (código - modelo - código) ... 67

5 Interfaz del usuario 76

5.1 Estructura del modelo ... 77
5.2 Árbol de diagramas ... 82
5.3 Favoritos ... 84
5.4 Propiedades ... 86
5.5 Estilos ... 88
5.6 Jerarquía ... 91
5.7 Vista general ... 94
5.8 Documentación ... 95
5.9 Capas ... 96
5.9.1 Menú contextual de las capas ... 101
5.10 Mensajes ... 103
5.11 Panel de diagramas ... 104

5.11.1 Propiedades de los diagramas ... 108
5.11.2 Cortar, copiar y pegar en los diagramas de UModel ... 111
5.12 Agregar/insertar elementos en los modelos ... 114
5.13 Crear hipervínculos entre los elementos de modelado ... 116
5.14 Ejemplos Bank ... 122

6 Interfaz de la línea de comandos 126

6.1 Archivo: Nuevo / Cargar / Opciones al guardar ... 130

7 Proyectos e ingeniería de código 134

7.1 Crear un proyecto de UModel desde cero ... 137
7.2 Importar código fuente a los proyectos ... 142
7.3 Importar binarios Java, C# y VB ... 148
7.4 Sincronizar el modelo y el código fuente ... 153
7.4.1 Consejos prácticos ... 156
7.4.2 Refactorización de código y sincronización ... 158
7.5 Requisitos para ingeniería directa ... 159
7.6 Correspondencia entre código Java y elementos de UModel ... 161
7.7 Correspondencia entre código C# y elementos de UModel ... 162
7.8 Correspondencia entre elementos de XML Schema y de UModel ... 163
7.9 Correspondencia entre código VB.NET y de elementos de UModel ... 164
7.10 Correspondencia entre elementos de BD y de UModel ... 165
7.11 Incluir otros proyectos de UModel ... 166
7.12 Combinar proyectos de UModel ... 168
7.12.1 Fusión de proyectos a 2 bandas ... 169
7.12.2 Fusión de proyectos a 3 bandas ... 170
7.12.3 Ejemplo de fusión manual a 3 bandas ... 172
7.13 Compartir paquetes y diagramas ... 175
7.14 Plantillas UML ... 178
7.14.1 Firmas de plantilla ... 180
7.14.2 Enlace de plantilla ... 181
7.14.3 Usar plantillas en operaciones y propiedades ... 182
7.15 Configuración del proyecto ... 183
7.16 Mejorar el rendimiento ... 184

8 Crear relaciones entre los modelos 186

8.1 Ver las relaciones entre los modelos ... 189
8.2 Asociaciones, realizaciones y dependencias ... 190

9 Generar documentación UML 194

9.1 Con una hoja de estilos SPS predeterminada ... 201
9.2 Con hojas de estilos predefinidas por el usuario ... 203

10 Diagramas UML 206

10.1 Diagramas de comportamiento ... 207
10.1.1 Diagrama de actividades ... 208
Insertar elementos ... 209
Crear bifurcaciones y convergencias ... 211
Elementos ... 213
10.1.2 Diagrama de máquina de estados ... 225
Insertar elementos ... 226
Crear estados, actividades y transiciones ... 227
Estados compuestos ... 233
Generar código a partir de diagramas de máquina de estados ... 236
Trabajar con código de máquina de estados ... 239
Elementos ... 243
10.1.3 Diagrama de máquina de estados de protocolos ... 247
Insertar elementos ... 248
Elementos ... 249
10.1.4 Diagrama de casos de uso ... 251
10.1.5 Diagrama de comunicación ... 252
Insertar elementos ... 252
10.1.6 Diagrama global de interacción ... 256
Insertar elementos ... 257
10.1.7 Diagrama de secuencia ... 261
Insertar elementos ... 262
Líneas de vida ... 263
Fragmentos combinados ... 265
Usos de interacción ... 268
Puertas ... 268
Invariantes de estado ... 269
Mensajes ... 269
Generar diagramas de secuencia a partir de código fuente ... 274
Generar varios diagramas de secuencia a partir de propiedades ... 277

Generar código a partir de diagramas de secuencia ... 279
Agregar código a diagramas de secuencia ... 283
10.1.8 Diagrama de ciclo de vida ... 287
Insertar elementos ... 288
Línea de vida ... 288
Marca de graduación ... 291
Evento/estímulo ... 292
Restricción de duración ... 292
Restricción de tiempo ... 293
Mensaje ... 294
10.2 Diagramas de estructura ... 295
10.2.1 Diagrama de clases ... 296
10.2.2 Diagrama de estructura de un compuesto ... 308
Insertar elementos ... 308
10.2.3 Diagrama de componentes ... 310
10.2.4 Diagrama de implementación ... 311
10.2.5 Diagrama de objetos ... 312
10.2.6 Diagrama de paquetes ... 313
Insertar elementos ... 314
10.2.7 Diagrama de perfil y estereotipos ... 316
Agregar estereotipos y definir valores etiquetados ... 317
Estereotipos y enumeraciones ... 321
Estilos de estereotipo definidos por el usuario ... 323
Asignar iconos de estereotipo personalizados ... 325
10.3 Otros diagramas ... 329
10.3.1 Diagramas de esquema XML ... 330
Importar esquemas XML ... 330
Insertar elementos ... 337
Crear y generar un esquema XML ... 340
10.3.2 Diagramas BPMN 1.0 / 2.0 ... 344
Objetos de flujo ... 345
Subprocesos expandidos ... 350
Subprocesos contraídos ... 351
Objetos de conexión ... 353
Contenedores y compartimentos ... 355
Artefactos ... 356
Diagrama de coreografía ... 357
Tareas de coreografía ... 357
Tareas y subprocesos ... 361
Objetos de datos ... 362
Diagrama de colaboración ... 362
Conversación ... 363

Tareas y subprocesos ... 364
Objetos de datos ... 365
Diagrama de procesos de negocio estándar BPMN 2.0 ... 365
Tareas y subprocesos ... 367
Objetos de datos ... 368
10.3.3 Diagramas SysML ... 369
Diagrama de estructura ... 369
Diagrama de definición de bloques ... 370
Diagrama de bloque interno ... 373
Diagrama paramétrico ... 376
Diagrama de paquetes ... 378
Diagrama de requisitos ... 380
Diagrama de comportamiento ... 381
Diagrama de actividades ... 381
Diagrama de secuencia ... 383
Diagrama de máquina de estados ... 384
Diagrama de casos de uso ... 385
10.3.4 Diagramas de BD (importar bases de datos SQL) ... 386
Importar bases de datos SQL en UModel ... 386
Conexiones JDBC ... 392
Ver elementos de modelado de BD ... 395
Agregar columnas a una tabla ... 398
Agregar tablas nuevas a un modelo / a una BD ... 401

11 XMI: intercambio de metadatos XML 410

12 Trabajo en equipo con UModel 414

12.1 Crear y editar subproyectos ... 416

13 Complemento de UModel para MS Visual Studio 424

13.1 Abrir archivos de UModel en MS Visual Studio ... 425
13.2 Diferencias con la versión independiente ... 427
13.3 Crear un proyecto de UModel en Visual Studio ... 432

14 Complemento de UModel para Eclipse 438

14.1 Instalar el complemento de UModel para Eclipse ... 439
14.2 Iniciar Eclipse y usar el complemento de UModel ... 445
14.3 UModel: editor, vista y perspectiva ... 447

14.4 Crear/abrir un archivo/proyecto de UModel ... 448
14.5 Importar/abrir ejemplos desde el navegador ... 450
14.6 Diferencias con la versión independiente ... 453
14.7 Generación de código ... 456

15 Control de código fuente 458

15.1 Instalar sistemas de control de código fuente ... 459
15.1.1 Sistemas de control de código fuente compatibles ... 460
15.1.2 Notas sobre la instalación ... 465
15.1.3 Comparación con Altova DiffDog ... 473
15.2 Comandos de control de código fuente ... 479
15.2.1 Abrir desde el control de código fuente ... 480
15.2.2 Habilitar control de código fuente ... 483
15.2.3 Obtener la versión más reciente ... 484
15.2.4 Obtener ... 485
15.2.5 Obtener carpetas ... 487
15.2.6 Desproteger ... 489
15.2.7 Proteger ... 491
15.2.8 Anular desprotección ... 492
15.2.9 Agregar al control de código fuente ... 494
15.2.10 Quitar del control de código fuente ... 497
15.2.11 Compartir desde el control de código fuente ... 498
15.2.12 Mostrar historial ... 499
15.2.13 Mostrar diferencias ... 502
15.2.14 Mostrar propiedades ... 504
15.2.15 Actualizar estado ... 505
15.2.16 Administrador del control de código fuente ... 506
15.2.17 Cambiar control de código fuente ... 507

16 Iconos en los diagramas de UModel 510

16.1 Diagramas de actividades ... 511
16.2 Diagramas de clases ... 513
16.3 Diagramas de comunicación ... 514
16.4 Diagramas de estructura de un compuesto ... 515
16.5 Diagramas de componentes ... 516
16.6 Diagramas de implementación ... 517
16.7 Diagramas global de interacción ... 518
16.8 Diagramas de objetos ... 519

16.9 Diagramas de paquetes ... 520
16.10 Diagramas de perfil ... 521
16.11 Diagramas de máquina de estados de protocolos ... 522
16.12 Diagramas de secuencia ... 523
16.13 Diagramas de máquina de estados ... 524
16.14 Diagramas de ciclo de vida ... 525
16.15 Diagramas de casos de uso ... 526
16.16 Diagramas de esquema XML ... 527
16.17 Diagramas BPMN ... 528
16.18 Diagramas BPMN 2.0 ... 529
16.19 Modelado de bases de datos ... 530

17 Referencia del usuario 532

17.1 Menú Archivo ... 533
17.2 Menú Edición ... 539
17.3 Menú Proyecto ... 542
17.4 Menú Diseño ... 553
17.5 Menú Vista ... 555
17.6 Menú Herramientas ... 556
17.6.1 Ortografía ... 557
17.6.2 Opciones de ortografía... ... 559
17.6.3 Editor de scripts... ... 564
17.6.4 Macros ... 565
17.6.5 Herramientas definidas por el usuario ... 566
17.6.6 Personalizar ... 567
Comandos ... 567
Barras de herramientas ... 568
Herramientas ... 569
Teclado ... 572
Menú ... 573
Macros ... 574
Complementos ... 574
Opciones ... 574
17.6.7 Restaurar barras de herramientas y ventanas ... 575
17.6.8 Opciones ... 576
17.7 Menú Ventanas ... 583
17.8 Menú Ayuda ... 584

18 Generador de código 590

18.1 Códigos de error ... 591

19 Referencia del programador 594

19.1 Notas sobre la versión ... 596
19.2 Editor de scripts ... 600
19.2.1 Información general ... 602
Proyectos de scripting en UModel ... 603
La interfaz gráfica del editor de scripts ... 605
Componentes de un proyecto de scripts ... 609
19.2.2 Crear un proyecto de scripting ... 612
19.2.3 Declaraciones globales ... 613
19.2.4 Formularios ... 614
Crear un formulario nuevo ... 614
Diseño de formularios y objetos ... 615
Eventos de formularios ... 618
19.2.5 Eventos ... 620
19.2.6 Macros ... 621
Crear y editar una macro ... 621
Ejecutar una macro ... 622
Depurar una macro ... 623
19.2.7 Notas sobre programación ... 624
Comandos integrados ... 625
Uso de los formularios y comandos ... 632
19.2.8 Ejemplo de uso (script de muestra de UModel) ... 634
19.3 Crear complementos en entornos IDE ... 638
19.3.1 Cómo crear un complemento en MS Visual Studio .NET ... 639
Agregar una referencia a la biblioteca del complemento ... 639
Agregar una referencia a la biblioteca de tipos de UModel ... 640
Cambiar la configuración del compilador (crear ensamblado visible) ... 641
Cambiar la configuración del compilador (exponer contenedor COM) ... 642
Firmar el componente con un nombre seguro (opcional) ... 642
Implementar IUModelPlugIn ... 643
Probar el complemento de UModel ... 645
19.3.2 Implementación de complementos ... 646

19.3.3 XML de configuración ... 648
19.3.4 Controles ActiveX ... 652
19.3.5 IUModelPlugIn ... 654
OnInitialize ... 656
OnRunning ... 656
OnShutdown ... 656
GetUIModifications ... 657
GetDescription ... 658
OnCommand ... 658
OnUpdateCommand ... 659
19.4 The UModel API ... 660
19.4.1 Overview ... 661
Object model ... 661
Object model UMLData ... 662
Object model UMLData styles ... 664
Graphical Objects ... 665
How to... ... 666
How to create Sequence Diagrams ... 666
How to generate Sequence Diagrams from code ... 666
How to create Sequence diagrams manually ... 667
Undo / Redo and UMLData transaction handling ... 671
How to use predefined UModel elements ... 672
How to work with Stereotypes and Tagged Values ... 672
How to use UMLData events and eventfilters ... 673
How to create and use Hyperlinks ... 675
Error handling ... 676
Programming languages ... 678
C# ... 679
Java ... 684
Example Java Project ... 685
JScript ... 689
Start application ... 690
Document Access ... 691
Generate documentation ... 692
Generate code ... 693
Update Documentation ... 698
19.4.2 Usage Examples ... 707
Styles (C#) ... 707
SetupStyles (Deployment) ... 713
CSharpDelegate (C#) ... 713
DefaultPrefix (C#) ... 718
CreateCode (JScript) ... 726
UpdateDocumentation (JScript) ... 731
StatisticsActiveX (C#) ... 740
19.5 UModel API Reference ... 747
19.5.1 UModel Plugins ... 748

UModelAPI - IUModelPlugIn ... 748
UModelAPI - UModelUpdateAction ... 749
19.5.2 UModel API ... 750
UModelAPI - IApplication ... 751
UModelAPI - IBinaryTypeEntries ... 754
UModelAPI - IBinaryTypeEntry ... 755
UModelAPI - ICollectionTemplate ... 755
UModelAPI - ICollectionTemplates ... 756
UModelAPI - IDiagramWindow ... 758
UModelAPI - IDiagramWindows ... 761
UModelAPI - IDialog ... 761
UModelAPI - IDialogs ... 762
UModelAPI - IDocument ... 766
UModelAPI - IExportXMIFileDlg ... 774
UModelAPI - IFocusedUMLDataNotifier ... 775
UModelAPI - IGenerateDocumentationDlg ... 776
UModelAPI - IGenerateSequenceDiagramDlg ... 783
UModelAPI - IGenerateStateMachineCodeDlg ... 785
UModelAPI - IImportBinaryTypesDlg ... 787
UModelAPI - IImportDatabaseDlg ... 791
UModelAPI - IImportSourceDirectoryDlg ... 793
UModelAPI - IImportSourceDlg ... 794
UModelAPI - IImportSourceProjectDlg ... 796
UModelAPI - IImportXMLSchemaDirectoryDlg ... 798
UModelAPI - IImportXMLSchemaFileDlg ... 799
UModelAPI - IIncludeSubprojectDlg ... 800
UModelAPI - IKindSelection ... 801
UModelAPI - IKindSelectionList ... 801
UModelAPI - ILocalOptions ... 802
UModelAPI - ILocalOptionsCodeEngineering ... 803
UModelAPI - ILocalOptionsDiagramEditing ... 806
UModelAPI - ILocalOptionsEditing ... 808
UModelAPI - ILocalOptionsFile ... 810
UModelAPI - ILocalOptionsView ... 811
UModelAPI - IMatchRenamedDlg ... 813
UModelAPI - IMatchRenamedEntries ... 814
UModelAPI - IMatchRenamedEntry ... 815
UModelAPI - IModelTransformationDlg ... 816
UModelAPI - IModelTransformationTypeMapping ... 818
UModelAPI - IModelTransformationTypeMappings ... 818
UModelAPI - IProjectSettingsDlg ... 820

UModelAPI - ISaveAllDiagramsAsImagesDlg ... 821
UModelAPI - ISynchronizationSettingsDlg ... 822
UModelAPI - ITransactionNotifier ... 823
UModelAPI - IURLDlg ... 823
Events ... 824
UModelAPI - _IApplicationEvents ... 825
UModelAPI - _IDiagramWindowEvents ... 825
UModelAPI - _IDocumentEvents ... 826
UModelAPI - _IFocusedUMLDataEvents ... 827
UModelAPI - _ISynchronizationEvents ... 828
UModelAPI - _ITransactionEvents ... 828
Enumerations ... 828
UModelAPI - ENUMApplicationFrameTitle ... 829
UModelAPI - ENUMApplicationStatus ... 829
UModelAPI - ENUMAutolayoutGrowDirectionKind ... 829
UModelAPI - ENUMCodeLang ... 830
UModelAPI - ENUMCodeLangVersion ... 831
UModelAPI - ENUMDiagramLayoutKind ... 831
UModelAPI - ENUMDocumentationFilePathKind ... 832
UModelAPI - ENUMDocumentationFontSetting ... 832
UModelAPI - ENUMDocumentationImageFormat ... 833
UModelAPI - ENUMDocumentationOutputFormat ... 833
UModelAPI - ENUMExportXMIType ... 833
UModelAPI - ENUMOpenMessageWindow ... 834
UModelAPI - ENUMOutputImageFormat ... 834
UModelAPI - ENUMSynchronizationDeleteKind ... 834
UModelAPI - ENUMSynchronizationKind ... 835
UModelAPI - ENUMSyntaxCheckKind ... 835
UMLData ... 835
UModelAPI - IUMLData ... 836
UModelAPI - IUMLDataList ... 838
IUMLElement ... 843
UModelAPI - IUMLAbstraction ... 843
UModelAPI - IUMLAcceptEventAction ... 844
UModelAPI - IUMLAction ... 846
UModelAPI - IUMLActionExecutionSpecification ... 847
UModelAPI - IUMLActionInputPin ... 848
UModelAPI - IUMLActivity ... 849
UModelAPI - IUMLActivityEdge ... 851
UModelAPI - IUMLActivityFinalNode ... 852
UModelAPI - IUMLActivityGroup ... 853
UModelAPI - IUMLActivityNode ... 854
UModelAPI - IUMLActivityParameterNode ... 856
UModelAPI - IUMLActivityPartition ... 857
UModelAPI - IUMLActor ... 859
UModelAPI - IUMLAnyReceiveEvent ... 859
UModelAPI - IUMLArtifact ... 860
UModelAPI - IUMLAssociation ... 862
UModelAPI - IUMLAssociationClass ... 863
UModelAPI - IUMLBehavior ... 864
UModelAPI - IUMLBehavioralFeature ... 866

UModelAPI - IUMLBehavioredClassifier ... 867
UModelAPI - IUMLBehaviorExecutionSpecification ... 869
UModelAPI - IUMLCallAction ... 869
UModelAPI - IUMLCallBehaviorAction ... 871
UModelAPI - IUMLCallEvent ... 871
UModelAPI - IUMLCallOperationAction ... 872
UModelAPI - IUMLCentralBufferNode ... 873
UModelAPI - IUMLChangeEvent ... 874
UModelAPI - IUMLClass ... 875
UModelAPI - IUMLClassifier ... 878
UModelAPI - IUMLClassifierTemplateParameter ... 881
UModelAPI - IUMLCollaboration ... 882
UModelAPI - IUMLCollaborationUse ... 883
UModelAPI - IUMLCombinedFragment ... 884
UModelAPI - IUMLComment ... 885
UModelAPI - IUMLCommentTextHyperlink ... 886
UModelAPI - IUMLComponent ... 887
UModelAPI - IUMLComponentRealization ... 889
UModelAPI - IUMLConnectableElement ... 890
UModelAPI - IUMLConnectionPointReference ... 891
UModelAPI - IUMLConnector ... 892
UModelAPI - IUMLConnectorEnd ... 893
UModelAPI - IUMLConstraint ... 894
UModelAPI - IUMLControlFlow ... 896
UModelAPI - IUMLControlNode ... 896
UModelAPI - IUMLDataStoreNode ... 897
UModelAPI - IUMLDataType ... 898
UModelAPI - IUMLDecisionNode ... 899
UModelAPI - IUMLDependency ... 900
UModelAPI - IUMLDeployedArtifact ... 900
UModelAPI - IUMLDeployment ... 901
UModelAPI - IUMLDeploymentTarget ... 902
UModelAPI - IUMLDevice ... 903
UModelAPI - IUMLDirectedRelationship ... 903
UModelAPI - IUMLDuration ... 904
UModelAPI - IUMLDurationConstraint ... 905
UModelAPI - IUMLDurationInterval ... 906
UModelAPI - IUMLDurationObservation ... 906
UModelAPI - IUMLElement ... 907
UModelAPI - IUMLElementImport ... 911
UModelAPI - IUMLEncapsulatedClassifier ... 912
UModelAPI - IUMLEnumeration ... 912
UModelAPI - IUMLEnumerationLiteral ... 914
UModelAPI - IUMLEvent ... 915
UModelAPI - IUMLExceptionHandler ... 916
UModelAPI - IUMLExecutableNode ... 917
UModelAPI - IUMLExecutionEnvironment ... 918
UModelAPI - IUMLExecutionSpecification ... 918
UModelAPI - IUMLExpansionNode ... 919
UModelAPI - IUMLExpansionRegion ... 920
UModelAPI - IUMLExpression ... 922
UModelAPI - IUMLExtend ... 923
UModelAPI - IUMLExtensionPoint ... 924
UModelAPI - IUMLFeature ... 925

UModelAPI.-.IUMLFinalNode ... 925
UModelAPI.-.IUMLFinalState ... 926
UModelAPI.-.IUMLFlowFinalNode ... 927
UModelAPI.-.IUMLForkNode ... 927
UModelAPI.-.IUMLFunctionBehavior ... 928
UModelAPI.-.IUMLGate ... 929
UModelAPI.-.IUMLGeneralization ... 929
UModelAPI.-.IUMLHyperlink ... 930
UModelAPI.-.IUMLHyperlink2File ... 931
UModelAPI.-.IUMLHyperlink2GuiElement ... 932
UModelAPI.-.IUMLHyperlink2Model ... 933
UModelAPI.-.IUMLInclude ... 934
UModelAPI.-.IUMLInformationFlow ... 935
UModelAPI.-.IUMLInitialNode ... 937
UModelAPI.-.IUMLInputPin ... 938
UModelAPI.-.IUMLInstanceSpecification ... 939
UModelAPI.-.IUMLInstanceValue ... 941
UModelAPI.-.IUMLInteraction ... 942
UModelAPI.-.IUMLInteractionConstraint ... 943
UModelAPI.-.IUMLInteractionFragment ... 945
UModelAPI.-.IUMLInteractionOperand ... 945
UModelAPI.-.IUMLInteractionUse ... 946
UModelAPI.-.IUMLInterface ... 947
UModelAPI.-.IUMLInterfaceRealization ... 950
UModelAPI.-.IUMLInterruptibleActivityRegion ... 951
UModelAPI.-.IUMLInterval ... 952
UModelAPI.-.IUMLIntervalConstraint ... 953
UModelAPI.-.IUMLInvocationAction ... 954
UModelAPI.-.IUMLJoinNode ... 955
UModelAPI.-.IUMLLifeline ... 956
UModelAPI.-.IUMLLiteralBoolean ... 957
UModelAPI.-.IUMLLiteralInteger ... 957
UModelAPI.-.IUMLLiteralNull ... 958
UModelAPI.-.IUMLLiteralSpecification ... 959
UModelAPI.-.IUMLLiteralString ... 959
UModelAPI.-.IUMLLiteralUnlimitedNatural ... 960
UModelAPI.-.IUMLManifestation ... 960
UModelAPI.-.IUMLMergeNode ... 961
UModelAPI.-.IUMLMessage ... 962
UModelAPI.-.IUMLMessageEnd ... 964
UModelAPI.-.IUMLMessageEvent ... 964
UModelAPI.-.IUMLMessageOccurrenceSpecification ... 965
UModelAPI.-.IUMLModel ... 966
UModelAPI.-.IUMLMultiplicityElement ... 966
UModelAPI.-.IUMLNamedElement ... 968
UModelAPI.-.IUMLNamespace ... 970
UModelAPI.-.IUMLNode ... 972
UModelAPI.-.IUMLObjectFlow ... 973
UModelAPI.-.IUMLObjectNode ... 974
UModelAPI.-.IUMLObservation ... 976
UModelAPI.-.IUMLOccurrenceSpecification ... 977
UModelAPI.-.IUMLOpaqueAction ... 978
UModelAPI.-.IUMLOpaqueBehavior ... 979
UModelAPI.-.IUMLOpaqueExpression ... 979

UModelAPI - IUMLOperation ... 980
UModelAPI - IUMLOutputPin ... 982
UModelAPI - IUMLPackage ... 983
UModelAPI - IUMLPackageableElement ... 985
UModelAPI - IUMLPackageImport ... 986
UModelAPI - IUMLPackageMerge ... 987
UModelAPI - IUMLParameter ... 988
UModelAPI - IUMLParameterableElement ... 989
UModelAPI - IUMLPin ... 990
UModelAPI - IUMLPort ... 991
UModelAPI - IUMLPrimitiveType ... 992
UModelAPI - IUMLProfile ... 993
UModelAPI - IUMLProfileApplication ... 994
UModelAPI - IUMLProperty ... 995
UModelAPI - IUMLProtocolStateMachine ... 998
UModelAPI - IUMLProtocolTransition ... 999
UModelAPI - IUMLPseudostate ... 1000
UModelAPI - IUMLRealization ... 1001
UModelAPI - IUMLRedefinableElement ... 1001
UModelAPI - IUMLRedefinableTemplateSignature ... 1002
UModelAPI - IUMLRegion ... 1003
UModelAPI - IUMLRelationship ... 1004
UModelAPI - IUMLSendSignalAction ... 1005
UModelAPI - IUMLSignal ... 1006
UModelAPI - IUMLSignalEvent ... 1007
UModelAPI - IUMLSlot ... 1007
UModelAPI - IUMLState ... 1009
UModelAPI - IUMLStateInvariant ... 1012
UModelAPI - IUMLStateMachine ... 1013
UModelAPI - IUMLStereotype ... 1014
UModelAPI - IUMLStereotypeApplication ... 1015
UModelAPI - IUMLStructuralFeature ... 1016
UModelAPI - IUMLStructuredActivityNode ... 1017
UModelAPI - IUMLStructuredClassifier ... 1019
UModelAPI - IUMLTemplateableElement ... 1020
UModelAPI - IUMLTemplateBinding ... 1021
UModelAPI - IUMLTemplateParameter ... 1022
UModelAPI - IUMLTemplateParameterSubstitution ... 1024
UModelAPI - IUMLTemplateSignature ... 1025
UModelAPI - IUMLTimeConstraint ... 1026
UModelAPI - IUMLTimeEvent ... 1026
UModelAPI - IUMLTimeExpression ... 1027
UModelAPI - IUMLTimeInterval ... 1029
UModelAPI - IUMLTimeObservation ... 1029
UModelAPI - IUMLTransition ... 1030
UModelAPI - IUMLTrigger ... 1032
UModelAPI - IUMLType ... 1033
UModelAPI - IUMLTypedElement ... 1034
UModelAPI - IUMLUsage ... 1034
UModelAPI - IUMLUseCase ... 1035
UModelAPI - IUMLValuePin ... 1037
UModelAPI - IUMLValueSpecification ... 1039
UModelAPI - IUMLValueSpecificationAction ... 1041
UModelAPI - IUMLVertex ... 1042

IUMLGuiElement 1043
.UModelAPI.-.IUMLGuiActivityDiagram 1044
.UModelAPI.-.IUMLGuiAttachedNode 1044
.UModelAPI.-.IUMLGuiBehaviorDiagram 1045
.UModelAPI.-.IUMLGuiBPMN2ChoreographyDiagram 1046
.UModelAPI.-.IUMLGuiBPMN2CollaborationDiagram 1046
.UModelAPI.-.IUMLGuiBPMN2Diagram 1047
.UModelAPI.-.IUMLGuiBPMNDiagram 1047
.UModelAPI.-.IUMLGuiClassDiagram 1048
.UModelAPI.-.IUMLGuiCommunicationDiagram 1048
.UModelAPI.-.IUMLGuiCommunicationLink 1049
.UModelAPI.-.IUMLGuiComponentDiagram 1050
.UModelAPI.-.IUMLGuiCompositeStructureDiagram 1050
.UModelAPI.-.IUMLGuiConstrainedNode 1051
.UModelAPI.-.IUMLGuiContainmentLink 1052
.UModelAPI.-.IUMLGuiDeploymentDiagram 1053
.UModelAPI.-.IUMLGuiDiagram 1053
.UModelAPI.-.IUMLGuiDiagramLayer 1057
.UModelAPI.-.IUMLGuiElement 1058
.UModelAPI.-.IUMLGuiEndWaypoint 1059
.UModelAPI.-.IUMLGuiExtensionDiagram 1059
.UModelAPI.-.IUMLGuiInteractionDiagram 1060
.UModelAPI.-.IUMLGuiInteractionOverviewDiagram 1060
.UModelAPI.-.IUMLGuiLabeledRelativeNodeLink 1061
.UModelAPI.-.IUMLGuiLineConnectionWaypoint 1062
.UModelAPI.-.IUMLGuiLineLink 1063
.UModelAPI.-.IUMLGuiLink 1065
.UModelAPI.-.IUMLGuiMiddleWaypoint 1066
.UModelAPI.-.IUMLGuiNodeLink 1067
.UModelAPI.-.IUMLGuiNote 1069
.UModelAPI.-.IUMLGuiNoteLink 1070
.UModelAPI.-.IUMLGuiObjectDiagram 1071
.UModelAPI.-.IUMLGuiPackageDiagram 1071
.UModelAPI.-.IUMLGuiProfileDiagram 1072
.UModelAPI.-.IUMLGuiProtocolStateMachineDiagram 1072
.UModelAPI.-.IUMLGuiRelativeNodeLink 1073
.UModelAPI.-.IUMLGuiRootElement 1074
.UModelAPI.-.IUMLGuiSeparatedNodeLink 1075
.UModelAPI.-.IUMLGuiSeparatedNodeLink2D 1076
.UModelAPI.-.IUMLGuiSequenceDiagram 1077
.UModelAPI.-.IUMLGuiStateMachineDiagram 1078
.UModelAPI.-.IUMLGuiStructureDiagram 1079
.UModelAPI.-.IUMLGuiStyle 1080
.UModelAPI.-.IUMLGuiStyles 1081
.UModelAPI.-.IUMLGuiSubDiagramNode 1082
.UModelAPI.-.IUMLGuiSysMLActivityDiagram 1083
.UModelAPI.-.IUMLGuiSysMLBlockDefinitionDiagram 1084
.UModelAPI.-.IUMLGuiSysMLInternalBlockDiagram 1085
.UModelAPI.-.IUMLGuiSysMLPackageDiagram 1085
.UModelAPI.-.IUMLGuiSysMLParametricDiagram 1086
.UModelAPI.-.IUMLGuiSysMLRequirementDiagram 1087
.UModelAPI.-.IUMLGuiSysMLSequenceDiagram 1087
.UModelAPI.-.IUMLGuiSysMLStateMachineDiagram 1088
.UModelAPI.-.IUMLGuiSysMLUseCaseDiagram 1088

.UModelAPI - IUMLGuiTextHyperlink ... 1089
.UModelAPI - IUMLGuiTextLabel ... 1091
.UModelAPI - IUMLGuiTextLabelWaypoint ... 1092
.UModelAPI - IUMLGuiTickMark ... 1093
.UModelAPI - IUMLGuiTimingDiagram ... 1094
.UModelAPI - IUMLGuiTimingDiagramLifeline ... 1096
.UModelAPI - IUMLGuiTimingDiagramMessage ... 1098
.UModelAPI - IUMLGuiUseCaseDiagram ... 1099
.UModelAPI - IUMLGuiVisibleElement ... 1099
.UModelAPI - IUMLGuiWaypoint ... 1100
.UModelAPI - IUMLGuiXMLSchemaDiagram ... 1102
Events ... 1102
.UModelAPI - _IUMLDataEvents ... 1102
Enumerations ... 1103
.UModelAPI - ENUMUMLAggregationKind ... 1103
.UModelAPI - ENUMUMLCallConcurrencyKind ... 1104
.UModelAPI - ENUMUMLConnectorKind ... 1104
.UModelAPI - ENUMUMLDataEventFilter ... 1105
.UModelAPI - ENUMUMLDBDataSourceMethod ... 1105
.UModelAPI - ENUMUMLExpansionKind ... 1106
.UModelAPI - ENUMUMLGuiStyleKind ... 1106
.UModelAPI - ENUMUMLGuiTextLabelKind ... 1107
.UModelAPI - ENUMUMLInteractionOperatorKind ... 1108
.UModelAPI - ENUMUMLMessageKind ... 1108
.UModelAPI - ENUMUMLMessageSort ... 1109
.UModelAPI - ENUMUMLObjectNodeOrderingKind ... 1109
.UModelAPI - ENUMUMLParameterDirectionKind ... 1109
.UModelAPI - ENUMUMLPredefinedElement ... 1110
.UModelAPI - ENUMUMLPseudostateKind ... 1111
.UModelAPI - ENUMUMLTransitionKind ... 1111
.UModelAPI - ENUMUMLVisibilityKind ... 1112
UModelAPI - IUMLDataAll ... 1112
UModelAPI - UMLData ... 1179

20 Anexos 1182

20.1 Información sobre licencias ... 1183
20.1.1 Distribución electrónica de software ... 1184
20.1.2 Activación del software y medición de licencias ... 1185
20.1.3 Derechos de propiedad intelectual ... 1186
20.1.4 Contrato de licencia para el usuario final de Altova ... 1187

Altova UModel® 2015

Altova UModel® 2015

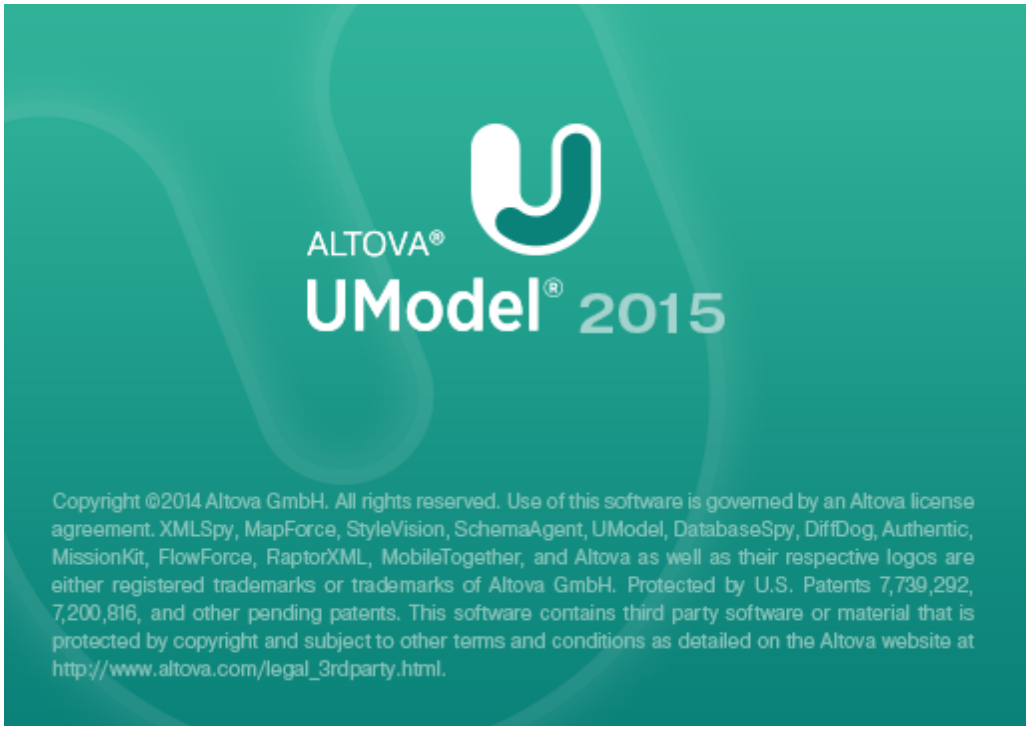
1 Altova UModel® 2015

UModel® 2015 Professional Edition es una asequible herramienta de modelado UML con una potente interfaz gráfica y avanzadas funciones que le facilitarán el trabajo con UML. Además incluye funciones para trabajar con los aspectos más prácticos de la especificación 2.4. UModel es una aplicación Windows de 32/64 bits compatible con Windows Server 2003/2008/2012, Windows XP, Windows Vista, Windows 7 y Windows 8. La versión de 64 bits solamente está disponible en las ediciones Enterprise y Professional Edition.

Principales características de UModel® 2015:

- Compatible con los 14 tipos de diagrama de modelado UML 2.4
- Opción para importar y exportar bases de datos SQL
- Funciones para trabajar en equipo y editar proyectos de forma simultánea
- Fusión de proyectos a 3 bandas
- Máquinas de estado de protocolos
- Compatible con diagramas SysML 1.2
- Generación de diagramas de secuencia a partir del código fuente
- Generación de código a partir de diagramas de máquina de estados
- API y complemento de UModel
- Entorno de scripting y editor de formularios integrados
- Integración en Eclipse
- Compatible con sistemas de control de versiones
- Compatible con diagramas de XML Schema
- Compatible con BPMN (Business Process Modeling Notation) 1.0 y 2.0
- Opción para usar varias capas en el diagrama UML
- Opción para importar binarios Java, C# y Visual Basic
- Opción para crear hipervínculos entre diagramas y elementos de modelado
- Color de sintaxis en los diagramas
- Estilos en cascada
- Número ilimitado de operaciones Deshacer/Rehacer
- Sofisticada generación de código Java, C# y Visual Basic a partir de los modelos
- Ingeniería inversa de código fuente Java, C# y Visual Basic ya existente
- Ingeniería de ida y vuelta para combinar el código con el modelo
- Importación y exportación de modelos con XMI 2.4 para UML 2.0, 2.1, & 2.1.2, 2.2, 2.3, 2.4
- Generación de documentación de proyectos de UModel

Todas estas características le ayudarán a incrementar la productividad y optimizar resultados con UML.



UML®, OMG™, Object Management Group™ y Unified Modeling Language™ son marcas o marcas registradas de Object Management Group, Inc. en EE UU y otros países.

Última actualización: 04/09/2014

Altova UModel® 2015

Introducción a UModel

2 Introducción a UModel

Sitio web de Altova: Herramienta de modelado UML Altova UModel

UML es un lenguaje de modelado completo pero no abarca una metodología para procesos de desarrollo, generación de código ni ingeniería de ida y vuelta. UModel está diseñado para ofrecer una total flexibilidad durante el proceso de modelado:

- Los diagramas de UModel se pueden crear en el orden que se quiera y en cualquier momento. No es necesario seguir un orden determinado durante el modelado.
- La actualización/combinación del código o del modelo se puede hacer por proyectos, por paquetes o por clases. Para poder realizar ingeniería de ida y vuelta en UModel no hace falta tener pseudocódigo ni que el código generado incluya comentarios.
- En UModel, la generación de código se realiza sobre plantillas SPL y, por tanto, se puede personalizar. Las personalizaciones son detectadas automáticamente durante la generación de código.
- La función de generación de código y de ingeniería inversa es compatible con Java 1.4.x, 5.0 y 1.6, con C# 1.2, 2.0, 3.0 y 4.0 y con Visual Basic 7.1, 8.0 y 9.0. Cada proyecto puede ser compatible con Java, C# y VB simultáneamente.
- UModel es compatible con plantillas y genéricos UML.
- Intercambio de metadatos XML (XMI 2.4) para UML 2.0 / 2.1.1 / 2.1.2 / 2.2 / 2.3 / 2.4
- Cuando se añaden propiedades u operaciones, UModel ofrece ayudantes de entrada contextuales donde el usuario puede elegir tipos, nivel de protección y otras propiedades normalmente disponibles en entornos IDE como XMLSpy, Visual Studio o Eclipse.
- Color de sintaxis en diagramas para poder trabajar de forma más intuitiva.
- Los elementos de modelado y sus propiedades (fuente, color, borde, etc.) se pueden personalizar de forma jerárquica por proyectos, por nodos/líneas, por familias de elementos o por elementos.
- Puede definir actores personalizables en diagramas de casos de uso para ilustrar terminales y otros símbolos.
- Puede buscar elementos de modelado por su nombre en la pestaña *Diagramas*, en el panel Estructura del modelo y en las ventanas Mensajes y Documentación.
- Puede buscar/resaltar clases, asociaciones de objetos, dependencias, generalizaciones, etc. con ayuda del menú contextual de los diagramas.
- Las operaciones de Deshacer/Rehacer no solo abarcan cambios en el contenido sino también cambios de estilo realizados en los elementos del modelo.

Nota: el objetivo de la presente documentación no es describir ni explicar el lenguaje UML. Su objetivo es explicar al usuario cómo utilizar la herramienta de modelado Altova UModel para modelar código y completar procesos de ingeniería de ida y vuelta correctamente.

 © 2014 Altova GmbH

Altova UModel® 2015

Novedades de UModel

3 Novedades de UModel

Novedades de la versión **2015**:

- Actualizaciones internas y mejoras

Novedades de la versión **2014 Release 2**:

- Compatibilidad con Visual Studio 2012 y 2013
- Actualizaciones internas y mejoras

Novedades de la **versión 2014**:

- Actualizaciones internas y mejoras
- Opción para definir plantillas SPL distintas para cada proyecto de UModel
- Compatibilidad con Eclipse 4.3

Novedades de la **versión 2013**:

- Compatibilidad con UML 2.4
- Compatibilidad con SysML 1.2
- Corrector ortográfico con nuevas funciones
- Opción para ver las propiedades .NET como asociaciones
- Opción para omitir métodos getters/setters en la generación de diagramas de secuencia para un proyecto
- Compatibilidad con los pins de excepción UML
- Compatibilidad con Eclipse 4.2

A partir de la **versión 2012** Altova ofrece la edición **UModel Basic Edition**. Esta edición ofrece todas las funciones de la antigua edición Professional Edition.

A partir de la **versión 2012** la edición **Professional Edition** pasa a tener todas las funciones de la antigua edición Enterprise Edition (es decir, compatibilidad con bases de datos SQL, diagramas SysML y BPMN, capas en diagramas, automatización con API de COM e integración en Visual Studio y Eclipse).

A partir de la **versión 2012** la edición **Enterprise Edition** tiene todas las características recién mencionadas y además es compatible con procesos de arquitectura dirigida por modelos, lo cual permite pasar elementos UML a otros lenguajes de programación.

Novedades de la **versión 2012 Release R2**:

- Opción para generar código a partir de un diagrama de secuencia
- Compatibilidad con archivos lógicos de la BD IBM iSeries (también muestra los archivos lógicos como vistas)
- Compatibilidad con archivos lógicos de IBM DB2. En las ediciones IBM iSeries de la BD DB2 un archivo lógico representa uno o varios archivos físicos. Los archivos lógicos permiten al usuario acceder a datos de una secuencia o de un formato que puede ser distinto al del archivo físico. Los usuarios que se conecten a equipos de IBM iSeries pueden encontrar BD construidas con archivos lógicos. En las versiones anteriores de UModel no se podía acceder a este tipo de archivos.

Novedades de la **versión 2012**:

 © 2014 Altova GmbH

- Compatibilidad con Eclipse 3.7
- Acceso a la API de UModel desde Java
- Ahora el motor de BD es compatible con conexiones ODBC y JDBC directas
- Posibilidad de arrastrar las líneas de ajuste de la cuadrícula y colocarlas en otra posición

Novedades de la **versión 2011 Release 3**:

- Opción para ver las propiedades de VS .NET (C# o VB.NET) en un compartimiento propio
- Compatibilidad con VS .NET 4.0 en el editor de scripts
- Opción para usar ensamblados de VS .NET 4.0 en el editor de scripts
- Compatibilidad con plantillas .NET, parámetros out y enumeradores java script en el editor de scripts
- Opción para crear directorios nuevos siguiendo la jerarquía del espacio de nombres Java
- Pequeñas mejoras en SPL para ordenar colecciones por nombre y tipo, así como nuevas funciones de comparación de cadenas

Novedades de la **versión 2011 Release 2**:

- Compatibilidad con BPMN 2.0, incluso con diagramas de coreografía, colaboración y con el diagrama de procesos de negocio estándar BPMN 2.0
- Opción para pasar diagramas BPMN 1.0 a BPMN 2.0 en cualquier momento. Ambos tipos de diagrama pueden existir al mismo tiempo en el mismo proyecto
- Generación de código Java, VB.NET y C# a partir de diagramas de máquina de estados
- Opción para personalizar la documentación UML con ayuda de hojas de estilos de StyleVision (archivos SPS)
- Opción para agregar operaciones automáticamente a las actividades de la máquina de estados
- Opción para activar/desactivar los nombres de región en los estados

Novedades de la **versión 2011**:

- Opción para importar y exportar bases de datos SQL
- Funciones para trabajar en equipo: opción para dividir proyectos en varias secciones y poder editarlo en equipo de forma simultánea
- Opción para crear una fusión de proyecto a 3 bandas
- Diagramas de máquina de estados: creación automática de operaciones cuando se crean transiciones
- Diagramas de secuencia: creación automática de operaciones (incluidos sus parámetros), por llamadas con tipo en mensajes

Novedades de la **versión 2010 Release 3**:

- Compatibilidad con Visual Studio 2010: compatible con nuevos archivos de proyecto y solución e incluye archivos de inclusión predefinidos para .NET Framework 4.0
- Compatibilidad con C# 4.0: las nuevas funciones de C# 4.0 son compatibles con las funciones de ingeniería de ida y vuelta, de importación de binarios y de generación de diagramas de secuencia
- Función para crear máquinas de estados de protocolos
- Nueva opción de diseño automático de resultados: diseño por bloques
- Opción para omitir los nombres de operación en la generación de diagramas de secuencia
- Opción para definir variables de argumentos para herramientas externas

Novedades de la **versión 2010 Release 2**:

- Compatible con sistemas de 64 bits
- Compatibilidad con UML 2.3
- Opción para dividir automáticamente los diagramas de secuencia generados en diagramas subordinados más pequeños
- La generación de documentación HTML incluye hipervínculos entre los elementos que se pueden definir manualmente
- Creación de diagramas de secuencia distintos para la parte getter o setter de una propiedad (C# o VB.NET)
- Mejoras en el editor de scripts (ir a la llave, extender e ir a la llave, etc.)

Nota: como resultado de la nueva versión 2.3 de UML la función de ingeniería de código incluye algunas novedades (consulte los apartados de generación de código Java / C# / VB.NET para obtener más información):

Los modificadores de clase “sealed” (C#), “NotInheritable” (VB.NET) y “final” (Java) ahora tienen asignado “isFinalSpecification” en UML.
El modificador Java “final” para campos y métodos ahora está asignado a “leaf” en UML.

Novedades de la **versión 2010**:

- Compatibilidad con SysML 1.1
- Nuevo entorno de scripting / editor de formularios
- Compatibilidad con Eclipse 3.5

Ingeniería de código:

- Opción para resolver alias durante la ingeniería inversa
- Opción para cancelar la generación de código a partir del código fuente desde VS. NET y Eclipse
- Pequeñas mejoras en la generación de diagramas de secuencia

API de UModel

- Mejoras generales
- Los controles ActiveX personalizados se pueden ver en cuadros de diálogo de UModel

Interfaz del usuario

- Los comentarios introducidos en la ventana Documentación se pueden ver/editar en los diagramas
- Los valores etiquetados ahora se pueden ver en un nuevo compartimiento en las clases, interfaces, etc.
- Ahora puede definir hipervínculos en la ventana Documentación
- Ahora puede crear hipervínculos entre los modelos en el panel Estructura del modelo
- Ahora pueden verse las líneas de contención UML

Generación de documentación:

- Ahora puede guardar el código CSS en archivos HTML o de forma externa
- Los archivos de imagen generados se pueden guardar en el subdirectorio indicado

Sistema de archivos:

- Opción **Guardar copia como**

Novedades de la **versión 2009**:

 © 2014 Altova GmbH

- Compatibilidad con UML 2.2
- Generación de diagramas de secuencia a partir de código fuente
- API de UModel y complemento para entornos IDE que permiten automatizar tareas comunes y leer y modificar estructuras de datos UML directamente
- Compatibilidad con sistemas de control de versiones en la versión independiente y en las versiones para Eclipse y Visual Studio: MS SourceSafe, CVS, SubVersion, Rational ClearCase etc.
- Nuevo diagrama de perfil UML
- Detección y control de tipos y espacios de nombres para la refactorización de código
- Función Ir a para líneas de vida en los diagramas de secuencia
- Ventaja Jerarquía con presentación y visualización mejoradas
- Opción para guardar archivos de proyecto de UModel en formato pretty print
- Mejoras en la función de documentación parcial
- Opción para abrir proyectos de UModel desde una dirección URL
- Opción para cargar/guardar con el archivo de proyecto los diagramas que están abiertos

Novedades de la **versión 2008 Release 2**:

- Compatibilidad con Business Process Modeling Notation.
- Compatibilidad con Visual Basic .NET 9.0 y C# 3.0 así como con Visual Studio 2008, Java 1.6
- Opción para usar varias capas en los diagramas
- Opción para combinar proyectos
- Estilos de estereotipo definidos por el usuario
- Mejoras en la función de finalización automática
- Generación automática de RealizacionesDeComponente
- Opción para importar varios esquemas XML desde un directorio
- Generación automática de directorios de espacio de nombres para el código generado
- Compatibilidad con NodosDeObjeto en los diagramas de actividades
- Opción para generar vínculos relativos para la documentación UML
- Iconos de visibilidad compatibles con UML para los diagramas de clases
- Compatibilidad con asociaciones de colecciones

Novedades de la **versión 2008**:

- Generación de código Visual Basic a partir de modelos e ingeniería inversa de código Visual Basic
- Integración en Visual Studio
- Integración en Eclipse
- Opción para guardar todos los diagramas del proyecto como imágenes en un solo paso
- Títulos de línea de vida multilínea en los diagramas de secuencia, de comunicación y de ciclo de vida
- Compatibilidad con los subelementos de eventos de los diagramas de máquina de estados: `EventoRecibirSeñal`, `EventoSeñal`, `EventoEnviarSeñal`, `EventoRecibirOperación`, `EventoEnviarOperación` y `EventoDeCambio`.
- Nueva opción Ir a la operación para mensajes de llamada en diagramas de secuencia y de comunicación
- Ahora las señales pueden tener generalizaciones y sus propios atributos
- Mejoras en la compatibilidad con valores etiquetados
- Función Buscar y Reemplazar elementos de modelado

Diagramas de secuencia:

- Generación automática de respuestas (sintácticamente correctas) cuando se añaden

mensajes a diagramas de secuencia
- Los nombres de operaciones estáticas aparecen subrayados en los diagramas de secuencia

Mejoras en el cuadro de diálogo Invalidar o implementar operaciones:
- Puede hacer visible las operaciones de plantillas enlazadas o invalidarlas
- Puede ver qué operaciones son abstractas o están sin definir

© 2014 Altova GmbH

Altova UModel® 2015

Tutorial de UModel

4 Tutorial de UModel

Este tutorial describe paso a paso cómo crear un proyecto de modelado en UModel.

El tutorial se ocupa principalmente del proceso de ingeniería directa, es decir, de cómo usar UModel para crear diagramas UML y generar código. Después describe el proceso de ingeniería inversa, tanto desde el punto de vista del código como del modelo.

Estos son los diagramas UML que abarca el tutorial, que además explica cómo manipular sus elementos de modelado:

- Casos de uso
- Diagramas de clase
- Diagramas de objetos
- Diagramas de componentes
- Diagramas de implementación

El apartado sobre ingeniería de ida y vuelta (modelo - código - modelo) explica:

- Cómo generar código desde UModel
- Cómo agregar una operación nueva al código externo
- Cómo combinar el código externo con el modelo de UModel otra vez

El apartado sobre ingeniería de ida y vuelta (código - modelo - código) explica:

- Cómo importar código generado con XMLSpy desde un directorio o archivo de proyecto
- Cómo agregar una clase nueva al modelo generado en UModel
- Cómo combinar el proyecto actualizado con el código externo

Todos los archivos utilizados en este tutorial están disponibles en la carpeta `C:\Documents and Settings\All Users\Application Data\Altova`. Cuando un usuario inicia la aplicación por primera vez, el archivo `Examples` del usuario se copia en la carpeta `C:\Documents and Settings\<usuario>\Mis Documentos\Altova\UModel2015\UModelExamples\`. Por tanto, es importante que no mueva, edite ni elimine los archivos de ejemplo del directorio `...\All Users\....` inicial.

BankView-start.ump

Este es el archivo de proyecto de UModel que incluye el estado inicial del ejemplo que se utiliza en el tutorial. Este archivo contiene varios diagramas y varias clases, objetos y otros elementos del modelo. Durante el tutorial se añadirán nuevos, paquetes, diagramas y otros elementos que le ayudarán a entender lo fácil que es modelar aplicaciones en UModel. Recuerde que la función de revisión de sintaxis emite un informe con errores y advertencias sobre el archivo y el tutorial explica cómo solucionar estos problemas.

BankView-finish.ump

Este es el archivo de proyecto de UModel que incluye el estado final del ejemplo que utiliza el tutorial, tras seguir paso a paso todas las instrucciones. Este archivo de proyecto es el que se utiliza para generar código y sincronizarlo con UModel.

OrgChart.zip

Este archivo está en la carpeta `C:\Documents and Settings\All Users\Application Data\Altova` y se usa para el proceso de ingeniería de ida y vuelta. Desempaquételo en la carpeta `...\UModelExamples` antes de empezar.

 © 2014 Altova GmbH

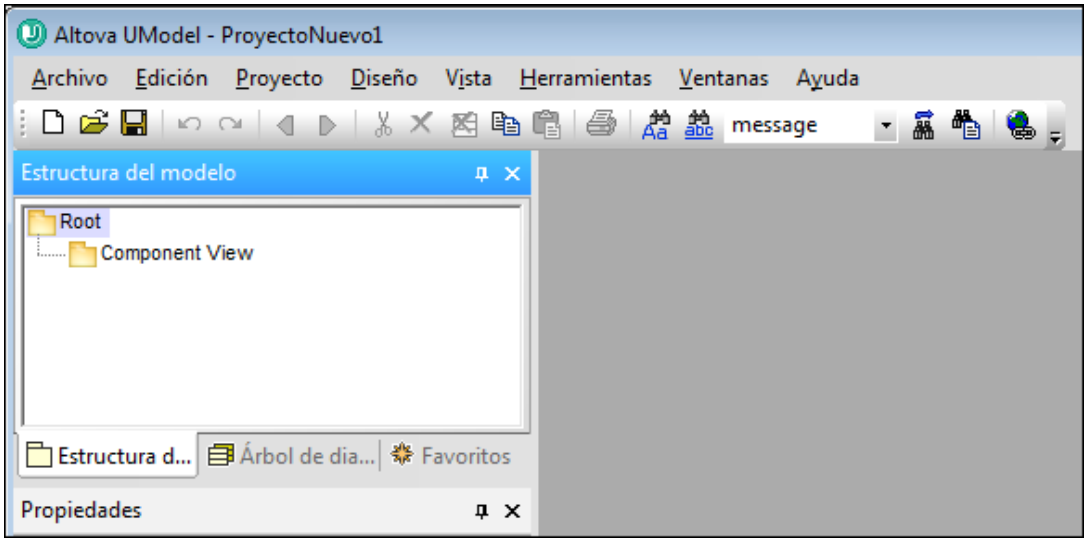
En el mismo directorio hay otros archivos de ejemplo para los lenguajes de programación **Java** y **C#** (es decir, **`Bank_Java.ump`**, **`Bank_CSharp.ump`** y **`Bank_MultiLanguage.ump`**). Estos archivos de proyecto también incluyen diagramas de secuencia que se describen en el resto de la documentación.

En la sección sobre proyectos y generación de código del tutorial encontrará información sobre cómo crear un proyecto desde cero y generar código.

4.1 Iniciar UModel

Tras instalar UModel en el equipo:

1. Inicie UModel haciendo doble clic en el icono de UModel del Escritorio o desde el menú **Inicio | Todos los programas**.
   UModel se inicia con el proyecto predeterminado **`ProyectoNuevo1`**, que aparece en la interfaz gráfica.



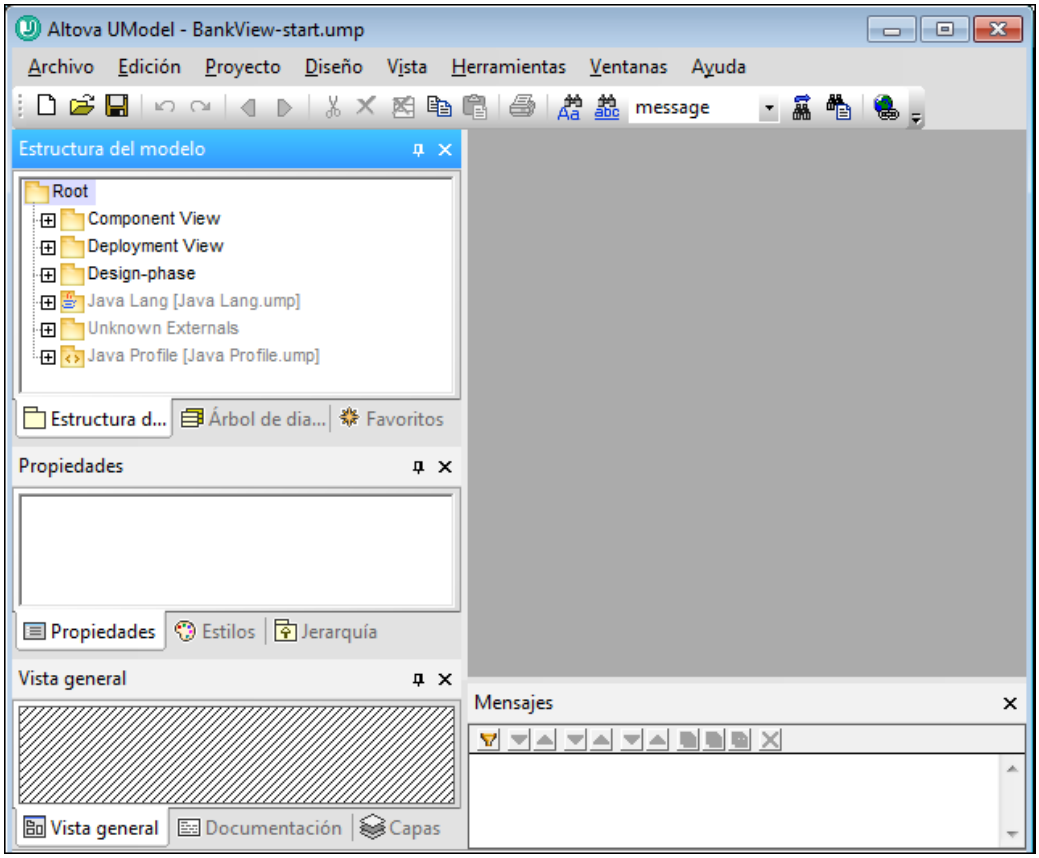
   Observe que la interfaz gráfica se divide en dos grandes secciones: el panel izquierdo formado por tres pestañas y el panel de diagramas de la derecha (que por ahora está vacío).

   En la pestaña Estructura del modelo puede ver dos paquetes predeterminados: `Root` y `Component View`. Estos dos paquetes no se pueden eliminar ni renombrar.

Para abrir el proyecto BankView-start:

1. Seleccione la opción de menú **Archivo | Abrir** y navegue hasta la carpeta . . . `\UModelExamples\Tutorial` de UModel. Recuerde que también puede abrir archivos `*.ump` desde una URL (haciendo clic en el botón **Cambiar a URL**).
2. Abra el archivo de proyecto **BankView-start.ump**.
   El archivo de proyecto se carga en UModel. Bajo el paquete `Root` aparecen ahora varios paquetes predefinidos. Observe que por ahora la ventana principal sigue vacía.

 © 2014 Altova GmbH



El panel superior izquierdo sirve para tener varias vistas del proyecto de modelado:

- La pestaña *Estructura del modelo* muestra todos los elementos de modelado del proyecto de UModel. Los elementos se pueden manipular en esta pestaña directamente, usando las teclas de edición estándar y operaciones de arrastrar y colocar.
- La pestaña *Árbol de diagramas* ofrece acceso rápido a los diagramas que componen el proyecto, independientemente de su posición en la estructura del proyecto. Los diagramas aparecen agrupados por tipo.
- La pestaña *Favoritos* es un repositorio de elementos de modelados que el usuario puede personalizar. En esta pestaña puede colocar todo tipo de elementos de modelado, haciendo clic en el comando **Agregar a favoritos** del menú contextual.

El panel central izquierdo ofrece varias vistas de determinadas propiedades del modelo:

- La pestaña *Propiedades* muestra las propiedades del elemento que está seleccionado en la *Estructura del modelo* o en el panel de diagramas. Las propiedades de los elementos se pueden definir y actualizar en esta pestaña.
- La pestaña *Estilos* muestra los atributos de los diagramas o de los elementos que están visibles en el panel de diagramas. Estos atributos de estilo se dividen en dos categorías: formato y presentación.
- La pestaña *Jerarquía* muestra todas las relaciones que tiene el elemento de modelado seleccionado. El elemento de modelado puede estar seleccionado en el diagrama, en la *Estructura del modelo* o en *Favoritos*.

El panel inferior izquierdo está compuesto por:

- La pestaña *Vista general*, que muestra la estructura general del diagrama activo.
- La pestaña *Documentación*, que sirve para documentar las clases una a una.
- La pestaña *Capas*, que sirve para definir todas las capas del diagrama. Puede ver tantas capas como quiera, bloqueadas u ocultas. Las capas sirven para crear agrupaciones lógicas de elementos de modelado en un diagrama.

Iconos de los elementos de modelado en la *Estructura del modelo*

Tipos de paquetes:

Paquete UML
Paquete raíz del espacio de nombres Java
Paquete raíz del espacio de nombres C#
Paquete raíz del espacio de nombres Visual Basic
Paquete raíz de XML Schema
Paquete de código Java, C# y VB (las declaraciones de paquete se crean cuando se genera el código)

Tipos de diagramas:

Diagrama de actividades
Diagrama de clases
Diagrama de comunicación
Diagrama de componentes
Diagrama de estructura de un compuesto
Diagrama de implementación
Diagrama global de interacción
Diagrama de objetos
Diagrama de paquetes
Diagrama de perfil
Diagrama de secuencia
Diagrama de máquina de estados
Diagrama de ciclo de vida
Diagrama de caso de uso
Diagrama de esquema XML
Diagramas BPMN
Diagramas SysML (9 tipos)

Tipos de elementos:

Los elementos que están visibles en el diagrama activo aparecen con un punto azul en la parte inferior. En este caso se trata de un elemento de tipo clase.

Instancia de clase / objeto
Slot de instancia

Clase
Propiedad
Operación
Parámetro

Actor (visible en el diagrama de casos de uso activo)
Caso de uso
Componente
Nodo

 © 2014 Altova GmbH

Artefacto
Interfaz

Relaciones (o paquete)
Restricciones

4.2 Casos de uso

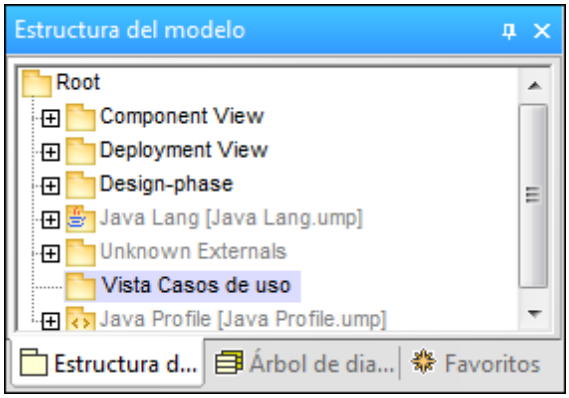
Sitio web de Altova: diagramas de casos de uso UML

En este apartado del tutorial aprenderá a:

- Agregar un **paquete** nuevo al proyecto.
- Agregar un **diagrama** de casos de uso nuevo al proyecto.
- Agregar **elementos** al diagrama y definir dependencias entre ellos.
- Alinear los elementos y ajustar su tamaño.

Para agregar un paquete nuevo al proyecto:

1. En la pestaña *Estructura del modelo* haga clic con el botón secundario en el paquete `Root` y seleccione **Elemento nuevo | Paquete**.
2. Escriba el nombre del nuevo paquete (p. ej. `Vista Casos de uso`) y pulse **Entrar**.

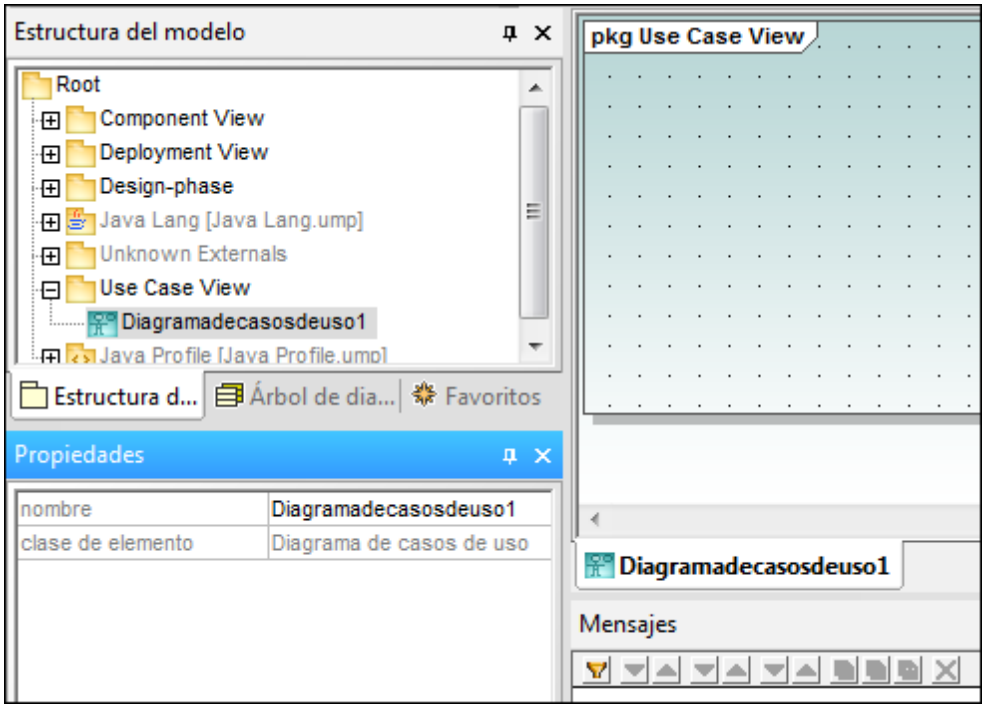


Para más información sobre los paquetes y sus propiedades consulte el apartado Paquetes de la documentación.

Agregar un diagrama a un paquete:

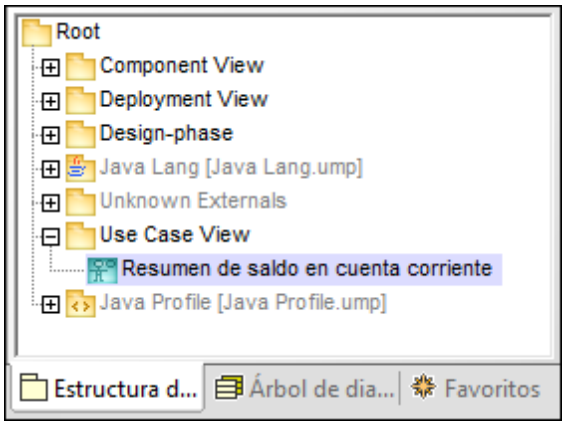
1. Haga clic con el botón secundario en el paquete `Vista Casos de uso` que acaba de crear.
2. Seleccione el comando **Diagrama nuevo | Diagrama de casos de uso**.

 © 2014 Altova GmbH



Al paquete se añade un diagrama de casos de uso (en la *Estructura del modelo*) y en el panel de diagramas aparece la pestaña *Diagramacasosdeuso1*, que lleva el nombre predeterminado creado automáticamente para el diagrama.

3. En la *Estructura del modelo* haga doble clic en el nombre del diagrama, escriba `Resumen de saldo en cuenta corriente` y pulse **Entrar** para confirmar.

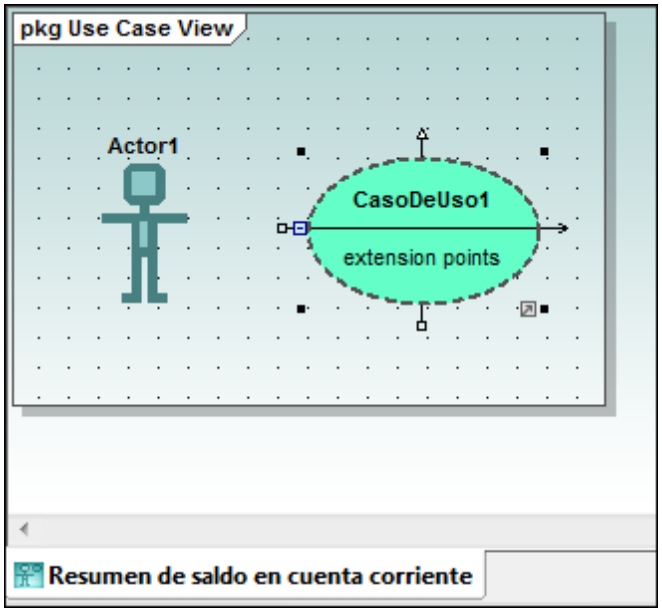


Para más información sobre los diagramas y sus propiedades consulte la sección Diagramas de la documentación.

Agregar elementos al diagrama de casos de uso:

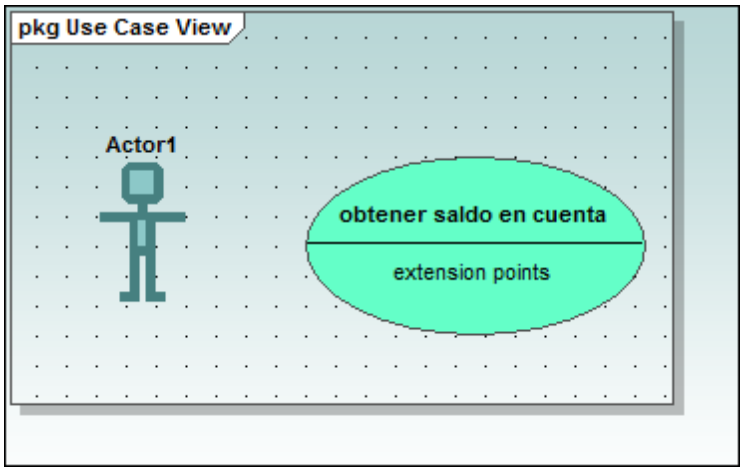
1. Haga clic con el botón secundario dentro del panel del diagrama recién creado y seleccione **Nuevo/a | Actor**.
   El elemento `actor` se inserta en la posición donde se hizo clic.
2. Haga clic en el icono **Caso de uso** de la barra de herramientas y después en el panel del diagrama para insertar el elemento.
   El elemento `CasoDeUso1` se inserta en el diagrama. Observe que el elemento y su

nombre están seleccionados y, por tanto, sus propiedades se pueden ver en la pestaña *Propiedades*.



3. Cambie el nombre de este elemento a **`obtener saldo en cuenta`** y pulse **Entrar** para confirmar. Si el nombre del elemento no está seleccionado, haga doble clic para seleccionarlo.

   Observe que el tamaño del caso de uso se ajusta automáticamente a la longitud del texto.



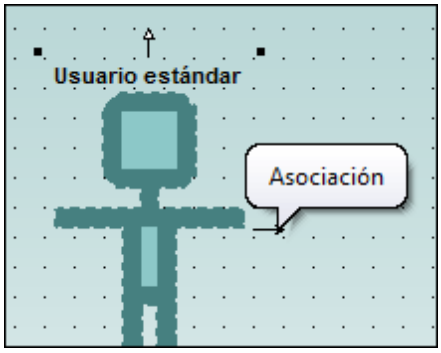
Los elementos del modelo tiene varios controladores de conexión y otros componentes que sirven para manipularlos.

Nota: use **Ctrl+Entrar** para añadir un salto de línea en el nombre del caso de uso.

Manipular elementos de UModel: controladores y compartimientos

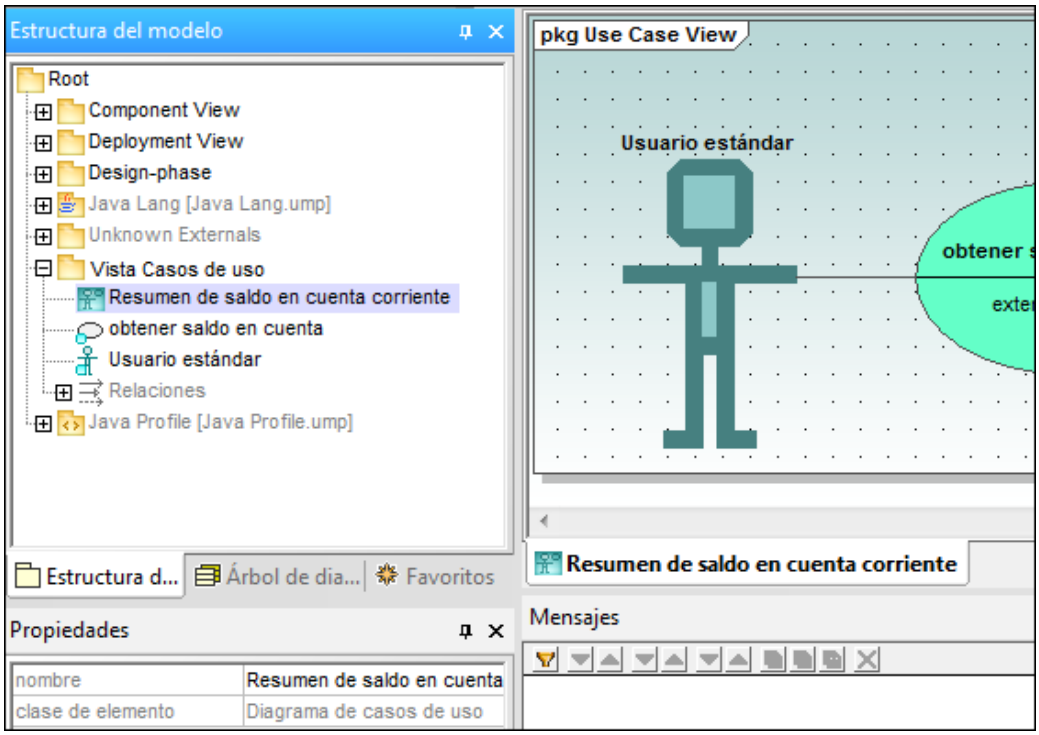
1. Haga doble clic en el texto **`Actor1`** del elemento `actor`, escriba el nuevo nombre **`Usuario estándar`** y pulse **Entrar** para confirmar.
2. Pase el cursor por encima del controlador derecho del actor.
   Aparece una nota de información rápida que dice `Asociación`.

 © 2014 Altova GmbH

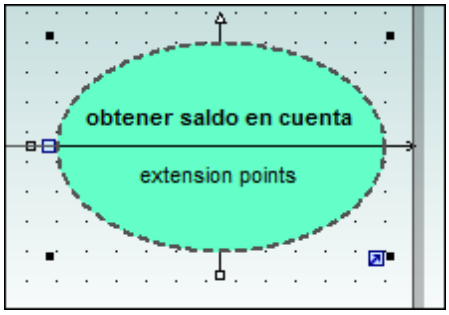


3. Haga clic en el controlador, arrastre la línea de asociación hacia la derecha y suéltela en el caso de uso **`obtener saldo en cuenta`**.

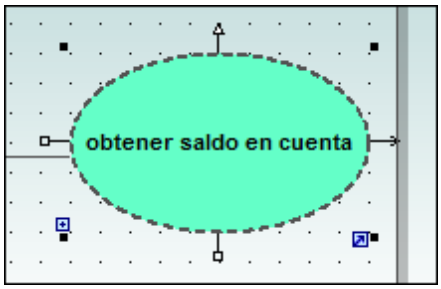
   Ahora se crea una asociación entre el actor y el caso de uso. Las propiedades de la asociación se pueden ver en la pestaña *Propiedades*. La nueva asociación también se añade a la *Estructura del modelo*, bajo el elemento `Relaciones` del paquete `Vista Casos de uso`.

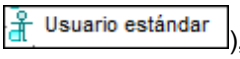


4. Haga clic en el caso de uso y arrástrelo hacia la derecha.
   Las propiedades de la asociación están visibles en el objeto asociación.
5. Haga clic en el caso de uso para seleccionarlo y después haga clic en el icono contraer situado en el borde izquierdo del caso de uso.



Esto oculta el compartimiento `puntos de extensión`.



Nota: si en la *Estructura del modelo* el icono del elemento incluye un punto azul (p. ej. 

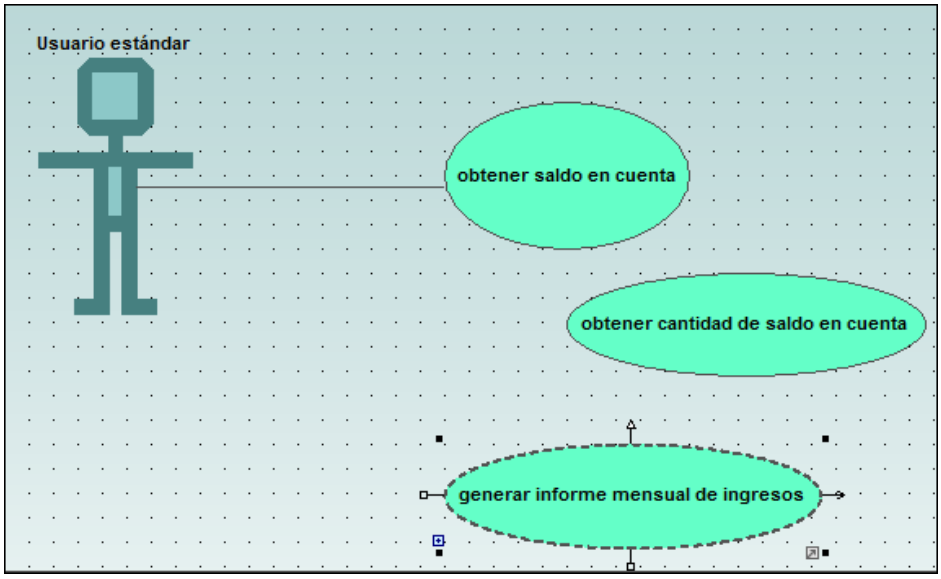
), esto significa que el elemento está visible en el panel de diagramas actual. Al ajustar el tamaño del actor también se ajusta el campo de texto, que puede ser multilínea. Puede insertar un salto de línea haciendo clic en **Ctrl+Entrar**.

Terminar el diagrama de casos de uso:

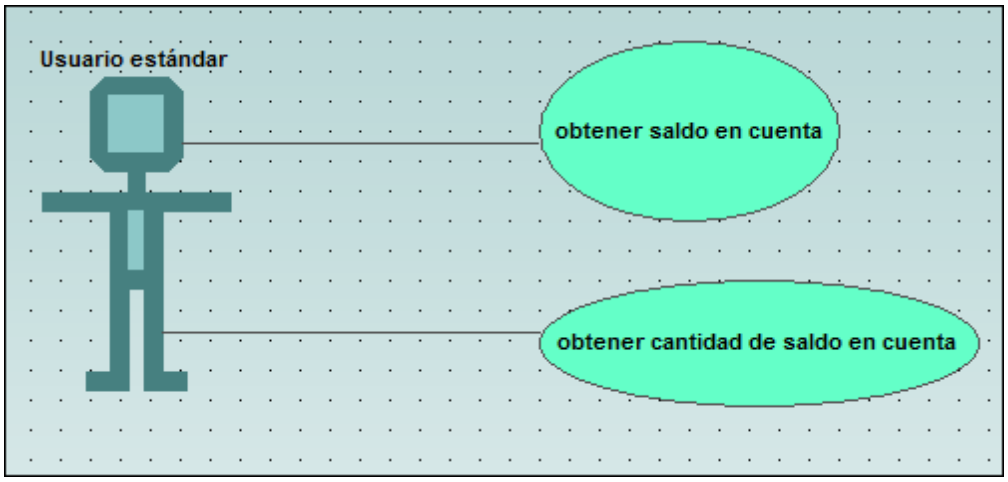
Ahora, usando los métodos aprendidos hasta ahora:

1. Haga clic en el icono **Caso de uso** de la barra de herramientas mientras pulsa la tecla **Ctrl**.
2. Haga clic en dos posiciones verticales distintas del panel del diagrama para añadir dos casos de uso más. Cuando termine puede soltar la tecla **Ctrl**.
3. Al primer caso de uso lo llamamos **`obtener cantidad de saldo en cuenta`** y al segundo **`generar informe mensual de ingresos`**.
4. Haga clic en el icono contraer de ambos casos de uso para ocultar el compartimiento de los puntos de extensión.

 © 2014 Altova GmbH

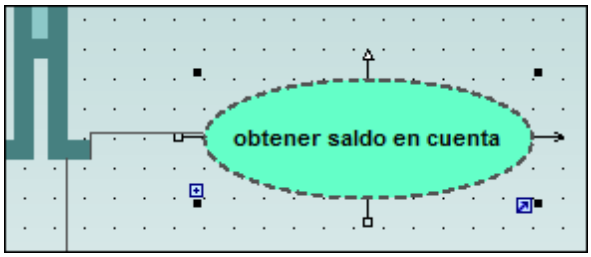


5. Haga clic en el actor y cree una asociación entre **`Usuario estándar`** y **`obtener saldo en cuenta`**.

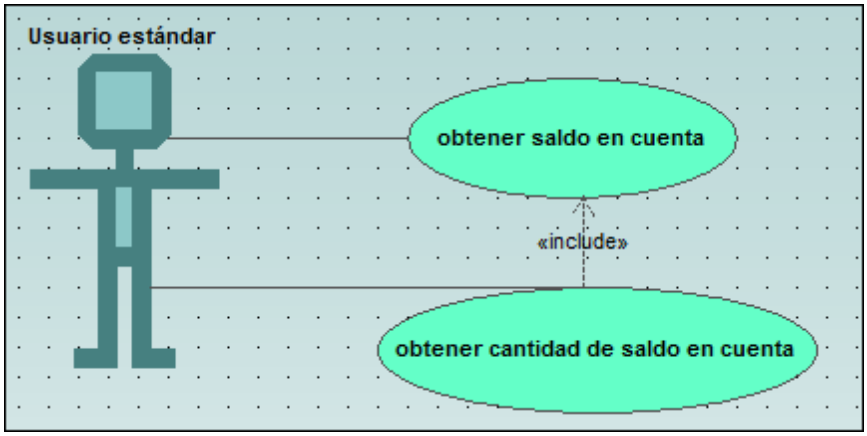


Para crear una dependencia de inclusión entre los casos de uso (creando un caso de uso subordinado):

1. Haga clic en el controlador *Inclusión* del caso de uso **`obtener cantidad saldo en cuenta`**, situado en el borde inferior, y arrástrelo hasta el caso de uso **`obtener saldo en cuenta`**.



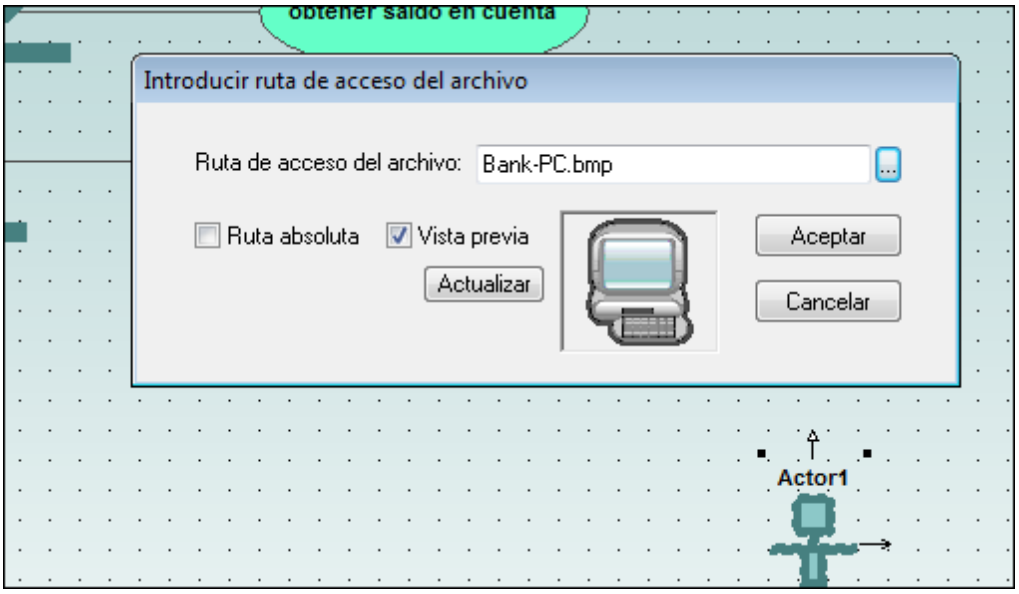
Se crea la dependencia de inclusión y el estereotipo `include` aparece en la flecha de línea discontinua.



Insertar actores definidos por el usuario:

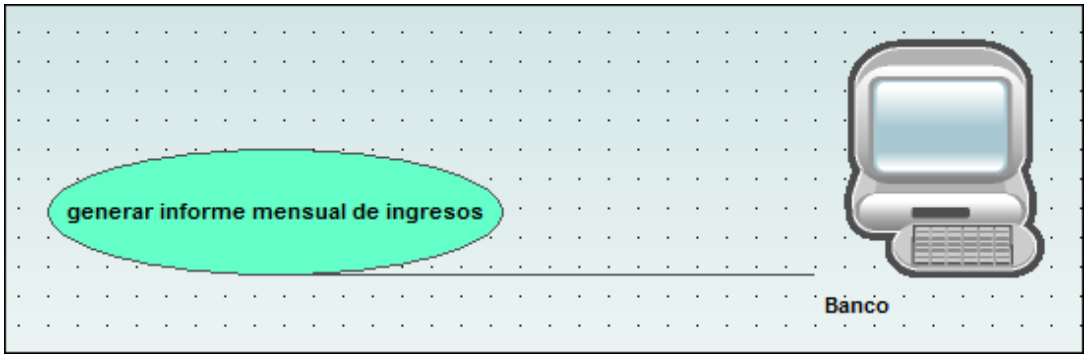
El actor del caso de uso **`generar informe mensual de ingresos`** no es una persona, sino un trabajo automatizado de procesamiento por lotes que ejecuta una computadora del banco.

1. Inserte un actor en el diagrama haciendo clic en el icono **Actor** de la barra de herramientas.
2. Cambie el nombre del actor por **`Banco`**.
3. En la pestaña *Propiedades*, junto a la entrada `nombre de archivo del icono`, haga clic en el icono ...
4. Busque el mapa de bits que desea usar (en este caso `Bank-PC.bmp`).
5. Desactive la casilla *Ruta absoluta* para que la ruta de acceso sea relativa. Aparece una vista previa del archivo seleccionado.



6. Haga clic en **Aceptar** para confirmar e insertar el nuevo actor.
7. Mueva el actor **`Banco`** a la derecha del caso de uso que está más abajo.
8. Haga clic en el icono **Asociación** → de la barra de herramientas. Seleccione el actor **`Banco`** y arrastre el puntero del mouse hasta el caso de uso **`generar informe mensual de ingresos`**. Esta es otra manera de crear asociaciones.

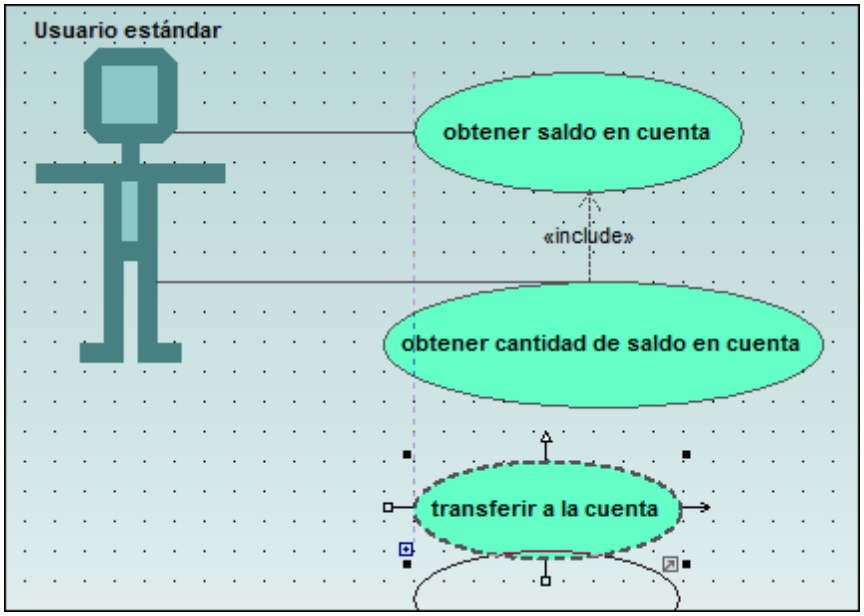
 © 2014 Altova GmbH



Nota: utilice los valores RGB 82.82.82 para el color de fondo si quiere que el mapa de bits sea transparente.

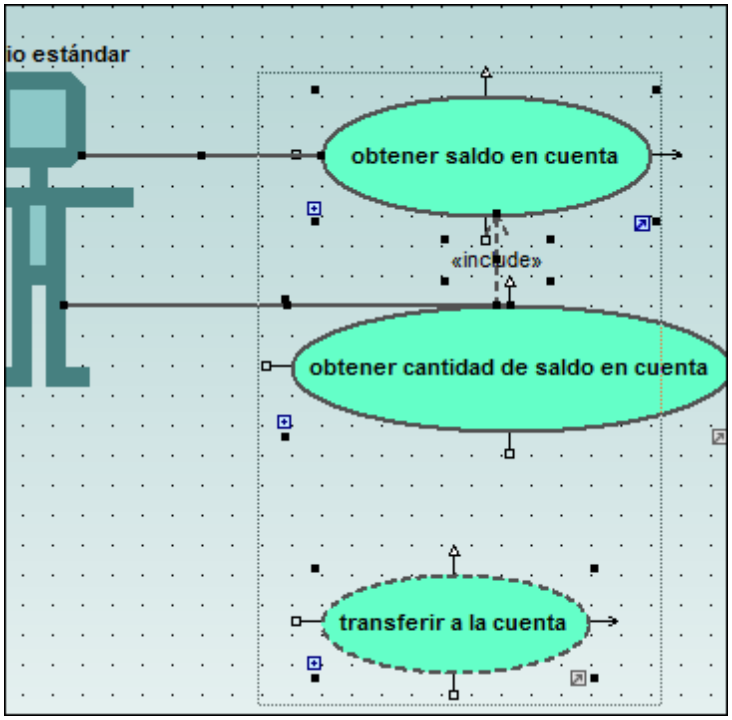
Líneas de ajuste mientras arrastramos elementos

Cuando se arrastran componentes del diagrama, aparecen unas líneas de cuadrícula que ayudan a alinear unos elementos con otros en el diagrama. Estas líneas de guía se pueden habilitar/deshabilitar con la opción de menú **Herramientas | Opciones | Vista** > *Alineación* > *Habilitar líneas de ajuste*).



Alinear los elementos y ajustar su tamaño:

1. Haga clic en el fondo del diagrama y arrastre el puntero para seleccionar los tres casos de uso.
   Observe que el último de los casos de uso seleccionados aparece con un contorno discontinuo, no solo en el panel del diagrama sino también en la *Vista general*.

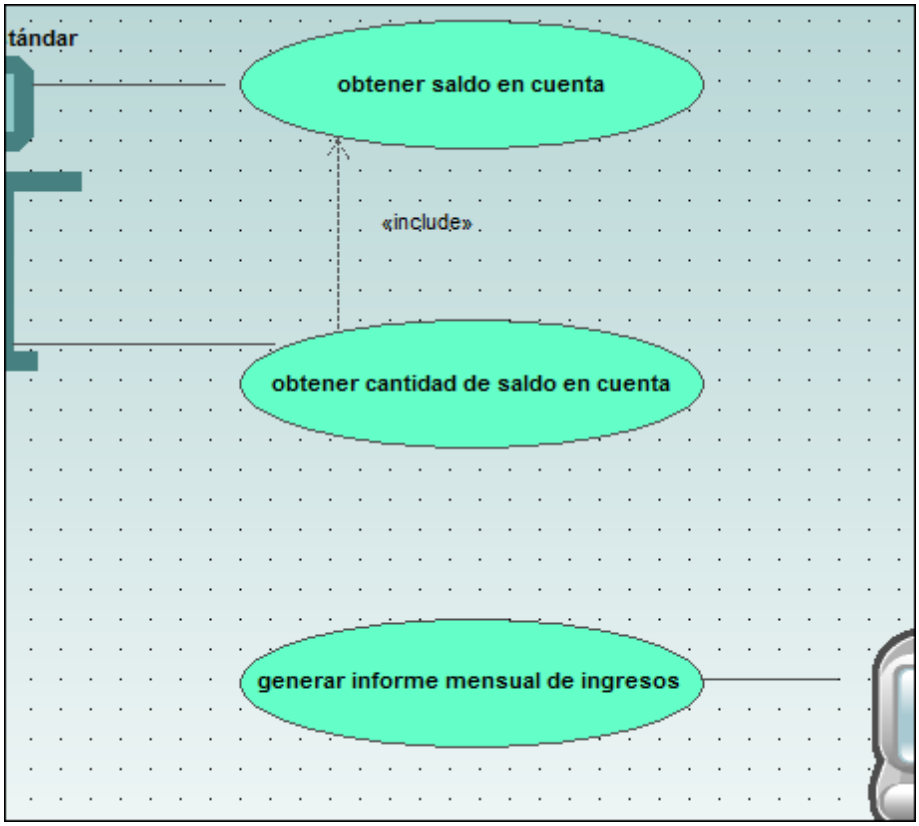


Ahora están seleccionados todos los casos de uso y el último se utiliza como referencia para los ajustes que vamos a hacer ahora.

2. Haga clic en el icono **Igualar tamaño** de la barra de herramientas.
3. Haga clic en el icono **Centrar horizontalmente** para alinear todos los casos de uso.
   Ahora los elementos caso de uso se centran en el panel y todos tienen el mismo tamaño.

Nota: también puede seleccionar varios elementos a la vez si pulsa la tecla **Ctrl** mientras hace clic en los elementos.

 © 2014 Altova GmbH


tándar
obtener saldo en cuenta
«include»
obtener cantidad de saldo en cuenta
generar informe mensual de ingresos

4.3 Diagramas de clases

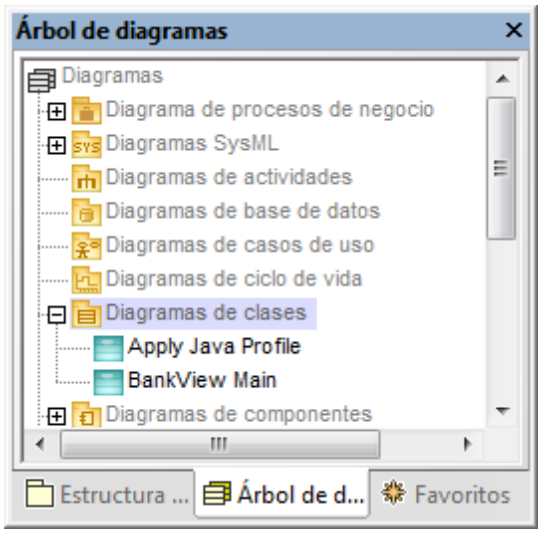
Sitio web de Altova: diagramas de clases UML

En este apartado del tutorial aprenderá a:

- Agregar una clase abstracta nueva (la clase `Account`), atributos y operaciones.
- Crear una asociación compuesta entre `Bank` y `Account`.

Para abrir un diagrama distinto en UModel:

1. Haga clic en la pestaña *Árbol de diagramas*.
2. Expanda el paquete *Diagramas de clases* para ver su contenido.



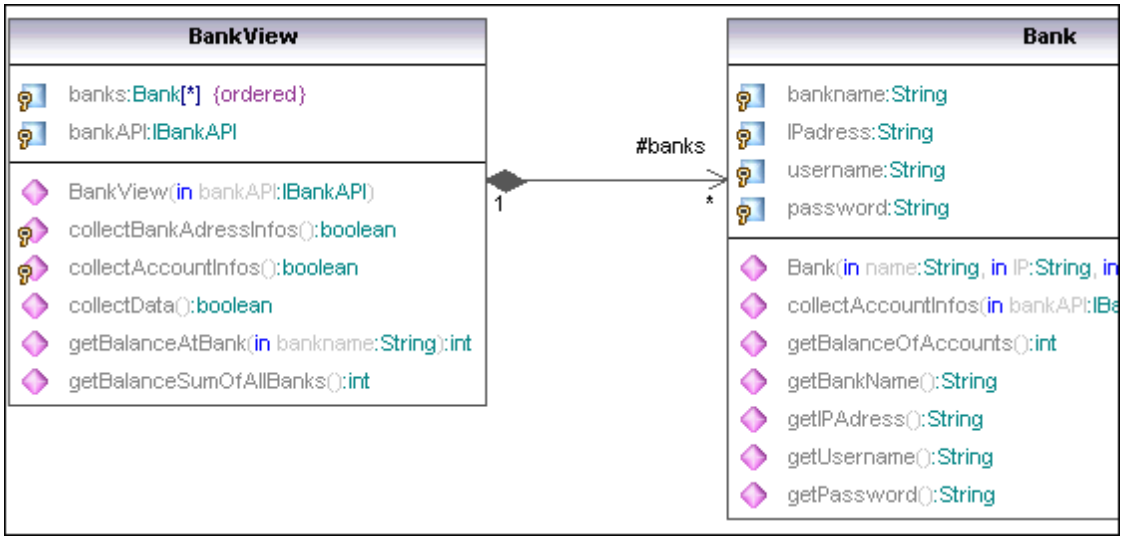
En la lista aparecen todos los diagramas de clases del proyecto.

3. Haga doble clic en el icono del diagrama `BankView Main` (en el icono, no en el nombre).
   El diagrama de clases se abre en el panel de diagramas.

Nota: también puede hacer doble clic en el icono de diagrama de clases de la *Estructura de modelos*, debajo del paquete `BankView`.

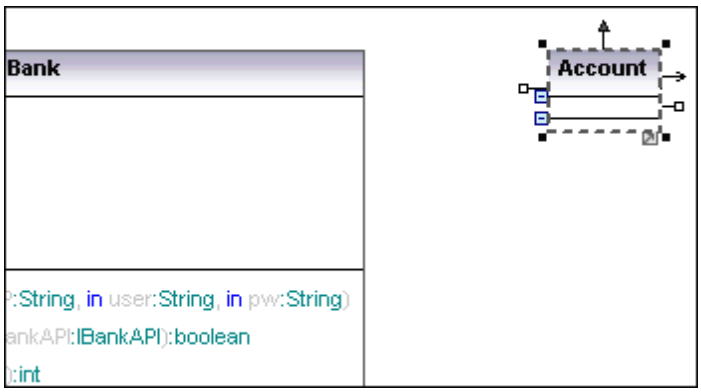
En el diagrama de clases puede ver dos clases concretas que tienen una asociación compuesta.

 © 2014 Altova GmbH



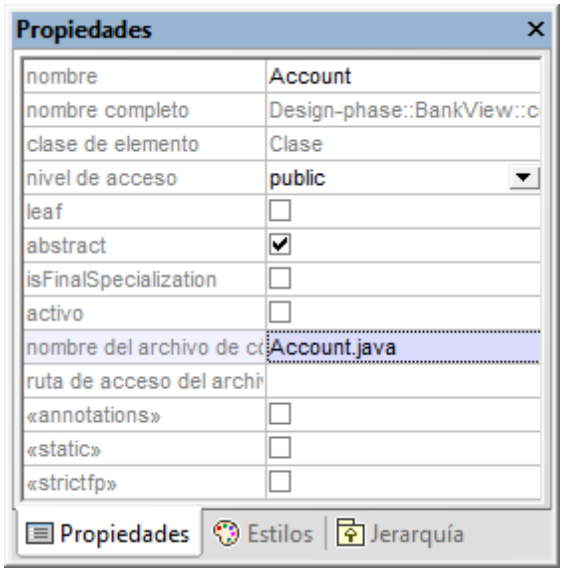
Para agregar una clase nueva y definirla como clase abstracta:

1. Haga clic en el icono **clase** de la barra de herramientas y después haga clic a la derecha de la clase `Bank`. Esto inserta una clase nueva llamada `Clase1`.
2. Cambie el nombre de la clase por `Account` y pulse **Entrar** para confirmar.

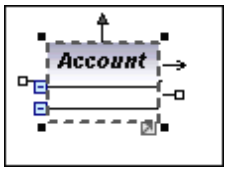


Observe que la pestaña *Propiedades* muestra las propiedades de la clase activa.

3. Marque la casilla `abstract` del panel *Propiedades* para que la clase sea abstracta.
4. Haga clic en el cuadro de texto `nombre del archivo de código` y escriba *Account.java* para definir la clase Java.

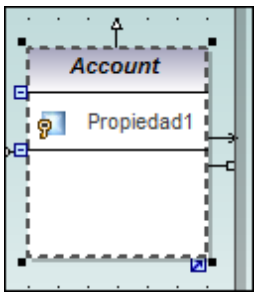


Ahora el nombre de la clase aparece en cursiva, lo cual nos sirve para identificarla como clase abstracta.



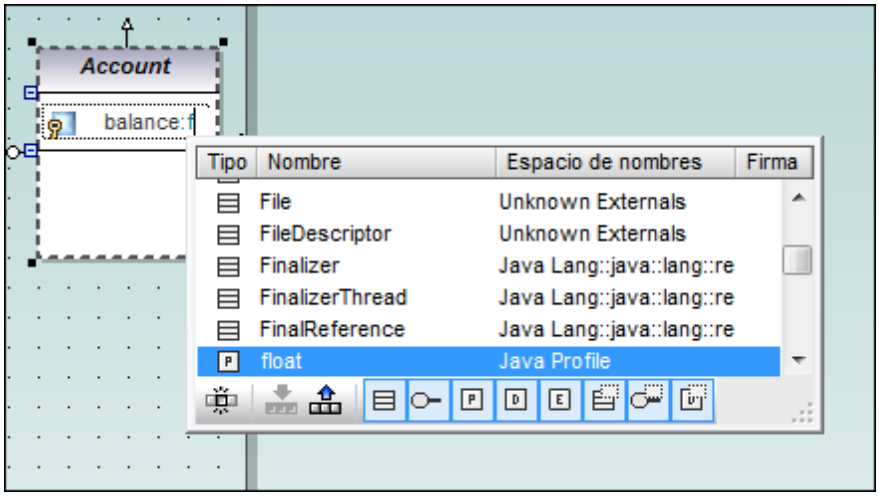
Para agregar propiedades a una clase:

1. Haga clic con el botón secundario en la clase `Account` y seleccione **Nuevo/a | Propiedad** (o pulse la tecla **F7**).
   Se inserta una propiedad predeterminada llamada `Propiedad1`, con los identificadores de estereotipo `<< >>`.

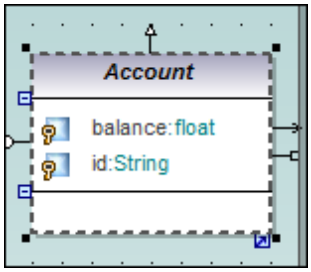


2. Cambie el nombre de la propiedad por `balance` y añada dos puntos "**:**".
   Aparece una lista desplegable con todos los tipos válidos.
3. Escriba la letra **f** y pulse **Entrar** para insertar el tipo de datos de valor devuelto `float`.
   Tenga en cuenta que esta lista desplegable distingue entre mayúsculas y minúsculas.

 © 2014 Altova GmbH

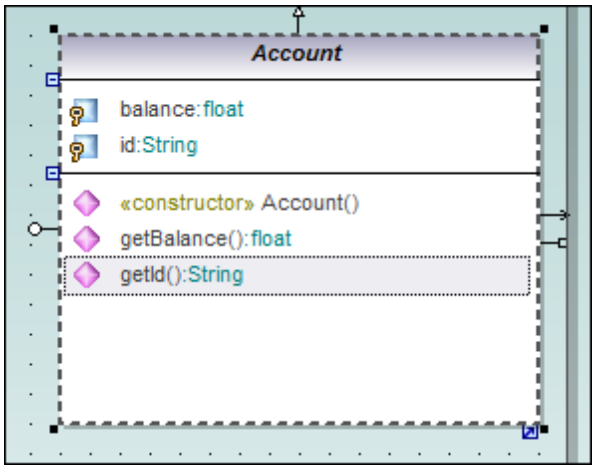


4. En la misma línea añada `=0` para definir el valor predeterminado.
5. Pulse la tecla **F7** para añadir otra propiedad más a la clase.
6. Escriba **Id:** y seleccione `String` en la lista desplegable.



Para agregar operaciones a una clase:

1. Haga clic con el botón secundario en la clase **`Account`** y seleccione **Nuevo/a | Operación** (o pulse la tecla **F8**).
2. Como constructor inserte `Account()`.
3. Ahora siga los mismos pasos que antes y cree dos operaciones más: `getBalance:float` y `getId:String`.

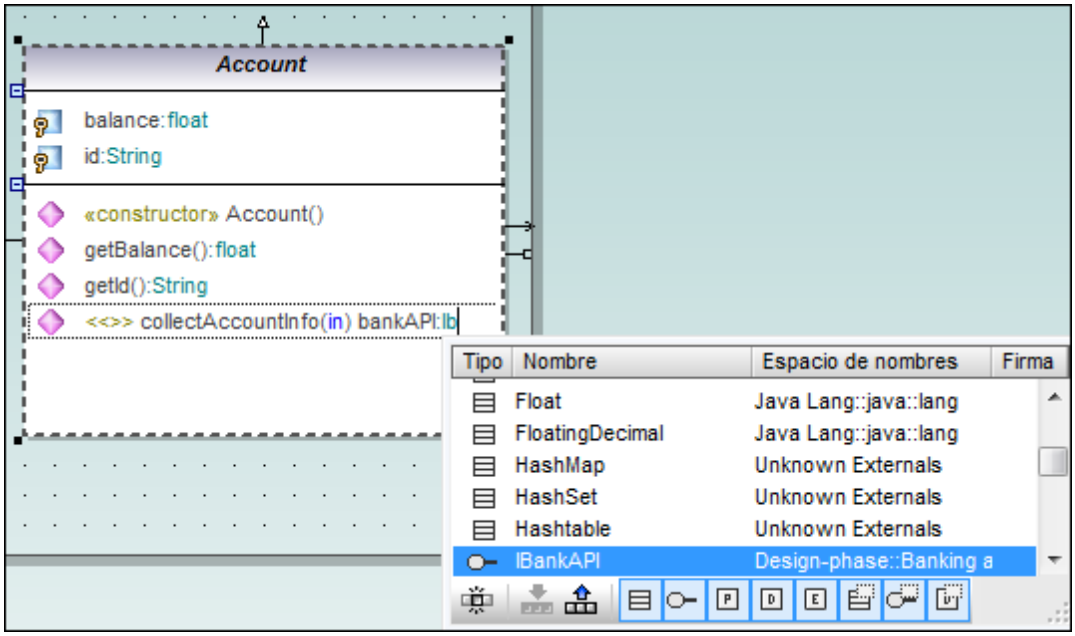


Ahora use la función de finalización automática para definir estas operaciones

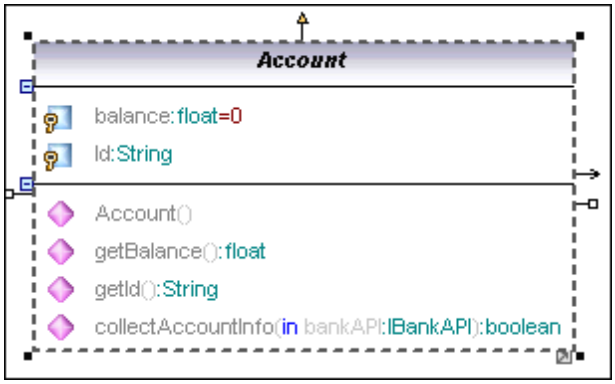
4. Pulse **F8** para crear la operación `collectAccountInfo` y escriba el carácter de

paréntesis "**(**".
Después escriba la letra **i**, lo cual abre la lista desplegable de finalización automática.
Seleccione uno de los parámetros de dirección: `in`, `inout`, `out`.

5. Seleccione `in` en la lista desplegable, inserte un carácter de espaciado y siga editando la misma línea.
6 Ahora escriba `bankAPI` y después dos puntos (:).
7. Seleccione `IBankAPI` de la lista desplegable, cierre el paréntesis con "**)**" y escriba dos puntos otra vez (:).



8. Pulse la tecla **b** para seleccionar el tipo de datos binario y después pulse **Entrar** para insertarlo.
9. Pulse **Entrar** para terminar la definición.



Nota: si hace clic en el icono de visibilidad situado a la izquierda de la operación ( ) o de la propiedad ( ), aparece una lista desplegable donde puede cambiar su nivel de acceso.

Para eliminar propiedades y operaciones de clase de un diagrama de clases:

1. Pulse **F8** y después **Entrar** para agregar la operación predeterminada `Operación1` a la

clase **Account**.

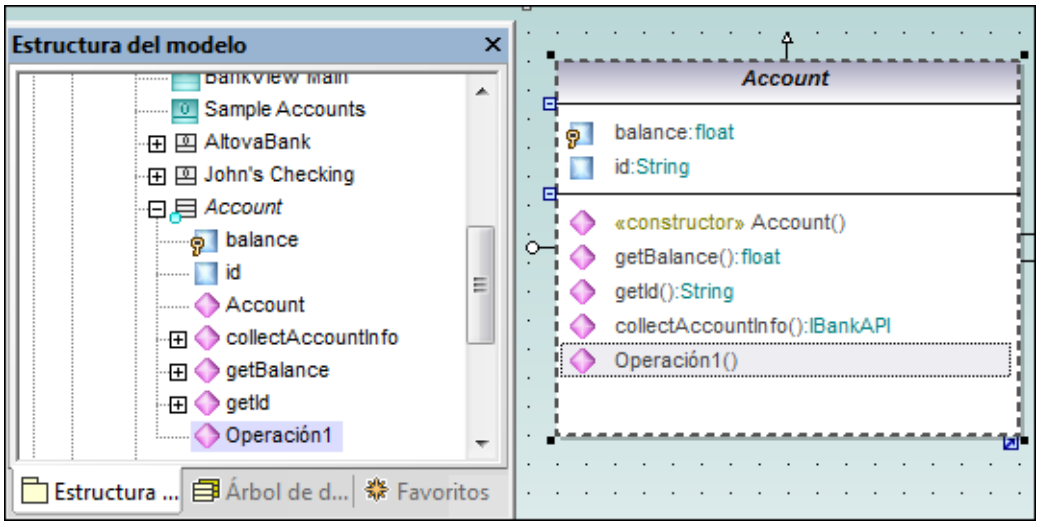
2. Haga clic en Operación1 y pulse **Supr** para eliminarla.

La aplicación pregunta si quiere eliminar el elemento del proyecto. Haga clic en **Sí** para eliminar Operación1 de la **clase** y también del **proyecto**.

Nota: si solo quiere eliminar la operación de la clase, pero no del proyecto, pulse la tecla **Ctrl +Supr**. También puede habilitar un aviso que aparecerá cuando elimine objetos (consulte **Herramientas | Opciones | Edición** para más información).

Para buscar (eliminar) propiedades de clase y opciones desde la *Estructura del modelo*:
Las propiedades y opciones también se pueden eliminar desde la *Estructura del modelo* directamente. Para hacerlo bien es importante buscar primero la propiedad que deseamos eliminar. Imaginemos que insertó la Operación1 en la clase **Account** (pulse **F8** y **Entrar** para insertarla):

1. Ahora haga clic con el botón secundario en Operación1 de la clase **Account**.
2. Seleccione la opción **Seleccionar en la estructura del modelo** (o pulse **F4**).
   El componente Operación1 aparece resaltado debajo de *Account* en la pestaña *Estructura del modelo*.

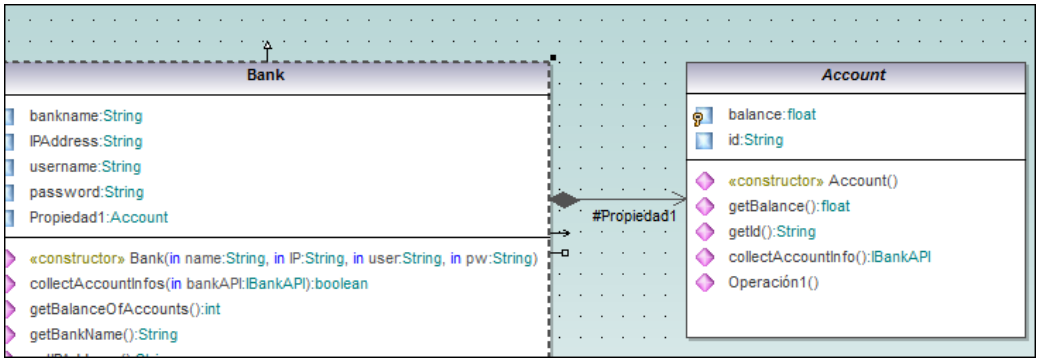


3. Pulse la tecla **Supr** para eliminar la operación de la clase y del proyecto.
   Recuerde que casi todos los elementos de modelado aparecen en la *Estructura del modelo* cuando se pulsa **F4**.

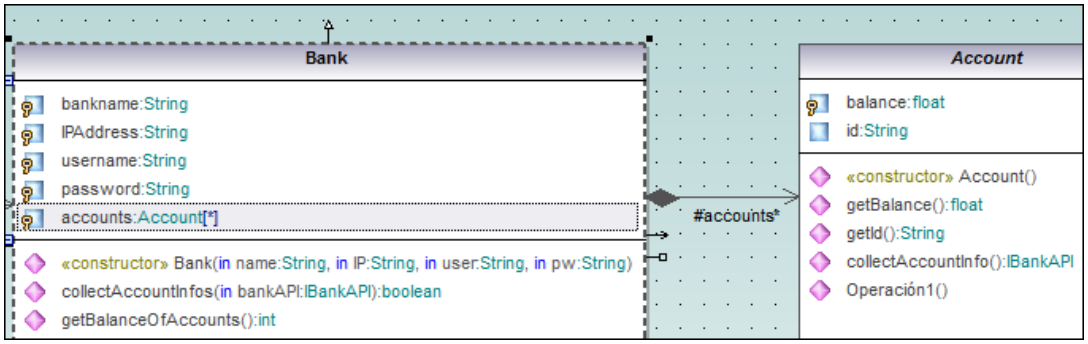
Nota: también puede navegar hasta la *Estructura del modelo* desde el panel *Propiedades*. Para más información consulte el apartado Propiedades de la sección *Interfaz del usuario*.

Crear una asociación de composición entre las clases Bank y Account:

1. Haga clic en el icono **Composición** de la barra de herramientas y cree una conexión entre la clase **Bank** y la clase **Account** arrastrando el puntero del mouse. La clase se resalta cuando la asociación se puede crear.
   En la clase **Bank** se crea una nueva propiedad (Propiedad1:Account) y una flecha de asociación compuesta une las dos clases.



2. Cambie el nombre de `Propiedad1` de la clase **`Bank`** por `accounts`, asegurándose de no eliminar la definición de tipo `Account` (que aparece en color verde).
3. Pulse la tecla **Fin** para poner el cursor al final de la línea.
4. Inserte el corchete "**[**" y seleccione ***** en la lista desplegable para definir la multiplicidad. Para confirmar pulse **Entrar**.



 © 2014 Altova GmbH

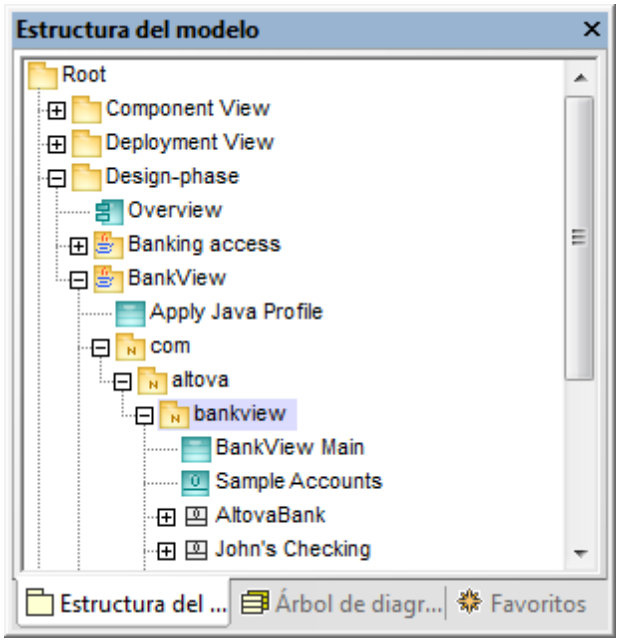
4.3.1 Crear clases derivadas

En este apartado del tutorial aprenderá a:

- Agregar al proyecto una diagrama de clases nuevo llamado **`Account Hierarchy`**
- Insertar clases ya existentes y crear una clase nueva llamada **`Savings account`**
- Crear tres clases **derivadas** de la clase base abstracta **`Account`** por medio de generalizaciones

Para crear un diagrama de clases nuevo:

1. En la *Estructura del modelo* haga clic con el botón secundario en el paquete **`bankview`** (bajo **Design-phase | BankView | com | altova**) y seleccione **Diagrama nuevo | Diagrama de clases**.

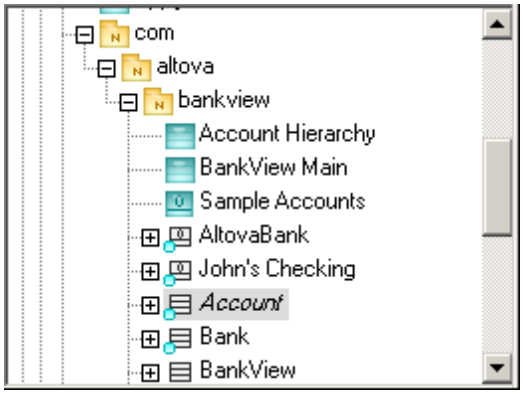


2. Haga doble clic en la nueva entrada **`Diagramadeclases1`**, llámelo **`Account Hierarchy`** y pulse **Entrar** para confirmar.

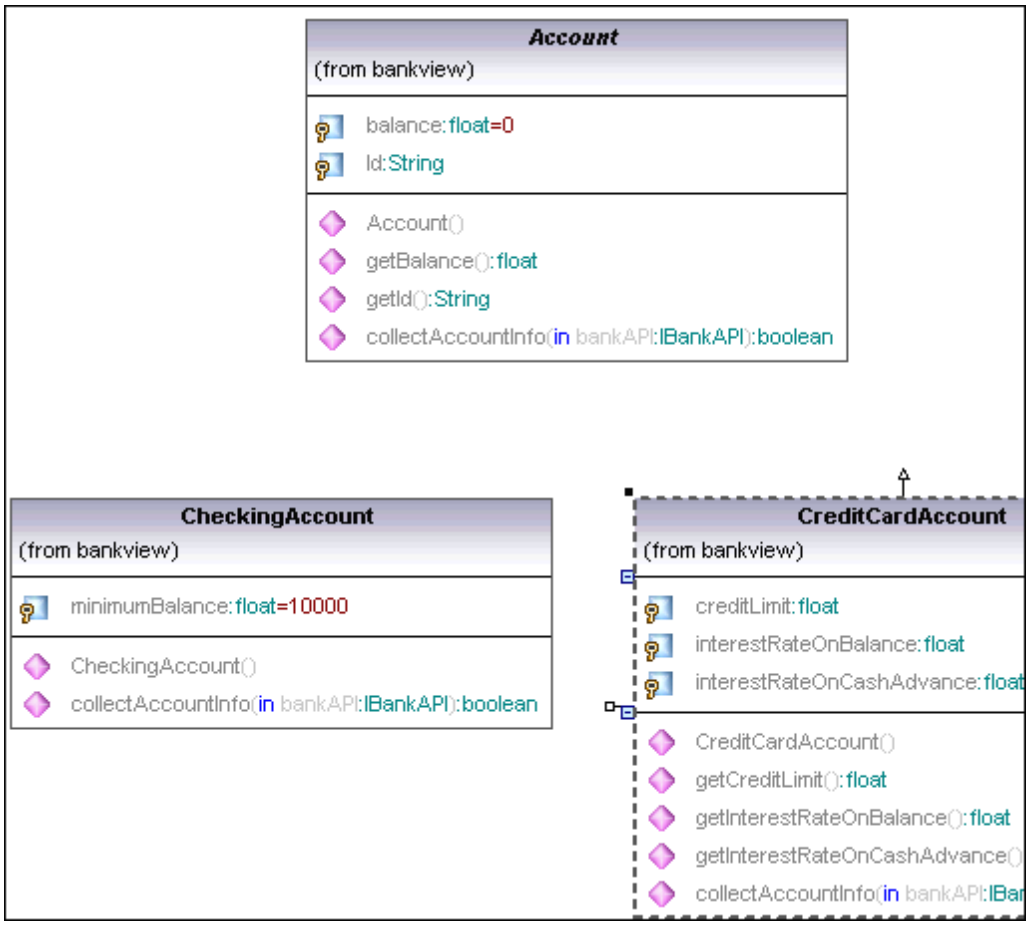
   En el área de trabajo se abre la pestaña del diagrama **`Account Hierarchy`**.

Insertar en el diagrama clases ya existentes:

1. Haga clic en la clase *`Account`* del paquete **`bankview`** (bajo **com | altova | bankview**).



2. Arrástrela hasta el panel del diagrama **`Account Hierarchy`**.
3. Haga clic en la clase **`CheckingAccount`** (del mismo paquete) y arrástrela hasta el panel del diagrama **`Account Hierarchy`**.
4. Ponga la clase debajo de la clase **`Account`** y a la izquierda.
5. De la misma manera inserte la clase **`CreditCardAccount`** y póngala a la derecha de la clase **`CheckingAccount`**.

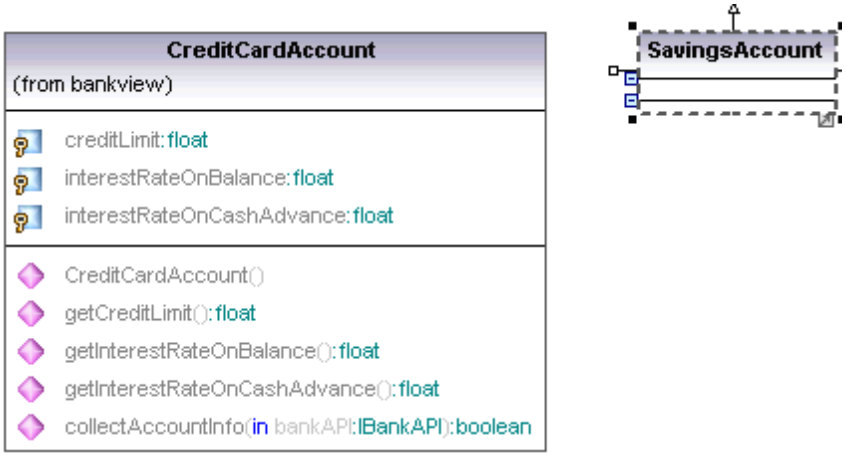


Agregar una clase nueva:

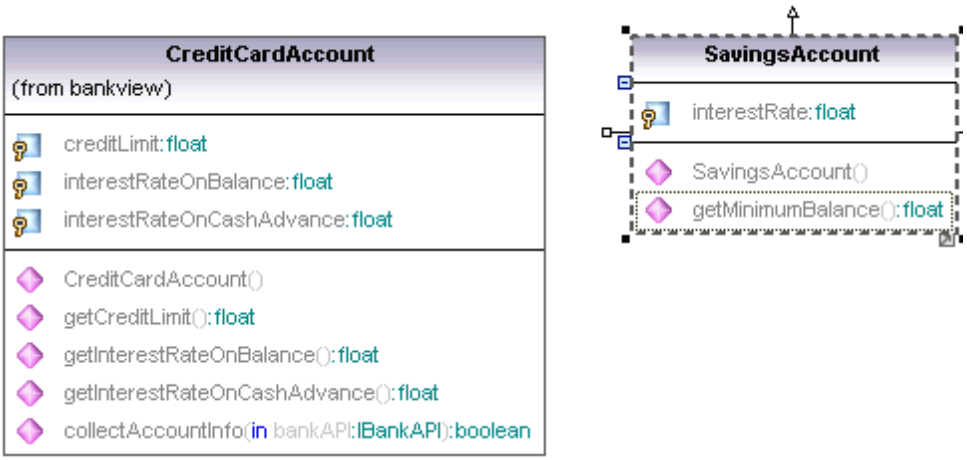
1. Haga clic con el botón secundario en el fondo del área de trabajo (p. ej. a la derecha de **`CreditAccountClass`**) y seleccione **Nuevo/a | Clase**.
   Una clase nueva se añade automáticamente al paquete correspondiente (es decir,

bankview), que contiene el diagrama de clases actual **Account Hierarchy**.

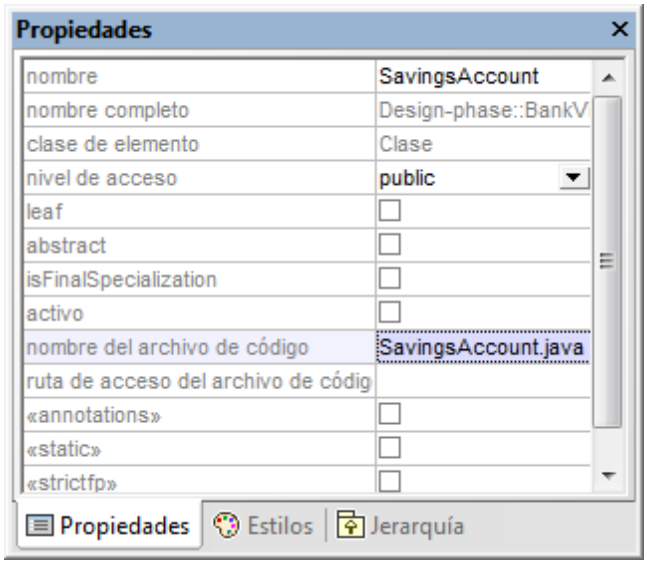
2. Haga doble clic en el nombre de la clase y cámbielo por **SavingsAccount**.



3. Pulse la tecla **F7** para añadir una propiedad nueva.
4. Escriba `interestRate` seguido de dos puntos y pulse la tecla "**f**" para seleccionar el tipo de datos `float` de la lista desplegable. Pulse dos veces la tecla **Entrar** para seleccionar ese tipo de datos y confirmar su inserción.
5. Ahora pulse la tecla **F8** y añada la operación/el constructor **SavingsAccount()**.
6. Pulse **F8** y añada la operación **getMinimumBalance:float**.



7. En la pestaña *Propiedades* haga clic en la casilla `nombre del archivo de código` y escriba *SavingsAccount.java* para definir la clase de código Java.

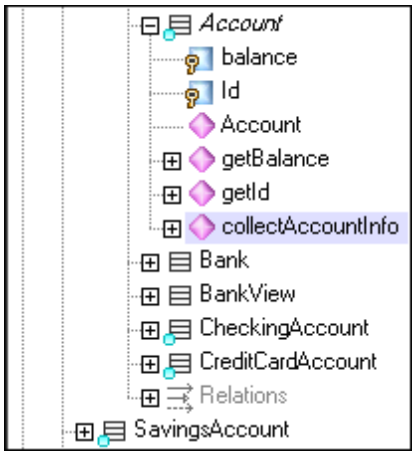


Reutilizar y copiar propiedades/operaciones ya existentes:
En UModel puede copiar (o mover) propiedades y operaciones de una clase a otra. Esto se hace mediante operaciones de arrastrar y colocar o con las teclas de acceso rápido estándar. El ámbito donde se pueden realizar estas operaciones es:

- dentro de una **misma clase** de la pestaña del diagrama
- entre **diferentes clases** de la pestaña del diagrama
- en la vista *Estructura del modelo*
- entre diferentes diagramas UML, soltando los datos copiados en una pestaña de diagrama diferente.

Para más información consulte el apartado Cortar, copiar y pegar en los diagramas de UModel.

1. En la vista *Estructura del modelo* expanda la clase ***Account*** para ver su contenido.
2. Haga clic con el botón secundario en la operación **`collectAccountInfo`** y seleccione el comando **Copiar**.



3. También en la *Estructura del modelo* haga clic con el botón secundario en la clase **`SavingsAccount`** y seleccione el comando **Pegar**.
   La operación se copia en la clase **`SavingsAccount`**, que se expande automáticamente

 © 2014 Altova GmbH

para mostrar la nueva operación.

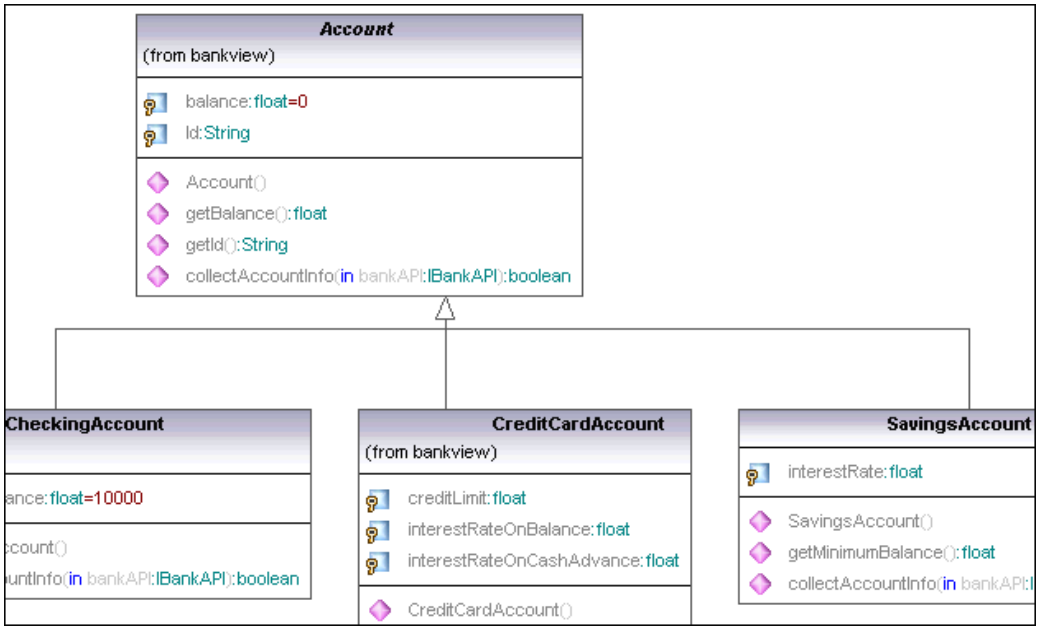
La operación nueva se puede ver en la clase `SavingsAccount` del diagrama de clases.

Nota: puede usar las teclas de acceso rápido para copiar/pegar (**Ctrl+C** o **V**) o arrastrar y colocar elementos en la vista Estructura del modelo. En algunos casos quizás sea recomendable deshabilitar las opciones de ordenación para poder colocar las operaciones entre determinados componentes.

Para crear clases derivadas (generalización/especialización):

Llegados a este punto, el diagrama de clases contiene la clase abstracta ***Account*** y tres clases concretas. Ahora queremos definir, o crear, una relación de generalización/especialización entre ***Account*** y las otras clases. Es decir, queremos crear tres clases concretas derivadas.

1. Haga clic en el icono **Generalización** de la barra de herramientas y pulse la tecla **Ctrl** al mismo tiempo.
2. Sin dejar de pulsar la tecla **Ctrl**, haga clic en `CreditCardAccount` (la clase que está en el medio del área de trabajo) y arrastre el puntero hasta la clase `Account`.
3. Sin dejar de pulsar la tecla **Ctrl**, haga clic en la clase `CheckingAccount` y arrastre el puntero hasta la **punta de flecha** de la generalización creada en el paso anterior.
4. Sin dejar de pulsar la tecla **Ctrl**, haga clic en la clase `SavingsAccount` y arrastre el puntero hasta la **punta de flecha** de la generalización creada en el paso anterior. Ya puede dejar de pulsar la tecla **Ctrl**.
5. Entre las tres subclases y la superclase `Account` se crean flechas de generalización.



 © 2014 Altova GmbH

4.4 Diagramas de objetos

Sitio web de Altova: Diagramas de objetos UML

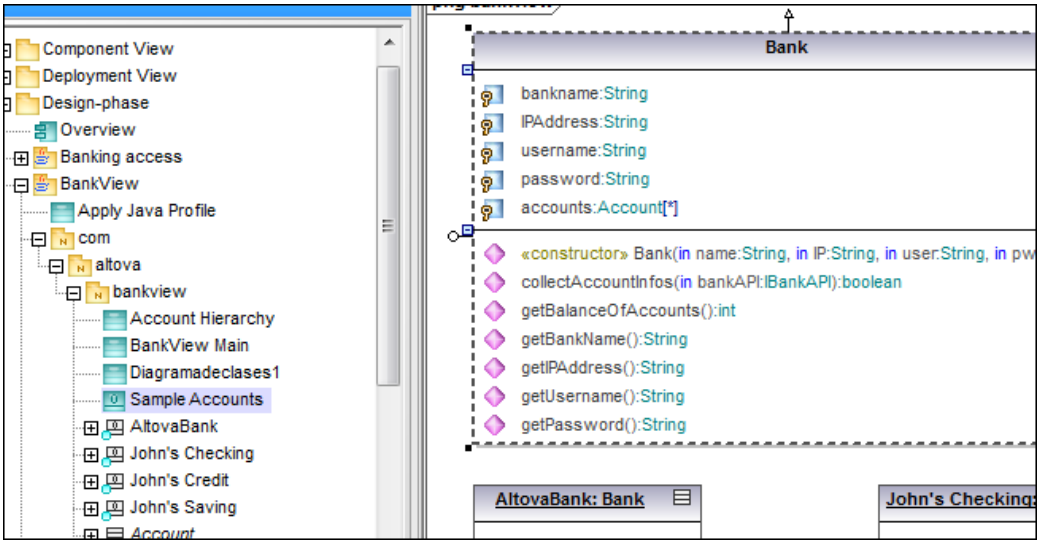
En este apartado del tutorial aprenderá a:

- Combinar **diagramas de clases** y **de objetos** en un solo diagrama para obtener una visión de conjunto de los objetos existentes en cada momento.
- Crear objetos/instancias y definir relaciones entre ellos.
- Dar formato a las asociaciones y vínculos.
- Introducir datos reales en objetos e instancias.

Para abrir el diagrama de objetos:

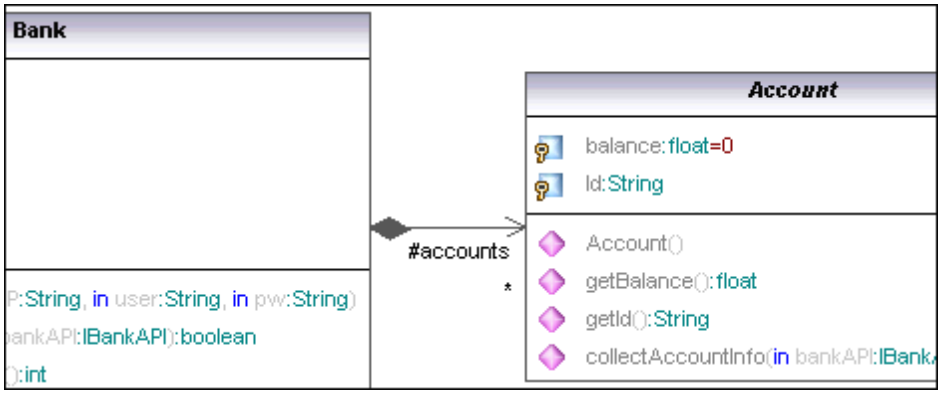
1. En la vista *Estructura del modelo* haga doble clic **en el icono** del diagrama **`Sample Accounts`** del paquete **`bankview`**.

   `AltovaBank:Bank` es el objeto/la instancia de la clase **`Bank`**, mientras que **`John's checking: CheckingAccount`** es una instancia de la clase **`CheckingAccount`**.



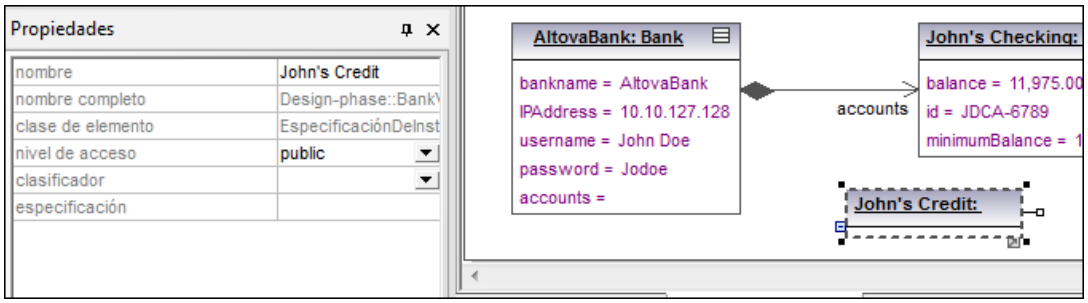
Para insertar una clase en un diagrama de objetos:

- En la *Estructura del modelo* haga clic en el icono de la clase *Account* y arrástrelo hasta la pestaña del diagrama **`Sample Accounts`**.
  La asociación compuesta definida previamente en el diagrama **`BankView Main`** se crea automáticamente.



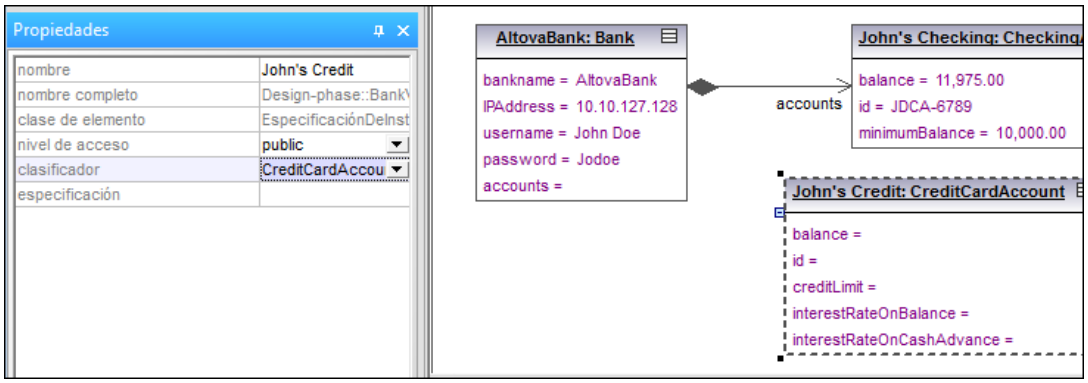
Para agregar un objeto nuevo / una instancia nueva seleccionando su tipo:

1. Haga clic en el icono **EspecificaciónDeInstancia** de la barra de herramientas y después haga clic justo debajo del objeto `John's Checking` en el área de trabajo.
2. Cambie el nombre de la instancia por `John's Credit` y pulse la tecla **Entrar**.



Si la instancia está activa, todas sus propiedades aparecen en la pestaña *Propiedades*.

3. Haga clic en el cuadro combinado `clasificador` y seleccione la opción **CreditCardAccount** de la lista desplegable.

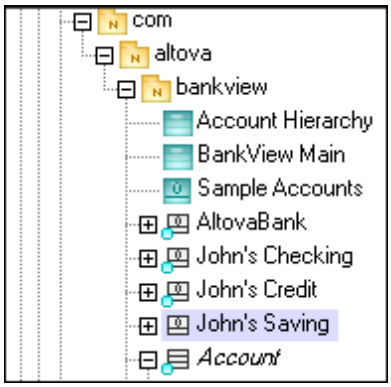


Puede ocultar/mostrar el contenido de un objeto haciendo clic con el botón secundario en la especificación de instancia y seleccionando el comando **Mostrar/ocultar el contenido del nodo**.

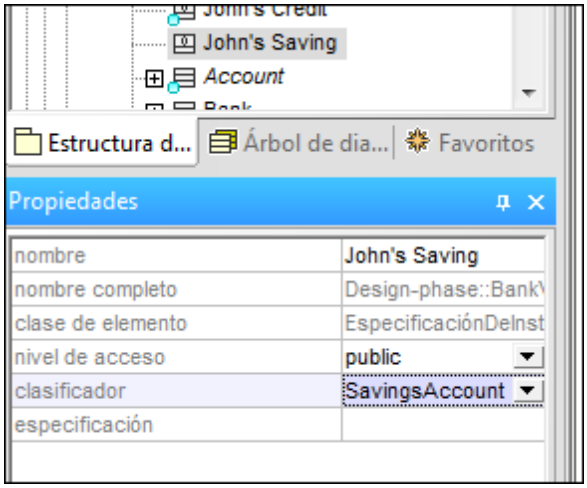
Para agregar un objeto nuevo en la vista *Estructura del modelo* (y después insertarlo en

el diagrama):

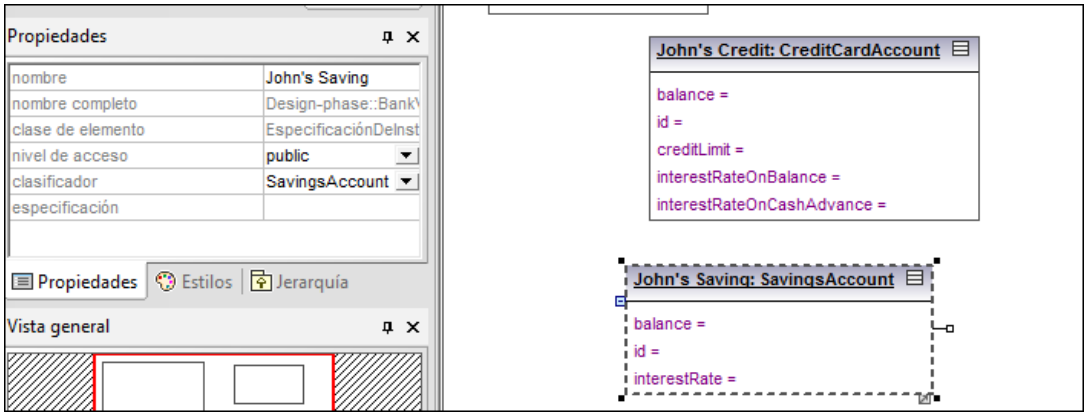
1. En la *Estructura del modelo* haga clic con el botón secundario en el paquete **`bankview`** y seleccione el comando **Elemento nuevo | EspecificaciónDeInstancia**.
2. Cambie el nombre predeterminado del objeto por **`John's Saving`** y pulse **Entrar**.
   El objeto nuevo se añade al paquete y se coloca en la posición correcta.



3. Compruebe que el objeto está seleccionado en la pestaña Estructura del modelo.
   Haga clic en el cuadro combinado `clasificador` de la pestaña *Propiedades* y seleccione la opción **SavingsAccount**.



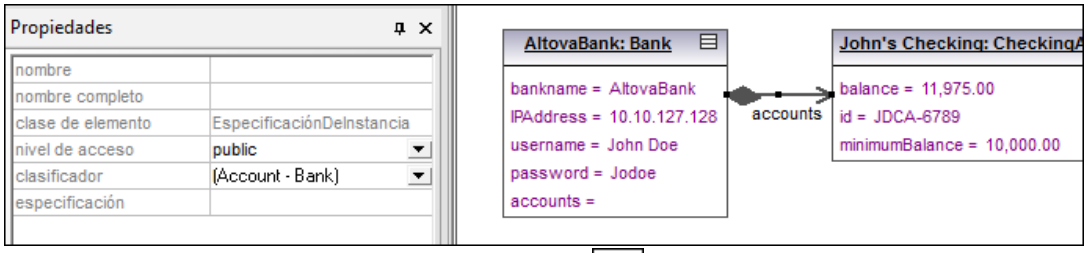
4. Ahora arrastre el objeto/la instancia **`John's Saving`** desde la *Estructura del modelo* hasta el área de trabajo (del diagrama la pestaña **`Sample Accounts`**) y colóquelo debajo de **`John's Credit`**.



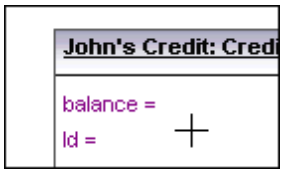
Para crear vínculos entre los objetos:

Los *vínculos* son las instancias de asociaciones de clases y describen las relaciones que existen entre los objetos/las instancias en un momento dado.

1. Haga clic en el vínculo (asociación) que existe entre **`AltovaBank`** y **`John's Checking`**.
2. En la pestaña *Propiedades* haga clic en el cuadro combinado `clasificador` y seleccione la entrada **Account - Bank.**
   El vínculo se convierte en una asociación compuesta, de acuerdo con las definiciones de clase.

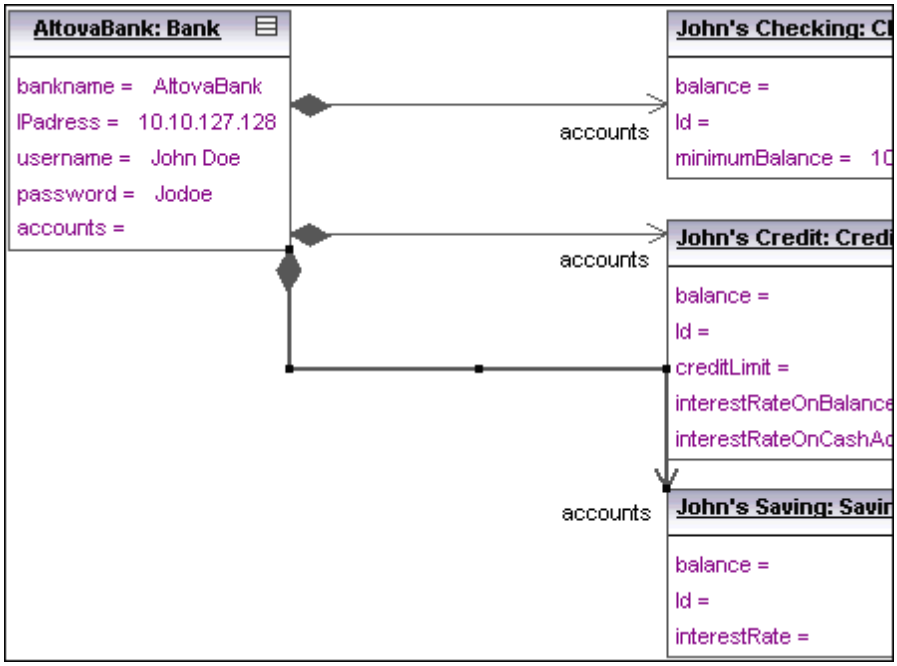


3. Haga clic en el icono **EspecificaciónDeInstancia** de la barra de herramientas y pase el cursor por encima de la clase **`John's Credit`**.
   El cursor ahora tiene forma de +.



4. Haga clic en el objeto **`John's Credit`** y arrastre el puntero hasta **`AltovaBank`** para crear un vínculo entre ellos.
5. En el cuadro combinado `clasificador` de la pestaña *Propiedades* cambie el tipo de vínculo a **Account - Bank**.
6. De la misma manera cree un vínculo entre **`John's Saving`** y **`AltovaBank`**.

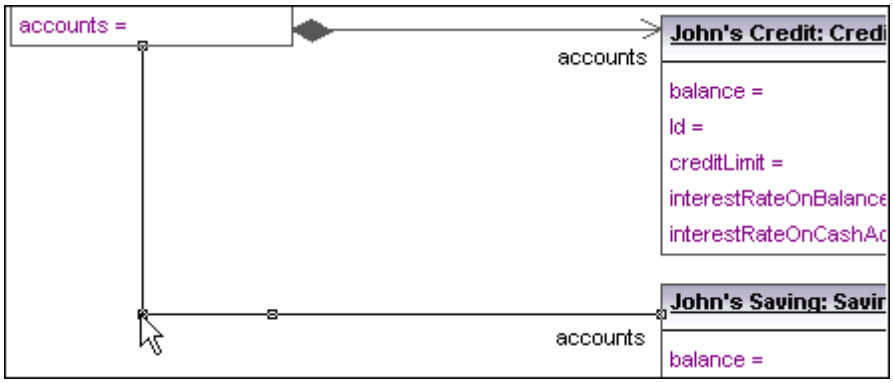
 © 2014 Altova GmbH



Nota: si cambia el tipo de asociación en un diagrama de clases, el tipo de asociación se actualiza automáticamente en el diagrama de objetos.

Para dar formato a las líneas de asociación/vínculo del diagrama:

1. Haga clic en el vínculo situado en la parte inferior del diagrama y arrastre el conector de la esquina a la izquierda.
   Esto sirve para ajustar la posición horizontal y vertical de la línea de asociación.

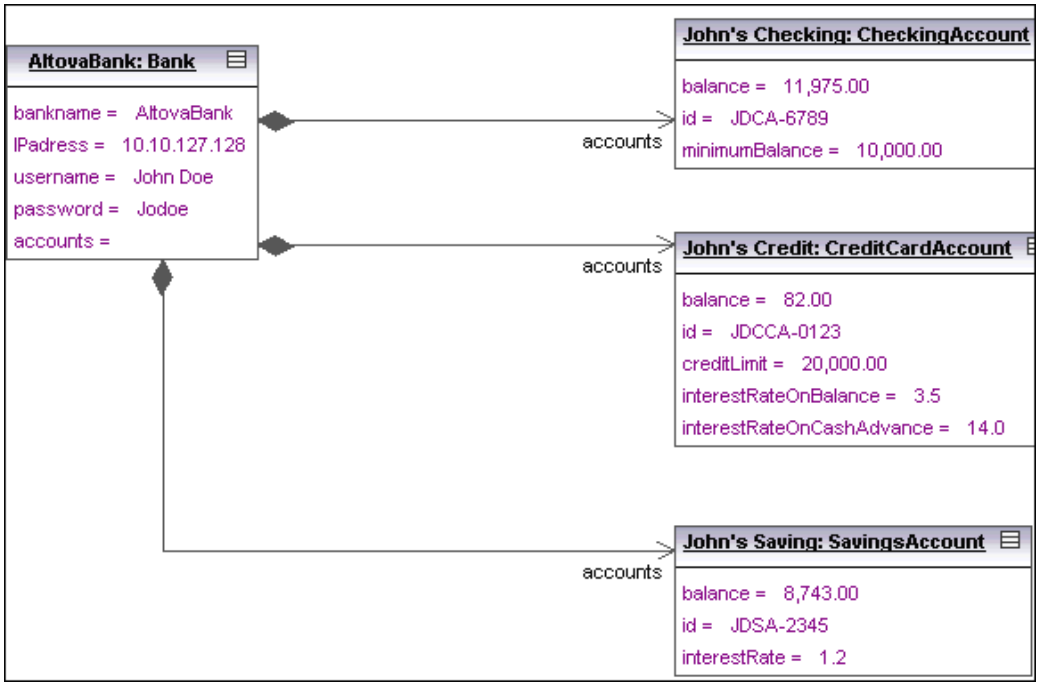


Ajuste la posición de los vínculos del diagrama siguiendo este método.

Para introducir datos de muestra en los objetos:

El valor de instancia de un atributo o de una propiedad de un objeto recibe le nombre de ***slot***.

1. Haga clic en los slots correspondientes e introduzca datos de muestra.
2. Por ejemplo, en el objeto **`John's Checking`** haga doble clic en el slot `balance` y escriba la cifra `11.975,00`.
3. Rellene el resto de los datos para tener una idea del estado actual de la instancia.



 © 2014 Altova GmbH

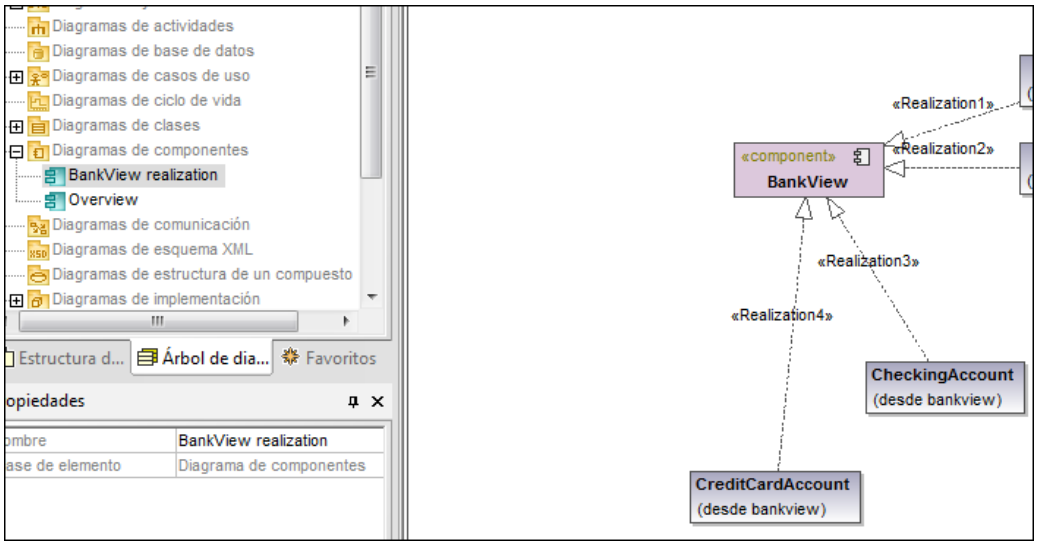
4.5 Diagramas de componentes

En este apartado del tutorial aprenderá a:

- Insertar clases en un diagrama de componentes
- Crear dependencias de realización entre las clases y el componente **`BankView`**
- Cambiar las propiedades de las líneas
- Insertar componentes en un diagrama de componentes y crear dependencias de utilización en una interfaz

Para abrir el diagrama de componentes:

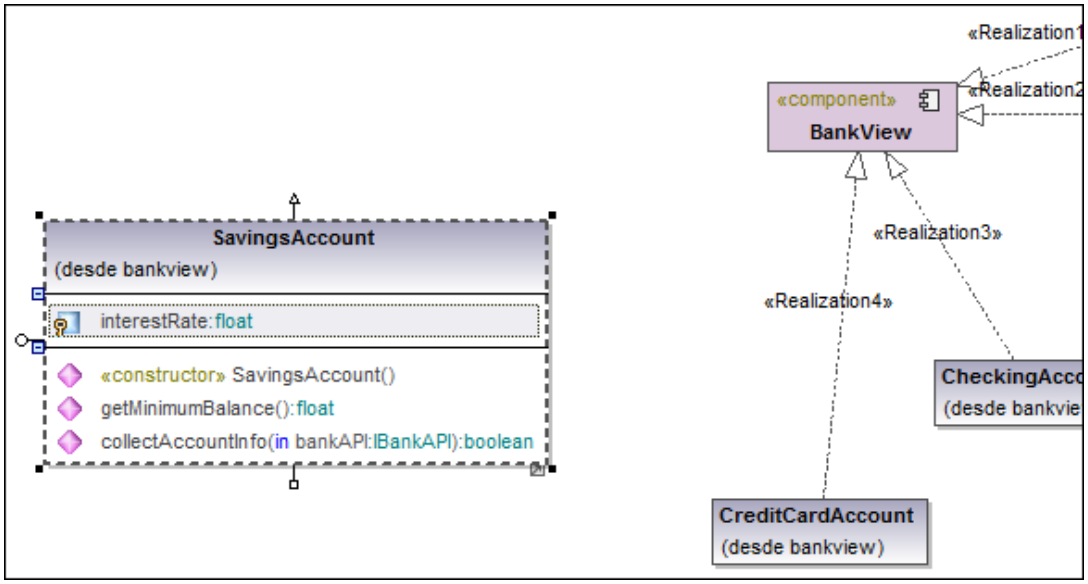
1. Haga clic en la pestaña *Árbol de diagramas*, expanda el nodo `Diagramas de componentes` y haga doble clic **en el icono** del diagrama **`BankView realization`**.
   En el área de trabajo se abre el diagrama de componentes **`BankView realization`**.



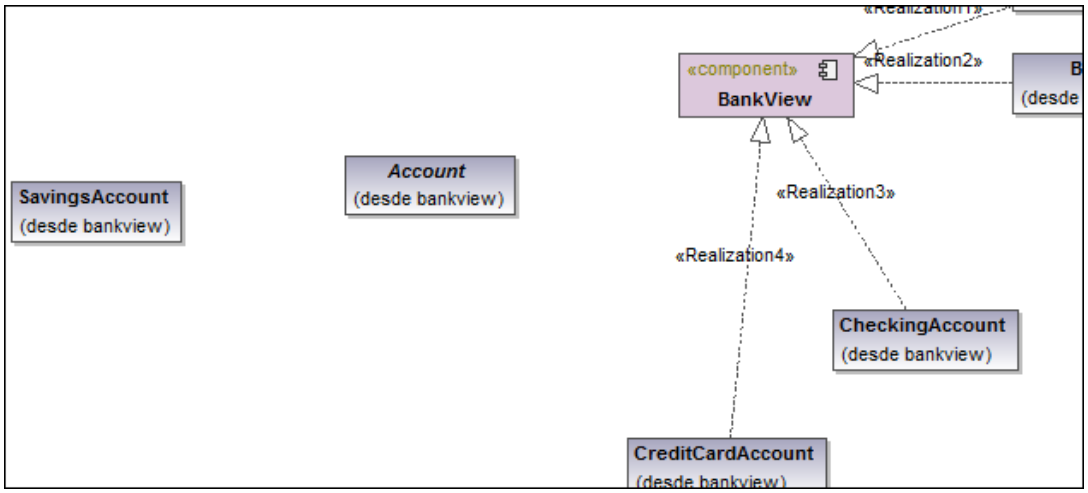
2. Vuelva a la pestaña *Estructura del modelo*.

Para insertar clases ya existentes en un diagrama de componentes:

1. En la *Estructura del modelo* busque la clase SavingsAccount (situada bajo el paquete **`bankview`**).
2. Arrastre la clase **`SavingsAccount`** hasta el diagrama de componentes.
   La clase aparece con todos sus compartimientos abiertos.



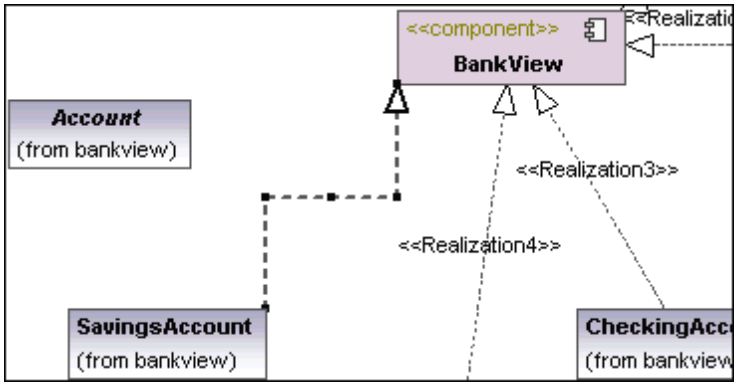
3. Haga clic en los iconos - para contraer los compartimientos de la clase (hasta que solamente se vea el nombre de la clase)
4. De la misma manera inserte la clase abstracta **`Account`**.



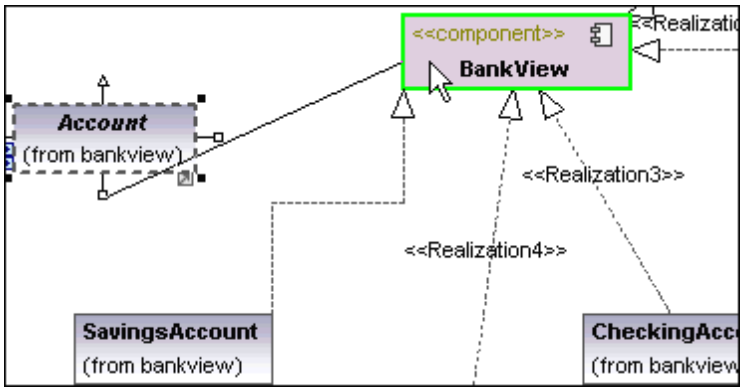
Nota: observe que el nombre del paquete que contiene las clases insertadas aparece en el compartimiento que incluye el nombre de la clase (p. ej. *desde bankview*).

Para crear dependencias de realización entre una clase y un componente:

1. Haga clic en el icono **Realización** de la barra de herramientas.
2. Haga clic en **`SavingsAccount`** y arrastre el puntero hasta el componente **`BankView`**.



3. Haga clic en el controlador **RealizaciónDeComponente** de la clase ***Account*** (en la parte inferior) y arrastre el puntero del mouse hasta el componente **BankView**.

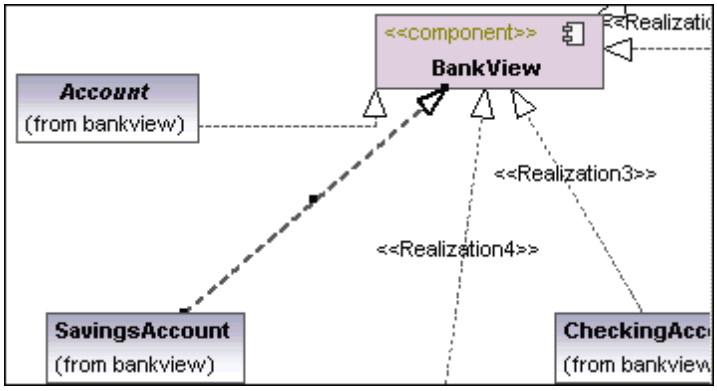


Estos dos métodos se pueden utilizar para crear dependencias de realización. Hay otro método más que permite crear dependencias de realización en la Estructura del modelo solamente. Para más información consulte el apartado Ingeniería de ida y vuelta (código - modelo - código).

Para cambiar las características de las líneas (de realización):

Al hacer clic en una dependencia o cualquier otro tipo de línea en un diagrama de UModel, se habilitan los comandos de la barra de herramientas Diseño que sirven para dibujar líneas.

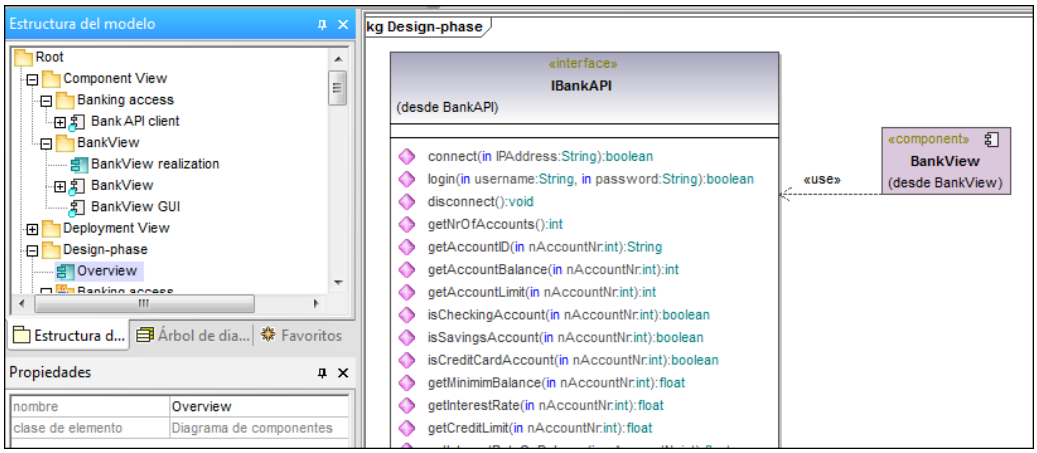
1. Haga clic en la línea de realización que une **SavingsAccount** con el componente **BankView**.
2. Haga clic en el icono **Línea directa** de la barra de herramientas Diseño.



Las propiedades de la línea se modifican inmediatamente. Las líneas tienen pequeños iconos denominados **puntos de referencia**. Puede hacer clic en estos puntos de referencia y moverlos para modificar las características de la línea.

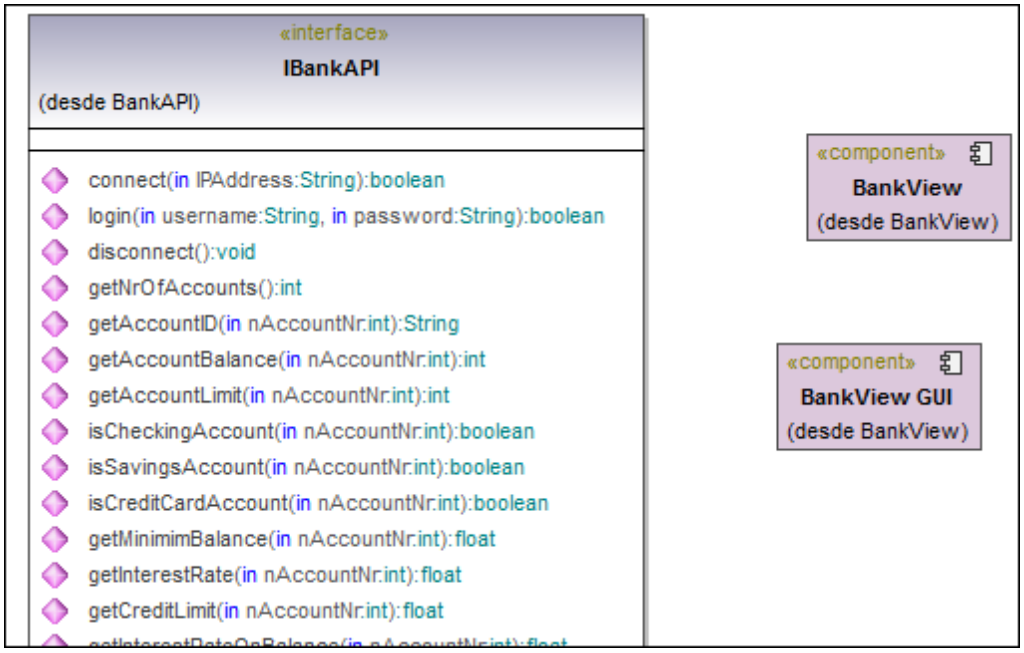
Para insertar componentes y crear dependencias de utilización:

1. En la *Estructura del modelo* haga doble clic **en el icono** del diagrama **`Overview`**, situado justo debajo del paquete **`Design-phase`**.
   El diagrama de componentes **`Overview`** se abre en el área de trabajo y muestra las dependencias de sistema definidas hasta ahora entre componentes e interfaces.



2. En la *Estructura del modelo* haga clic en el componente **`BankView GUI`** situado justo debajo del paquete **`Component View | BankView`** y arrástrelo hasta la pestaña del diagrama **`Overview`**.
   El paquete que contiene el componente insertado aparece en el compartimiento del nombre (es decir, desde `BankView`).
3. De la misma manera inserte el componente **`BankView`**.

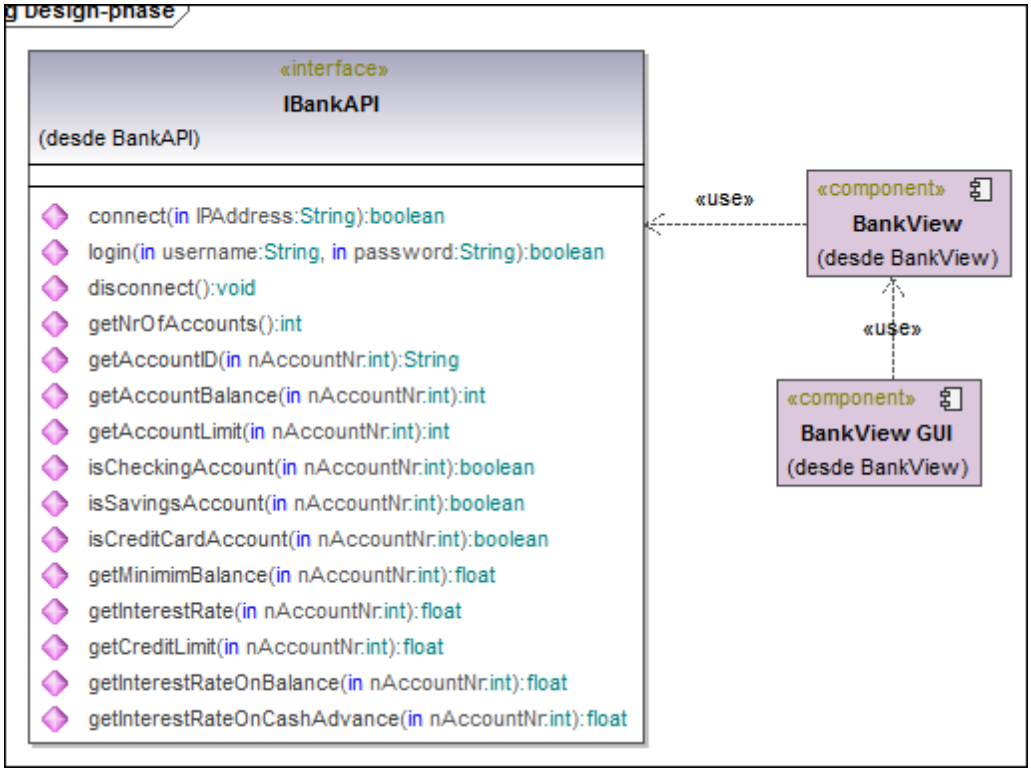
 © 2014 Altova GmbH



El componente **`BankView`** es el componente generado por el proceso de ingeniería directa descrito en este tutorial.

Para crear dependencias de utilización entre interfaces y componentes:

1. Haga clic en el icono **Utilización** de la barra de herramientas.
2. Ahora haga clic en el componente **`BankView GUI`** y arrastre el puntero hasta el componente **`BankView`**.
3. Haga clic otra vez en el icono **Utilización**. Ahora haga clic otra vez en el componente **`BankView`** y arrastre el puntero hasta la interfaz **`IBankAPI`**.



La dependencia de utilización (`<<use>>`) conecta un elemento **cliente** con un elemento **proveedor**. En este caso la interfaz **`IBankInterfaceAPI`** utiliza los servicios de los componentes **`BankView`** y **`BankView GUI`**.

 © 2014 Altova GmbH

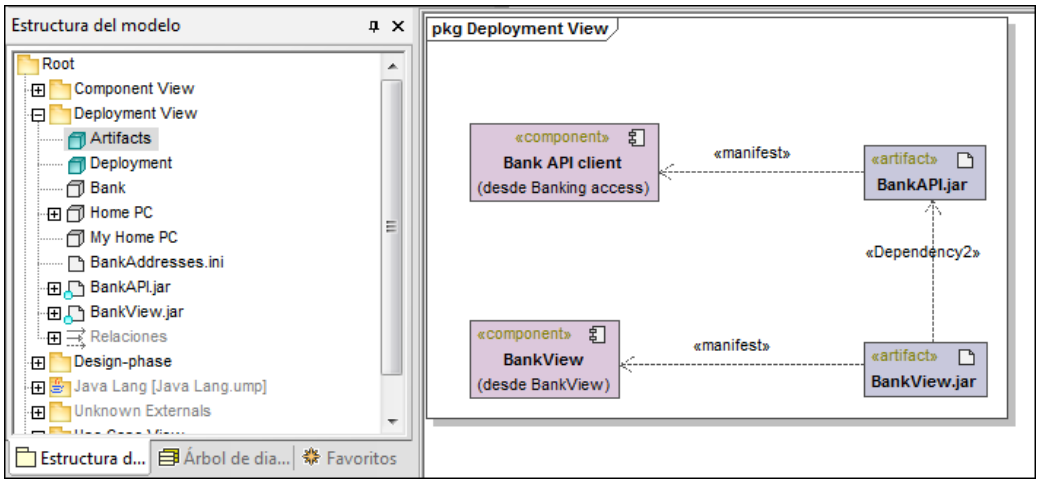
4.6 Diagramas de implementación

En este apartado del tutorial aprenderá a:

- Ver las manifestaciones artefacto de los componentes
- Agregar un nodo y una dependencia nuevos al diagrama de implementación
- Agregar artefactos a un nodo y crear relaciones entre ellos

Para abrir el diagrama de implementación Artifacts:

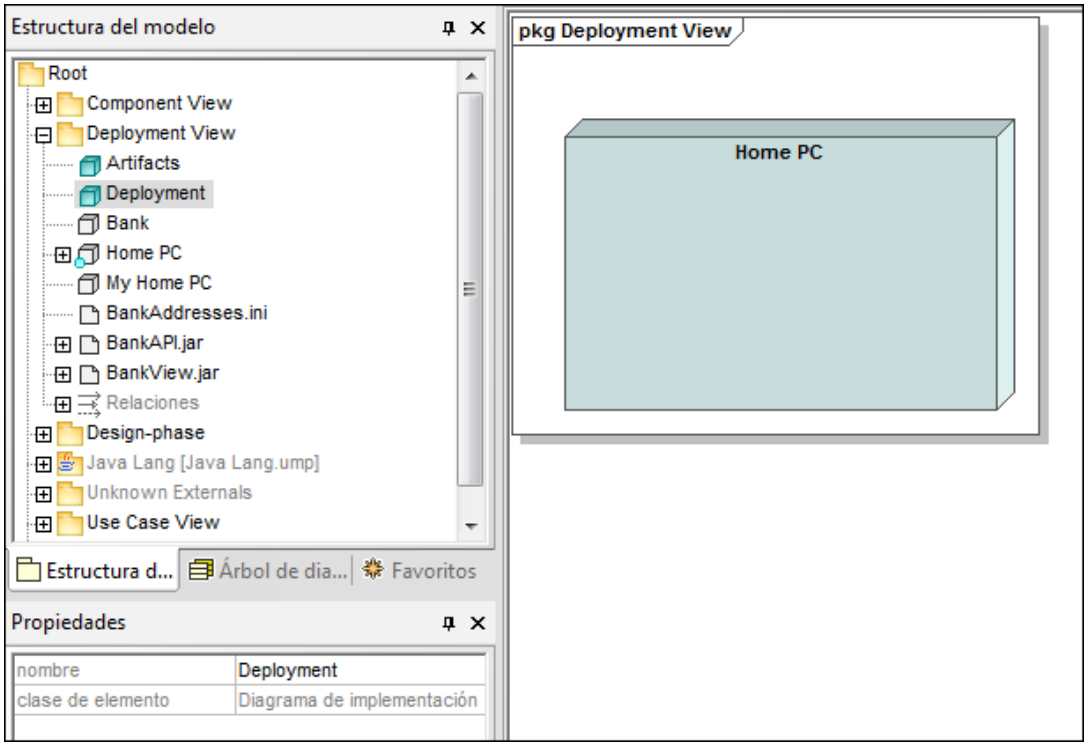
1. En la *Estructura del modelo* expanda el paquete de diagramas **`Deployment View`** y haga doble clic **en el icono** del diagrama **`Artifacts`**.



Este diagrama muestra la manifestación de los componentes **`Bank API client`** y **`BankView`** hacia sus correspondientes archivos Java .jar compilados.

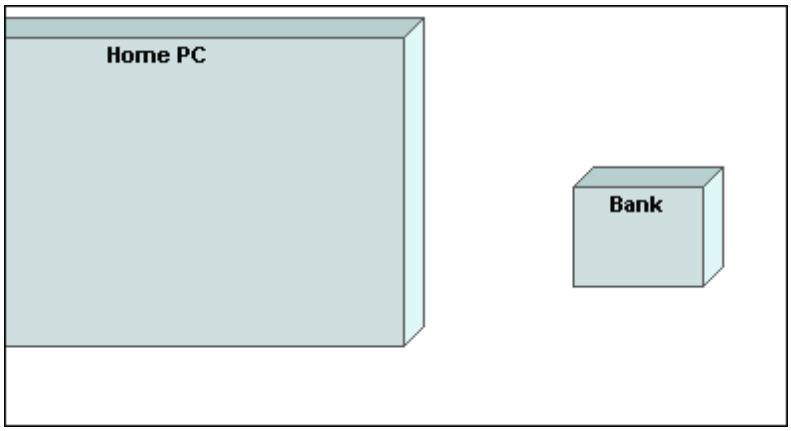
Para abrir el diagrama de implementación Deployment:

1. En la *Estructura del modelo* expanda el paquete de diagramas **`Deployment View`** y haga doble clic **en el icono** del diagrama **`Deployment`**.
   El diagrama se abre en el área de trabajo y muestra la arquitectura física del sistema, que actualmente solo está formado por el nodo **`Home PC`**.



Para agregar un nodo al diagrama de implementación:

1. Haga clic en el icono **Nodo** de la barra de herramientas y después haga clic en el nodo **`Home PC`**.
2. Cambie el nombre predeterminado del nodo por **`Bank`** y arrastre sus bordes para aumentar un poco su tamaño.

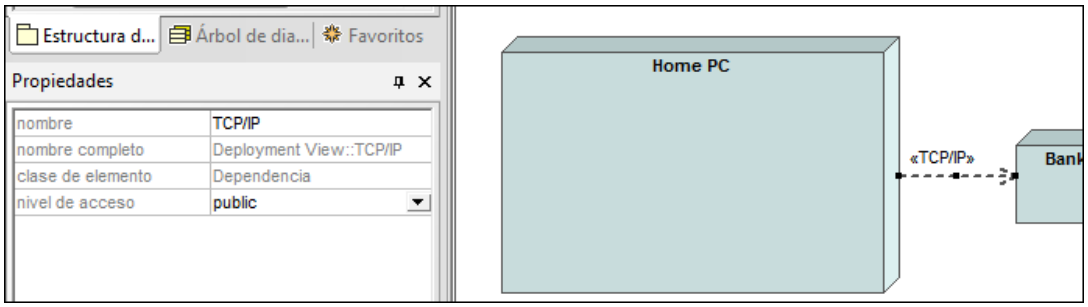


Para crear una dependencia entre dos nodos:

1. Haga clic en el icono **Dependencia** . Haga clic en el nodo **`Home PC`** y arrastre el puntero hasta el nodo **`Bank`**.
   Esto crea una dependencia entre los dos nodos.
2. En la pestaña *Propiedades* haga clic en el campo `nombre` y escriba *TCP/IP*. Pulse

 © 2014 Altova GmbH

Entrar para confirmar.
El nombre de la dependencia aparece encima de la línea de dependencia.

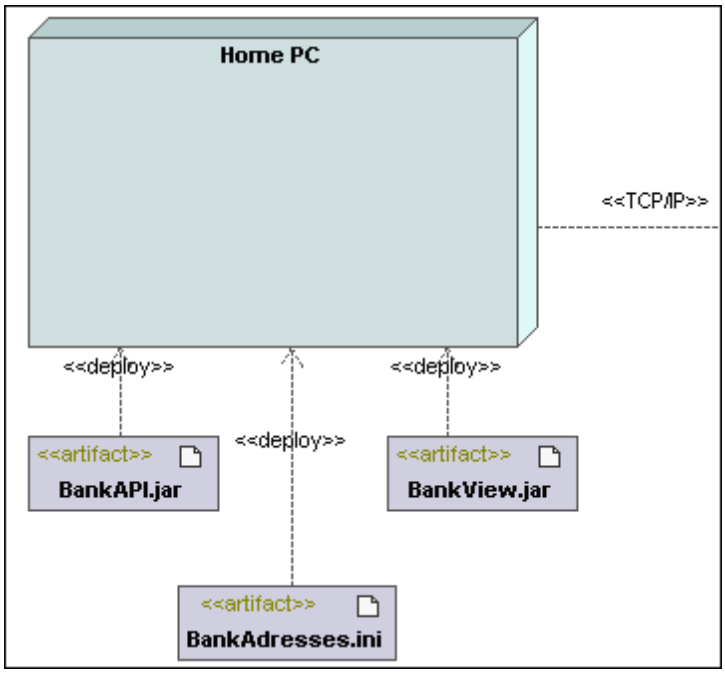


Nota: tras hacer clic en la flecha de la dependencia (o en cualquier elemento con nombre) puede teclear directamente texto nuevo, sin necesidad de hacer clic primero en el campo `nombre` de la pestaña *Propiedades*.

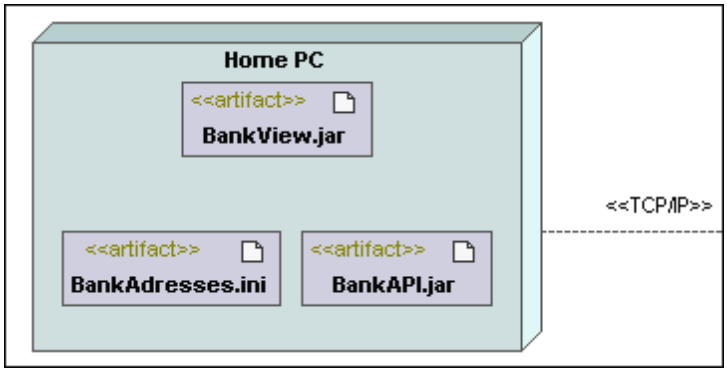
Para agregar artefactos a un nodo y crear dependencias entre ellos:

Expanda el paquete **`Deployment View`** de la *Estructura del modelo*:

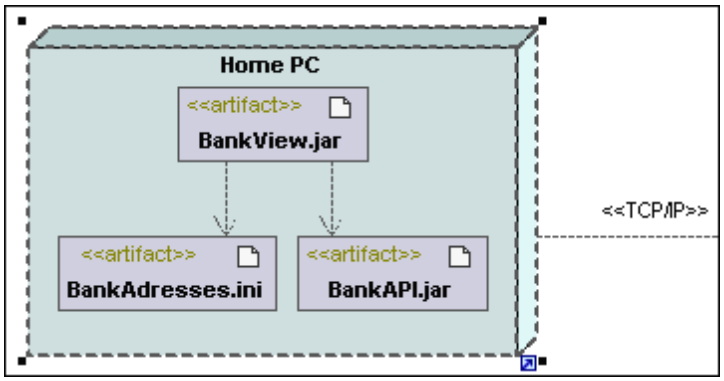
1. Ahora haga clic en los artefactos **`BankAddresses.ini`**, **`BankAPI.jar`** y **`BankView.jar`** y arrástrelos uno a uno hasta el área de trabajo del diagrama. Aparecen las dependencias de cada artefacto.



2. Haga clic en el artefacto **`BankView.jar`** y arrástrelo hasta el nodo **`Home PC`**.
   Cuando esté permitido soltar el artefacto, el nodo de destino aparece resaltado.
3. Haga lo mismo para arrastrar los demás artefactos hasta el nodo **`Home PC`**.
   Ahora los artefactos forman parte del nodo y se mueven con él si lo cambiamos de posición.



4. Haga clic en el icono **Dependencia** de la barra de herramientas y pulse la tecla **Ctrl**.
5. Sin dejar de pulsar la tecla **Ctrl**, haga clic en el artefacto `BankView.jar` y arrastre el puntero hasta el artefacto `BankAddresses.ini`.
6. Sin dejar de pulsar la tecla **Ctrl**, haga clic en el artefacto `BankView.jar` y arrastre el puntero hasta el artefacto `BankAPI.jar`.



Nota: si arrastra un artefacto fuera del nodo, se crea una dependencia de implementación automáticamente.

Para eliminar un artefacto de un nodo y del proyecto:

- Haga clic en el artefacto que desea eliminar y pulse la tecla **Supr**.

  El artefacto y las dependencias se eliminan del nodo y del **proyecto**.

Para eliminar un artefacto de un nodo y de su diagrama:

1. Arrastre el artefacto fuera del nodo.
2. Pulse **Ctrl+Supr**.

   El artefacto y sus dependencias se eliminan del **diagrama** actual, pero no del proyecto.

Para crear operaciones / propiedades o anidar artefactos:

1. Haga clic **con el botón secundario** en el artefacto en la *Estructura del modelo*.
2. Seleccione la acción correspondiente en el menú contextual (p. ej. **Elemento nuevo** | **Operación** / **Propiedad** / **Artefacto**)
   El elemento nuevo aparece bajo el artefacto seleccionado en la *Estructura del modelo*.

 © 2014 Altova GmbH

4.7 Ingeniería de ida y vuelta (modelo - código - modelo)

Sitio web de Altova: Generación de código UML e ingeniería de código UML de ida y vuelta

En este apartado del tutorial aprenderá a:

- Revisar la sintaxis del proyecto
- Generar código de proyecto
- Agregar código externo de método nuevo (a la clase **SavingsAccount**)
- Sincronizar el código nuevo con el modelo

Sincronizar paquetes con el código / modelo:
El código se puede combinar/sincronizar a varios niveles:

- A nivel de proyecto / paquete raíz (con ayuda de un comando de menú)
- A nivel de paquete (es posible seleccionar varios paquetes y generar código a partir de ellos)
- A nivel de clase (es posible seleccionar varias clases y generar código a partir de ellas)
- A nivel de componente

El diagrama **BankView** muestra cómo se realiza el componente **BankView** por medio de sus seis clases constituyentes. Este es el componente que se obtiene al seguir las instrucciones para ingeniería directa del tutorial.

Requisitos para poder generar código:

- El componente debe estar **realizado** por una clase como mínimo.
- El componente debe tener asignada una ubicación física (es decir, un directorio). El código generado se guarda en ese directorio.
- Para poder incluirlos en el proceso de generación de código, los componentes deben establecerse por separado.
- Debe definirse un paquete raíz de espacio de nombres Java, C# o VB.

Nota: la raíz del espacio de nombres Java se definió en el paquete **Design-phase | BankView | com**, en la *Estructura del modelo*.

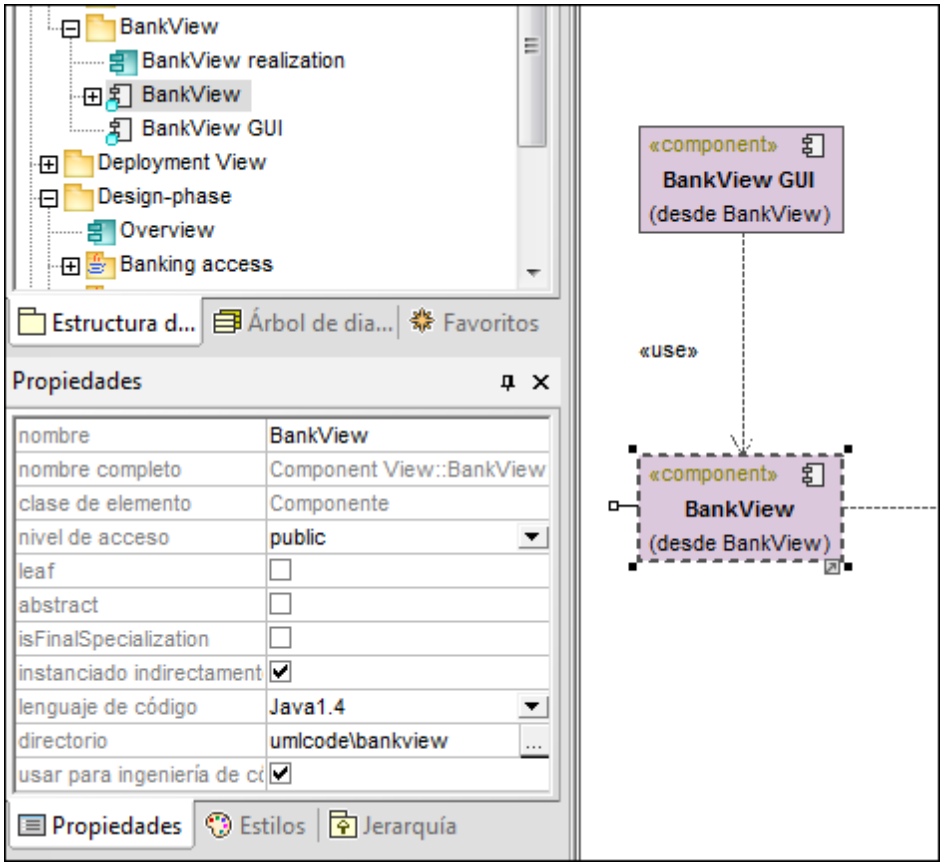
En un mismo proyecto puede combinar código Java, C# y VB y UModel se ocupa del código automáticamente durante el proceso de ingeniería de ida y vuelta. Por ejemplo, el archivo de proyecto `Bank_MultiLanguage.ump` de la carpeta de ejemplos `...\UModelExamples` incluye código Java y C#.

Para definir el directorio de destino para la generación de código:

1. En la *Estructura del modelo* haga doble clic en el icono de **Overview**, justo debajo del paquete **Design-phase**.
2. Ahora en el diagrama haga clic en el componente **BankView** y observe su configuración en la pestaña *Propiedades*.
3. Haga clic en el botón ... situado a la derecha del campo `directorio`.
4. Introduzca/seleccione el directorio de destino en el cuadro de diálogo (p. ej. `DirectorioInstalación\UModelExamples\Tutorial\umlcode\bankview`) o haga clic en el botón **Carpeta nueva** para crear una carpeta nueva.

 © 2014 Altova GmbH

La ruta de acceso aparece ahora en el campo `directorio`.

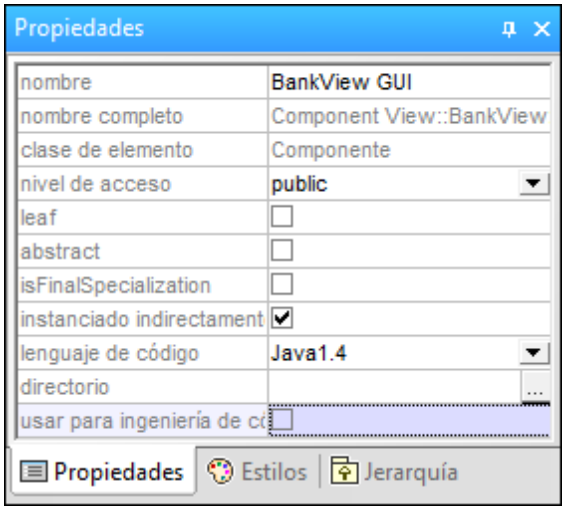


Nota: en UModel el código Java crea código en un directorio u otro dependiendo de su espacio de nombres (p. ej. `...\code\namespace1\C1.java`).

Si quiere usar la misma norma para nombrar directorios para C# y VB .NET, seleccione la opción de menú **Herramientas | Opciones | Ingeniería de código** y elija las opciones correspondientes en el grupo de opciones *Usar espacio de nombres para la ruta del archivo de código*.

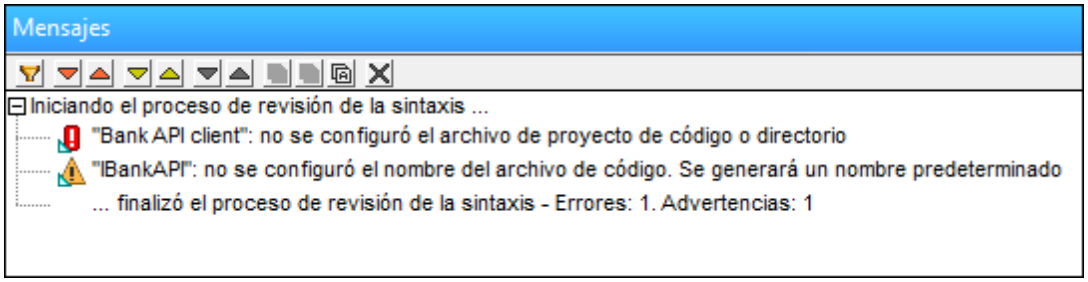
Para incluir/excluir componentes en la generación de código:

1. Haga clic en el componente **`BankView GUI`**.
2. Si está marcada, desactive la casilla `usar para ingeniería de código` de la pestaña *Propiedades*.



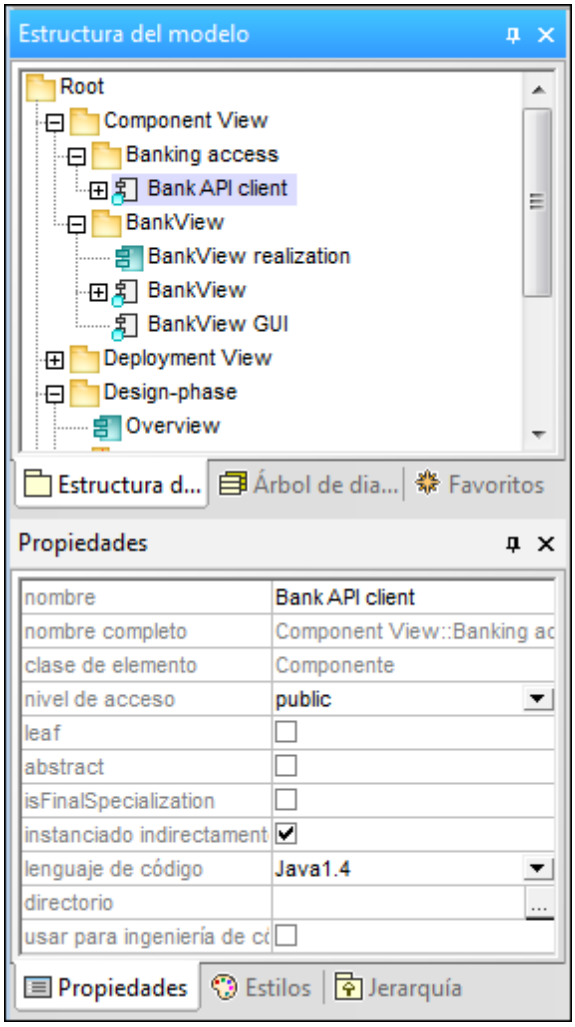
Revisar la sintaxis del proyecto antes de generar el código:

1. Seleccione la opción de menú **Proyecto | Revisar la sintaxis del proyecto**.
2. Cuando termina la revisión de la sintaxis aparece un mensaje en la ventana Mensajes: `"Bank API-client: no se configuró el archivo de proyecto de código o directorio" - "IBankAPI: no se configuró el nombre del archivo de código".`

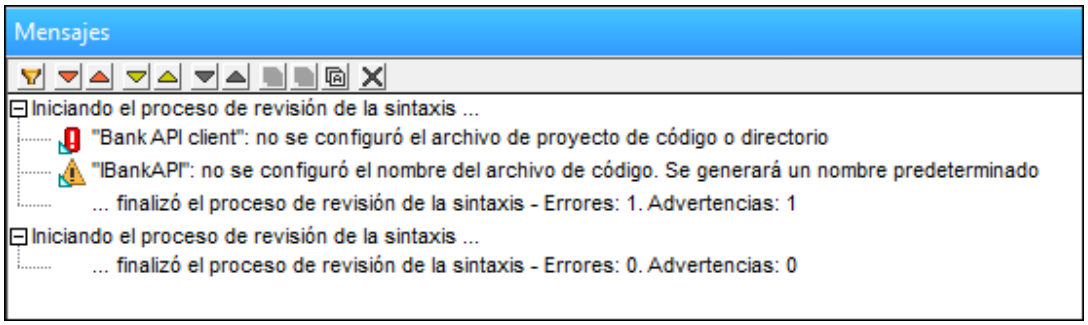


3. Haga clic en el primer mensaje de la ventana.
4. El paquete **`Bank API client`** aparece resaltado en la vista *Estructura del modelo* y sus propiedades aparecen en la pestaña *Propiedades*.
5. Desactive la casilla `usar para ingeniería de código` de la pestaña *Propiedades*.

 © 2014 Altova GmbH



6. Revise otra vez la sintaxis del proyecto con el comando **Proyecto | Revisar la sintaxis del proyecto**.

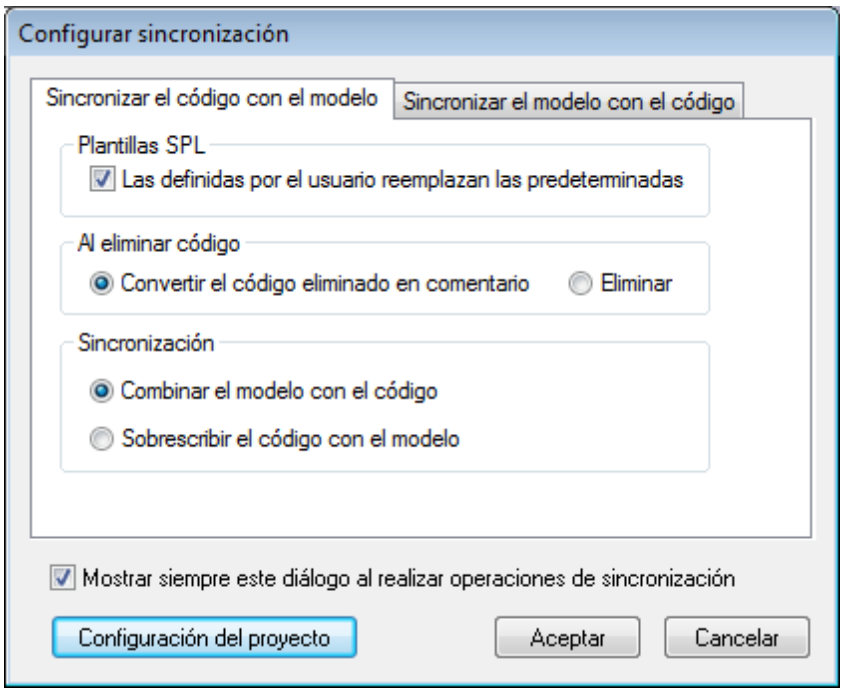


La revisión ya no encuentra más errores. Ahora podemos generar código de programa para el proyecto. Para más información consulte el apartado Revisar la sintaxis del proyecto.

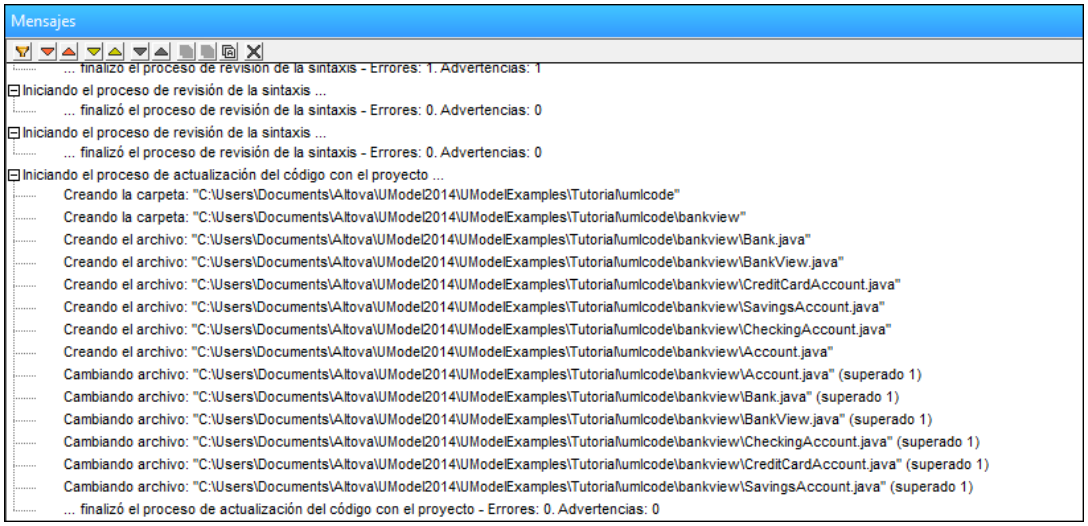
Para generar código de proyecto:

1. Haga clic en el paquete `BankView` para seleccionarlo.

2. Seleccione la opción de menú **Proyecto | Combinar el código de programa con el proyecto de UModel**.
3. Seleccione las opciones de sincronización en el cuadro de diálogo y pulse **Aceptar** para continuar (para este ejemplo del tutorial no hace falta cambiar ninguna opción. Consulte el apartado Combinar el código de programa con el proyecto de UModel para más información).



El panel de mensajes muestra el resultado del proceso de generación de código.



4. Navegue hasta el directorio de destino.
   Se crearon seis archivos **.Java** para el proyecto.

Sincronizar el modelo de UModel después de actualizar el código Java en una

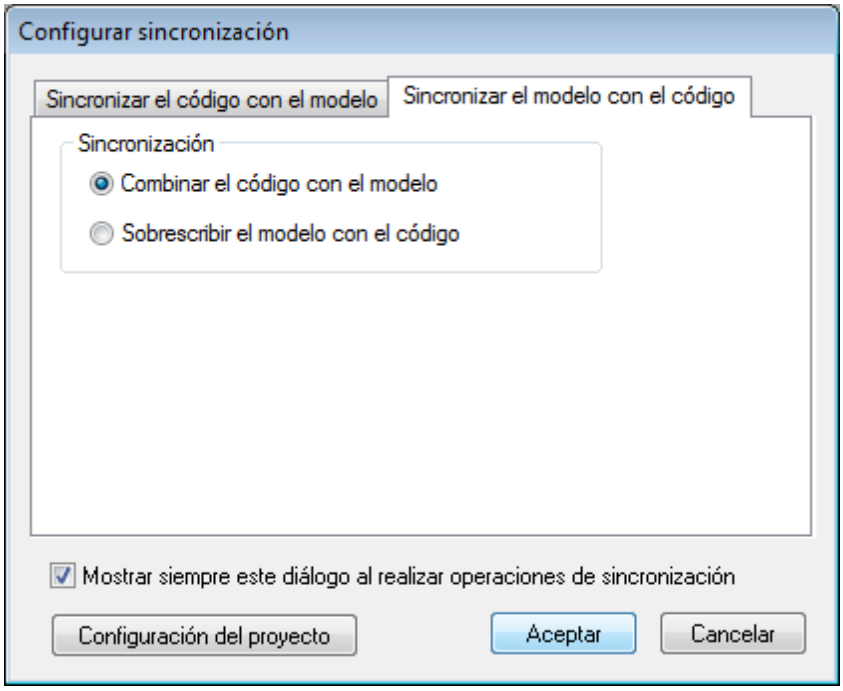
aplicación externa:

1. Abra el archivo `SavingsAccount.java` en un editor de texto (en XMLSpy por ejemplo).
2. Añada el método **`public float getInterestRate() {}`** al código y guarde el archivo.

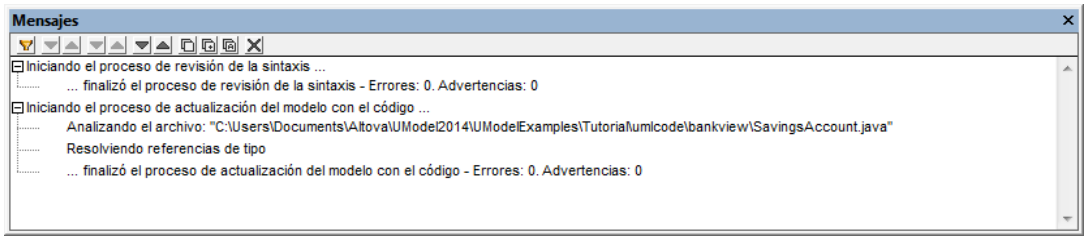
```
1
2   public class SavingsAccount extends Account
3   {
4
5       protected float interestRate;
6
7       public SavingsAccount()
8       {
9       }
10
11      public float getMinimumBalance()
12      {
13      }
14
15      public float getInterestRate()
16      {
17      }
18
19      public boolean collectAccountInfo(IBankAPI bankAPI)
20      {
21      }
22  }
23
```

3. Vuelva a UModel y, en la *Estructura del modelo*, haga clic con el botón secundario en la clase **`SavingsAccount`** situada debajo del paquete **`BankView`**.
4. Seleccione el comando **Ingeniería de código | Combinar la clase de UModel con el código de programa**.

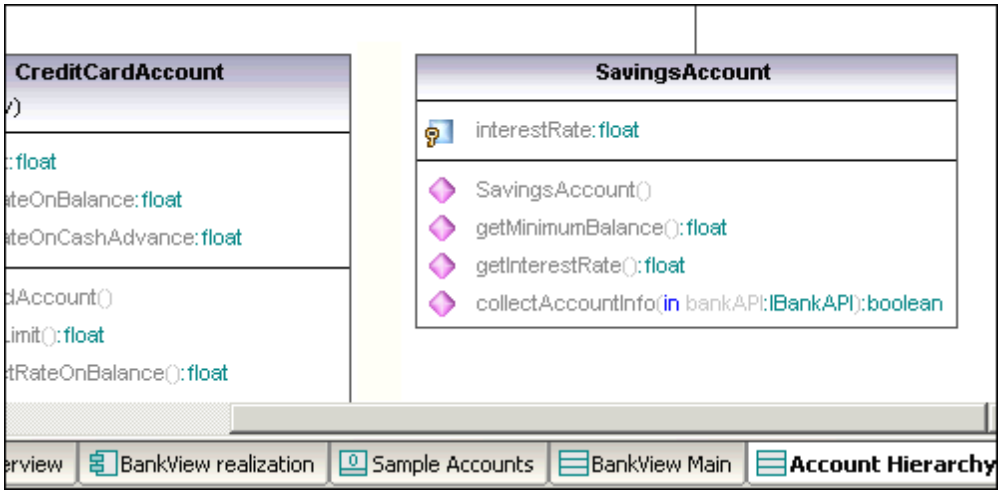
   Esto abre el cuadro de diálogo "Configurar sincronización" por la pestaña *Sincronizar el modelo con el código* (para este ejemplo del tutorial no hace falta cambiar ninguna opción. Consulte el apartado Combinar el proyecto de UModel con el código para más información).



5. Haga clic en **Aceptar** para combinar el modelo con el código.



6. Ahora haga clic en la pestaña del diagrama **Account Hierarchy** para ver el resultado del proceso de combinación.



El método nuevo se añadió al código (`getInterestRate`) y generó una **operación** nueva en la **clase** `SavingsAccount`.

 © 2014 Altova GmbH

4.8 Ingeniería de ida y vuelta (código - modelo - código)

Sitio web de Altova: Ingeniería inversa de modelos UML a partir de código Java, C# y Visual Basic

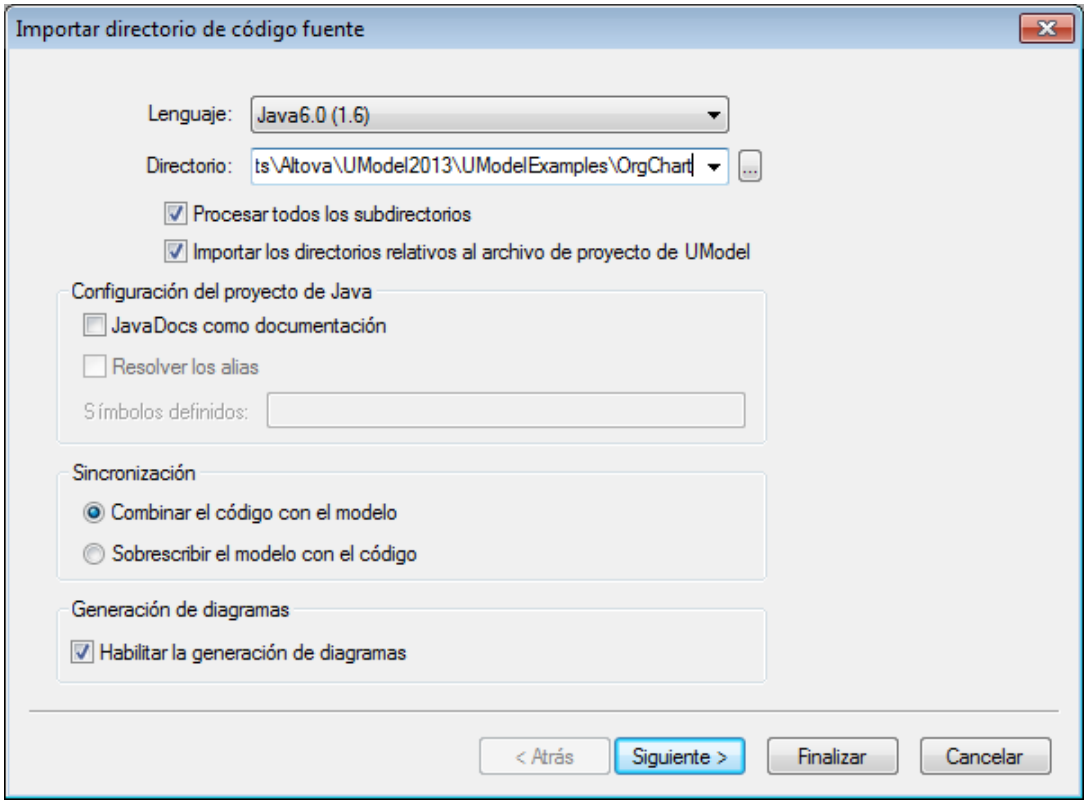
En este apartado del tutorial aprenderá a:

- Importar un directorio que contiene código Java generado con XMLSpy
- Agregar una clase nueva al proyecto en UModel
- Combinar un paquete de UModel con el código de programa

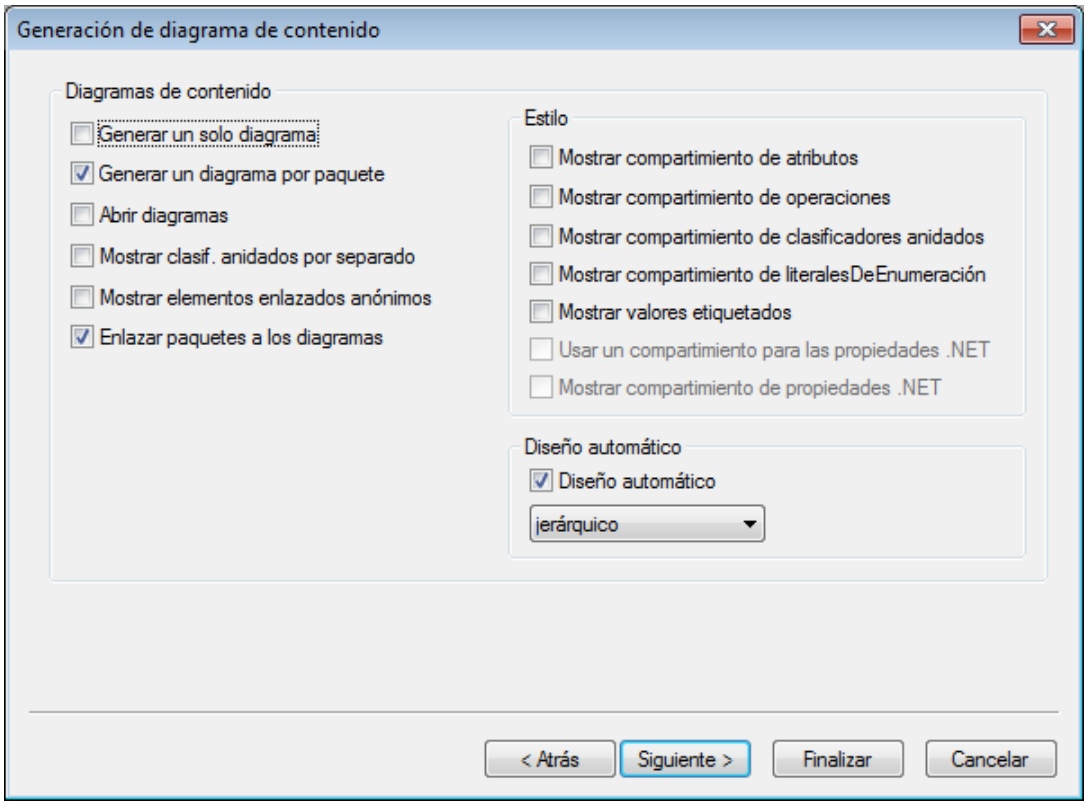
Los archivos utilizados para este ejemplo están disponibles en el archivo comprimido `OrgChart.zip` de la carpeta `...\UModelExamples`. Descomprima el archivo antes de empezar con este apartado del tutorial. El archivo se descomprime en el directorio `OrgChart`, que después se usará para importar el código ya existente.

Para aplicar ingeniería inversa (importar) a código ya existente en un directorio:

1. Seleccione la opción **Archivo | Nuevo** para crear un proyecto nuevo.
2. Ahora seleccione la opción **Proyecto | Importar directorio de código fuente**.
3. Seleccione el tipo de lenguaje del código que desea importar.
4. Ahora haga clic en el botón **Examinar** y seleccione el directorio `OrgChart` de la carpeta `...\UModelExamples`.



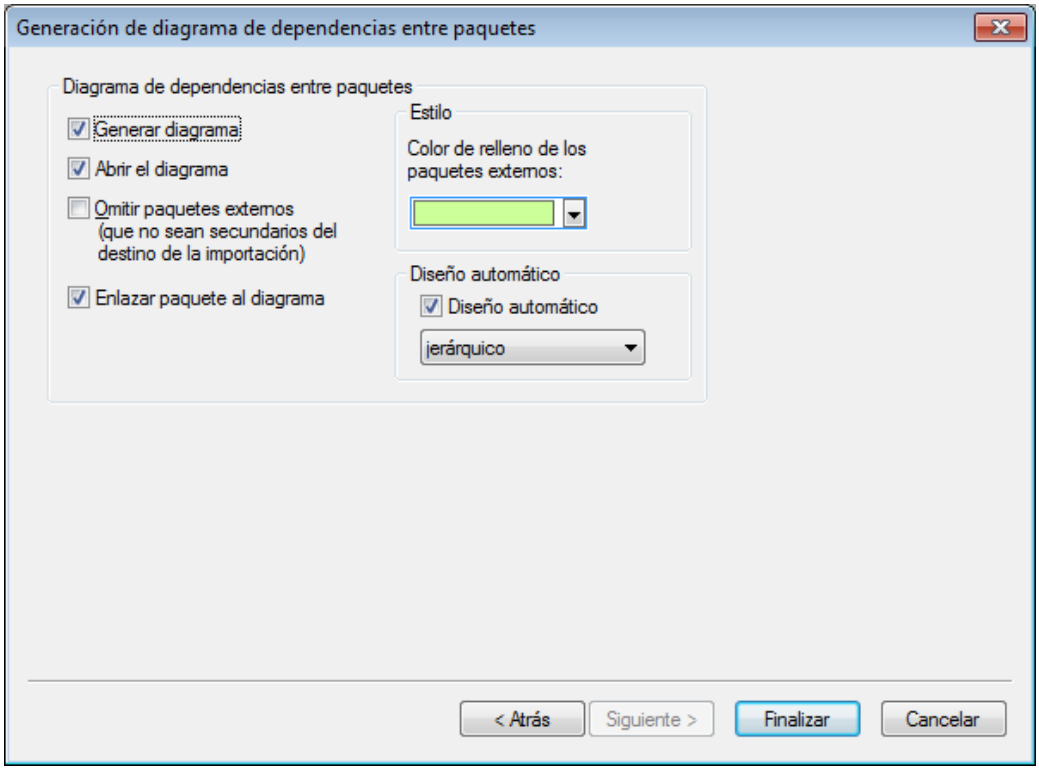
5. Compruebe que está activa la casilla *Habilitar la generación de diagramas*, seleccione las opciones de importación necesarias y haga clic en **Siguiente**.



Recuerde que UModel puede generar un solo diagrama global o un diagrama por cada paquete. En la imagen anterior puede ver las opciones de generación predeterminadas.

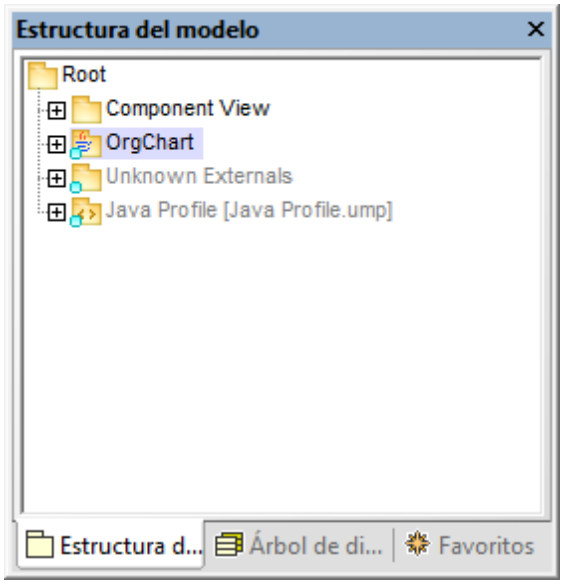
6. Haga clic en **Siguiente**.

 © 2014 Altova GmbH

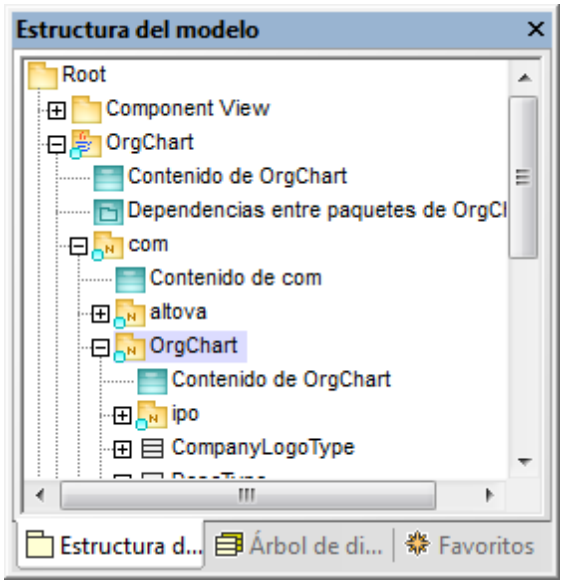


Esta pantalla del cuadro de diálogo sirve para definir las opciones de generación de dependencias entre paquetes.

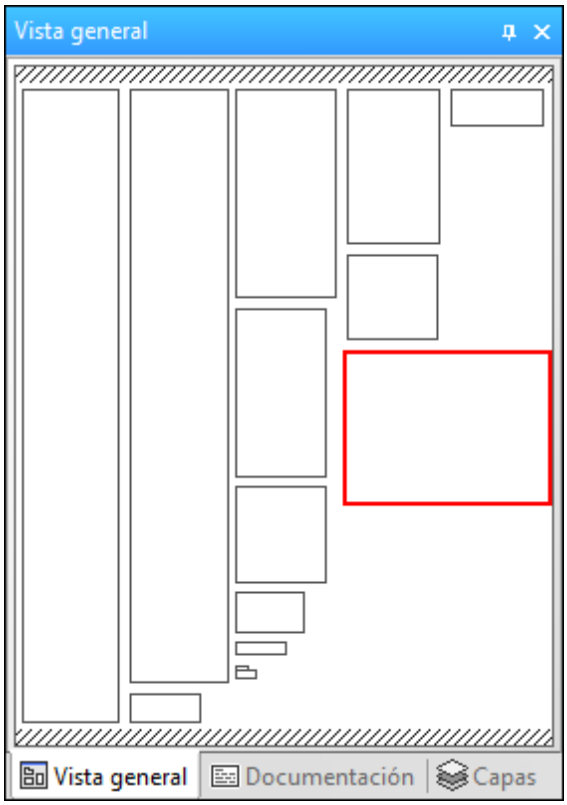
7. Para terminar haga clic en **Finalizar**.
   Los datos se analizan y se crea un paquete nuevo llamado `OrgChart`.



8. Expanda el paquete y sus subpaquetes hasta llegar al paquete `OrgChart` (`com` | `OrgChart`).

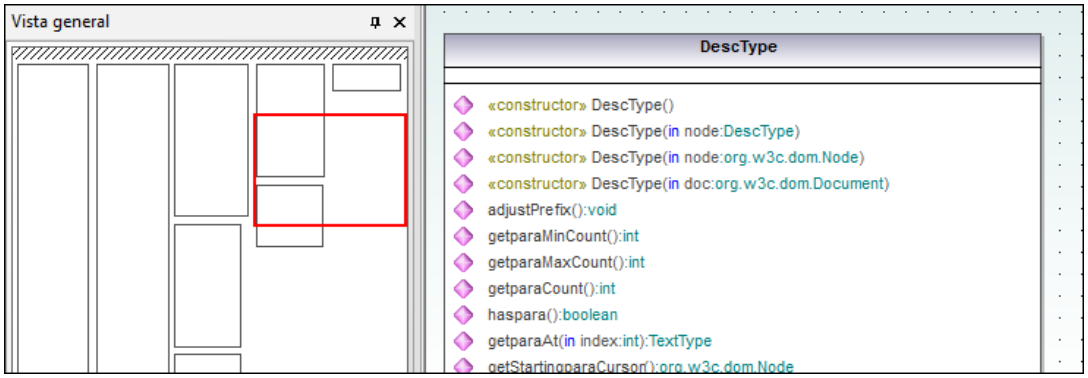


9. Haga doble clic en el icono del diagrama **`Contenido de OrgChart`**.
   En la pestaña principal aparecen las clases contraídas que componen **`OrgChart`**.
   La vista actual aparece en un recuadro rojo en la pestaña *Vista general*, que en este momento muestra una zona vacía del diagrama.



10. Haga clic en el recuadro rojo y arrástrelo hasta el centro de la ventana.

 © 2014 Altova GmbH



Nota: también puede seleccionar la opción **Proyecto | Importar proyecto de código fuente** y seleccionar el archivo de proyecto Borland JBuilder `OrgChart.jpx` para importar el proyecto creado con XMLSpy.

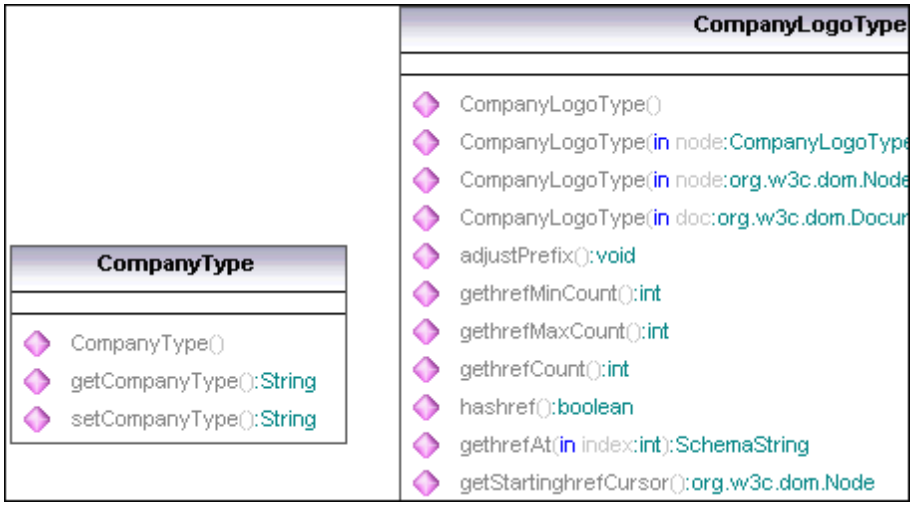
Ingeniería de ida y vuelta y relaciones entre los elementos de modelado:

Cuando actualice el modelo con código, las asociaciones existentes entre los elementos de modelado aparecen automáticamente si está activada la opción **Edición | Crear asociaciones automáticamente en** el cuadro de diálogo **Herramientas | Opciones**. UModel las asociaciones de todos los elementos que tengan definido el tipo de atributo y de los que tengan el elemento de modelado `type` en el mismo diagrama.

En el diagrama aparecen automáticamente las RealizacionesDeInterfaz y las Generalizaciones cuando el modelo se actualiza con el código.

Para agregar una clase nueva al diagrama OrgChart:

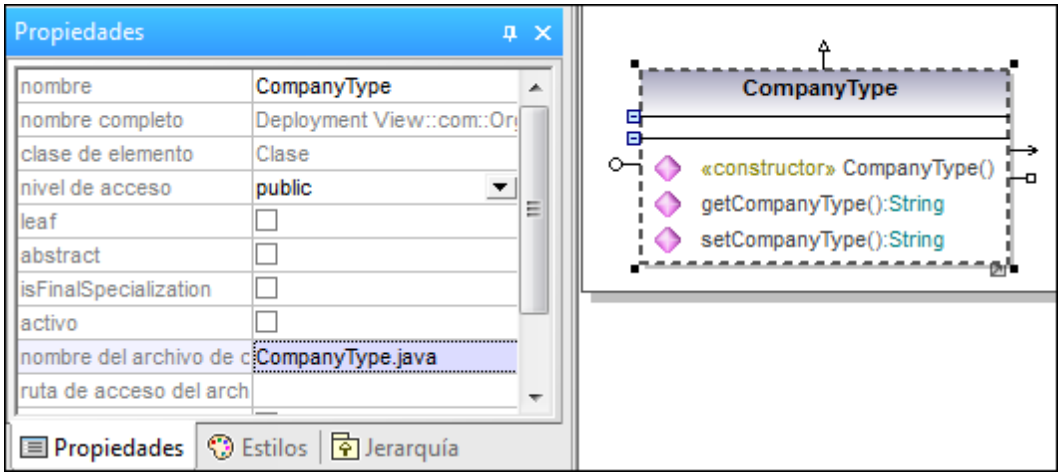
1. Haga clic en el icono **Clase** de la barra de herramientas .
2. Haga clic en el área de trabajo y cree una clase nueva llamada **`CompanyType`**.
3. Ahora, con la tecla **F8**, añada estas operaciones a la nueva clase: **`CompanyType()`**, **`getCompanyType():String`**, **`setCompanyType():String`**.



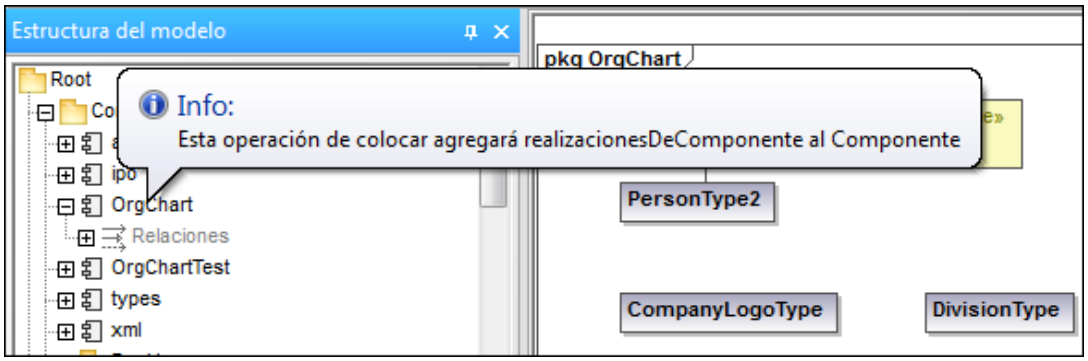
Habilitar la clase nueva para la generación de código:

Compruebe que la clase activa es **`CompanyType`**.

1. En la pestaña *Propiedades* haga clic en el campo `nombre del archivo de código` y escriba el nombre del archivo Java de la nueva clase `CompanyType.java`.



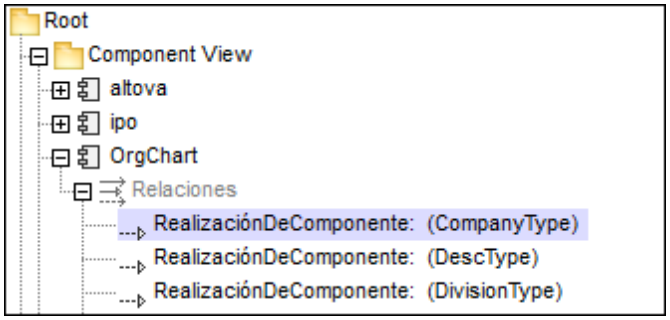
2. En la *Estructura del modelo* haga clic en la nueva clase **`CompanyType`**, arrástrela hacia arriba y suéltela encima del componente **`OrgChart`** (situado justo debajo del paquete **`Component View`**). Al pasar el puntero del mouse por el componente aparece información rápida.



Nota: este método crea una Realización entre una clase y un componente, sin necesidad de usar diagramas de componentes ni de implementación.

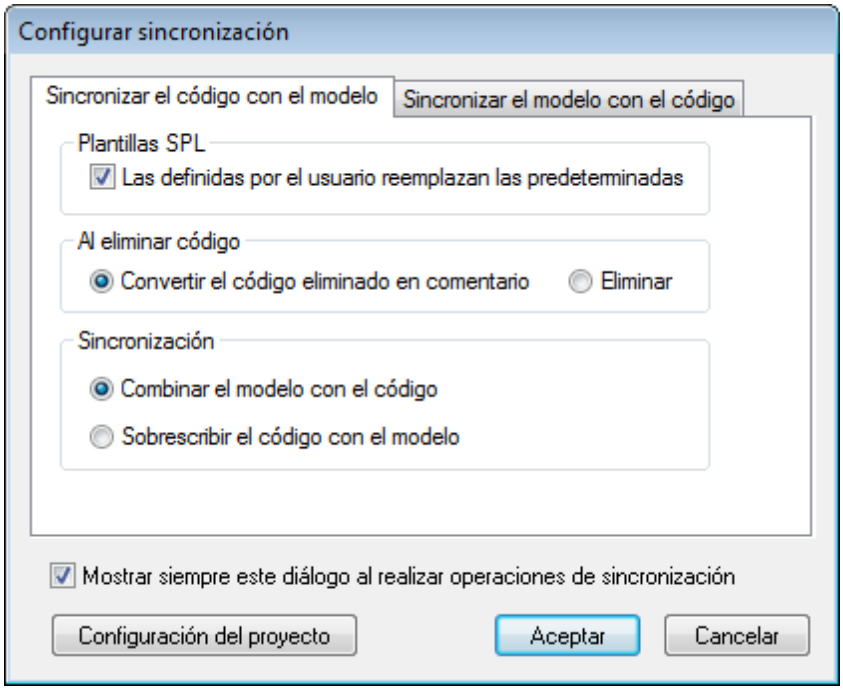
3. Expanda el nodo `Relaciones` situado debajo del componente **`OrgChart`** para ver la nueva Realización que acabamos de crear.

 © 2014 Altova GmbH



Para combinar el código de programa con un paquete:

1. Haga clic con el botón secundario en el paquete `OrgChart`, seleccione el comando **Ingeniería de código | Combinar el código de programa con el paquete de UModel** y pulse **Entrar**.



La ventaja de mensajes muestra el resultado de la revisión sintáctica y el estado del proceso de sincronización.

Una vez finalizado el proceso, la nueva clase `CompanyType.java` se añade a la carpeta `...\OrgChart\com\OrgChart\`.

Nota: el cuerpo de los métodos y los cambios realizados en el código se convertirán en comentarios o se eliminarán, dependiendo de la opción elegida en el grupo *Al eliminar código* del cuadro de diálogo "Configurar sincronización".

¡Ya está!
Ahora ya sabe crear un proyecto de modelado usando el proceso de ingeniería directa y completar un ciclo de ingeniería de ida y vuelta con UModel. El resto de la documentación describe cómo obtener los mejores resultados para su proyecto de modelado con UModel.

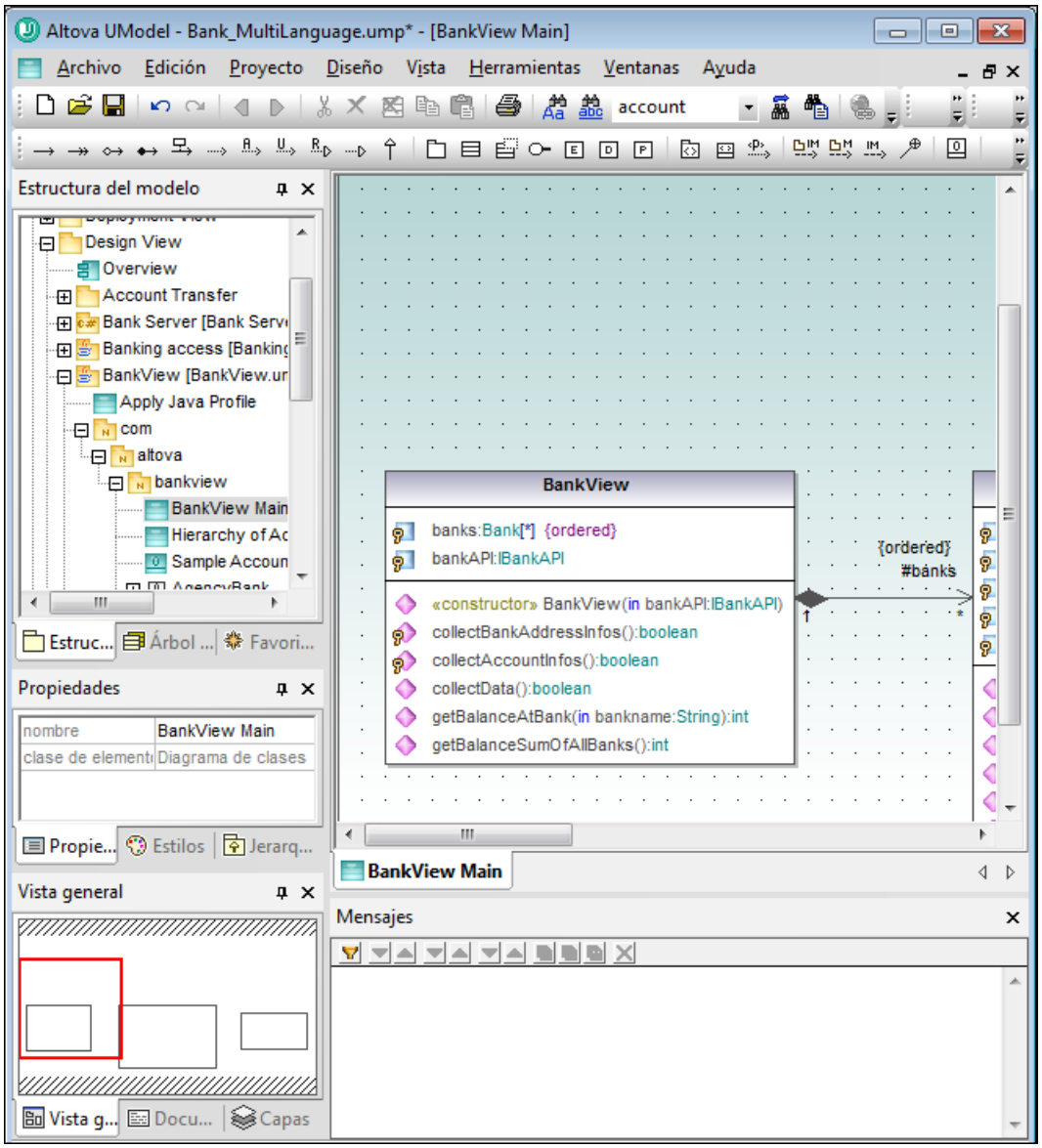
Altova UModel® 2015

Interfaz del usuario

5 Interfaz del usuario

La interfaz del usuario de UModel se divide en dos secciones: a la izquierda de la pantalla se encuentra una serie de paneles y a la derecha está la zona de trabajo donde se abren las pestañas de los diagramas. Los paneles de la izquierda permiten visualizar el proyecto de UModel desde diferentes puntos de vista y editar datos. Se trata de los paneles *Estructura del modelo*, *Propiedades* y *Vista general*. El área de trabajo situada a la derecha muestra la pestaña del diagrama activo (p. ej. en la imagen siguiente aparece el diagrama de clases del paquete **`BankView Main`**).

Nota: en todos los paneles y en todas las pestañas de diagramas puede buscar elementos y componentes desde el cuadro combinado *Buscar* de la barra de herramientas principal (p. ej. en la imagen siguiente se buscó el texto `account`) o pulsando la tecla **Ctrl+F**.



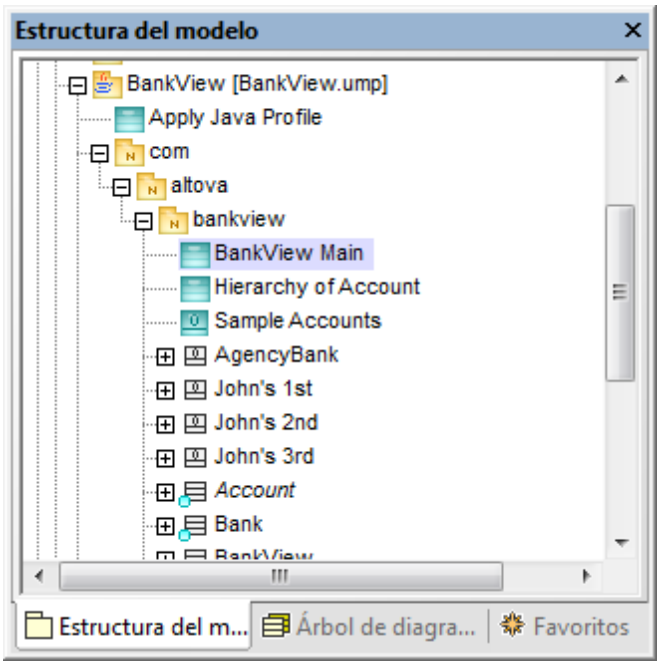
 © 2014 Altova GmbH

5.1 Estructura del modelo

La pestaña *Estructura del modelo* sirve para manipular componentes de los modelos y para navegar por determinados componentes del área de trabajo. Al hacer clic con el botón secundario en un elemento aparece un menú contextual con varios comandos. Dependiendo del tipo de elemento seleccionado, el menú contextual incluye unos comandos u otros.

Estas son las operaciones que se pueden realizar en los elementos de modelado desde la pestaña *Estructura del modelo* directamente:

- Agregar / insertar
- Copiar / mover
- Eliminar
- Cambiar de nombre
- Ordenar según diferentes criterios
- Restringir



En la pestaña *Estructura del modelo* cada icono en forma de carpeta equivale a un paquete UML.

Para agregar un paquete nuevo (o cualquier otro elemento de modelado):

1. Haga clic con el botón secundario en la **carpeta** bajo la que debe aparecer el nuevo paquete o elemento.
2. Seleccione **Nuevo/a | Paquete** (o el elemento de modelado que corresponda).

Para copiar o mover elementos de modelado:

Tiene dos opciones

1. Usar los comandos estándar de Windows para copiar, cortar o pegar o
2. arrastrar elementos de modelado a otro paquete. Si pulsa la tecla **Ctrl** mientras arrastra un elemento, se crea una copia del elemento.

   En algunas ocasiones, al arrastrar el elemento aparece un mensaje diciendo que se debe

activar la opción **No ordenar** para poder completar la acción. Consulte el apartado Cortar, copiar y pegar en diagramas de UModel para más información.

Para ordenar elementos en la *Estructura del modelo* (activando la opción No ordenar):

1. Haga clic con el botón secundario en el fondo de la pestaña *Estructura del modelo*.
2. Seleccione **Ordenar | No ordenar**.
   Ahora los elementos se pueden colocar en cualquier posición de la *Estructura del modelo*.

Nota: en el submenú **Ordenar** también hay opciones para ordenar las propiedades y las operaciones de los diagramas.

Para cambiar el nombre de un elemento:

1. Haga doble clic en el **nombre** del elemento y edítelo.
   Los paquetes `Root` y `Component View` son los únicos dos elementos cuyo nombre no se puede editar.

Para eliminar un elemento:

1. Haga **clic** en el elemento que desea eliminar (o **Ctrl+clic** para seleccionar varios elementos a la vez).
2. Pulse la tecla **Supr**.

   El elemento de modelado se elimina de la Estructura del modelo. Esto significa que también se elimina de la pestaña del diagrama y, si está en el proyecto, también del proyecto. Si quiere puede eliminar los elementos del diagrama pero no del proyecto. Para ello use **Ctrl+Supr**. Para más información consulte el apartado Eliminar elementos.

Para abrir un diagrama en el área de trabajo principal:

1. Haga doble clic en el icono del diagrama que desea abrir en el área de trabajo.

Iconos de los elementos de modelado en la *Estructura del modelo*

Tipos de paquetes:

Paquete UML
Paquete raíz del espacio de nombres Java
Paquete raíz del espacio de nombres C#
Paquete raíz del espacio de nombres Visual Basic
Paquete raíz de XML Schema
Paquete de código Java, C# y VB (las declaraciones de paquete se crean cuando se genera el código)

Tipos de diagramas:

Diagrama de actividades
Diagrama de clases
Diagrama de comunicación
Diagrama de componentes
Diagrama de paquetes
Diagrama de perfil
Diagrama de secuencia
Diagrama de máquina de

 © 2014 Altova GmbH

Diagrama de estructura de un compuesto
Diagrama de implementación
Diagrama global de interacción
Diagrama de objetos
estados
Diagrama de ciclo de vida
Diagrama de caso de uso
Diagrama de esquema XML
Diagramas BPMN
Diagramas SysML (9 tipos)

Tipos de elementos:

Los elementos que están visibles en el diagrama activo aparecen con un punto azul en la parte inferior. En este caso se trata de un elemento de tipo clase.

Instancia de clase / objeto
Slot de instancia

Clase
Propiedad
Operación
Parámetro

Actor (visible en el diagrama de casos de uso activo)
Caso de uso
Componente
Nodo
Artefacto
Interfaz

Relaciones (o paquete)
Restricciones

Para abrir y expandir paquetes en la *Estructura del modelo*:

Hay dos maneras de abrir paquetes en la *Estructura del modelo*. Una consiste en abrir todos los paquetes y paquetes subordinados y la otra consiste en abrir el paquete seleccionado solamente.

Haga clic en el paquete que desea abrir y elija una de estas dos opciones:

- Pulse la tecla ***** para abrir el paquete seleccionado y todos sus paquetes subordinados.
- Pulse la tecla **+** para abrir el paquete seleccionado solamente.

  Para contraer los paquetes pulse la tecla **-** del teclado. Para contraerlos haga clic en el paquete `Root` y pulse **-**.
  Recuerde que puede usar las teclas estándar o las del teclado numérico.

Para buscar elementos de modelado en las pestañas de diagramas:

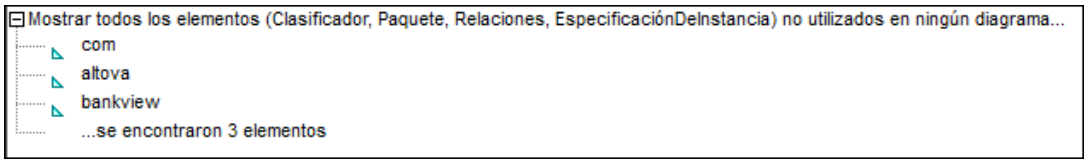
Mientras navega por los elementos de la *Estructura de modelo* puede ver si el elemento está en el diagrama y, si lo está, en qué posición. Hay dos maneras de buscar elementos:

1. En la *Estructura del modelo* haga clic con el botón secundario en el elemento que desea buscar y después seleccione:

- **Mostrar el elemento en el diagrama activo**
- **Mostrar el elemento en todos los diagramas**

Para generar una lista de elementos no utilizados en ningún diagrama:

1. Haga clic con el botón secundario en el paquete que desea inspeccionar.
2. Seleccione el comando **Mostrar elementos no utilizados en ningún diagrama**.
   En el panel Mensajes aparece una lista de elementos no utilizados. Los elementos que aparecen entre paréntesis son los tipos de elemento no utilizados. Consulte el apartado sobre la pestaña *Vista* del cuadro de diálogo "Opciones" de la *Referencia del usuario*.



   Para buscar los elementos que faltan en la Estructura del modelo, haga clic en el nombre del elemento en el resultado del panel Mensajes.

Nota: los elementos no utilizados son los del paquete actual y de sus paquetes subordinados.

Paquetes en la *Estructura del modelo*:

Al iniciar UModel por primera vez solamente se pueden ver los paquetes `Root` y `Component View` (porque no hay cargado ningún proyecto).

En la *Estructura del modelo* puede:

- **Crear** paquetes o eliminarlos (los paquetes son contenedores que almacenan los demás elementos de modelado UML, los diagramas de casos de uso, etc.).
- **Mover**/copiar paquetes y su contenido a otros paquetes (y a diagramas válidos del área de trabajo).
- **Ordenar** paquetes y su contenido según determinados criterios.
- **Colocar** los paquetes dentro de otros paquetes.
- **Utilizar** los paquetes como elementos de origen y destino cuando genere o sincronice código.

Para generar y combinar código:

En UModel puede generar o combinar código de programa desde la *Estructura del modelo* directamente. Consulte el apartado Sincronizar el modelo y el código fuente para obtener más información.

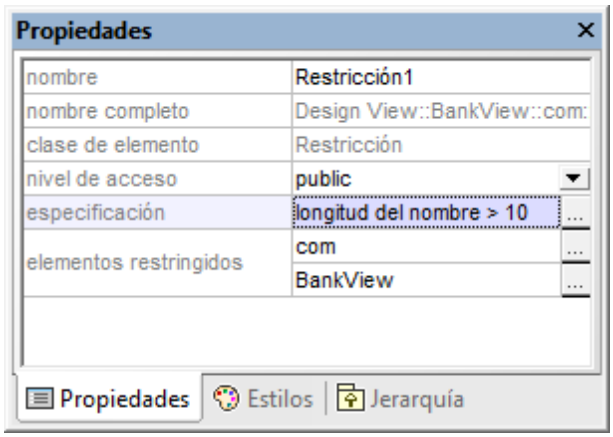
Para restringir elementos UML desde la *Estructura del modelo*:

Puede definir restricciones para la mayoría de los elementos de modelado. Tenga en cuenta que la revisión de sintaxis pasa por alto las restricciones porque no participan en el proceso de generación de código Java.

1. En la *Estructura del modelo* haga clic con el botón secundario en el elemento que desea restringir y seleccione **Nuevo/a | Restricción**.

 © 2014 Altova GmbH

2. Escriba el nombre de la restricción y pulse **Entrar**.
3. Haga clic en el campo `especificación` de la pestaña *Propiedades* y escriba la restricción (p. ej. longitud del nombre > 10).



Para restringir elementos UML desde el área de trabajo del diagrama:

1. Haga doble clic en el elemento para poder editarlo.
2. Añada la restricción entre llaves (p. ej. `interestRate:float #{interestRate >=0}`).

Para asignar restricciones a varios elementos de modelado:

1. Haga clic con el botón secundario en el campo `elementos restringidos` de la pestaña *Propiedades*.
2. Seleccione el comando **Agregar elemento a elementos restringidos** en el menú contextual.
   Esto abre el cuadro de diálogo "Seleccione el elemento que se debe restringir".
3. Seleccione el elemento al que desea asignar la restricción actual.

   El campo elementos restringidos ahora contiene los nombres de los elementos de modelado a los que se asignó la restricción.

5.2 Árbol de diagramas

La pestaña *Árbol de diagramas* muestra todos los diagramas disponibles actualmente en UModel de dos formas diferentes:

- Agrupados por tipo de diagrama (y en orden alfabético)
- En una lista alfabética de todos los diagramas del proyecto

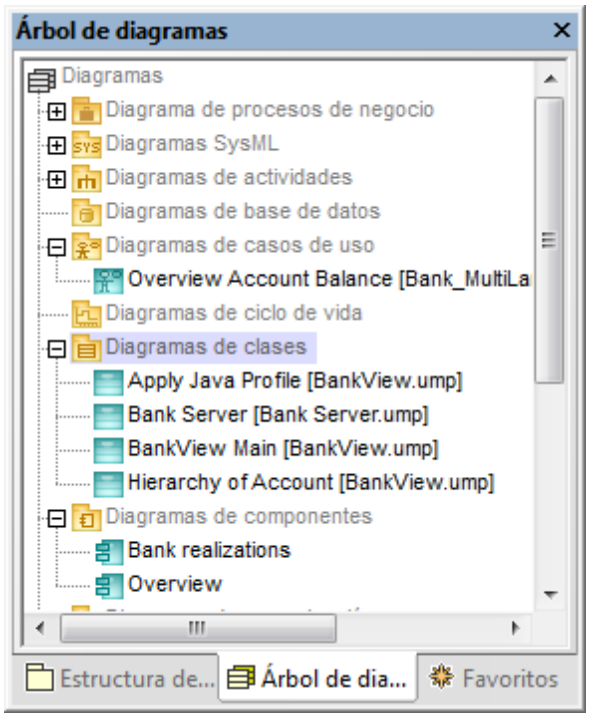
Nota: en la pestaña *Árbol de diagramas* puede añadir o eliminar diagramas haciendo clic con el botón secundario y seleccionando el comando correspondiente.

Para abrir un diagrama en la pestaña Diagramas:

- Haga doble clic en el diagrama que desea ver en el área de trabajo.

Para ver los diagramas agrupados según el tipo de diagrama:

- Haga clic con el botón secundario en el fondo de la pestaña y active la opción **Agrupar según el tipo de diagrama**.

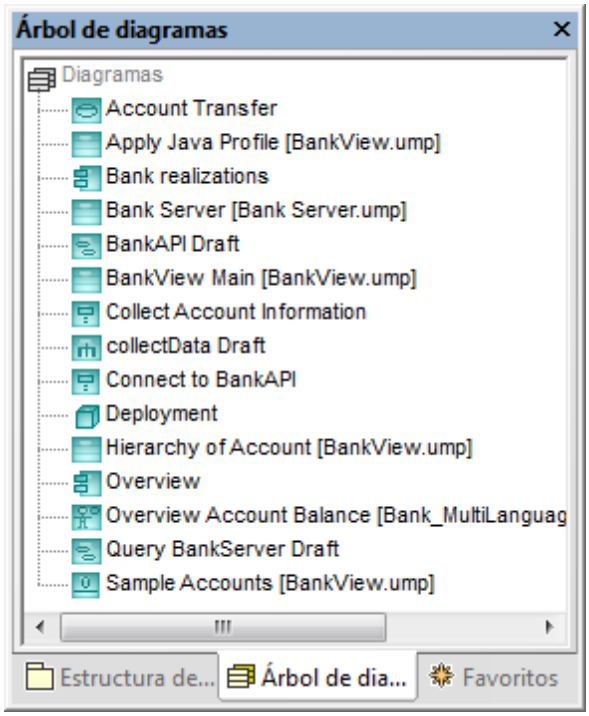


Los diagramas aparecen organizados por grupos.

Para ver todos los diagramas en una lista alfabética:

- Haga clic con el botón secundario y desactive la opción **Agrupar según el tipo de diagrama**.

 © 2014 Altova GmbH



Todos los diagramas del proyecto aparecen ordenados alfabéticamente.

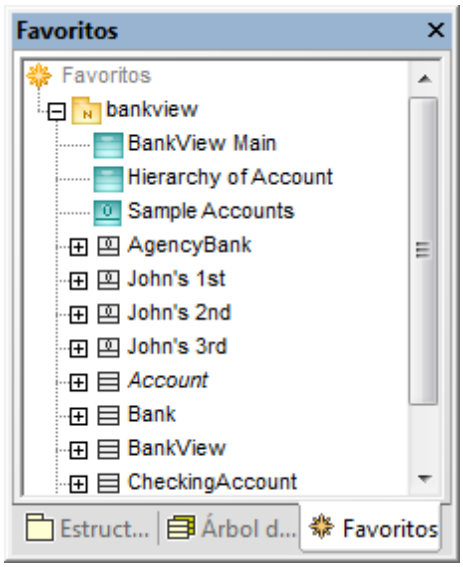
5.3 Favoritos

La pestaña *Favoritos* sirve para crear un repositorio (o biblioteca) personal donde puede almacenar elementos UML **con nombre** (es decir, clases, objetos, asociaciones, etc.) pero no dependencias de tipo `AplicaciónDePerfil` ni `Generalización`. Por tanto, en esta pestaña puede crear una lista de elementos de modelado favoritos y así poder acceder a ellos rápidamente.

El contenido de la pestaña *Favoritos* se guarda automáticamente en el archivo de proyecto. Esto se puede cambiar en la pestaña *Archivo* del cuadro de diálogo "Opciones" (**Herramientas | Opciones**), activando/desactivando la casilla *Cargar y guardar con el archivo de proyecto*.

Para agregar un elemento de modelado a la pestaña *Favoritos*:

1. En la *Estructura del modelo* o en el área de trabajo haga clic con el botón secundario en el elemento.
2. Seleccione el comando **Agregar a favoritos** del menú contextual.
3. Haga clic en la pestaña *Favoritos* y compruebe que el elemento se añadió efectivamente al repositorio.



El elemento que aparece en la pestaña *Favoritos* es una vista del elemento (es decir, no es ni una copia ni un clon del elemento).

Para agregar un elemento nuevo a la pestaña *Favoritos*:

1. Haga clic con el botón secundario en el paquete al que desee añadir un elemento.
2. Seleccione la opción **Elemento nuevo | <elemento de modelado>** (una clase, un componente, etc.).
   Al mismo paquete/elemento del proyecto se añade el elemento nuevo, que por tanto se podrá ver también en la *Estructura del modelo*.

Para quitar un elemento de la pestaña *Favoritos*:

1. Haga clic con el botón secundario en el elemento/paquete en la pestaña *Favoritos*.
2. Seleccione **Quitar de favoritos**.

 © 2014 Altova GmbH

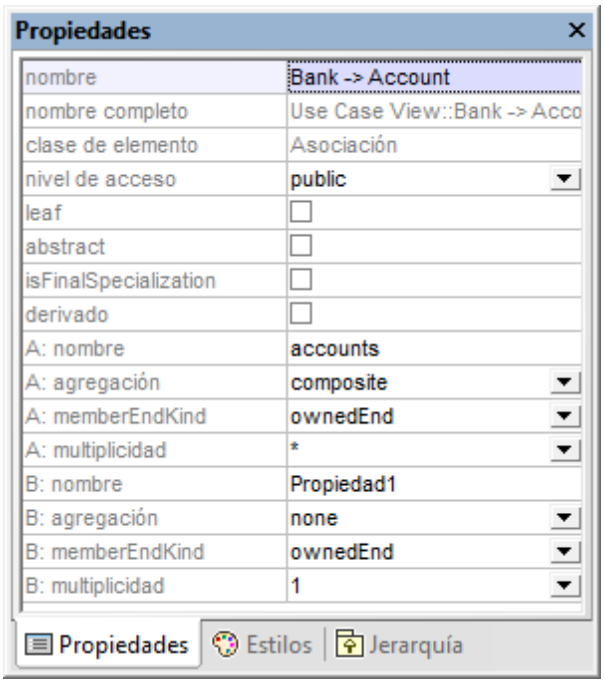
Nota: los elementos se pueden quitar o agregar a favoritos desde la pestaña *Favoritos* y desde la pestaña *Estructura del modelo*.

Para eliminar elementos de la pestaña *Favoritos*:

1. Haga clic con el botón secundario en el elemento que desea eliminar y pulse la tecla **Supr**.
   Aparece un mensaje informando de que el elemento se eliminará del proyecto.
2. Haga clic en **Aceptar** si de verdad quiere eliminarlo del proyecto.
3. Haga clic en **Cancelar** para conservar el elemento. Puede usar el comando **Quitar de favoritos** para eliminarlo de la pestaña *Favoritos* solamente.

5.4 Propiedades

La pestaña *Propiedades* muestra las propiedades UML del elemento de modelado activo.



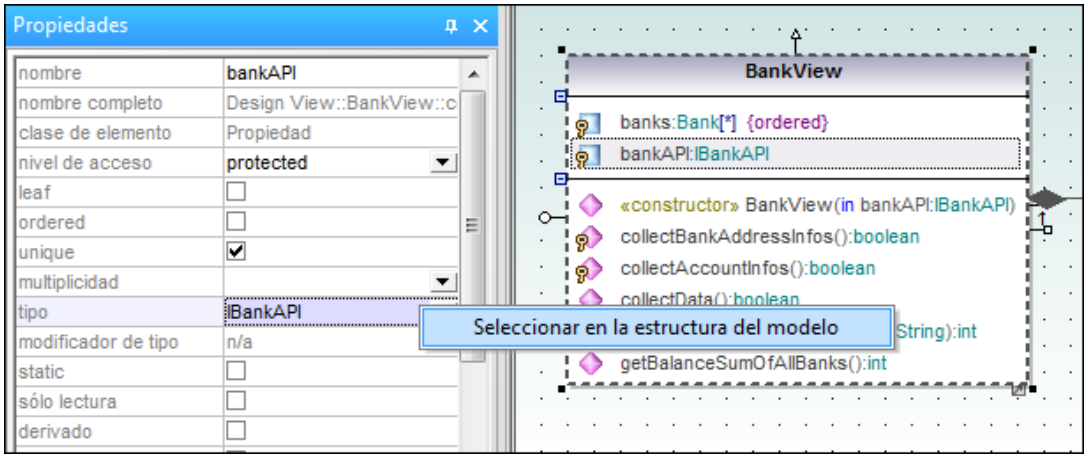
- Al hacer clic en un elemento de modelado en cualquiera de las vistas y pestañas, la pestaña *Propiedades* muestra las propiedades del elemento.
- Las propiedades que aparecen en la pestaña *Propiedades* se pueden editar, rellenar y modificar seleccionando diferentes opciones disponibles en menús contextuales y desplegables.
- Las propiedades seleccionadas en la pestaña *Propiedades* se pueden ver en la pestaña del diagrama haciendo clic en la opción **Mostrar el elemento en el diagrama activo** del menú contextual.

Selección en la *Estructura del modelo*

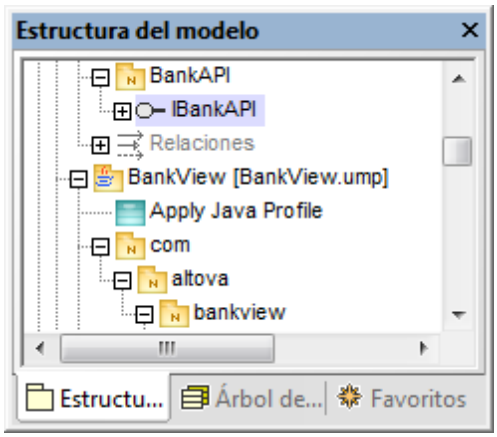
Si hace clic en un atributo de un diagrama de clases, sus propiedades aparecen en la pestaña *Propiedades*. Para buscar el atributo en la *Estructura del modelo*:

1. Haga clic con el botón secundario en la entrada `tipo` en la pestaña *Propiedades*.
2. En el menú contextual elija **Seleccionar en la estructura del modelo**.

 © 2014 Altova GmbH

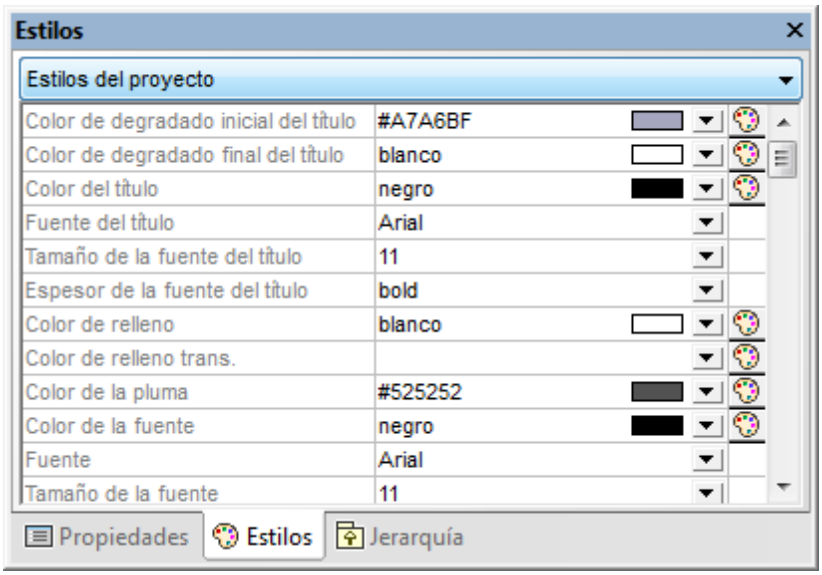


Ahora puede ver la interfaz **`IBankAPI`** en la *Estructura del modelo*.



5.5 Estilos

La pestaña *Estilos* sirve para ver y modificar los atributos de los diagramas o elementos que aparecen en el área de trabajo.



Estos atributos de estilo se pueden dividir en dos grupos:

- Atributos de formato (p. ej. tamaño, espesor, color de la fuente, etc.)
- Atributos de presentación (p. ej. ver color de fondo, ver cuadrícula, opciones de visibilidad, etc.)

La pestaña *Estilos* está dividida a su vez en varias secciones. Para cambiar de sección haga clic en el cuadro combinado situado en la parte superior y elija una opción del menú desplegable. Dependiendo del elemento seleccionado, el menú desplegable ofrece unas opciones u otras.

Si selecciona un elemento en la pestaña del diagrama, se selecciona automáticamente el contexto **`Estilos del elemento`**. Si hace clic en un elemento en la pestaña *Estructura del modelo*, se selecciona el contexto **`Estilos del proyecto`**.

El orden de prioridad de los estilos es ascendente, es decir, los cambios realizados en un nivel más concreto, reemplazan los estilos más generales. Por ejemplo, los cambios realizados en un objeto en el contexto **`Estilos del elemento`** reemplazan la configuración actual en el contexto **`Estilos de la familia de elementos`** y **`Estilos del proyecto`**. Sin embargo, si selecciona otro objeto y cambia alguna opción en **`Estilos de la familia de elementos`**, se actualizan

 © 2014 Altova GmbH

los demás objetos excepto el que acabamos de cambiar en el contexto **`Estilos del elemento`**.

Nota: los cambios realizados en los estilos de los elementos de modelado no se pueden deshacer.

Estilos del elemento:
Este contexto afecta el elemento seleccionado en el diagrama activo. Puede seleccionar varios elementos a la vez.

Estilos de los elementos con este estereotipo:
Se aplican a la clase de estereotipo seleccionada en el diagrama (consulte Estilos de estereotipo definidos por el usuario para más información).

Estilos de la familia de elementos:
Abarca todos los elementos que sean del mismo tipo que el elemento seleccionado. P. ej. si quiere que todos los elementos de tipo `Componente` estén en color azul.

Estilos del nodo / de la línea:
"Nodo" abarca todos los objetos rectangulares. "Líneas" se refiere a todos los conectores: asociaciones, dependencias, líneas de realización, etc. de todo el proyecto.

Estilos del proyecto:
Se aplican a todo el proyecto de UModel activo (p. ej. si quiere cambiar la fuente Arial predeterminada por Times New Roman en todo el texto de todos los diagramas del proyecto).

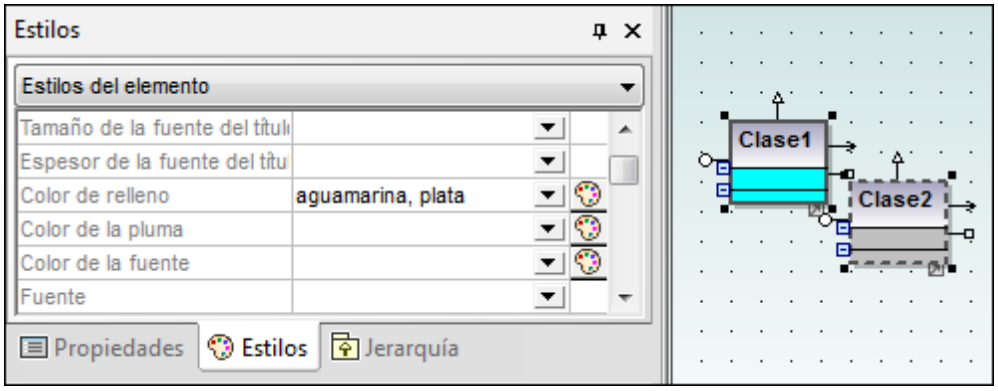
Estilos del diagrama:
Estos estilos están disponibles si hace clic/selecciona el fondo de un diagrama. Estos estilos solo se aplican al diagrama UML para el que se definieron opciones en el proyecto.

Para cambiar la configuración de todos los diagramas del proyecto:

1. Haga clic en el diagrama correspondiente.
2. En el cuadro combinado de la pestaña *Estilos* seleccione la entrada **`Estilos del proyecto`** y desplácese hasta el final de la lista de la pestaña.
3. Seleccione uno de los estilos `Diagrama - xxxx` (p. ej. `Diagrama - Color de fondo`). A continuación el color de fondo de todos los diagramas del proyecto actual cambiar al color elegido.

¿Cómo se ven los estilos cuando hay varios elementos seleccionados?

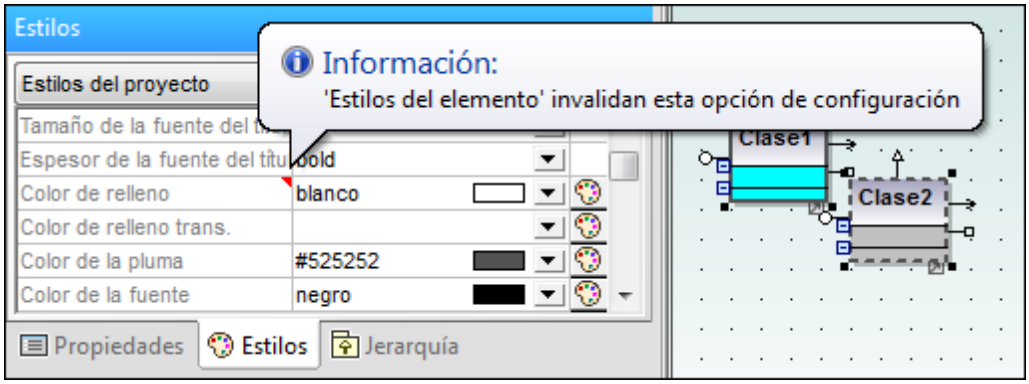
Si selecciona varios elementos en el área de trabajo, todos los valores de estilo aparecen en el campo correspondiente de la pestaña *Estilos*. En la imagen siguiente, p. ej., se seleccionaron **`Clase1`** y **`Clase2`**. El campo `Color de relleno` muestra los valores de ambos elementos (es decir, `aguamarina` y `plata`).



Para ver estilos en cascada:

Si un estilo es reemplazado en un nivel más concreto, entonces aparece un pequeño triángulo rojo en el campo del estilo en la pestaña *Estilos*.

Al pasar el puntero del mouse por encima del campo aparece información sobre el orden de prioridad de los estilos.



Por ejemplo, los elementos **`Enumeración`**, **`Paquete`** y **`Perfil`** tienen definido su color de relleno predeterminado en el contexto **`Estilos de la familia de elementos`**. Para cambiar el color de fondo a nivel de proyecto, borre el valor en el contexto **`Estilos de la familia de elementos`** (seleccione la entrada vacía de la lista), seleccione **`Estilos del proyecto`** y cambie aquí el color de relleno.

 © 2014 Altova GmbH

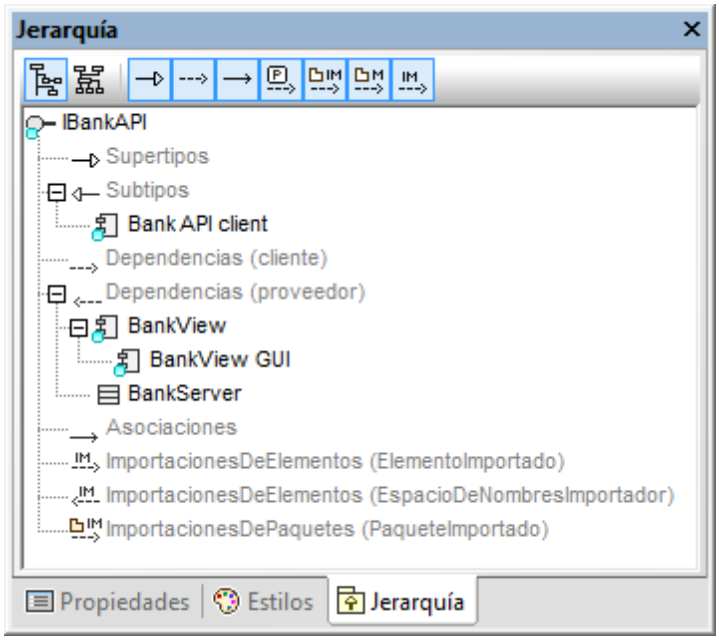
5.6 Jerarquía

La pestaña *Jerarquía* muestra de dos formas distintas todas las relaciones que tiene el elemento seleccionado. El elemento de modelado se puede seleccionar en el área de trabajo del diagrama, en la *Estructura del modelo* o en la pestaña *Favoritos*.

Nota: la configuración general de la pestaña *Jerarquía* se define en la pestaña *Vista* del cuadro de diálogo "Opciones locales" (**Herramientas | Opciones**), en el grupo de opciones *Jerarquía*.

Para ver las relaciones del elemento en forma jerárquica (de árbol):

Seleccione el elemento y después pulse el icono de la barra de herramientas de la pestaña *Jerarquía*.

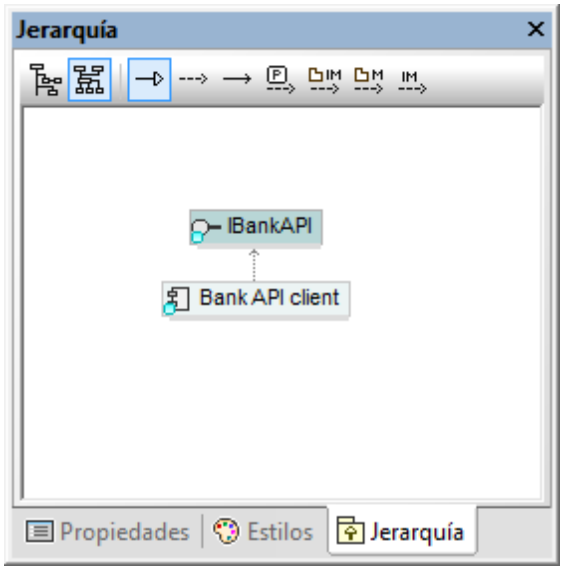


Esta vista de la pestaña *Jerarquía* muestra varias relaciones del elemento seleccionado p. ej. `IBankAPI`. Haga clic en los iconos de la barra de herramientas para seleccionar qué tipo de relaciones aparecen en la vista. En la imagen anterior, por ejemplo, están activos todos los iconos. Es decir, la vista muestra todas las relaciones del elemento.

Haga doble clic **en el icono** de un elemento de la lista para ver las relaciones de dicho elemento.

Para ver las relaciones del elemento en forma de diagrama:

Seleccione el elemento y después pulse el icono de la barra de herramientas de la pestaña *Jerarquía*.

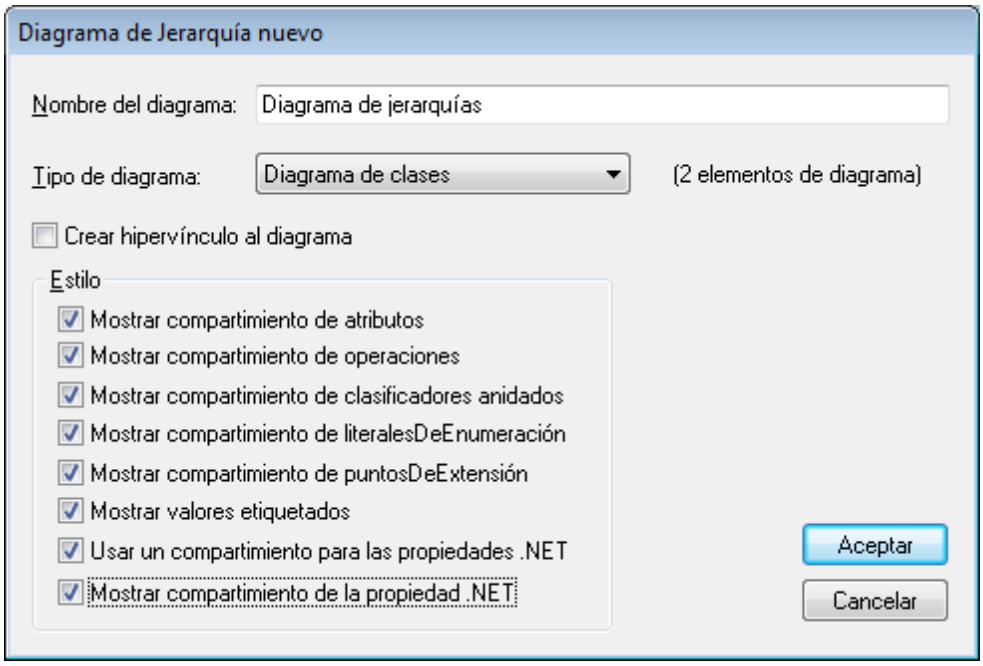


Esta vista de la pestaña *Jerarquía* resume de forma jerárquica un solo conjunto de relaciones. En esta vista solo puede haber un icono activo en la barra de herramientas. En la imagen anterior, por ejemplo, está activo el icono **Mostrar generalizaciones**.

Haga doble clic **en el icono** de un elemento de la vista (p. ej. `Bank API client`) para ver las relaciones de dicho elemento.

Para crear un diagrama nuevo a partir del contenido de la pestaña *Jerarquía*:
El contenido actual de la vista en forma de diagrama de la pestaña *Jerarquía* puede verse en un diagrama nuevo.

1. Haga clic con el botón secundario en el fondo de la pestaña y seleccione **Crear diagrama basado en este gráfico**.

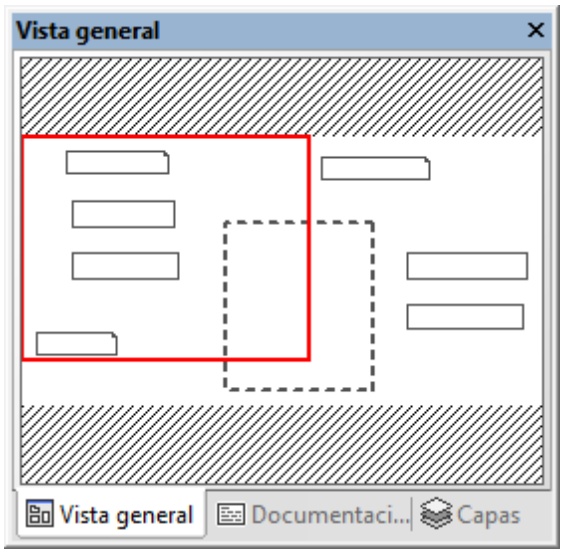


 © 2014 Altova GmbH

2. Edite el nombre del diagrama, seleccione las opciones de estilo y haga clic en **Aceptar**.

5.7 Vista general

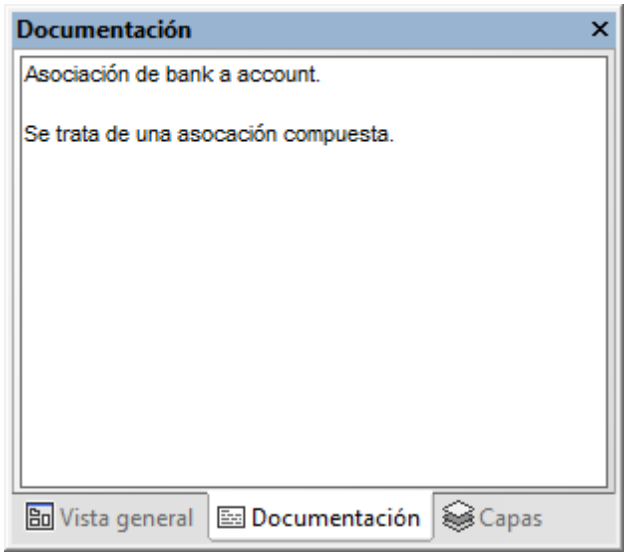
La pestaña *Vista general* ofrece una vista resumen del diagrama activo. Haga clic en el rectángulo rojo y arrástrelo para desplazarse por la vista del diagrama.



© 2014 Altova GmbH

5.8 Documentación

La pestaña *Documentación* sirve para documentar los elementos UML que están disponibles en la pestaña *Estructura del modelo*. Haga clic en el elemento que desea documentar y escriba sus comentarios en la pestaña *Documentación*. Aquí puede usar las teclas de acceso rápido estándar para cortar, copiar y pegar.



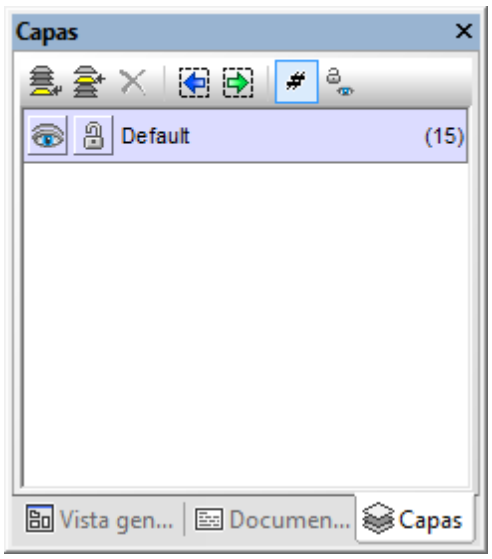
Documentación y generación de código:
Durante la generación de código la única entrada/salida es la documentación de clases e interfaces. Esto incluye la documentación definida para propiedades y operaciones de clase/interfaz.

1. Seleccione la opción de menú **Proyecto | Configuración del proyecto**.
2. Marque la casilla *JavaDocs como documentación*.

Nota: cuando importe esquemas XML, en la pestaña *Documentación* solo aparece la primera anotación de un tipo complejo o simple.

5.9 Capas

La pestaña *Capas* sirve para crear capas en los diagramas y configurarlas. La principal función de las capas es agrupar de forma lógica los elementos de modelado de un diagrama. Por ejemplo, puede agrupar información interna en una capa de clases o tener una capa formada por clases inacabadas solamente. En la pestaña *Capas* puede ver capas determinadas, varias capas a la vez, las capas que están bloqueadas o las capas que están ocultas.



Nota: puede asignar los elementos UML que quiera a las capas. Por ejemplo, en una capa puede tener las asociaciones y en otra las notas, etc. Los elementos nuevos se añaden automáticamente a la capa activa, es decir, la que está resaltada en la pestaña *Capas*. Un diagrama puede tener 20 capas como máximo.

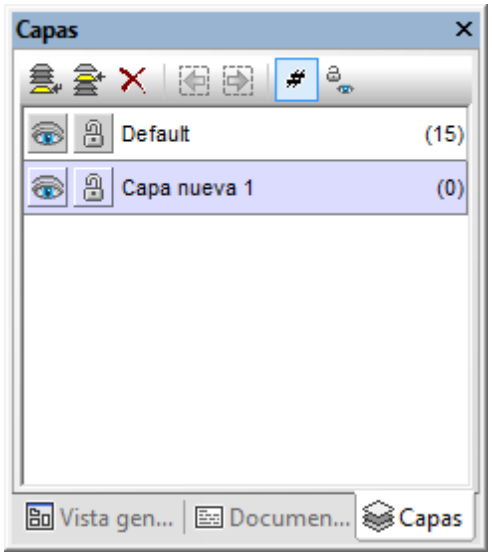
A continuación describimos las opciones del menú contextual que aparece al hacer clic con el botón secundario en la pestaña *Capas:*

- Puede crear asociaciones, dependencias, etc. entre elementos en capas diferentes, si las capas están visibles.
- Puede ver y trabajar en varias capas al mismo tiempo.
- Puede mover los elementos de una capa a otra.
- Puede eliminar las capas y combinar sus elementos con los de otra capa.
- Cuando imprima diagramas, solo se imprimen los elementos de las capas que están visibles.

Anexar una capa nueva

Este icono sirve para anexar una capa nueva a la lista actual de capas. La capa nueva recibe el nombre predeterminado `Capa nueva 1` (*imagen siguiente*).

 © 2014 Altova GmbH



Insertar una capa nueva

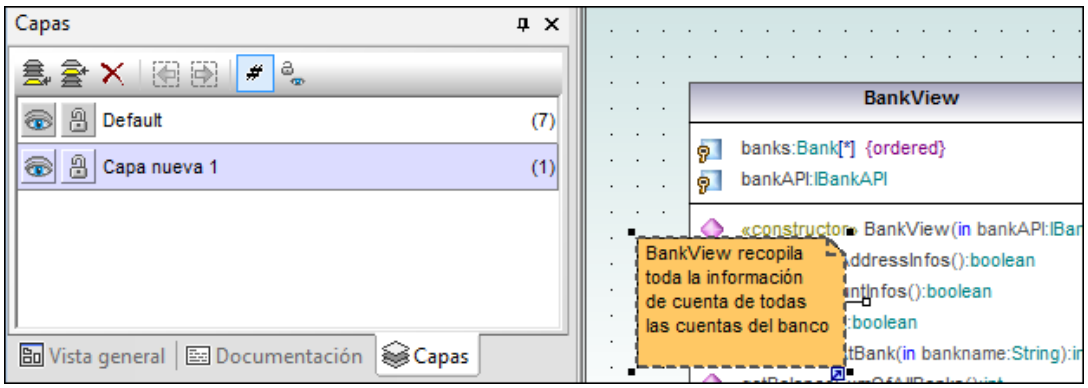
Este icono sirve para insertar una capa nueva por encima de la capa activa de la lista.

Una vez insertada, la capa nueva se convierte en la capa activa. Por ejemplo, en la imagen anterior, la pestaña *Capas* muestra la capa predeterminada `Default` y la capa nueva `Capa nueva 1` y, por cada una de ellas, indica:

- Si la capa está visible.
- Si la capa está bloqueada.
- Cuántos elementos incluye.

Para agregar elementos a una capa nueva:

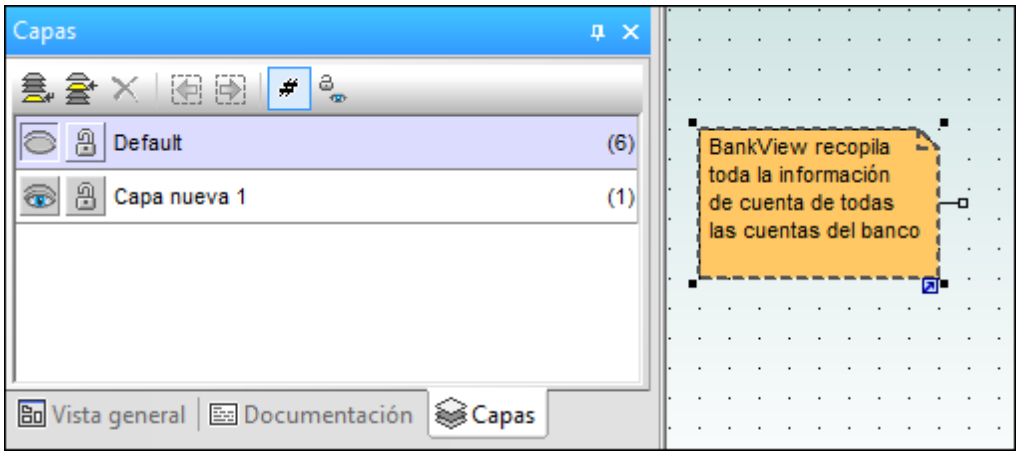
1. En la pestaña *Capas* haga clic en la capa a la que desea añadir elementos (p. ej. en `Capa nueva 1`).
2. Inserte los objetos nuevos en el área de trabajo del diagrama (p. ej. una nota con texto que describa una clase).



Observe que el número de elementos de esta capa pasa de `0` a `1` y que los elementos de la capa predeterminada `Default` también están visibles.

Para ver los elementos de una capa determinada:

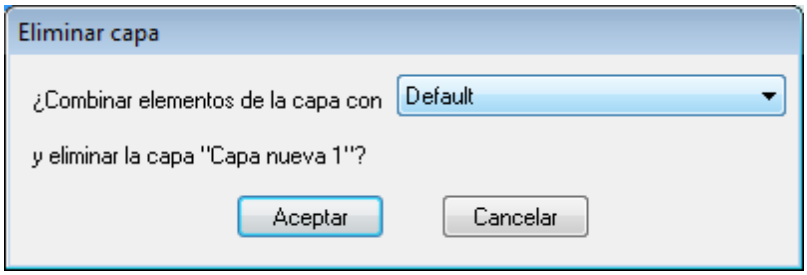
1. Haga clic en el icono **mostrar/ocultar** de las capas pertinentes para que en el área de trabajo estén visibles solamente los elementos de la capa que le interesa en ese momento.
2. Por ejemplo, haga clic en el icono **mostrar/ocultar** de la capa predeterminada `Default` para ver los elementos de `Capa nueva 1` solamente.



Observe que el icono **mostrar/ocultar** ha cambiado y que en el área de trabajo solamente está visible la **`nota`** de `Capa nueva 1`.

Para eliminar una capa:

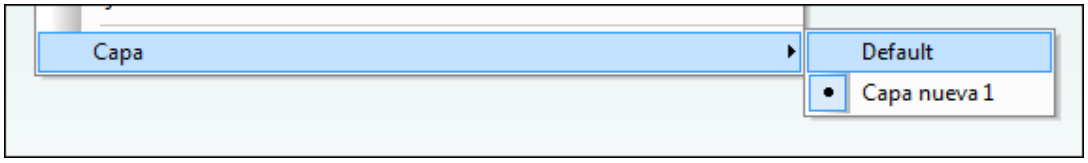
1. Haga clic con el botón secundario en la capa que desea eliminar y seleccione **Eliminar** en el menú contextual.
   UModel le pregunta si quiere combinar los elementos de la capa con una de las capas ya existentes antes de eliminar la capa elegida.
   Haga clic en **Aceptar** para combinar los elementos y eliminar la capa. Haga clic en **Cancelar** para cancelar la acción.



Para mover elementos a otra capa:

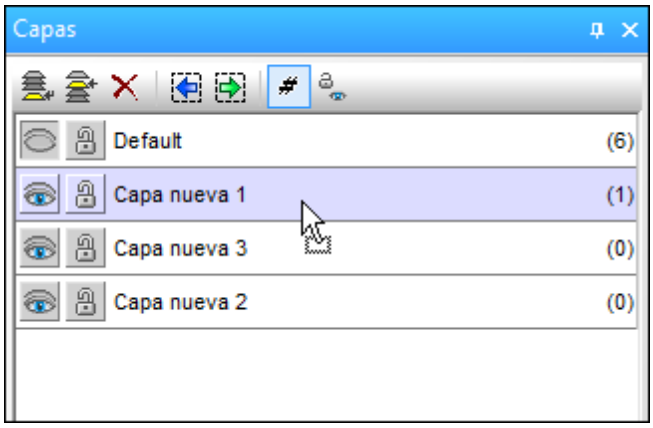
1. Haga clic con el botón secundario en el elemento que desea mover y en el menú contextual seleccione la opción **Capa | <Nombre de la capa>** (es decir, la capa a la que desea mover el elemento).
2. Por ejemplo, haga clic con el botón secundario en el elemento **`nota`** y seleccione **Capa | Default**.

 © 2014 Altova GmbH



Otra manera de hacer lo mismo:

1. Haga clic en el elemento y arrástrelo hasta la capa de destino en la lista de la pestaña *Capas* (*imagen siguiente*).



Para bloquear elementos de una capa:

- Haga clic en el icono **Bloquear** de la capa pertinente para bloquear la posición de todos los elementos de la capa.

  Recuerde que los elementos nuevos se pueden colocar en la capa activa, pero una vez insertados no se pueden seleccionar ni mover.

Para seleccionar elementos de una capa determinada:

Primero debe asegurarse de que la capa está desbloqueada.

1. Seleccione la capa que contiene los elementos que desea seleccionar.
2. Haga clic en los iconos **Elemento anterior** o **Siguiente elemento** de la barra de herramientas de la pestaña *Capas*.
   Los elementos de la capa activa se van seleccionando uno después del otro con cada clic (es decir, con estos iconos puede recorrer todos los elementos de la capa activa hacia adelante y hacia detrás).

Para seleccionar todos los elementos de una capa:

1. En la pestaña *Capas* haga clic con el botón secundario en la capa pertinente y seleccione **Elementos | Seleccionar**.
   Esto selecciona todos los elementos de la **capa activa**.

Para seleccionar los elementos de todas las capas excepto de la capa activa:

1. En la pestaña *Capas* haga clic con el botón secundario en la capa pertinente y seleccione **Elementos | Seleccionar el resto**.

Esto selecciona todos los elementos de todas las capas que están visibles, **excepto** los de la capa activa.

Para activar/desactivar el contador de elementos de la lista de capas:

- Haga clic en el icono de la barra de herramientas.

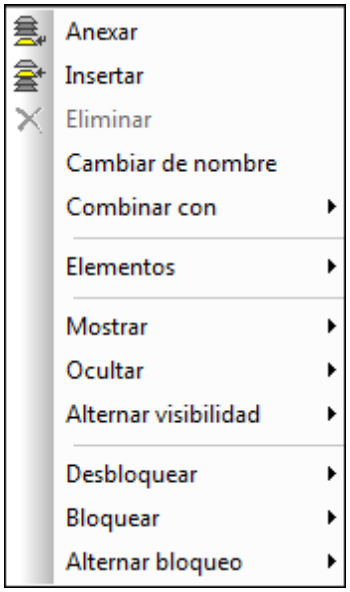
Para restaurar todos los estados de las capas:

- Haga clic en el icono de la barra de herramientas para restaurar todas las capas al estado **visible** y **desbloqueado**.

© 2014 Altova GmbH

5.9.1 Menú contextual de las capas

Al hacer clic con el botón secundario en la lista de capas se abre este menú contextual:



Anexar
Anexa una capa nueva a la lista de capas y le asigna un nombre predeterminado que se puede cambiar inmediatamente o más tarde con el comando **Cambiar de nombre** del menú contextual.

Insertar
Inserta una capa nueva encima de la capa activa de la lista.

Eliminar
Elimina la capa activa y mueve sus elementos a otra capa seleccionada por el usuario en un cuadro de diálogo.

Cambiar de nombre
Sirve para cambiar el nombre de la capa activa.

Combinar con
Mueve todos los elementos de la capa activa a la capa seleccionada por el usuario en el menú emergente. El menú emergente enumera todas las listas disponibles.

Estos comandos afectan solamente a los elementos que están visibles en la pestaña del diagrama, es decir, a los que están presentes en las capas que están activas/visibles en el diagrama.

Opciones del menú Combinar con:

Capa seleccionada	afecta solamente a la capa activa y a los elementos que están en esa capa
No seleccionada	afecta a **las demás capas** y a los elementos que están en ellas

Elementos
Elementos | Seleccionar selecciona/marca todos los elementos de la capa seleccionada.
Elementos | Seleccionar el resto selecciona/marca todos los elementos de las demás capas.

Mostrar
Mostrar | la capa seleccionada muestra todos los elementos de la capa seleccionada.
Mostrar | las otras capas muestra todos los elementos de las demás capas.

Ocultar
Ocultar | la capa seleccionada oculta todos los elementos de la capa seleccionada.
Ocultar | las otras capas oculta todos los elementos de las demás capas.

Alternar visibilidad
Alternar visibilidad | de la capa seleccionada alterna la visibilidad de todos los elementos de la capa seleccionada.
Alternar visibilidad | de las otras capas alterna la visibilidad de todos los elementos de las demás capas.

Desbloquear
Desbloquear | la capa seleccionada desbloquea todos los elementos de la capa seleccionada.
Desbloquear | las otras capas desbloquea todos los elementos de las demás capas.

Bloquear
Bloquear | la capa seleccionada bloquea todos los elementos de la capa seleccionada.
Bloquear | las otras capas bloquea todos los elementos de las demás capas.

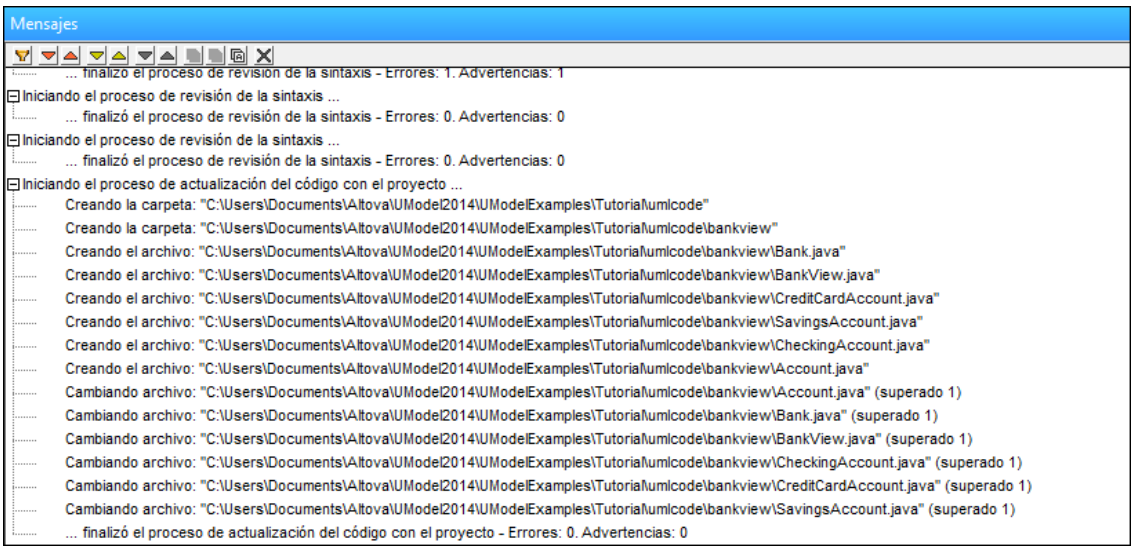
Alternar bloqueo
Alternar bloqueo | de la capa seleccionada alterna el bloqueo de todos los elementos de la capa seleccionada.
Alternar bloqueo | de las otras capas alterna el bloqueo de todos los elementos de las demás capas.

 © 2014 Altova GmbH

5.10 Mensajes

La ventana Mensajes muestra mensajes de advertencia, sugerencias y errores durante las operaciones de combinación de código y revisión de la sintaxis del proyecto.

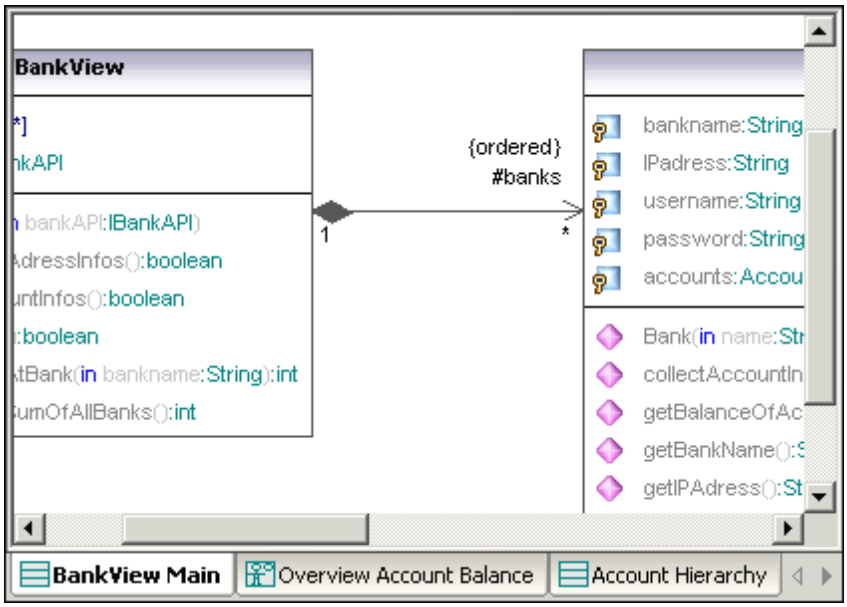
Al hacer clic en un mensaje de error el elemento correspondiente se resalta en la *Estructura del modelo* y en el diagrama pertinente, si está activo/abierto.



En la edición de UModel para Visual Studio y Eclipse, al hacer clic en un error de análisis de la ventana *Mensajes* el archivo de código fuente correspondiente se abre con el cursor situado en la línea que contiene el error.

5.11 Panel de diagramas

El panel de diagramas es el área de trabajo principal. Aquí se abren los diagramas de UModel en forma de pestañas.



Para crear un diagrama nuevo:

1. En la *Estructura del modelo* haga clic con el botón secundario en un paquete.
2. Seleccione el comando **Diagrama nuevo | <tipo de diagrama>**.

Para crear un diagrama nuevo con el contenido de un paquete ya existente:

1. En la *Estructura del modelo* haga clic con el botón secundario en un paquete.
2. Seleccione el comando **Mostrar en un diagrama nuevo | Contenido**.

Para abrir un diagrama tiene dos opciones:

- En la *Estructura del modelo*, en *Favoritos* o en el *Árbol de diagramas* haga doble clic **en el icono** del diagrama que quiere abrir.
- Haga clic en la pestaña del diagrama al que desea acceder en el área de trabajo principal.

Para cerrar todos los diagramas menos el diagrama activo:

- Haga clic con el botón secundario en la pestaña del diagrama que debe seguir abierto y seleccione la opción **Cerrar ventanas inactivas**.

Para eliminar un diagrama:

- En la *Estructura del modelo* haga clic en el icono del diagrama que desea eliminar y pulse la tecla **Supr**.

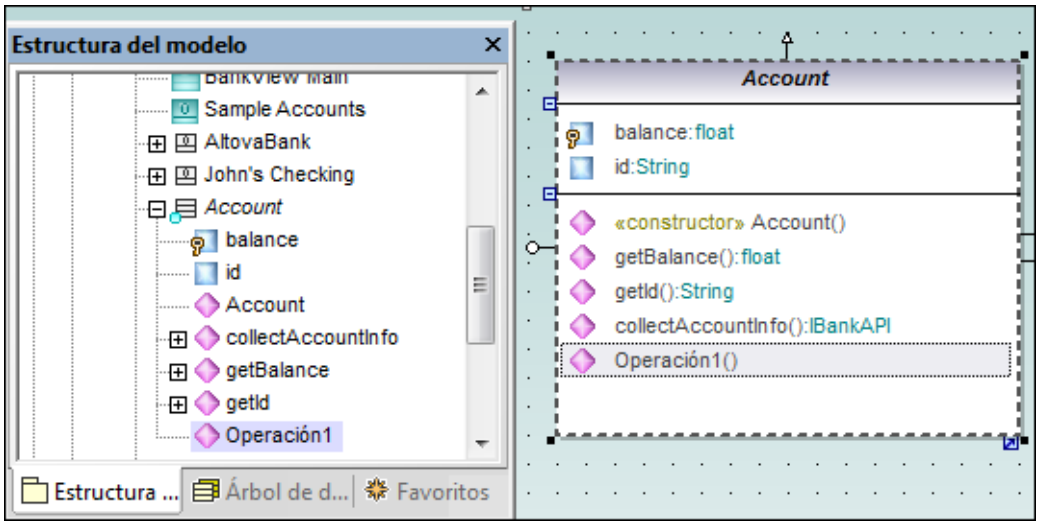
Para mover los diagramas:

 © 2014 Altova GmbH

- En la *Estructura del modelo* haga clic en el icono del diagrama y arrástrelo a otro paquete.
  A veces es necesario habilitar la opción **No ordenar** para poder mover el diagrama.

Para buscar (eliminar) propiedades de clase y opciones desde la *Estructura del modelo*:
Las propiedades y opciones también se pueden eliminar desde la *Estructura del modelo* directamente. Para hacerlo bien es importante buscar primero la propiedad que deseamos eliminar. Imaginemos que insertó la `Operación1` en la clase **`Account`** (pulse **F8** y **Entrar** para insertarla):

1. Ahora haga clic con el botón secundario en `Operación1` de la clase **`Account`**.
2. Seleccione la opción **Seleccionar en la estructura del modelo** (o pulse **F4**).
   El componente `Operación1` aparece resaltado debajo de *Account* en la pestaña *Estructura del modelo*.



3. Pulse la tecla **Supr** para eliminar la operación de la clase y del proyecto.
   Recuerde que casi todos los elementos de modelado aparecen en la *Estructura del modelo* cuando se pulsa **F4**.

Nota: también puede navegar hasta la *Estructura del modelo* desde el panel *Propiedades*. Para más información consulte el apartado Propiedades de la sección *Interfaz del usuario*.

Para eliminar elementos de un diagrama y del proyecto:

- Seleccione el elemento y pulse la tecla **Supr**.

Para eliminar elementos en el diagrama (y no del proyecto):

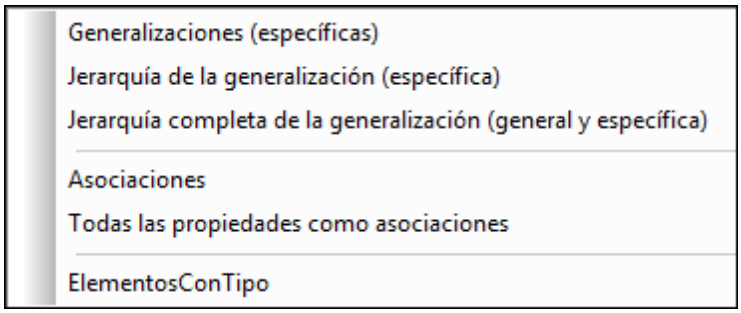
- Seleccione el elemento que desea eliminar en el diagrama y pulse **Ctrl+Supr**.

UModel también ofrece una función de diseño automático con la que puede definir la estructura visual de los diagramas. Haga clic con el botón secundario en el fondo del diagrama y en el menú contextual seleccione uno de estos comandos:

- **Aplicar diseño automática a todo | Diseño dirigido por fuerzas**
- **Aplicar diseño automática a todo | Diseño jerárquico**
- **Aplicar diseño automática a todo | Diseño por bloques**

Para ver las relaciones que existen entre los elementos de modelado:

Haga clic con el botón secundario en el elemento que le interesa y seleccione **Mostrar**. Este submenú ofrece diferentes opciones dependiendo del elemento seleccionado.

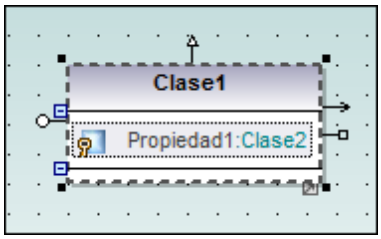


Para ver / ocultar las etiquetas de texto:

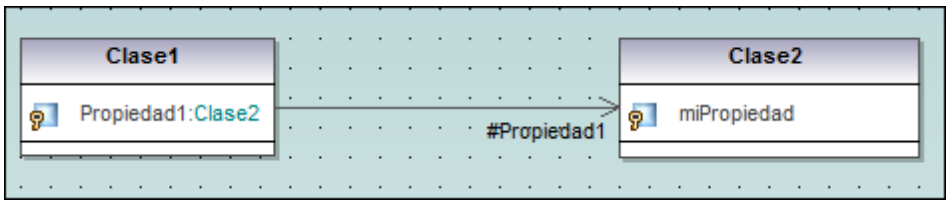
Haga clic con el botón secundario en una clase o en una flecha de asociación y seleccione **Etiquetas de texto | Mostrar (ocultar) todas las etiquetas de texto**.

Para ver un atributo / una propiedad de una clase en forma de asociación:

1. Haga clic con el botón secundario en la propiedad de la clase.



2. Seleccione la opción **Mostrar | <nombre de la propiedad> como asociación**. Esto inserta/abre la clase a la que se hace referencia y muestra la correspondiente asociación.

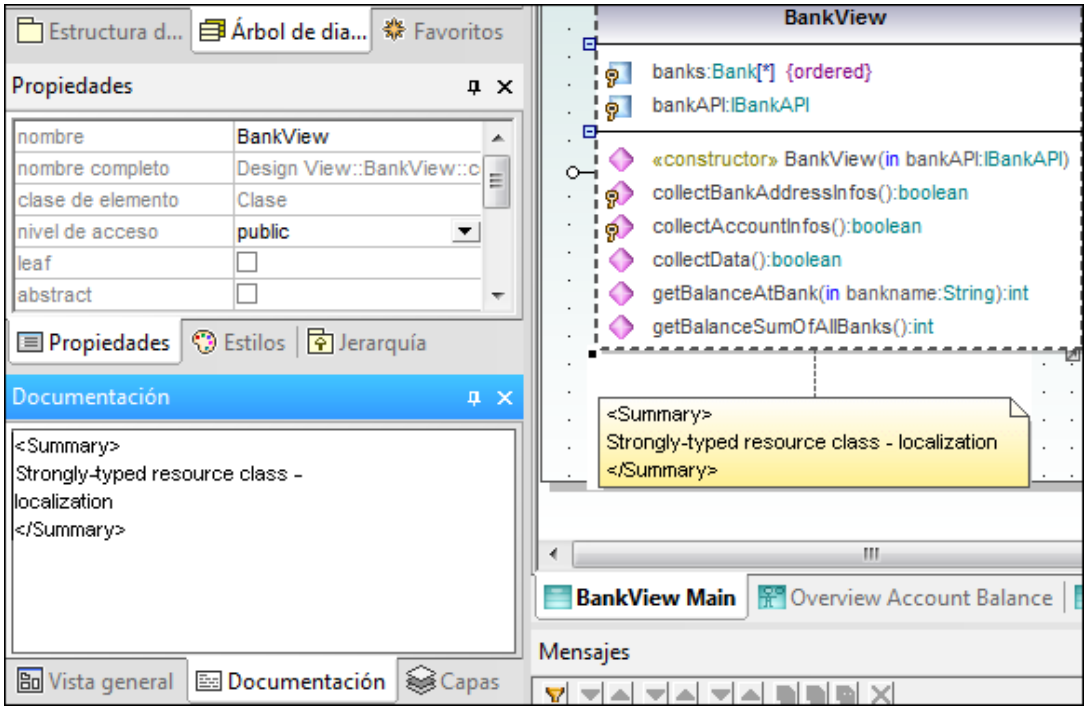


Para ver los comentarios del código fuente en el diagrama:

- Haga clic con el botón secundario en la clase en el panel del diagrama y seleccione **Mostrar | Comentarios de anotación**.

  Esto permite editar los `DocComment` del código fuente desde el diagrama directamente (los `DocComment` también se pueden ver en la pestaña *Documentación*).

 © 2014 Altova GmbH



Para ajustar el grosor (espesor) de las líneas en un proyecto:

1. Haga clic en cualquier diagrama y después en la pestaña *Estilos*.
   Compruebe que en el cuadro combinado superior está activa la opción **`Estilos del proyecto`**.
2. Desplácese hasta la entrada **`Grosor de la línea`** y seleccione un valor.
   Este estilo se aplica a las líneas de asociación, agregación, generalización, etc. de todos los diagramas del proyecto actual.

5.11.1 Propiedades de los diagramas

Para configurar las propiedades de los diagramas:

1. Haga clic en el fondo del área de trabajo.
2. Seleccione uno de los estilos de la pestaña *Estilos*.
   Para más información consulte el apartado *Estilos*.

Para aumentar el tamaño del diagrama:

El tamaño de la pestaña del diagrama depende de los elementos del diagrama y de su posición.

- Arrastre un elemento hasta el borde del diagrama para aumentar el tamaño del diagrama automáticamente.

Para cambiar los elementos de posición con ayuda de la cuadrícula:

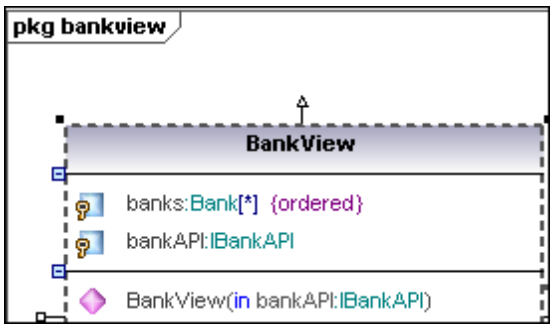
Puede cambiar a mano la posición de los elementos de modelado o puede usar una opción para colocarlos sobre las líneas de la cuadrícula automáticamente.

Este icono de la barra de herramientas sirve para mostrar/ocultar la cuadrícula.

Este icono de la barra de herramientas sirve para ajustar o no los elementos a la cuadrícula visible/invisible automáticamente.

Para ver el título del diagrama UML:

Este comando muestra/oculta el título del diagrama UML, es decir, el marco que rodea el diagrama y que incluye el nombre del diagrama en la esquina superior izquierda.



Para alinear los elementos de modelado:

Puede alinear los elementos y ajustar su tamaño en relación a otros elementos. P. ej. puede centrar todos los elementos o alinearlos a la izquierda o a la derecha.

Nota: cuando marque varios objetos, el comando toma como referencia el **último** elemento que seleccionó. Por ejemplo, si marca tres elementos `clase` y hace clic en el comando **Igualar ancho**, las tres clases cobrarán el ancho de la última clase que seleccionó. Lo mismo ocurre cuando se marcan varios objetos creando un recuadro con el puntero del mouse. El último elemento que se selecciona con el recuadro se toma como referencia para los demás objetos.

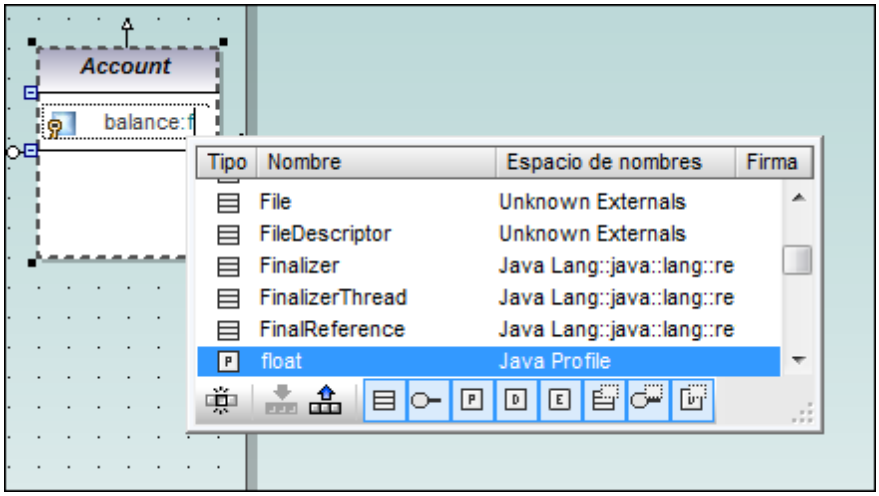
 © 2014 Altova GmbH

Estos son los iconos de la barra de herramientas anterior (de izquierda a derecha):

Alinear a la izquierda
Alinear a la derecha
Alinear arriba
Alinear abajo
Centrar verticalmente
Centrar horizontalmente
Espaciar en horizontal
Espaciar en vertical
Poner en fila horizontal
Poner en fila vertical
Igualar ancho
Igualar alto
Igualar tamaño

Ventana de finalización automática: cómo seleccionar tipos de datos
Cuando necesite insertar tipos de datos para operaciones o propiedades, la ventana de finalización automática se abre automáticamente. Esta ventana ofrece estas características y opciones:

- Para ordenar la ventana en orden ascendente/descendente: haga clic en el nombre de una columna.
- Para ajustar el tamaño de la ventana: arrastre la esquina inferior derecha de la ventana.
- Para filtrar el contenido de la ventana: sírvase de los iconos situados en la parte inferior de la ventana.
- Para trabajar con cada categoría, una por una, active el icono **Modo único** . Desactive el modo único si prefiere trabajar con diferentes categorías a la vez. Por ejemplo, en la imagen siguiente, el modo único no está seleccionado.
- Puede activar/desactivar todas las categorías con los iconos **Activar todas las categorías** y **Desactivar todas las categorías**.



Categorías:
Clase, Interfaz, TipoPrimitivo, TipoDeDatos, Enumeración, Plantilla de clase, Plantilla de interfaz,

Plantilla de tipo de datos.

Nota: la función de finalización automática se puede habilitar/deshabilitar en la pestaña *Edición de diagramas* del cuadro de diálogo "Opciones locales" (**Herramientas | Opciones**), activando/desactivando la casilla *Habilitar ayudante de entrada automática*.

© 2014 Altova GmbH

5.11.2 Cortar, copiar y pegar en los diagramas de UModel

Cortar, copiar y pegar elementos dentro del panel del diagrama
Todos los elementos de los diagramas de UModel se pueden cortar, copiar y pegar dentro de elementos del mismo tipo e incluso de otro. Para ello puede usar también teclas de acceso rápido.

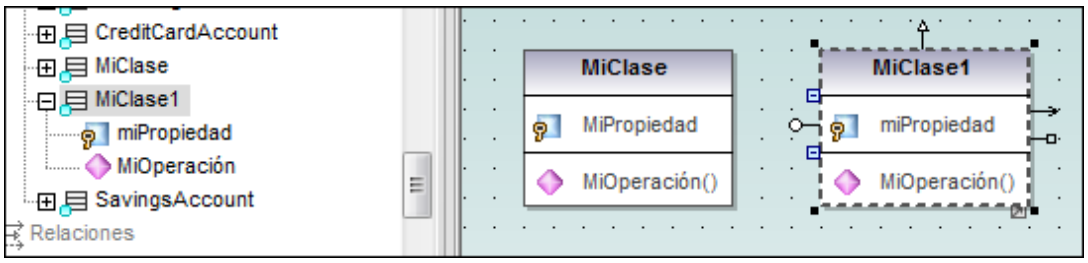
Tras copiar el elemento:

- Puede **pegarlo** usando la combinación de teclas **Ctrl+V**, el comando **Pegar** del menú contextual o el comando **Pegar** del menú **Edición**. Esto añade un **elemento nuevo** al diagrama y a la *Estructura del modelo*.
- Puede **pegarlo sólo en el diagrama** usando el menú contextual (clic con el botón secundario en el fondo del diagrama y después en **Pegar sólo en el diagrama**). Esto añade una vista (o vínculo) del elemento ya existente en el diagrama actual y no en la *Estructura del modelo*.

Por ejemplo, imaginemos que estamos trabajando con un diagrama de clases:

Para pegar (Ctrl+V) una clase copiada previamente:

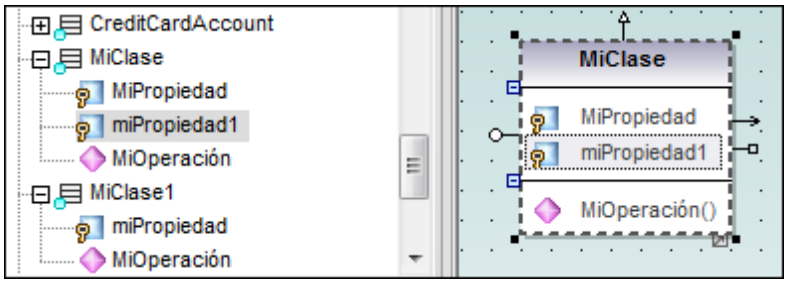
- Si pega la clase copiada **en el mismo diagrama o paquete**, se inserta una **clase nueva** con el nombre de clase original más un número secuencial. Por ejemplo, si el nombre original era `MiClase`, la clase pegada se llamará `MiClase1`. En la clase nueva también se copian todas las operaciones y propiedades de la clase original.



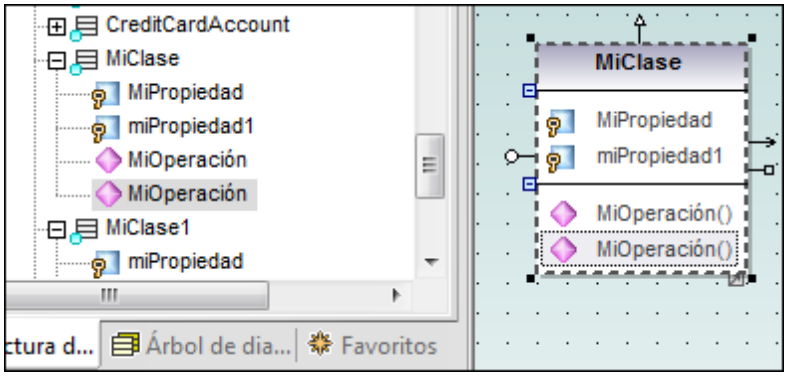
- Si pega la clase copiada **en un paquete distinto**, se inserta una clase nueva que se llama igual que la clase original.
- En ambos casos la clase nueva también se añade a la *Estructura del modelo*.

Para pegar (Ctrl+V) propiedades y operaciones copiadas previamente:

- Si pega la propiedad copiada **en la misma clase**, se inserta una **propiedad nueva** con el nombre de propiedad original más un número secuencial (p. ej. `MiPropiedad1`).



- Si pega la operación copiada **en la misma clase**, se inserta una **operación nueva** que se llama igual que la operación original.

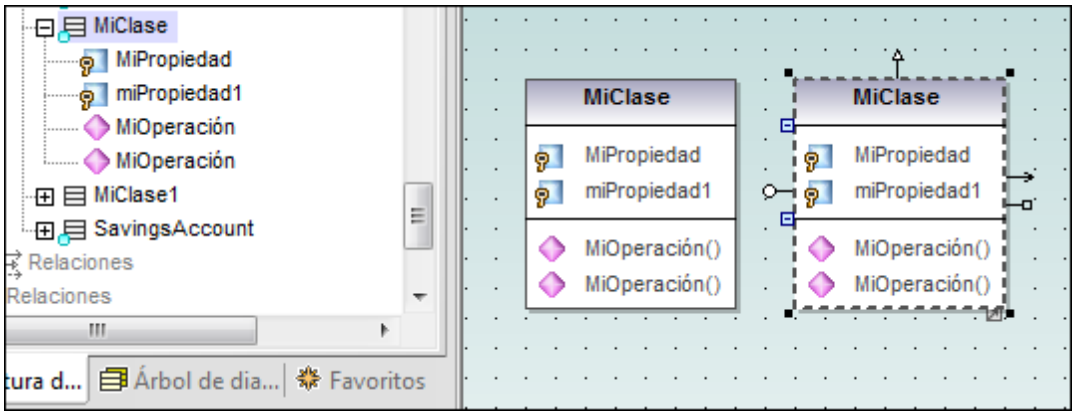


- En ambos casos la operación/propiedad nueva también se añade a la *Estructura del modelo*.

Pegar sólo en el diagrama

Cuando use este comando del menú contextual, se crea un *enlace* o una *vista* del elemento en el **diagrama** donde se ejecuta la operación. Sigamos con el ejemplo de nuestro diagrama de clases:

- La opción **Pegar sólo en el diagrama** del menú contextual crea una vista de la clase original.
- La clase se inserta en el diagrama y aparece exactamente igual que la clase original.
- En la *Estructura de modelo* no se añade ninguna clase nueva.
- El nombre de la clase y de sus propiedades/operaciones no cambia.
- Si cambia las propiedades en una de las *vistas*, las propiedades cambian en la otra *vista* automáticamente.



Para copiar y pegar elementos con el mouse:

1. Haga clic en el elemento de modelado que desea copiar.
2. Arrastre el puntero del mouse hasta la posición donde quiere colocar el nuevo elemento.
3. Pulse la tecla **Ctrl**. Aparece un signo **+** bajo el puntero del mouse. Esto significa que se trata de una operación de copia.
4. Suelte el botón del mouse.

 © 2014 Altova GmbH

Aparece un menú contextual donde puede elegir entre **Pegar** y **Pegar sólo en el diagrama**.

5. Elija una opción.

Nota: utilice este método (botón del mouse + tecla **Ctrl**) para copiar o mover propiedades y operaciones dentro de una clase.

5.12 Agregar/insertar elementos en los modelos

En UModel los elementos se pueden crear e insertar en los diagramas de varias maneras:

- Añadiendo elementos a determinados paquetes, desde la pestaña *Estructura del modelo.*
- Arrastrando elementos ya existentes desde la pestaña *Estructura del modelo.*
- Haciendo clic en un icono determinado e insertando el tipo de elemento en el diagrama.
- Usando el menú contextual para agregar elementos al diagrama (y a la *Estructura del modelo* automáticamente).

  Recuerde que en la *Estructura del modelo* puede seleccionar varios elementos con las teclas **Mayús+Clic** o **Ctrl+Clic**.

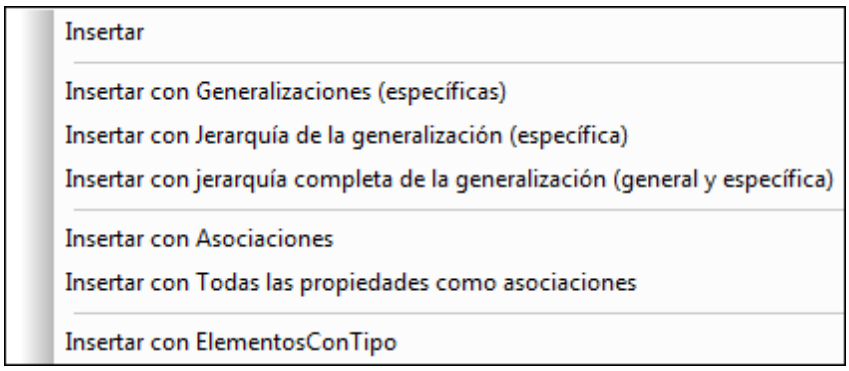
Para agregar elementos en la pestaña *Estructura del modelo* y *Favoritos*:

- Haga clic con el botón secundario en el paquete, seleccione **Nuevo/a** y elija un elemento del submenú.
  El elemento nuevo se añade en la pestaña *Estructura del modelo* en el proyecto actual.

Para insertar elementos de la vista *Estructura del modelo* en un diagrama:

Puede insertar elementos de modelado por separado o en grupo. Para marcar varios elementos haga clic en los elementos mientras pulsa la tecla **Ctrl**. Hay dos maneras de insertar los elementos en el diagrama: arrastrándolos mientras pulsa el botón principal del mouse y arrastrándolos mientras pulsa el botón secundario.

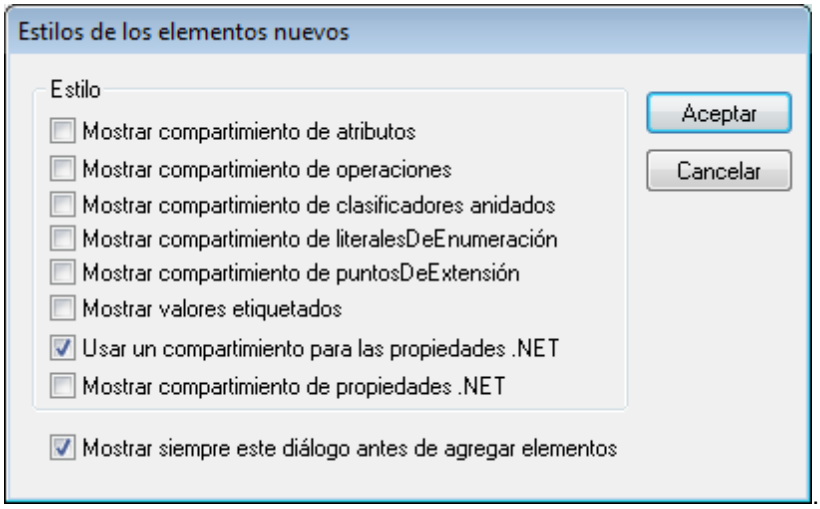
- **Arrastre el elemento mientras pulsa el botón principal** (operación arrastrar y colocar normal) para insertar los elementos inmediatamente en la posición del cursor (las asociaciones, dependencias, etc. que existan entre los elementos insertados y el nuevo se muestran automáticamente).
- **Arrastre el elemento mientras pulsa el botón secundario** para abrir un menú contextual donde puede elegir qué asociaciones y generalizaciones se muestran.



En nuestro ejemplo vamos a intentar crear un duplicado del diagrama `Account Hierarchy`.

1. Haga clic con el botón secundario en el paquete `bankview` y seleccione **Nuevo diagrama | Diagrama de clases**.
2. Busque la clase abstracta `Account` en la *Estructura del modelo* y arrástrela **mientras pulsa el botón secundario** hasta el nuevo diagrama.
   Aparece el menú contextual de la imagen anterior.
3. Seleccione la opción **Insertar con jerarquía de la generalización (específica)**.

 © 2014 Altova GmbH



4. Desactive las casillas de los componentes que deben aparecer en los elementos (en este caso, las propiedades y las operaciones).
5. Haga clic en **Aceptar**.
   La clase `Account` y sus tres clases subordinadas se insertan en la pestaña del diagrama. Las flechas de `Generalización` se muestran automáticamente.

Para agregar elementos a un diagrama con los iconos de la barra de herramientas:

1. Seleccione el icono del elemento que desea insertar en la barra de herramientas.
2. Haga clic en el área de trabajo para insertar el elemento.

Nota: pulse la tecla **Ctrl** mientras hace clic en la pestaña del diagrama para insertar varios elementos del mismo tiempo con cada clic.

Para agregar elementos a un diagrama con el menú contextual:

- Haga clic con el botón secundario en el fondo del diagrama y seleccione la opción **Nuevo/a | <nombre de elemento>**.

Nota: si añade elementos nuevos en el área de trabajo directamente, el mismo elemento se añade automáticamente en la pestaña *Estructura del modelo*. El elemento se añade al paquete que contiene el diagrama UML en la *Estructura del modelo*.

- Haga clic con el botón secundario en un elemento y seleccione **Mostrar | xxxx**
  Por, ejemplo, haga clic con el botón secundario en la clase `Account` y seleccione **Mostrar | Jerarquía de la generalización**. Esto también inserta las clases derivadas en el diagrama.

5.13 Crear hipervínculos entre los elementos de modelado

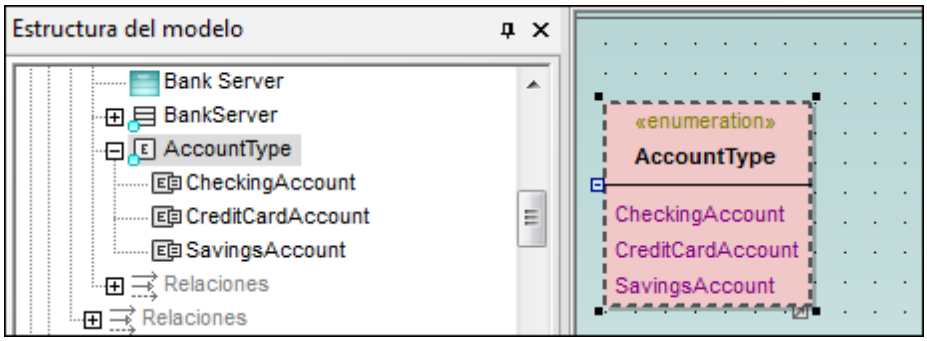
UModel ofrece opciones para crear hipervínculos entre elementos de modelado, tanto automática como manualmente. Los hipervínculos automáticos se crean al importar código fuente o archivos binarios en el modelo.

Los hipervínculos manuales se pueden crear entre casi todos los elementos de modelado (excepto entre las líneas) y:

- cualquier diagrama del proyecto `*.ump` actual
- cualquier elemento de un diagrama
- cualquier elemento de la Estructura del modelo
- documentos externos (p. ej. documentos PDF, Excel o Word)
- páginas web

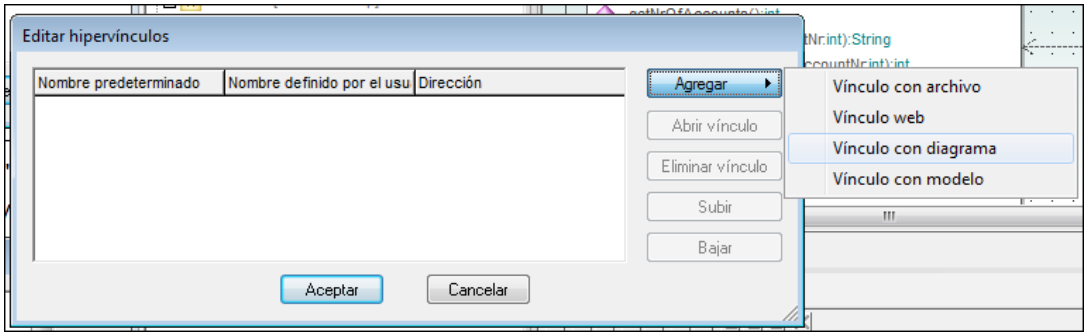
Note: todos los hipervínculos creados manualmente también están disponibles en la documentación HTML del proyecto cuando se genera documentación HTML.

Al abrir el diagrama **`Bank Server`** situado bajo el paquete **`Bank Server`** se puede ver la interfaz **`IBankAPI`**, así como la clase **`BankServer`**. También se puede ver un elemento **`enumeration`** con los nombres de los **`LiteralesDeEnumeración`**. Nuestro objetivo es crear un hipervínculo entre el elemento **`enumeration`** y el diagrama de clases **`Account Hierarchy`**.



Para crear un hipervínculo de diagrama:

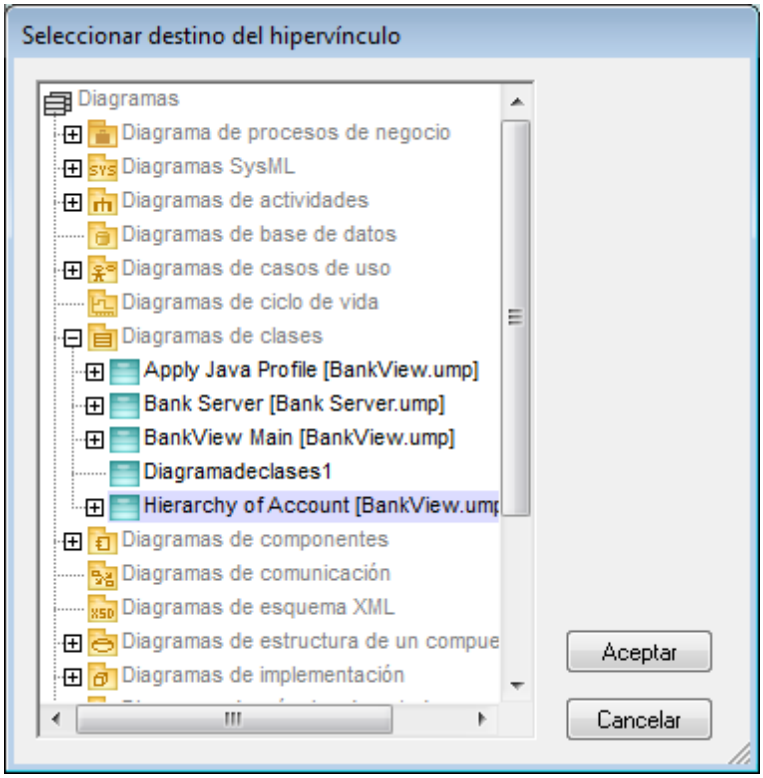
1. Haga clic con el botón secundario en el elemento y seleccione **Hipervínculos | Insertar o editar hipervínculos**.



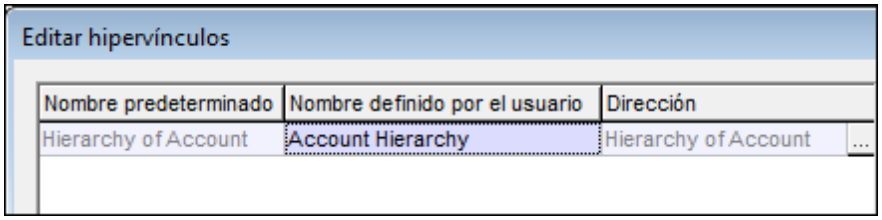
Esto abre el cuadro de diálogo "Editar hipervínculos", que sirve para gestionar los hipervínculos.

 © 2014 Altova GmbH

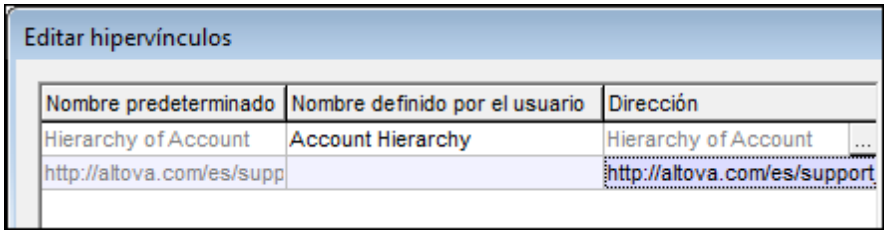
2. Haga clic en el botón **Agregar** y seleccione **Vínculo con diagrama** para definir un vínculo con un diagrama ya existente.



3. Seleccione el destino del hipervínculo (p. ej. el diagrama `Hierarchy of Account`) y haga clic en **Aceptar**.

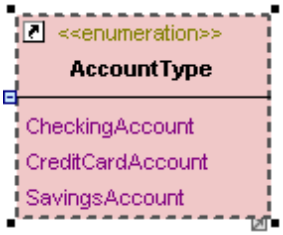


Haga doble clic en la columna *Nombre definido por el usuario* para definir el nombre de vínculo que desea usar.
Recuerde que puede agregar varios vínculos, incluso de tipos diferentes, en el mismo elemento de modelado (p. ej. también puede añadir un vínculo web a http://altova.com/es/support_help.html haciendo clic en **Agregar | Vínculo web**).

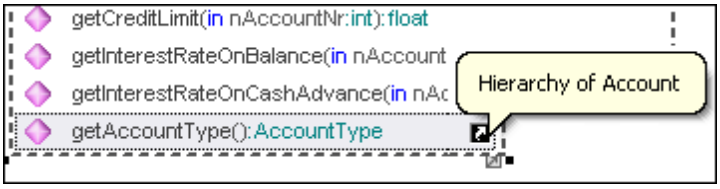


4. Cuando termine de definir los hipervínculos haga clic en **Aceptar**.
Observe que en la esquina superior izquierda del elemento `enumeration` se añadió un icono en forma de flecha. Pase el cursor por encima del icono para ver el nombre del

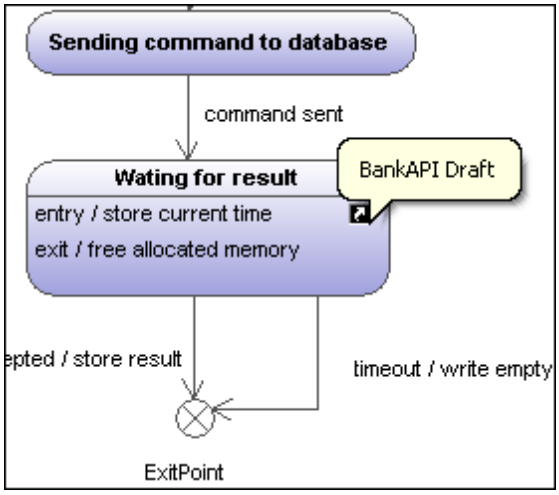
elemento de destino.



Hipervínculos en operaciones



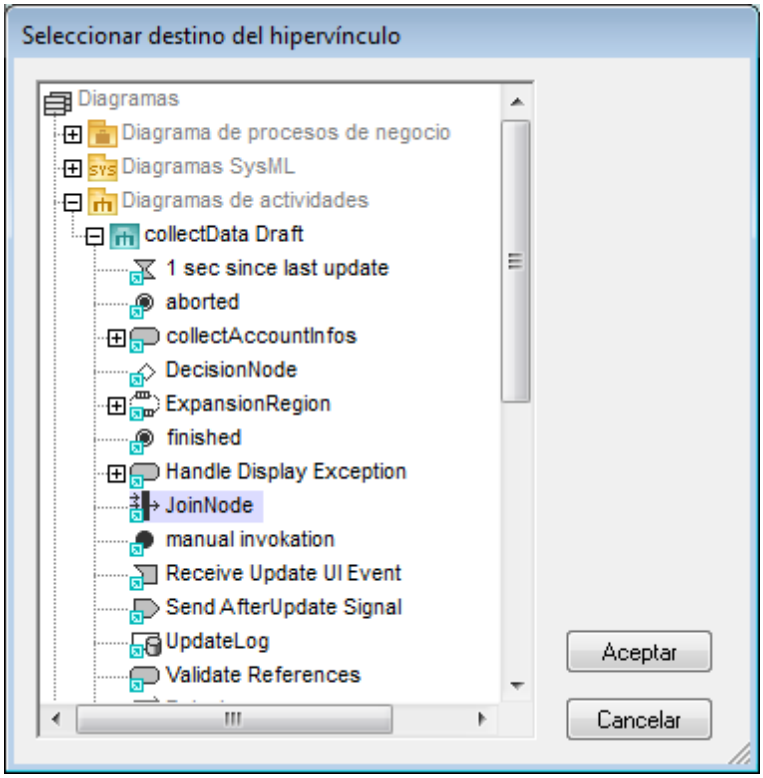
Hipervínculo desde una máquina de estados



Para crear un vínculo a un elemento concreto de un diagrama (vínculo con diagrama):

1. Cree el hipervínculo igual que antes pero esta vez haga clic en el signo **+** para expandir el contenido del diagrama.

 © 2014 Altova GmbH



2. Seleccione el elemento de modelado al que debe apuntar el vínculo y haga clic en **Aceptar**.

   Al hacer clic en el icono del hipervínculo se abre el diagrama y observará que en él está seleccionado el elemento designado.

Para crear un vínculo a un elemento en la Estructura del modelo (vínculo con modelo):

- Haga lo mismo que en el ejemplo anterior pero esta vez haga clic en **Agregar | Vínculo con modelo** en el cuadro de diálogo.

  Una vez insertado, el hipervínculo conduce al elemento seleccionado en la *Estructura del modelo* desde el cuadro de diálogo.

  Nota: cuando genere documentación del proyecto, los hipervínculos a los elementos de la Estructura del modelo se ajustan para que apunten a las definiciones correctas en la documentación generada.

Para crear un vínculo a un documento:

1. En el cuadro de diálogo "Editar hipervínculos" haga clic en el botón **Agregar | Vínculo con archivo**.
2. Seleccione el documento al que debe apuntar el vínculo (p. ej. `*.doc`, `*.xls`, `*.pdf`, etc.).

Para crear un hipervínculo desde una nota:

1. Seleccione el texto de la nota que debe servir de hipervínculo.
2. Haga clic con el botón secundario en el texto seleccionado y seleccione la opción

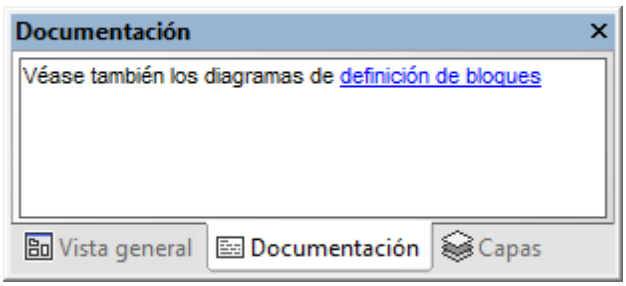
Insertar o editar hipervínculos.

3. En el cuadro de diálogo "Editar hipervínculos" cree el vínculo a un diagrama.

Nota: puede usar el mismo método para crear hipervínculos desde elementos `Comentario`.

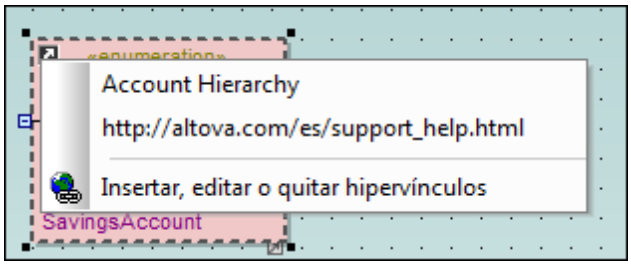
Para crear un vínculo desde la pestaña *Documentación*:

1. Escriba el texto descriptivo en la pestaña *Documentación*.
2. Seleccione el texto que debe servir de hipervínculo.
3. Haga clic con el botón secundario y seleccione qué tipo de vínculo desea crear.



Para navegar hasta el destino del hipervínculo:

1. Haga clic en el icono del hipervínculo del elemento de modelado.
   Si solo definió un destino, el diagrama o la web de destino, por ejemplo, aparecerá inmediatamente.
   Si definió varios destinos, aparece un menú contextual donde puede seleccionar uno de ellos.



   Por ejemplo, en el ejemplo de la imagen anterior la primera opción abre el diagrama `Hierarchy of Account`.

Para navegar por los hipervínculos:

- Haga clic en los iconos **Anterior** y **Siguiente** de la barra de herramientas principal para navegar por el origen y el destino de los vínculos.

Para editar/cambiar el destino de un hipervínculo:

1. Haga clic con el botón secundario en el icono del vínculo y seleccione la opción **Insertar, editar o quitar hipervínculos**.

 © 2014 Altova GmbH

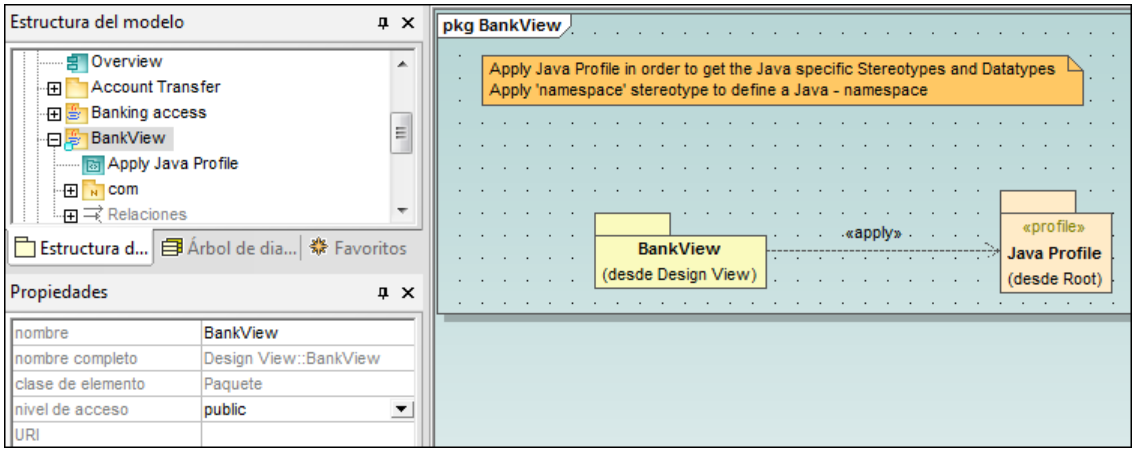
2. Realice los cambios necesarios en el cuadro de diálogo "Editar hipervínculos".

5.14 Ejemplos Bank

La carpeta ...**UModelExamples** contiene archivos de ejemplo que muestran diferentes aspectos del modelado UML en UModel. Estos archivos están diseñados para ilustrar modelos propios de los lenguajes Java y C# y de una mezcla de ambos.

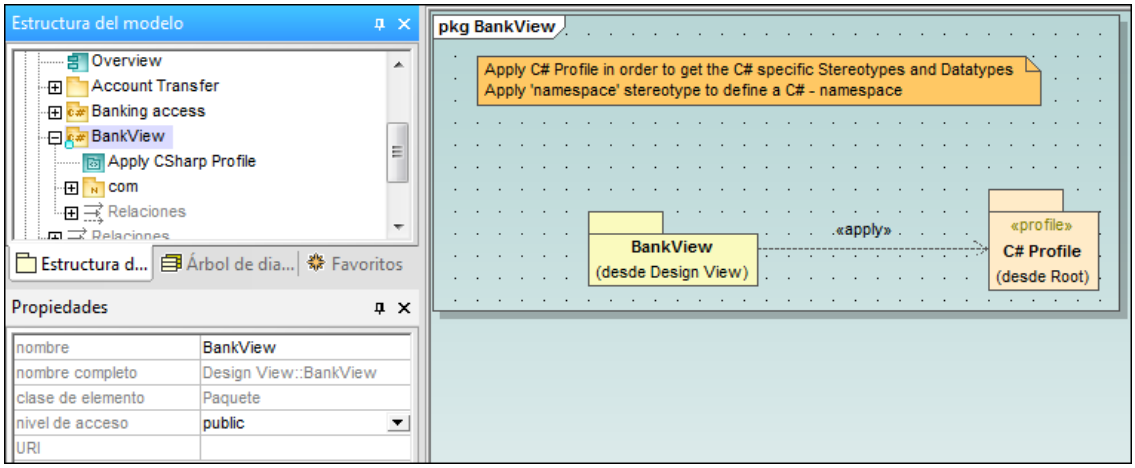
El archivo de ejemplo **Bank_Java.ump** (*imagen siguiente*) tiene estas características:

- El perfil `Java` está asignado al paquete `Bankview`.
- La raíz de espacio de nombres Java está asignada a los paquetes `Banking access` y `BankView`.
- El paquete `Interaction View` contiene dos elementos de interacción, con sendos diagramas de secuencia.



El archivo de ejemplo **Bank_CSharp.ump** (*imagen siguiente*) tiene estas características:

- El perfil C# está asignado al paquete `BankView`.
- La raíz de espacio de nombres C# está asignada a los paquetes `Banking access` y `BankView`.
- El paquete `Interaction View` contiene dos elementos de interacción, con sendos diagramas de secuencia.

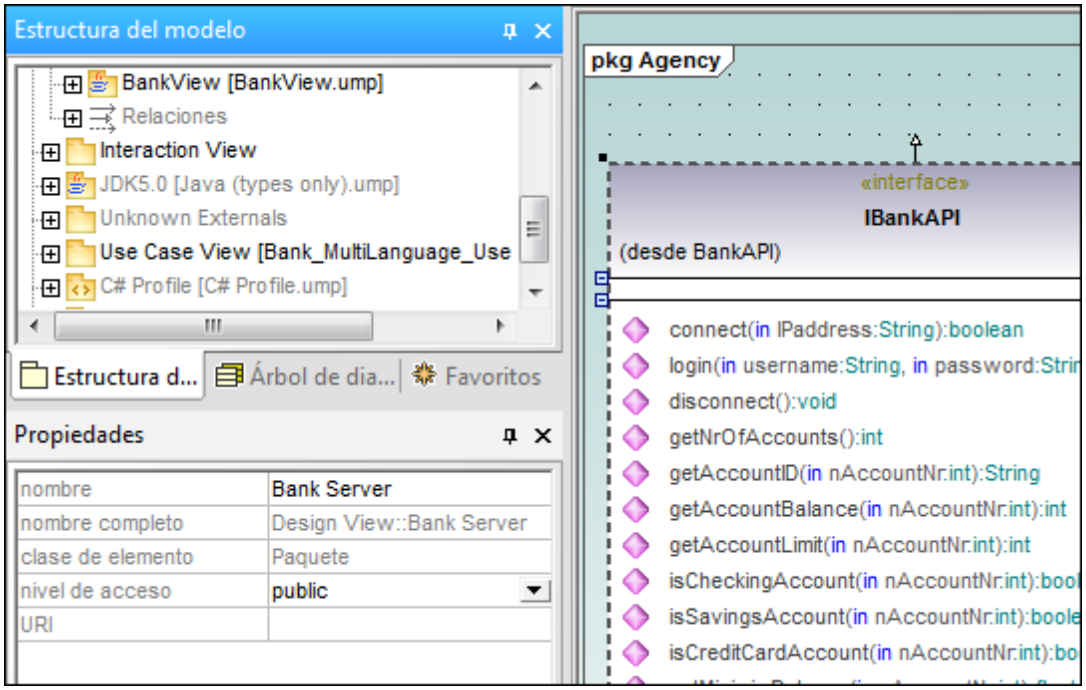


El archivo de ejemplo **Bank_MultiLanguage.ump** (*imagen siguiente*) tiene estas características:

- El perfil Java está asignado al paquete `BankView`.

 © 2014 Altova GmbH

- La raíz de espacio de nombres C# está asignada al paquete **`Bank Server`**.
- La raíz de espacio de nombres Java está asignada al paquete **`BankView`**.
- El paquete **`Interaction View`** contiene dos elementos de interacción, con sendos diagramas de secuencia.
- El proyecto está dividido en 4 subproyectos editables: **`Bank Server.ump`**, **`Banking access.ump`**, **`BankView.ump`** y **`Bank_Multilanguage_Use Case View.ump`**.



Altova UModel® 2015

Interfaz de la línea de comandos

6 Interfaz de la línea de comandos

UModel es compatible con tareas de procesamiento por lotes y en la carpeta ...**UModel2015** encontrará el archivo `UModelBatch.exe`.

En este apartado puede consultar la sintaxis de parámetros de la línea de comandos. Para ver la sintaxis en una ventana del símbolo del sistema, escriba `umodelbatch /?` en la línea de comandos.

Nota: si contiene espacios, la ruta de acceso o el nombre de archivo debe ir entre comillas (p. ej. `"c:\Archivos de programa\...\Nombre del archivo"`)

```
utilización: umodelbatch [proyecto] [opciones]

/? o /help ... mostrar esta información de ayuda

proyecto         ... archivo de proyecto (*.ump). Ver también Archivo:
Nuevo / Cargar / Opciones al guardar
/new[=archivo]   ... crear, guardar o guardar como proyecto nuevo
/set             ... establecer operaciones permanentes
/gui             ... mostrar la interfaz de usuario de UModel


comandos (ejecutados en este orden):
/chk             ... revisar la sintaxis del proyecto
/isd=ruta        ... importar directorio de código fuente
/isp=archivo     ... importar archivo de proyecto de código fuente
                (*.project,*.xml,*.jpx,*.csproj,*.csdproj,*.vbproj,*.vbdproj,
*.sln,*.bdsproj)
/ibt=lista       ... importar tipos binarios (especificar lista de [nombres
de tipos] binarios)
                (";"=separador, "*"=todos los tipos, "#" antes de los nombres
de ensamblado)
/ixd=ruta        ... importar directorio del esquema XML
/ixs=archivo     ... importar archivo de esquema XML (*.xsd)
/m2c             ... actualizar código de programa con el modelo (exportar/
ingeniería directa)
/c2m             ... actualizar modelo con el código de programa (importar/
ingeniería inversa)
/ixf=archivo     ... importar archivo XMI
/exf=archivo     ... exportar a archivo XMI
/inc=archivo     ... incluir archivo
/mrg=archivo     ... combinar archivo
/doc=archivo     ... escribir documentación en el archivo especificado
/lue[=cpri]      ... mostrar una lista de los elementos que no se utilizan en
ningún diagrama (es decir, no utilizados)
/ldg             ... mostrar una lista de todos los diagramas
/lcl             ... mostrar una lista de todas las clases
/lsp             ... mostrar una lista de todos los paquetes compartidos
/lip             ... mostrar una lista de todos los paquetes incluidos


opciones para guardar como proyecto nuevo:
```

 © 2014 Altova GmbH

```
/npad=opc   ... ajustar rutas de acceso relativas (Yes | No | MakeAbsolute),
es decir Sí, No o Convertir en absolutas


opciones para comandos de importación:
/iclg=leng  ... lenguaje de código (Java1.4 | Java5.0 | Java6.0 | C#1.2 |
C#2.0 | C#3.0 | C#4.0 | VB7.1 | VB8.0 | VB9.0)
/ipsd[=0|1] ... procesar subdirectorios (recursivo)
/irpf[=0|1] ... importar relativos al archivo de proyecto de UModel
/ijdc[=0|1] ... JavaDocs como comentarios de Java
/icdc[=0|1] ... DocComments como comentarios de C#
/icds[=lst] ... Símbolos definidos de C#
/ivdc[=0|1] ... DocComments como comentarios de VB
/ivds[=lst] ... Símbolos definidos de VB (constantes personalizadas)
/imrg[=0|1] ... sincronizar combinados
/iudf[=0|1] ... utilizar filtro de directorios
/iflt[=lst] ... filtro de directorios (restablece /iudf)


opciones para importar tipos binarios (después de /iclg):
/ibrt=vers  ... versión en tiempo de ejecución
/ibpv=path  ... reemplazo de la variable PATH para buscar bibliotecas de
código nativo
/ibro[=0|1] ... usar el contexto de sólo reflexión
/ibua[=0|1] ... usar la opción de agregar tipos referenciados con filtro de
paquetes
/ibar[=flt] ... agregar el filtro de paquetes de tipos referenciados
(restablece /ibua)
/ibot[=0|1] ... sólo importar los tipos
/ibuv[=0|1] ... utilizar el filtro de nivel de acceso mínimo
/ibmv[=key] ... palabra clave para el nivel de acceso mínimo necesario
(restablece /ibuv)
/ibsa[=0|1] ... omitir secciones de atributo o modificadores de anotación
/iboa[=0|1] ... crear un solo atributo por sección de atributos
/ibss[=0|1] ... omitir el sufijo "Attribute" en los nombres de tipo de
atributo


opciones para la generación de diagramas:
/dgen[=0|1] ... generar diagramas
/dopn[=0|1] ... abrir diagramas generados
/dsac[=0|1] ... mostrar compartimiento de atributos
/dsoc[=0|1] ... mostrar compartimiento de operaciones
/dscc[=0|1] ... mostrar compartimiento de clasificadores anidados
/dstv[=0|1] ... mostrar valores etiquetados
/dudp[=0|1] ... usar compartimiento de la propiedad .NET
/dspd[=0|1] ... mostrar compartimiento de la propiedad .NET


opciones par comandos de exportación:
/ejdc[=0|1] ... comentarios de Java como JavaDocs
/ecdc[=0|1] ... comentarios de C# como DocComments
/evdc[=0|1] ... comentarios de VB como DocComments
/espl[=0|1] ... utilizar plantillas SPL definidas por el usuario
/ecod[=0|1] ... convertir código eliminado en comentario
/emrg[=0|1] ... sincronizar combinados
/egfn[=0|1] ... generar los nombres de archivo que falten
```

```
/eusc[=0|1] ... usar revisión de sintaxis


opciones para la exportación de XMI:
/exid[=0|1] ... exportar identificadores UUID
/exex[=0|1] ... exportar extensiones específicas de UModel
/exdg[=0|1] ... exportar diagramas (restablece /exex)
/exuv[=ver] ... versión de UML (UML2.0 | UML2.1.2 | UML2.2 | UML2.3)


opciones para combinar archivos:
/mcan=archivo  ... archivo antecesor común


opciones para la generación de documentación:
/doof=fmt   ... formato de salida (HTML | RTF | MSWORD | PDF)
/dsps=archivo  ... archivo de diseño SPS
```

En la sección de proyectos:

- el parámetro **/new** define la ruta de acceso y el nombre de archivo de proyecto nuevo (*.ump). También se puede usar para guardar un proyecto con otro nombre distinto (p. ej. `UModelBatch.exe MiArchivo.ump /new=MiArchivoBackup.ump`). Para más información consulte el apartado Archivo: Nuevo / Cargar / Opciones al guardar.
- el parámetro **/set** reemplaza las opciones de configuración predeterminadas del registro con las opciones definidas aquí.
- el parámetro **/gui** muestra la interfaz de UModel durante el procesamiento por lotes.

Ejemplo 1:
Importar código fuente y crear un archivo de proyecto nuevo:

```
"C:\Archivos de programa\Altova\UModel2015\UModelBatch.exe" /new="C:\Archivos de
programa\Altova\UModel2015\UModelBatchOut\Fred.ump" /isd="X:TestCases\UModel
\Fred" /set /gui /iclg=Java5.0 /ipsd=1 /ijdc=1 /dgen=1 /dopn=1 /chk
```

/new:	indica que el archivo de proyecto nuevo debe llamarse "Fred.ump" y se debe guardar en `C:\Archivos de programa\Altova\UModel2015\UModelBatchOut\`
/isd=	indica que el directorio raíz en el que se debe importar es `X:\TestCases\UModel\Fred`
/set:	indica que las opciones utilizadas en la línea de comandos se guardarán en el registro (y estas opciones formarán la configuración predeterminada la próxima vez que se abra UModel).
/gui:	mostrar la interfaz gráfica de UModel durante el procesamiento por lotes
/iclg:	UModel importará el código como Java 5.0
/ipsd=1:	procesar recursivamente todos los subdirectorios del directorio raíz suministrado con el parámetro **/isd**

 © 2014 Altova GmbH

/**pfd**=1:	crea un paquete por cada directorio importado en el proyecto de UModel
/**ijdc**=1:	crea JavaDocs donde corresponda a partir de los comentarios
/**dgen**=1:	genera diagramas
/**dopn**=1:	abre los diagramas generados
/**chk**:	revisa la sintaxis

Ejemplo 2:
Importar código fuente desde `X:\TestCases\UModel` y guardar el archivo de proyecto resultante en `"C:\Archivos de programa..."`.

```
"C:\Archivos de programa\Altova\UModel2015\UModelBatch.exe" /new="C:\Archivos de
programa\Altova\UModel2015\UModelBatchOut\finalclass.ump" /isd="X:\TestCases
\UModel\" /iclg=Java5.0 /ipsd=1 /ijdc=1 /dgen=1 /dopn=1 /dsat=1 /dsnc=1 /chk
```

/**dsat**=1	suprime los atributos en los diagramas generados
/**dsnc**=1	suprime los clasificadores anidados en los diagramas generados

Ejemplo 3:
Sincronizar código usando un archivo de proyecto existente (p. ej. uno de los creados en los ejemplos anteriores).

```
"C:\Archivos de programa\Altova\UModel2015\UModelBatch.exe" "C:\Archivos de
programa\Altova\UModel2015\UModelBatchOut\Fred.ump" /m2c /ejdc=1 /ecod=1 /emrg=1
/egfn=1 /eusc=1
```

El archivo de proyecto que queremos usar es `"C:\Archivos de programa\Altova\UModel2015\UModelBatchOut\Fred.ump"`.

/**m2c**	actualizar el código con el modelo
/**ejdc**:	generar JavaDoc a partir de los comentarios del modelo del proyecto
/**ecod**=1	convertir el código eliminado en comentarios
/**emrg**=1	sincronizar el código combinado
/**egfn**=1	generar en el proyecto los nombres de archivo que falten
/**eusc**=1	usar la revisión de sintaxis

6.1 Archivo: Nuevo / Cargar / Opciones al guardar

Modo por lotes total (es decir, sin usar el parámetro **/gui**).

new

UModelBatch /new=xxx.ump (opciones)
crea un proyecto nuevo, ejecuta las opciones y xxx.ump se guarda **siempre** (independientemente de las opciones utilizadas)

auto save

UModelBatch xxx.ump (opciones)
carga el proyecto xxx.ump, ejecuta las opciones y xxx.ump **solo** se guarda si hubo cambios en el documento (como **/ibt**)

save

UModelBatch xxx.ump (opciones) /new
carga el proyecto xxx.ump, ejecuta las opciones y xxx.ump se guarda **siempre** (independientemente de las opciones utilizadas)

save as

UModelBatch xxx.ump (opciones) /new=yyy.ump
carga el proyecto xxx.ump, ejecuta las opciones y xxx.ump se guarda siempre como yyy.ump (independientemente de las opciones utilizadas)

Modo por lotes con la interfaz de UModel (es decir, usando el parámetro **/gui**).

new

UModelBatch /gui /new (opciones)
crea un proyecto nuevo, ejecuta las opciones y no se guarda nada. La interfaz gráfica permanece abierta.

save new

UModelBatch /gui /new=xxx.ump (opciones)
crea un proyecto nuevo, ejecuta las opciones y se guarda el archivo xxx.ump. La interfaz gráfica permanece abierta.

user mode

UModelBatch /gui xxx.ump (opciones)
carga el proyecto xxx.ump, ejecuta las opciones y no se guarda nada. La interfaz gráfica permanece abierta.

save

UModelBatch /gui xxx.ump (opciones) /new
carga el proyecto xxx.ump, ejecuta las opciones y se guarda el archivo xxx.ump. La interfaz gráfica permanece abierta.

save as

UModelBatch /gui xxx.ump (opciones) /new=yyy.ump
carga el proyecto xxx.ump, ejecuta las opciones y el archivo xxx.ump se guarda como yyy.ump. La interfaz gráfica permanece abierta.

Nota: si durante la ejecución se producen errores, no se guardará nada.

 © 2014 Altova GmbH

Altova UModel® 2015

Proyectos e ingeniería de código

7 Proyectos e ingeniería de código

UModel admite todas las construcciones propias de Java, como por ejemplo:

- anotaciones Java
- atributos, operaciones y calificadores anidados para LiteralesDeEnumeración
- las Enumeraciones pueden realizar interfaces
- archivos de proyecto Netbeans

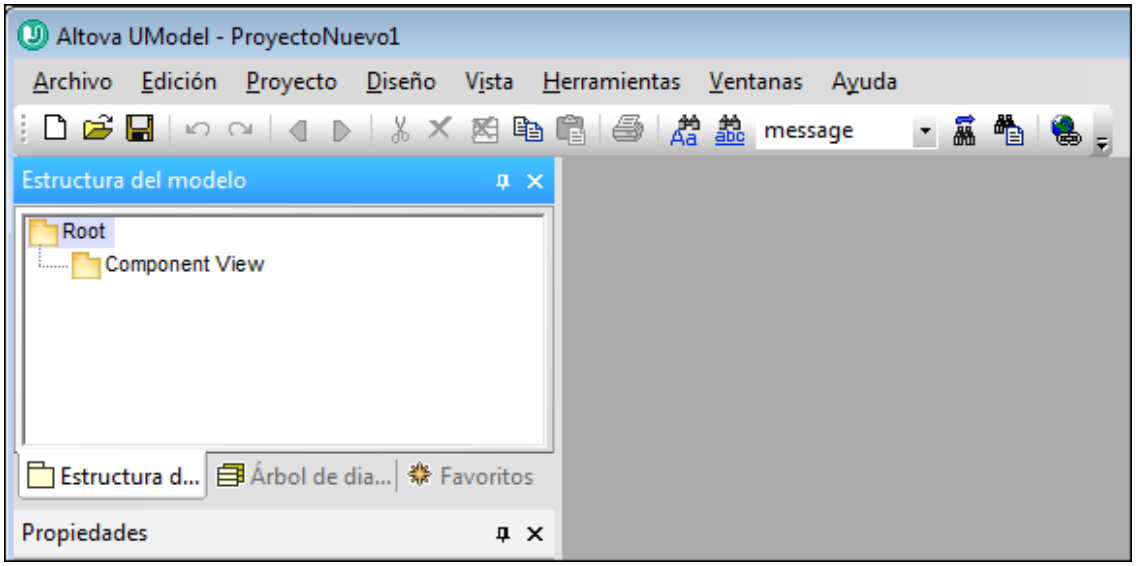
La función de ingeniería inversa ofrece:

- la posibilidad de generar un solo diagrama para todos los elementos creados con ingeniería inversa.
- la posibilidad de mostrar/ocultar elementos enlazados anónimos en diagramas.
- la posibilidad de crear hipervínculos automáticamente desde los paquetes hasta los correspondientes diagramas de contenido del paquete durante el proceso de importación.
- la posibilidad de resolver alias.
- la posibilidad de escribir caracteres Unicode en archivos de código fuente nuevos.
- la posibilidad de crear asociaciones a partir de propiedades .NET.

Para crear un proyecto nuevo:

1. Haga clic en el icono **Nuevo** de la barra de herramientas (o seleccione el comando **Archivo | Nuevo**).

Los paquetes `Root` y `Component` se insertan automáticamente cuando se crea un proyecto nuevo y están visibles en el panel Estructura del modelo. El proyecto nuevo se crea con el nombre predeterminado `ProyectoNuevo1`. Recuerde que UModel se abre con un proyecto nuevo automáticamente.



Los proyectos de UModel nuevos están formados por estos dos paquetes:

- `Root`
- `Component View`

Estos dos paquetes son los únicos paquetes que no se pueden renombrar ni eliminar.

 © 2014 Altova GmbH

Todos los datos sobre el proyecto se almacenan en el archivo de proyecto de UModel, que tiene la extensión `*.ump`. En la Estructura del modelo cada icono en forma de carpeta representa un paquete UML.

Proceso de trabajo con proyectos de UModel:

En primer lugar, UModel no obliga a seguir ninguna secuencia de modelado predeterminada.

Puede añadir al proyecto cualquier tipo de elemento de modelado (diagramas UML, paquetes, actores, etc.) en el orden que quiera y en la posición que quiera. No olvide que puede insertar, renombrar y eliminar todos los elementos de modelado en el panel Estructura de modelado directamente. Es decir, no hace falta crearlos dentro del diagrama.

Para insertar un paquete nuevo:

1. Haga clic con el botón secundario en el paquete donde desea que aparezca el paquete nuevo (en `Root` o en `Component View`).
2. Seleccione **Elemento nuevo | Paquete** en el menú contextual.
   Se crea un paquete nuevo dentro del paquete elegido. El campo del nombre del paquete se resalta para que pueda escribir inmediatamente el nombre del paquete.

- Los paquetes son contenedores para todos los demás elementos de modelado UML (diagramas, clases, instancias, etc.).
- Los paquetes se pueden **crear** en cualquier posición de la Estructura del modelo.
- Los paquetes (el contenido) se pueden **mover/copiar** en otros paquetes de la Estructura del modelo (y en diagramas válidos que estén abiertos en el área de trabajo).
- Los paquetes y su contenido se pueden **ordenar** (en la Estructura del modelo) en función de varios criterios.
- Los paquetes se pueden **colocar** dentro de otros paquetes.
- Los paquetes se pueden usar como elementos de **origen** o **destino** cuando se combina o sincroniza el código.

Para que los elementos aparezcan en un diagrama UML:

1. Inserte un diagrama UML nuevo, haciendo clic con el botón secundario y seleccionando **Diagrama nuevo | Diagrama de clases**.
2. Arrastre un elemento de modelado desde la Estructura del modelo hasta el diagrama que acaba de crear.
3. Utilice el menú contextual del área de trabajo del diagrama para añadir elementos nuevos en el diagrama.

Para guardar un proyecto:

Seleccione la opción de menú **Archivo | Guardar como...** (o **Archivo | Guardar**).

Nota: en la pestaña *Archivo* del cuadro de diálogo "Opciones locales" (**Herramientas | Opciones**) puede configurar la aplicación para que los archivos `*.ump` se guarden en formato pretty-print.

Para abrir un proyecto:

Seleccione la opción de menú **Archivo | Abrir** o seleccione un archivo de la lista de archivos.

Nota: los cambios realizados en el archivo de proyecto o en sus archivos incluidos desde una aplicación externa se registran automáticamente y generan un aviso. Puede elegir si el proyecto se carga otra vez o no.

Para mover un proyecto:

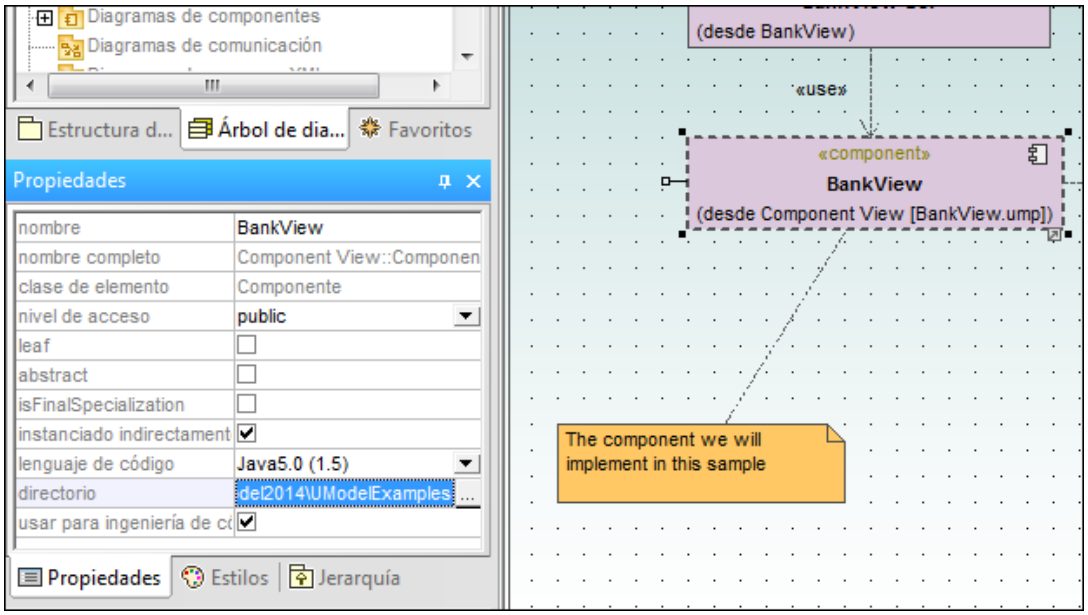
Los proyectos de UModel y el código generado se pueden mover a otro directorio (o a otro equipo) y volver a sincronizarse en el directorio (o equipo) de destino.

Esto se puede hacer de dos maneras:

- Con el comando **Archivo | Guardar como...** y haciendo clic en **Sí** cuando la aplicación solicita adaptar las rutas de acceso del archivo a la nueva ubicación del proyecto.
- Copiando el proyecto de UModel (`*.ump`) a una ubicación nueva y adaptando las rutas de acceso de los paquetes de generación de código de **`Component View`** a la ubicación nueva.

Por ejemplo, tomemos el archivo `BankMultilanguage.ump`:

1. Abra el diagrama de componentes **`Overview`** y del paquete **`Design View`**. Ahora seleccione el componente **`BankView`** del diagrama.
2. En el panel Propiedades actualice la ruta de acceso del campo `directorio` con la nueva ubicación del proyecto.
3. Sincronice el modelo y el código.



 © 2014 Altova GmbH

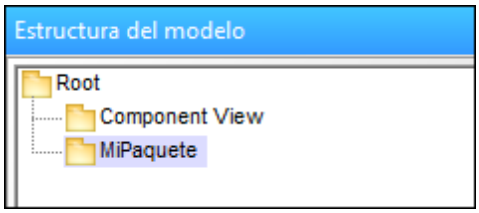
7.1 Crear un proyecto de UModel desde cero

Este apartado describe los pasos necesarios para crear un proyecto desde cero y generar código para una sola clase. Aunque es un proyecto muy básico, utiliza varios diagramas para explicar cómo agregar métodos, etc. Todas las operaciones de los ejemplos se pueden conseguir desde el panel Estructura del modelo directamente, sin necesidad de usar diagramas de modelado para crear clases ni métodos.

Crear un proyecto nuevo y definir una raíz de espacio de nombres

En el panel Estructura del modelo:

1. Seleccione **Archivo | Nuevo** para crear un proyecto de modelado nuevo.
2. Haga clic con el botón secundario en el paquete `Root`. Seleccione **Elemento nuevo | Paquete** y cree un paquete nuevo llamado `MiPaquete`.



3. Haga clic con el botón secundario en `MiPaquete` y seleccione **Ingeniería de código | Establecer como raíz de espacio de nombres Java**. Aparece un aviso. Haga clic en **Sí** para aplicar el perfil Java de UModel al paquete. El paquete `Java Profile` se añade a la Estructura del modelo.

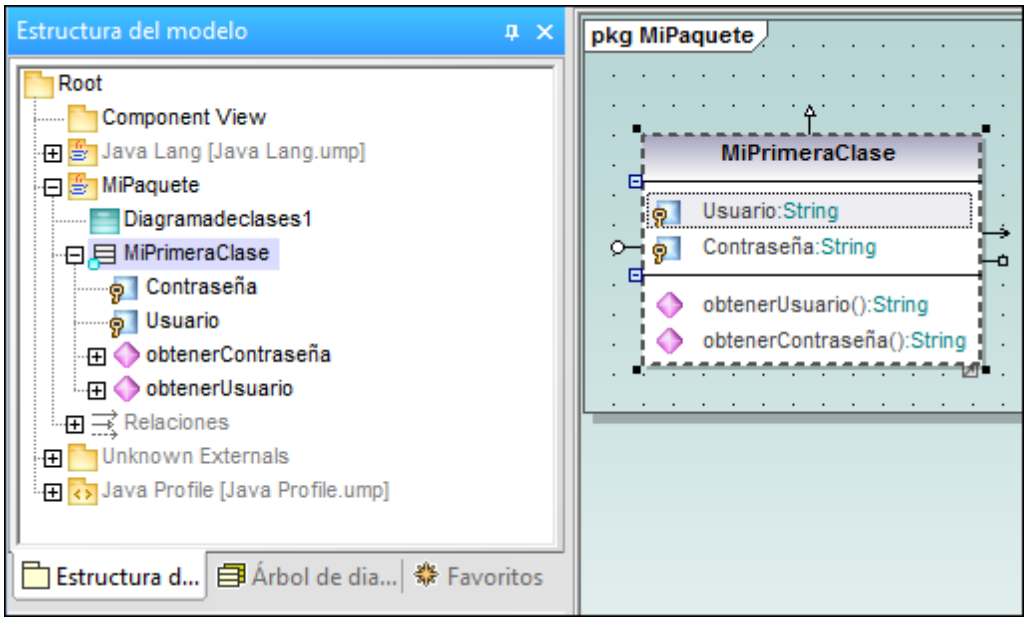
Incluir Java Lang para aportar tipos de datos JDK

1. Haga clic en el paquete `Root` y seleccione el comando de menú **Proyecto | Incluir un subproyecto**.
2. En el cuadro de diálogo que aparece seleccione la pestaña *Java 1.4* y después el paquete `Java Lang.ump`. En el siguiente cuadro de diálogo haga clic en **Aceptar** para usar la opción predeterminada *Incluir mediante referencia*.

Los paquetes **`Java Lang`** y **`Unknown Externals`** se añaden a la Estructura del modelo (*imagen anterior*).

Crear las propiedades y los métodos de las clases

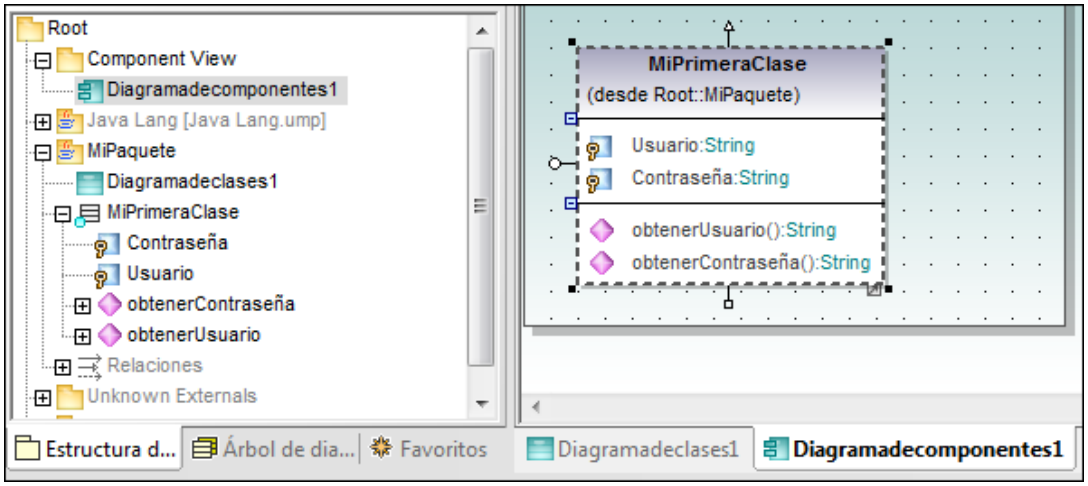
1. Haga clic con el botón secundario en **`MiPaquete`** y seleccione **Diagrama nuevo | Diagrama de clases** en el menú contextual.
2. Haga clic con el botón secundario en el diagrama de clases y seleccione **Elemento nuevo | Clase** para crear una clase nueva en el diagrama (p. ej. **`MiClase`**).
3. Pulse **F7** y añada algunos atributos (p. ej. `Usuario:String` y `Contraseña:String`).
4. Pulse **F8** y añada algunas operaciones (p. ej. `obtenerUsuario():String` y `obtenerContraseña():String`).



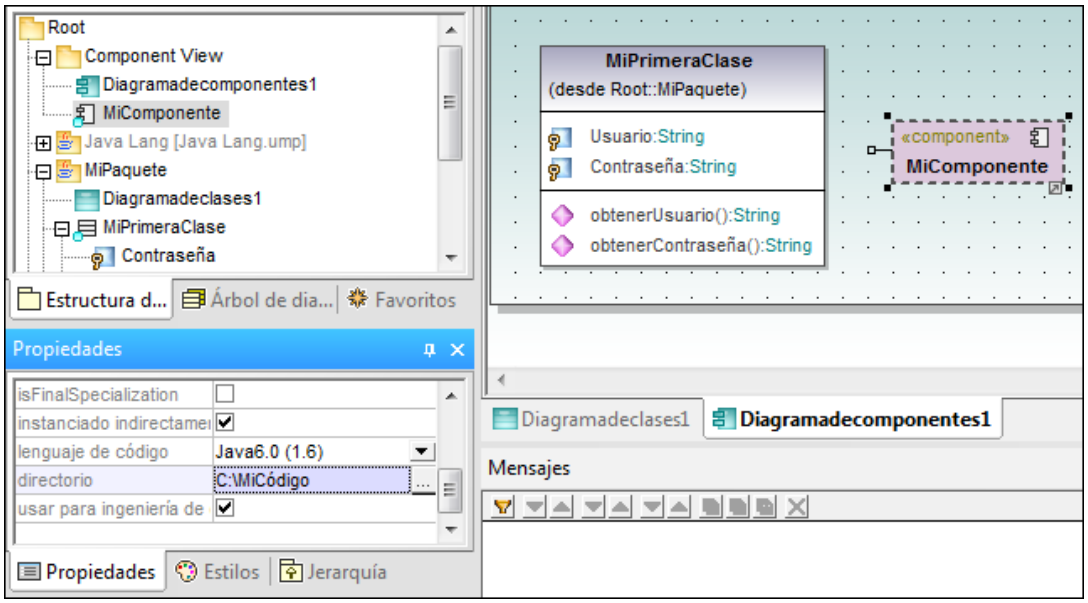
Crear un componente y definir el directorio del código

1. Haga clic con el botón secundario en el paquete **`Component View`** y añada un diagrama de componentes nuevo.
2. Arrastre la clase **`MiPrimeraClase`** desde la Estructura del modelo hasta el diagrama de componentes.

 © 2014 Altova GmbH

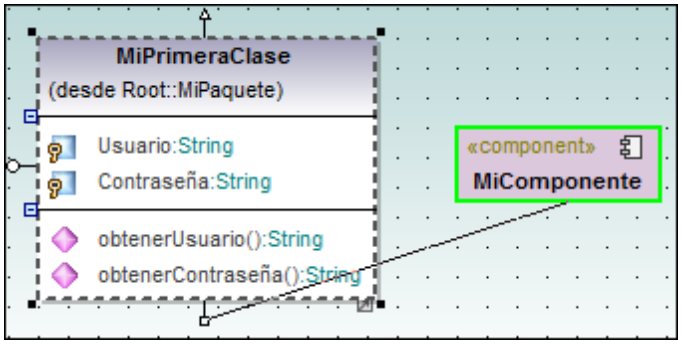


3. Añada un componente nuevo al diagrama (p. ej. **`MiComponente`**).
4. Haga clic en el componente y después en el campo directorio del panel Propiedades. Ahora inserte el directorio donde desea guardar el código (p. ej. `c:\MiCódigo`).



Realizar la clase

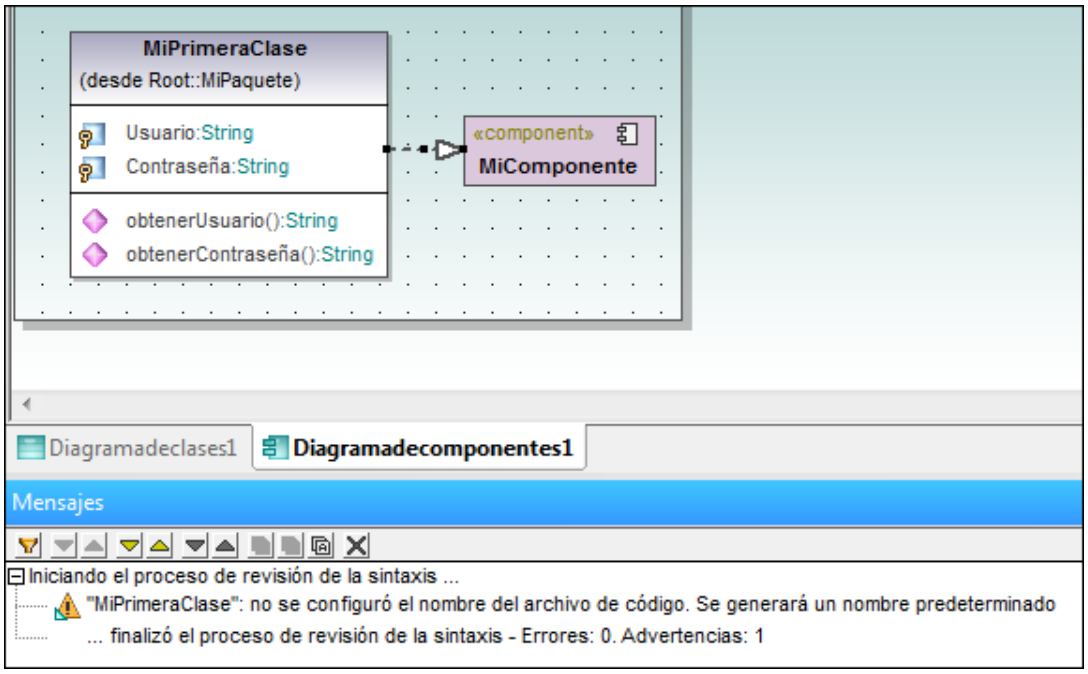
1. Haga clic en **`MiClase`** y arrastre el controlador **`RealizaciónDeComponente`** (situado en la parte inferior del recuadro) hasta el componente **`MiComponente`**.



Para poder generar código es necesario realizar las clases. No olvide que también puede arrastrar la clase y colocarla en el componente desde la Estructura del modelo directamente.

Revisar la sintaxis y generar código

1. Seleccione el comando de menú **Proyecto | Revisar la sintaxis del proyecto** para comprobar que todo es correcto.



No se genera ningún error pero hay un mensaje de advertencia. Aunque no definió un nombre para el archivo de código, UModel generará un nombre predeterminado automáticamente.

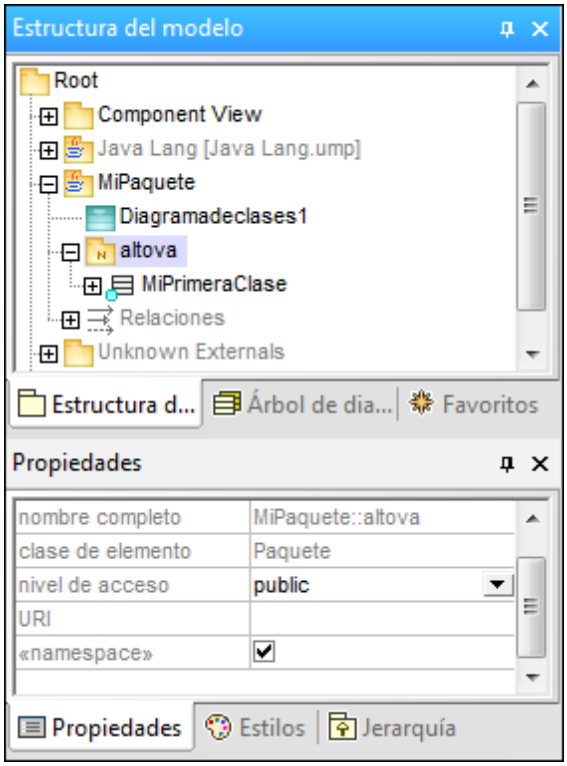
2. Seleccione el comando de menú **Proyecto | Combinar el código de programa con el proyecto de UModel...** para generar el código Java.

Crear un espacio de nombres

Si quiere generar la **clase** dentro de un espacio de nombres determinado:

1. Añada un paquete nuevo dentro de `MiPaquete` (p. ej. `altova`).
2. Haga clic en el paquete `altova` y marque la casilla <<**namespace**>> del panel

Propiedades.



3. Ahora, en la Estructura del modelo, arrastre la clase **`MiPrimeraClase`** hasta el paquete **`altova`**.
   Cuando se genere el código, la clase estará en el espacio de nombres **`altova`**.

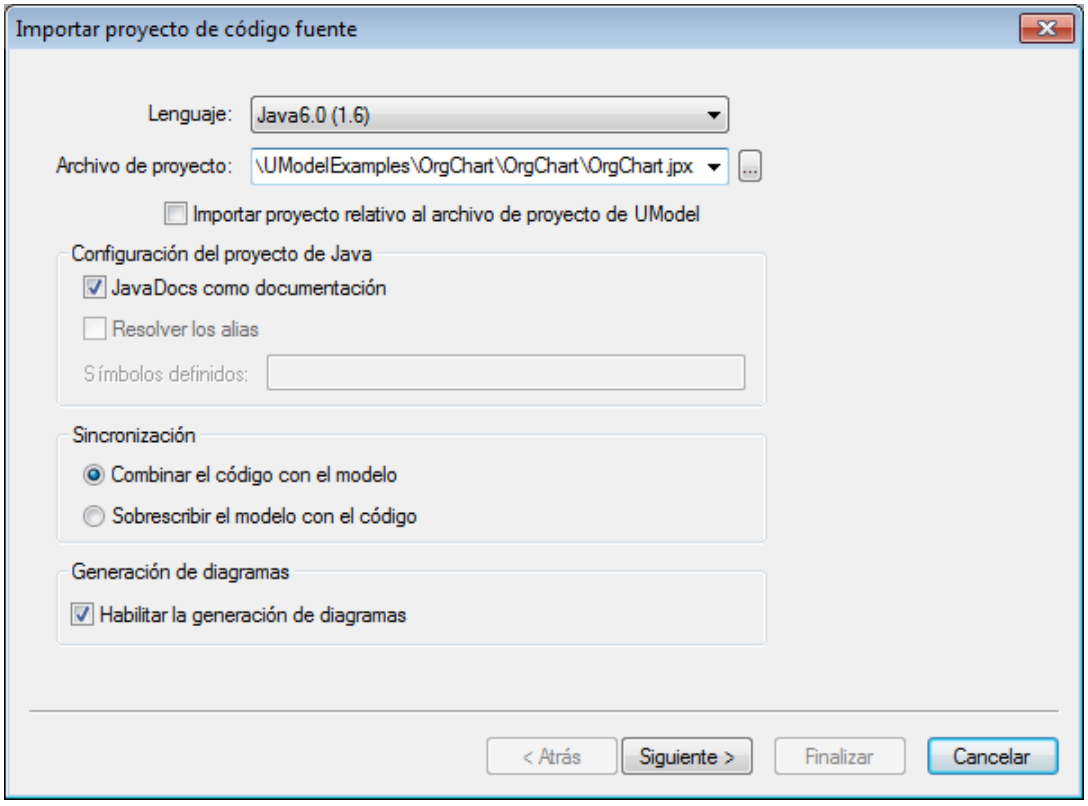
7.2 Importar código fuente a los proyectos

Puede importar código fuente como proyecto de código fuente o como directorio de código fuente. Para ver un ejemplo de cómo importar código como **directorio de código fuente**, consulte el apartado Ingeniería de ida y vuelta (código - modelo - código) del tutorial.

- Puede importar archivos JBuilder `.jpx`, archivos de proyecto Eclipse `.project` y archivos NetBeans (`proyecto.xml`).
- También puede importar proyectos en C# / Visual Basic:
  - proyectos de MS Visual Studio (`sln, csproj, csdprj..., vbproj, vbp`) y
  - archivos de proyectos Borland `.bdsproj`.

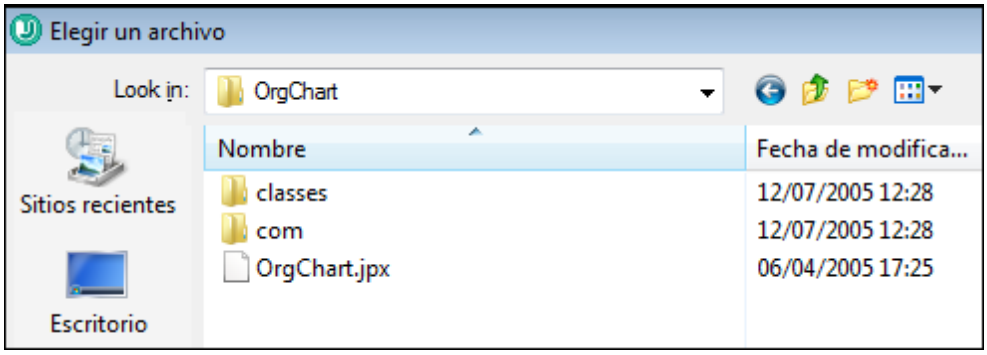
Para importar a UModel un proyecto ya existente:

1. Seleccione el comando de menú **Proyecto | Importar proyecto de código fuente**.
2. En el cuadro de diálogo haga clic en el botón **Examinar** para seleccionar el archivo de proyecto.

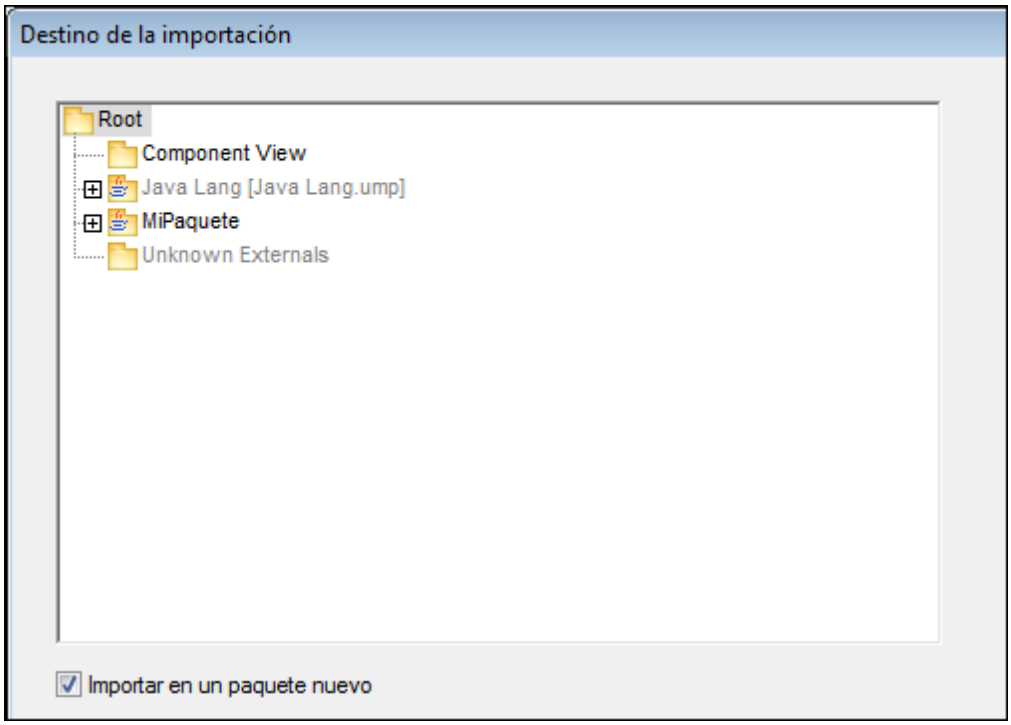


3. Seleccione el tipo de archivo de proyecto (p. ej. `.jpx`) y haga clic en **Abrir**. Este archivo de proyecto JBuilder está disponible en el archivo `OrgChart.zip` de la carpeta `...\UModelExamples`.
   No olvide que está activa la opción para importar el proyecto relativo al archivo de proyecto de UModel.

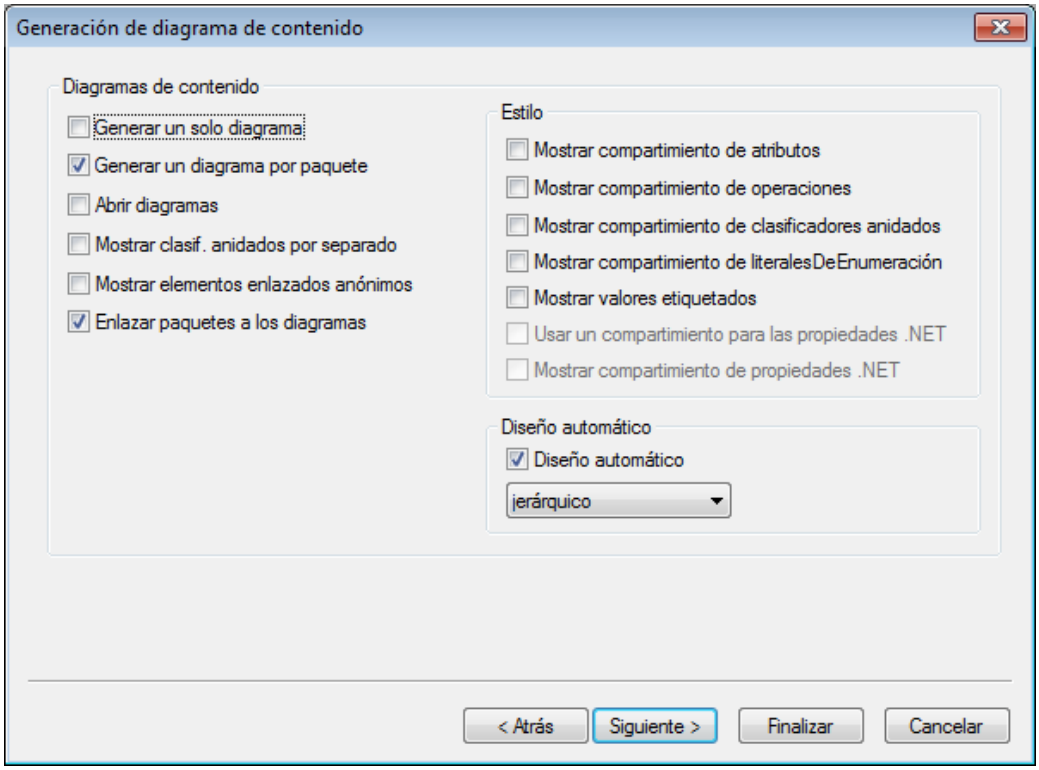
 © 2014 Altova GmbH



4. Compruebe que la casilla *Habilitar la generación de diagramas* está marcada y termine de configurar la importación. Haga clic en **Siguiente**.
5. En la siguiente pantalla, marque la casilla *Importar en un paquete nuevo* (o haga clic en el paquete donde desea importar el código).

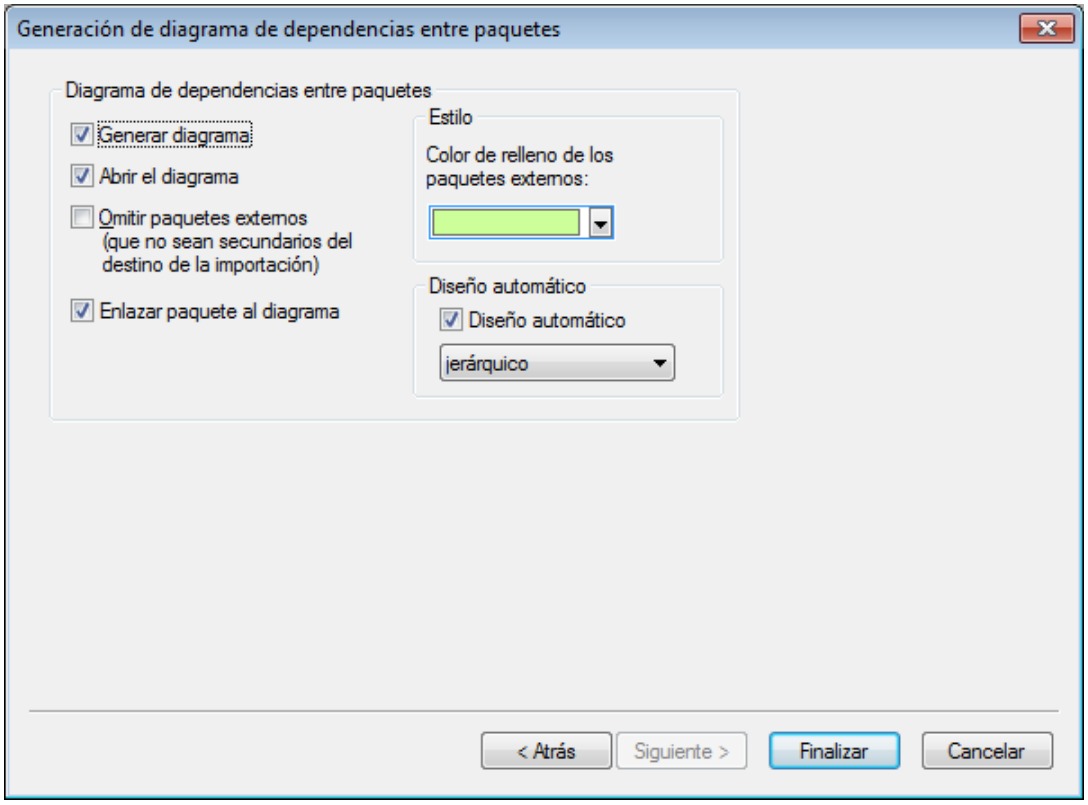


No olvide que UModel puede generar un solo diagrama global o un diagrama por cada paquete. Las opciones de configuración que aparecen a continuación son las opciones predeterminadas.

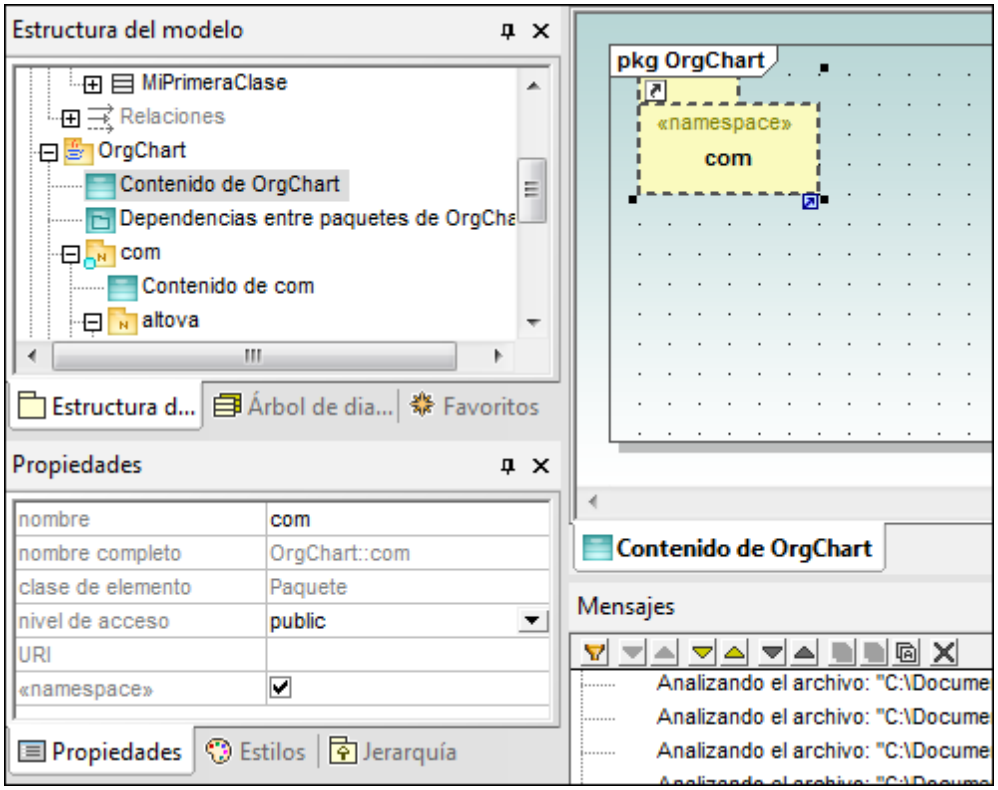


6. Haga clic en **Siguiente** para continuar.
   En la siguiente pantalla puede configurar la generación del diagrama de dependencias entre paquetes.

 © 2014 Altova GmbH



7. Haga clic en **Finalizar** para usar la configuración predeterminada.
   El proyecto se analiza y el modelo de UModel se genera.



Nota: si importa el código en un proyecto ya disponible, UModel le pregunta en qué paquete se debe importar. Si usa un proyecto nuevo, se crea automáticamente la carpeta **`OrgChart`**.

Resolver alias:
Cuando aplique ingeniería inversa a código que contiene alias de espacio de nombres o alias de clase, podrá elegir si el alias se resuelve o no (activando/desactivando la casilla *Resolver alias* del cuadro de diálogo "Importar proyecto/directorio de código fuente").

Cuando actualice el código (con el modelo), las definiciones de alias se retienen en el código tal y como están. El ámbito de las definiciones de alias son los archivos en los que aparecen.

Por ejemplo:

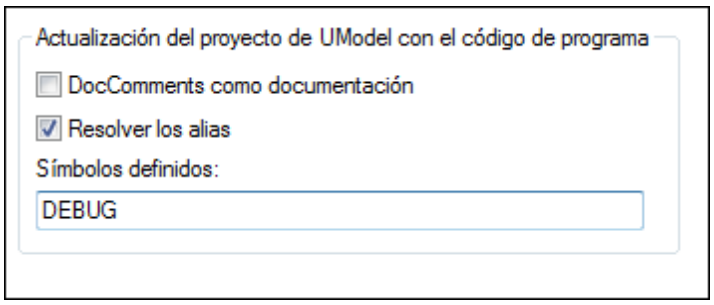
```
using Q = System.Collections.Generic.Queue<String>;
Q myQueue;
```

Los alias que puedan estar en conflicto porque su uso sea ambiguo se añaden a **`Unknown Externals`**.

Nota: la configuración de la opción *Resolver alias* puede cambiarse en cualquier momento en el cuadro de diálogo "Configuración del proyecto".

Símbolos definidos
El código C# o Visual Basic permite insertar una lista de símbolos definidos en el campo *Símbolos definidos* del cuadro de diálogo "Configuración del proyecto". Esta lista se usa para compilar secciones del código de forma condicional. En la lista los símbolos deben ir separados por punto y coma. UModel tiene en cuenta los símbolos definidos durante el proceso de ingeniería de código.

Una vez finalizado el proceso de ingeniería inversa, UModel da salida a todos los símbolos utilizados en el código fuente en la ventana Mensajes.

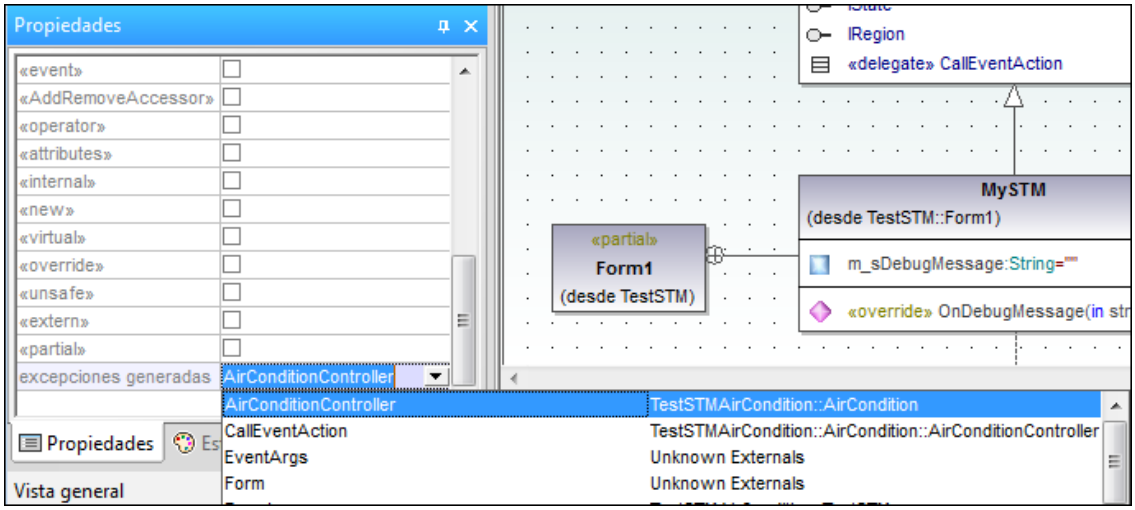


Por ejemplo:
```
#If DEBUG Then
    Dim i = 10
    #Else
    dim a = 20
    #End If
```

Excepciones generadas
Si hace clic en una operación de una clase y después hace clic en el cuadro combinado `excepciones generadas` (panel Propiedades), podrá ver qué información ofrece la excepción que

 © 2014 Altova GmbH

se genera para una operación.



7.3 Importar binarios Java, C# y VB

UModel permite la importación de binarios C# , Java y VB, lo cual es de gran utilidad cuando se trabaja con binarios de terceros o cuando el código fuente original ya no está disponible.

Si tiene pensado importar archivos binarios Java o C#, es necesario instalar estos programas/ componentes:

Java 1.4 / 5.0:
Sun Java Runtime Environment (JRE) o el kit de desarrollo JDK en las versiones 1.4, 1.5 o 1.6.

Compatibilidad con UModel:
La importación de tipos es compatible con todos los archivos de clases que apunten a estos entornos (es decir, que cumplan la especificación de Java Virtual Machine).

C# 2.0 / C# 3.0: / C# 4.0 / Visual Basic 7.1, 8.0 y 9.0
.NET Framework 2.0, 3.0, 3.5 y 4.0

Compatibilidad con UModel:
La importación de tipos es compatible con todos los ensamblados que apunten a:
.NET Framework 1.1, 2.0, 3.0, 3.5, 4.0
.NET Compact Framework v1.0, v2.0, v3.5 (para PocketPC, teléfonos inteligentes, WindowsCE)

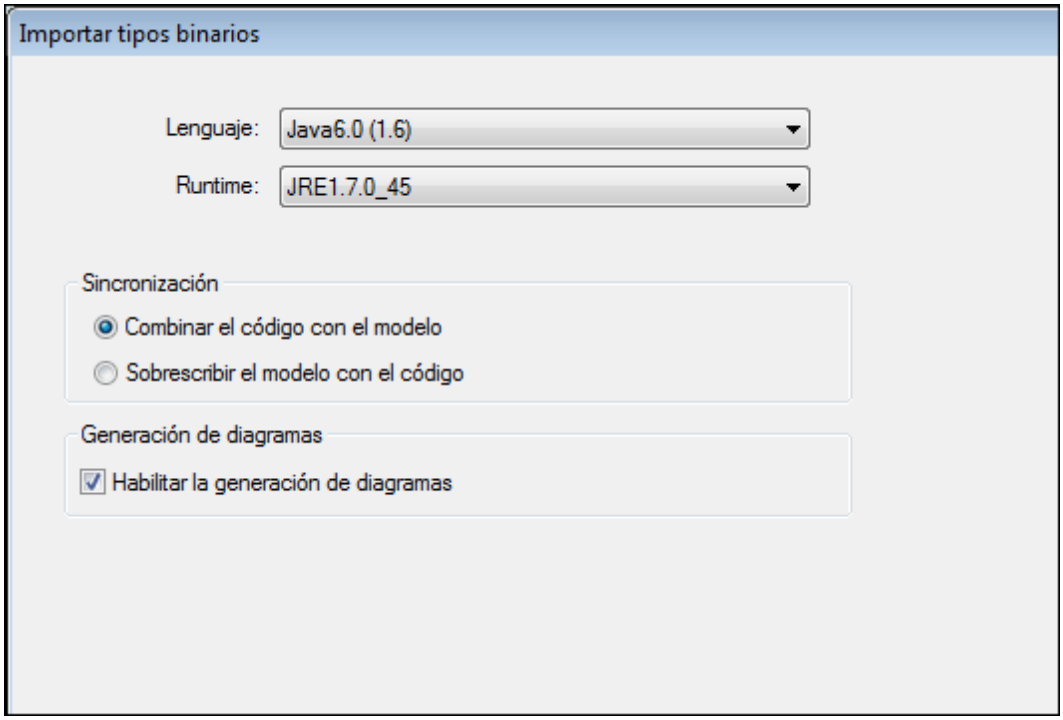
Estos requisitos solamente son relevantes si tiene pensado importar binarios Java, C# o VB. NET. Si no es así, no hace falta tener instalado Java Runtime Environment ni MS .NET Framework.

No es posible importar binarios confusos.

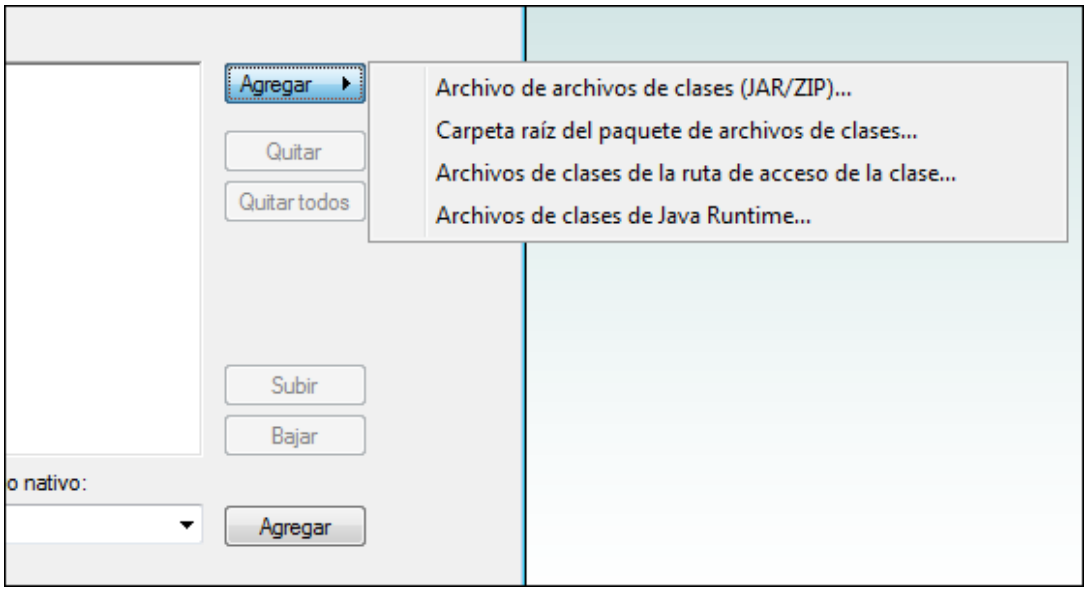
Para importar archivos binarios:

1. Seleccione el comando de menú **Proyecto | Importar tipos binarios**.

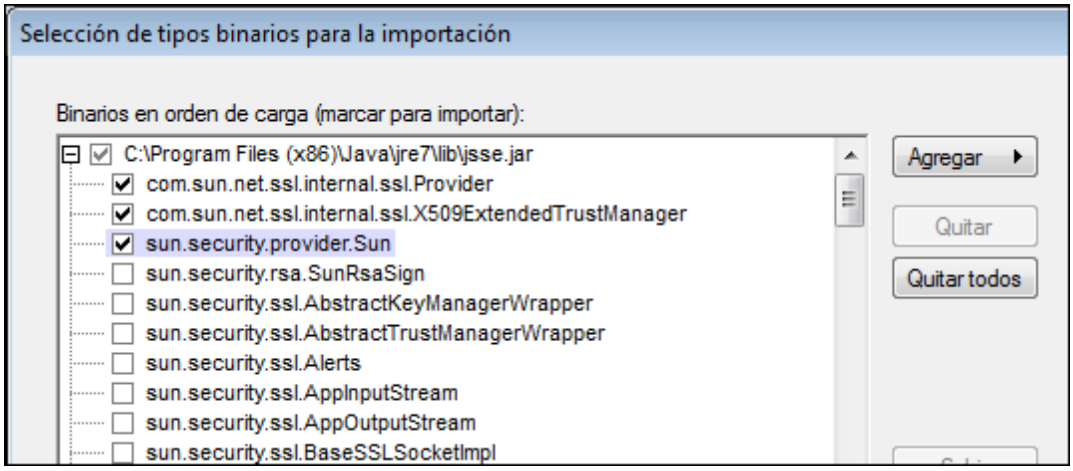
 © 2014 Altova GmbH



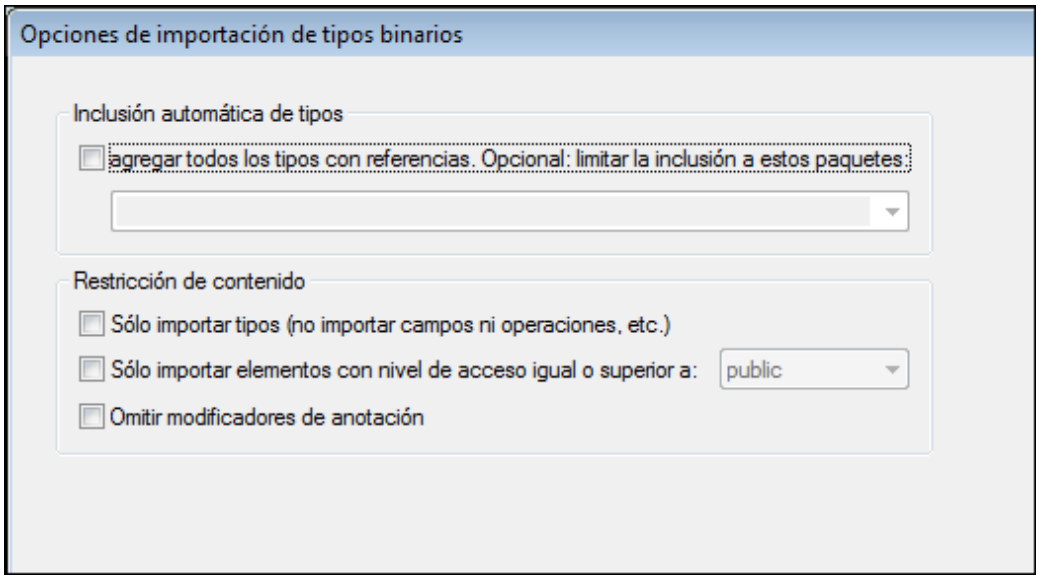
2. En el cuadro de diálogo "Importar tipos binarios", seleccione el lenguaje y la edición de Runtime. Después haga clic en **Siguiente**.
   La siguiente pantalla es la de selección de tipos binarios.
3. Haga clic en el botón **Agregar** y seleccione el archivo de clases en el menú emergente (p. ej. **Archivos de clases de Java Runtime**).



3. Ahora haga clic en el botón de expansión **+** para expandir la lista de binarios y marque las casillas de los binarios que desea importar (p. ej. los tres primeros). Después haga clic en **Siguiente**.

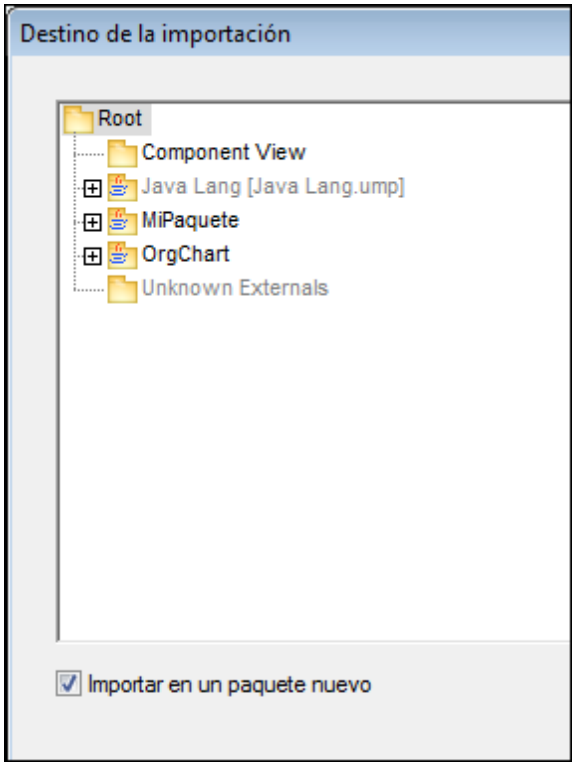


La siguiente pantalla ofrece las opciones de importación:

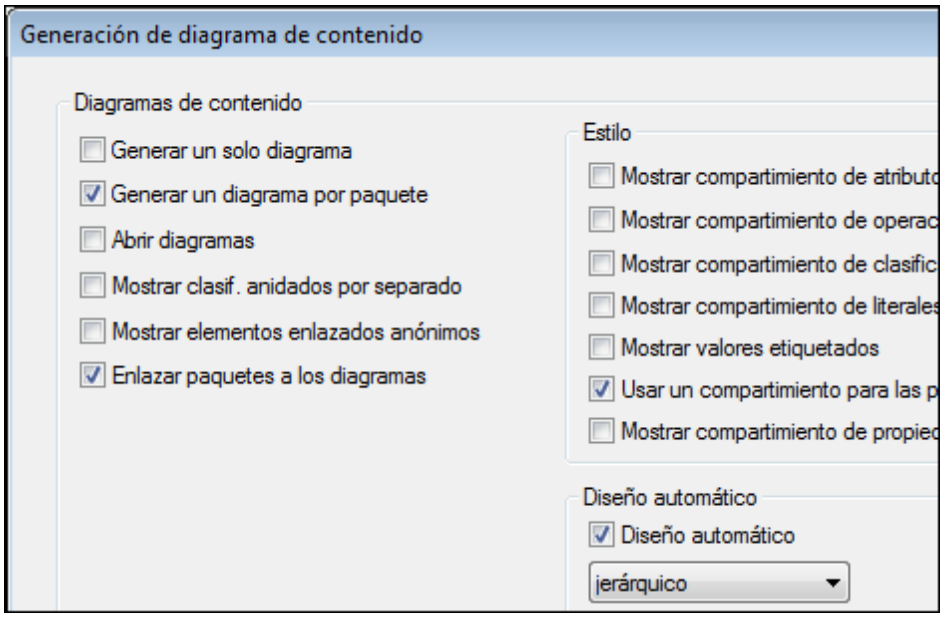


4. Seleccione las opciones pertinentes y haga clic en **Siguiente** para continuar.
   En la siguiente pantalla puede elegir el destino de la importación: un paquete ya disponible o uno nuevo. Haga clic en **Siguiente**.

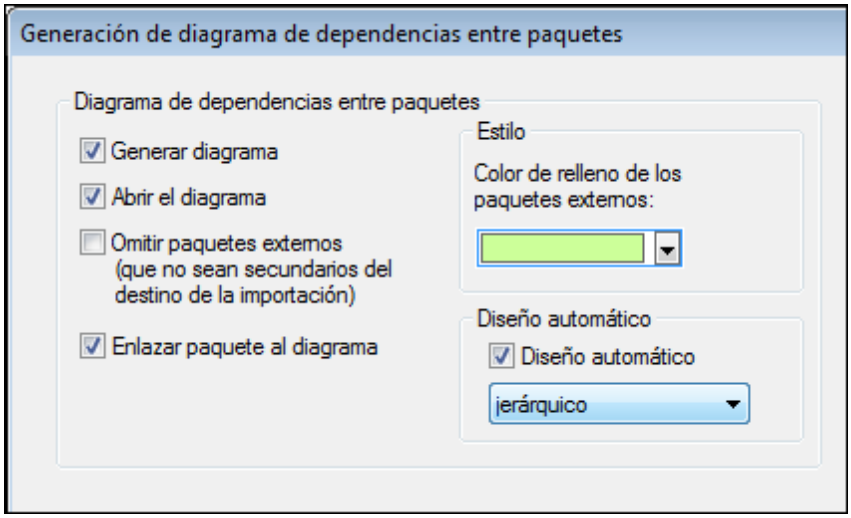
 © 2014 Altova GmbH



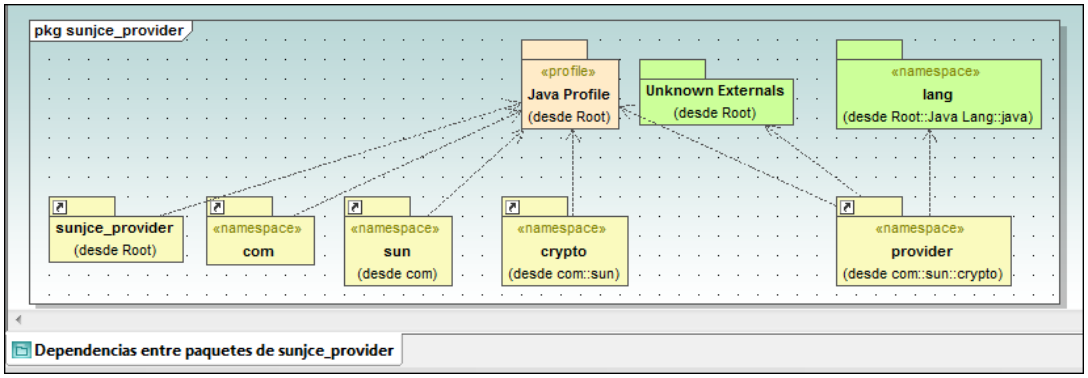
5. En la siguiente pantalla puede configurar la generación de diagramas de contenidos. Cuando termine haga clic en **Siguiente**.
   Recuerde que puede generar un diagrama por cada paquete o un solo diagrama global.



6. En la siguiente pantalla configure la generación de diagramas de dependencias entre paquetes y haga clic en **Finalizar** para terminar el proceso de importación.

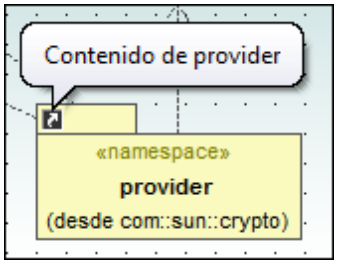


7. La imagen siguiente muestra el diagrama que contiene las dependencias entre paquetes de los binarios de Java.



8. Haga clic en las demás pestañas para ver los archivos de clases, etc.

Nota: al hacer clic en el icono de hipervínculo de una carpeta, se abre automáticamente el diagrama al que remite.



 © 2014 Altova GmbH

7.4 Sincronizar el modelo y el código fuente

UModel ofrece una función para sincronizar el modelo con el código y viceversa.

Sincronización de código y modelo
El código se puede combinar y sincronizar por niveles, tal y como se describe a continuación. Si la función se ejecuta desde el menú **Proyecto**, el código solamente se sincroniza a nivel raíz/de proyecto. Si la función se ejecuta desde el menú contextual (de la Estructura del modelo), se sincroniza el componente en el que se hizo clic.

Para sincronizar el modelo/código a nivel raíz/de proyecto:

1. Haga clic con el botón secundario en el paquete `Root`.
2. Seleccione una opción de combinación de código: **Ingeniería de código | Combinar código de programa...** o **Ingeniería de código | Combinar proyecto de UModel...**
   Si lo prefiere, utilice el comando correspondiente del menú **Proyecto**.

Para sincronizar el modelo/código a nivel de paquetes:

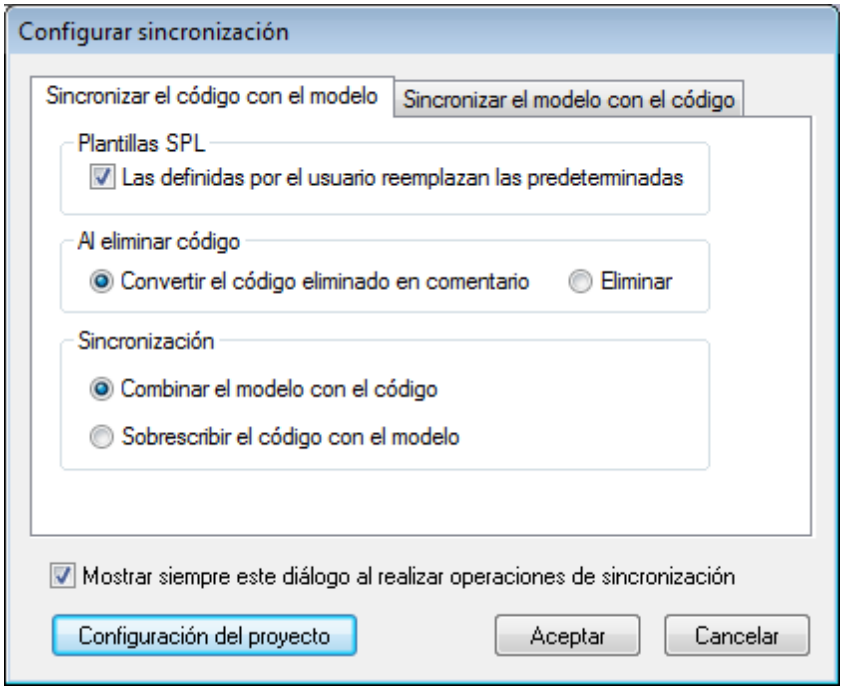
1. Seleccione los paquetes que desea combinar.
2. Haga clic con el botón secundario y seleccione una opción de combinación de código: **Ingeniería de código | Combinar código de programa...** o **Ingeniería de código | Combinar proyecto de UModel...**.

Para sincronizar el modelo/código a nivel de clases:

1. Seleccione las clases que desea combinar.
2. Haga clic con el botón secundario y seleccione una opción de combinación de código: **Ingeniería de código | Combinar código de programa...** o **Ingeniería de código | Combinar proyecto de UModel...**.

Para definir las opciones de sincronización:

1. Seleccione el comando **Proyecto | Configurar sincronización** para abrir el cuadro de diálogo "Configurar sincronización".
   Este cuadro de diálogo incluye una pestaña para la sincronización del código y otra para la sincronización del modelo.
2. Haga clic en el botón **Configuración del proyecto** para definir las opciones relacionadas con el lenguaje de programación.
3. Cuando termine, haga clic en **Aceptar**.



Nota: durante la sincronización de código puede aparecer un aviso pidiendo que actualice el proyecto de UModel antes de iniciar la sincronización. Esto solo ocurre si usa proyectos de UModel creados con una versión de UModel anterior a la versión más reciente. Para actualiza el proyecto haga clic en **Sí** y después guarde el archivo de proyecto. Una vez actualizado el proyecto, el aviso no vuelve a aparecer.

¿Qué efectos tiene actualizar el modelo?

- Si actualiza el modelo con el comando **Proyecto | Combinar el proyecto de UModel con el código de programa**:
  - UModel repasa todos los directorios (archivos de proyecto) de los lenguajes de código definidos en el proyecto.
  - UModel detecta los archivos nuevos y los añade al proyecto.
  - En la ventana Mensajes aparece el mensaje `Recopilando los archivos de código fuente en C:\UMTest`.

- Si en la Estructura del modelo hace clic con el botón secundario en una clase, interfaz, etc. y selecciona el comando **Ingeniería de código | Combinar la clase de UModel con el código de programa**:
  - UModel actualiza la clase, interfaz. etc. seleccionada del proyecto.
  - Como no se pueden seleccionar clases que faltan, al proyecto no se añade ninguna clase.

- Si en la Estructura del modelo hace clic con el botón secundario en un componente (del paquete `Component View` por lo general) y selecciona el comando **Ingeniería de código | Combinar el componente de UModel con el código de programa**:
  - UModel actualiza solamente el directorio (archivo de proyecto) correspondiente.
  - UModel detecta los archivos nuevos en el directorio (archivo de proyecto) y los añade al proyecto.
  - En la ventana Mensajes aparece el mensaje `Recopilando los archivos de código fuente en C:\UMTest`.

 © 2014 Altova GmbH

Plantillas SPL:
Las plantillas SPL se utilizan durante la generación de código Java, C# y VB.

Para modificar las plantillas SPL:

1. Busque las plantillas SPL que vienen con UModel en el directorio predeterminado `...\UModel2015\UModelSPL\Java\Default\` (o `...\C#\Default` o `...\VB\Default`.)
2. Copie los archivos SPL que desea editar/modificar en el directorio primario (es decir, `...\UModel2015\UModelSPL\Java\`).
3. Realice los cambios y guárdelos.

Para usar las plantillas SPL definidas por el usuario:

1. Seleccione la opción de menú **Proyecto | Configurar sincronización**.
2. Marque la casilla *Las definidas por el usuario reemplazan las predeterminadas* (del grupo de opciones *Plantillas SPL*).

Para iniciar el proceso de sincronización, seleccione uno de estos dos comandos:

- **Proyecto | Combinar el código de programa con el proyecto de UModel** (consulte el apartado Ingeniería de ida y vuelta (modelo - código - modelo para más información)
- **Proyecto | Combinar el proyecto de UModel con el código de programa** (consulte el apartado Ingeniería de ida y vuelta (código - modelo - código para más información).

Nota: las plantillas SPL solamente se usan cuando se genera código nuevo (es decir, cuando se añaden clases nuevas, operaciones nuevas, etc. al modelo). El código ya disponible no accede a las plantillas SPL ni las utiliza.

7.4.1 Consejos prácticos

Cambiar el nombre de los clasificadores y aplicar ingeniería inversa:
El proceso descrito más abajo tiene lugar durante la ingeniería inversa y la sincronización automática, tanto en la versión independiente de UModel como en los complementos de UModel para Visual Studio y Eclipse.

Si cambia el nombre de un clasificador en la aplicación de programación, el clasificador se elimina o se vuelve a insertar en el panel Estructura del modelo de UModel como clasificador nuevo.

El clasificador nuevo solo se vuelve a insertar en los diagramas de modelado que se crean automáticamente durante el proceso de ingeniería inversa o cuando se genera un diagrama con el comando **Mostrar en un diagrama nuevo de | Contenido**. El clasificador nuevo se inserta en una posición predeterminada en el diagrama, que probablemente no coincida con su ubicación anterior.

UModel también ofrece funciones para refactorizar clases (consulte el apartado Refactorizar código y sincronización para más información).

Para habilitar la generación automática de RealizacionesDeComponente:

- Haga clic en la opción de menú **Herramientas | Opciones**. En el cuadro de diálogo que aparece abra la pestaña *Ingeniería de código* y active la casilla *Generar las RealizacionesDeComponente que falten*.

Generación automática de RealizacionesDeComponente
UModel puede generar RealizacionesDeComponente automáticamente durante el proceso de ingeniería de código. Las RealizacionesDeComponente solo se generan cuando está totalmente claro a qué componente se debe asignar una clase, es decir:

- Cuando solo existe un archivo de proyecto de Visual Studio en el archivo `*.ump`.
- Cuando existen varios proyectos de Visual Studio pero sus clases están totalmente separadas en el modelo.

Las RealizacionesDeComponente automáticas se crean para un clasificador al que se le puede asignar un solo componente. Es decir:

- un clasificador que no tiene ninguna RealizaciónDeComponente o
- un clasificador que está dentro del espacio de nombres de un lenguaje de código

Hay varias maneras de buscar componentes, dependiendo del tipo de componente.

Si se trata de componentes que representan un archivo de proyecto de código (cuando tiene definida la propiedad `projectfile`):

- se busca si hay UN componente que tiene/realiza clasificadores en el paquete que lo contiene.
- se busca si hay UN componente que tiene/realiza clasificadores en un subpaquete del paquete que lo contiene (de arriba a abajo).
- se busca si hay UN componente que tiene/realiza clasificadores en uno de los paquetes primarios (de abajo a arriba).
- se busca si hay UN componente que tiene realiza clasificadores en un subpaquete de uno de los paquetes primarios (de arriba a abajo).

Si se trata de componentes que representan un directorio (cuando tiene definida la propiedad

 © 2014 Altova GmbH

`directory`):

- se busca si hay UN componente que tiene/realiza clasificadores en el paquete que lo contiene
- se busca si hay UN componente que tiene/realiza clasificadores en uno de los paquetes primarios (de abajo a arriba)

Notas:

- Es necesario activar la opción *Generar las realizacionesDeComponente que falten* (de la pestaña *Ingeniería de código* del cuadro de diálogo "Opciones locales").
- En cuanto UModel encuentra UN componente viable durante los pasos descritos más arriba, el componente se utiliza y se omiten los pasos siguientes.

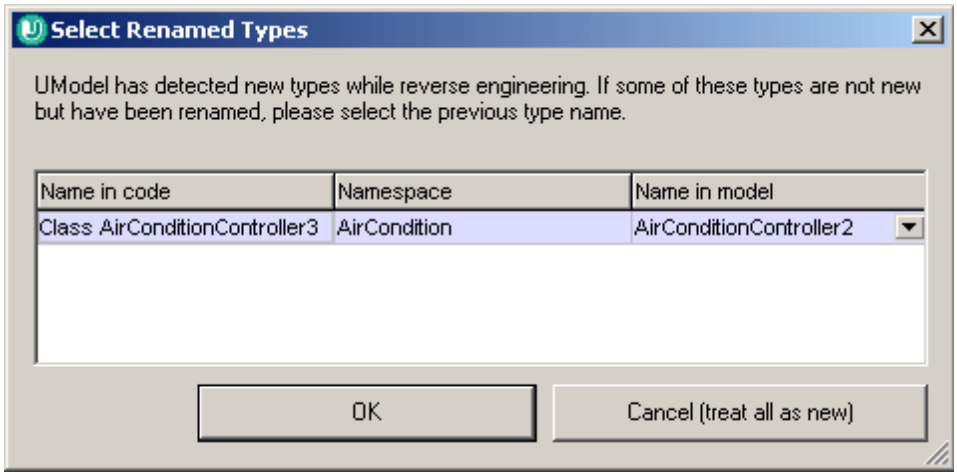
Errores/advertencias:

- Si no se encuentra ningún componente viable, se genera una advertencia.
- Si se encuentran varios componentes viables, se genera un error.

7.4.2 Refactorización de código y sincronización

Cuando se refactoriza código, el nombre de las clases suele cambiar o actualizarse. Las versiones de UModel anteriores a la versión 2009 eliminaban las clases antiguas e insertaban clases nuevas durante el proceso de sincronización del código/modelo.

Si UModel detecta que durante la fase de ingeniería inversa se añadieron tipos nuevos o se cambió el nombre de algunos tipos, aparece el cuadro de diálogo "Seleccionar tipos con nombre nuevo" (*imagen siguiente*). La columna *Nombre en el código* enumera los tipos nuevos, mientras que el nombre original de cada tipo aparece en la columna *Nombre en el modelo*. UModel trata de averiguar cuál era el nombre original del tipo a partir del espacio de nombres, el contenido de la clase, las clases bases y otros datos.



Si se cambió el nombre de una clase, seleccione el nombre antiguo de la clase en la lista desplegable de la columna *Nombre en el modelo* (p. ej. `C1`). Esto permite conservar todos los datos relacionados y que el proceso de ingeniería de código funcione con precisión.

Cambiar el nombre de las clases en el modelo y volver a generar código
Tras crear un modelo y generar código a partir de él, si quiere puede volver a realizar cambios en el modelo antes de iniciar el proceso de sincronización.

Por ejemplo, imagine que quiere cambiar el nombre de las clases antes de generar código por segunda vez. Como previamente asignó un nombre de archivo a cada clase, en el campo `nombre del archivo de código`, la clase nueva y el nombre de archivo no coinciden.

Cuando inicie el proceso de sincronización, UModel le pregunta si quiere que el nombre del archivo de código coincida con el nombre de la clase nueva. Recuerde que también tiene la opción de cambiar los constructores de clase.

 © 2014 Altova GmbH

7.5 Requisitos para ingeniería directa

Requisitos para generar código para ingeniería directa:

- Un componente es **realizado** por una **clase** o varias o por una **interfaz** o varias.
- El componente debe tener asignada una ubicación física (es decir, un directorio). El código generado se coloca después en este directorio.
- Cada componente debe estar configurado para incluirse en el proceso de generación de código.
- El paquete raíz de espacio de nombres Java, C# o VB debe estar definido.

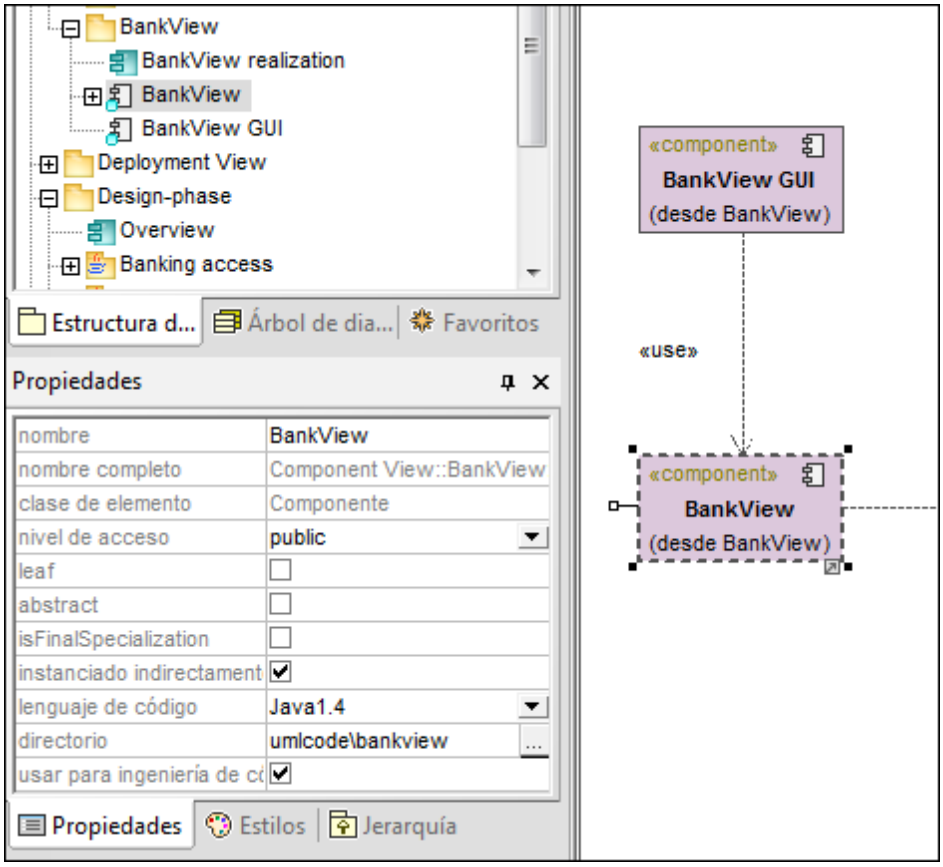
Para crear una realización de componente:

1. Arrastre la clase o interfaz al componente correspondiente en el panel Estructura del modelo.

   También puede crear una realización en un diagrama de componentes usando el icono **Realización** de la barra de herramientas.

Para asignar una ubicación física:

1. Seleccione el componente en la Estructura del modelo o en el diagrama.
2. Haga clic en el botón **Examinar** ... de la propiedad directorio y seleccione un `directorio`.



Para incluir componentes en el proceso de ingeniería de código:

1. Seleccione el componente en la Estructura del modelo o en el diagrama.
2. En el panel Propiedades active la casilla `usar para ingeniería de código`.

Para definir la raíz de espacio de nombres Java:

1. Haga clic con el botón secundario en un paquete y seleccione **Establecer como raíz de espacio de nombres de Java**.

   Esto significa que este paquete y todos los subpaquetes se habilitan durante el proceso de ingeniería de código. En el panel Estructura del modelo la raíz de espacio de nombres Java se señala con el icono .

- Seleccione el comando **Establecer como raíz de espacio de nombres de Java** otra vez para quitar el espacio de nombres Java del paquete.

© 2014 Altova GmbH

7.6 Correspondencia entre código Java y elementos de UModel

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de código Java cuando se genera código a partir del modelo.
- Elementos de código Java y elementos UModel cuando se actualiza el modelo con el código.

A continuación puede ver un resumen de la tabla de correspondencia. Para ver la versión completa en su explorador web, haga clic aquí.

Java ◁▷ UModel					
Java			UModel		
Project	projectfile		projectfile		Component
	directory		directory		
Package	name		name		Package <<namespace>>
	name		name		
	modifiers	package	visibility	package	
		public		public	
		protected		protected	
		private		private	
		abstract	abstract		
		final	isFinalSpecification		
		strictfp	<<strictfp>>		
	filename		code file name		
	associated projectfile/directory		ComponentRealization		
	extends clause		Generalization		
	implements clause		InterfaceRealization(s)		
	java docs		Comment(->Documentation)		

7.7 Correspondencia entre código C# y elementos de UModel

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de código C# cuando se genera código a partir del modelo.
- Elementos de código C# y elementos UModel cuando se actualiza el modelo con el código.

A continuación puede ver un resumen de la tabla de correspondencia. Para ver la versión completa en su explorador web, haga clic aquí.

C# ◁▷ UModel					
C#			UModel		
Project	projectfile		projectfile		Component
	directory		directory		
Namespace	name		name		Package <<namespace>>
	name		name		
	modifiers	internal	visibility	package	
		protected internal		protected <<internal>>	
		public		public	
		protected		protected	
		private		private	
		sealed	isFinalSpecification		
		abstract	abstract		
		static	<<static>>		
		unsafe	<<unsafe>>		
		partial	<<partial>>		
		new	<<new>>		
	filename		code file name		
	associated projectfile/directory		ComponentRealization		
	base types		Generalization, InterfaceRealization(s)		
	attribute sections		<<attributes>>		
	doc comments		Comment(->Documentation)		

 © 2014 Altova GmbH

7.8 Correspondencia entre elementos de XML Schema y de UModel

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de XML Schema cuando se genera código a partir del modelo.
- Elementos de XML Schema y elementos UModel cuando se actualiza el modelo con el código.

Los elementos UML/XSD aparecen en **color azul**.

Las propiedades de estereotipo (=valores etiquetados) aparecen en **color verde**.

A continuación puede ver un resumen de la tabla de correspondencia. Para ver la versión completa en su explorador web, haga clic aquí.

XSD ◁▷ UModel						
XSD				UModel		
file path				project file		Component
	target namespace			name		Package <<namespace>>
	attributeFormDefault			attributeFormDefault		
	blockDefault			blockDefault		
	elementFormDefault			elementFormDefault		
	finalDefault			finalDefault		
	version			version		
	xml:lang			xml:lang		
	xmlns			xmlns		
	annotation	source		source		
		appinfo			Comment <<appinfo>>	
		documentation	xml:lang	xml:lang	Comment <<documentation>>	

7.9 Correspondencia entre código VB.NET y de elementos de UModel

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de código VB.NET cuando se genera código a partir del modelo.
- Elementos de código VB.NET y elementos UModel cuando se actualiza el modelo con el código.

A continuación puede ver un resumen de la tabla de correspondencia. Para ver la versión completa en su explorador web, haga clic aquí.

VB.NET ◁▷ UModel					
VB.NET			UModel		
Project	projectfile		projectfile		Component
	directory		directory		
Namespace	name		name		Package <<namespace>>
	name		name		
	modifiers	Friend	visibility	package	
		Protected Friend		protected <<Friend>>	
		Public		public	
		Protected		protected	
		Private		private	
		NotInheritable	isFinalSpecification		
		MustInherit	abstract		
		Partial	<<Partial>>		
		Shadows	<<Shadows>>		
	filename		code file name		
	associated projectfile/directory		ComponentRealization		
	base types		Generalization, InterfaceRealization(s)		
	attribute sections		<<Attributes>>		
	doc comments		Comment(->Documentation)		

 © 2014 Altova GmbH

7.10 Correspondencia entre elementos de BD y de UModel

La tabla que aparece más abajo muestra la correspondencia entre:

- Elementos de UModel y elementos de BD cuando se genera código a partir del modelo.
- Elementos de BD y elementos UModel cuando se actualiza el modelo con el código.

| Database | | |

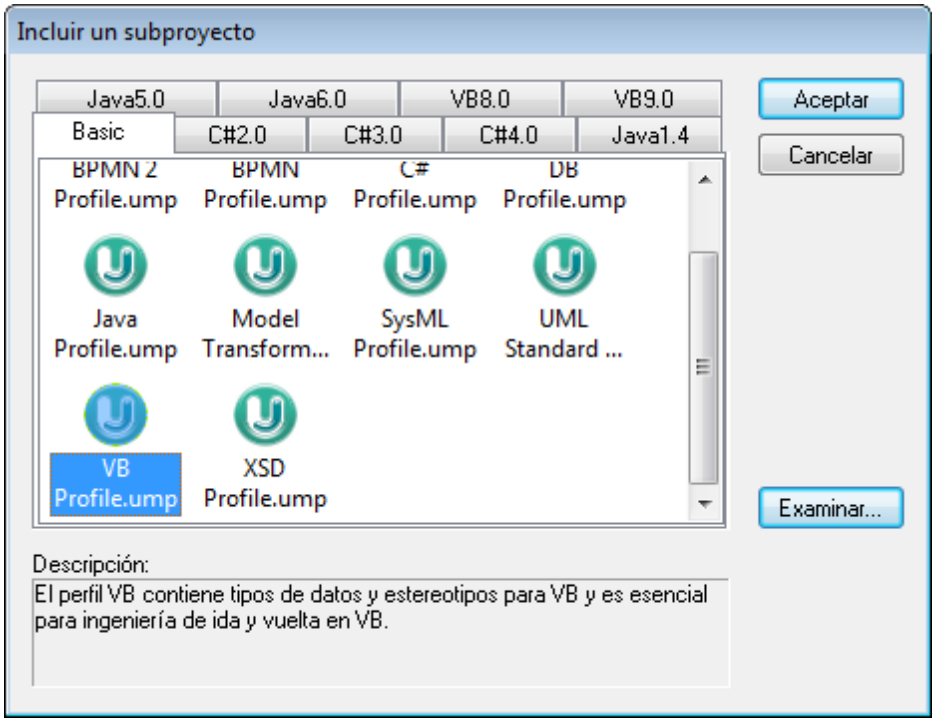
7.11 Incluir otros proyectos de UModel

UModel viene con varios archivos que se pueden incluir en los proyectos de UModel. Estos archivos incluyen clases, interfaces, paquetes Java, etc.

1. Seleccione el comando de menú **Proyecto | Incluir un subproyecto**.
   Esto abre el cuadro de diálogo "Incluir un subproyecto" (*imagen siguiente*).
2. Seleccione la pestaña pertinente y después haga clic en el archivo de proyecto de UModel que desea incluir y pulse **Aceptar**.

Los proyectos de UModel se pueden incluir dentro de otros proyectos de UModel. Para incluir un proyecto en otro:

- Guárdelo en la carpeta `...\UModel2015\UModelInclude` (si quiere que aparezca en la pestaña *Basic* del cuadro de diálogo "Incluir un subproyecto") o
- Guárdelo en la carpeta `...\UModel2015\UModelInclude\Java1.4 / Java5.0 / Java6.0` (si quiere que aparezca en la pestaña *Java* correspondiente del cuadro de diálogo "Incluir un subproyecto").

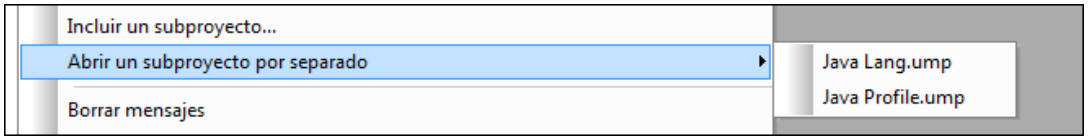


Nota: en la pestaña C# 2.0 encontrará un archivo de inclusión que contiene todos los tipos de Microsoft .NET Framework 2.0.

Para ver todos los proyectos que están incluidos actualmente:

- Seleccione la opción de menú **Proyecto | Abrir un subproyecto por separado**.
  El menú emergente enumera los subproyectos que están incluidos en el proyecto.

 © 2014 Altova GmbH



Para crear una pestaña/carpeta definida por el usuario:

1. Navegue hasta la carpeta `...\UModel2015\UModelInclude` **y cree/añada la carpeta nueva bajo la carpeta** `...\UModelInclude` (por ejemplo `...\UModelInclude\miCarpeta`).

Para crear una descripción para cada archivo de proyecto de UModel:

1. Cree un archivo de texto en la misma carpeta en la que se encuentra el archivo `*.ump` y con el mismo nombre.
   P. ej. para el archivo `MiModelo.ump` debería crear el archivo `MiModelo.txt`. Asegúrese de usar la codificación UTF-8 para este archivo de texto.

Para quitar un proyecto ya incluido:

1. Haga clic en el paquete incluido en el panel Estructura del modelo y pulse la tecla **Supr**.
2. La aplicación le pregunta si desea continuar con la eliminación.
3. Haga clic en **Aceptar** para eliminar el archivo incluido del proyecto.

Nota: para eliminar o quitar un proyecto del cuadro de diálogo "Incluir un subproyecto" debe eliminar el proyecto `*.ump` en la correspondiente carpeta.

7.12 Combinar proyectos de UModel

En UModel puede realizar fusiones a 2 y 3 bandas para combinar varios proyectos de UModel distintos en un solo modelo `*.ump`.

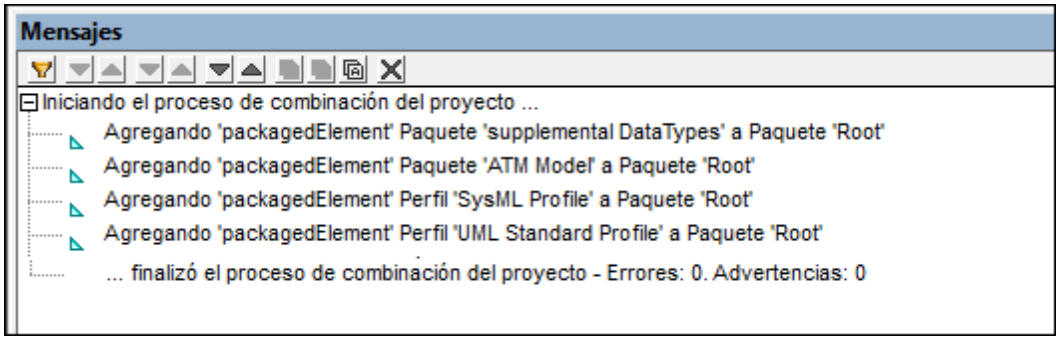
Esta función es de gran utilidad si en el mismo proyecto trabajan varias personas a la vez o si quiere reunir todo su trabajo en un solo modelo.

© 2014 Altova GmbH

7.12.1 Fusión de proyectos a 2 bandas

Para combinar dos proyectos UML:

1. Abra el archivo UML que debe hacer de archivo de destino (es decir, el archivo con el que se debe combinar el otro modelo).
2. Seleccione el comando **Proyecto | Combinar el proyecto**.
3. Seleccione el proyecto UML que debe combinarse con el proyecto principal.
   En la ventana Mensajes puede comprobar cómo se desarrolla el proceso de combinación.



Nota: si hace clic en una entrada de la ventana Mensajes el elemento de modelado correspondiente aparece resaltado en la Estructura del modelo.

¿Qué efectos tiene la fusión de proyectos a 2 bandas?

- **Elementos de modelado nuevos:** los elementos que no existían en el proyecto principal se añaden al proyecto.
- **Diferencias entre los mismos elementos de modelado:** los elementos del segundo modelo tienen prioridad. Por ejemplo, solo puede haber un valor predeterminado de un atributo, así que se usa el valor predeterminado del segundo modelo.
- **Diferencias entre diagramas**: UModel comprueba si hay diferencias entre los diagramas de los dos modelos.

  - Si hay diferencias, los diagramas nuevos/diferentes se añaden al modelo principal (con un sufijo numérico tipo `actividad1` etc.) y el diagrama original se conserva.

  - Si no hay diferencias, no se realiza ningún cambio. Después puede eliminar los diagramas que quiera.
- El proceso de combinación entero se puede deshacer paso a paso con el comando **Deshacer** (**Ctrl+Z** o con el botón de la barra de herramientas).
- Al hacer clic en una entrada de la ventana Mensajes aparece resaltado el elemento de modelado correspondiente en la Estructura del modelo
- El proyecto principal combinado conserva el mismo **nombre de archivo**.

7.12.2 Fusión de proyectos a 3 bandas

UModel también ofrece una función para combinar varios proyectos de UModel editados por programadores diferentes. Esta función se conoce como *fusión de proyectos a 3 bandas*.

Combinar varios proyectos de UModel
La fusión de proyectos a 3 bandas sirve para combinar proyectos de UModel de nivel superior, es decir, proyectos principales que pueden contener subproyectos. La fusión a 3 bandas no funciona con archivos independientes si tienen referencias sin resolver a otros archivos.

Cuando se combinan proyectos principales, sus subproyectos editables también se combinan automáticamente. Es decir, no hace falta combinar los subproyectos por separado.

Para ver un ejemplo consulte el apartado Ejemplo de fusión manual a 3 bandas.

- El proceso de combinación se puede deshacer entero paso a paso con el comando **Deshacer** (**Ctrl+Z** o con el botón de la barra de herramientas).
- Si hace clic en una entrada de la ventana Mensajes, el elemento de modelado correspondiente aparece resaltado en la Estructura del modelo.
- El proyecto principal combinado (el que se eligió como destino de la combinación) conserva el mismo **nombre de archivo**.

¿Qué efectos tiene la fusión de proyectos a 3 bandas?
Es importante tener en cuenta que cuando decimos "proyecto original", nos referimos al primer archivo de proyecto que se eligió como destino de la combinación al principio del proceso.

- **Elementos de modelado nuevos:**
  - si el segundo proyecto tiene elementos que no existen en el proyecto original, estos elementos se añaden al proyecto de destino (resultado de la fusión).
  - si el proyecto original tiene elementos que no existen en el segundo proyecto, estos elementos se conservan en el proyecto de destino (resultado de la fusión).
- **Elementos de modelado eliminados:**
  - si en el segundo proyecto se eliminaron elementos que todavía existen en el proyecto original, estos elementos se eliminan en el proyecto de destino (resultado de la fusión).
  - si en el proyecto original se eliminaron elementos que todavía existen en el segundo proyecto, estos elementos se conservan en el proyecto de destino (resultado de la fusión).
- **Diferencias entre los mismos elementos de modelado:**
  - si una propiedad (p. ej. el nivel de acceso de una clase) se modificó en el proyecto original o en el segundo proyecto, el modelo de destino incluirá el valor más reciente.
  - si una propiedad (p. ej. el nivel de acceso de una clase) se modificó tanto en el archivo original como en el segundo archivo, el modelo de destino incluirá el valor del segundo archivo (y aparece una advertencia en la ventana Mensajes).
- **Elementos con una posición distinta:**
  - si un elemento cambió de posición en el proyecto original o en el segundo proyecto, en el modelo de destino también se cambia la posición del elemento.
  - si un elemento cambió de posición tanto en el proyecto original como en el segundo

 © 2014 Altova GmbH

proyecto, aparece un aviso y el usuario puede seleccionar el elemento primario donde se debe insertar el elemento en el modelo de destino.

- **Diferencias entre diagramas**:
  UModel comprueba si hay diferencias entre los diagramas de los dos modelos.
  - si hay diferencias, el diagrama nuevo o diferente se añade al modelo de destino (con un sufijo numérico tipo `actividad1`, etc.) y el diagrama original se conserva.
  - si no hay diferencias, no se realizan cambios. Después puede eliminar los diagramas que quiera.

Sistemas de control de versiones para fusiones a 3 bandas
Cuando trabaje con un sistema de control de versiones y proteja/desproteja archivos de proyecto, UModel genera automáticamente archivos de instantánea denominados *archivo antecesor común,* que se utilizan para el proceso de fusión a 3 bandas y que permiten una combinación mucho más precisa que la fusión a 2 bandas.

La fusión a 3 bandas **automática** de UModel no funciona con todos los sistemas de control de versiones, pero en casos así puede usar la fusión a 3 bandas **manual**.

- Los sistemas de control de versiones que combinan archivos automáticamente sin la intervención del usuario no suelen ser compatibles con la fusión a 3 bandas automática de UModel.
- Los sistemas de control de versiones que piden que el usuario elija entre **Reemplazar** o **Combinar** cuando cambia un archivo de proyecto suelen ser compatibles con la fusión a 3 bandas automática de UModel. Después de que el control de versiones reemplace el archivo, seleccione el comando **Reemplazar** para activar el aviso de UModel que permite realizar una fusión a 3 bandas. Para el proceso de protección/desprotección de los archivos es necesario usar UModel.
- Puede poner bajo control de versiones tanto el proyecto principal como sus subproyectos. Si cambia los datos de un subproyecto, recibirá un aviso automático para desproteger el subproyecto.
- Cada acción de protección/desprotección crea un *archivo antecesor común* o de instantánea que se utiliza después durante el proceso de fusión a 3 bandas.

Nota: los archivos de instantánea solo se crean y emplean automáticamente con la versión independiente de UModel (es decir, esta función no está disponible en el complemento de UModel para Eclipse o Visual Studio).

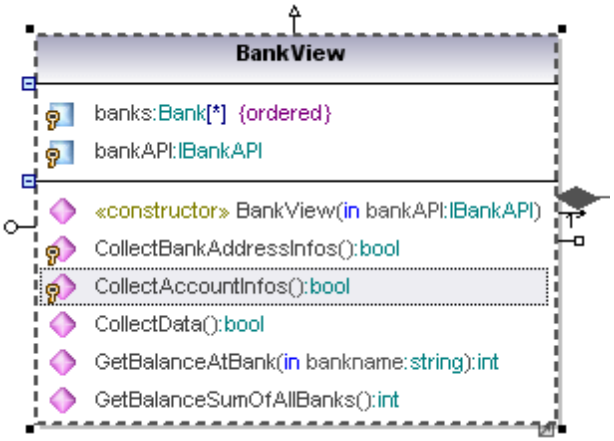
Por ejemplo, imagine que `Usuario A` edita un archivo de proyecto de UModel y cambia el nombre de una clase en el diagrama `BankView Main`. Ahora imagine que `Usuario B` abre el mismo archivo de proyecto y cambia el nivel de acceso de esa misma clase.

Como UModel genera un archivo de instantánea para cada usuario, el historial de edición de instantáneas permite combinar los diferentes cambios en el proyecto. Como resultado del proceso de fusión a 3 bandas, el nuevo nombre de la clase y su nuevo nivel de acceso se pasan al archivo de proyecto de destino (resultado de la fusión).

7.12.3 Ejemplo de fusión manual a 3 bandas

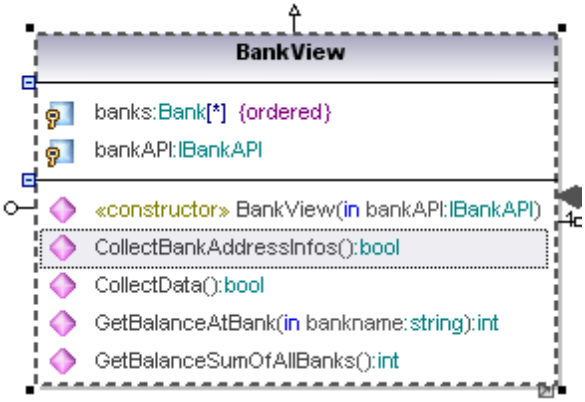
Para este sencillo ejemplo utilizamos el archivo de muestra `Bank_CSharp.ump` disponible en la carpeta `...\UModelExamples`. Las otras dos instancias del mismo proyecto se copian en dos carpetas secundarias (`\C#_1` y `\C#_2`) bajo la carpeta `UModelExamples`.

Para empezar, imagine que el `Usuario1` abre el archivo de proyecto `Bank_CSharp.ump` de la carpeta `C#_1` y realiza cambios en la clase **`BankView`**.



Cambios realizados por Usuario1 en la clase BankView:

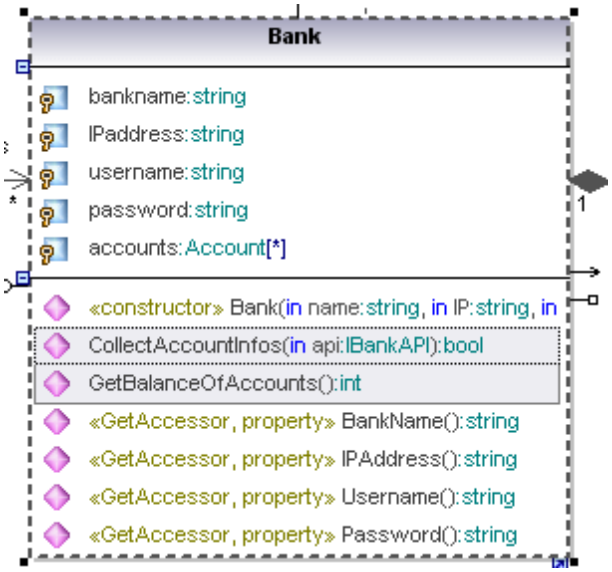
1. `Usuario1` **elimina** la operación **`CollectAccountInfos():bool`** de la clase **`BankView`**.
2. Después cambia el **nivel de acceso** de la operación **`CollectBankAddressInfos():bool`** de `protected` a `public`.



3. Y, por último, guarda el proyecto.

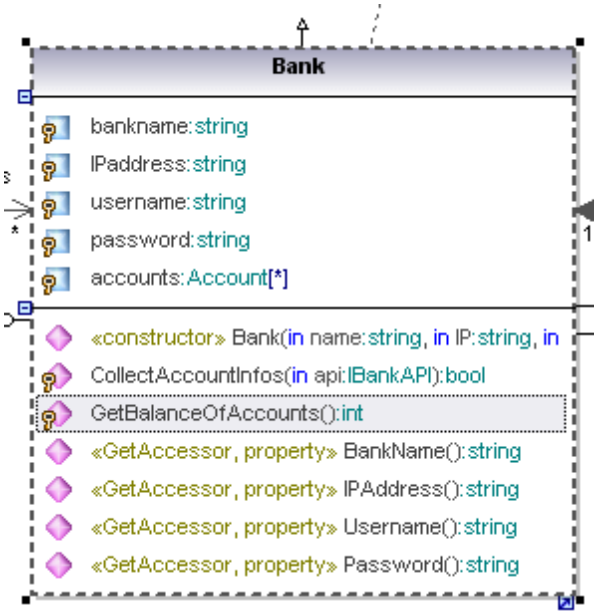
Ahora imagine que el `Usuario2` abre el archivo de proyecto `Bank_CSharp.ump` de la carpeta `C#_2` y realiza cambios en la clase **`Bank`**.

 © 2014 Altova GmbH



Cambios realizados por Usuario2 en la clase Bank:

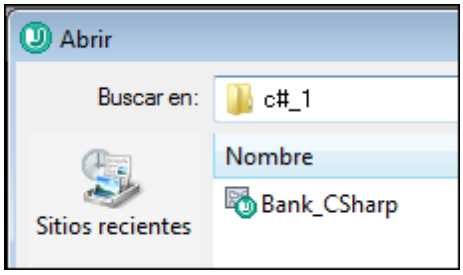
1. `Usuario2` cambia el nivel de acceso de las operaciones **`CollectAccountInfos`** y **`GetBalanceOfAccounts`** de `public` a `protected`.



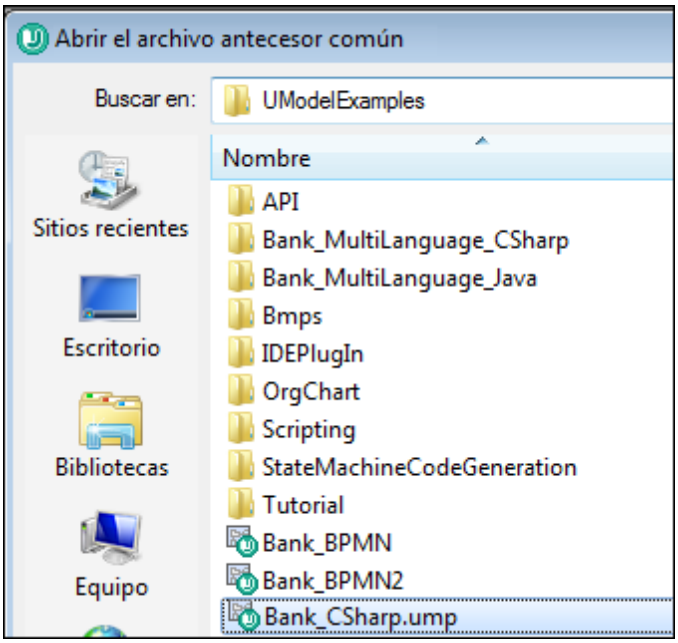
2. Y después guarda el proyecto.

Usuario2 inicia una fusión de proyecto a 3 bandas:

1. Primero selecciona el comando **Proyecto | Combinar el proyecto (fusión a 3 bandas)**. Aparece el cuadro de diálogo "Abrir".
2. Después selecciona el archivo de proyecto que modificó `Usuario1` en la carpeta . . . `\C#_1`.

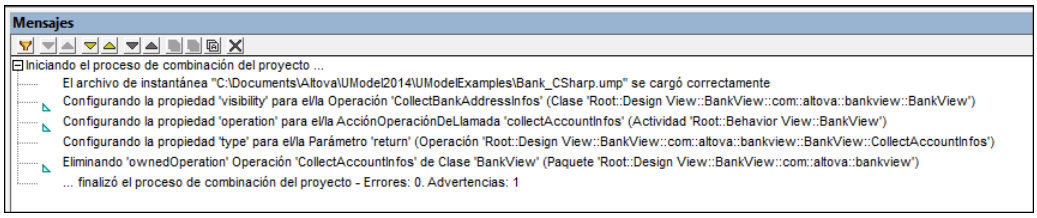


Ahora aparece un cuadro de diálogo "Abrir el archivo antecesor común", que es el archivo de proyecto original de la carpeta `...\UModelExamples`.



El proceso de fusión a 3 bandas se inicia y se retorna al archivo de proyecto desde el que se inició el proceso de fusión (es decir, desde el archivo de proyecto de la carpeta `C#_2`).

La ventana Mensajes muestra el desarrollo del proceso de combinación.



- Los cambios realizados en el proyecto de la carpeta `C#_1` se reproducen en el proyecto de la carpeta `C#_2`.
- Los cambios realizados en el proyecto de la carpeta `C#_2` se conservan en el archivo de proyecto.
- El archivo de proyecto de la carpeta `C#_2` debería utilizarse como archivo antecesor común para los procesos de fusión a 3 bandas que se realicen a partir de ahora con los archivos de proyecto de las carpetas `C#_1` y `C#_2`.

 © 2014 Altova GmbH

7.13 Compartir paquetes y diagramas

En UModel puede compartir paquetes y diagramas UML con otros proyectos. A su vez, los paquetes se pueden incluir en otros proyectos de UModel mediante referencia o como copia.

Además los subproyectos se pueden separar del proyecto principal en cualquier momento e incluirse otra vez en el proyecto principal como subproyectos editables o de solo lectura. Asimismo, los paquetes se comparten y se guardan como archivos de subproyecto.

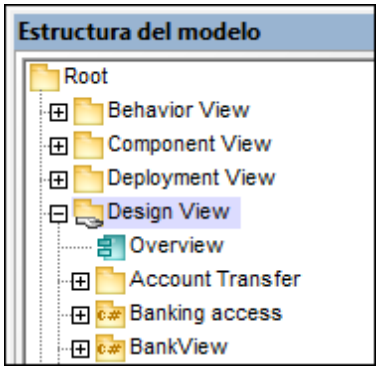
Nota: no olvide que los subproyectos también se pueden añadir al sistema de control de código fuente. Para más información consulte el apartado Trabajo en equipo con UModel.

Requisitos para compartir paquetes:
Es importante tener en cuenta que no se puede compartir un paquete que tenga vínculos a otros paquetes situados fuera del ámbito compartido.

Nota: cuando cree archivos de proyecto de UMOdel no use archivos de proyecto como plantilla/ copia de otro archivo de proyecto en el que quiere compartir un paquete. Esto daría lugar a conflictos porque cada elemento debe ser único de forma global (ver información sobre identificadores UUID) y en este caso los dos proyectos tendrían elementos con el mismo identificador uuid.

Para compartir un paquete con otros proyectos:

1. En la Estructura del modelo haga clic con el botón secundario en el paquete que desea compartir. Seleccione **Subproyecto | Compartir paquete**.

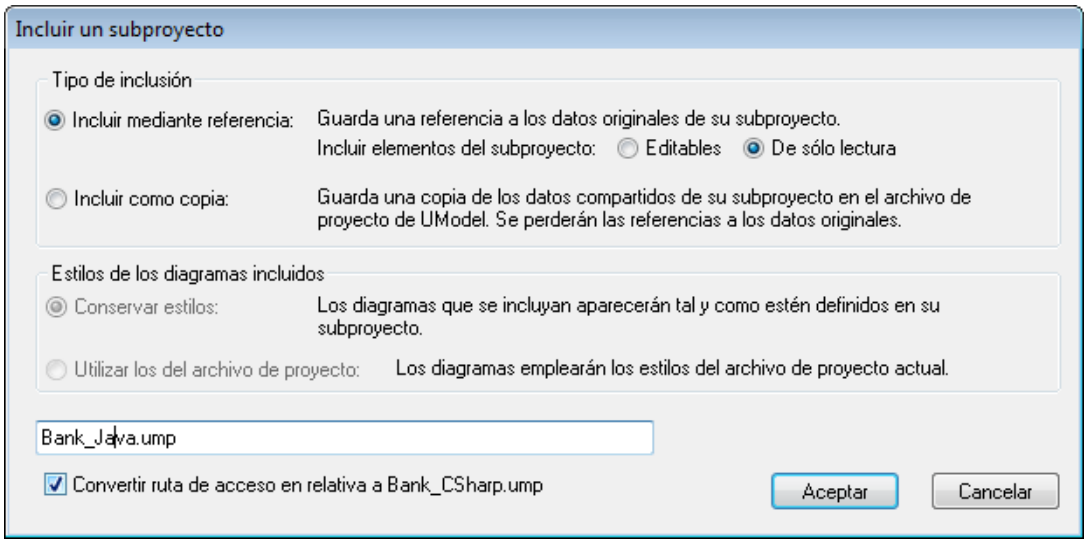


El icono del paquete ahora incluye una mano que indica que el paquete está compartido. A partir de ahora este paquete se podrá incluir en cualquier otro proyecto de UModel.

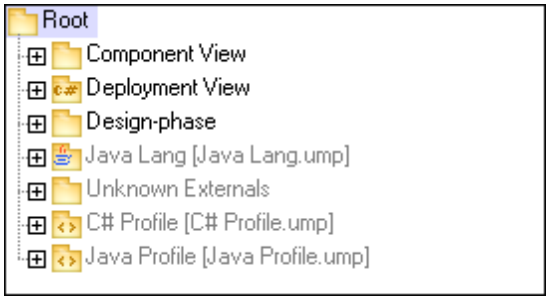
Para incluir/importar una carpeta compartida en un proyecto:

1. Abra el proyecto en el que desea incluir el paquete compartido (en este caso, un archivo vacío).

2. Seleccione la opción de menú **Proyecto | Incluir un subproyecto...**.
3. Aparece el cuadro de diálogo "Incluir un subproyecto" (*imagen siguiente*). Haga clic en el botón **Examinar**, seleccione el proyecto que incluye el paquete compartido y haga clic en **Abrir**.



4. En el cuadro de diálogo "Incluir un subproyecto" (*imagen anterior*) puede elegir si el paquete/proyecto se incluye mediante referencia o como copia.
   Seleccione una de las dos opciones y haga clic en **Aceptar**.

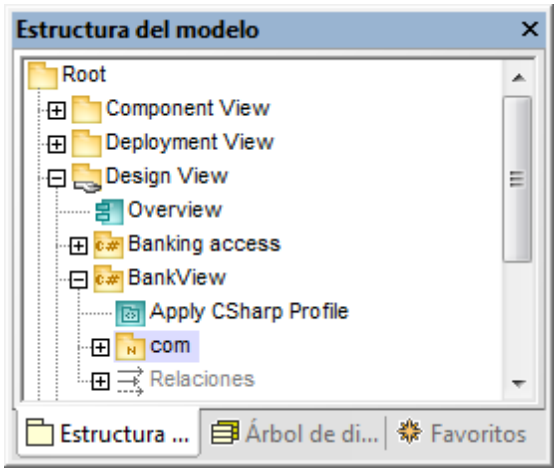


   El paquete `Deployment View` aparece ahora en el paquete nuevo. El proyecto de origen del paquete aparece entre paréntesis (`BankView-start.ump`).

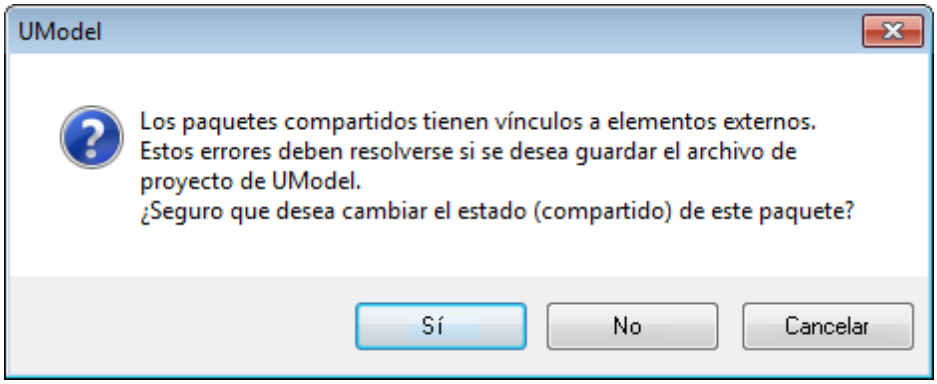
   Las carpetas compartidas que se incluyeron mediante referencia se pueden cambiar a *Incluir como copia* en cualquier momento. Esto se consigue haciendo clic con el botón secundario en la carpeta y seleccione el comando **Subproyecto | Incluir como copia**.

   Nota: también se incluyen todos los proyectos incluidos en el proyecto de origen: `Java Lang`, `Unknown Externals` y `Java Profile`.

Paquetes compartidos con vínculos a elementos externos:

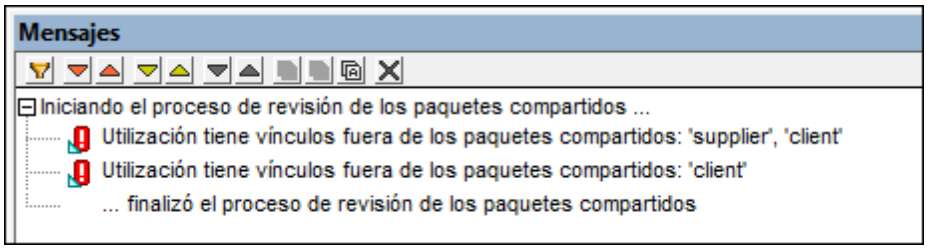


Si intenta compartir un paquete que tiene vínculos a elementos externos, aparece un aviso.

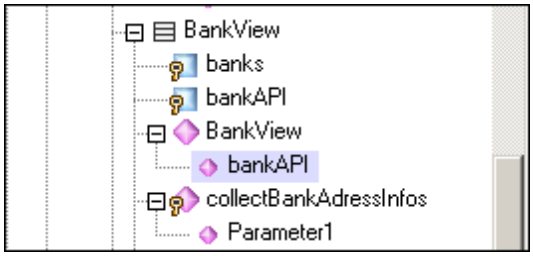


Si hace clic en **Sí**, UModel resuelve los vínculos externos antes de guardar el proyecto.

El panel Mensajes ofrece información sobre los vínculos externos.



Si hace clic en una entrada del panel Mensajes, el elemento correspondiente se resalta en el panel Estructura del modelo.



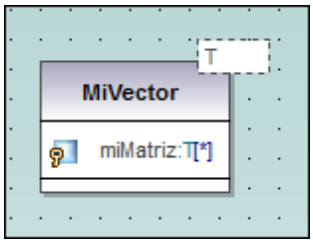
7.14 Plantillas UML

En UModel puede usar plantillas UML y crear asignaciones entre ellas y genéricos Java 5.0, C# y Visual Basic.

- Las plantillas son elementos de modelado potenciales con parámetros formales sin enlazar.
- Estos elementos de modelado parametrizados describen un grupo de elementos de modelado de un tipo concreto: clasificadores u operaciones.
- Las plantillas no se pueden usar como tipos directamente, sus parámetros deben estar enlazados.
- *Generar instancias* significa enlazar los parámetros de la plantilla con valores reales.
- Los valores reales de los parámetros son expresiones.
- El enlace que existe entre una plantilla y un elemento de modelado produce un elemento de modelado nuevo (elemento enlazado) basado en la plantilla.
- Si en C# existen varios clasificadores de restricción, los parámetros de la plantilla se pueden editar directamente en el panel Propiedades cuando se selecciona el parámetro de la plantilla.

Presentación de firmas de plantilla en UModel:

- En la imagen que aparece a continuación puede verse la plantilla de clase **`MiVector`**, cuyo parámetro de plantilla formal (`T`) aparece en la esquina superior derecha en un rectángulo discontinuo.
- Los parámetros formales sin información de tipo (`T`) son clasificadores implícitos: clase, tipo de datos, enumeración, tipo primitivo e interfaz. Los demás tipos de parámetro deben mostrarse explícitamente (p. ej. los enteros).
- La propiedad **`miMatriz`** tiene un número ilimitado de elementos de tipo `T`.

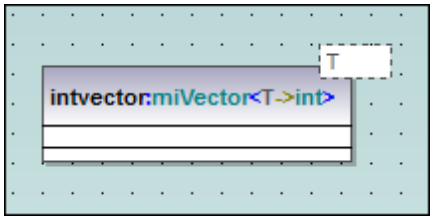


Cuando se hace clic con el botón secundario en la plantilla, aparece un menú contextual con el comando **Mostrar | Elementos enlazados**. Este comando muestra los elementos enlazados propiamente dichos.

Presentación de enlaces de plantilla en UModel:

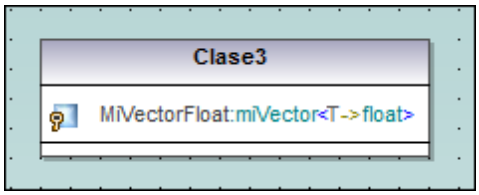
- En la imagen que aparece a continuación puede verse la plantilla con nombre enlazada **`intvector`**.
- Se trata de una plantilla de tipo **`miVector`**.
- El parámetro `T` se sustituye con `int`.
- Los caracteres `->` significan *sustituido con*.

 © 2014 Altova GmbH



Uso de plantillas en operaciones/propiedades:

- En la imagen anterior puede ver un enlace de plantilla anónimo.
- Tiene la propiedad **`MiVectorFloat`** de tipo **`MiVector<T->float>`**.



También puede definir plantillas cuando defina propiedades u operaciones. La función de finalización automática le ayudará a utilizar la sintaxis correcta.

- La operación `Operación1` devuelve un vector de tipo float.

7.14.1 Firmas de plantilla

Una *firma de plantilla* es una cadena de texto que especifica los parámetros de plantilla formales. Por su parte, una *plantilla* es un elemento parametrizado que se utiliza para generar elementos de modelado nuevos mediante la sustitución o el enlace de parámetros formales con parámetros reales (valores).

Parámetro de plantilla formal

`T`

Plantilla con un solo parámetro formal sin tipo
(almacena elementos de tipo T)

Varios parámetros de plantilla formales

`KeyType:DateType, ValueType`

Sustitución de parámetros

`T>unaClaseBase`

La sustitución de parámetros debe ser de tipo `unaClaseBase` o derivarse de ese tipo.

Valores predeterminados para parámetros de plantilla

`T=unValorPredeterminado`

Clasificadores de sustitución

`T>{contract}unaClaseBase`

`allowsSubstitutable` es `true`

El parámetro debe ser un clasificador que puede ser sustituido con el clasificador designado por el nombre de clasificador.

Parámetros de plantilla de restricción

`T:Interface>unaInterfaz`

Cuando la restricción limite a un elemento que no sea una clase (una interfaz, un tipo de datos), la restricción aparece después del carácter ":". Por ejemplo, `T` está restringido a una interfaz (`T:Interfaz`), que debe ser de tipo "`unaInterfaz`" (>`unaInterfaz`).

Usar comodines en firmas de plantilla

`T>vector<T->?<unaClaseBase>`

El parámetro de plantilla `T` debe ser de tipo "vector" que contiene objetos que son un supratipo de `unaClaseBase`.

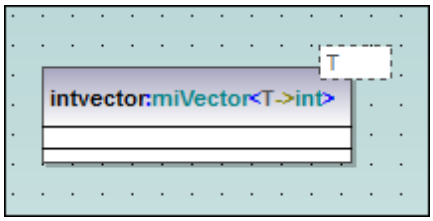
Parámetros de plantilla de extensión

`T>Comparable<T->T>`

 © 2014 Altova GmbH

7.14.2 Enlace de plantilla

Un *enlace de plantilla* es el resultado de sustituir los parámetros formales con los valores reales (es decir, se crea una instancia de la plantilla). UModel genera automáticamente clases enlazadas anónimamente cuando se produce este enlace. Los enlaces se pueden definir en el campo del nombre de la clase, como puede ver en la imagen siguiente.



Parámetros formales de sustitución/enlace

```
vector <T->int>
```

Crear enlaces usando el nombre de la clase

```
a_float_vector:vector<T->float>
```

Enlazar varias plantillas simultáneamente

```
Clase5:vector<T->int, map<KeyType->int, ValueType<T->int>
```

Usar comodines ? como parámetros (Java 5.0)

```
vector<T->?>
```

Restringir comodines - límites superiores (extensión de UModel)

```
vector<T->?>unaClaseBase>
```

Restringir comodines - límites inferiores (extensión de UModel)

```
vector<T->?<unaClaseDerivada>
```

7.14.3 Usar plantillas en operaciones y propiedades

Operación que devuelve una plantilla enlazada

```
Clase1
Operación1():vector<T->int>
```

El parámetro `T` está enlazado con `int`. La `Operación1` devuelve un vector de tipos `int`.

Clase que contiene una operación de plantilla

```
Clase1
Operación1<T>(in T):T
```

Usar comodines

```
Clase1
Propiedad1:vector<T->?>
```

Esta clase contiene un vector genérico cuyo tipo no se ha especificado (? es el comodín).

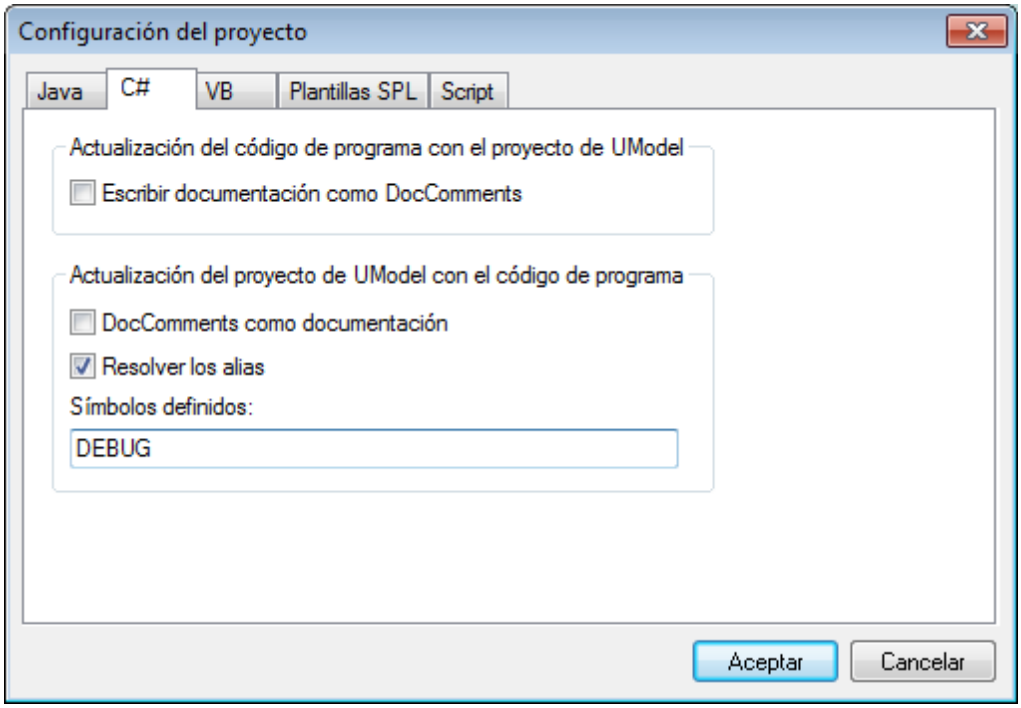
Para ver las propiedades con tipo como asociaciones:

- haga clic con el botón secundario en una propiedad y seleccione **Mostrar | PropiedadX como asociación** o
- arrastre una propiedad hasta el diagrama.

 © 2014 Altova GmbH

7.15 Configuración del proyecto

El comando **Proyecto | Configuración del proyecto** sirve para definir las opciones globales de configuración del proyecto.



Para más información consulte el apartado **Herramientas | Opciones** de la *Referencia del usuario*.

7.16 Mejorar el rendimiento

Dado que algunos proyectos de modelado pueden alcanzar un tamaño considerable, hay varias maneras de mejorar el rendimiento del modelo:

- Para empezar, compruebe que utiliza el controlador más reciente de su tarjeta gráfica.
- Deshabilite la función de color de sintaxis (en la pestaña *Estilos* cambie el valor del estilo `usar color de sintaxis` de `true` a `falso`).
- Deshabilite el color de fondo `degradado` para los diagramas. En su lugar utilice un color sólido (p. ej. seleccione el valor `blanco` para el estilo `Diagrama - color de fondo` en la pestaña *Estilos*).
- La finalización automática se habilita por defecto, pero puede desactivarla en la pestaña *Edición de diagramas* del cuadro de diálogo "Opciones locales" (**Herramientas | Opciones**). Para ello desactive la casilla *Habilitar ayudante de entrada automática*.

 © 2014 Altova GmbH

Altova UModel® 2015

Crear relaciones entre los modelos

8 Crear relaciones entre los modelos

Hay varias maneras de crear e insertar relaciones entre los modelos en los diagramas:

- Puede usar los controladores de conexión (por ejemplo, consulte el ejemplo del apartado Casos de uso).
- O puede hacer clic en el icono correspondiente de la barra de herramientas y arrastrar el puntero del mouse para crear las conexiones entre los elementos:

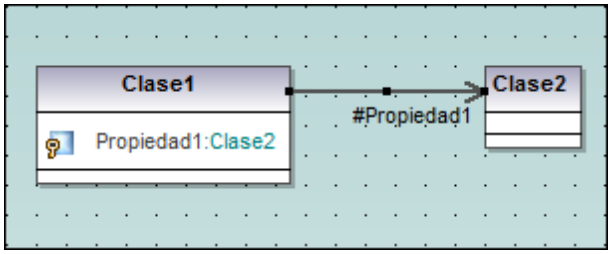
icono de asociación

agregación

composición

contención

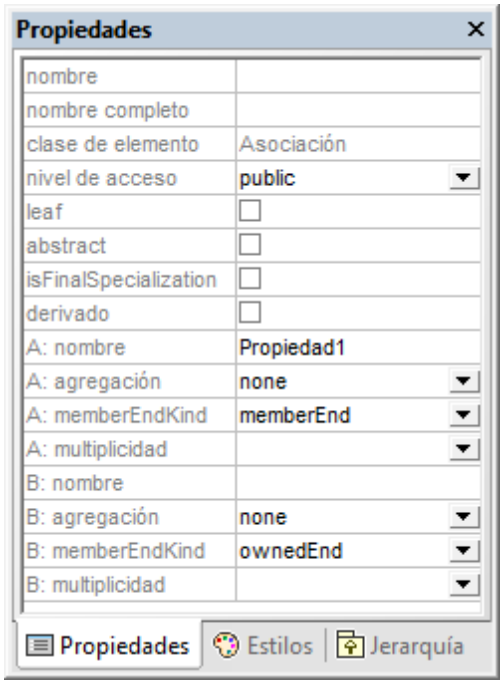
Cuando se crea una asociación, en la clase de origen (`A:nombre`) se inserta un atributo nuevo automáticamente (ver ejemplo de la siguiente imagen: `Propiedad1:Clase2`).



Una vez creada, la asociación está activa y la pestaña *Propiedades* muestra sus propiedades. El nombre predeterminado del `memberEnd` de la asociación aparece en una etiqueta de texto (es decir, **`Propiedad1`**). Recuerde que la opción **Etiqueta de texto...** del menú contextual permite ver/ocultar las etiquetas.

Al hacer clic en la línea de asociación, sus propiedades aparecen en la pestaña *Propiedades*. Las propiedades `A:Nombre` y `B:Nombre` indican qué papel desempeña cada clase en la otra.

 © 2014 Altova GmbH



Dependiendo del valor de la propiedad `memberEndKind` (de `A:nombre Propiedad1`), el atributo pertenece:

- a la clase (es decir, **A**:memberEndKind = **memberEnd**) (el atributo está visible en **`clase1`**), o
- a la asociación (es decir, **B**:memberEndKind = **ownedEnd**) (el atributo no está visible en **`clase2`**).

  Si ambos atributos pertenecen a la **asociación** (es decir si ambos extremos se definen como `ownedEnd`), entonces esta asociación es bilateral y desaparece la dirección de la flecha. Ambos extremos de la asociación son de tipo `ownedEnd`.

Si la propiedad `memberEndKind` de la asociación tiene el valor `navigableOwnedEnd`, entonces el atributo todavía forma parte de la asociación pero la flecha de dirección reaparece dependiendo del extremo en el que se defina (`A:nombre` o `B:nombre`).

Para definir el tipo de asociación (asociación, agregación o composición):

1. Haga clic en la flecha de asociación.
2. Desplácese hasta el campo `agregación` de la pestaña *Propiedades*.
3. Seleccione una opción: **none** (asociación estándar), **shared** (asociación por agregación) o **composite** (asociación por composición).

Nota: puede crear asociaciones usando la misma clase como origen y como destino. Esto recibe el nombre de *autovínculo* y hace referencia a la capacidad de un objeto para enviarse un mensaje a sí mismo, para llamadas recursivas. Haga clic en el icono de relación, después haga clic en el elemento y arrastre el puntero del mouse hasta otra parte del mismo elemento. Se crea un autovínculo.

Para ver automáticamente las asociaciones en los diagramas:

Si la opción *Crear asociaciones automáticamente* está activada en la pestaña *Edición* del cuadro de diálogo "Opciones locales " (**Herramientas | Opciones**), cuando inserte elementos en los

diagramas, las asociaciones ya existentes entre los elementos se pueden crear/ver automáticamente en el diagrama. Esto ocurre si está definido el tipo de atributo y el elemento de modelado `type` al que se hace referencia está en el diagrama.

Para eliminar relaciones/asociaciones:

1. Haga clic en la relación, bien en el área de trabajo, bien en la *Estructura del modelo*.
2. Pulse la tecla **Supr**.
   La dependencia se elimina en el diagrama y en el proyecto.

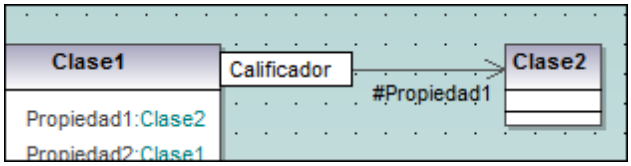
Para eliminar asociaciones de clases:

Cuando se elimina una asociación de **clases**, el **atributo**/la **propiedad** generado automáticamente no se elimina de la clase.

1. Haga clic con el botón secundario en el atributo / la propiedad de la clase.
2. Seleccione la opción **Eliminar propiedad X de la clase X**.

Para crear calificadores de asociación:

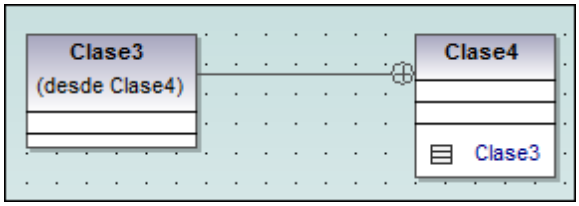
1. Defina la asociación entre las dos clases.
2. Haga clic con el botón secundario en la línea de asociación y seleccione **Nuevo/a | Calificador**.
   Recuerde que los calificadores son atributos de una asociación.



Para crear un asociación de contención:

Las asociaciones de contención muestran relaciones entre primarios y secundarios.

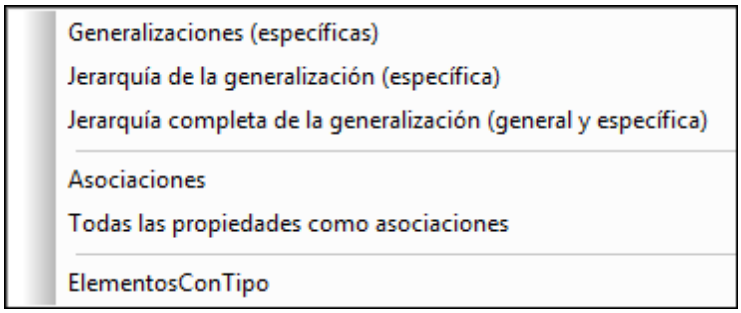
1. Haga clic en el icono **contención** de la barra de herramientas.
2. Haga clic en la clase que debe contenerse y arrastre el puntero hasta la clase que debe contenerla.
   Observe que la clase contenida (`Clase3` en este caso) ahora aparece dentro del compartimiento de `Clase4`. Por lo general, esto coloca la clase contenida en el mismo espacio de nombres que la clase contenedora.



 © 2014 Altova GmbH

8.1 Ver las relaciones entre los modelos

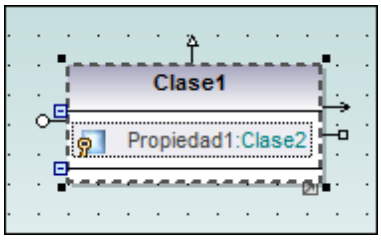
Para ver las relaciones que existen entre los elementos de modelado:
Haga clic con el botón secundario en el elemento que le interesa y seleccione **Mostrar**. Este submenú ofrece diferentes opciones dependiendo del elemento seleccionado.



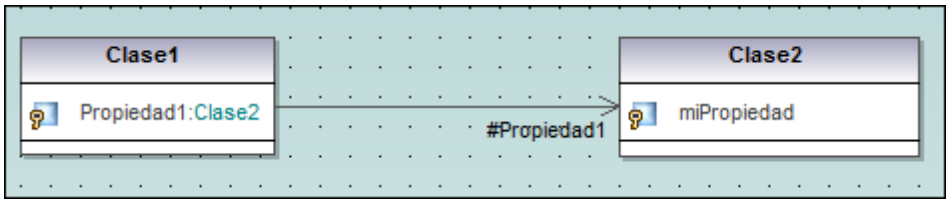
Para ver / ocultar las etiquetas de texto:
Haga clic con el botón secundario en una clase o en una flecha de asociación y seleccione **Etiquetas de texto | Mostrar (ocultar) todas las etiquetas de texto**.

Para ver un atributo / una propiedad de una clase en forma de asociación:

1. Haga clic con el botón secundario en la propiedad de la clase.



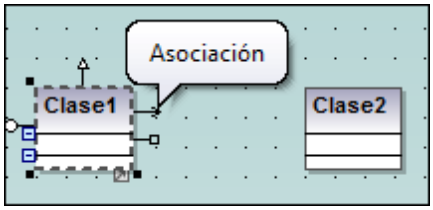
2. Seleccione la opción **Mostrar | <nombre de la propiedad> como asociación**. Esto inserta/abre la clase a la que se hace referencia y muestra la correspondiente asociación.



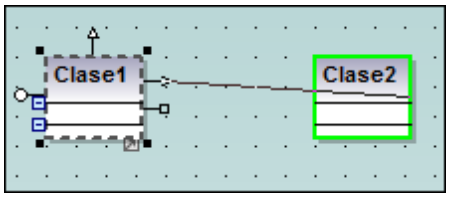
8.2 Asociaciones, realizaciones y dependencias

Para crear relaciones usando los controladores de conexión:

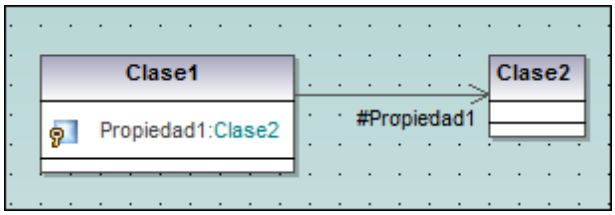
1. Imagine que tiene dos clases en un diagrama de clases.
2. Haga clic en la primera clase para seleccionarla.
   En los bordes de la clase aparecen controladores de conexión.
3. Pase el puntero del mouse por encima del controlador superior del lado derecho de la clase.



   Aparece información sobre el tipo de relación que crea este controlador. En este caso, se trata de una `asociación`.
4. Haga clic en la punta de flecha de la `asociación` y arrastre el puntero del mouse hasta la segunda clase para crear una línea de conexión.
   Si se puede crear este tipo de relación, la clase de destino se resalta. Suelte el botón del mouse.



   Se crea una asociación entre las dos clases.



   Tenga en cuenta que el controlador inferior del lado derecho de la clase crea una asociación de colección.

Los elementos de los diferentes diagramas tienen controladores de conexión distintos. Por ejemplo, las clases de los diagramas de clases incluyen estos controladores de relación (en el orden de las agujas del reloj):

- `RealizaciónDeInterfaz`
- `Generalización`
- `Asociación`

Y los artefactos de la vista **`Deployment View`** tienen estos controladores de relación:

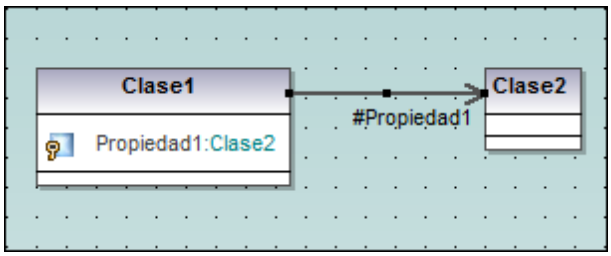
- `Manifestación`
- `Asociación`
- `Implementación`

 © 2014 Altova GmbH

Para crear relaciones con los iconos de la barra de herramientas:

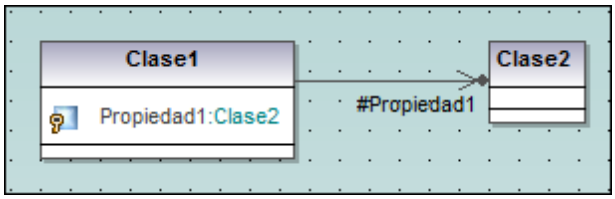
1. Imagine que tiene un diagrama de modelado con dos elementos.
2. Haga clic en el icono que representa la relación que desea crear (p. ej. `asociación`, `agregación`, `composición`, etc.).
3. Arrastre el puntero del mouse desde un objeto hasta el otro y deje de pulsar el botón del mouse cuando el segundo objeto esté resaltado.

   Cuando se crea una asociación nueva, en la clase de origen (`A:nombre`) se crea automáticamente un atributo nuevo (`Propiedad1:Clase2`) como en el ejemplo de esta imagen:



   UModel siempre muestra todos los atributos de una clase.

   Nota: las imágenes de este manual no muestran el **punto de propiedad** de la asociación.



   Para habilitarlo, asigne el valor `true` al estilo `Mostrar pto. de propiedad de la asoc.` de la pestaña *Propiedades*.

Para eliminar relaciones/asociaciones:

1. Haga clic en la relación, bien en el área de trabajo, bien en la *Estructura del modelo*.
2. Pulse la tecla **Supr**.
   La dependencia se elimina en el diagrama y en el proyecto.

Para eliminar asociaciones de clases:

Cuando se elimina una asociación de **clases**, el **atributo**/la **propiedad** generado automáticamente no se elimina en la clase.

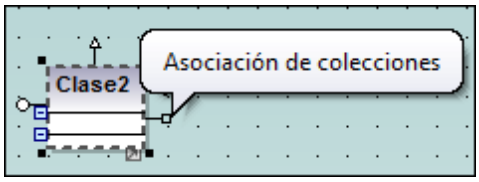
1. Haga clic con el botón secundario en el atributo / en la propiedad de la clase.
2. Seleccione la opción **Eliminar propiedad X de la clase X**.

Asociaciones de colección

UModel permite ver las asociaciones de colección de forma especial.

Las asociaciones de colección son asociaciones especiales para las plantillas de colecciones. Las propiedades de una clase (p. ej. una interfaz) se suelen mostrar como asociaciones con el `type` de la propiedad. Pero en UModel este tipo de asociación se puede ver de otra manera distinta.

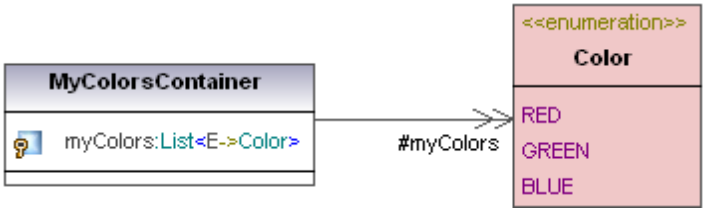
Haga clic en el controlador inferior del lado derecho de la clase y arrastre el conector hasta la clase de destino.



Al soltar el botón del mouse, aparece un menú emergente donde puede seleccionar el tipo de colección:

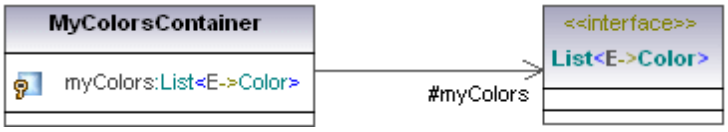
Ahora se crea un tipo distinto de flecha de asociación (con doble punta de flecha).

Ejemplo:
Imagine que durante la ingeniería inversa se crearon asociaciones automáticamente. Si en la pestaña *Edición de diagramas* del cuadro de diálogo "Opciones locales" seleccionó la opción *Resolver asociaciones a colecciones* y si están en el cuadro de diálogo "Plantillas de colecciones", entonces las asociaciones creadas durante la ingeniería inversa aparecerán como `Asociaciones de colección`.



La doble flecha indica que el tipo de **`myColors`** no sólo es **`Color`** sino una colección de varios **`Color`**.
La asociación no aparecerá como **`<E->Color>`**, sino unida directamente a la enumeración **`Color`**, ocultando el hecho de que **`Color`** se utiliza en un enlace de plantilla.

El tipo de colección específico de **`myColors`** todavía se puede ver en **`ColorsContainer`**, pero no en la asociación.



En la pestaña *Edición de diagramas* del cuadro de diálogo "Opciones locales" (**Herramientas | Opciones**) puede indicar en qué plantillas debe tener lugar este comportamiento y si quiere que las colecciones se resuelvan o no.

 © 2014 Altova GmbH

Altova UModel® 2015

Generar documentación UML

9 Generar documentación UML

Sitio web de Altova: Documentación de proyectos UML

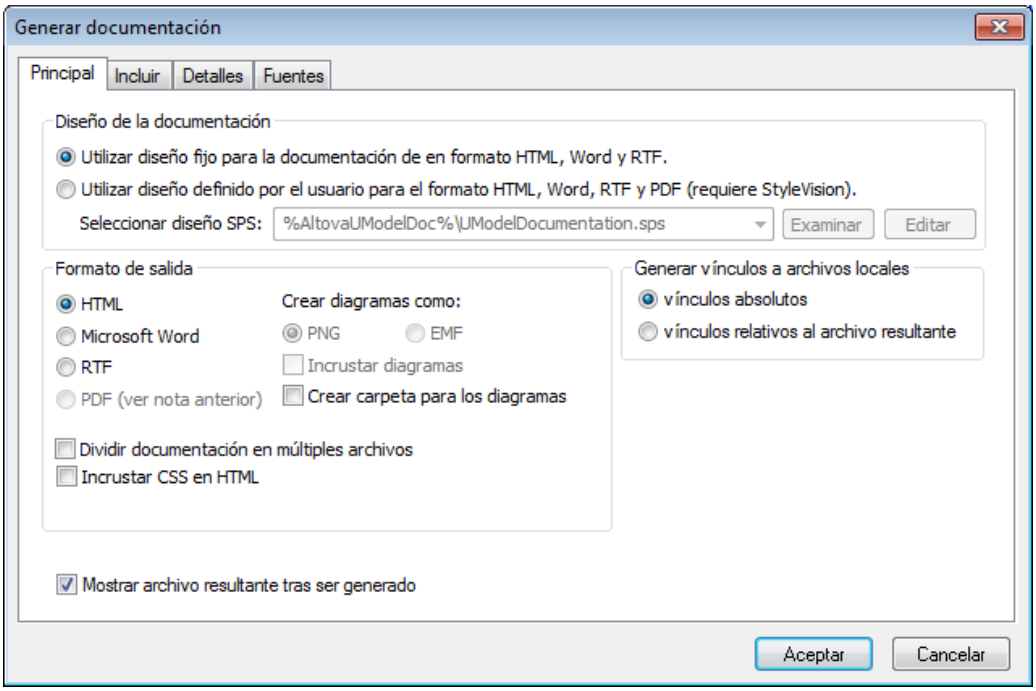
El comando **Generar documentación** genera documentación detallada sobre el proyecto UML en formato HTML, MS Word, RTF y PDF. Una vez generada, la documentación se puede modificar y utilizar sin permiso de Altova.

La documentación abarca los elementos de modelado seleccionados por el usuario en el cuadro de diálogo "Generar documentación". Además, puede generar la documentación con un diseño fijo o usar una hoja de estilos SPS de StyleVision para personalizarla. Consulte el apartado Hojas de estilos definidas por el usuario para más información.

Nota: si quiere usar una hoja de estilos SPS para generar la documentación, debe tener StyleVision instalado en el equipo. Los elementos relacionados se unen por medio de hipervínculos que permiten navegar de un componente a otro. Para generar documentación en formato Word, necesita tener instalado MS Word 2000 (o superior).

Para este ejemplo utilizaremos el proyecto de muestra `Bank_MultiLanguage.ump`:

1. Abra el proyecto en UModel y seleccione la opción de menú **Proyecto | Generar documentación**.
   Esto abre el cuadro de diálogo "Generar documentación" (*imagen siguiente*) en su configuración predeterminada.



Recuerde que también puede generar documentación de determinados elementos de modelado por separado. Esto se consigue haciendo clic con el botón secundario en uno o varios elementos en la Estructura del modelo y seleccionando **Generar documentación** en el menú contextual. El elemento puede ser una carpeta, una clase, una interfaz, etc. En cualquier caso, las opciones de documentación son las mismas en ambos casos.

 © 2014 Altova GmbH

Los elementos relacionados se unen por medio de hipervínculos que permiten navegar por la documentación de un componente a otro. En la documentación también aparecen todos los hipervínculos creados a mano.

Nota: también se genera documentación para los subproyectos (perfiles) C#, Java y VB **incluidos**, si estos se habilitan en la pantalla *Incluir* del cuadro de diálogo "Generar documentación".

Pestaña *Principal*
A continuación describimos las opciones de esta pestaña.

Diseño de la documentación

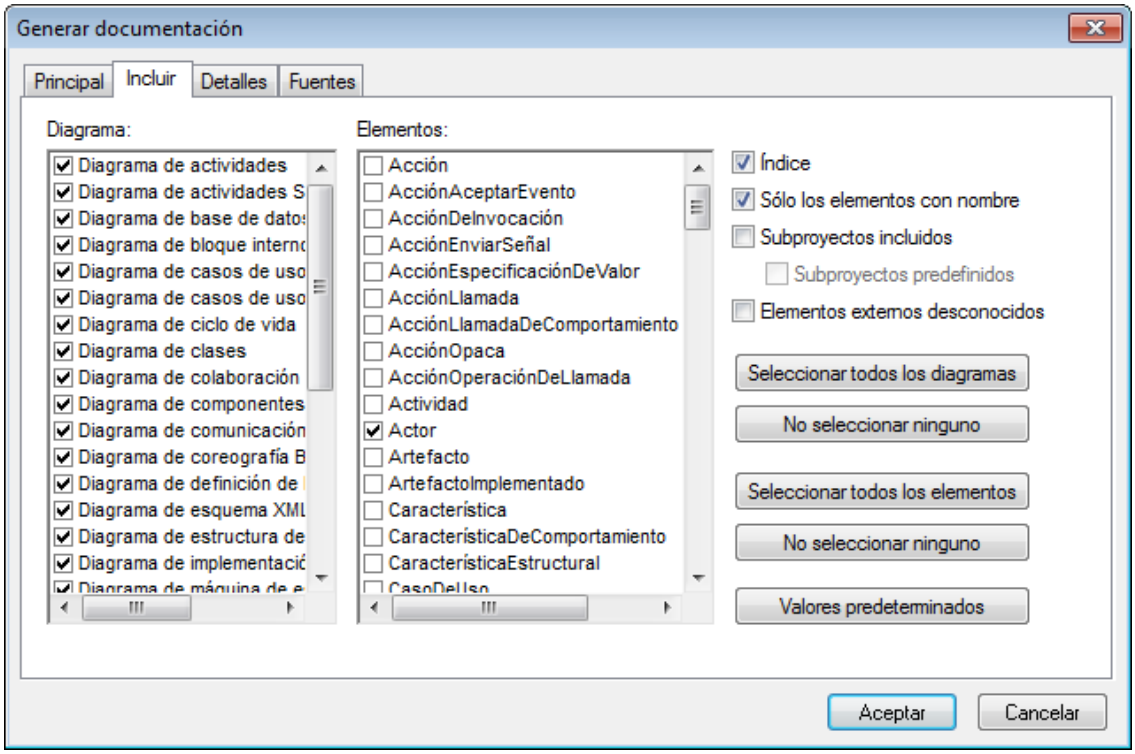
- *Utilizar diseño fijo para....* seleccione esta opción si quiere usar la plantilla integrada para generar la documentación.
- *Utilizar diseño definido por el usuario...* seleccione esta opción si quiere usar una hoja de estilos de StyleVision predefinida. Los archivos SPS están disponibles en la carpeta `...\Mis Documentos\Altova\UModel2015\Documentation\UModel\`.
- **Examinar:** haga clic en este botón para buscar el archivo SPS predefinido.
- **Editar:** haga clic en este botón para abrir el SPS seleccionado en StyleVision.

Formato de salida:

- Aquí se selecciona el formato de salida: *HTML*, *Word*, *RTF* o *PDF*.
  Los documentos de **Microsoft Word** se crean con la extensión de archivo `.doc` si utiliza un diseño fijo y con la extensión `.docx` si utiliza un archivo SPS.
  El formato **PDF** solamente está disponible si usa un archivo SPS para generar la documentación.
- *Dividir documentación en múltiples archivos:* marque esta casilla si quiere que UModel genere un archivo de salida por cada elemento de modelado de la tabla de contenido (por ejemplo, si hay una clase llamada `C1` que tiene una clase anidada llamada `CNest`, el archivo `C1.html` contiene toda la información sobre `C1` y `CNest`, así como sus atributos, propiedades, etc.).
- *Incrustar CSS en HTML:* marque esta casilla para incrustar un archivo CSS ya existente en la documentación HTML. Si quiere que la referencia al archivo y el archivo CSS propiamente dicho sean externos, desactive esta casilla.
- *Incrustar diagramas:* esta casilla se habilita si el formato de salida elegido es *Microsoft Word* o *RTF*. Si marca esta casilla, los diagramas se incrustan en el archivo generado. Los diagramas se crean como archivo PNG (para HTML) o como archivos PNG/EMF (para MS Word y RTF) y en el archivo de resultado se presentan a través de vínculos de objeto.
- *Crear carpeta para los diagramas:* seleccione esta opción para crear una subcarpeta dentro de la carpeta de salida seleccionada. En esta subcarpeta se guardan todos los diagramas.
- *Mostrar archivo resultante tras ser generado*: esta opción está disponible con todos los formatos de salida. Selecciónela para ver los archivos resultantes en la vista Explorador (para HTML), en MS Word (para formato Word) o en la aplicación predeterminada para archivos RTF y PDF (para RTF y PDF respectivamente).

Generar vínculos a archivos locales
En este grupo de opciones puede indicar si los vínculos generados deben ser absolutos o relativos al archivo de salida.

Pestaña *Incluir*
En esta pestaña (*imagen siguiente*) puede seleccionar qué diagramas y elementos de modelado debe abarcar la documentación.

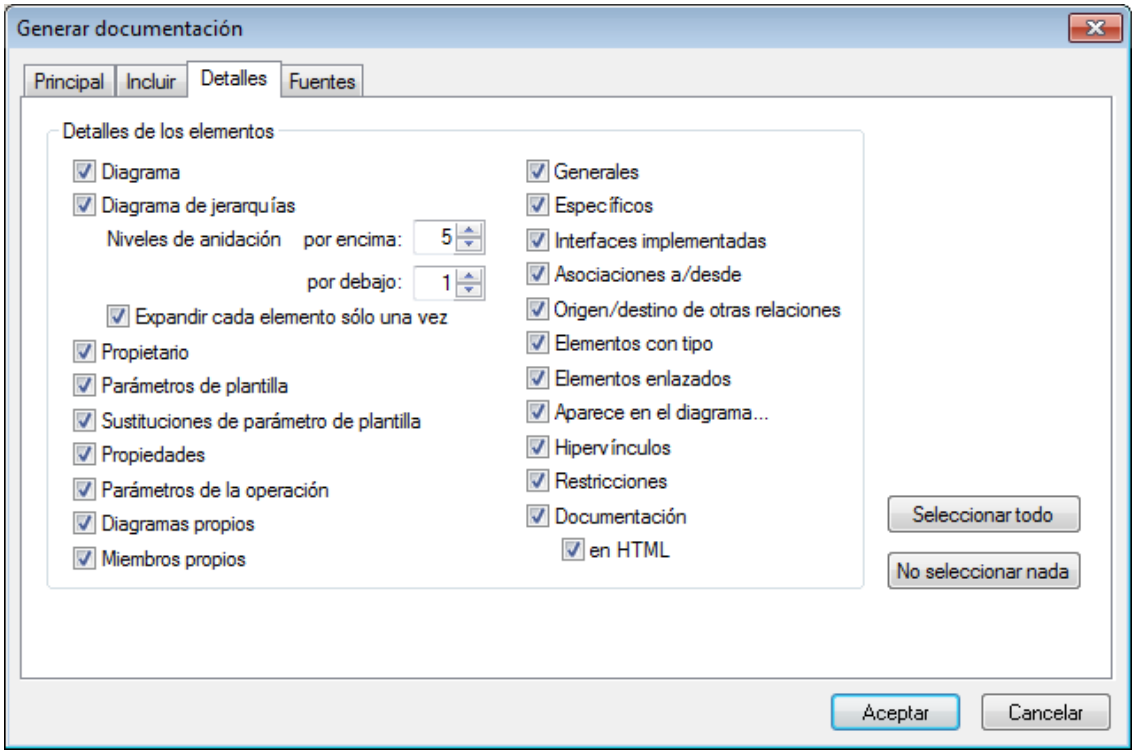


Recuerde que puede deshabilitar la documentación de subproyectos desactivando la casilla *Subproyectos incluidos*.

Pestaña *Detalles*
En esta pestaña (*imagen siguiente*) puede seleccionar los detalles de los elementos que se deben incluir en la documentación.

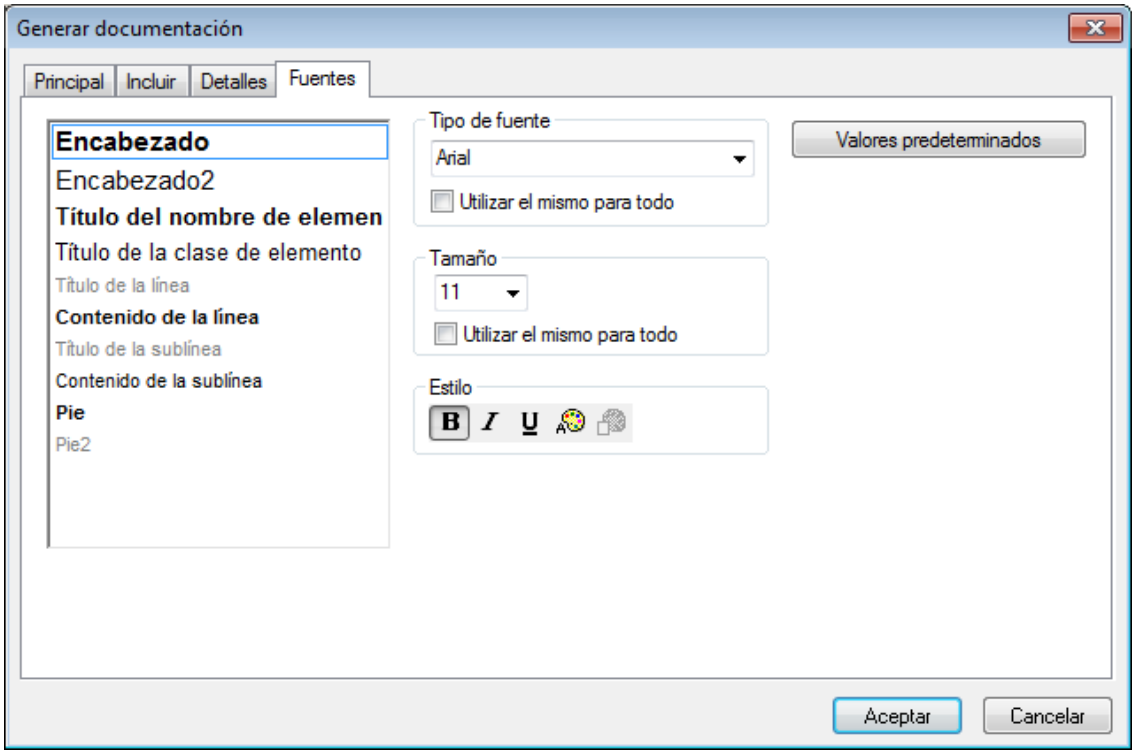
- Si tiene pensado importar el texto de **etiquetas XML** en la documentación, desactive la casilla *en HTML* situada debajo de la opción *Documentación*.
- En los cuadros combinados *por encima* y *por debajo* puede definir cuántos niveles se debe anidar la clase actual en el diagrama de jerarquías.
- Marque la casilla *Expandir cada elemento solo una vez* si quiere que solo un clasificador de cada tipo se pueda expandir en la misma imagen/en el mismo diagrama.

 © 2014 Altova GmbH



Pestaña *Fuentes*

En esta pestaña (*imagen siguiente*) puede configurar la fuente utilizada para el contenido de texto y para los títulos y encabezados de la documentación.



© 2014 Altova GmbH

Las imágenes que aparecen a continuación muestran la documentación generada para el archivo `Bank_MultiLanguage.ump` (directorio `...\UModelExamples`) utilizando el diseño fijo.

La primera imagen muestra la documentación generada con los índices de diagramas y de elementos al principio del archivo HTML.

La segunda imagen muestra los detalles de la clase **`Account`** y sus relaciones con otras clases. Observe que los atributos y propiedades de los diagramas de clases también están unidos a sus definiciones por medio de hipervínculos. Haga clic en una propiedad para ver su definición. Las clases de la jerarquía también incluyen hipervínculos (así como el texto subrayado).

Bank_MultiLanguage.ump

ubicación del proyecto C:\Documents\Altova\UModel2014\UModelExamples\Bank_MultiLanguage.ump

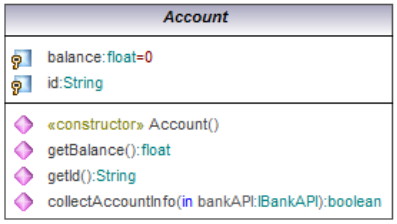
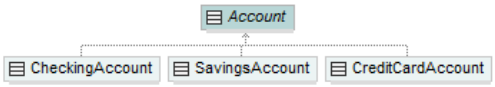
Índice de diagramas:

Diagrama de actividades	collectData Draft	
Diagrama de procesos de negocio BPMN 1	DiagramadeprocesosdenegocioBPMN11	
Diagrama de componentes	Bank realizations	Overview
Diagrama de estructura de un compuesto	Account Transfer	
Diagrama de implementación	Deployment	
Diagrama de secuencia	Collect Account Information	Connect to BankAPI
Diagrama de máquina de estados	BankAPI Draft	Query BankServer Draft

Índice de elementos:

Actividad	Actividad1	BankView	
Artefacto	BankAddresses.ini	BankAPI.jar	BankServer
Colaboración	Account Transfer	Bank Account Transfer	
Componente	BankView GUI		
Interacción	Collect Account Information	Connect to BankAPI	
Nodo	Bank	Home PC	My Home PC
Paquete	Account Transfer Deployment View Root	BankView Design View	Behavior View Interaction View
MáquinaDeEstados	BankAPI	BankServer	

Clase **Account**

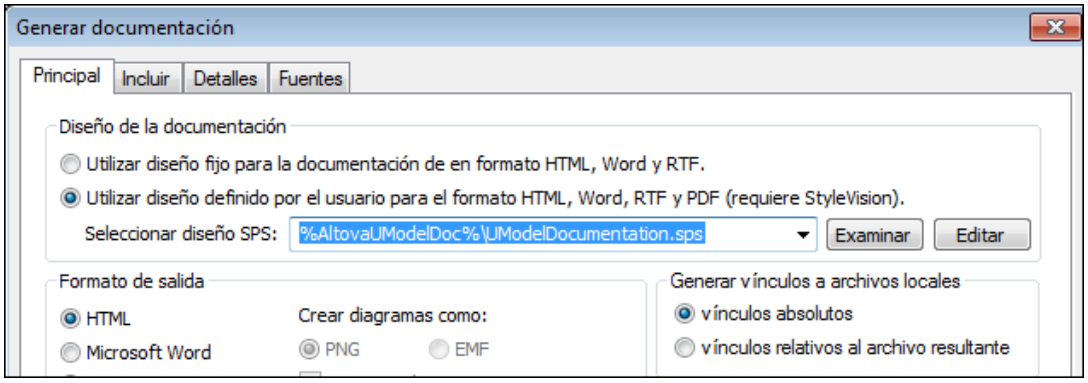
diagrama	
jerarquía	
propietario	**bankview**
propiedades	nombre completo Design View::BankView::com::altova::bankview::Account nivel de acceso public leaf false abstract true isFinalSpecialization false activo false nombre del archivo de código Account.java ruta de acceso del archivo de código C:\Documents\Altova\UModel2014\UModelExamples\Bank_MultiLanguage_Java\Src\Account.java «annotations» false «static» false «strictfp» false
miembroPropio	**Account** **balance** **collectAccountInfo** **getBalance** **getId** **id**
específico	**CheckingAccount** **CreditCardAccount** **SavingsAccount**
asociaciones desde	desde Nombre de la asociación **Bank** **(accounts)**
destino de la relación	RealizaciónDeComponente **BankView**
elementosConTipo	Clase **Bank** Propiedad **accounts** Interacción **Collect Account Information** Propiedad **b**
aparece en el diagrama	**BankView Main** **Hierarchy of Account**

 © 2014 Altova GmbH

9.1 Con una hoja de estilos SPS predeterminada

Para generar documentación con la hojas de estilo SPS predeterminada:

1. Seleccione el comando de menú **Proyecto | Generar documentación.**
2. En el cuadro de diálogo "Generar documentación" seleccione el botón de opción *Utilizar diseño definido por el usuario...* (en la pestaña *Principal*).
3. Seleccione la hoja de estilos `UModelDocumentation.sps` (si no está seleccionada todavía).



   Aparece un aviso pidiendo que guarde el archivo.
4. Escriba el nombre del archivo y seleccione la ubicación donde desea guardarlo.

En la documentación generada (*imagen siguiente*) haga clic en un vínculo para navegar al elemento de modelado.

Bank_MultiLanguage.ump

ubicación del proyecto C:\Documents\Altova\UModel2014\UModelEx

Índice de diagramas:

Diagrama de actividades	collectData Draft
Diagrama de procesos de negocio BPMN 1	DiagramadeprocesosdenegocioBPMN11
Diagrama de componentes	Bank realizations
Diagrama de estructura de un compuesto	Account Transfer
Diagrama de implementación	Deployment
Diagrama de secuencia	Collect Account Information
Diagrama de máquina de estados	BankAPI Draft

Índice de elementos:

Actividad	Actividad1
Artefacto	BankAddresses.ini

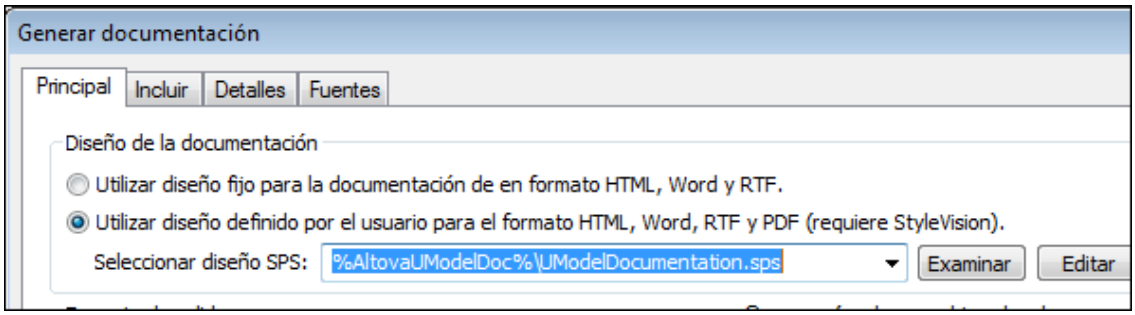
© 2014 Altova GmbH

9.2 Con hojas de estilos predefinidas por el usuario

Si lo prefiere, en lugar de usar el diseño fijo o la hoja de estilos SPS predeterminada, puede crear un diseño personal para su documentación UML. El diseño se crea en una hoja de estilos SPS de StyleVision. Recuerde que también puede editar la hoja de estilos SPS que viene con UModel.

Seleccionar el diseño SPS para generar la documentación

Debe especificar qué diseño SPS se utiliza para generar la documentación en el cuadro de diálogo "Generar documentación" (**Proyecto | Generar documentación**). Seleccione el botón de opción *Utilizar diseño definido por el usuario...* y después seleccione un archivo SPS en el cuadro combinado. Si el SPS no está en el cuadro combinado, haga clic en el botón **Examinar** para buscarlo. El SPS predeterminado disponible en este cuadro combinado es `UModelDocumentation.sps`, que está en la carpeta `...\Documentation\UModel`.



Nota: para poder generar la documentación usando un SPS necesita tener StyleVision instalado en el equipo.

¿Cómo se crea una hoja de estilos SPS?

Las hojas de estilos Power Stylesheet de StyleVision (o archivos SPS) se crean con la aplicación Altova StyleVision. El SPS utilizado para generar documentación UML de UModel debe estar basado en un esquema XML que especifique la estructura del documento XML que almacenará la documentación de UModel.

Se trata del esquema `UModelDocumentation.xsd` que viene con el paquete de instalación de UModel y está disponible en la carpeta `...\Mis Documentos\Altova\UModel2015\Documentation\UModel`.

Para crear el diseño SPS en StyleVision, coloque los nodos del esquema `UModelDocumentation.xsd` en el diseño y asígneles estilos y propiedades. Recuerde que el esquema `UModelDocumentation.xsd` **ya incluye** el archivo `Documentation.xsd` situado en la carpeta superior.

También puede añadir al diseño componentes adicionales, como vínculos e imágenes. Para más información consulte el *Manual del usuario de Altova StyleVision*.

La ventaja de usar diseños SPS para generar la documentación es que permite un mayor control sobre el diseño. Además recuerde que el formato de salida PDF solamente está disponible si utiliza un diseño SPS para generar la documentación. En otras palabras, la documentación no se puede generar en formato PDF si usa el diseño fijo.

Altova UModel® 2015

Diagramas UML

10 Diagramas UML

Hay dos grandes tipos de diagramas UML: (i) los diagramas de estructura, que muestran la vista estática del modelo, y (ii) los diagramas de comportamiento, que muestran su vista dinámica. UModel es compatible con los 14 tipos de diagramas de la especificación UML 2.4 y con los diagramas BPMN y diagramas de esquema XML.

- Diagramas de comportamiento: actividades, máquina de estados, máquina de estados de protocolos, casos de uso, interacción, comunicación, interacción global, secuencia y ciclo de vida.
- Diagramas de estructura: clases, estructura de un compuesto, componentes, implementación, objetos y paquetes.
- Otros diagramas: diagramas de esquema XML, diagramas BPMN (Business Processing Modeling Notation), diagramas SysML y diagramas de base de datos.

Nota: en la mayoría de los diagramas de modelado puede pulsar **Ctrl+Entrar** para crear una línea nueva en el nombre de algunos elementos (por ejemplo, en las etiquetas de las líneas de vida de los diagramas de secuencia y de ciclo de vida o en las condiciones de protección, nombres de estado y nombres de actividades).

 © 2014 Altova GmbH

10.1 Diagramas de comportamiento

Este tipo de diagramas ilustran las características de la conducta de un sistema o proceso de negocio e incluye un subconjunto de diagramas que subrayan cómo interactúan los objetos.

Diagramas de comportamiento

- Diagrama de actividades
- Diagrama de máquina de estados
- Diagrama de máquina de estados de protocolos
- Diagrama de casos de uso

El subconjunto de diagramas de comportamiento que ilustran cómo interactúan los objetos está compuesto por estos diagramas:

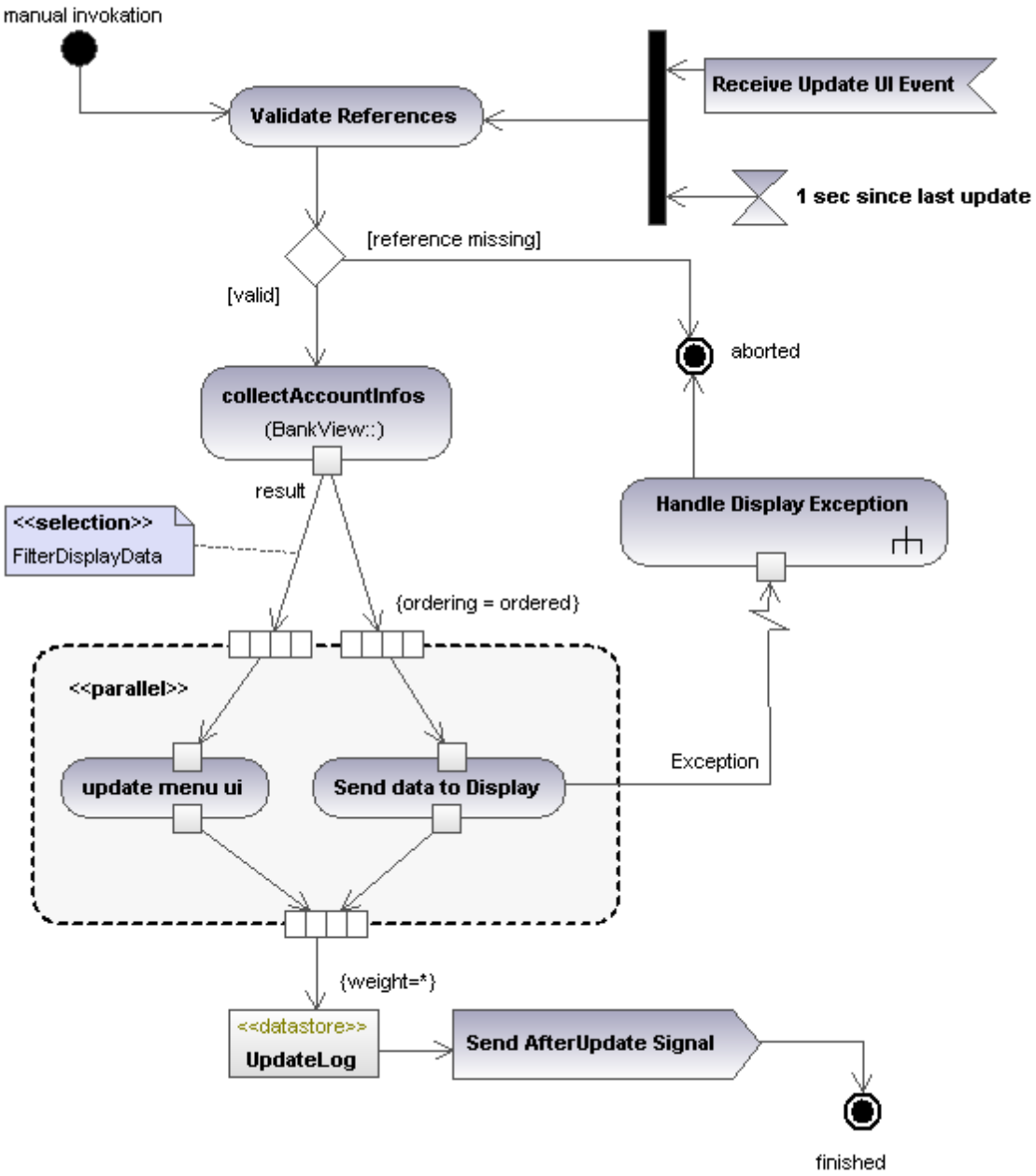
- Diagrama de comunicación
- Diagrama global de interacción
- Diagrama de secuencia
- Diagrama de ciclo de vida

10.1.1 Diagrama de actividades

Sitio web de Altova: Diagramas de actividades UML

Los diagramas de actividades sirven para modelar flujos de trabajo de procesos de negocios. Permiten ver qué acciones deben tener lugar y qué dependencias de comportamiento existen. Los diagramas de actividades describen el orden concreto de las actividades y permiten un procesamiento tanto condicional como en paralelo. Los diagramas de actividades son una especie de diagrama de estados, con actividades en lugar de estados.

Nota: el diagrama de actividades que aparece a continuación está disponible en el ejemplo `Bank_MultiLanguage.ump`, en la carpeta `...\UModelExamples`.



 © 2014 Altova GmbH

Insertar elementos

Iconos de las barras de herramientas:

1. Haga clic en el icono pertinente de la barra de herramientas Diagrama de actividades.
2. Ahora haga clic en el área de trabajo del diagrama para insertar el elemento.
   Recuerde que si mantiene pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo podrá insertar varios elementos del tipo seleccionado.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de actividades.

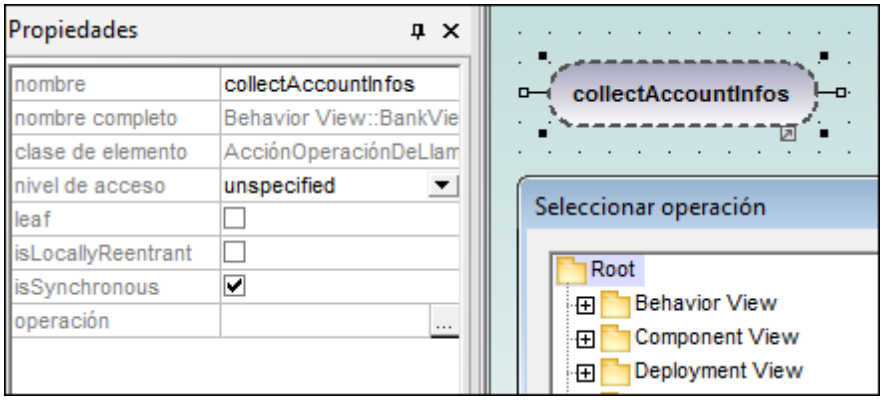
Insertar una acción (ComportamientoDeLlamada):

1. Haga clic en el icono **Acción (ComportamientoDeLlamada)** de la barra de herramientas y haga clic en el diagrama de actividades para insertar la acción.
2. Escriba el nombre de la Acción (p. ej. `Validar referencias`) y pulse **Entrar** para confirmar.

Nota: pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la acción.

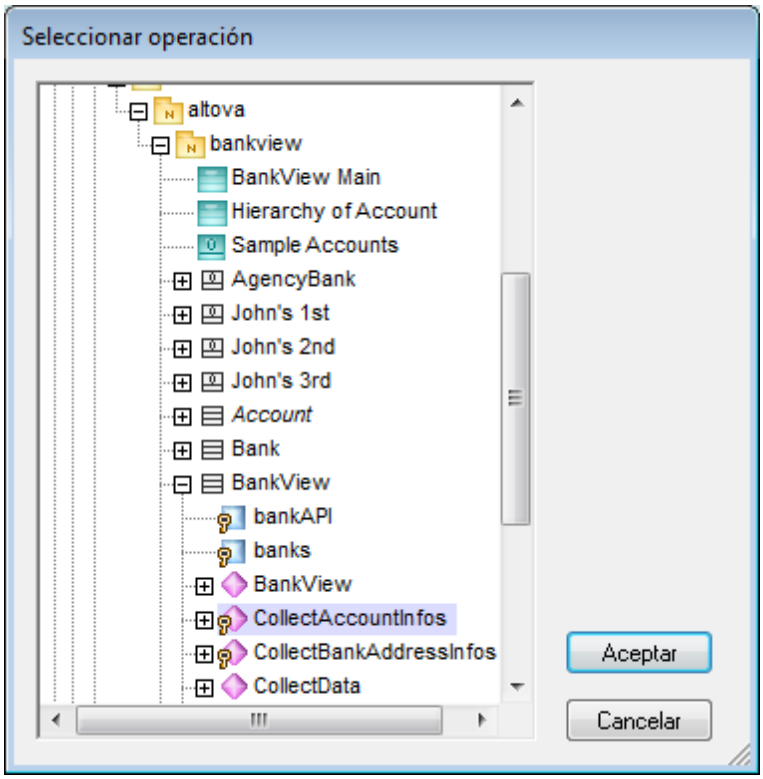
Insertar una acción (OperaciónDeLlamada) y seleccionar una operación determinada:

1. Haga clic en el icono **Acción (OperaciónDeLlamada)** de la barra de herramientas y haga clic en el diagrama de actividades para insertar la acción.
2. Escriba el nombre de la Acción (p. ej. `collectAccountInfos`) y pulse **Entrar** para confirmar.
3. En el panel Propiedades haga clic en el botón **Examinar** del campo `operation`.



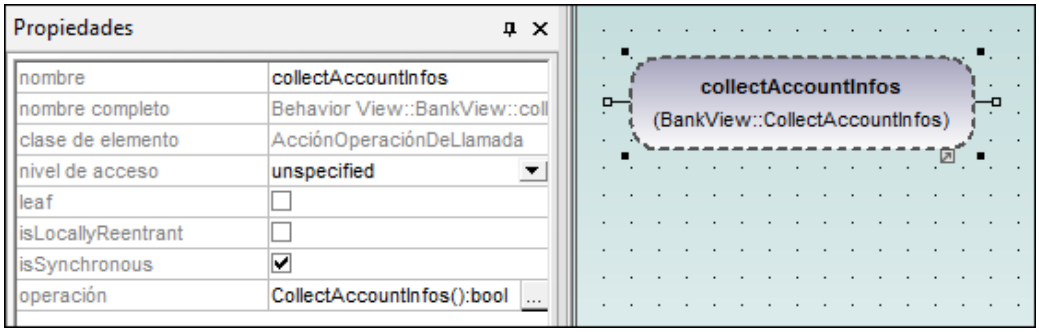
Esto abre el cuadro de diálogo "Seleccionar operación", donde puede seleccionar una operación determinada.

4. Navegue hasta la operación que desea insertar y haga clic en **Aceptar** para confirmar.



Para este ejemplo seleccionamos la operación `collectAccountInfos` de la clase **`BankView`**.

 © 2014 Altova GmbH



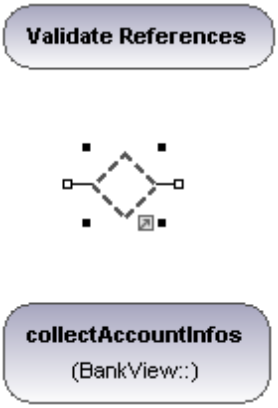
Crear bifurcaciones y convergencias

Crear una rama (flujo alterno)
Una rama tiene un flujo de entrada y varios flujos de salida protegidos por guardas. Solo se puede recorrer uno de esos flujos de salida, así que los guardas deben excluirse mutuamente.

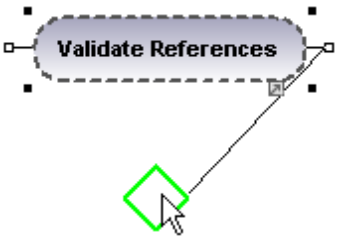
En el ejemplo que utilizamos a continuación vamos a validar las referencias de **`BankView`**:

- la rama1 tiene el guarda `reference missing`, que pasa a la actividad `abort`.
- la rama2 tiene el guarda `valid`, que pasa a la actividad `collectAccountInfos`.

1. Haga clic en el icono **NodoDeDecisión** de la barra de herramientas y haga clic en el área de trabajo del diagrama de actividades.

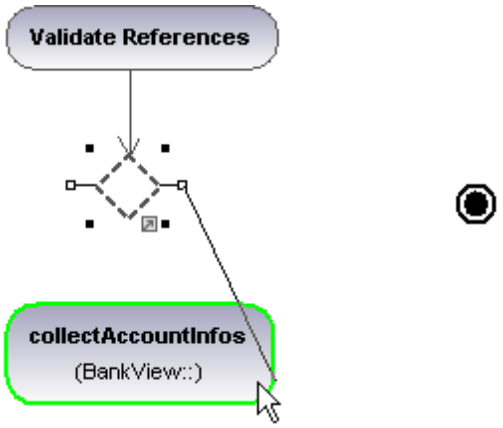


2. Haga clic en el icono **NodoFinalDeActividad**, que representa la actividad `abort`, e insértelo en el diagrama de actividades.
3. Haga clic en la actividad `Validate References` y después haga clic en su conector derecho (el controlador `FlujoDeControl`). Ahora arrastre el conector hasta el elemento `NodoDeDecisión`.

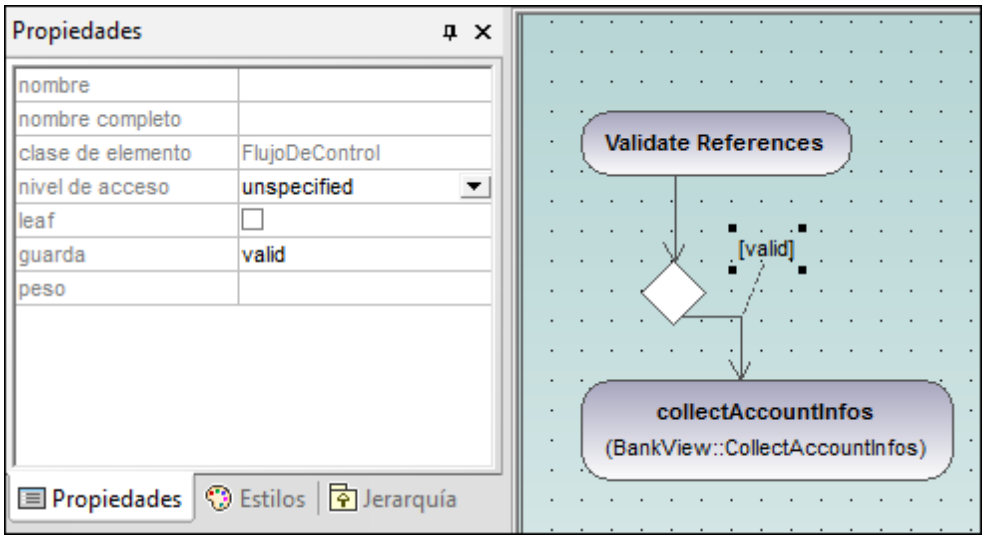


El elemento se resalta cuando sea posible colocar el conector.

4. Haga clic en el elemento `NodoDeDecisión` y después en su conector derecho (el controlador `FlujoDeControl`). Arrástrelo hasta la acción `collectAccountInfos`. Consulte el apartado Insertar una acción (OperaciónDeLlamada) para obtener más información.

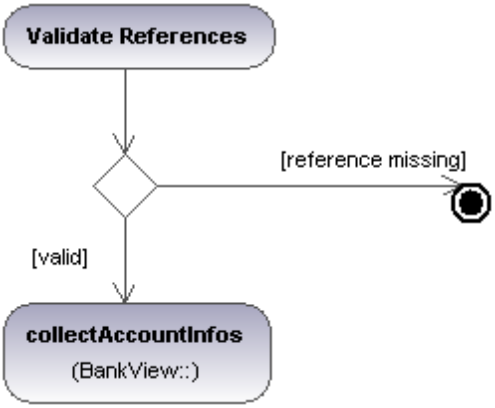


5. En el panel Propiedades seleccione el valor `valid` para la propiedad `guarda`.



6. Haga clic en el elemento **`NodoDeDecisión`** y después en su conector derecho (el controlador `FlujoDeControl`). Arrástrelo hasta el elemento **`NodoFinalDeActividad`**. La condición de guarda de esta transición se define automáticamente como "else". Haga doble clic en la condición de guarda del diagrama para cambiarla por "reference missing".

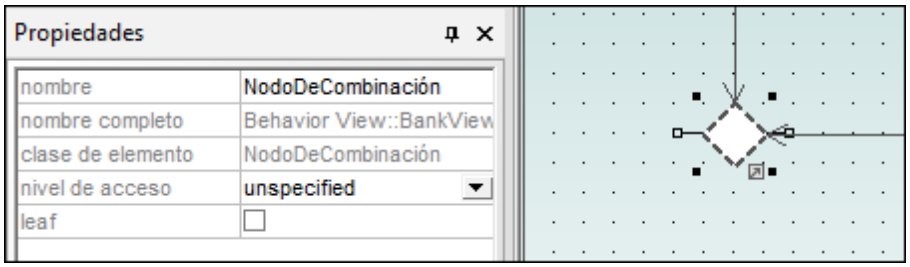
 © 2014 Altova GmbH



Nota: recuerde que UModel no valida ni revisa el número flujos de control/objetos del diagrama.

Crear una combinación

1. Haga clic en el icono **NodoDeCombinación** de la barra de herramientas y después haga clic en el diagrama para insertarlo.



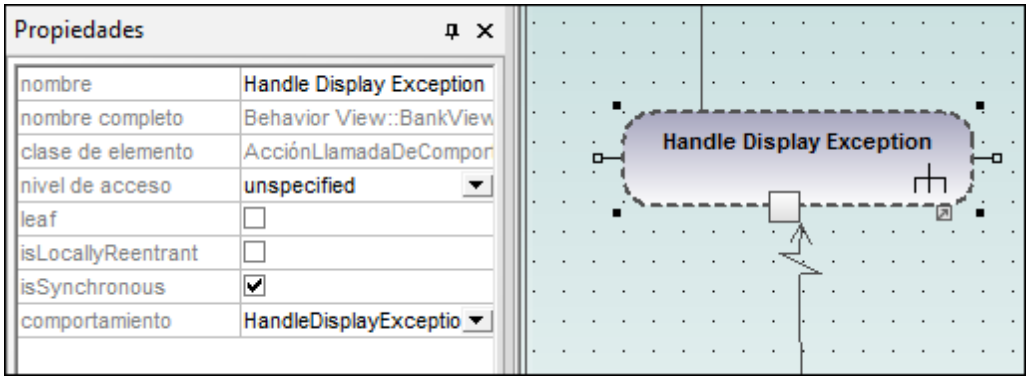
2. Haga clic en el conector `FlujoDeControl` (`FlujoDeObjetos`) de las acciones que desea combinar y arrástrelas hasta el símbolo del `NodoDeCombinación`.

Elementos

Acción (ComportamientoDeLlamada)

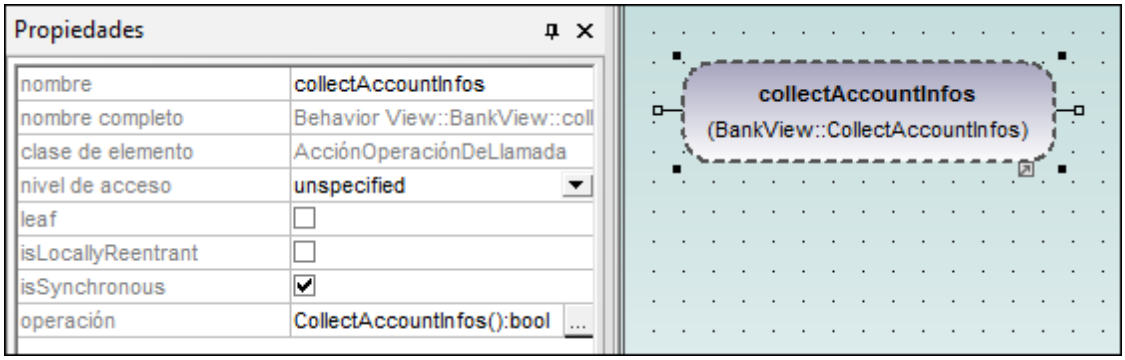
Inserta el elemento **`AcciónComportamientoDeLlamada`**, que invoca un comportamiento concreto directamente.

Si selecciona un comportamiento ya existente desde el cuadro combinado `comportamiento` (del panel Propiedades), en el elemento aparece un icono en forma de rastrillo.



Acción (OperaciónDeLlamada)

Inserta el elemento **`AcciónOperaciónDeLlamada`**, que invoca un comportamiento concreto como método directamente. Para más información consulte el apartado Insertar una acción (OperaciónDeLlamada).



Acción (AcciónOpaca)

Un tipo de acción utilizada para especificar la información de implementación. Se puede usar como marcador de posición hasta que decida qué tipo de acción desea utilizar.

Acción (AcciónEspecificaciónDeValor)

Un tipo de acción que evalúa (o genera) un valor determinado en el pin de salida. Viene definido por las propiedades (p. ej. `upperBound` para el límite superior.)

AcciónAceptarEvento

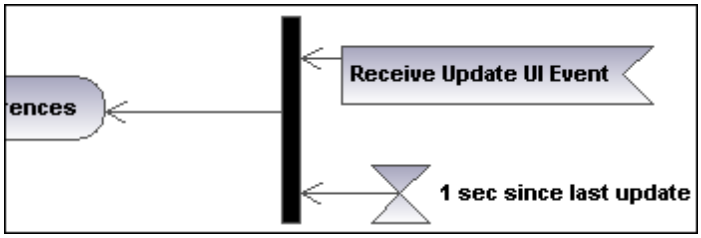
Inserta la acción **`AceptarEvento`**, que espera que tenga lugar un evento que cumpla determinados requisitos.

AcciónAceptarEvento (EventoDeTiempo)

Inserta una acción **`AceptarEvento`**, que está desencadenada por un evento de tiempo (un

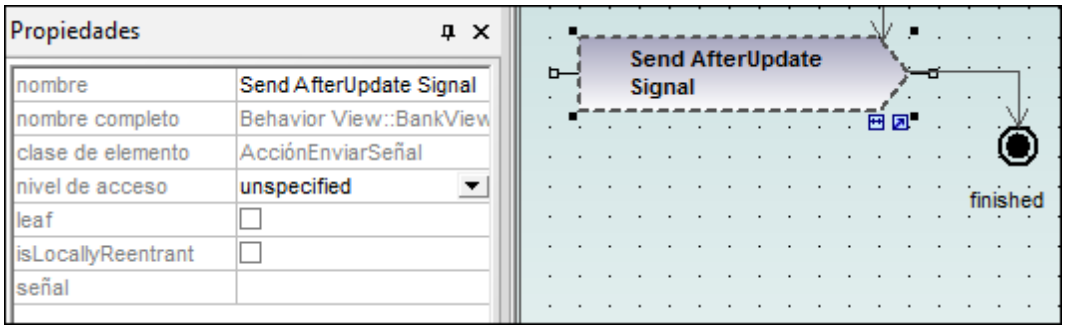
 © 2014 Altova GmbH

instante de tiempo que viene dado por una expresión).



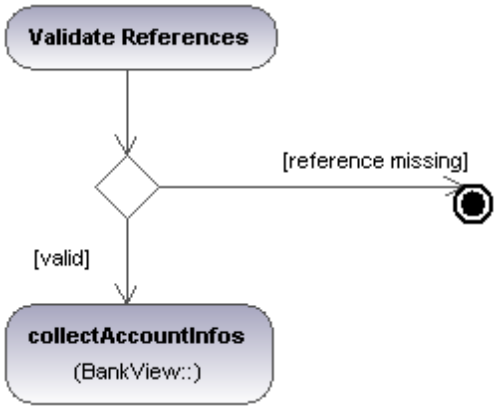
AcciónEnviarSeñal

Inserta la acción `EnviarSeñal`, que crea una señal desde sus entradas y transmite la señal al objeto de destino, donde puede provocar la ejecución de una actividad.



NodoDeDecisión

Inserta un `NodoDeDecisión`, que tiene una sola transición de entrada y varias transiciones de salida protegidas por guardas. Para más información consulte el apartado Crear ramas.



NodoDeCombinación

Inserta un `NodoDeCombinación`, que combina varias transiciones alternas definidas por el `NodoDeDecisión`. El `NodoDeCombinación` no sincroniza procesos simultáneos, sino que selecciona uno de los procesos.

NodoInicial

Se trata del comienzo del proceso de actividades. Una actividad puede tener varios nodos iniciales.

NodoFinalDeActividad

Se trata del final del proceso de actividades. Una actividad puede tener varios nodos finales y todos los flujos de la actividad se detienen cuando se encuentra el primer nodo final.

NodoFinalDeFlujo

Inserta un `NodoFinalDeFlujo`, que termina un flujo pero no los demás flujos de la actividad.

NodoDeBifurcación

Inserta un nodo de bifurcación vertical. Sirve para dividir los flujos en varios flujos simultáneos.

NodoDeBifurcación (Horizontal)

Inserta un nodo de bifurcación horizontal. Sirve para dividir flujos en varios flujos simultáneos.

NodoDeReunión

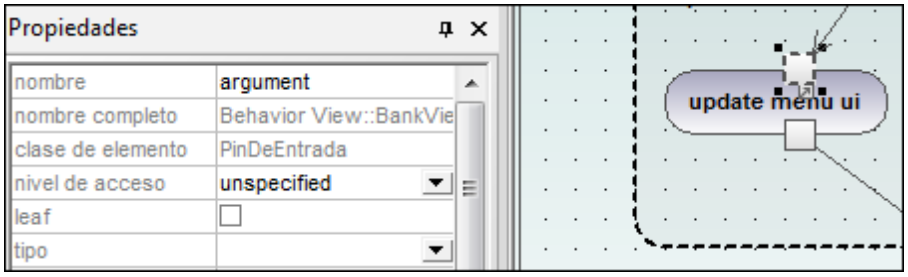
Inserta un nodo de reunión vertical. Sirve para sincronizar varios flujos definidos por un nodo de bifurcación.

NodoDeReunión (horizontal)

Inserta un nodo de reunión horizontal. Sirve para sincronizar varios flujos definidos por un nodo de bifurcación.

PinDeEntrada

Inserta un pin de entrada en un `ComportamientoDeLlamada` o en una `OperaciónDeLlamada`. Los pins de entrada aportan los valores de entrada que utiliza la acción. A los pins de entrada se les asigna un nombre predeterminado (argumento) de forma automática.

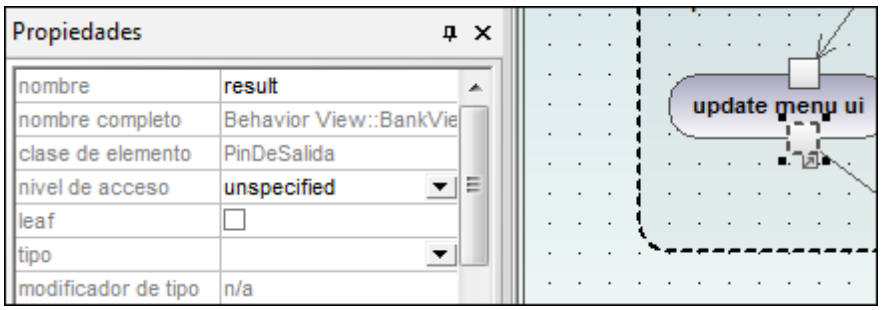


 © 2014 Altova GmbH

Si al arrastrar el pin de entrada sobre un elemento el puntero se convierte en este símbolo , significa que puede colocar el pin de entrada en el elemento.

PinDeSalida

Inserta un pin de salida. Los pins de salida contienen los valores de salida que produce una acción. Al pin de salida se le asigna automáticamente un nombre equivalente a la propiedad UML de la acción (p. ej. `result`).



Si al arrastrar el pin de salida sobre un elemento el puntero se convierte en este símbolo , significa que puede colocar el pin de salida en el elemento.

Pin de Excepción

Puede convertir un **`PinDeSalida`** en un pin de **`Excepción`** haciendo clic en el pin y seleccionando `esPinDeExcepción` en el panel Propiedades.

PinDeValor

Inserta un pin de valor que es un pin de entrada que aporta un valor a una acción que no viene de un flujo de objeto de entrada. Se representa con el símbolo de un pin de entrada y tiene las mismas propiedades que un pin de entrada.

NodoDeObjeto

Inserta un nodo de objeto que es un nodo de actividad abstracto que define el flujo de objeto de una actividad. Los nodos de objeto solo contiene valores en tiempo de ejecución que se ajustan al tipo del nodo de objeto.

NodoDeBúferCentral

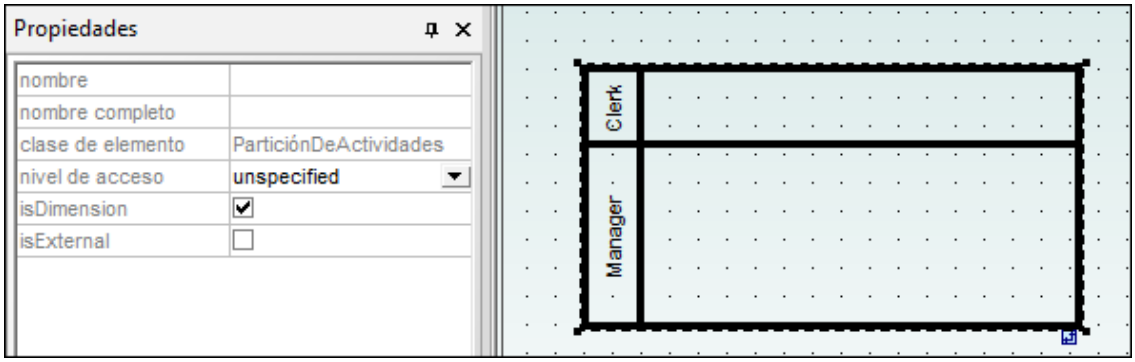
Inserta un nodo de búfer central que funciona de búfer para varios flujos de entrada y salida de otros nodos de objeto.

NodoAlmacénDeDatos

Inserta un nodo de almacén de datos, un nodo de búfer central especial que almacena datos persistentes (es decir, no transitorios).

ParticiónDeActividades (horizontal)

Inserta una partición de actividades horizontal, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Suelen equivaler a las unidades de organización de los modelos de negocio.



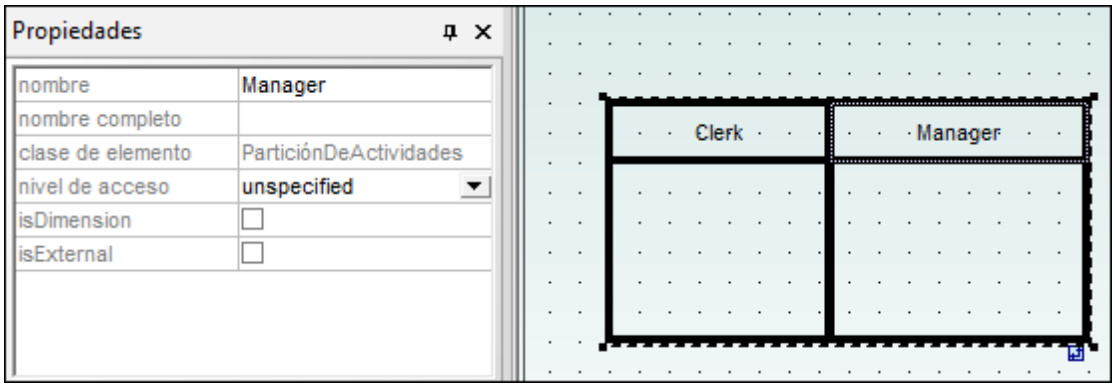
Para editar una etiqueta, haga doble clic en ella. Para orientar el texto correctamente, pulse **Entrar**.

Recuerde que las particiones de actividades de UML 2.0 son el equivalente de los *compartimentos* (`swimlanes`) de las versiones antiguas de UML.

- Los elementos colocados dentro de una **`ParticiónDeActividades`** pasan a formar parte de ella cuando su contorno esté resaltado.
- Los objetos colocados dentro de una **`ParticiónDeActividades`** se pueden seleccionar uno por uno con **Ctrl+clic** o con el recuadro de selección.
- Para mover la **`ParticiónDeActividades`** a otra posición, haga clic en su contorno o en su título y arrástrela.

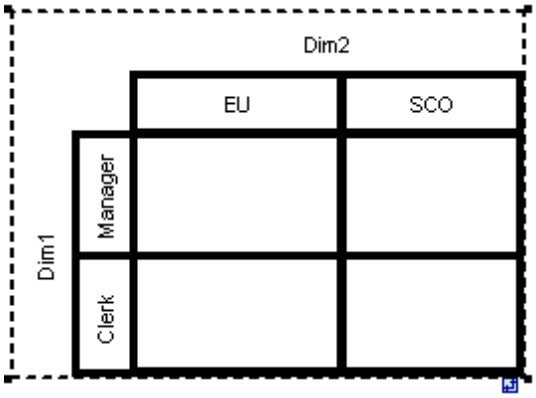
ParticiónDeActividades (vertical)

Inserta una partición de actividades vertical, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Suelen equivaler a las unidades de organización de los modelos de negocio.



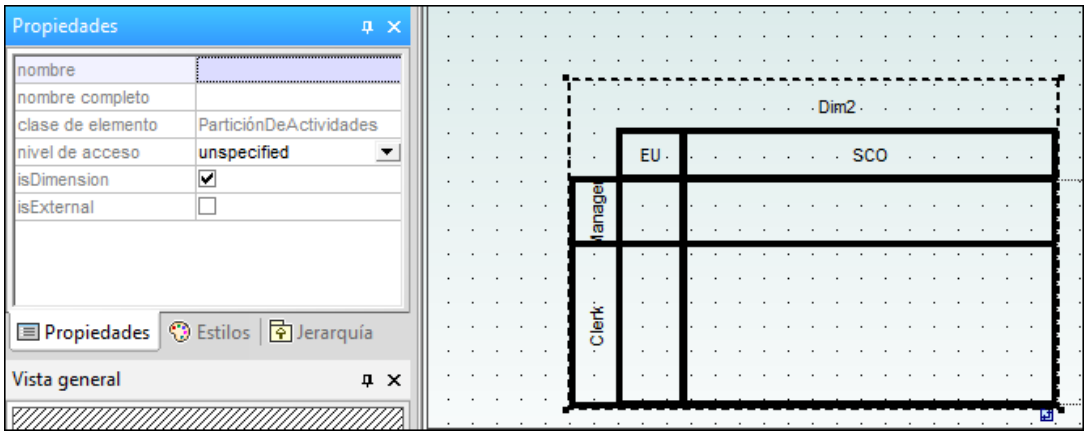
ParticiónDeActividades (2D)

Inserta una partición de actividades bidimensional, un tipo de grupo de actividades que sirve para identificar acciones que tienen características en común. Las etiquetas de los dos ejes son editables.



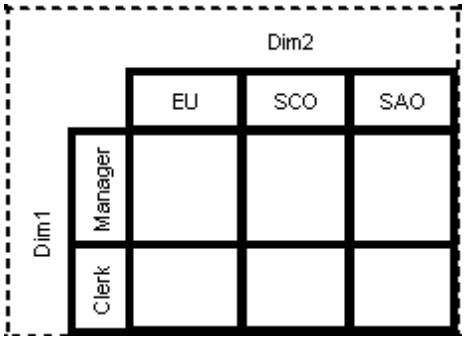
Para quitar las etiquetas de las dimensiones Dim1 y Dim2:

1. Haga clic en la etiqueta que desea eliminar (p. ej. `Dim1`).
2. En el panel Propiedades haga doble clic en la entrada `Dim1`, elimínela y pulse **Entrar** para confirmar.



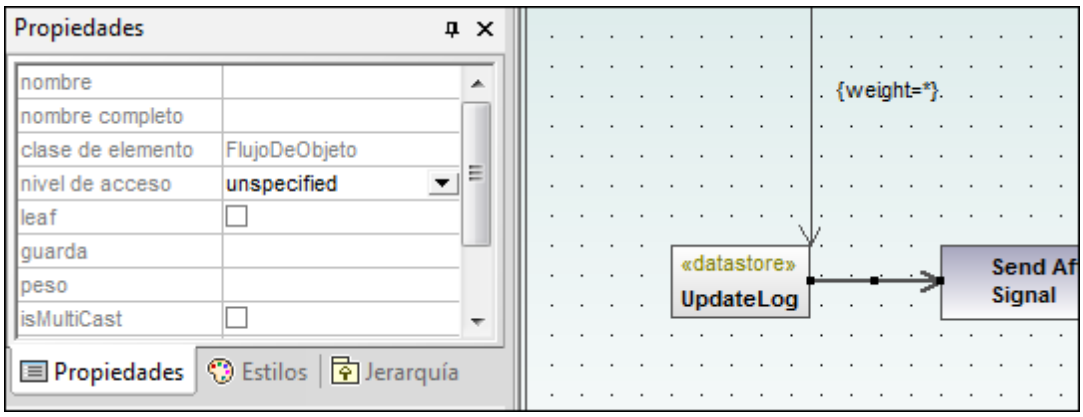
Para anidar particiones de actividades:

1. Haga clic con el botón secundario en la etiqueta de la dimensión donde desea insertar un partición nueva.
2. Seleccione **Nuevo/a | ParticiónDeActividades**.



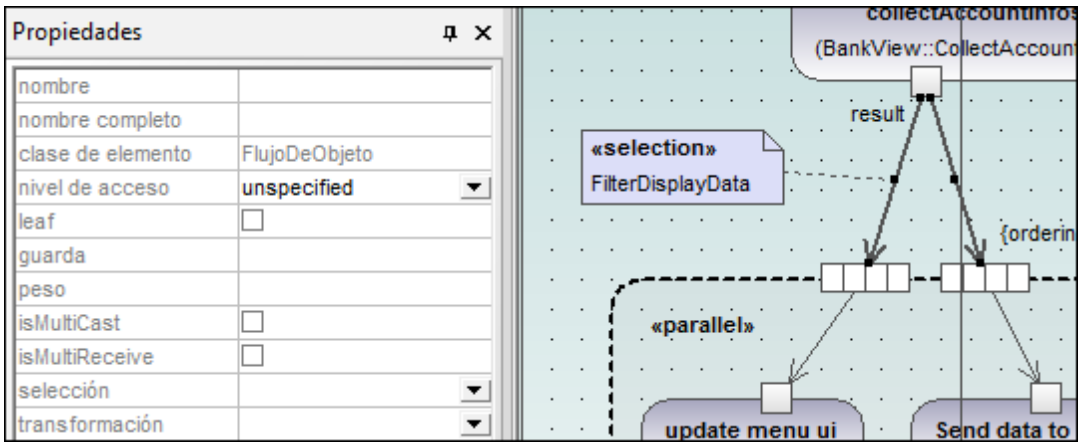
FlujoDeControl

Un flujo de control es una línea con una flecha que conecta dos actividades/comportamientos e inicia una actividad una vez finaliza la actividad anterior.



FlujoDeObjeto

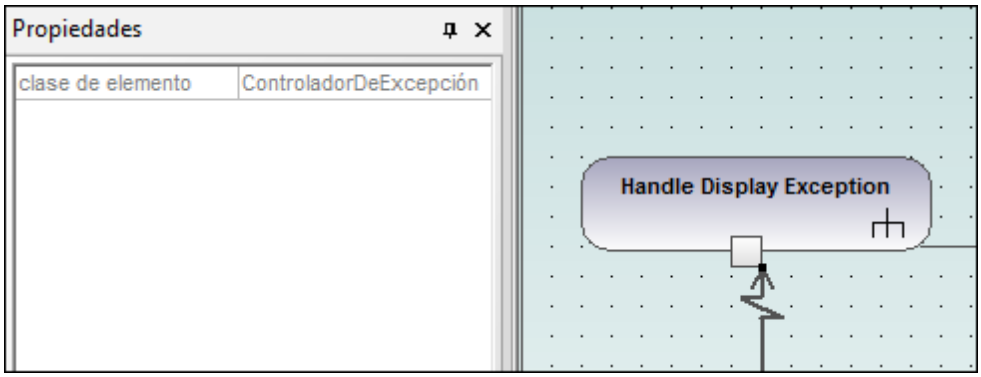
Un flujo de objeto es una línea con una flecha que conecta dos acciones/nodos de objeto e inicia una actividad una vez finaliza la actividad anterior. Los objetos y datos se pueden pasar a través del flujo de objeto.



 © 2014 Altova GmbH

ControladorDeExcepción

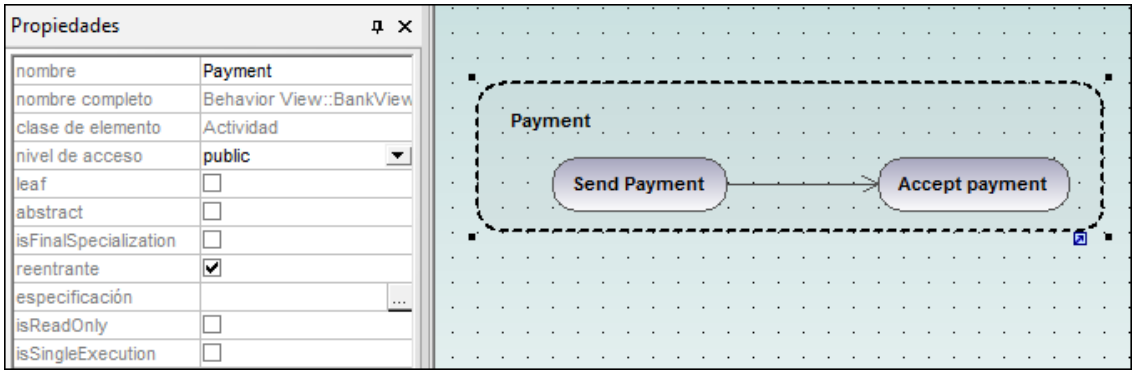
Un controlador de excepción es un elemento que especifica qué acción debe ejecutarse si se genera determinada excepción durante la ejecución del nodo protegido.



Los controladores de excepción solo se pueden colocar en el pin de entrada de una acción.

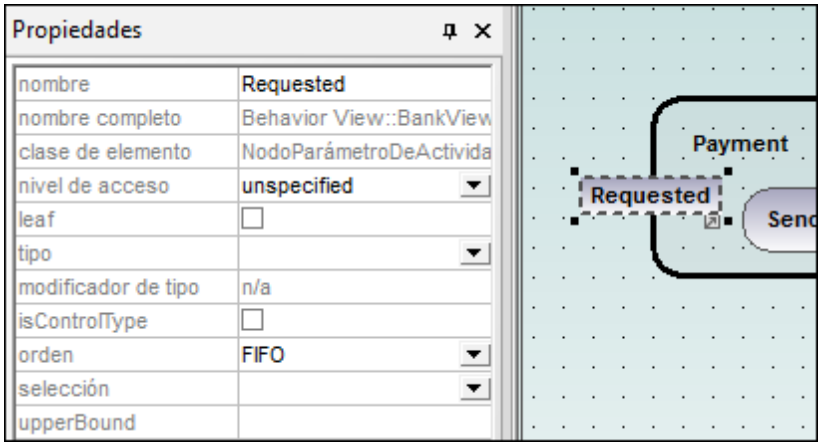
Actividad

Inserta una actividad en el diagrama de actividades.



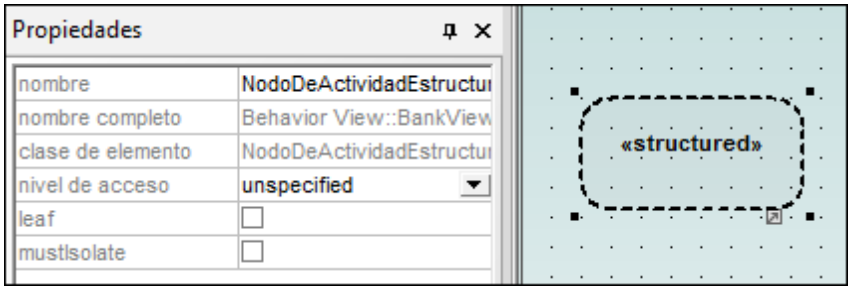
NodoParámetroDeActividad

Inserta un nodo parámetro de actividad en una actividad. Al hacer clic en la actividad se inserta el nodo parámetro en el contorno de la actividad.



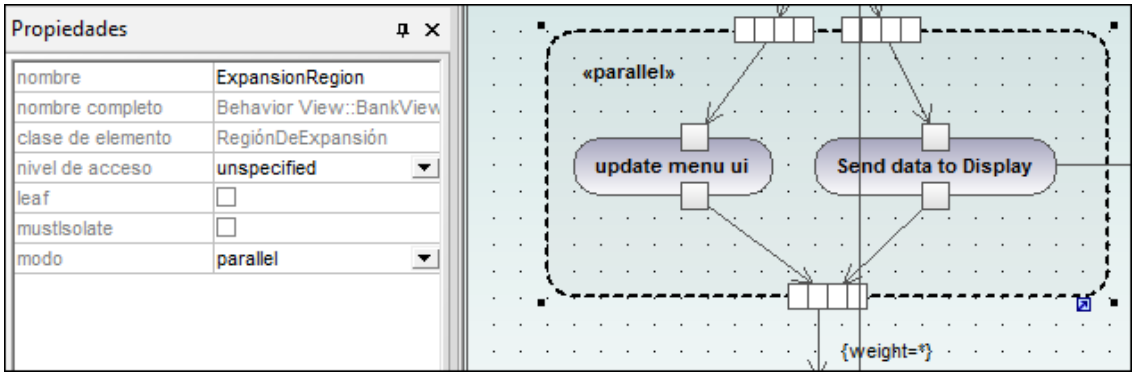
NodoDeActividadEstructurada

Inserta un nodo de actividad estructurada, que es una parte estructurada de la actividad que no se comparte con ningún otro nodo estructurado.



RegiónDeExpansión

Una región de expansión es una región de una actividad que tiene entradas y salidas explícitas (usando **`NodosDeExpansión`**). Cada entrada es una colección de valores.

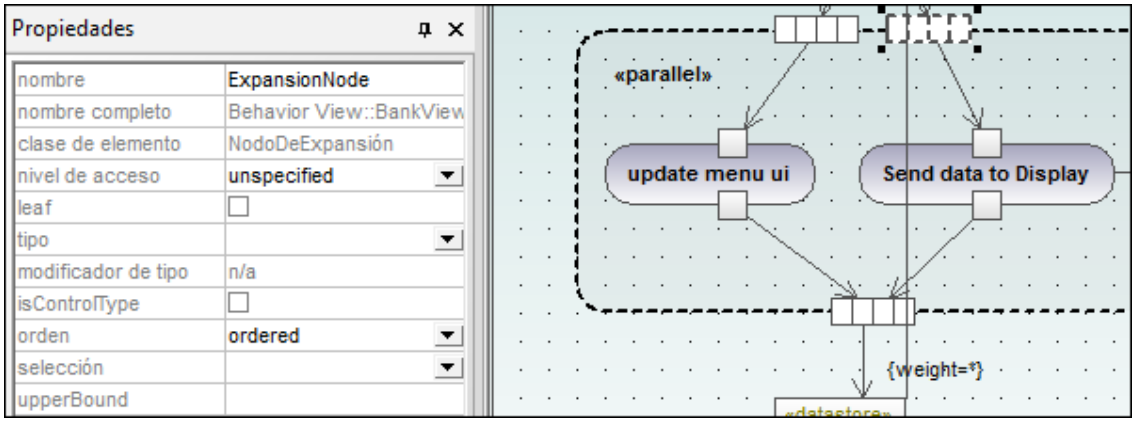


El modo región de expansión aparece como palabra clave y para cambiarlo basta con hacer clic en el cuadro combinado `modo` del panel Propiedades. Las opciones disponibles son: `parallel`, `iterative` o `stream`.

 © 2014 Altova GmbH

NodoDeExpansión

Inserta un nodo de expansión en una región de expansión. Los nodos de expansión son nodos de entrada y salida para la región de expansión, donde cada entrada/salida es una colección de valores. Las flechas que entran y salen de la región de expansión determinan el tipo concreto de nodo de expansión.

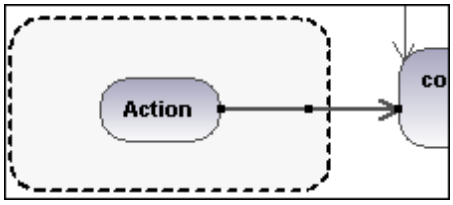


RegiónDeActividadInterrumpible

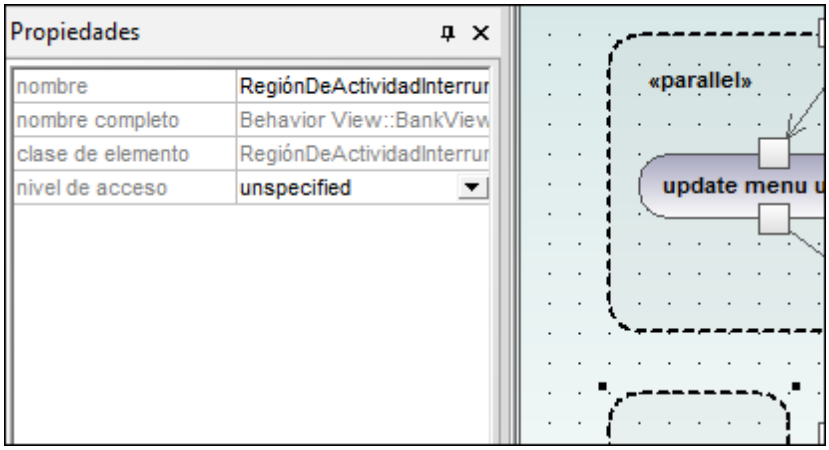
Una región interrumpible contiene nodos de actividad. Cuando un flujo de control abandona una región interrumpible, todos los flujos y comportamientos de la región finalizan.

Para añadir un encadenamiento interruptor:

Primero debe comprobar que hay un elemento `Acción` en la `RegiónDeActividadInterrumpible`, así como un flujo de control de salida hacia otra acción:



1. Haga clic con el botón secundario en la flecha del `FlujoDeControl` y seleccione **Nuevo/a | EncadenamientoInterruptor**.



Nota: hay otra manera de añadir un `EncadenamientoInterruptor`: haga clic en la `RegiónDeActividadInterrumpible`, después haga clic con el botón secundario en el panel Propiedades y seleccione **Agregar encadenamientoInterruptor** en el menú emergente.

 © 2014 Altova GmbH

10.1.2 Diagrama de máquina de estados

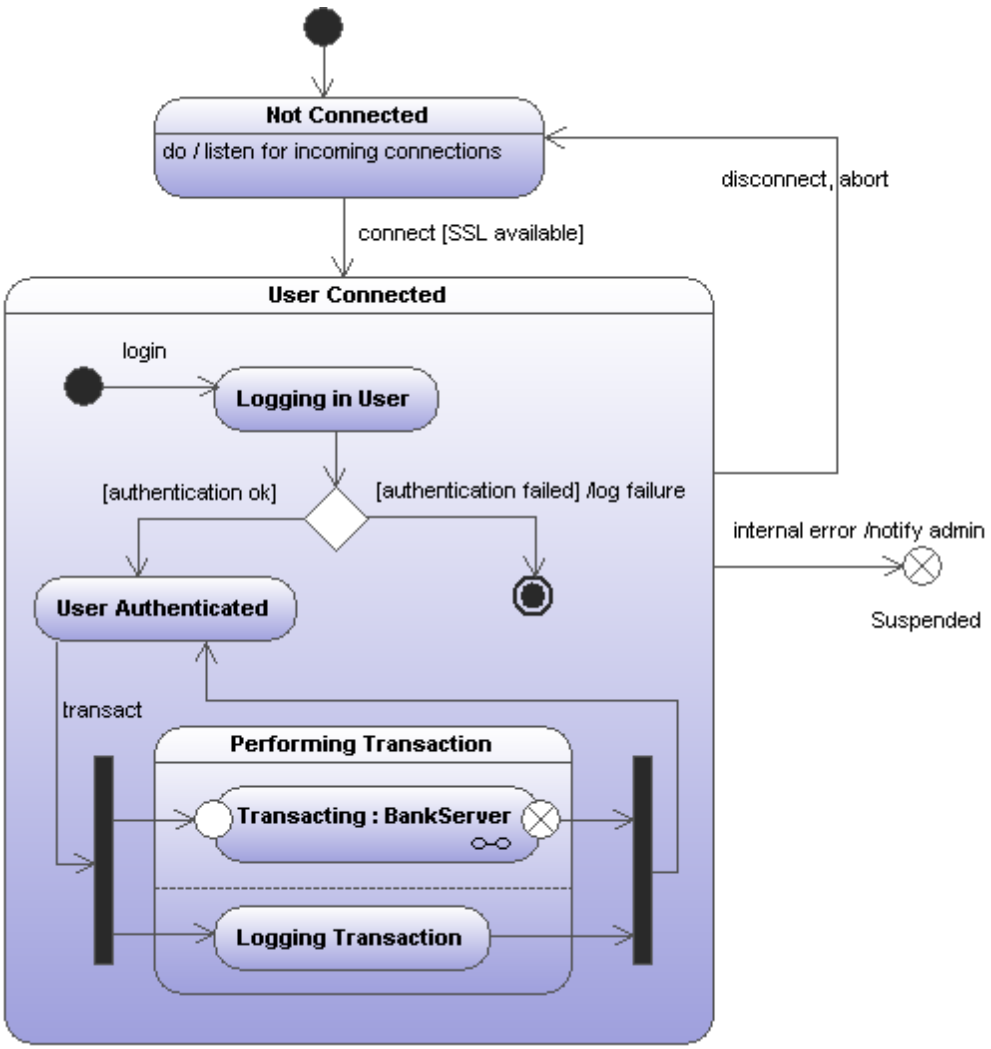
Sitio web de Altova: diagramas de máquina de estados

Los diagramas de máquina de estados modelan el comportamiento de un sistema, describiendo los diferentes estados por los que puede pasar un objeto y las transiciones de unos estados a otros. Se suelen utilizar para describir el comportamiento de un objeto que pasa por varios casos de uso. Una máquina de estados puede tener un número ilimitado de diagramas de máquina de estados (o de diagramas de estado).

Esto se puede conseguir con dos tipos de procesos:

- **Acciones:** están asociadas a las **transiciones** y son procesos a corto plazo que no se pueden interrumpir. Por ejemplo: una transición inicial **`error interno / notificar admin`**.
- **Actividades de estado (comportamientos):** están asociadas a los estados y son procesos a largo plazo que pueden ser interrumpidos por otros eventos. Por ejemplo: **`escuchar si hay conexiones entrantes`**.

Nota: el diagrama de máquina de estados que aparece a continuación está disponible en el ejemplo `Bank_MultiLanguage.ump`, en la carpeta `...\UModelExamples`.



Insertar elementos

Usar los iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de máquina de estados.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de máquina de estados.

 © 2014 Altova GmbH

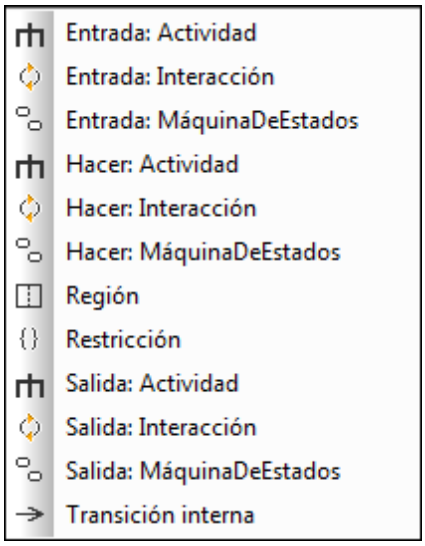
Crear estados, actividades y transiciones

Para insertar un estado simple:

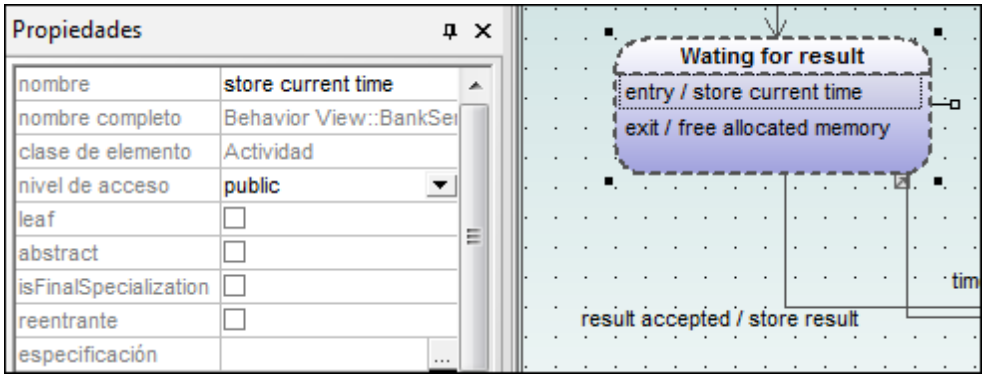
1. Haga clic en el icono **Estado** de la barra de herramientas y haga clic en el diagrama para insertarlo.
2. Escriba el nombre del estado y pulse **Entrar** para confirmar.
   Los estados simples no tienen regiones ni subestructuras. En UModel puede añadir actividades y regiones a un estado simple desde un menú contextual.

Para añadir una actividad al estado:

1. Haga clic con el botón secundario en el estado. En el menú contextual seleccione **Nuevo/a** y después una de las opciones del submenú.



Hay tres tipos de acciones: **Hacer** (`Do`), **Entrada** (`Entry`) y **Salida** (`Exit`). Las actividades se colocan dentro de su propio compartimiento en el estado (no en una región distinta). El tipo de actividad seleccionada se utiliza como prefijo para la actividad (p. ej. `entry / store current time`).

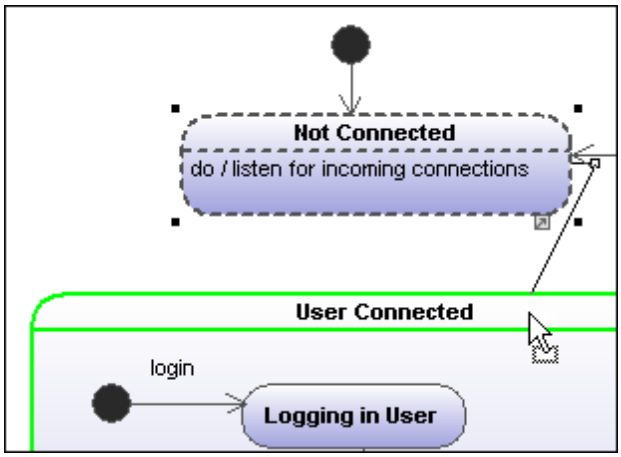


Para eliminar una actividad:

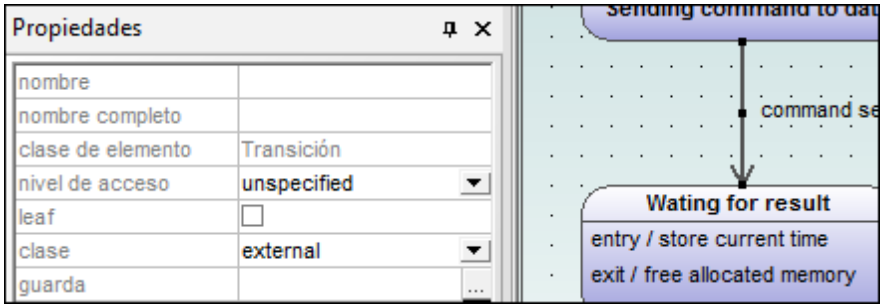
1. Haga clic en la actividad del estado y pulse la tecla **Supr**.

Para crear una transición entre dos estados:

1. Haga clic en el controlador `Transición` del estado de origen (situado a la derecha del elemento).
2. Arrastre la flecha de la transición hasta el estado de destino.



Las propiedades de la transición se pueden ver en la ventana Propiedades. En el cuadro combinado del campo `clase` puede definir el tipo de transición: externa, interna o local.



Las transiciones pueden tener un disparador de eventos, una condición de protección y una acción con el formato **disparadorEvento [condición de protección] /actividad**.

Para crear operaciones desde transiciones automáticamente

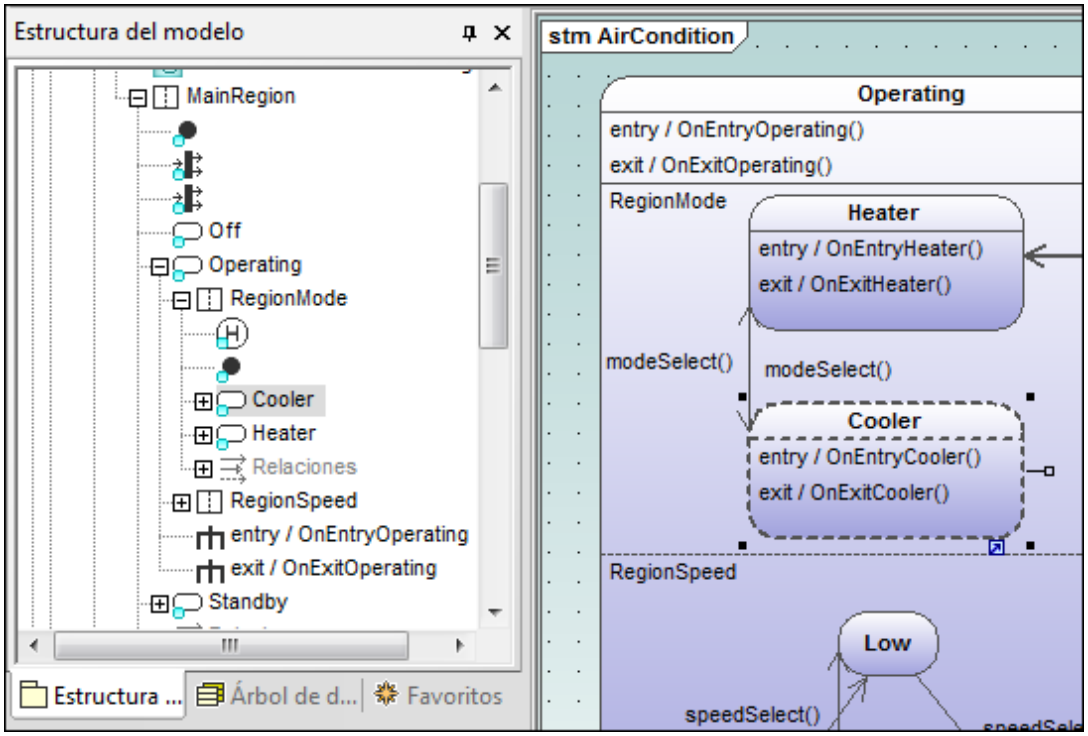
Si activa el icono **Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación** , operación correspondiente se crea automáticamente en la clase referenciada cuando se crea una transición y se escribe un nombre (p. ej. `miOperación()`).

Nota: solamente se pueden crear operaciones automáticamente cuando la máquina de estados está dentro de una clase o de una interfaz.

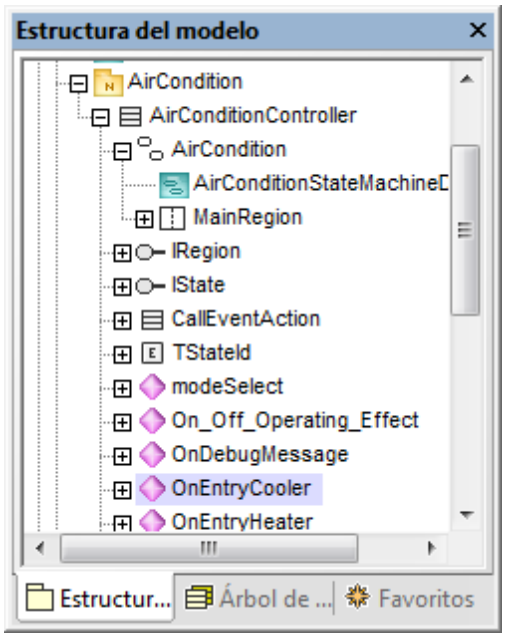
Para crear operaciones automáticamente desde actividades:

1. Haga clic con el botón secundario en el estado y seleccione la actividad/acción que desea insertar (**Nuevo/a | Entrada:Actividad**).
2. Escriba el nombre de la actividad, asegurándose de que termina con ().

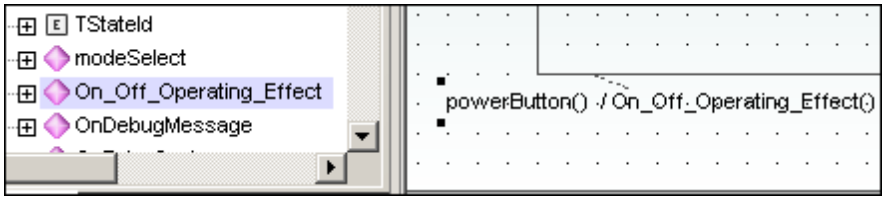
 © 2014 Altova GmbH



El elemento nuevo también está disponible en la Estructura del modelo. Desplácese hacia abajo en la Estructura del modelo y observe que la operación `OnEntryCooler` se añadió a la clase primaria `AirConditionController`.

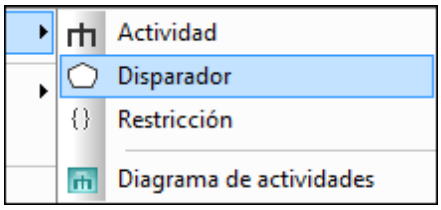


Nota: se añaden operaciones automáticamente para: **Hacer:Actividad, Entrada:Actividad, Salida:Actividad**.



Para crear un disparador de transición:

1. Haga clic con el botón secundario en una transición (en la flecha).
2. Selección **Nuevo/a | Disparador**.

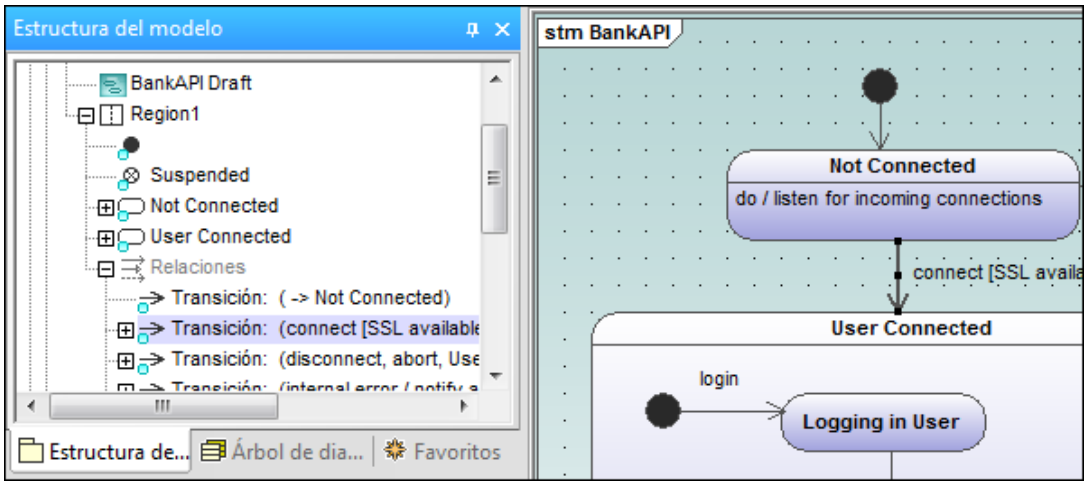


Si se trata del primer disparador del diagrama, en la etiqueta de la transición, situada encima de la flecha, aparece el carácter "a". Los disparadores tienen asignados valores predeterminados en forma de letra, estado de origen -> estado de destino.

3. Haga doble clic en el nuevo carácter e inserte las propiedades de la transición en el formato **disparadorEvento [condición de protección] / actividad**.

   Sintaxis de las propiedades de la transición: el texto insertado antes de los corchetes es el disparador. Entre los corchetes va la condición de protección y después de la barra diagonal va la actividad. Manipule esta cadena para crear o eliminar automáticamente los elementos correspondientes en la Estructura del modelo.

   Nota: para ver las propiedades de una transición, haga clic con el botón secundario en la transición y seleccione **Seleccionar en la Estructura del modelo**. El evento, la actividad y los elementos de restricción aparece debajo de la transición seleccionada.

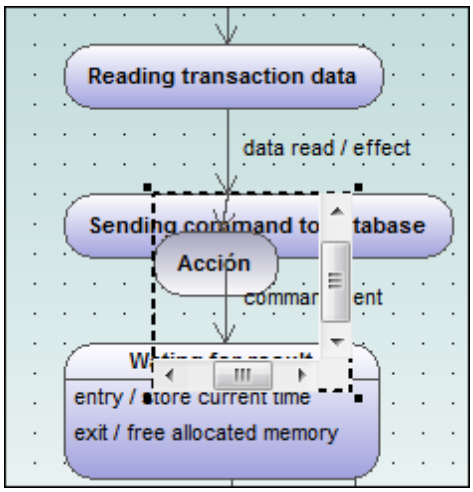


Para agregar un diagrama de actividades a una transición:

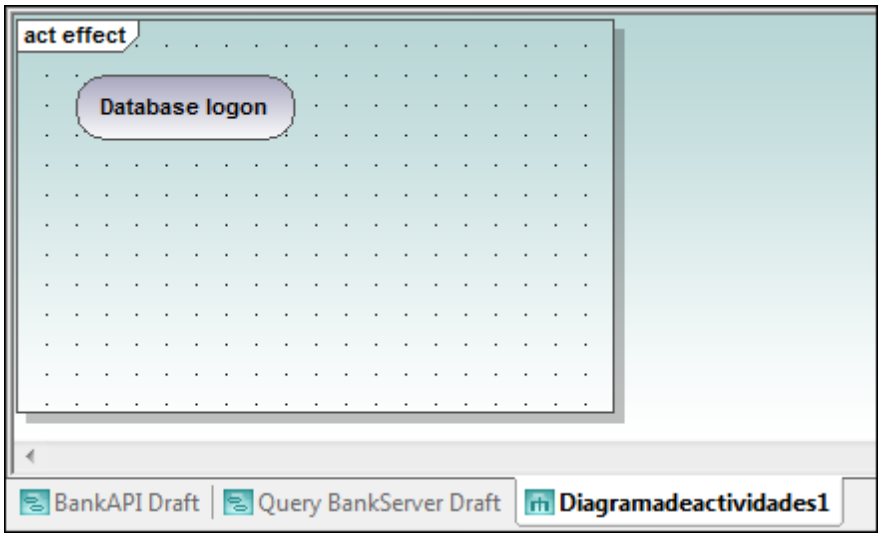
UModel ofrece una función única para añadir diagramas de actividades a las transiciones a fin de describir la transición en detalle.

 © 2014 Altova GmbH

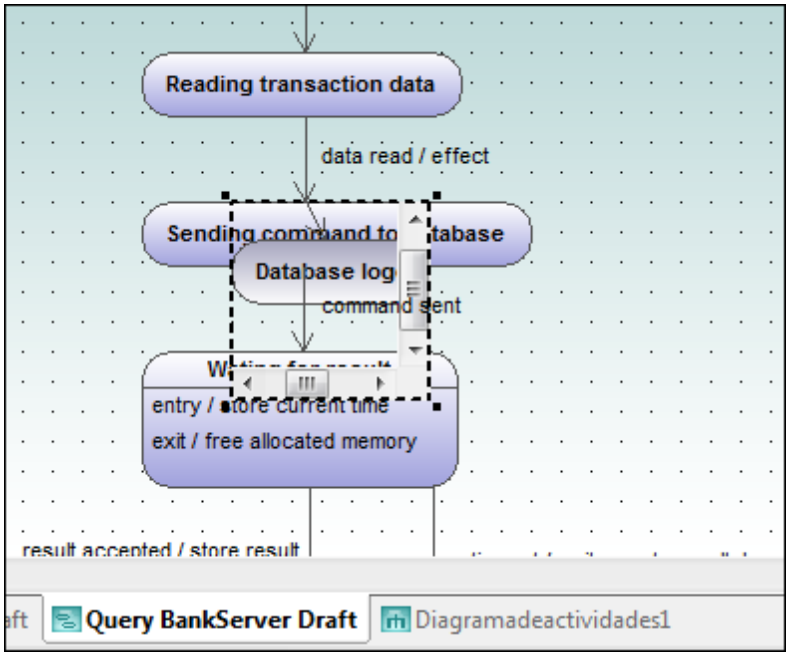
1. Haga clic con el botón secundario en la transición y seleccione **Nuevo/a | Diagrama de actividades**.
   Esto inserta una ventana con un diagrama de actividades en la posición de la flecha de transición.
2. Haga clic en la ventana recién insertada y utilice las barras de desplazamiento para desplazarse por la ventana.



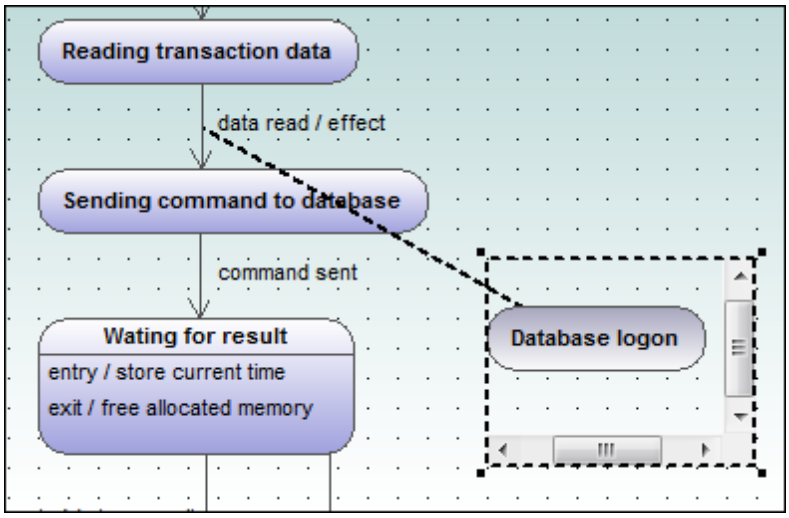
3. Haga doble clic en la ventana para abrir el diagrama de actividades en otra pestaña y seguir definiendo la transición (p. ej. cambiando el nombre de la acción a **`Database logon`**).



   Al proyecto se añade un diagrama de actividades nuevo. Para aprender a añadir elementos de modelado de actividades nuevos al diagrama, consulte el apartado Diagrama de actividades.
4. Haga clic en la pestaña del diagrama de máquina de estados para ver la transición actualizada.

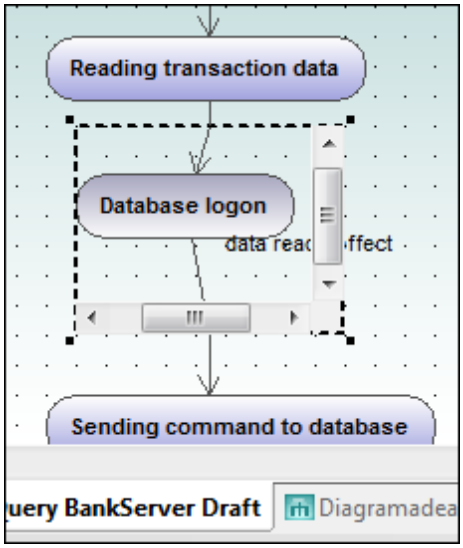


5. Arrastre la ventana de la actividad a una posición nueva donde no moleste y ajuste el tamaño de la ventana si es necesario.



Si arrastra la ventana de la actividad y la pone entre los dos estados, la ventana ilustra la transición hacia y desde la actividad.

 © 2014 Altova GmbH



Estados compuestos

Estado compuesto

Este tipo de estado contiene un compartimiento más, formado por una región. Dentro de esta región puede colocar un número ilimitado de estados.

Para añadir una región a un estado compuesto:

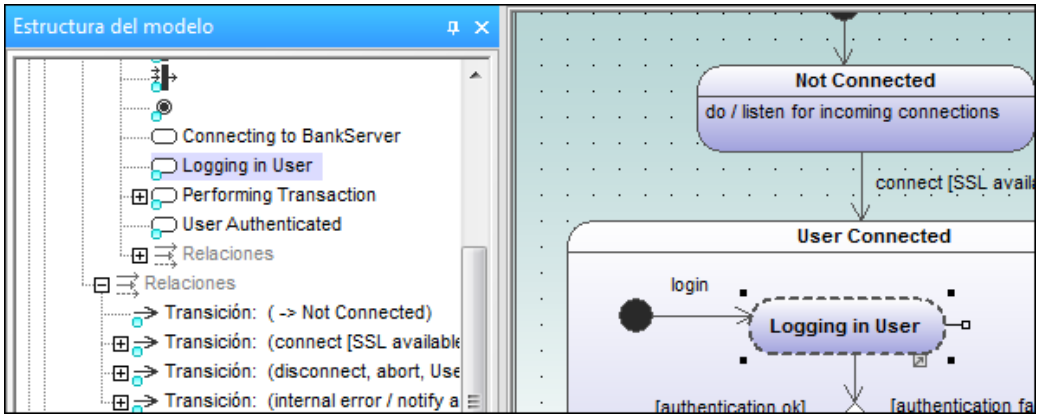
1. Haga clic con el botón secundario en el estado compuesto y seleccione **Nuevo/a | Región** del menú contextual.
   Al estado se le añade una región nueva. Las regiones se dividen con líneas discontinuas.

Para eliminar una región:

1. Haga clic en la región que desea eliminar y pulse **Supr**.
   Cuando se elimina una región de un estado ortogonal, el estado vuelve a ser un estado compuesto. Cuando se elimina la última región de un estado compuesto, el estado pasa a ser un estado simple.

Para poner un estado dentro de un estado compuesto:

1. Haga clic en el estado que desea insertar (p. ej. **Logging in User**) y arrástrelo hasta el compartimento de la región del estado compuesto.
   El compartimento de la región se resalta al soltar el elemento. El elemento insertado ahora forma parte de la región y aparece como elemento secundario de la región en el panel Estructura del modelo.

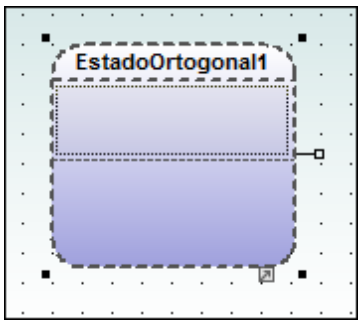


Cuando se mueve el estado compuesto, también se mueven los estados que están dentro de él.

Estado ortogonal

Este tipo de estado contiene un compartimiento más, formado por dos o más regiones, que indican simultaneidad.

Haga clic con el botón secundario en un estado y seleccione **Nuevo/a | Región** para añadir regiones nuevas.



Para mostrar/ocultar el nombre de las regiones:

Haga clic en el panel Estilos, desplácese hasta el estilo `Mostrar los nombres de región en los estados` y seleccione el valor `verdadero/falso`.

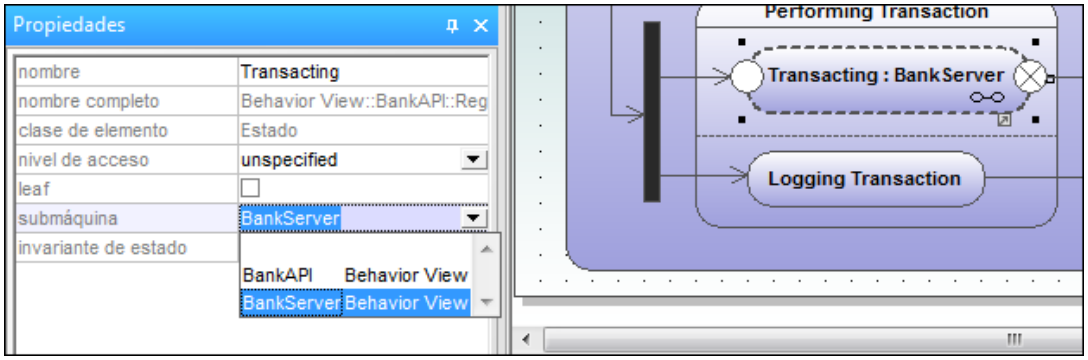
Estado de submáquina

Este estado sirve para ocultar los detalles de una máquina de estados. Este estado no tiene regiones, sino que está asociado a una máquina de estados distinta.

Para definir un estado de submáquina:

1. Tras seleccionar un estado, haga clic en el cuadro combinado `submáquina` del panel Propiedades.
   Aparece una lista con todas las máquinas de estados que están definidas.
2. Seleccione la máquina de estados a la que debe hacer referencia esta submáquina.

 © 2014 Altova GmbH

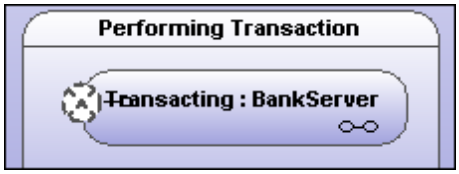


Observe que en la submáquina aparece automáticamente un icono de hipervínculo. Al hacer clic en este icono se abre la máquina de estados a la que se hace referencia (**`BankServer`**, por ejemplo).

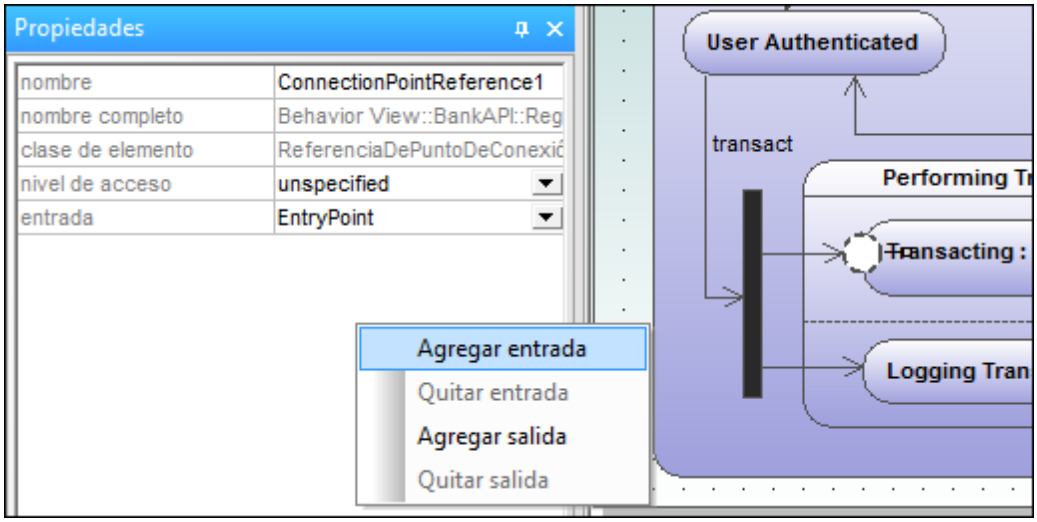
Para añadir puntos de entrada/ salida a un estado de submáquina:

- El estado de submáquina al que está conectado el punto de entrada/salida debe hacer referencia a una máquina de estados (visible en el panel Propiedades).
- Esta submáquina debe contener un punto de entrada y de salida como mínimo.

1. Haga clic en el icono **ReferenciaDePuntoDeConexión** de la barra de herramientas y después haga clic en el estado de submáquina en el que quiere insertar el punto de entrada/salida.



2. Haga clic con el botón secundario en el panel Propiedades y seleccione **Agregar entrada**. Recuerde que este menú emergente solamente aparece si en el diagrama ya existe un punto de entrada o salida.



El comando **Agregar entrada** añade un punto de entrada (`EntryPoint`) nuevo en el panel Propiedades y cambia el aspecto de la referencia de punto de conexión

`ConnectionPointReference`.

3. Use el mismo método para insertar un punto de salida (`ExitPoint`) con la opción **Agregar salida** del menú emergente.

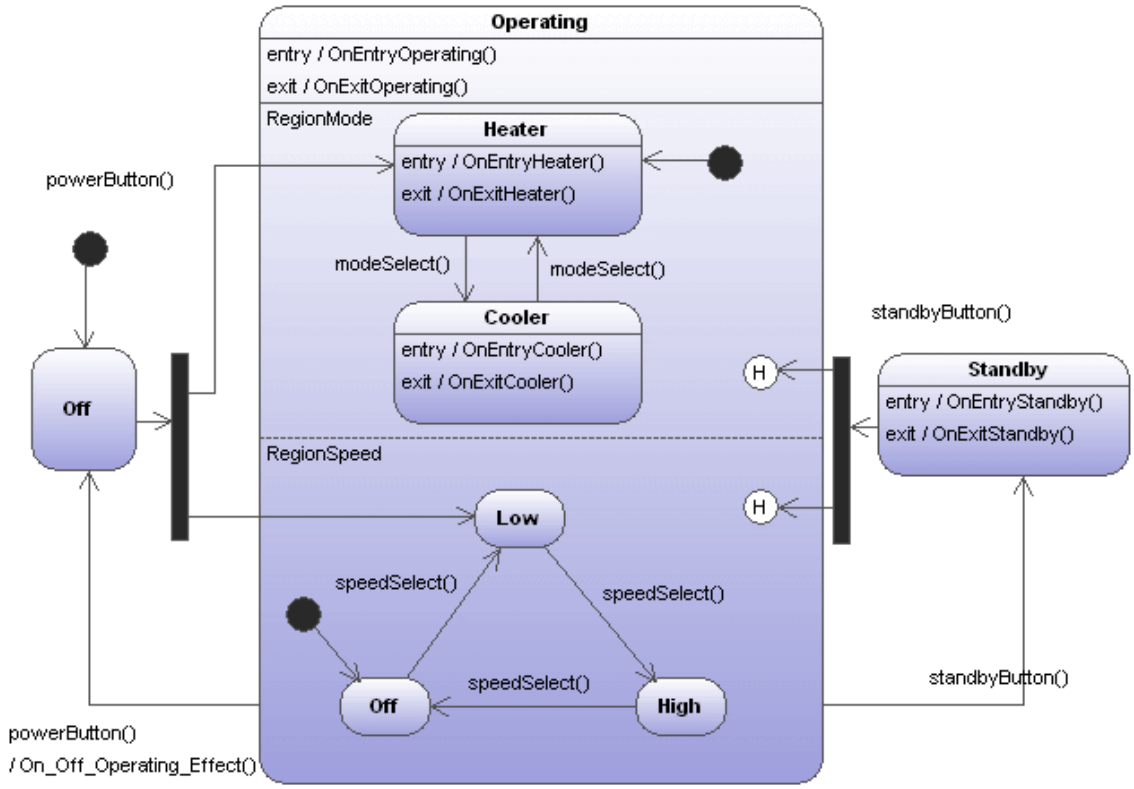
Generar código a partir de diagramas de máquina de estados

Con UModel puede generar código ejecutable a partir de diagramas de máquina de estados (Java, VB.NET o C#). Esta función de generación de código es compatible con casi todos los elementos y características de los diagramas de máquina de estados:

- Estado
- EstadoCompuesto, con cualquier nivel jerárquico
- EstadoOrtogonal, con cualquier número de regiones
- Región
- EstadoInicial
- EstadoFinal
- Transición
- Guarda
- Disparador
- Evento de llamada
- Bifurcación
- Reunión
- Elección
- Unión
- HistorialDetallado
- HistorialSuperficial
- Acciones de entrada/salida/hacer
- Efectos

La generación de código de máquina de estados se integra en el proceso "normal" de ingeniería de ida y vuelta. Esto significa que el código de máquina de estados se puede actualizar automáticamente durante el proceso de ingeniería directa.

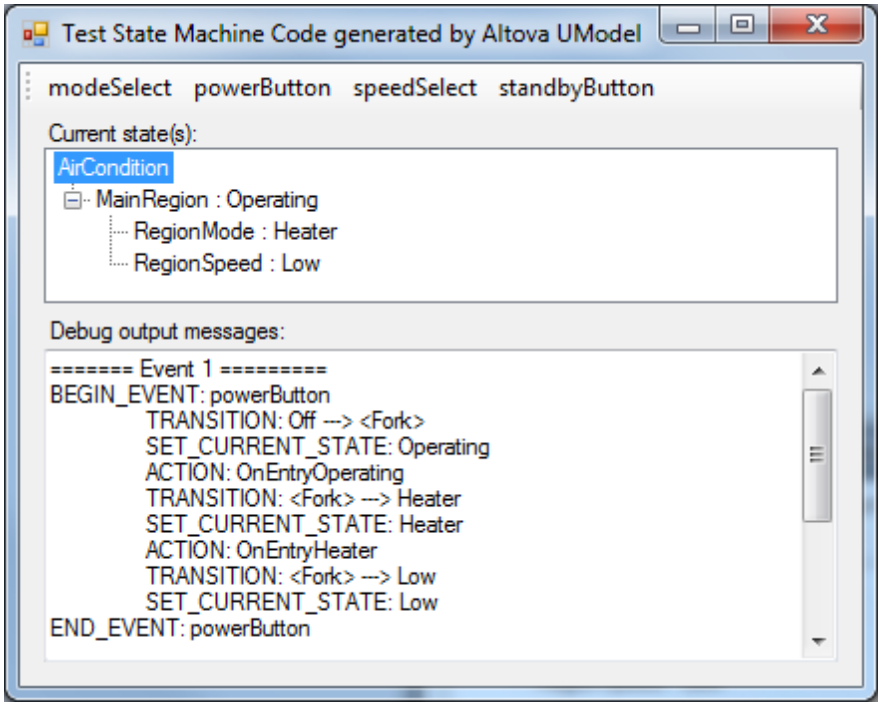
 © 2014 Altova GmbH



La imagen anterior muestra el diagrama de máquina de estados **`AirCondition`** de la carpeta `..\StateMachineCodeGeneration` del directorio `...\UModelExamples`. Por cada lenguaje de programación hay una carpeta (C#, Java y VB).

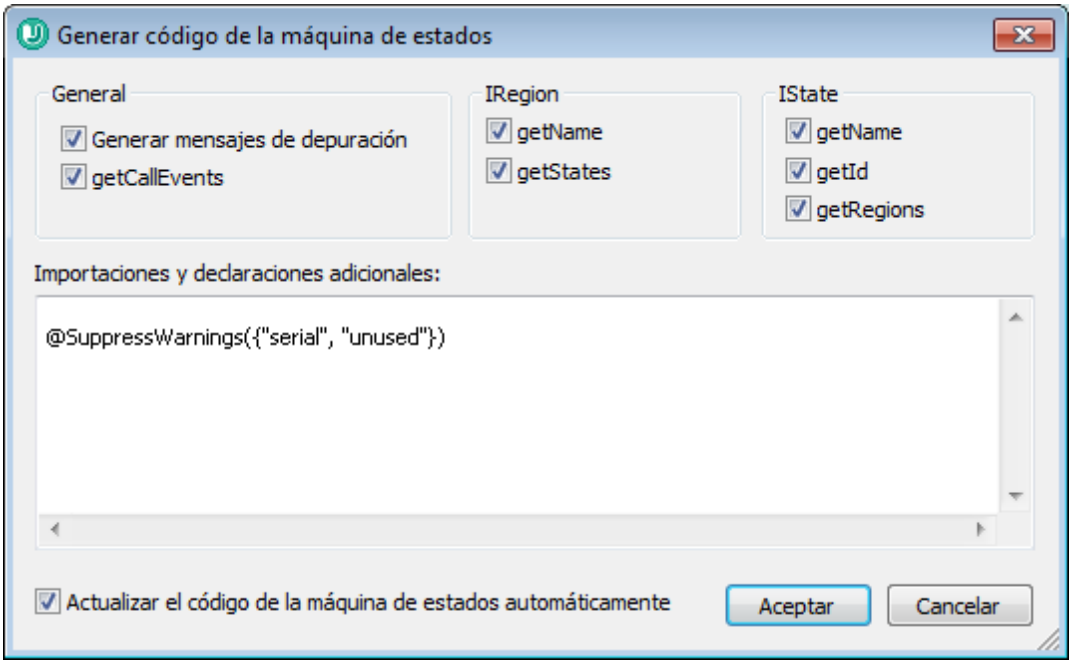
Cada directorio contiene dos carpetas: `AirCondition` y `Complex`. Cada una contiene el proyecto de UModel correspondiente, los archivos de proyecto del lenguaje de programación y los archivos de código generados. El archivo de proyecto `Complex.ump` contiene casi todos los elementos y funciones de modelado compatibles con la función de generación de código de UModel para diagramas de máquina de estados.

Además, cada carpeta contiene una aplicación de prueba (p. ej. `TestSTMAirCondition.sln` para C#) para que pueda trabajar inmediatamente con los archivos de código generados.



Para generar código a partir de un diagrama de máquina de estados:

1. Haga clic con el **botón secundario** en el diagrama de máquina de estados y seleccione el comando **Generar código de la máquina de estados** o
2. Haga clic en **Proyecto | Generar código de la máquina de estados**.

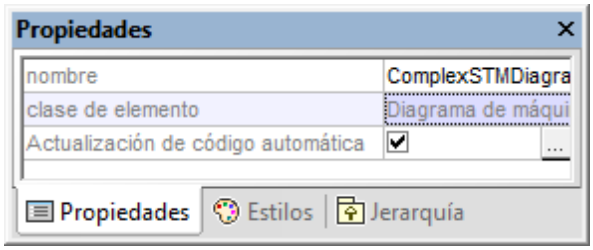


Aparece un cuadro de diálogo (*imagen siguiente*). Si es necesario, ajuste las opciones de configuración predeterminadas y haga clic en **Aceptar** para generar el código.

 © 2014 Altova GmbH

El código de máquina de estados se actualiza automáticamente cuando se inicia el proceso de ingeniería directa. Sin embargo, esta configuración se puede cambiar. Para ello haga clic en el fondo del diagrama de máquina de estados y marque la casilla Actualización de código automática del panel Propiedades.

No es recomendable realizar cambios a mano en el código generado porque estos cambios no se traspasarán al diagrama de máquina de estados durante el proceso de ingeniería inversa.



En el panel Propiedades haga clic en el icono **Examinar** del campo Actualización de código automática para abrir el cuadro de diálogo "Generar código de la máquina de estados" y cambiar las opciones de configuración.

Nota: puede revisar la sintaxis del diagrama de máquina de estados haciendo clic con el botón secundario en el diagrama y seleccionando **Revisar la sintaxis de la máquina de estados**.

Trabajar con código de máquina de estados

La clase primaria de la máquina de estados (es decir, la clase controladora `controller` o la clase de contexto) es la única interfaz que existe entre el usuario de la máquina de estados y su implementación.

La clase controladora `controller` aporta los métodos que se pueden usar desde "fuera" para cambiar los estados (p. ej. después de que tengan lugar eventos externos).

No obstante, la implementación de la máquina de estados llama a los métodos de la clase `controller` (devoluciones de llamada) para informar al usuario de la máquina de estados sobre cambios de estado (`OnEntry`, `OnExit`, ...), efectos de las transiciones y la posibilidad de invalidar e implementar métodos para condiciones (guardas).

UModel puede crear operaciones simples (sin parámetros) automáticamente para comportamientos entrar/salir/hacer, efectos de transición, etc. cuando se activa la opción correspondiente (consulte el apartado Crear estados, actividades y transiciones.) Estos métodos se pueden cambiar (añadiéndoles parámetros, configurándolos como métodos abstractos, etc.).

Puede generar instancias de una máquina de estados (es decir, de su clase controladora `controller`) y todas las instancias funcionan independientemente.

- La ejecución de la máquina de estados UML está diseñada para el *modelo de ejecución hasta el final*.
- Las máquinas de estados UML suponen que el procesamiento de cada evento finaliza antes de que empiece a procesarse el siguiente evento.
- Esto también significa que las acciones entrar/salir/hacer y los efectos de las transiciones no pueden disparar transiciones/cambios de estado nuevos directamente.

Inicialización:

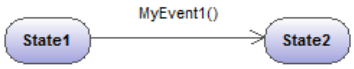
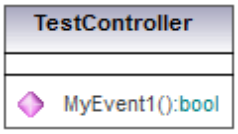
- Cada región de una máquina de estados debe tener un estado inicial.
- El código generado con UModel inicializa automáticamente todas las regiones de la máquina de estados (o cuando se llama al método `Initialize()` de la clase controladora).
- Si no necesita eventos `OnEntry` durante la inicialización, puede llamar a mano al método `Initialize()` e ignorar los eventos `OnEntry` durante el inicio.

Obtener el estado actual:
UModel admite estados compuestos y estados ortogonales, así que no hay un solo estado actual: cada región (de cualquier nivel jerárquico) puede tener un estado actual.

En el proyecto de ejemplo `AirCondition.ump` puede ver cómo se pueden recorrer las regiones hasta llegar a los estados actuales:

```
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);
    }
}
```

Ejemplo nº1: una transición simple

	UModel genera automáticamente la operación correspondiente.
	 
		Método generado en el código:
		`private class CTestStateMachine : IState`
		`{`
		`…`
		`public bool MyEvent1()`
		`{`
		`…`
		`}`
		`}`

- El usuario de la máquina de estados debería llamar al método generado "MyEvent1" cuando tenga lugar el evento correspondiente (fuera de la máquina de estados).
- El parámetro de devolución de estos métodos-evento aporta información si el evento provocó un cambio de estado (es decir, si tuvo un efecto o no en la máquina de estados).

  P. ej. estando en el estado "State1" y teniendo lugar el evento "MyEvent1()", el estado actual cambia a "State2" y "MyEvent1()" devuelve el valor `true`.

  Si el estado "State2" está activo y tiene lugar el evento "MyEvent1()" en la máquina de estados no se producen cambios y MyEvent1() devuelve el valor `false`.

 © 2014 Altova GmbH

Ejemplo nº2: una transición simple con un efecto

MyEvent1() / OnState1State2Effect()
State1 → State2
	UModel genera automáticamente la operación correspondiente.
	TestController
	MyEvent1():bool
	OnState1State2Effect(in text:String):void
		Método generado en el código:
		`private class CTestStateMachine : IState`
		`{`
		`…`
		`// Overwrite to handle entry/exit/do actions, transition effects,...:`
		`public virtual void OnState1State2Effect() {}`
		`}`

- La implementación de la máquina de estados llamará a "OnState1State2Effect()" cuando se dispare la transición del estado "State1" al estado "State2".
- Para reaccionar a este efecto "OnState1State2Effect()" debería sobrescribirse en una clase derivada de "CTestStateMachine".
- "CTestStateMachine:: OnState1State2Effect()" también puede configurarse como `abstract` y obtendrá errores de compilación hasta que se sobrescriba el método.
- Cuando "OnState1State2Effect()" no es abstracto y está activa la opción Generar mensajes de depuración, UModel genera este resultado:

```
// Overwrite to handle entry/exit/do actions, transition
effects,...:
public virtual void OnState1State2Effect()
{OnDebugMessage("ACTION: OnState1State2Effect");}
```

Ejemplo nº3: una transición simple con un efecto y un parámetro

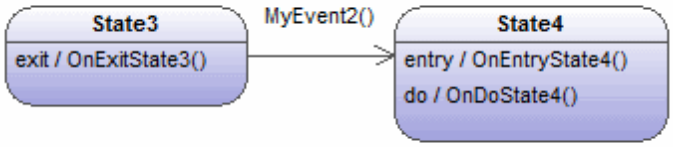
MyEvent1() / OnState1State2Effect("1 => 2")
State1 → State2
	UModel genera automáticamente la operación correspondiente.
	TestController
	MyEvent1():bool
	OnState1State2Effect(in text:String):void
		Método generado en el código:
		`private class CTestStateMachine : IState`
		`{`
		`…`
		`// Additional defined operations of the controller class:`
		`public virtual void OnState1State2Effect (String text)`
		`{`
		`}`
		`}`

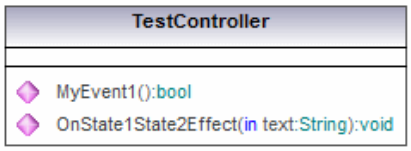
- Para llevar a cabo las operaciones (creadas automáticamente por UModel), puede añadir parámetros manualmente (UModel no puede conocer el tipo necesario).
- En este ejemplo el parámetro "text:String" se añadió al método Effect de TestController. Es necesario especificar un argumento adecuado cuando se llame a este método (en

este caso: "1 => 2").

- Otra posibilidad es llamar a los métodos estáticos ("MyStatic.OnState1State2Effect("1 => 2")") o a los métodos de singleton ("getSingleton().OnState1State2Effect("1 => 2")").

Ejemplo nº4: acciones entrar/salir/hacer

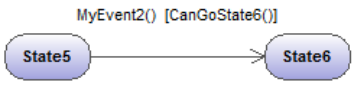


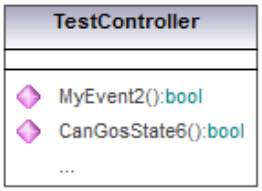
UModel genera automáticamente las operaciones correspondientes.
 
 	
	Método generado en el código:
	`private class CTestStateMachine : IState`
	`{`
	`…`
	`// Overwrite to handle entry/exit/do actions, transition effects,...:`
	`public virtual void OnExitState3() {}`
	`public virtual void OnEntryState4() {}`
	`public virtual void OnDoState4() {}`
	`}`

- Los estados pueden tener comportamientos entrar/salir/hacer. UModel crea automáticamente las operaciones necesarias para ellos.
- Cuando tiene lugar "MyEvent2()", la implementación de la máquina de estados llama a "OnExitState3()", si "MyEvent2" tuviera un efecto, se le llamaría después y posteriormente se llamaría a "OnEntryState4" y "OnDoState4".
- Por lo general estos métodos deberían sobrescribirse. Cuando no son abstractos y está activa la opción *Generar mensajes de depuración*, UModel genera el resultado que se describe en el ejemplo nº2.
- Estos métodos también pueden tener los parámetros que aparecen en el ejemplo nº3.

Ejemplo nº5: guardas

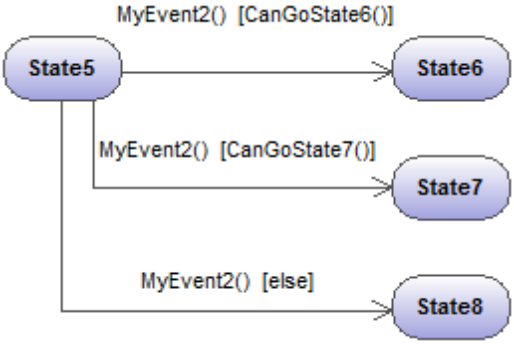
Las transiciones pueden tener guardas, que determinan si la transición se dispara verdaderamente.


 	
	UModel genera automáticamente la operación correspondiente.	Método generado en el código:
		`private class CTestStateMachine : IState`
		`{`
		`…`
		`// Additional defined operations of the controller class:`
		`public virtual bool CanGoState6()`
		`{`

		`return true; // Overwrite! }` `}`

- Si "State5" es el estado activo y tiene lugar "MyEvent2", la implementación de la máquina de estados llamará a "CanGoState6" y, dependiendo de su resultado, la transición se disparará o no.
- Por lo general estos métodos deberían sobrescribirse. Cuando no son abstractos y está activa la opción *Generar mensajes de depuración*, UModel genera el resultado que se describe en el ejemplo nº2.
- Estos métodos también pueden tener los parámetros que aparecen en el ejemplo nº3.
- Varias transiciones pueden tener el mismo evento, pero guardas diferentes. No hay un orden definido para sondear los guardas. Si una transición no tiene guarda o si su guarda es "else", se trata como la última transición (es decir, esta transición solo se disparará si los guardas de las demás transiciones devuelven false.

Ejemplo:
No está definido si primero se dispara CanGoState6 **o** CanGoState7, pero lo que está claro es que la tercera transición solo se disparará si CanGoState6 y CanGoState7 devuelven `false`.



Para ver más funciones y construcciones consulte los ejemplos de los archivos `AirCondition.ump` y `Complex.ump`.

Elementos

EstadoInicial (pseudoestado)
El inicio del proceso.

EstadoFinal
El final de la secuencia de los procesos.

PuntoDeEntrada (pseudoestado)
El punto de entrada de una máquina de estados o de un estado compuesto.

PuntoDeSalida (pseudoestado)
El punto de salida de una máquina de estados o de un estado compuesto.

Elección
Representa una rama condicional dinámica donde se evalúan disparadores de guardas que se excluyen mutuamente (operación OR).

Unión (pseudoestado)
Representa el final de la operación OR definida por el elemento `Elección`.

Terminar (pseudoestado)
La detención de la ejecución de la máquina de estados.

Bifurcación (pseudoestado)
Inserta una barra de bifurcación vertical. Sirve para dividir secuencias en subsecuencias simultáneas.

Bifurcación horizontal (pseudoestado)
Inserta una barra de bifurcación horizontal. Sirve para dividir secuencias en subsecuencias simultáneas.

Reunión (pseudoestado)
Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

Reunión horizontal (pseudoestado)
Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

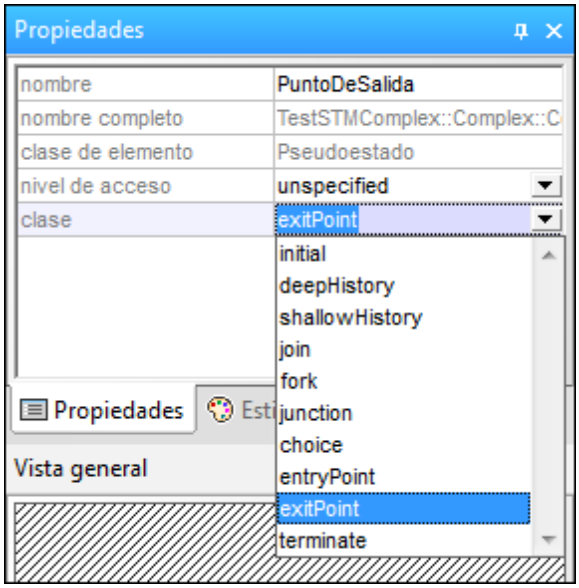
HistorialDetallado
Pseudoestado que restaura el estado activo previo del estado dentro de un estado compuesto.

 © 2014 Altova GmbH

HistorialSuperficial

Pseudoestado que restaura el estado inicial de un estado compuesto.

Para cambiar el tipo de pseudoestado, cambie el valor del cuadro combinado `clase` en el panel Propiedades.



ReferenciaDePuntoDeConexión

Una referencia de punto de conexión representa un uso (como parte de un estado de submáquina) de un punto de entrada/salida definido en la referencia de máquina de estados al que hace referencia el estado de submáquina.

Para agregar puntos de entrada o salida a una referencia de punto de conexión:

- El estado al que está conectado el punto debe hacer referencia a una máquina de estados de submáquina (visible en el panel Propiedades).
- Esta submáquina debe contener un punto de entrada y salida como mínimo.

Transición

La relación directa que existe entre dos estados. Un objeto del primer estado realiza una acción o más y después hace referencia al segundo estado, dependiendo de un evento y de que se cumplan las condiciones de protección.

Las transiciones tienen un disparador de eventos, condiciones de protección, una acción (comportamiento) y un estado de destino.

Subelementos de evento compatibles:

- **EventoRecibirSeñal, EventoSeñal, EventoEnviarSeñal, EventoRecibirOperación, EventoEnviarOperación y EventoDeCambio**.

Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

Active este icono para crear automáticamente la operación correspondiente en la clase a la que se hace referencia cuando se cree una transición y se inserte el nombre de la operación.

Nota: solamente se pueden crear operaciones automáticamente cuando la máquina de estado está dentro de una clase o de una interfaz.

© 2014 Altova GmbH

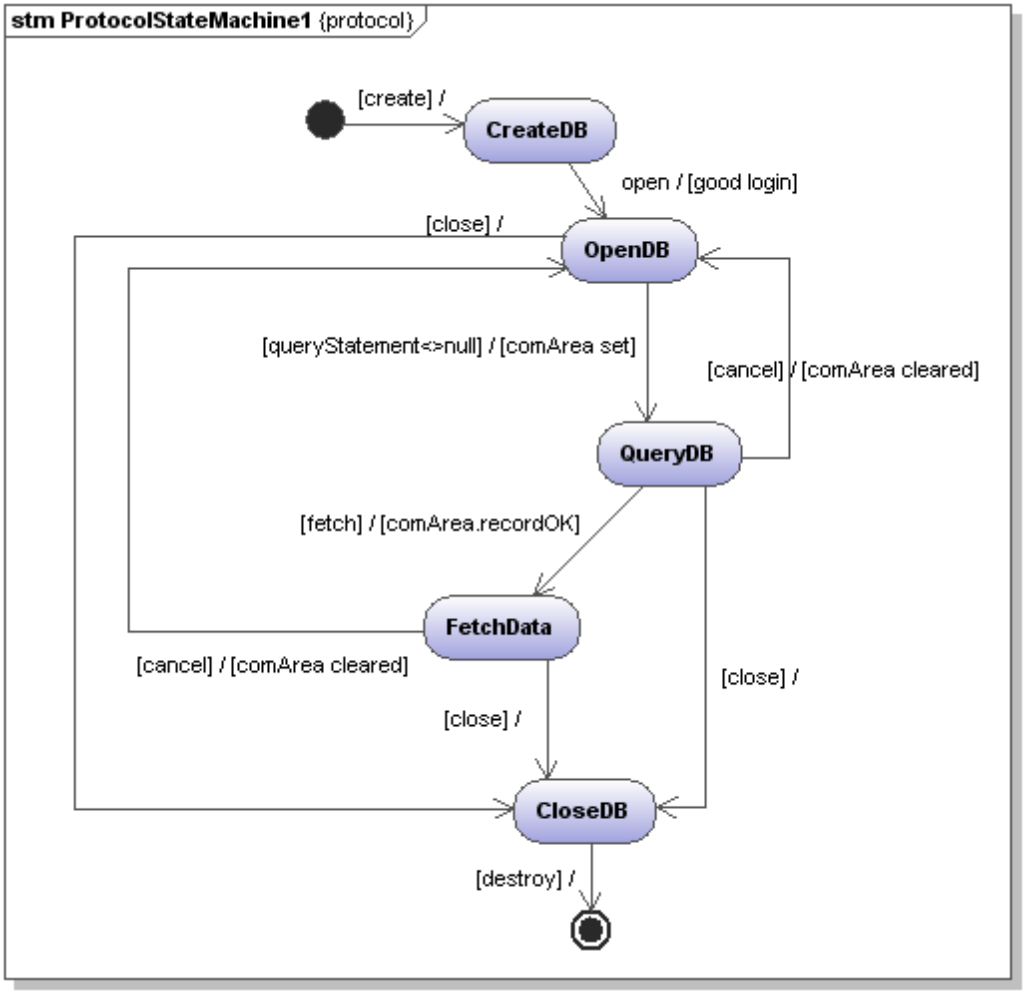
10.1.3 Diagrama de máquina de estados de protocolos

Sitio web de Altova: Diagramas de máquina de estados de protocolos UML

Las máquinas de estados de protocolos ilustran una **secuencia** de eventos a los que responde un objeto, sin necesidad de ilustrar su comportamiento propiamente dicho. La secuencia necesaria de eventos y los cambios resultantes en el estado del objeto se modelan en este tipo de diagramas.

Las máquinas de estados de protocolos se usan sobre todo para describir protocolos complejos. Por ejemplo, el acceso a bases de datos a través de una interfaz determinada o protocolos de comunicación como TCP/IP.

Las máquinas de estados de protocolos se crean igual que los diagramas de máquina de estados, pero tienen menos elementos de modelado. Las transiciones de protocolo entre los estados pueden tener condiciones previas o posteriores que definen qué debe ocurrir para que tenga lugar la transición a otro estado o cuál debe ser el estado resultante una vez tiene lugar la transición.



Insertar elementos

Usar los iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de máquina de estados de protocolos.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento.
   Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

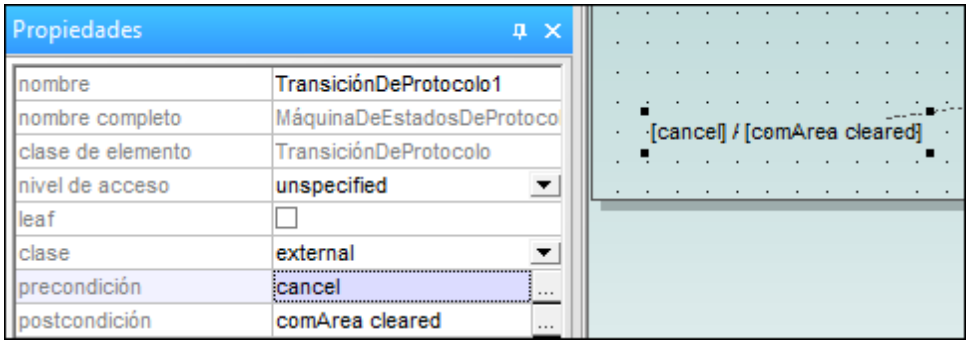
1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de máquina de estados de protocolos.

Para insertar un estado simple:

1. Haga clic en el icono **Estado** de la barra de herramientas y haga clic en el diagrama para insertarlo.
2. Escriba el nombre del estado y pulse **Entrar** para confirmar.
   Los estados simples no tienen regiones ni subestructuras.

Para crear una transición de protocolo entre dos estados:

1. Haga clic en el controlador `Transición` del estado de origen (situado a la derecha del elemento) o en el icono **TransiciónDeProtoclo** de la barra de herramientas.
2. Arrastre la flecha de la transición hasta el estado de destino.
   El cursor de texto se habilita automáticamente para que pueda insertar la condición previa o posterior. Recuerde que es obligatorio utilizar corchetes y la barra diagonal en las condiciones.
   Si inserta la condición (previa o posterior) en el panel Propiedades, los corchetes y la barra diagonal se escriben automáticamente en el diagrama.



Para crear e insertar estados compuestos y estados de submáquina:

- Consulte el apartado Estados compuestos

 © 2014 Altova GmbH

Elementos

Estado
Estado simple con un compartimiento.

Estado compuesto
Este tipo de estado contiene un compartimiento más que tiene una sola región. Dentro de esta región puede colocar un número ilimitado de estados.

Estado ortogonal
Este tipo de estado contiene un compartimiento más, formado por dos o más regiones, que indican simultaneidad.

Haga clic con el botón secundario en un estado y seleccione **Nuevo/a | Región** para añadir una región nueva.

Estado de submáquina
Este estado sirve para ocultar detalles de una máquina de estados. Este estado no tiene regiones pero está asociado a una máquina de estados distinta.

EstadoInicial (pseudoestado)
El principio del proceso

EstadoFinal
El fin de la secuencia de los procesos

PuntoDeEntrada (pseudoestado)
El punto de entrada de una máquina de estados o de un estado compuesto.

PuntoDeSalida (pseudoestado)
El punto de salida de una máquina de estados o de un estado compuesto.

Elección
Representa una rama condicional dinámica en la que se evalúan disparadores de guardas que se excluyen mutuamente (operación OR).

Unión (pseudoestado)
Representa el final de la operación OR definida por el elemento Elección.

Terminar (pseudoestado)
La detención de la ejecución de la máquina de estados.

Bifurcación (pseudoestado)
Inserta una barra de bifurcación vertical. Sirve para dividir secuencias en subsecuencias simultáneas.

Bifurcación horizontal (pseudoestado)
Inserta una barra de bifurcación horizontal. Sirve para dividir secuencias en subsecuencias simultáneas.

Reunión (pseudoestado)
Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

Reunión horizontal (pseudoestado)
Reúne/combina subsecuencias definidas previamente. Para poder continuar todas las actividades deben completarse.

ReferenciaDePuntoDeConexión
Representa un uso (como parte de un estado de submáquina) de un punto de entrada/salida definido en la referencia de máquina de estados por el estado de submáquina.

Para añadir puntos de entrada/salida en una referencia de punto de conexión:

- El estado al que está conectado el punto debe hacer referencia a una máquina de estado de submáquina (visible en el panel Propiedades).
- Esta submáquina debe contener un punto de entrada y otro de salida como mínimo.

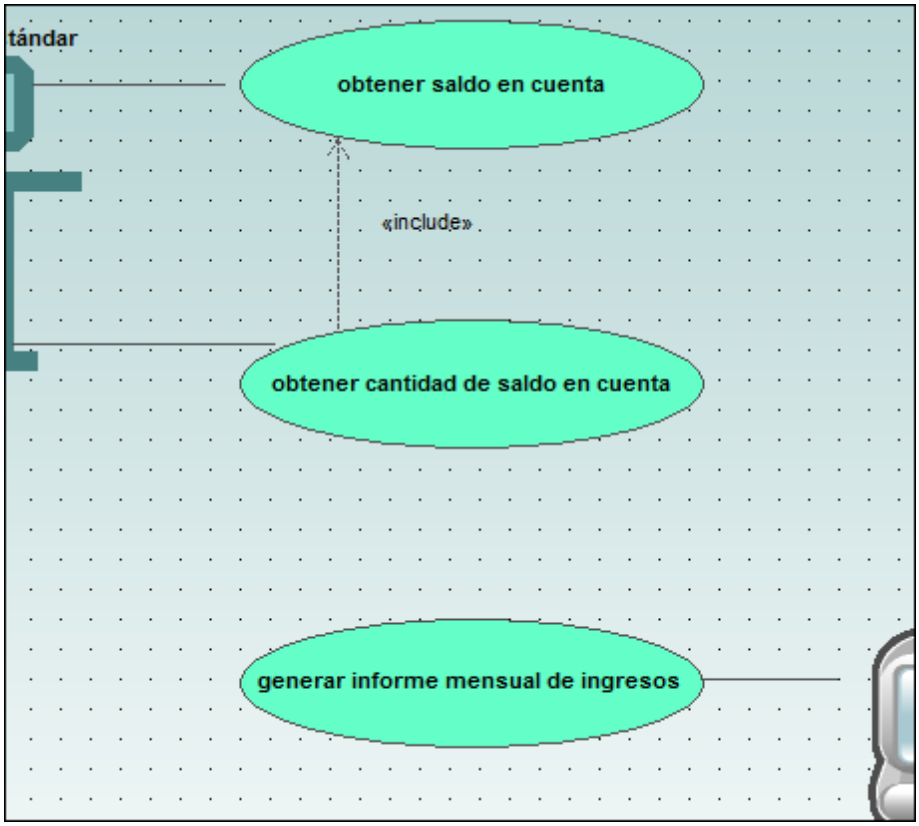
TransiciónDeProtocolo
Relación directa entre dos estados. Un objeto del primer estado realiza una operación o más y después hace referencia al segundo estado, dependiendo de un evento y de que se cumplan las condiciones previas o posteriores.

Para más información consulte el apartado Insertar elementos.

 © 2014 Altova GmbH

10.1.4 Diagrama de casos de uso

Consulte la sección Casos de uso del tutorial para obtener más información sobre cómo usar los diagramas de casos de uso.



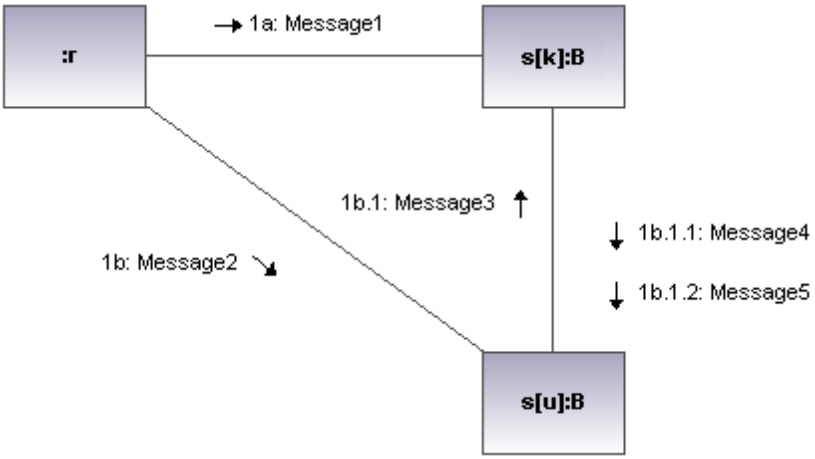
10.1.5 Diagrama de comunicación

Sitio web de Altova: Diagramas de comunicación UML

Los diagramas de comunicación muestran cómo interactúan los objetos en tiempo de ejecución (p. ej. los flujos de mensaje) e ilustran las relaciones que existen entre los objetos. Básicamente modelan el comportamiento dinámico de los casos de uso.

Los diagramas de comunicación se diseñan igual que los diagramas de secuencia, excepto que la notación tiene otro formato. Los mensajes se numeran para ilustrar su secuencia y su anidamiento.

Con UModel puede generar diagramas de comunicación a partir de diagramas de secuencia y viceversa. Para más información consulte el apartado Generar diagramas de secuencia.



Insertar elementos

Usar iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de comunicación.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de comunicación.

Línea de vida

El elemento línea de vida es un participante de la interacción. En UModel puede insertar otros elementos (clases, por ejemplo) en el diagrama de secuencia. Cada elemento aparece como una

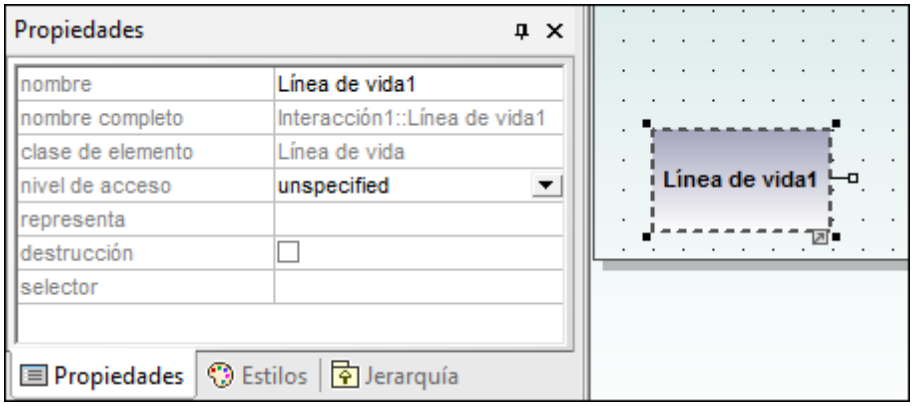
 © 2014 Altova GmbH

línea de vida nuevas. El color y el degradado de las líneas de vida se pueden redefinir en el cuadro combinado `Título - color de degradado` del panel Estilos.

Para crear un nombre de línea de vida **multilínea** pulse **Ctrl+Entrar**.

Para insertar una línea de vida de comunicación:

1. Haga clic en el icono **Línea de vida** de la barra de herramientas y después haga clic en el área de trabajo del diagrama para insertarla.



2. Escriba el nombre de la línea de vida o conserve el nombre predeterminado `Línea de vida 1`.

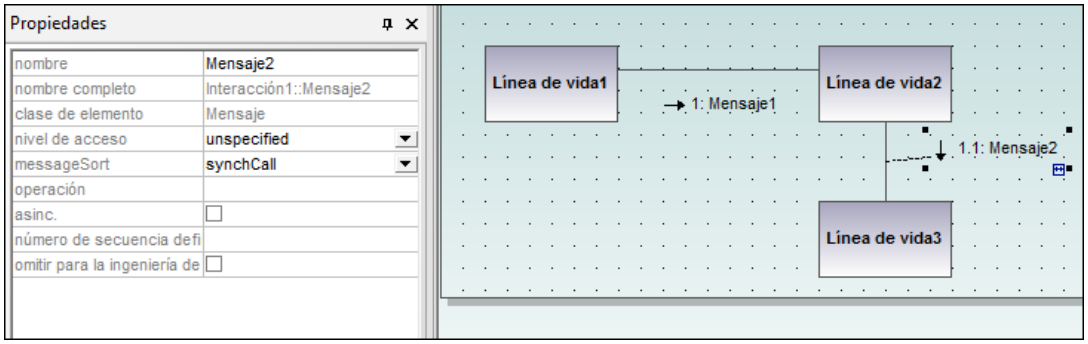
Mensajes

Un mensaje es un elemento de modelado que define un tipo concreto de comunicación en una interacción. Una comunicación puede lanzar una señal, invocar una operación, crear o destruir una instancia, etc. El mensaje especifica el tipo de comunicación, así como el remitente y el destinatario.

Mensaje (Llamada) **Mensaje (Respuesta)** **Mensaje (Creación)** **Mensaje (Destrucción)**

Para insertar un mensaje:

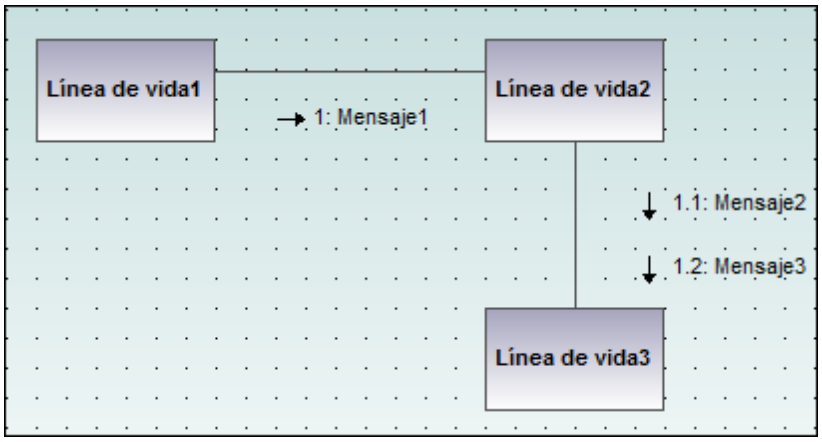
1. En la barra de herramientas haga clic en el icono del mensaje que desea insertar.
2. Ahora haga clic en el remitente y arrastre el puntero hasta el destinatario (un destinatario válido es el que aparece resaltado al pasar el puntero por encima).



Nota: mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.

Para insertar más mensajes:

1. Haga clic con el botón secundario en una línea de comunicación del diagrama y seleccione **Nuevo/a | Mensaje**.



Nota: la dirección en la que se arrastra la flecha define la dirección del mensaje. Los mensajes de respuesta pueden apuntar en ambas direcciones.

Numeración de los mensajes

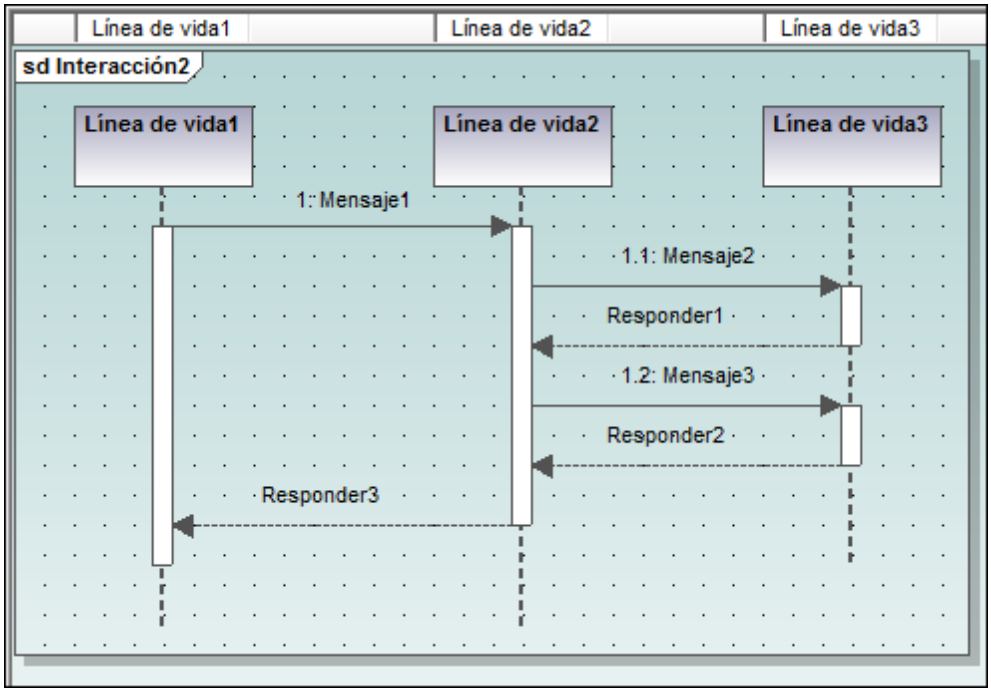
Los diagramas de comunicación utilizan la notación decimal para numerar los mensajes, lo cualquier facilita comprender la estructura jerárquica de los mensajes del diagrama. La secuencia es una lista separada por puntos de números en secuencia seguidos por dos puntos y el nombre del mensaje.

Generar diagramas de secuencia a partir de diagramas de comunicación:

UModel puede generar diagramas de comunicación a partir de diagramas de secuencia y viceversa:

- Haga clic con el botón secundario en el área de trabajo del diagrama de comunicación y seleccione **Generar diagrama de secuencia** del menú contextual.

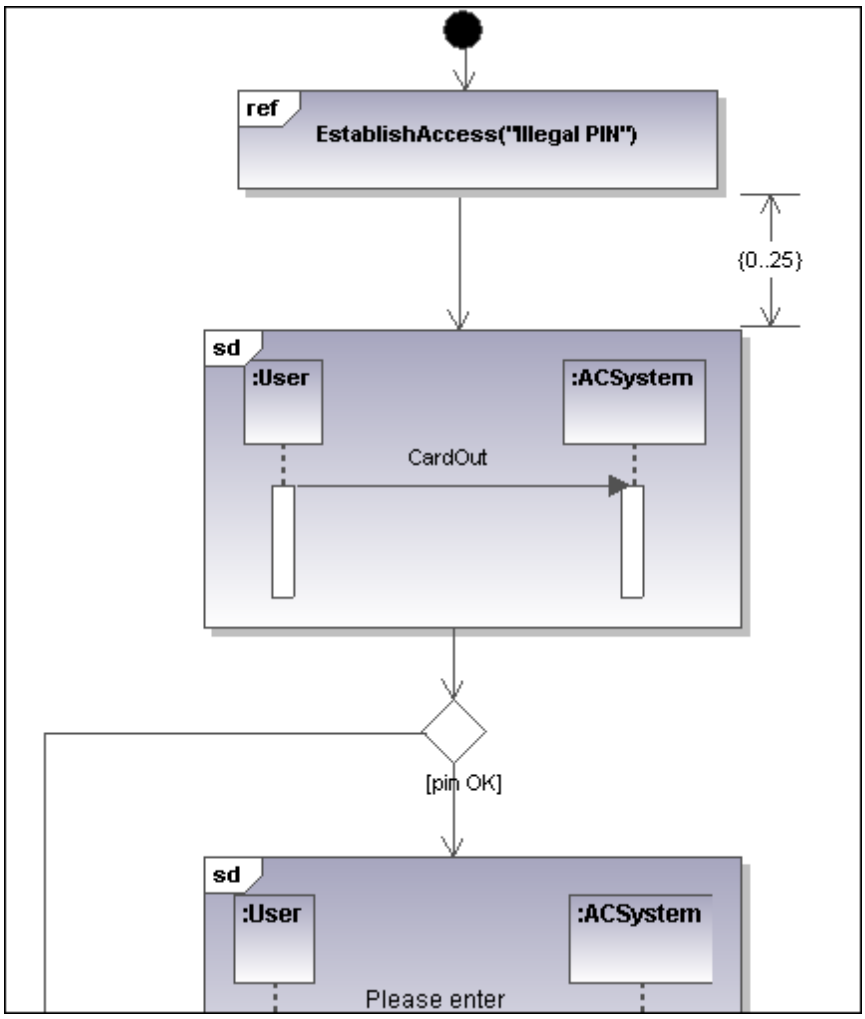
 © 2014 Altova GmbH


Línea de vida1
Línea de vida2
Línea de vida3
sd Interacción2
Línea de vida1
Línea de vida2
Línea de vida3
1: Mensaje1
1.1: Mensaje2
Responder1
1.2: Mensaje3
Responder2
Responder3

10.1.6 Diagrama global de interacción

Sitio web de Altova: Diagramas globales de interacción

Los diagramas globales de interacción son un tipo de diagrama de actividades que ofrecen un resumen de la interacción entre otros diagramas de interacción como diagramas de secuencia, de actividades, de comunicación o de ciclo de vida. El método para construir este tipo de diagramas es similar al utilizado para los diagramas de actividades y usa los mismos elementos de modelado: bifurcaciones, reuniones, nodo inicial, nodo final, etc.



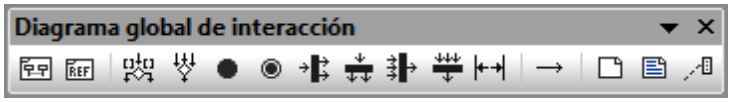
En lugar de actividades, este diagrama utiliza dos tipos distintos de interacciones: **Interacción** y **UsoDeInteracción**.

Los elementos **Interacción** se presentan como iconos de un diagrama de secuencia, comunicación, ciclo de vida o diagrama global de interacción, en un marco que tiene la abreviatura `SD` en la esquina superior izquierda.

Las instancias de los elementos **Interacción** son referencias a diagramas de interacción ya disponibles. Éstas tienen la abreviatura `Ref` y el nombre de la instancia en el marco de título.

 © 2014 Altova GmbH

Insertar elementos



Usar iconos de la barra de herramientas:

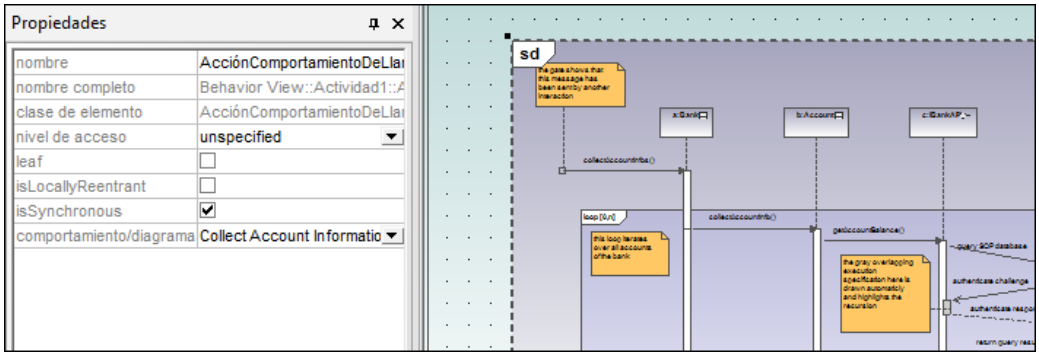
1. Haga clic en un icono de la barra de herramientas Diagrama global de interacción.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama global de interacción.

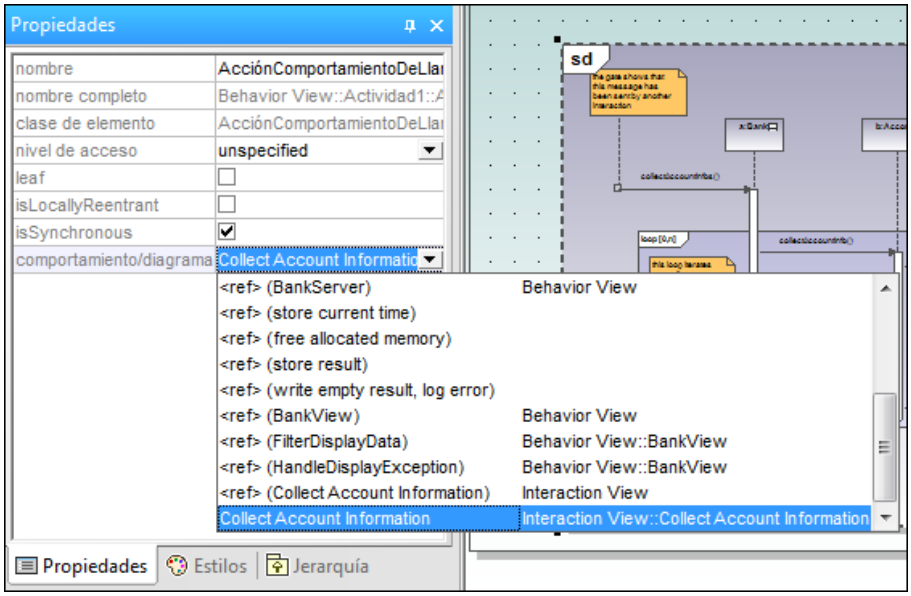
Insertar un elemento Interacción:

1. Haga clic en el icono **AcciónComportamientoDeLlamada (Interacción)** de la barra de herramientas y haga clic en el área de trabajo del diagrama para insertar la interacción.

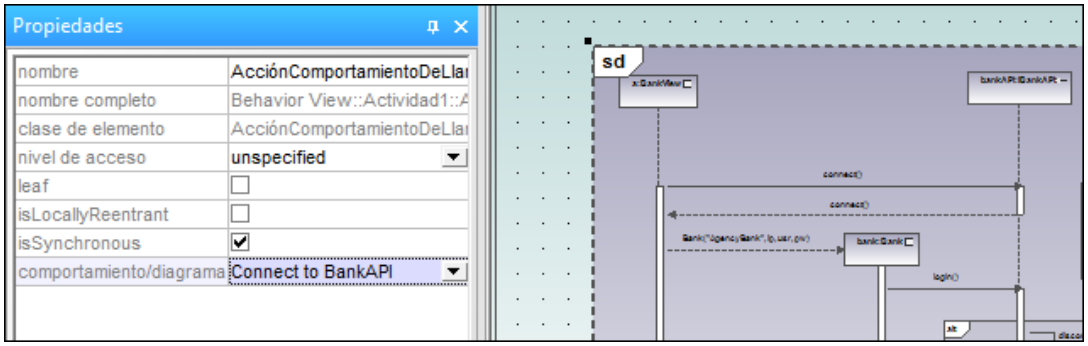


Si usa el archivo de ejemplo `Bank_MultiLanguage.ump` de la carpeta `...\UModelExamples`, el diagrama de secuencia **`Collect Account Information`** se inserta automáticamente. El primer diagrama de secuencia de la Estructura del modelo se selecciona por defecto.

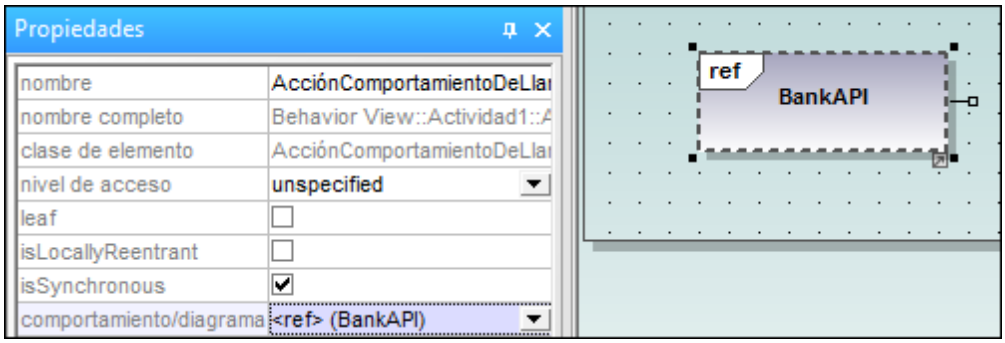
2. Para cambiar el elemento **Interacción** predeterminado, haga clic en el cuadro combinado `comportamiento/diagrama` del panel Propiedades. La lista desplegable incluye todos los elementos que se pueden insertar.



3. Haga clic en el elemento que desea insertar (p. ej. `Connect to BankAPI`).



Como este también es un diagrama de secuencia, el elemento **Interacción** aparece como un icono que representa el diagrama de secuencia.
Si selecciona `<ref> BankAPI`, aparece la instancia del elemento **Interacción**.

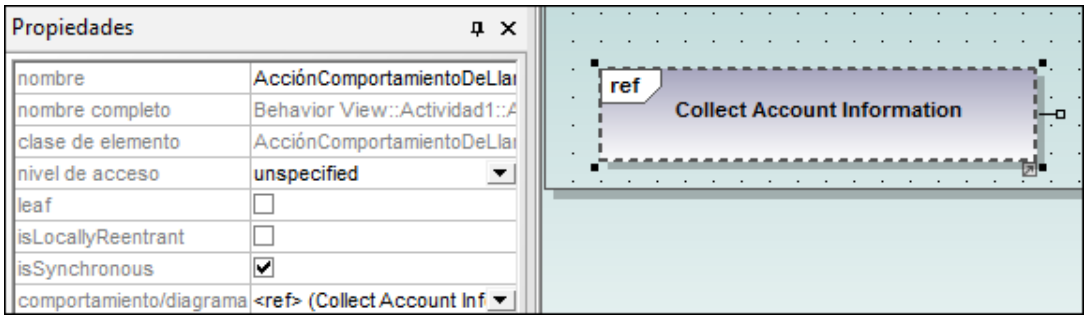


Insertar una instancia del elemento Interacción:

1. Haga clic en el icono **AcciónComportamientoDeLlamada (UsoDeInteracción)** de la barra de herramientas y haga clic en el área de trabajo del diagrama para insertar la instancia.

 © 2014 Altova GmbH

Si usa el archivo de ejemplo `Bank_MultiLanguage.ump` de la carpeta `...\UModelExamples`, UModel inserta **`Collect Account Information`** automáticamente como instancia de interacción. El primer diagrama de secuencia disponible se selecciona por defecto.



2. Para cambiar de elemento **Interacción** haga clic en el cuadro combinado `comportamiento/diagrama` del panel Propiedades. La lista desplegable incluye todos los elementos disponibles que se pueden insertar.
3. Seleccione la instancia que desea insertar.
   Recuerde que todos los elementos que se insertan de esta manera aparecen como en la imagen anterior, es decir, con la abreviatura **ref** en el marco de título.

NodoDeDecisión
Inserta un nodo de decisión que tiene una sola transición entrante y varias transiciones salientes protegidas con guardas. Para más información consulte el apartado Crear una rama.

NodoDeCombinación
Inserta un nodo de combinación que une las transiciones alternas definidas por el nodo de decisión. El nodo de combinación no sincroniza los procesos simultáneos, sino que selecciona uno de los procesos.

NodoInicial
El principio del proceso. Una interacción puede tener más de un nodo inicial.

NodoFinalDeActividad
El final del proceso de interacción. Una interacción puede tener más de un nodo final. Todos los flujos se detienen cuando se encuentra el primer nodo final.

NodoDeBifurcación
Inserta un nodo de bifurcación vertical. Sirve para dividir flujos en varios flujos simultáneos.

NodoDeBifurcación (Horizontal)
Inserta un nodo de bifurcación horizontal. Sirve para dividir flujos en varios flujos simultáneos.

NodoDeReunión
Inserta un nodo de reunión vertical. Sirve para sincronizar varios flujos definidos por el nodo de bifurcación.

NodoDeReunión (horizontal)
Inserta un nodo de reunión horizontal. Sirve para sincronizar varios flujos definidos por el nodo de bifurcación.

RestricciónDeDuración
Una duración define una EspecificaciónDeValor que denota una duración entre un punto inicial y un punto final. Las duraciones suelen ser expresiones que representan el tiempo que puede pasar.

FlujoDeControl
Un flujo de control es una línea con una flecha que conecta dos comportamientos e inicia una interacción después de que finalice la anterior.

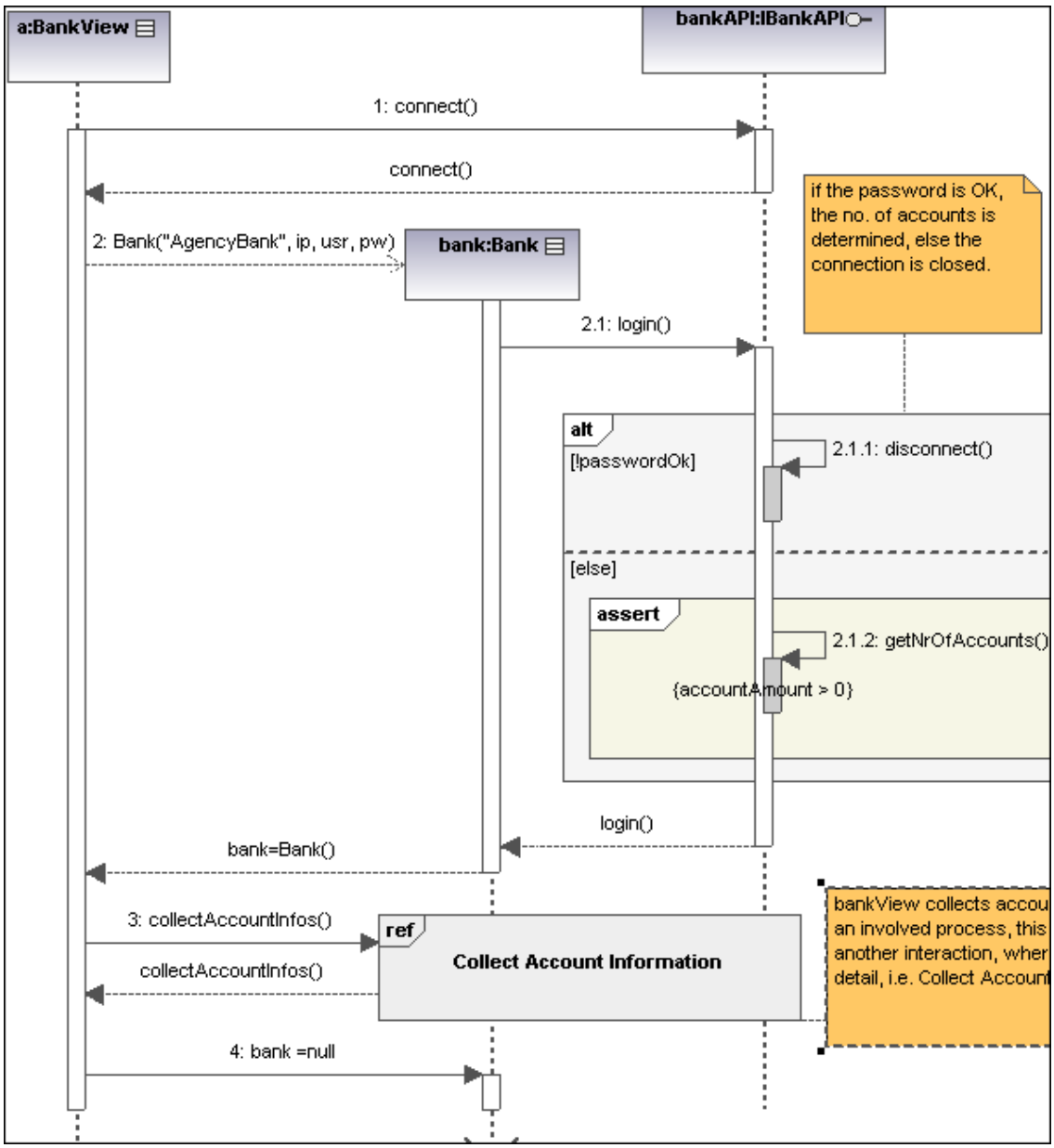
 © 2014 Altova GmbH

10.1.7 Diagrama de secuencia

Sitio web de Altova: Diagramas de secuencia UML

En UModel puede crear los diagramas de secuencia estándar definidos por UML y manipular objetos y mensajes con total facilidad para modelar casos de uso. Los diagramas de secuencia que aparecen en esta sección proceden de los proyectos de ejemplo `Bank_Java.ump`, `Bank_CSharp.ump` y `Bank_MultiLanguage.ump` del directorio `...\UModelExamples`.

Nota: también puede generar diagramas de secuencia a partir de código fuente o desde la API de UModel. Para más información consulte el apartado Generar diagramas de secuencia y el apartado Cómo crear diagramas de secuencia.



Insertar elementos

Los diagramas de secuencia modelan las interacciones dinámicas de los objetos en tiempo de ejecución por medio de mensajes. Suelen utilizarse para explicar casos de uso.

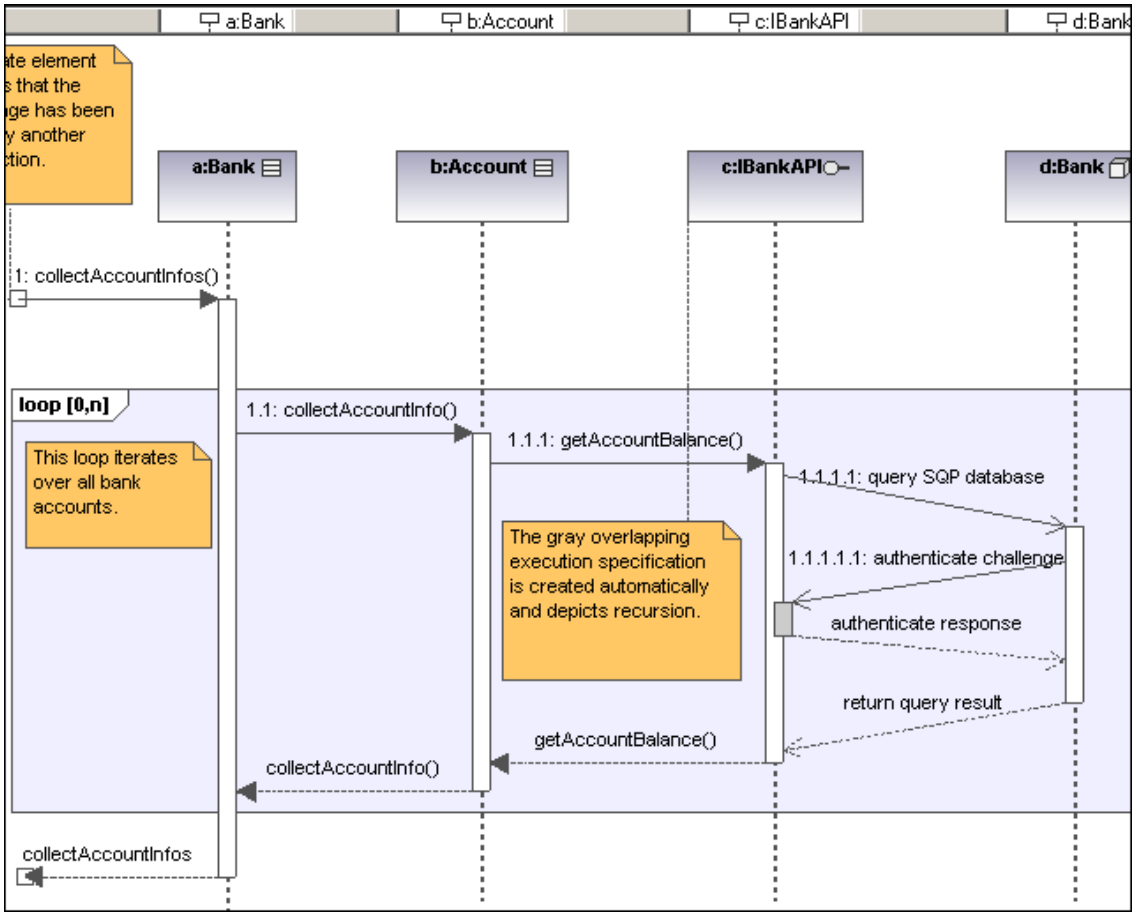
- Las **líneas de vida** son recuadros alineados horizontalmente en la parte superior del diagrama y tienen una línea de puntos vertical que representa la vida del objeto durante la interacción. Los mensajes se dibujan como flechas entre las líneas de vida de los objetos.

- Los **mensajes** se envían de un objeto a otro, se dibujan en forma de flecha y tienen una etiqueta de texto. Pueden tener un número secuencial y otros atributos opcionales como listas de argumentos, etc. Los mensajes pueden ser condicionales, opcionales y alternativos. Para más información consulte el apartado Fragmentos combinados.

Esta sección se divide en varios apartados:

Líneas de vida
Fragmentos combinados
Usos de interacción
Puertas
Invariantes de estado
Mensajes

En UModel hay varias maneras de insertar elementos en los diagramas de secuencia.



 © 2014 Altova GmbH

Usar iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de secuencia.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama:

1. En la Estructura del modelo busque el elemento que quiere insertar en el diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de secuencia.

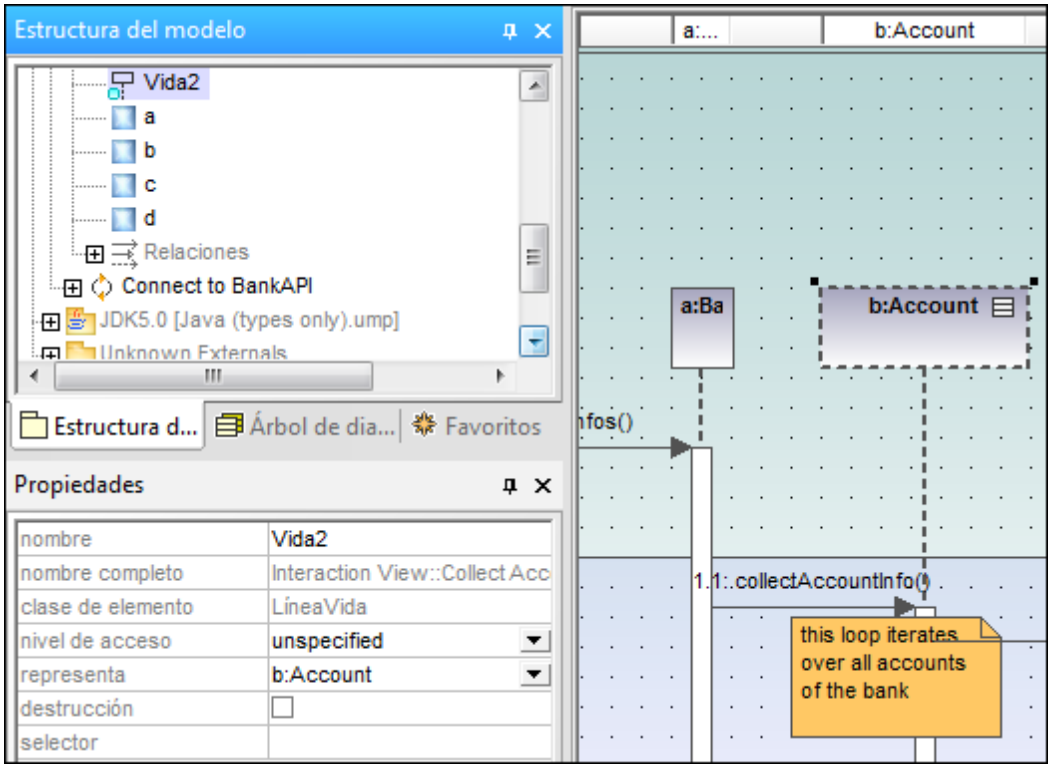
Líneas de vida

Línea de vida

Una línea de vida es un participante de una interacción. En los diagramas de secuencia de UModel también puede insertar elementos como clases y actores. Estos elementos se representan como una línea de vida nueva.

La etiqueta de la línea de vida aparece en una barra situada en la parte superior del diagrama. Puede cambiar la posición de las etiquetas y también su tamaño. Además puede redefinir el color y el degradado de las etiquetas (en el cuadro combinado `Título - color de degradado` del panel Estilos). Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la línea de vida.

En el diagrama de secuencia también puede insertar clasificadores. El campo `representa` del panel Propiedades muestra el tipo de elemento que actúa como línea de vida. Si arrastra una propiedad **con tipo** hasta el diagrama de secuencia, también se crea una línea de vida.



Especificación de ejecución (activación de objetos):
Una especificación de ejecución (activación) se representa en forma de cuadro (rectángulo) en la línea de vida del objeto. Una activación es la ejecución de un procedimiento y el tiempo necesario para ejecutar los procedimientos anidados correspondientes.

Cuando se crea un mensaje entre dos líneas de vida, se crean automáticamente los cuadros de activación.

Y un mensaje recursivo o automensaje (es decir, uno que llama a otro método de la misma clase) crea cuadros de activación apilados.

Para mostrar/ocultar los cuadros de activación:

1. Abra el panel Estilos y desplácese hasta el cuadro combinado Mostrar especificaciones de ejecución.
   En este estilo puede definir si los cuadros de activación se muestran o se ocultan en el diagrama de secuencia.

Atributos de las líneas de vida:
La casilla destrucción sirve para añadir un marcador de destrucción (o freno) a la línea de vida sin necesidad de usar un mensaje de destrucción.

En el campo selector puede insertar una expresión que indique la parte que representa la línea de vida si el `ElementoConectable` tiene más de un valor (es decir, si su multiplicidad es mayor que 1).

Ir a una línea de vida

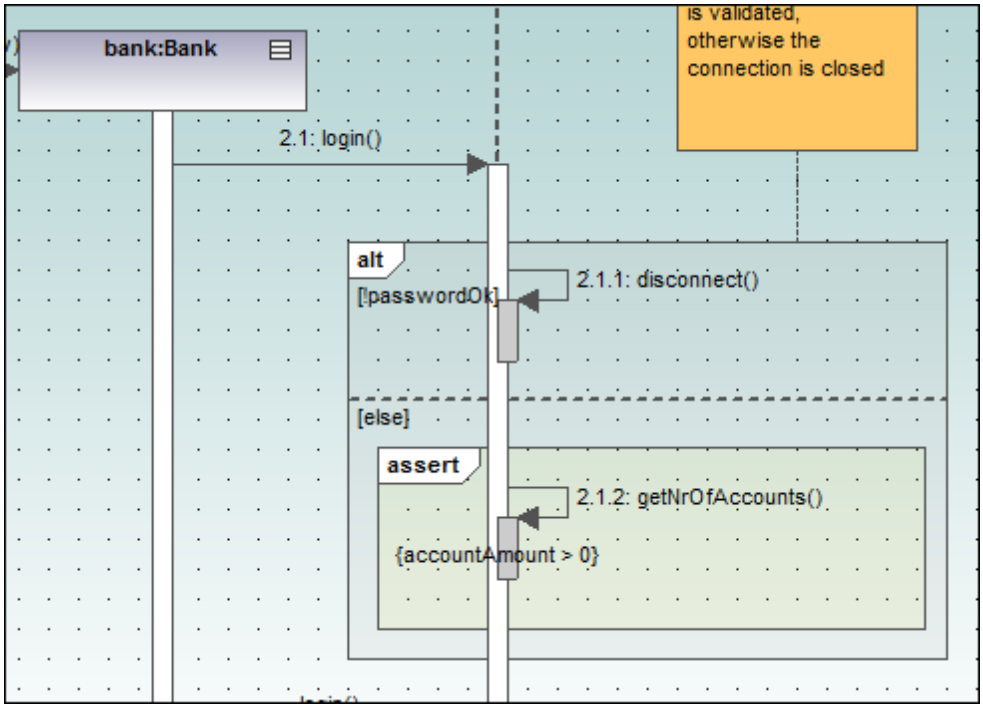
 © 2014 Altova GmbH

Haga clic con el botón secundario en una línea de vida y en el menú contextual elija la opción **Ir a XXX** (XXX es el tipo de línea de vida seleccionada). El elemento se resalta en el panel Estructura del modelo.

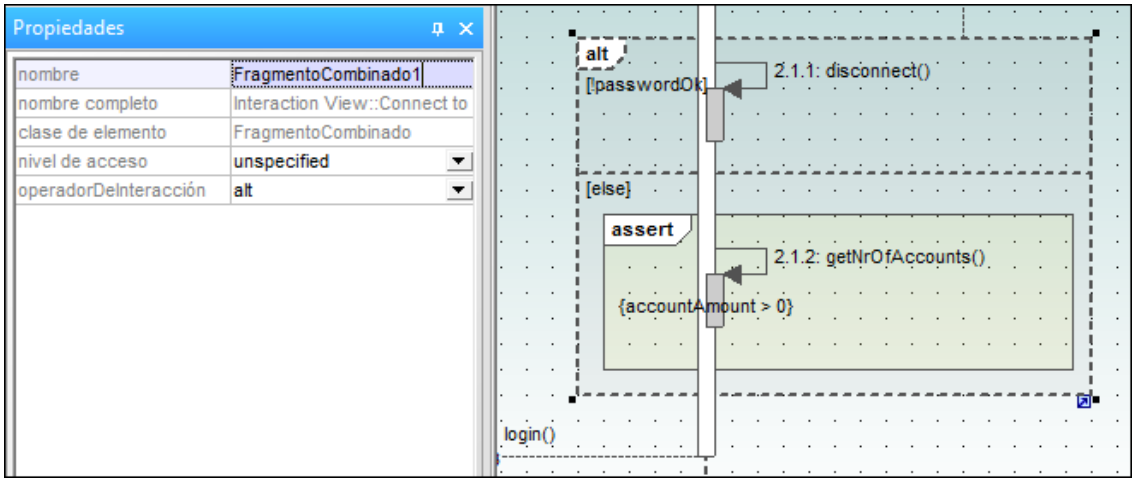
Fragmentos combinados

FragmentoCombinado

Los fragmentos combinados son subunidades o secciones de una interacción. El operador de interacción que aparece en el pentágono de la esquina superior izquierda define el tipo de fragmento combinado. Por tanto, la restricción define el tipo de fragmento (p. ej. de bucle, alternativo, etc.) utilizado en la interacción.



La barra de herramientas de los diagramas de secuencia también incluye iconos para insertar fragmentos combinados en el diagrama: **`seq`** (secuencia), **`alt`** (alternativo) o **`loop`** (bucle). Haga clic en el cuadro combinado `operadorDeInteracción` para definir el tipo de fragmento de interacción.



OperadoresDeInteracción

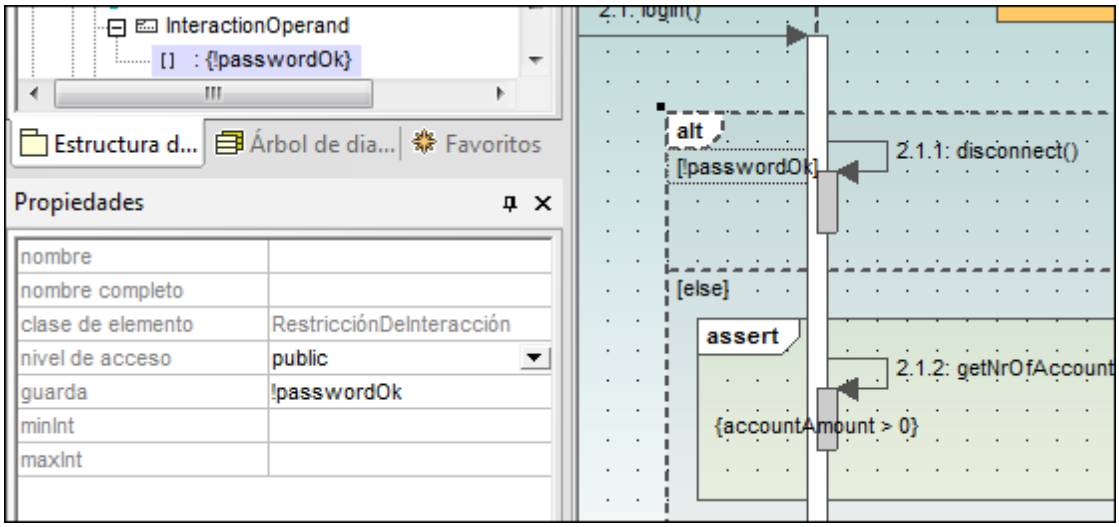
Secuencias débiles **seq**

El fragmento combinado representa secuencias débiles entre los comportamientos de los operandos.

Alternativas **alt**

Solo se eligirá uno de los operandos definidos. El operando debe tener una expresión de guarda cuyo resultado sea `true`.

Si uno de los operandos utiliza el guarda "else", el operando se ejecuta si todos los demás guardas devuelven `false`. La expresión de guarda se puede introducir inmediatamente después de la inserción (y aparecerá entre corchetes).



 © 2014 Altova GmbH

La **RestricciónDeInteracción** es de hecho la expresión de guarda que aparece entre corchetes.

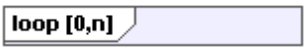
Opción **opt**
Representa una opción entre ejecutar el operando o no hacer nada.

Pausa **break**
El operador **break** se elige cuando el guarda es `true`. El resto del fragmento se ignora.

Paralelo **par**
Indica que el fragmento combinado representa una combinación paralela de operandos.

Secuencias estrictas **strict**
El fragmento combinado representa una secuencia estricta entre los comportamientos de los operandos.

Bucle **loop**
El operando **loop** se repetirá tantas veces como defina la expresión de guarda.



Tras seleccionar este operando puede editar la expresión directamente (en el pentágono `loop`) haciendo doble clic.

Región crítica **critical**
El fragmento combinado representa una región crítica. La secuencia no se puede interrumpir ni intercalar con otros procesos.

Negativo **neg**
El fragmento no es válido y los demás se suponen válidos.

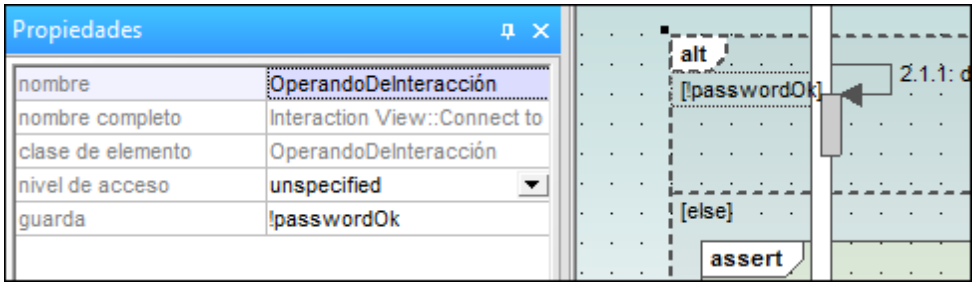
Aserción **assert**
Designa el fragmento combinado válido y sus secuencias. Se suele usar junto con los operandos **consider** o **ignore**.

Ignorar **ignore**
Define qué mensajes deben ignorarse en la interacción. Se suele usar junto con los operandos **assert** o **consider**.

Considerar **consider**
Define qué mensajes se deben tener en cuenta en la interacción. Se suele usar junto con los operandos **assert** o **ignore**.

Agregar operandosDeInteracción a un fragmento combinado:

1. Haga clic con el botón secundario en el fragmento combinado y seleccione **Nuevo/a | OperandoDeInteracción**.
   La condición de guarda se puede editar inmediatamente.
2. Inserte la condición de guarda para el **OperandoDeInteracción** (p. ej. `!passwordOK`) y pulse **Entrar** para confirmar.



Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre del operando de interacción.

3. Use el mismo método para añadir otro operando de interacción con la condición de guarda "else".
   Los operandos aparecen separados por líneas de puntos en el fragmento.

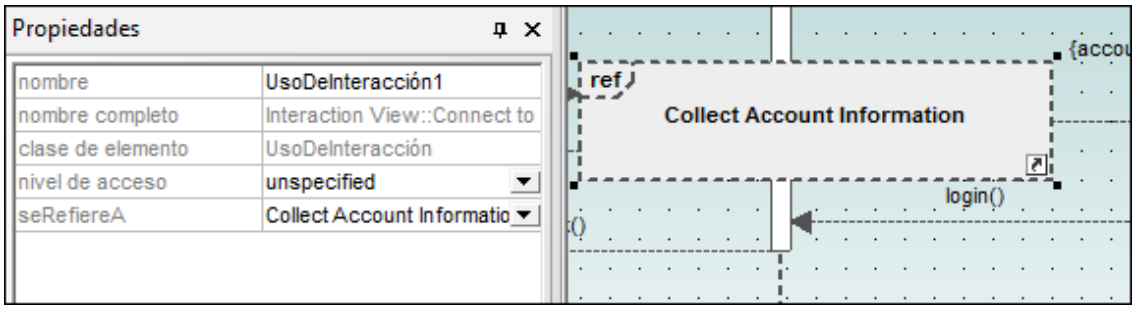
Eliminar operandos de interacción:

1. Haga doble clic en la expresión de guarda del fragmento combinado en el área de trabajo del diagrama (no en el panel Propiedades).
2. Elimine la expresión de guarda y pulse **Entrar** para confirmar.
   Como resultado se elimina la expresión de guarda / el operando de interacción y el tamaño del fragmento combinado se ajusta automáticamente.

Usos de interacción

UsoDeInteracción

El elemento **UsoDeInteracción** es una referencia a un elemento de interacción y sirve para compartir porciones de una interacción con otras interacciones.



La casilla `seRefiereA` del panel Propiedades sirve para seleccionar la interacción a la que desea hacer referencia. El nombre del uso de interacción seleccionado aparecerá en el elemento.

Nota: también puede arrastrar un **`UsoDeInteracción`** desde la Estructura del modelo hasta el área de trabajo del diagrama.

Puertas

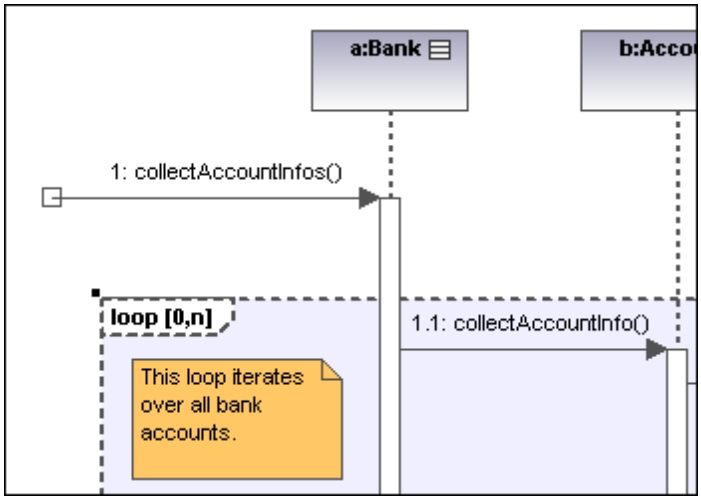
Puerta

Una puerta es un punto de conexión que permite transmitir mensajes de un fragmento a otro de la interacción. Las puertas se conectan por medio de mensajes.

1. Inserte una puerta en el diagrama.
2. Cree un mensaje nuevo, haga clic en la puerta y arrastre el puntero hasta la línea de vida (o desde la línea de vida hasta la puerta).

 © 2014 Altova GmbH

Esto conecta los dos elementos. El cuadrado pequeño representa la puerta.



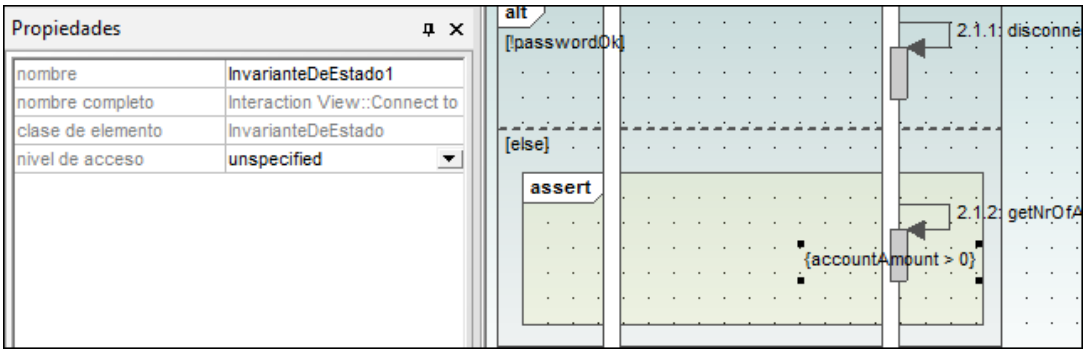
Invariantes de estado

InvarianteDeEstado

Una invariante de estado es una condición o restricción aplicada a una línea de vida. Para que exista la línea de vida es obligatorio que la condición se cumpla.

Para definir una InvarianteDeEstado:

1. Haga clic en el icono **InvarianteDeEstado** de la barra de herramientas y después en la línea de vida o en la activación de objetos.
2. Inserte la condición/restricción que desea aplicar (p. ej. `accountAmount > 0`) y pulse **Entrar** para confirmar.



Mensajes

Las líneas de vida envían y reciben mensajes que se representan en forma de flechas etiquetadas. Los mensajes pueden estar numerados de forma secuencial y tener atributos opcionales (como listas de argumentos, etc.). Los mensajes se presentan de arriba a abajo, es decir, el eje vertical es el componente de tiempo del diagrama de secuencia.

- Una **llamada** es una comunicación síncrona o asíncrona que invoca una operación que permite al control volver al objeto remitente. La flecha de una llamada apunta a la parte superior de la activación que inicia la llamada.
- La **recursión** (o llamada a otra operación del mismo objeto) se representa apilando recuadros de activación (`EspecificacionesDeEjecución`).

Para insertar un mensaje:

1. En la barra de herramientas Diagrama de secuencia haga clic en el icono del mensaje que desea insertar.
2. Haga clic en la línea de vida o cuadro de activación del objeto remitente.
3. Arrastre la línea del mensaje hasta la línea de vida o el cuadro de activación del objeto destinatario.
   Si el mensaje se puede colocar en una línea de vida, esta se resalta.

- La dirección en la que se arrastra la flecha define la dirección del mensaje. Los mensajes de respuesta pueden apuntar en ambas direcciones.
- Mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.
- En los objetos remitentes/destinatarios se crean automáticamente cuadros de activación. Para ajustar el tamaño de los cuadros a mano haga clic en los controladores de tamaño y arrástrelos.
- La secuencia de numeración se actualiza, dependiendo de las opciones de numeración que estén activas.

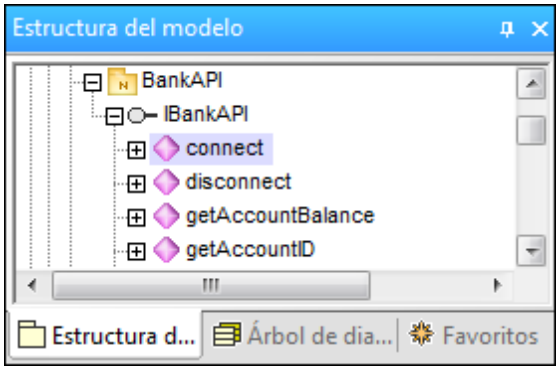
Para eliminar un mensaje:

1. Haga clic en el mensaje.
2. Pulse la tecla **Supr** para eliminar el mensaje del modelo. Para eliminarlo del diagrama solamente, haga clic con el botón secundario en el mensaje y elija **Eliminar solo en el diagrama**.
   La numeración de los mensajes y los cuadros de activación de los demás objetos se actualizan.

"Ir a la operación" para mensajes de llamada:

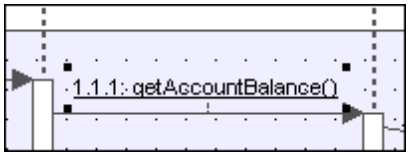
En los diagramas de secuencia y de comunicación puede buscar las operaciones a las que hacen referencia los mensajes de llamada.

1. Haga clic con el botón secundario en un mensaje de llamada y elija la opción **Ir a la operación**.
   La operación aparece resaltada en el panel Estructura del modelo.



Nota: los nombres de operaciones estáticas aparecen subrayados.

 © 2014 Altova GmbH



Para cambiar la posición de mensajes dependientes:

1. Haga clic en el mensaje que quiere mover y arrástrelo hasta su nueva posición.
   Cuando cambia la posición de un mensaje, UModel mueve también los mensajes dependientes relacionados con el mensaje activo.

   Para seleccionar varios mensajes utilice **Ctrl+clic**.

Para cambiar la posición de los mensajes uno por uno:

1. Desactive el icono **Activar/desactivar el movimiento de mensajes dependientes** de la barra de herramientas.
2. Haga clic en el mensaje que desea mover y arrástrelo hasta su nueva posición.
   En este caso solamente se mueve el mensaje seleccionado. Este mensaje se puede colocar en cualquier posición del eje vertical entre las líneas de vida de los objetos.

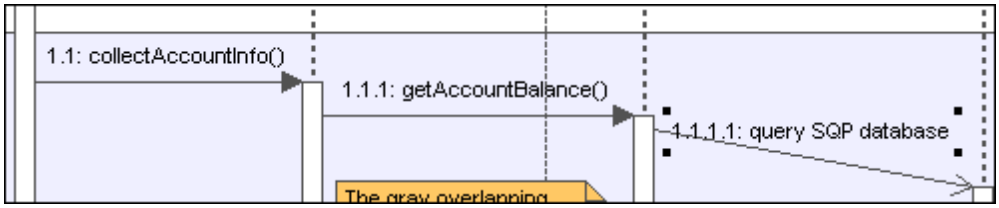
Para crear mensajes de respuesta automáticamente:

1. Active el icono **Activar/desactivar la creación automática de respuestas para mensajes**.
2. Cree un mensaje nuevo entre dos líneas de vida.
   UModel inserta automáticamente un mensaje de respuesta.

Numeración de los mensajes:

Puede usar tres tipos de numeración para los mensajes: numeración anidada, numeración sencilla o ninguna numeración.

- **Sin numeración**: este icono elimina la numeración de todos los mensajes.
- **Sencilla**: asigna una secuencia numérica a todos los mensajes de arriba a abajo, es decir, en el orden en el que aparecen a lo largo del eje temporal del diagrama.
- **Anidada**: utiliza la notación decimal, que permite ver la estructura jerárquica de los mensajes del diagrama. La secuencia de numeración es una lista de números secuenciales separada por puntos, seguidos de dos puntos y del nombre del mensaje.



Para seleccionar el tipo de numeración:

Hay dos formas de seleccionar el tipo de numeración:

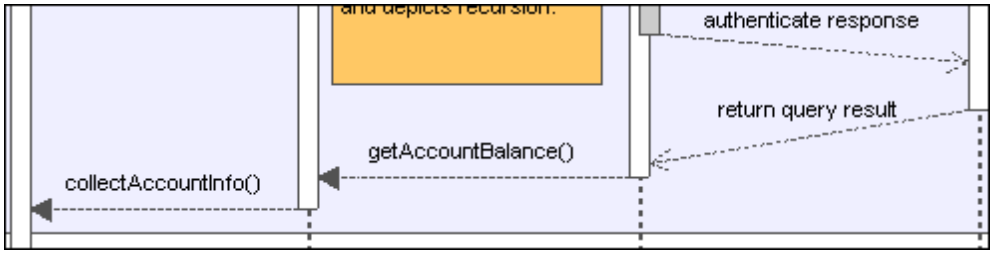
- Con el icono correspondiente de la barra de herramientas Diagrama de secuencia.
- En el panel Estilos.

Para seleccionar el tipo de numeración en el panel Estilos:

1. Haga clic en la pestaña *Estilos* y desplácese hasta el campo `Numeración de los mensajes`.
2. Haga clic en el cuadro combinado y seleccione el tipo de numeración en la lista desplegable.
   La opción de numeración seleccionada aparece en el diagrama de secuencia automáticamente.

   Nota: en ocasiones el tipo de numeración no numera todos los mensajes correctamente si existen ambigüedades. Si esto ocurre, puede solucionar el problema añadiendo mensajes de respuesta.

Mensajes de respuesta:
Los mensajes de respuesta se representan con flechas de línea discontinua.

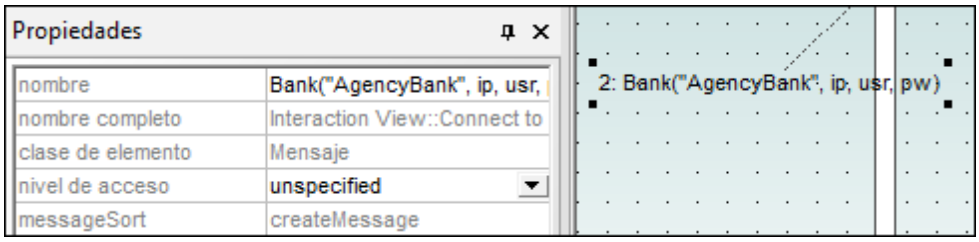


Por lo general, los mensajes de respuesta vienen señalados por la parte inferior del cuadro de activación. Si los cuadros de activación están deshabilitados (panel Estilos, `Mostrar especificaciones de ejecución=false`), entonces se recomienda usar flechas de respuesta para evitar ambigüedades.

Si activa el icono **Activar/desactivar la creación automática de respuestas para mensajes** , cada vez que cree un mensaje de llamada entre dos líneas de vida/cuadros de activación UModel creará mensajes de respuesta sintácticamente correctos de forma automática.

Para crear objetos con mensajes:

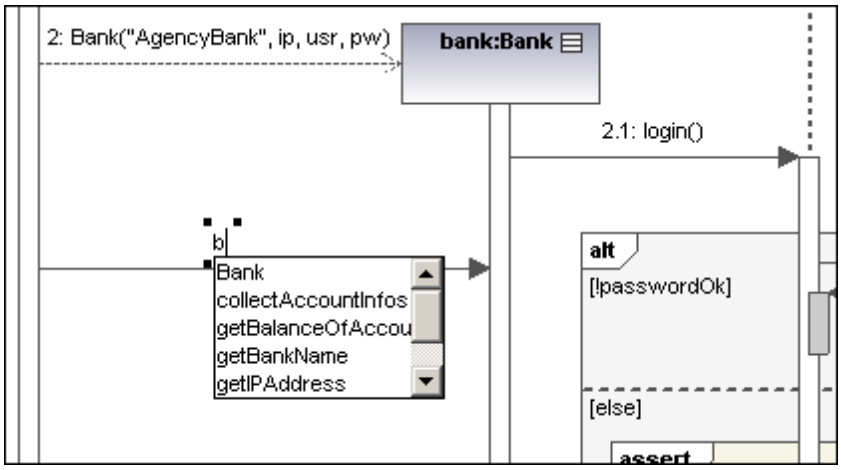
1. Los mensajes pueden crear objetos nuevos. Esto se hace con el icono **Mensaje (Creación)** .
2. Arrastre la flecha del mensaje hasta la línea de vida de un objeto para crear ese objeto. Este tipo de mensaje termina en la mitad del rectángulo del objeto y a menudo pone el cuadro del objeto en posición vertical.



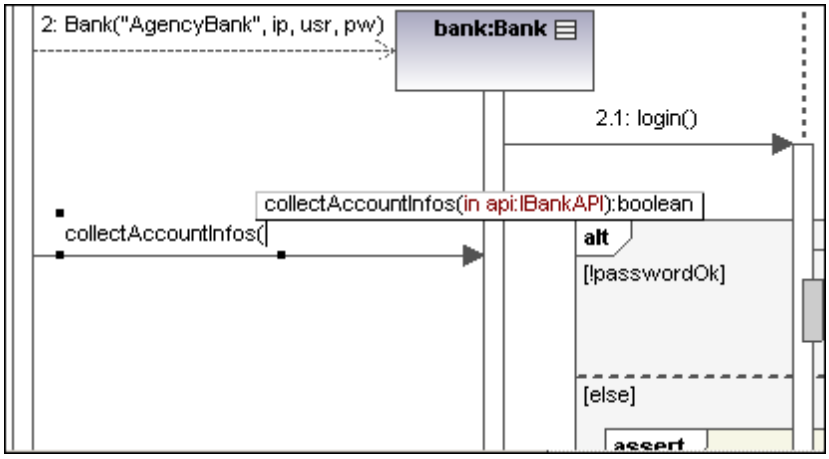
 © 2014 Altova GmbH

Para enviar mensajes a métodos/operaciones de clases de diagramas de secuencia:
Tras insertar una clase de la Estructura del modelo en un diagrama de secuencia, puede crear un mensaje entre una línea de vida y un método de la clase destinataria (línea de vida). Para ello puede usar la ayuda sintáctica y de las funciones de finalización automática de UModel.

1. Cree un mensaje entre dos líneas de vida (el objeto destinatario debe ser una línea de vida de una clase).
   En cuanto suelte la flecha del mensaje, el nombre del mensaje se resalta automáticamente.
2. Escriba un carácter (p. ej. "b").
   Aparece una ventana emergente que enumera los métodos de clase ya existentes.



3. Seleccione una operación de la lista y pulse **Entrar** para confirmar (p. ej. `collectAccountInfos`).
4. Pulse la barra espaciadora y después la tecla **Entrar** para seleccionar el paréntesis que sugiere automáticamente UModel.
   Ahora aparece una ventana de ayuda sintáctica, que le ayuda a insertar correctamente el parámetro.



Crear operaciones en clases referenciadas:
Si activa el icono **Activar/desactivar la creación automática de operaciones en el destino**

al escribir el nombre de la operación , cada vez que cree un mensaje y escriba un nombre (p. ej. `miOperación()`) UModel creará automáticamente la clase correspondiente en la clase referenciada.

Nota: solamente se pueden crear operaciones automáticamente cuando la línea de vida hace referencia a una clase, a una interfaz...

Iconos de la barra de herramientas para trabajar con mensajes:

Mensaje (Llamada)

Mensaje (Respuesta)

Mensaje (Creación)

Mensaje (Destrucción)

Mensaje asíncrono (Llamada)

Mensaje asíncrono (Respuesta)

Mensaje asíncrono (Destrucción)

Activar/desactivar el movimiento de mensajes dependientes

Activar/desactivar la creación automática de respuestas para mensajes

Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

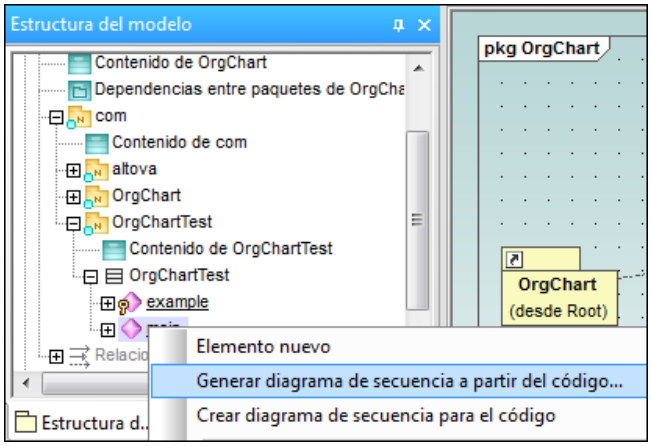
Generar diagramas de secuencia a partir de código fuente

UModel puede modelar diagramas de secuencia a partir de código o desde la API de UModel. Consulte el apartado Crear diagramas de secuencia para más información.

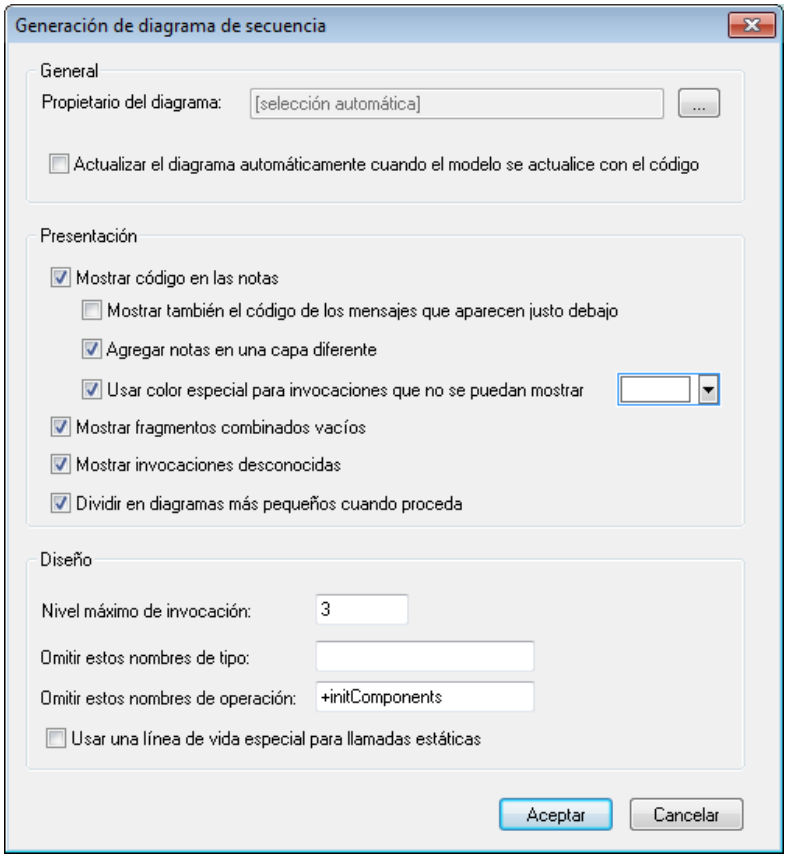
Con el siguiente ejemplo puede aprender a crear un diagrama de secuencia a partir de un método automáticamente. Si quiere, puede hacer lo mismo con su propio código. El método está en el paquete `OrgChart`, que se importó al modelo con el comando **Proyecto | Importar directorio de código fuente** (para más información consulte el apartado Ingeniería de ida y vuelta (código - modelo - código).

 © 2014 Altova GmbH

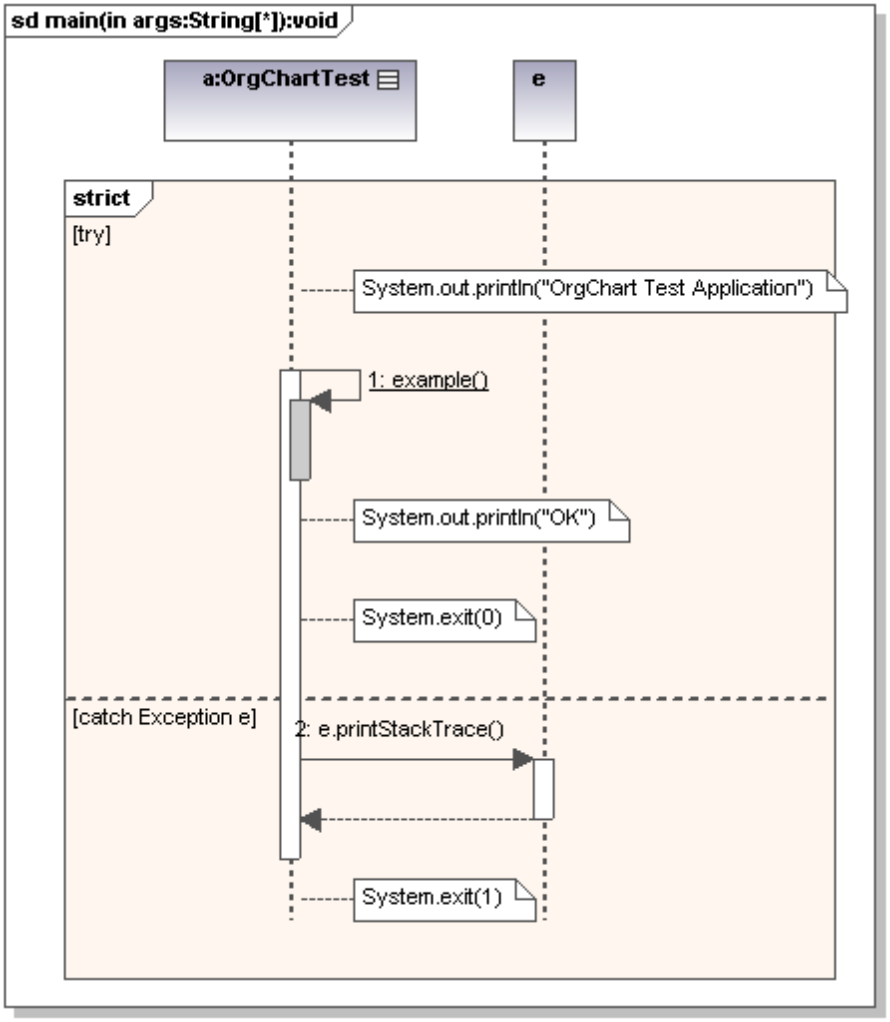
1. Una vez importado el código, haga clic con el botón secundario en el método `main` de la clase `OrgChartTest` en la Estructura del modelo. Ahora elija el comando **Generar diagrama de secuencia a partir del código** en el menú contextual.



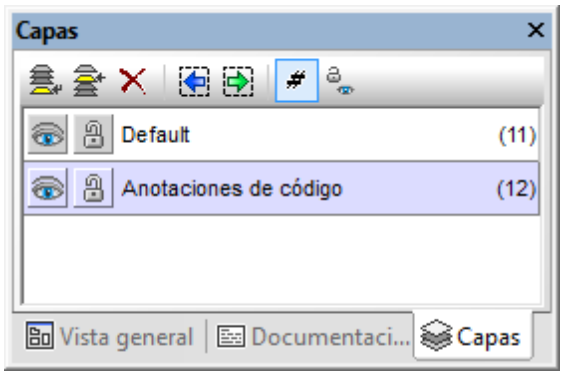
Esto abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar la generación.



2. Seleccione las opciones de presentación y diseño y después haga clic en **Aceptar**.
La imagen siguiente muestra el diagrama de secuencia que se genera con las opciones seleccionadas en la imagen anterior.



La opción **Agregar notas en una capa diferente** asigna las notas de código a la capa `Anotaciones de código`. Las notas de código aparecen en color naranja.



Notas:

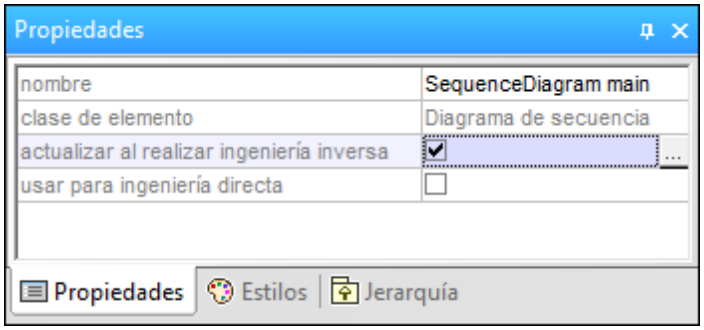
- Puede asignar un color distinto a las invocaciones que no se puedan mostrar.
- El nivel máximo de invocación define el nivel de recursión que se debe usar en el diagrama.

 © 2014 Altova GmbH

- La opción *Omitir estos nombres de tipo:* sirve para crear una lista delimitada por comas de los tipos que no deben aparecer en el diagrama de secuencia generado.
- La opción *Omitir estos nombres de operación:* sirve para crear una lista delimitada por comas de las operaciones que no deben aparecer en el diagrama de secuencia generado (**`initComponents`** se añade automáticamente a la lista).

  Al añadir nombres de operación a la lista (p. ej. **`InitComponents`**), se omite la operación completa. Si añade el carácter + delante del nombre de la operación (p. ej. **`+InitComponent`**), las llamadas de la operación aparecen en el diagrama, pero sin su contenido.
- La opción *Dividir en diagramas más pequeños cuando proceda* divide automáticamente los diagramas de secuencia en varios subdiagramas y genera hipervínculos entre ellos para poder navegar por ellos con facilidad.

Los diagramas de secuencia se actualizan automáticamente cuando se actualiza el proyecto de UModel entero. Si, por el contrario, solo se actualizan las clases o algunos archivos desde una aplicación externa, los diagramas no se actualizan. No obstante, esto puede configurarse marcando la casilla del campo `Actualización de código automática`.



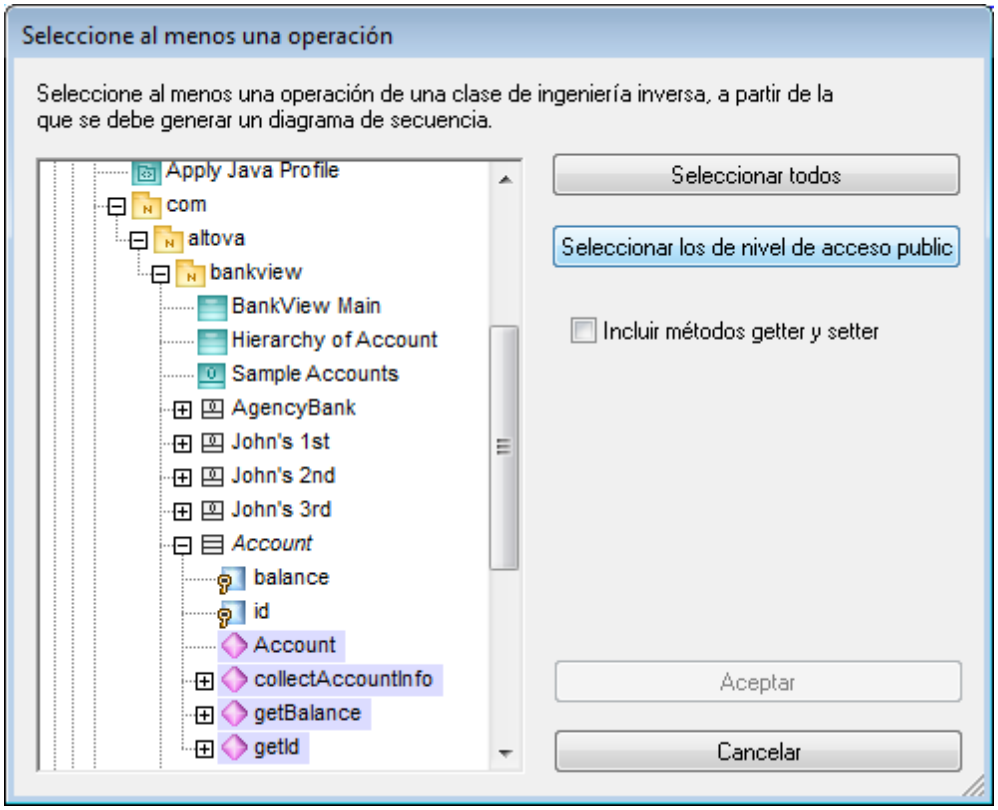
Haga clic en el icono ... del campo `Actualización de código automática` para abrir el cuadro de diálogo "Generación de diagrama de secuencia" y cambiar las opciones de generación.

Generar varios diagramas de secuencia a partir de propiedades

Además de crear diagramas de secuencia a partir de operaciones, UModel puede crear diagramas de secuencia a partir de propiedades Getter/Setter.

Crear varios diagramas de secuencia a partir de varias operaciones:

1. Seleccione la opción de menú **Proyecto | Generar diagramas de secuencia a partir del código**.
   Aparece el cuadro de diálogo "Seleccione al menos una operación".



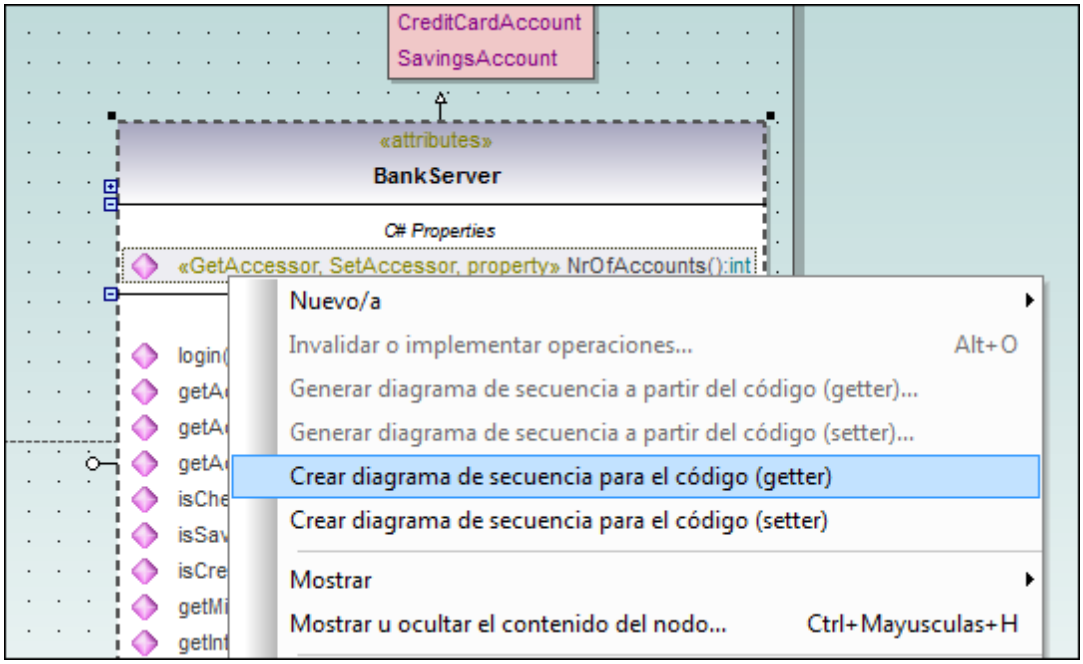
2. Seleccione las operaciones para las que desea generar un diagrama de secuencia y haga clic en **Aceptar** (si quiere puede usar los botones **Seleccionar todos** y **Seleccionar los de nivel de acceso public**). Al hacer clic en **Aceptar** se abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede elegir las opciones de generación.
3. Haga clic en **Aceptar** para generar los diagramas de secuencia. Por cada operación seleccionada se genera un diagrama de secuencia distinto.

   Nota: cuando genere diagramas de secuencia también puede elegir si se incluyen o excluyen los getter y setter.

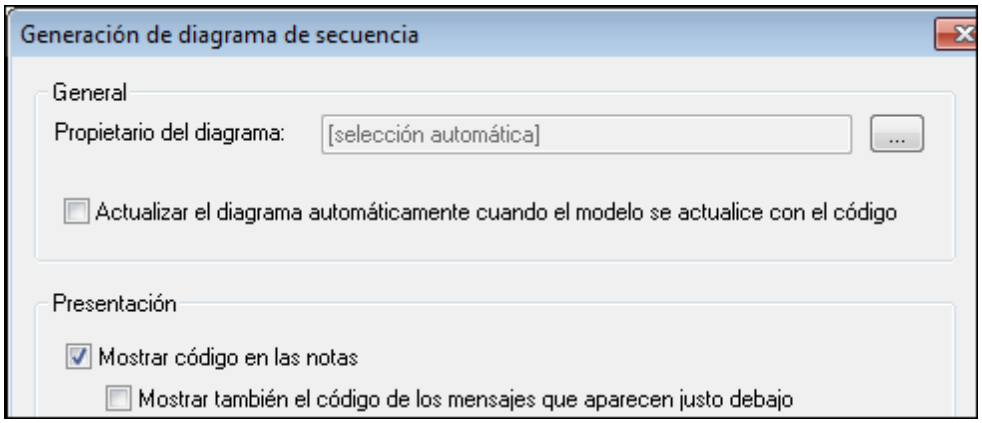
Crear un diagrama de secuencia a partir de propiedades getter/setter (C#, VB .NET)

1. Haga clic con el botón secundario en una operación que tenga el estereotipo `GetAccessor/SetAccessor`.

 © 2014 Altova GmbH



2. Seleccione la opción del menú contextual correspondiente (p. ej. **Crear diagrama de secuencia para el código (getter)**).
   Esto abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar la presentación del diagrama de secuencia que se generará.



3. Haga clic en **Aceptar** para generar el diagrama de secuencia.

Generar código a partir de diagramas de secuencia

UModel puede crear código a partir de un diagrama de secuencia que esté vinculado a una operación o a varias.

A partir de diagramas de secuencia puede generar código para:

- VB.NET, C# y Java.
- UModel y la edición Eclipse y Visual Studio de UModel.
- las tres ediciones de UModel.

Hay dos maneras de crear código a partir de diagramas de secuencia:

- Comenzando con una operación creada con ingeniería inversa (consulte el apartado

Generar diagramas de secuencia a partir de código fuente).

- Creando desde cero un diagrama de secuencia **nuevo** que esté vinculado a una operación (haciendo clic con el botón secundario en la Estructura del modelo y seleccionando **Crear diagrama de secuencia para el código**).

Nota: cuando use como base un diagrama de secuencia creado por ingeniería inversa, compruebe que la opción *Mostrar código en las notas* esté activa durante el proceso de ingeniería inversa. Así no perderá código cuando vuelva a iniciar el proceso de ingeniería directa.

Esto se debe a que UML no puede mostrar todas las características de los lenguajes VB.NET, Java y C# en el diagrama de secuencia y las características que no puede mostrar se presentan como notas.

Para agregar texto sin formato como código durante la creación de diagramas de secuencia:

1. Anexe una nota a una línea de vida del diagrama de secuencia.
2. Escriba el código que se debe escribir en el código fuente final.
   Marque la casilla `Es código` (panel Propiedades) de la nota para poder acceder a ella.

   Para ver un ejemplo consulte el apartado Agregar código a los diagramas de secuencia.

Si quiere que un diagrama de secuencia se utilice automáticamente para ingeniería de código cada vez que se inicie la ingeniería de código:

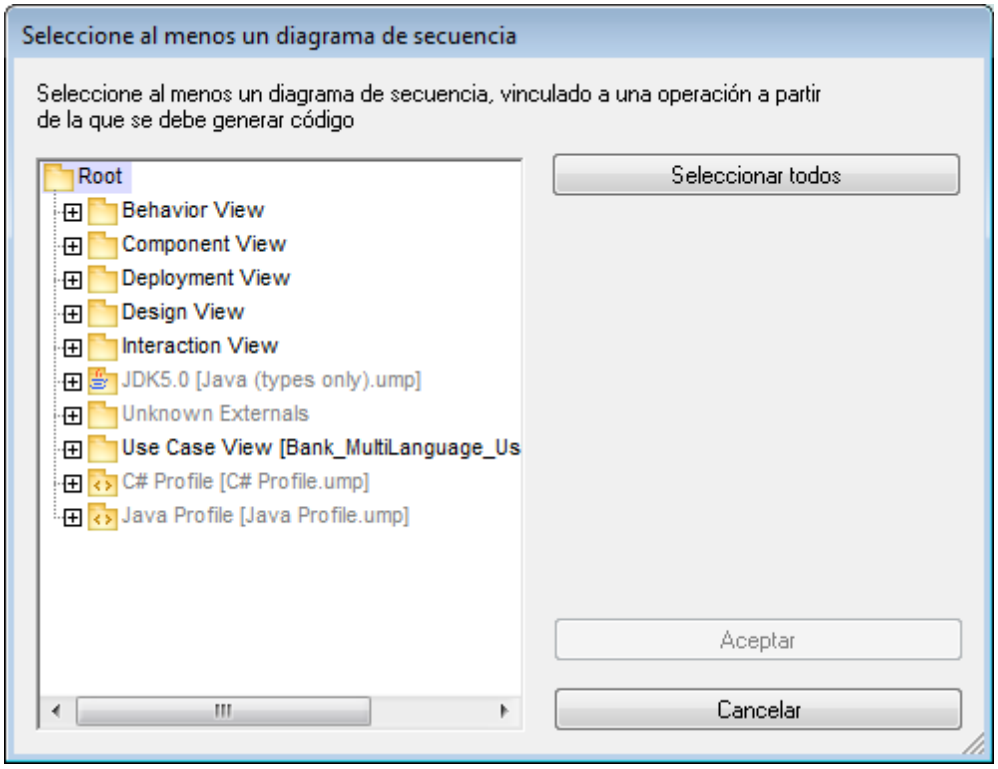
- Active la casilla `Usar para ingeniería directa` del panel Propiedades.

Cuando se crea código por ingeniería directa a partir de un diagrama de secuencia, siempre se pierde código antiguo porque lo sobrescribe el código nuevo.

Menú Proyecto:

1. Seleccione la opción de menú **Proyecto | Generar código a partir de diagramas de secuencia**.
   Aparece un diálogo donde debe seleccionar el diagrama de secuencia.

 © 2014 Altova GmbH



Con el botón **Seleccionar todos** se seleccionan todos los diagramas de secuencia del proyecto de UModel.

2. Haga clic en **Aceptar** para generar el código.
   La ventana Mensajes muestra el estado del proceso de generación de código.

Panel Estructural del modelo:

- Haga clic con el botón secundario en un diagrama de secuencia y elija **Generar código a partir del diagrama de secuencia** en el menú contextual.

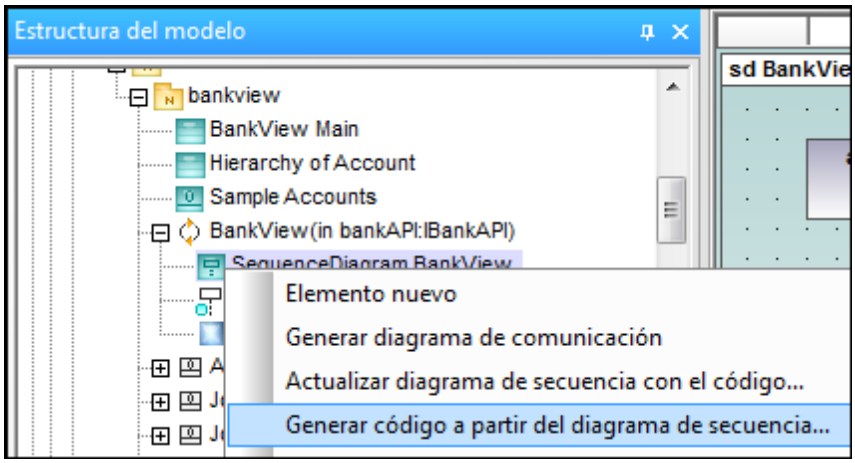
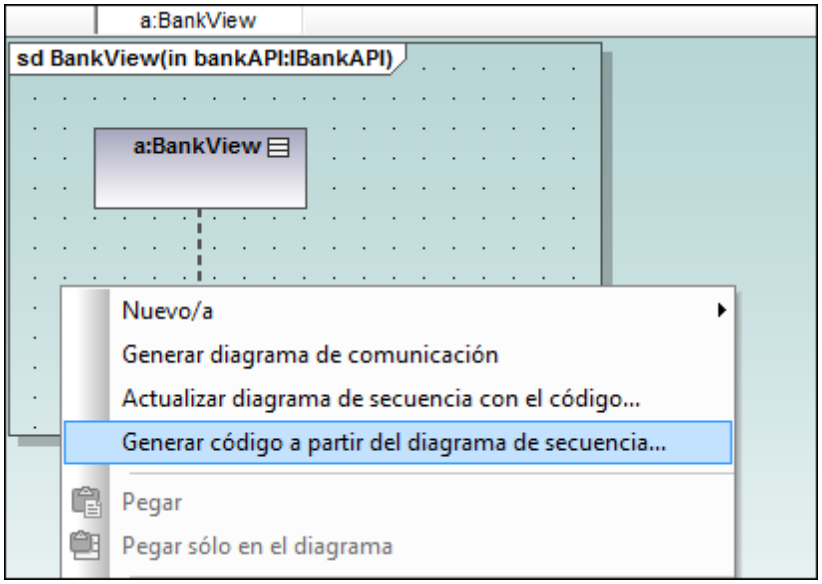


Diagrama de secuencia con código de una operación:

1. Haga clic con el botón secundario en el fondo del diagrama de secuencia que contiene el código de una operación.

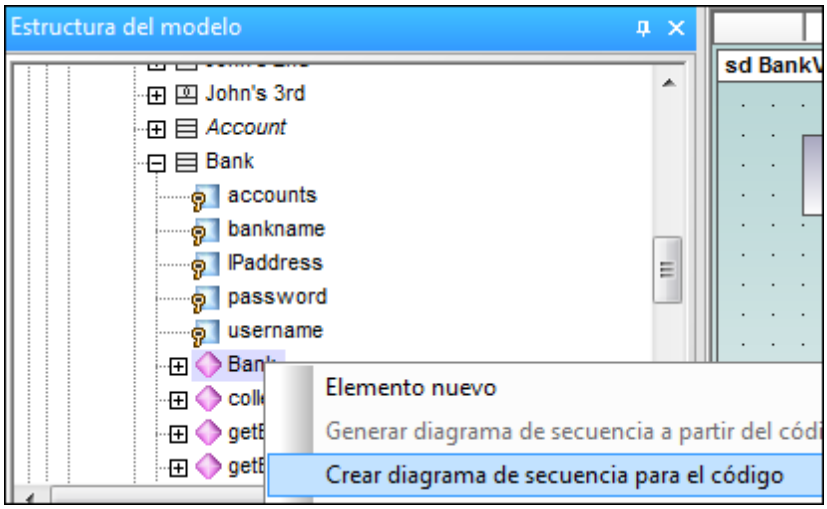
2. Elija la opción **Generar código a partir del diagrama de secuencia**.



Este comando inicia el proceso de ingeniería directa.

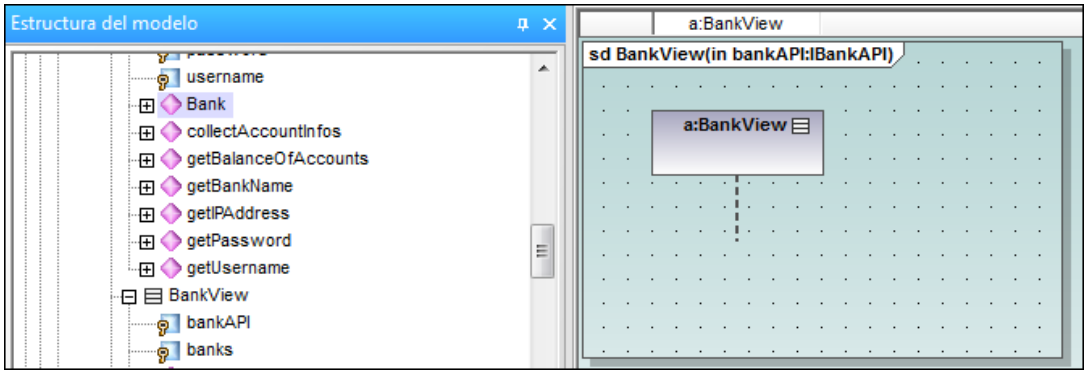
Para crear un diagrama de secuencia para el código (ingeniería):

1. En la Estructura del modelo haga clic con el botón secundario en una operación y elija la opción **Crear diagrama de secuencia para el código**.



UModel le pregunta si quiere usar el diagrama nuevo para la ingeniería directa.

 © 2014 Altova GmbH



El resultado es un diagrama de secuencia nuevo que contiene la línea de vida de esa clase.

Agregar código a diagramas de secuencia

En UModel puede generar código a partir de diagramas de secuencia nuevos y de diagramas de secuencia generados por ingeniería inversa, pero solo si el diagrama está vinculado a la operación principal.

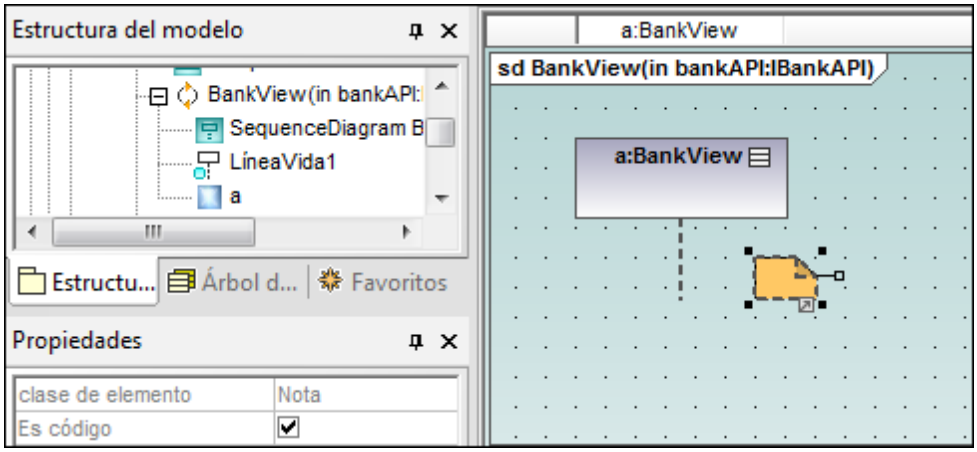
Cuando se aplica ingeniería inversa a código, los elementos estándar de los diagramas de secuencia (p. ej. los `FragmentosCombinados`) se asignan a los elementos del código (p. ej. instrucciones `if`, bucles, etc.).

Para las instrucciones de programación que no tengan elementos equivalentes en el diagrama de secuencia (p. ej. `i = i+1`), UModel utiliza las notas del código y añade código a los diagramas. Estas notas se deben vincular a la línea de vida.

Recuerde que UModel no revisa ni analiza estos fragmentos de código. Por eso es importante comprobar que los fragmentos de código son correctos y se podrán compilar.

Para agregar código a un diagrama de secuencia:

1. Haga clic en el icono **Nota** y después en el elemento de modelado donde desea insertar la nota (p. ej. `FragmentoCombinado`).
2 Escriba el fragmento de código dentro de la nota (p. ej. `return`).
3. Haga clic en el controlador `Enlace de nota` de la nota que acaba de insertar y arrastre el cursor hasta la línea de vida.
4. Marque la casilla `Es código` en el panel Propiedades para incluir este fragmento de código cuando UModel genere código.



Cuando seleccione una nota de un diagrama de secuencia que se **pueda** usar para generación de código, la propiedad `Es código` aparece en la ventana Propiedades. Esta propiedad permite alternar entre notas normales y corrientes y notas para generación de código.

Notas normales y corrientes:

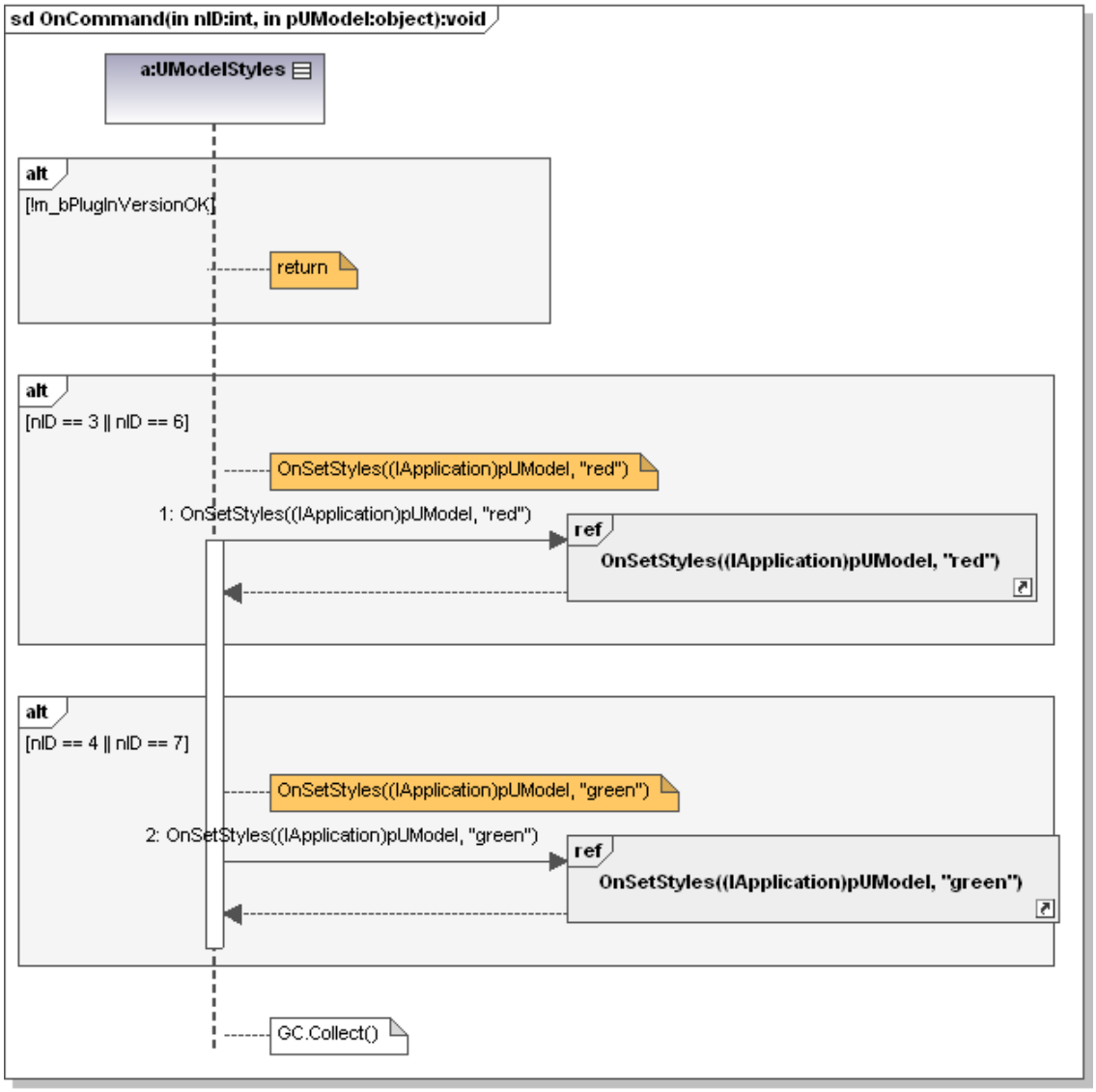
Notas para generación de código:

(la pestaña de la esquina superior derecha es más oscura)

Las actualizaciones de código tienen lugar automáticamente en cada proceso de ingeniería directa si está activa la casilla `Usar para ingeniería directa`. Si se realizaron cambios en el diagrama de secuencia, el código de la operación se sobrescribe siempre.

El diagrama de secuencia que aparece más abajo se generó haciendo clic con el botón secundario en la operación `OnCommand` y seleccionando la opción **Generar diagrama de secuencia a partir del código**. El código C# de este ejemplo está disponible en la carpeta `c:\Documents and Settings\<usuario>\Mis Documentos\Altova\UModel2012\UModelExamples\IDEPlugIn\Styles`. Utilice la opción de menú **Proyecto | Importar proyecto de código fuente** para importar el proyecto.

 © 2014 Altova GmbH



El código que aparece a continuación se generó a partir del diagrama de secuencia.

```
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();
}
```


© 2014 Altova GmbH

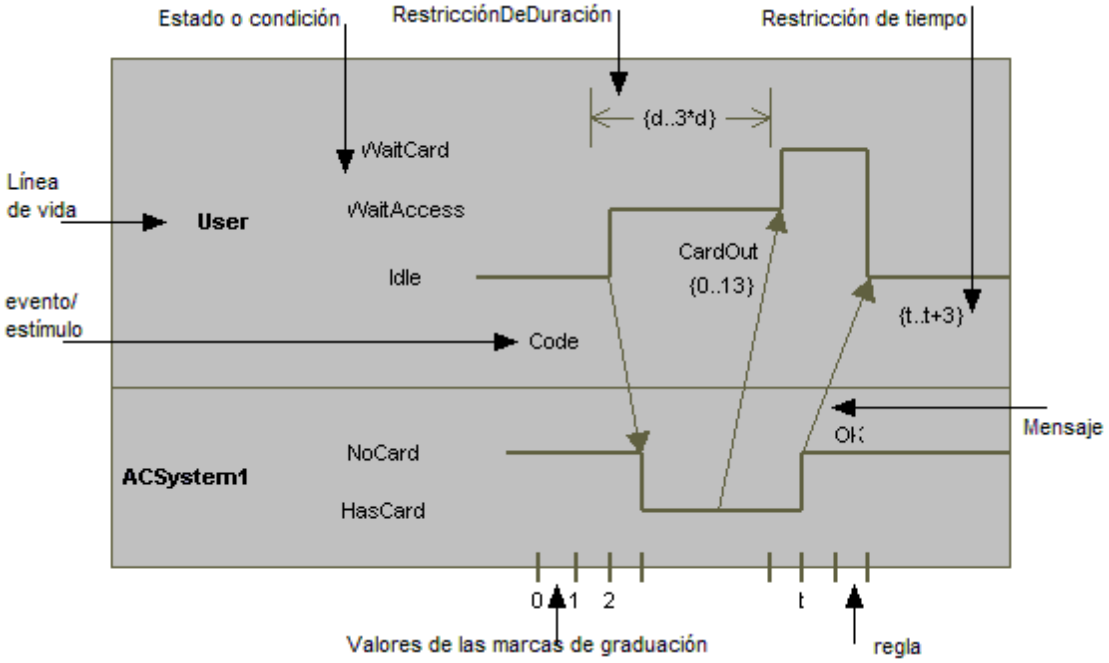
10.1.8 Diagrama de ciclo de vida

Sitio web de Altova: Diagramas de ciclo de vida UML

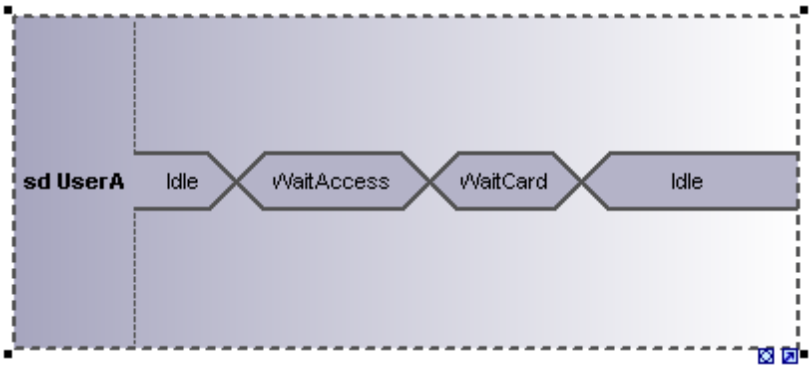
Los diagramas de ciclo de vida modelan los cambios de estado o la condición de los objetos que interactúan entre sí a lo largo de un período de tiempo. Los estados o condiciones se representan como escalas de tiempo que responden a eventos de mensaje y las líneas de vida representan instancias de clasificador y roles clasificador.

Los diagramas de ciclo de vida son un tipo especial de diagrama de secuencia. La diferencia es que los ejes están invertidos, es decir, el tiempo aumenta de izquierda a derecha, y las líneas de vida aparecen por separado en compartimentos apilados verticalmente.

Además, este tipo de diagramas suelen utilizarse para el diseño de software integrado o de sistemas en tiempo real.



Hay dos tipos de diagramas de ciclo de vida: los que contienen la escala de tiempo del estado/de la condición (*imagen anterior*) y los que muestran el ciclo de vida general (*imagen siguiente*).



Insertar elementos

Usar iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de ciclo de vida.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama de ciclo de vida:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de ciclo de vida.

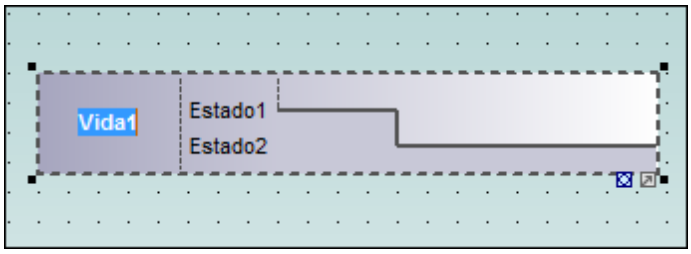
Línea de vida

o **Línea de vida**

La línea de vida es un participante de la interacción y tiene dos representaciones distintas: un **Estado/Condición** o un **Valor general**. Pulse **Ctrl+Entrar** para crear una línea nueva en el nombre de la línea de vida.

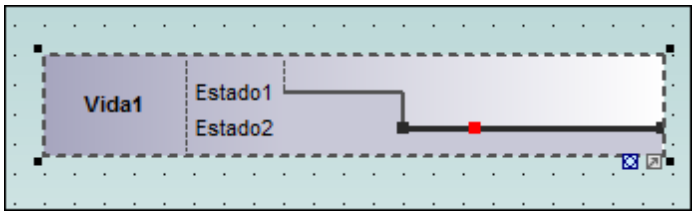
Para insertar un Estado/Condición (InvarianteDeEstado) y definir cambios de estado:

1. Haga clic en el icono **Línea de vida (Estado o Condición)** de la barra de herramientas y después haga clic en el área de trabajo del diagrama.



2. Escriba el nombre de la línea de vida o utilice el nombre predeterminado `Líneadevida1`.
3. Haga clic en una sección de la línea de tiempo para seleccionarla.
4. Haga clic en la posición de la línea de tiempo donde quiere que se produzca el cambio de estado.

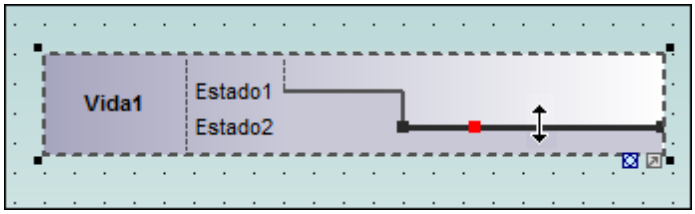
   En este momento aparece una flecha con dos puntas.



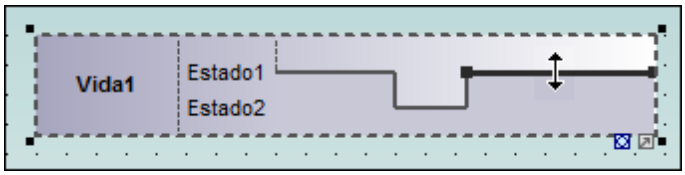
   Y además aparece un recuadro rojo en la posición donde hizo clic, que divide la línea por ese punto.

 © 2014 Altova GmbH

5. Ponga el cursor en la parte derecha de la línea y arrastre la línea hacia arriba.



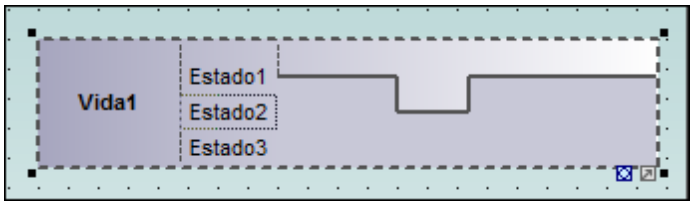
No olvide que solo puede mover líneas entre estados de la línea de vida actual.



Puede definir un número ilimitado de cambios de estado en una línea de vida. El recuadro rojo de la línea desaparece al hacer clic en otra parte del diagrama.

Para añadir un estado nuevo a la línea de vida:

1. Haga clic con el botón secundario en la línea de vida y seleccione **Nuevo/a | Estado o Condición (InvarianteDeEstado)**.
   El estado nuevo `Estado3` se añade a la línea de vida.



Para cambiar la posición de un estado (dentro de una línea de vida):

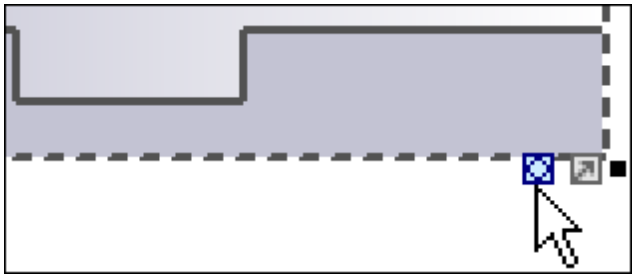
1. Haga clic en la etiqueta del estado.
2. Arrástrela hasta la posición nueva.

Para eliminar un estado de una línea de vida:

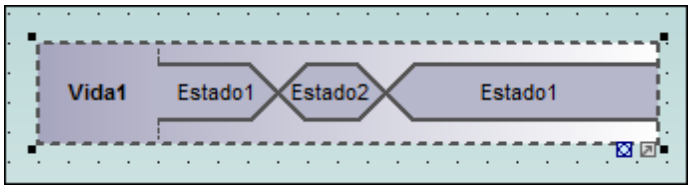
1. Haga clic en la etiqueta del estado y pulse la tecla **Supr**. También puede hacer clic con el botón secundario en el estado y seleccionar el comando **Eliminar**.

Para cambiar de tipo de diagrama de ciclo de vida:

1. Haga clic en el icono **Alternar estilo de notación** que aparece en la esquina inferior derecha de la línea de vida.



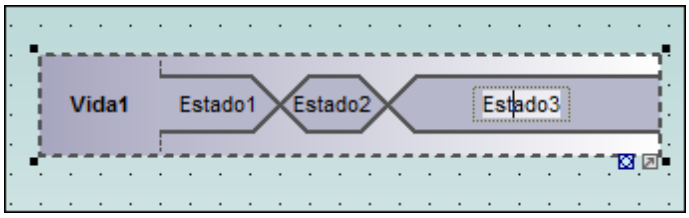
Ahora la línea de vida se presenta con su `Valor general`. Cada punto donde se cruzan las líneas es un cambio de estado/valor.



Recuerde que al hacer clic en el icono **Línea de vida (Valor general)** se inserta una línea de vida como la de la imagen anterior. Puede cambiar a la otra presentación cuando quiera.

Para añadir un estado nuevo a la línea de vida Valor general:

1. Haga clic con el botón secundario en la línea de vida y seleccione **Nuevo/a | Estado o condición (InvarianteDeEstado)**.
2. Edite el nombre del estado nuevo y pulse **Entrar** para confirmar.

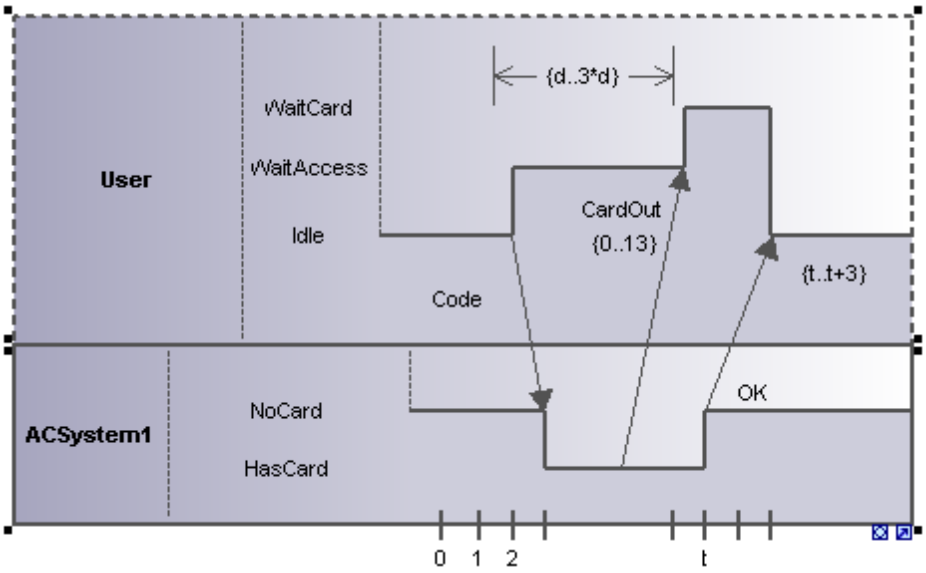


Se añade un estado nuevo en la línea de vida.

Agrupar las líneas de vida

Si apila las líneas de vida, UModel las reorganiza automáticamente de forma correcta y conserva las marcas de graduación disponibles hasta ese momento. También puede crear mensajes entre las líneas de vida. Para ello arrastre el objeto de mensaje correspondiente hasta la posición deseada.

 © 2014 Altova GmbH



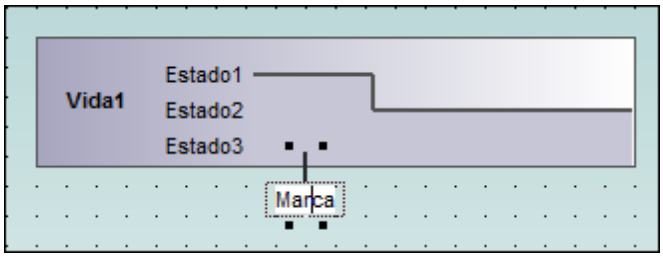
Marca de graduación

MarcaDeGraduación

Este icono permite insertar las marcas de graduación de una escala de tiempo en la línea de vida.

Para insertar una marca de graduación:

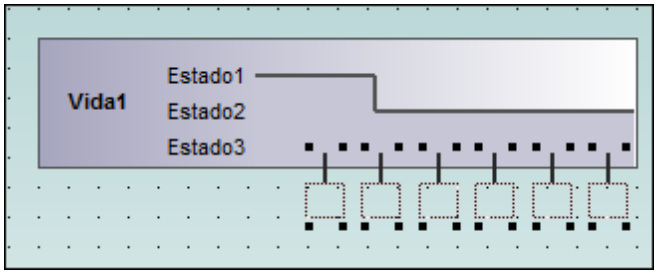
1. Haga clic en el icono de la marca de graduación y después en la línea de vida para insertarla.



2. Para insertar varias marcas, pulse la tecla **Ctrl** mientras hace clic en la línea de vida tantas veces como sea necesario.
3. Escriba el nombre de la marca.
   Si quiere cambiar la posición de una marca de graduación, simplemente arrástrela hasta la posición nueva.

Para espaciar las marcas de graduación uniformemente:

1. Seleccione todas las marcas de graduación.
2. Haga clic en el icono **Espaciar uniformemente en horizontal** de la barra de herramientas.



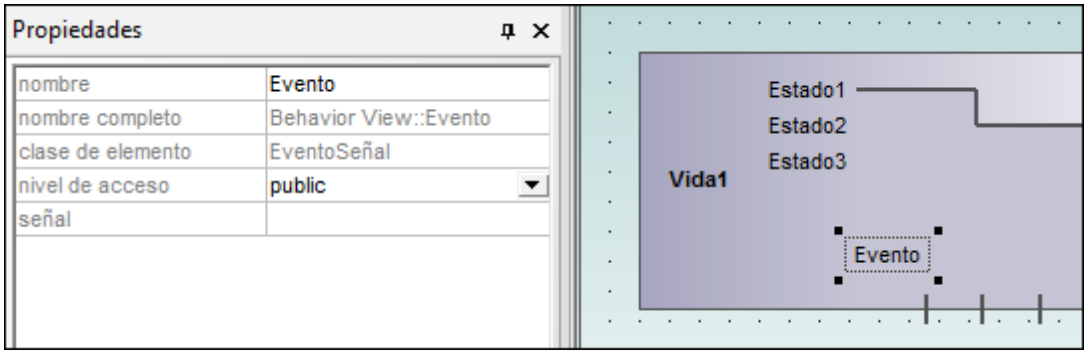
Evento/estímulo

Evento/ Estímulo

Este icono sirve para ilustrar el cambio de estado de un objeto causado por el correspondiente evento o estímulo. Los eventos recibidos se anotan para mostrar qué evento provoca el cambio en la condición o en el estado.

Para insertar un evento o estímulo:

1. Haga clic en el icono **Evento o estímulo** y después haga clic en la posición de la escala de tiempo donde tiene lugar el cambio de estado.



2. Escriba el nombre del evento. Observe que las propiedades del evento aparece en el panel Propiedades.

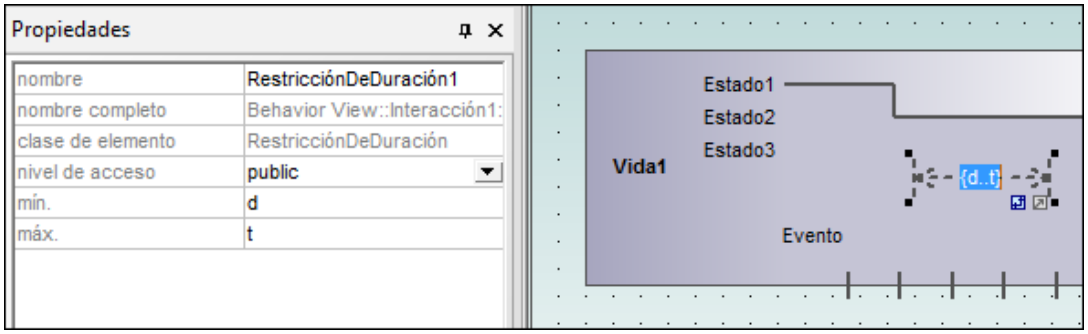
Restricción de duración

RestricciónDeDuración

Una duración define una `EspecificaciónDeValor` que denota el período de tiempo comprendido entre el punto de inicio y el punto final. Por lo general una duración es una expresión que representa el tiempo que puede pasar durante este período.

Para insertar una RestricciónDeDuración:

1. Haga clic en el icono **RestricciónDeDuración** y después haga clic en la posición de la línea de vida donde debe aparecer la restricción.

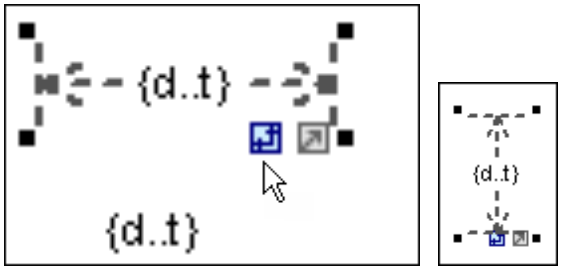


El valor mínimo y máximo predeterminado ("`d..t`") aparece automáticamente. Para editar estos valores haga doble clic en la restricción o edite los valores en el panel Propiedades.

2. Si quiere, use los controladores para cambiar el objeto de tamaño.

Para cambiar la orientación de la RestricciónDeDuración:

1. Haga clic en el icono rotación para poner la restricción en vertical.



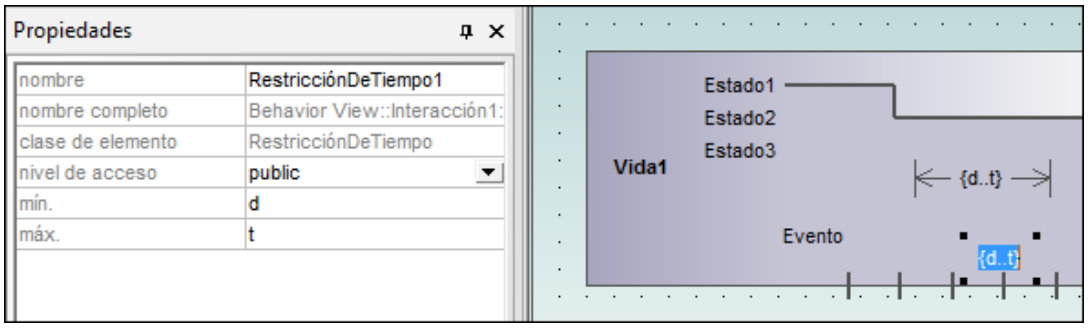
Restricción de tiempo

RestricciónDeTiempo

Una restricción de tiempo suele representarse como una asociación gráfica entre un `IntervaloDeTiempo` y la construcción que limita. Lo normal es que sea una asociación gráfica entre un evento y un intervalo de tiempo.

Para insertar una restricción de tiempo:

1. Haga clic en el icono **RestricciónDeTiempo** en la barra de herramientas y después haga clic en la posición de la línea de vida donde debe aparecer la restricción.



El valor mínimo y máximo predeterminado ("`d..t`") aparece automáticamente. Para editar

estos valores haga doble clic en la restricción o edite los valores en el panel Propiedades.

Mensaje

Mensaje (Llamada) **Mensaje (Respuesta)** **Mensaje asincrónico (Llamada)**

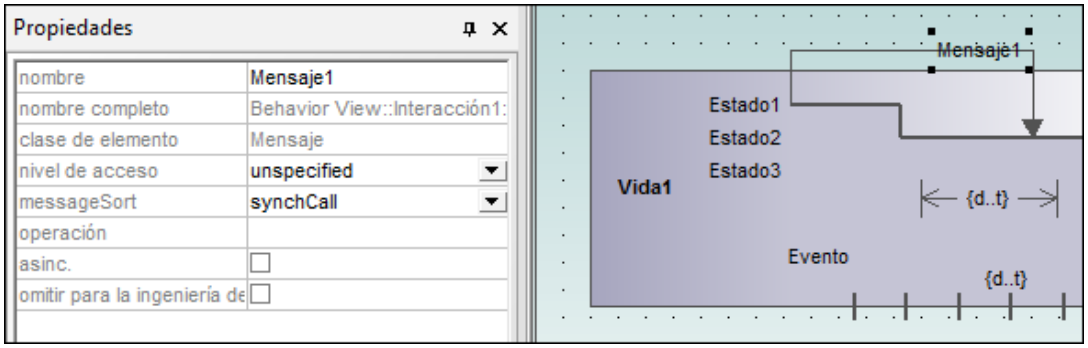
Un mensaje es un elemento de modelado que define un tipo concreto de comunicación dentro de una interacción. Una comunicación puede lanzar una señal, invocar una operación, crear o destruir una instancia, etc. El mensaje especifica el tipo de comunicación definida por la `EspecificaciónDeEjecución` emisora, así como el remitente y el destinatario.

Los mensajes se transmiten entre la escala de tiempo remitente y la escala de tiempo destinataria y se representan en forma de flechas con etiqueta.

Para insertar un mensaje:

1. Haga clic en el icono del mensaje que desea insertar en la barra de herramientas.
2. Haga clic en el objeto remitente.
3. Arrastre la línea del mensaje y suéltela encima del objeto destinatario.

   La línea de vida se resalta cuando el mensaje se puede soltar.



- La dirección en la que se arrastra la flecha define la dirección del mensaje. Los mensajes de respuesta pueden apuntar en ambas direcciones.
- Mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo para insertar varios mensajes.

Para eliminar un mensaje:

1. Haga clic en el mensaje que desea eliminar.
2. Pulse la tecla **Supr** para eliminarlo del modelo. También puede hacer clic con el botón secundario en el mensaje y elegir el comando **Eliminar solo en el diagrama**.

 © 2014 Altova GmbH

10.2 Diagramas de estructura

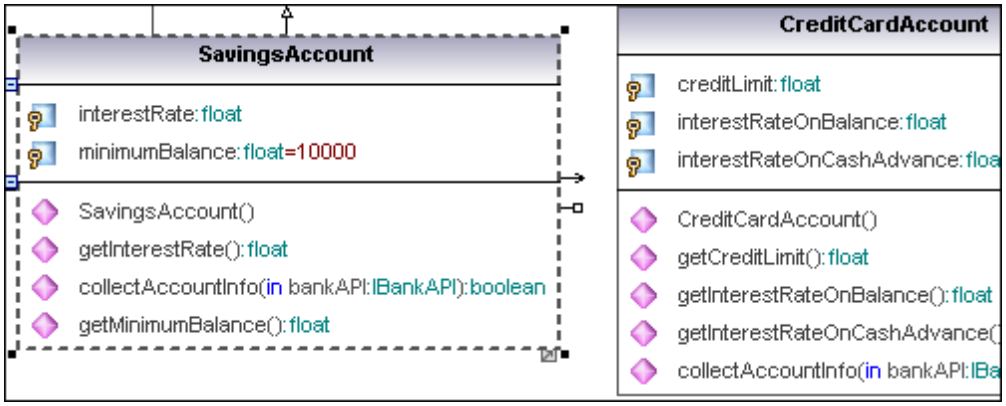
Los diagramas de estructura muestran qué elementos estructurales componen un sistema o una función. Pueden representar tanto las relaciones estáticas (p. ej. diagramas de clases) como las dinámicas (p. ej. diagramas de objetos).

Diagramas de estructura

Diagramas de clases
Diagramas de componentes
Diagramas de estructura de un compuesto
Diagramas de implementación
Diagramas de objetos
Diagramas de paquetes
Diagramas de perfil

10.2.1 Diagrama de clases

Para más información sobre cómo agregar clases a un diagrama, consulte el apartado Diagramas de clases del tutorial.



Expandir/ocultar los compartimientos de las clases en el área de trabajo:
Hay varias maneras de expandir los compartimientos de los diagramas de clases.

- Haga clic en los botones **+** o **-** de la clase activa para expandir/contraer el compartimiento correspondiente.
- Use el recuadro de selección (arrastrando el puntero por el diagrama) para marcar varias clases y después haga clic en el botón expandir/ocultar. También puede usar **Ctrl+clic** para seleccionar varias clases.
- Pulse **Ctrl+A** para seleccionar **todas las clases** y después haga clic en el botón expandir/ocultar en una de las clases para expandir/contraer los compartimientos correspondientes.

Expandir/ocultar los compartimientos de las clases en la Estructura del modelo:
En la Estructura del modelo, las clases son subelementos de los paquetes y la acción expandir/ocultar se puede ejecutar en los paquetes o en las clases.

Haga clic en el paquete / en la clase que desea **expandir** y:
- Pulse la tecla ***** para expandir el paquete/la clase actual y todos los subelementos.
- Pulse la tecla **+** para abrir el paquete/la clase actual.

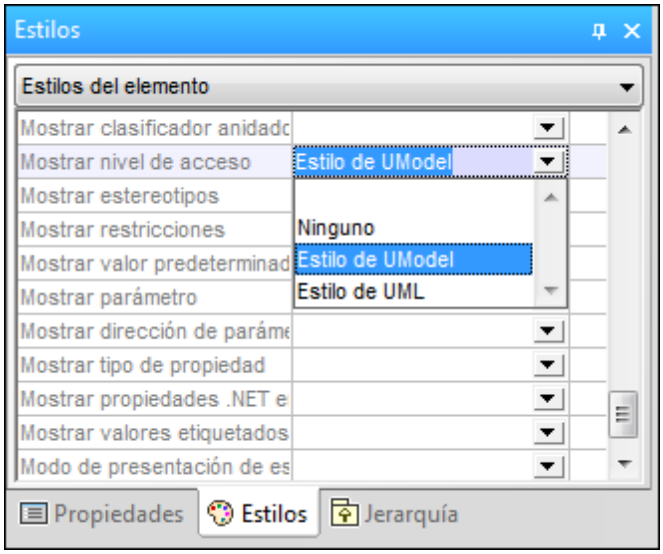
Para **contraer** los paquetes/las clases, pulse la tecla del teclado **-**.
Recuerde que puede usar las teclas del teclado estándar o del teclado numérico para hacer esto.

Cambiar el icono de nivel de acceso
Haga clic en el icono de nivel de acceso situado a la izquierda de una operación o propiedad para abrir una lista desplegable en la que puede elegir el nivel de acceso del elemento.

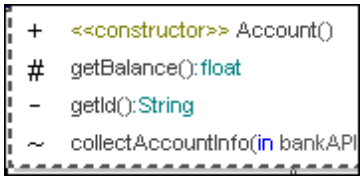
En UModel también puede elegir qué tipo de símbolo se utiliza para identificar los niveles de acceso nivel de acceso:

- Haga clic en una clase del diagrama y abra la pestaña *Estilos*. Desplácese hasta la el estilo `Mostrar nivel de acceso`.

 © 2014 Altova GmbH



Aquí puede elegir entre usar el estilo de UModel, el estilo de UML (*imagen siguiente*) o no utilizar ninguno.

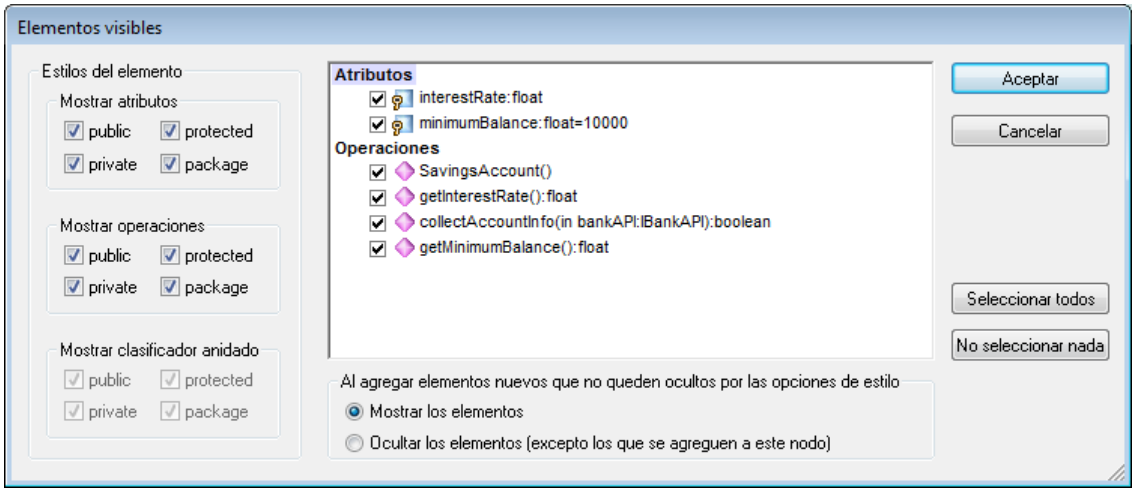


Mostrar/ocultar el contenido de los nodos (atributos de clase, operaciones, slots)
En UModel también puede elegir qué atributos y operaciones de una clase aparecen en el diagrama y elegir qué nuevos atributos y operaciones se muestran cuando se añaden.

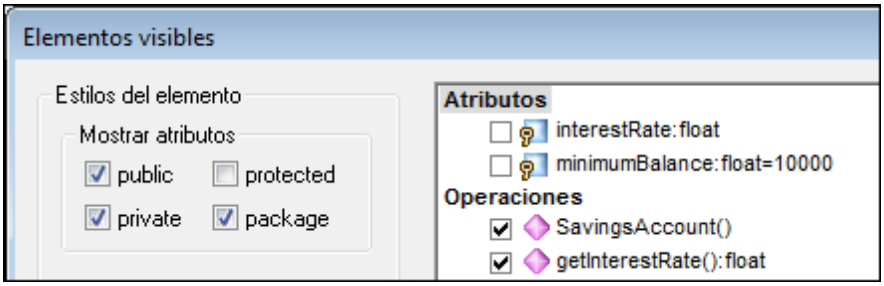
No olvide que los slots de objetos (es decir, EspecificacionesDeInstancia) se pueden mostrar/ ocultar de la misma manera.

Haga clic con el botón secundario en una clase (p. ej. `SavingsAccount`) y elija el comando **Mostrar u ocultar el contenido del nodo** del menú contextual.

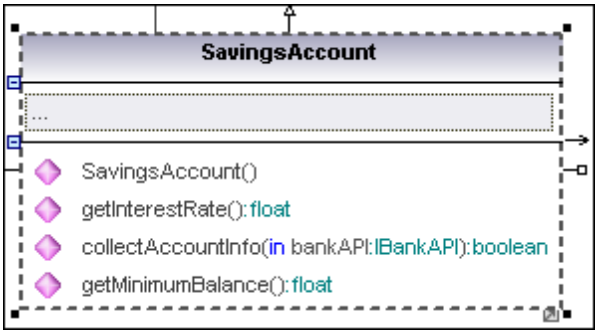
Esto abre el cuadro de diálogo "Elementos visibles".



Si desactiva la casilla *protected* del grupo *Mostrar atributos*, los atributos que tengan el nivel `protected` se desactivan en la vista previa del cuadro de diálogo.



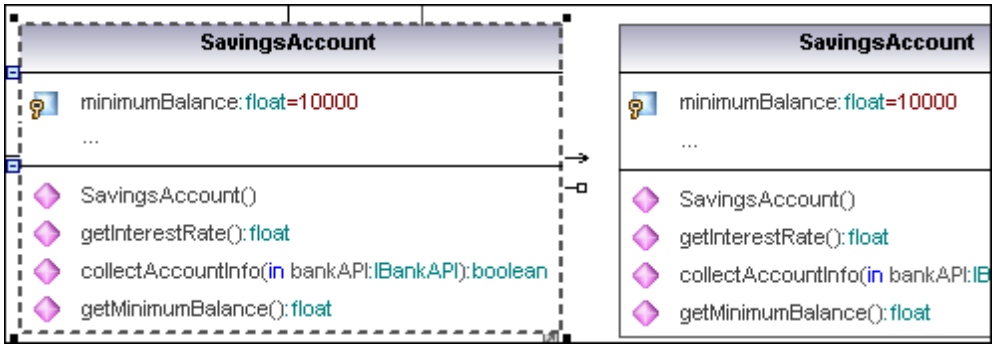
Haga clic en **Aceptar** para cerrar el cuadro de diálogo. Los atributos de la clase que tienen el nivel `protected` se reemplazan con tres puntos (...). Haga doble clic en los tres puntos para abrir otra vez el cuadro de diálogo "Elementos visibles".



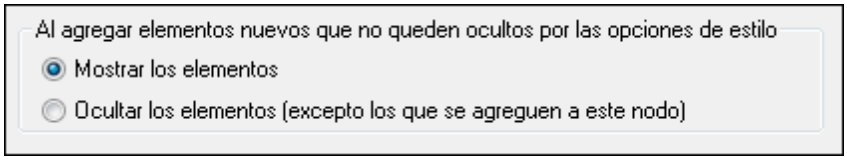
No olvide que en la vista previa del cuadro de diálogo puede desactivar los atributos uno a uno.

Mostrar/ocultar atributos y operaciones (Estilos del elemento)

En UModel puede insertar varias instancias de la misma clase en un el mismo diagrama o incluso en diagramas distintos. Cada una de las vistas de esta clase puede tener niveles de acceso diferentes. Por ejemplo, la imagen siguiente muestra dos vistas de la misma clase: **`SavingsAccount`**.

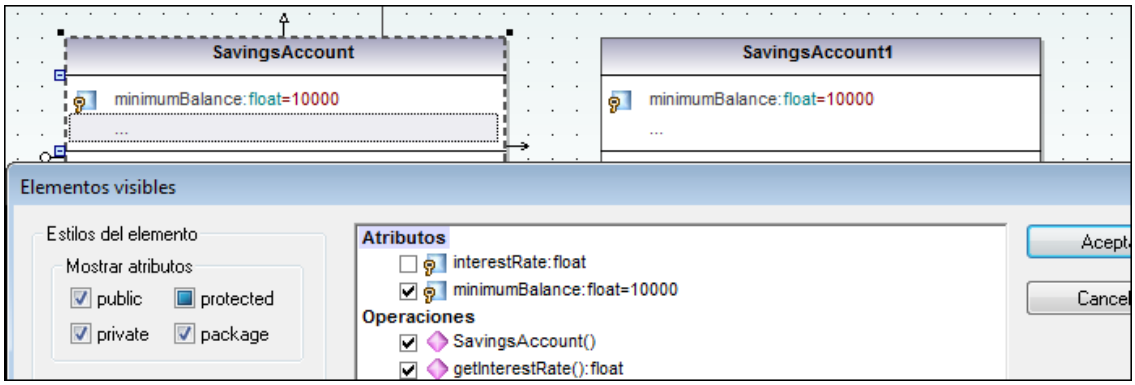


En el cuadro de diálogo "Elementos visibles" las opciones del grupo *Al agregar elementos nuevos que no queden ocultos por las opciones de estilo* sirven para definir qué será visible cuando se añadan elementos nuevos a la clase. Los elementos se pueden añadir a mano en el diagrama o en la Estructura del modelo o automáticamente durante el proceso de ingeniería de código.

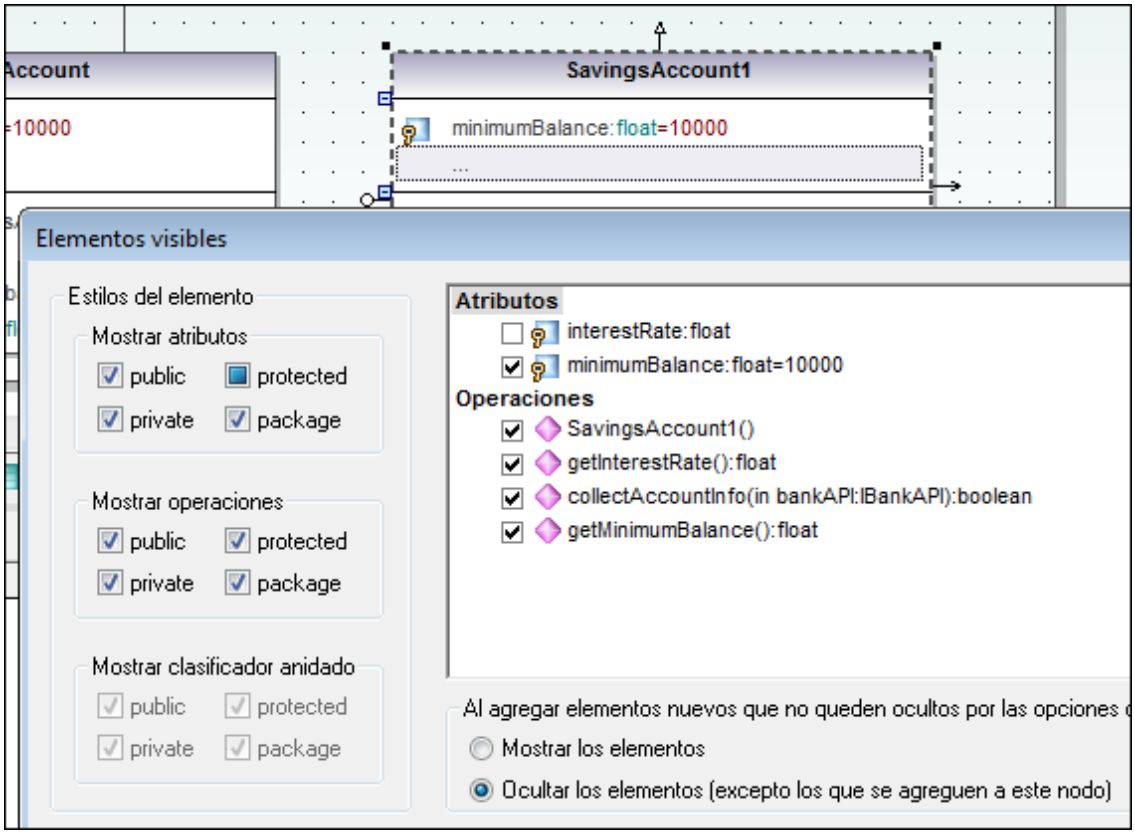


Mostrar los elementos: elija esta opción si quiere que se muestren todos los elementos que se añadan a cualquier vista de la clase.

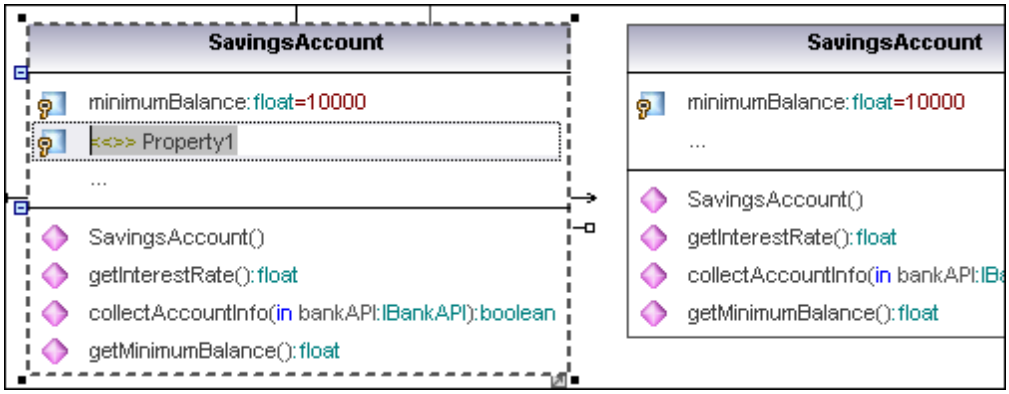
P. ej. el atributo `interestRate:float` se ocultó en ambas vistas de **`SavingsAccount`** y se dejó visible el atributo `minimumBalance`. El botón de opción *Mostrar los elementos* está activo para la vista izquierda de la clase. Al hacer doble clic en los tres puntos (...) del compartimiento del atributo de la vista **izquierda** de la clase, se abre el cuadro de diálogo "Elementos visibles", donde podemos ver que el botón de opción *Mostrar los elementos* está activo.



Al hacer doble clic en los tres puntos (...) del compartimiento del atributo de la vista **derecha** de la clase, se abre el cuadro de diálogo "Elementos visibles", donde podemos ver que está activo el botón de opción *Ocultar los elementos (excepto los que se agreguen a este nodo)*.



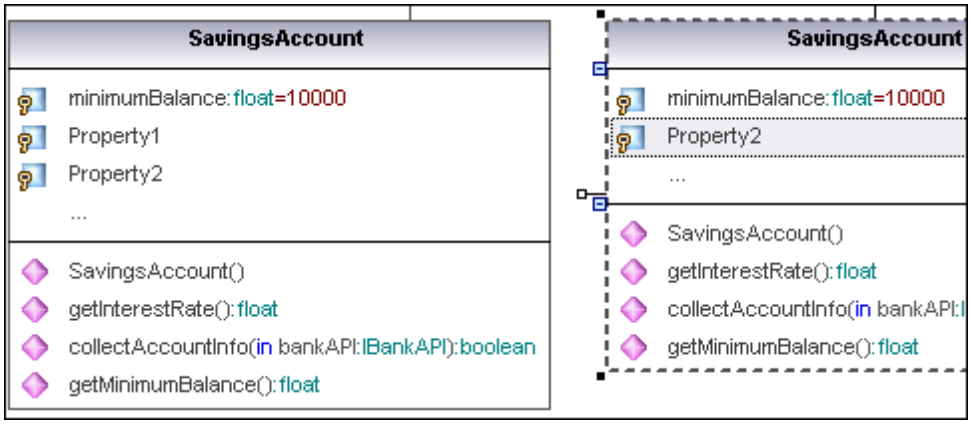
Si hacemos clic en la vista **izquierda** de la clase y pulsamos **F7** (o si hacemos clic en la clase en la Estructura del modelo y pulsamos **F7**), en la clase se añade un atributo nuevo (`Propiedad1`).



El nuevo elemento solamente está visible en la vista **izquierda** de la clase (porque en esta vista está activa la opción *Mostrar los elementos*). En la vista **derecha** de la clase el elemento está oculto (porque en esa vista está activa la opción *Ocultar los elementos*).

Si hacemos clic en la vista derecha de la clase y pulsamos F7, en la clase se añade un atributo nuevo (`Propiedad2`). Este atributo nuevo está visible en la vista derecha porque la opción oculta los elementos *excepto los que se agreguen a este nodo*. Aquí *nodo* significa esta clase o elemento de modelado.

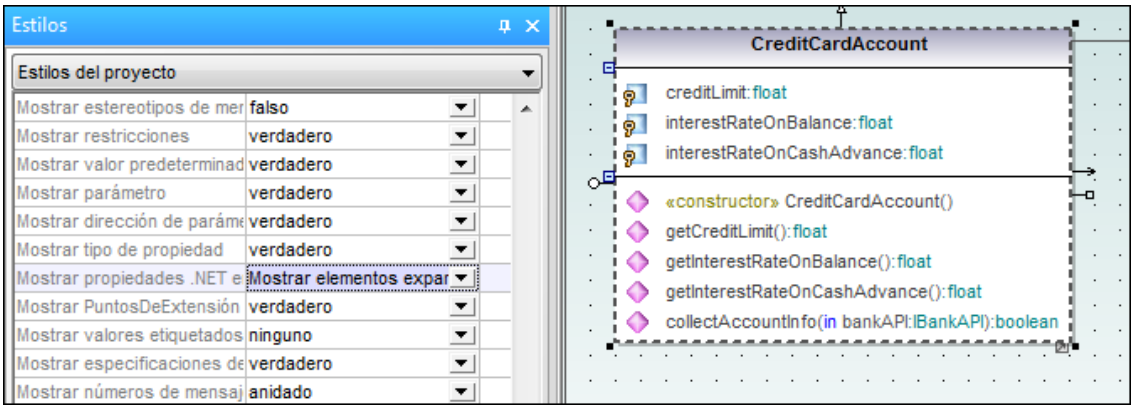
 © 2014 Altova GmbH



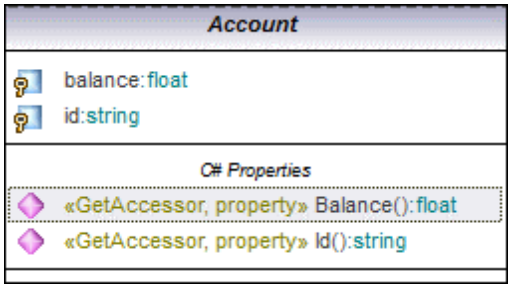
Ahora el atributo Propiedad2 está visible en la vista izquierda de la clase (porque en esta vista está activa la opción *Mostrar los elementos*).

Nota: los valores etiquetados de los elementos ocultos también se ocultan si se elige la opción *Ocultar los elementos*.

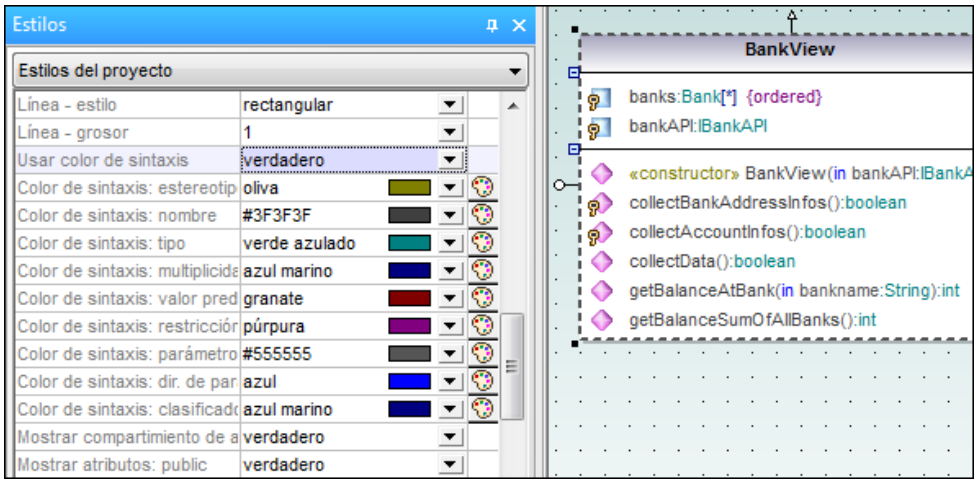
Mostrar/ocultar compartimientos de propiedades VS .NET:
UModel puede mostrar las propiedades .NET en un compartimiento separado. Para ello, habilite el estilo `Mostrar propiedades .NET en un compartimiento propio` del panel Estilos.



Mostrar las propiedades VS .NET como asociaciones
UModel también puede mostrar las propiedades .NET como asociaciones.
Haga clic con el botón secundario en una propiedad C# y elija **Mostrar | Todas las propiedades .NET como asociaciones** en el menú contextual.

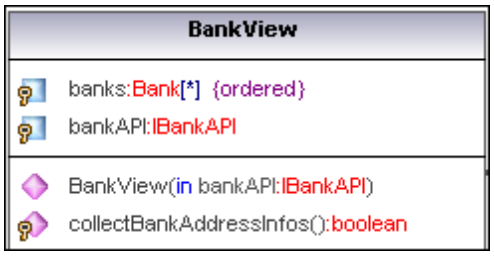


Cambiar el color de sintaxis de las operaciones/propiedades
UModel habilita automáticamente la función de color de sintaxis, pero esta función se puede personalizar. A continuación puede ver la configuración predeterminada.



Para cambiar las opciones predeterminadas de color de sintaxis:

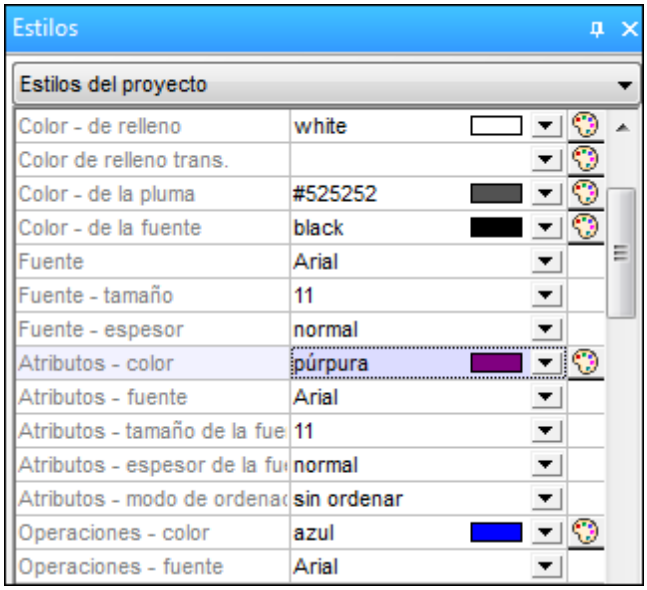
1. Abra la pestaña *Estilos* y desplácese hasta los estilos que empiezan por `Color de sintaxis`.
2. Cambie el valor de uno de estos estilos. Por ejemplo: cambie el valor de `Color de sintaxis: tipo` a `red`.



Para deshabilitar el color de sintaxis:

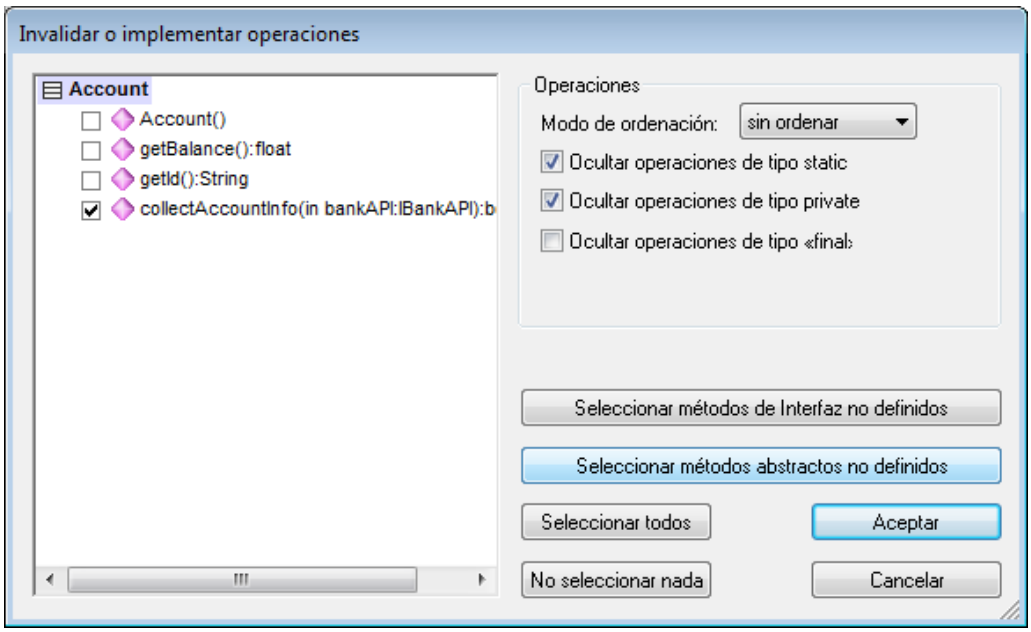
1. Abra la pestaña Estilos y asigne el valor `false` al estilo `Usar color de sintaxis`.
2. Y ahora use los estilos `Atributos - color` o los estilos `Operaciones - color` para personalizar estos elementos en las clases.

 © 2014 Altova GmbH



Invalidar operaciones de la clase base e implementar operaciones de interfaz
UModel ofrece la posibilidad de invalidar las operaciones de la clase base o implementar operaciones de interfaz de una clase. Esto se puede hacer desde la Estructura del modelo, desde Favoritos o desde los diagramas de clases.

1. Haga clic con el botón secundario en una de las clases derivadas del diagrama (p. ej. `CheckingAccount`) y elija **Invalidar o implementar operaciones** en el menú contextual. Esto abre el cuadro de diálogo "Invalidar o implementar operaciones" (*imagen siguiente*).



2. Seleccione las operaciones que desea invalidar y haga clic en **Aceptar** para confirmar. Con los botones **Seleccionar métodos...** puede seleccionar esos tipos de métodos en la vista previa de la izquierda.

Nota: cuando se abre este cuadro de diálogo, se marcan (se activan) las operaciones de las clases base y de las interfaces implementadas que tiene la misma firma que las operaciones disponibles.

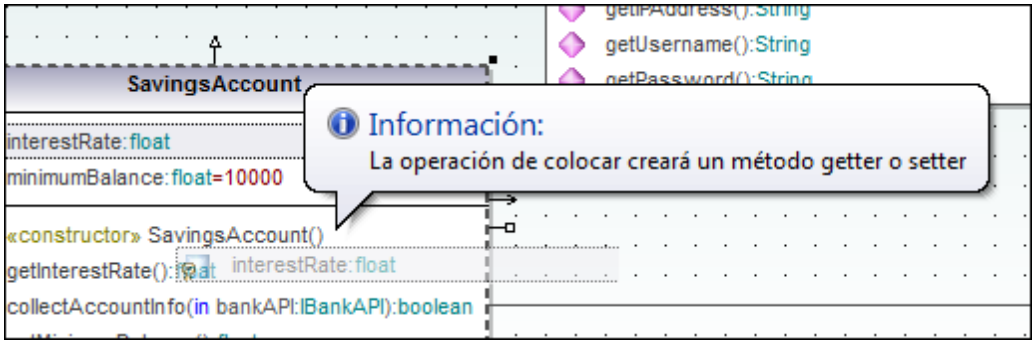
Crear métodos getter / setter

Durante el proceso de modelado a veces es necesario crear métodos getter/setter para los atributos existentes. UModel ofrece dos métodos para crear métodos getter/setter:

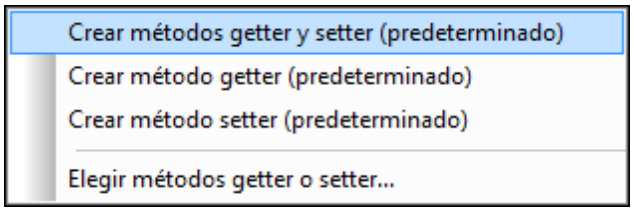
- Arrastrando y colocando un atributo en el compartimiento de una operación.
- Usando el menú contextual para abrir un cuadro de diálogo donde puede gestionar los métodos getter/setter.

Para crear métodos getter/setter mediante operaciones arrastrar y colocar:

1. Arrastre un atributo desde el compartimiento de atributos hasta el compartimiento de operaciones.

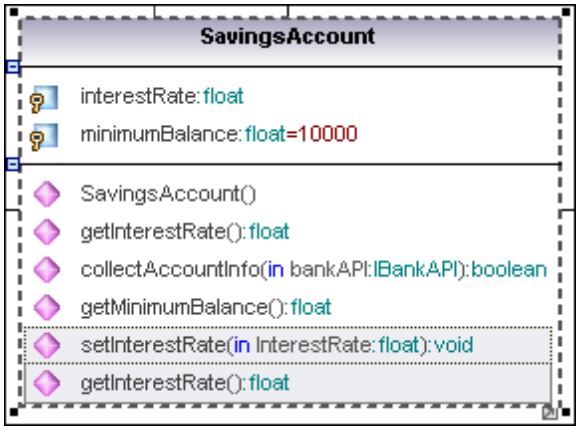


Aparece un menú contextual donde puede elegir el tipo de método getter/setter que se debe crear.



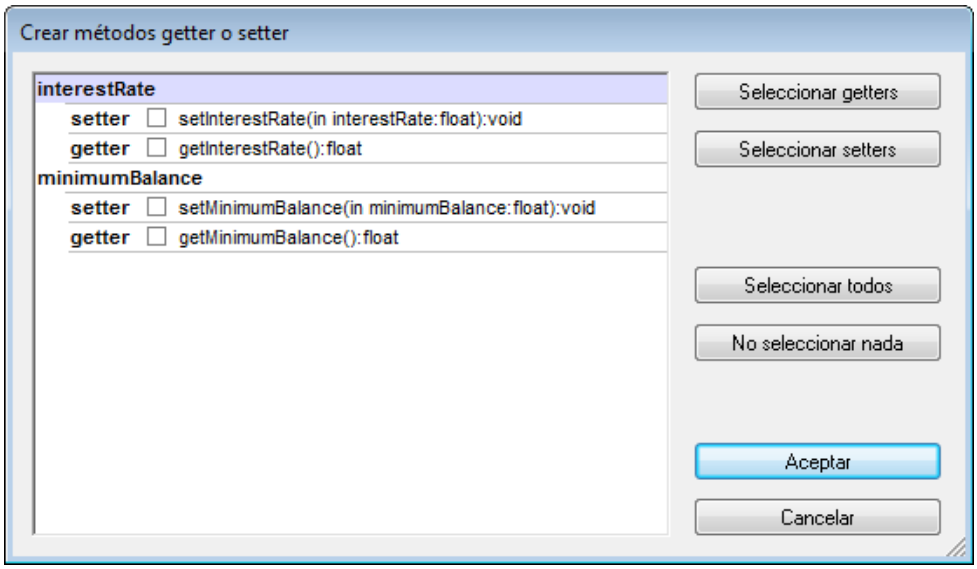
Seleccione el primer comando del menú para crear un método getter y setter para `interestRate:float`.

 © 2014 Altova GmbH



Para crear métodos getter/setter con el menú contextual:

1. Haga clic con el botón secundario en el título de una clase (p. ej. `SavingsAccount`) y elija la opción **Crear operaciones de método getter o setter...** del menú contextual. Esto abre el cuadro de diálogo "Crear métodos getter o setter", que enumera los atributos disponibles en la clase activa.

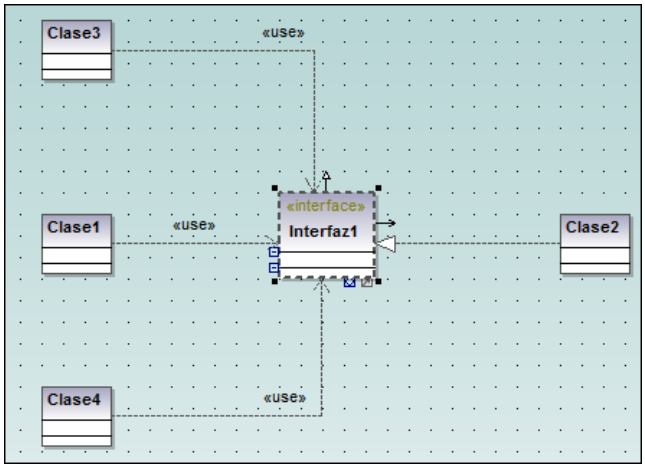


2. Use los botones o marque las casillas para seleccionar los métodos que desea crear.

   Nota: también puede hacer clic con el botón secundario en un solo atributo y usar el mismo método para crear una operación para el atributo.

Notación en forma esférica

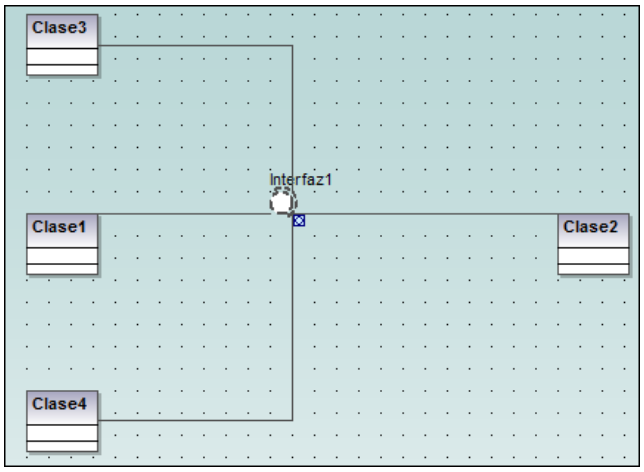
UModel es compatible con la notación en forma esférica de UML. Las clases que necesitan una interfaz muestran un círculo y el nombre de la interfaz, mientras que las clases que implementan la interfaz muestran una esfera.



En la imagen anterior, la `Clase2` realiza la `Interfaz1`, que es utilizada por las clases `clase1`, `clase3` y `clase4`. Para crear la relación de utilización entre las clases e interfaz se utilizó el icono **Utilización** de la barra de herramientas Diagrama de clases.

Para cambiar entre la vista estándar y la vista esférica:

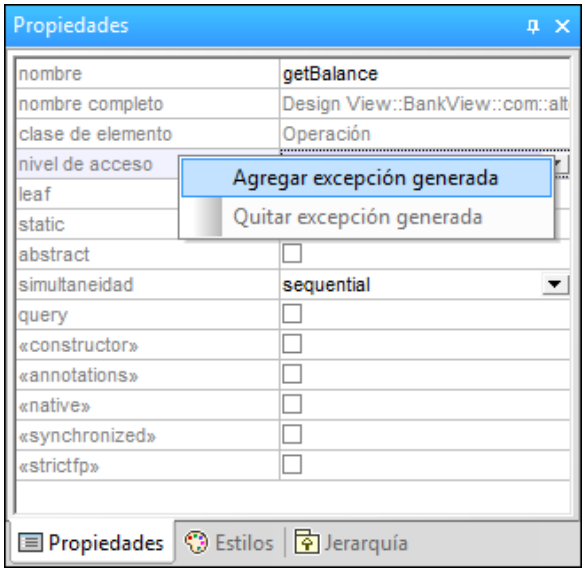
- Haga clic en el icono **Alternar estilo de notación de interfaz** situado en la parte inferior del elemento interfaz.



Para agregar excepciones generadas a los métodos de una clase

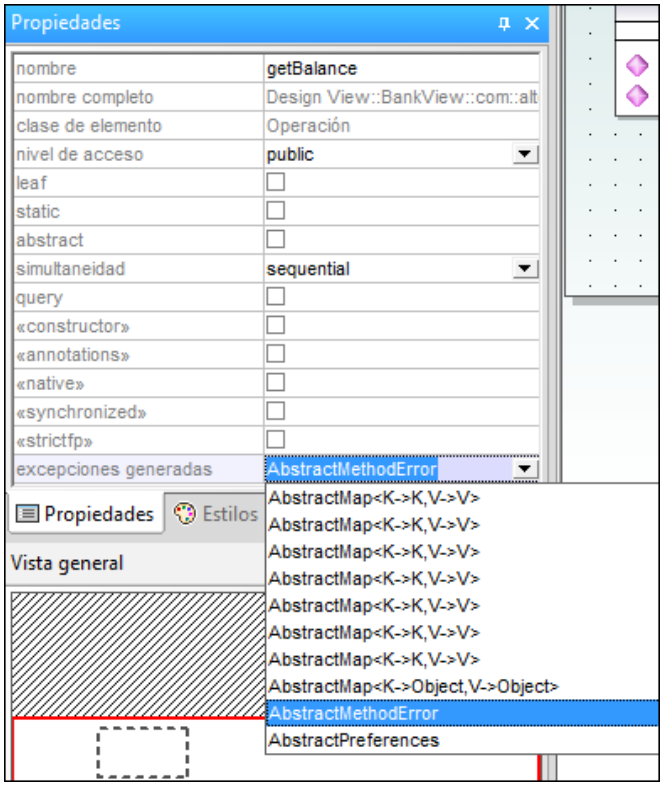
1. En la Estructura del modelo haga clic en el método de la clase en la que desea agregar la excepción generada (p. ej. el método `getBalance` de la clase `Account`).
2. Ahora haga clic con el botón secundario dentro del panel Propiedades y elija la opción **Agregar excepción generada** del menú contextual.

 © 2014 Altova GmbH



Esto añade el campo `Excepciones generadas` al panel Propiedades y selecciona la primera opción de la lista desplegable.

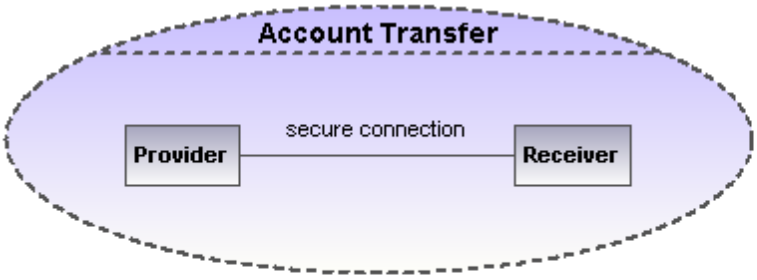
3. Seleccione una opción de la lista despegable del campo `Excepciones generadas` o escriba el nombre que quiera.

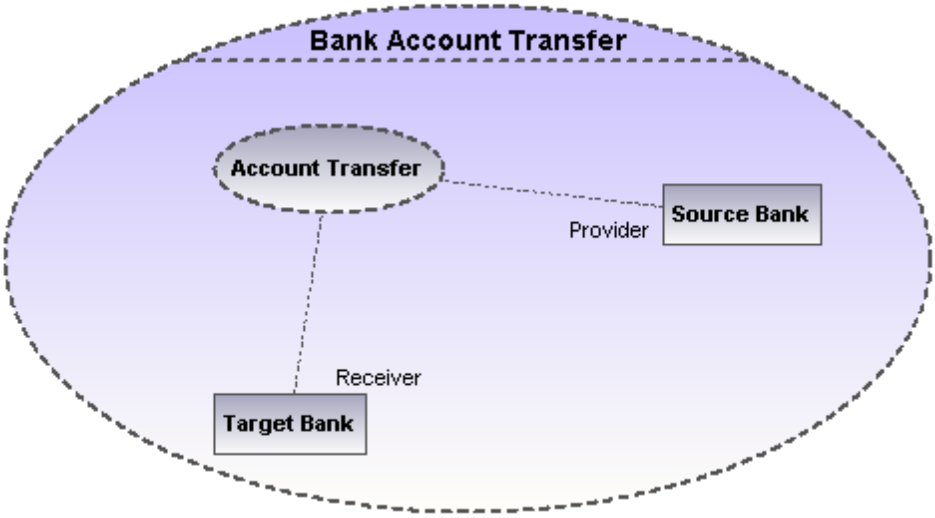


10.2.2 Diagrama de estructura de un compuesto

Sitio web de Altova: diagramas de estructura de un compuesto UML

Los diagramas de estructura de un compuesto son un tipo de diagrama añadido en la especificación UML 2.0 y sirven para mostrar la estructura interna (partes, puertos, conectores...) de un clasificador estructurado o colaboración.





Insertar elementos

Usar iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de estructura de un compuesto.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama de estructura de un compuesto:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de estructura de un compuesto.

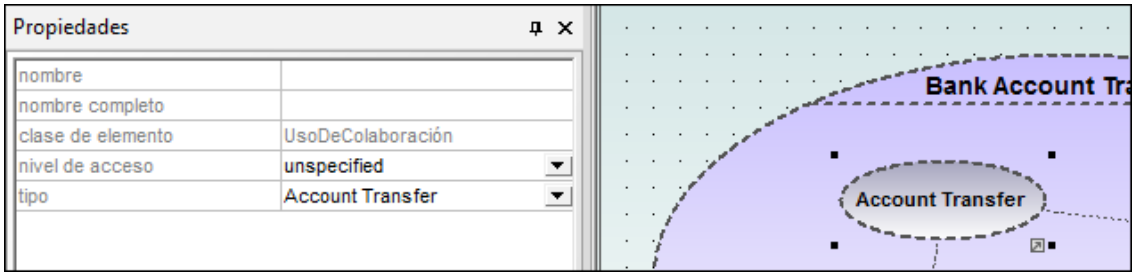
 © 2014 Altova GmbH

Colaboración

Este icono inserta un elemento `colaboración`, que es un tipo de clasificador/instancia que se comunica con otras instancias para producir el comportamiento del sistema.

UsoDeColaboración

Este icono inserta un elemento `usoDeColaboración`, que representa un uso determinado de una colaboración en la que participan determinadas clases o instancias que desempeñan el papel de la colaboración. Un uso de colaboración se representa por medio de una elipse con una línea de puntos. Dentro de la elipse aparece el nombre de la instancia, un punto y coma y el nombre del tipo de colaboración.



Cuando cree dependencias entre elementos uso de colaboración, es necesario rellenar el campo tipo para poder crear el enlace de roles y la colaboración de destino debe tener una parte/un rol como mínimo.

Parte (Propiedad)

Inserta un elemento `parte`, que representa un conjunto de instancias propiedad de un clasificador contenedor. Los elementos parte pueden añadirse a colaboraciones y a clases.

Puerto

Inserta un elemento `puerto`, que define el punto de interacción entre un clasificador y su entorno. Los elementos puerto pueden añadirse en partes con un tipo definido.

Clase

Inserta un elemento `clase`, que es el clasificador que tiene lugar en ese uso concreto de la colaboración.

Conector

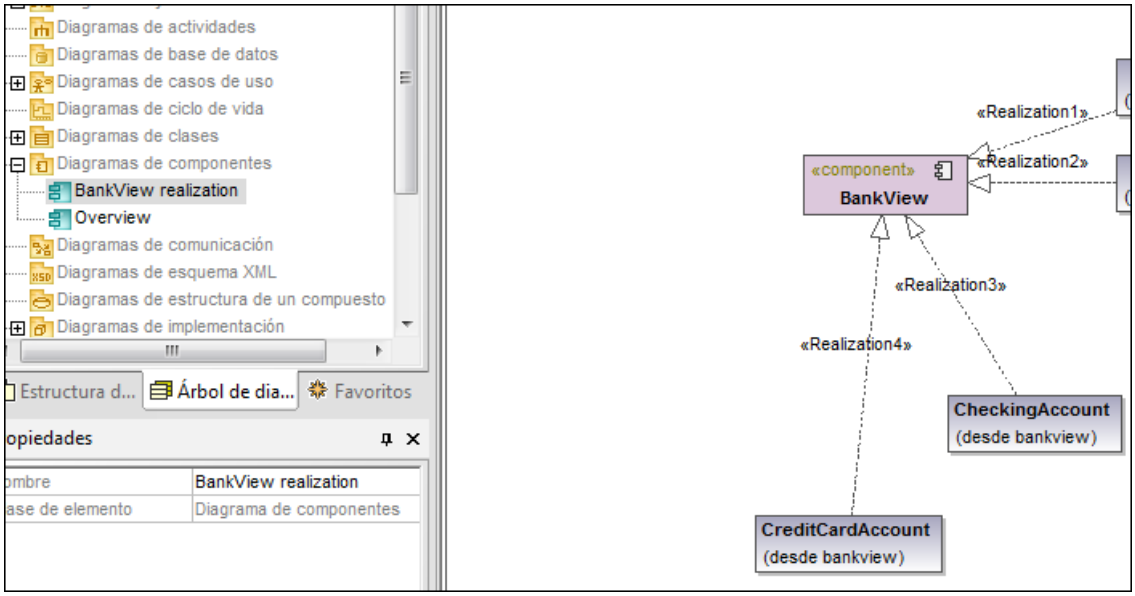
Inserta un elemento `conector`, que se puede usar para conectar dos o más instancias de una parte o de un puerto. El conector define la relación entre los objetos e identifica la comunicación entre los roles.

Dependencia (Enlace de roles)

Inserta el elemento `dependencia`, que indica qué rol desempeña cada elemento conectable del clasificador o de la operación en la colaboración.

10.2.3 Diagrama de componentes

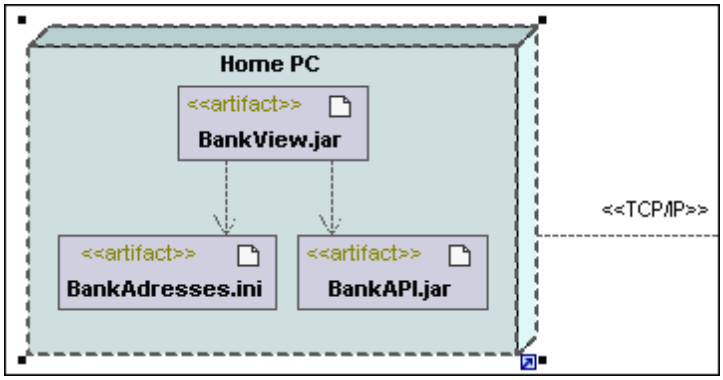
Para más información sobre cómo añadir componentes al diagrama consulte el apartado Diagramas de componentes.



 © 2014 Altova GmbH

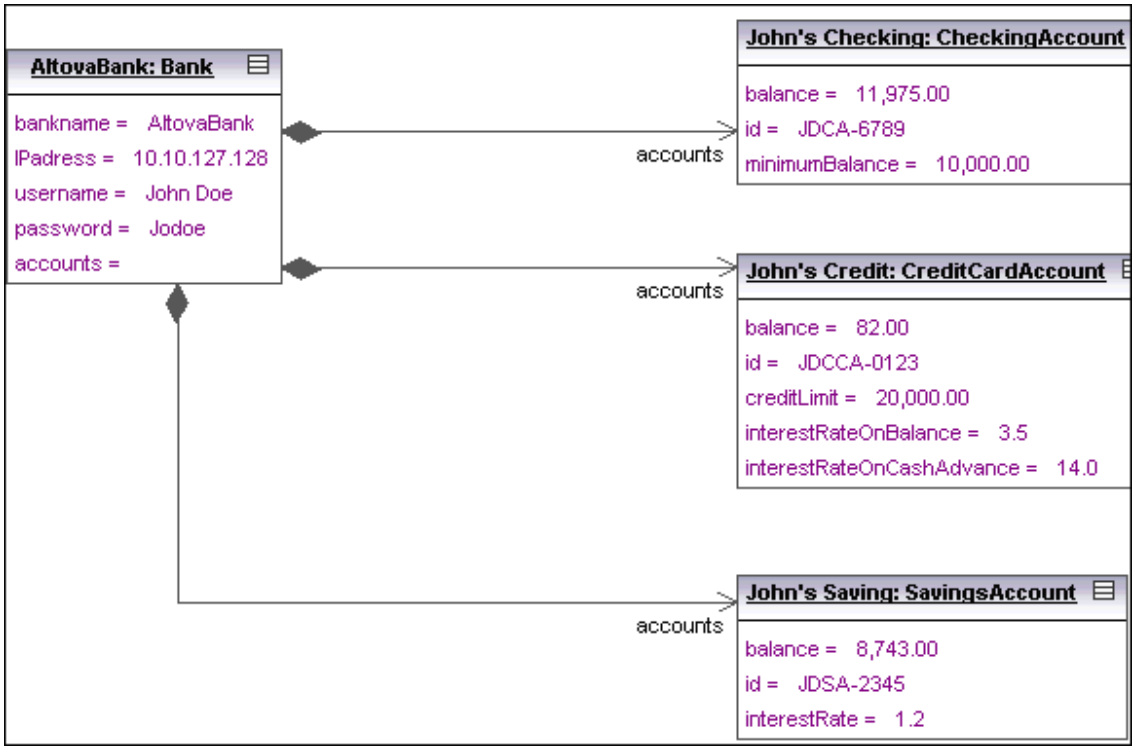
10.2.4 Diagrama de implementación

Para más información sobre cómo agregar nodos y artefactos al diagrama consulte el apartado Diagramas de implementación del tutorial.



10.2.5 Diagrama de objetos

Para más información sobre cómo agregar objetos/instancias nuevos al diagrama consulte el apartado Diagramas de objetos del tutorial.

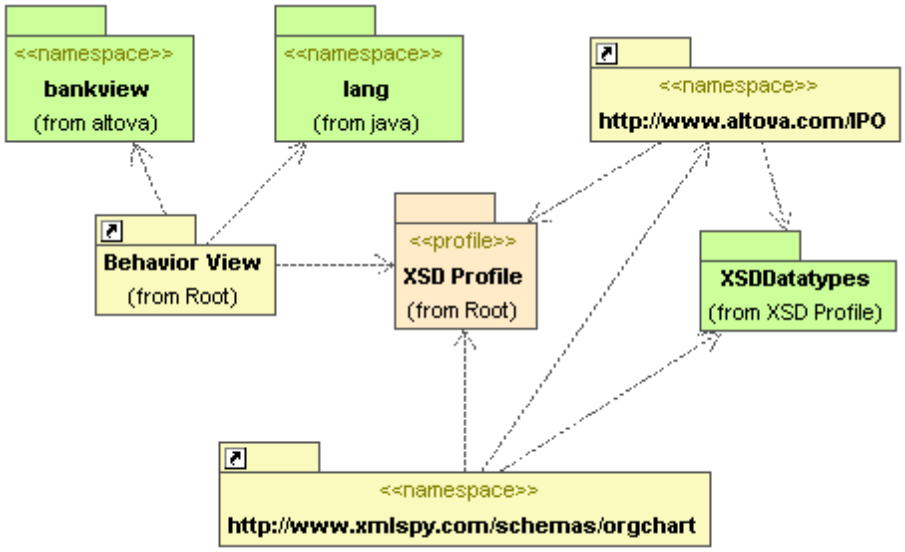


 © 2014 Altova GmbH

10.2.6 Diagrama de paquetes

Los diagramas de paquetes ilustran la organización de los paquetes y de sus elementos, así como sus correspondientes espacios de nombres. Además en UModel puede crear hipervínculos para navegar hasta el contenido del paquete correspondiente.

Los paquetes se dibujan en forma de carpetas y se pueden usar en cualquier diagrama UML, aunque se usan sobre todo en diagramas de casos de uso y de clases.

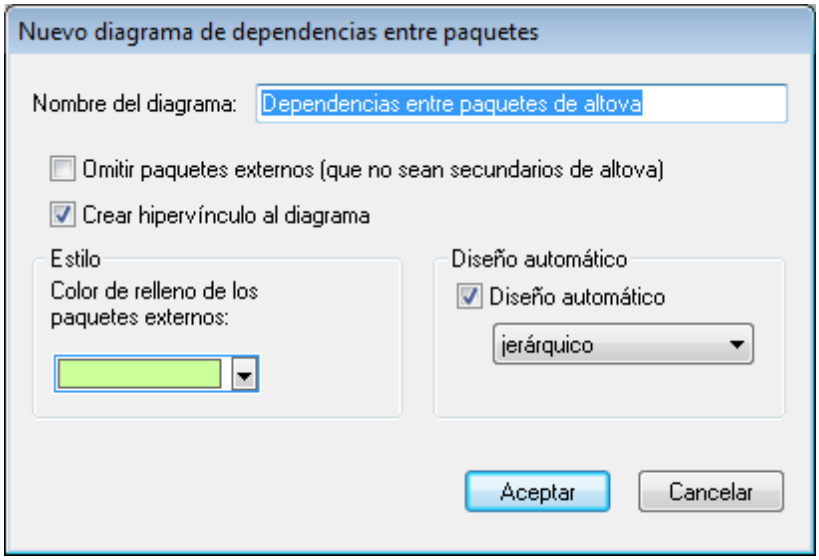


Generación automática de diagramas de dependencias entre paquetes
UModel tiene una función para generar diagramas de dependencias entre paquetes a partir de cualquier paquete del panel Estructura del modelo.

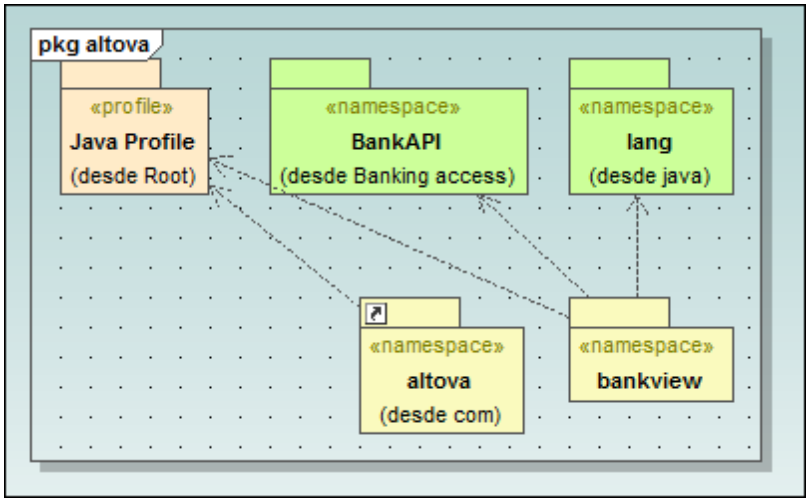
Esta función crea vínculos de dependencia entre los paquetes si existen referencias entre sus elementos de modelado. P. ej. si hay dependencias entre clases, si hay clases derivadas o si los atributos tienen tipos que están definidos en otro paquete.

Para generar un diagrama de dependencias entre paquetes:

1. En la Estructura del modelo haga clic con el botón secundario en un paquete (p. ej. `altova`) y elija **Mostrar en un diagrama nuevo de | Dependencias entre paquetes...**. Esto abre el cuadro de diálogo "Nuevo diagrama de dependencias entre paquetes".



2. Seleccione las opciones que necesite en el cuadro de diálogo y haga clic en **Aceptar**.



UModel genera un diagrama nuevo y muestra las dependencias entre los paquetes del paquete `altova`.

Insertar elementos

Usar iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de paquetes.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos desde la Estructura del modelo hasta el diagrama de paquetes:

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama

 © 2014 Altova GmbH

(puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de paquetes.

Paquete

Inserta el elemento `paquete` en el diagrama. Los paquetes sirven para agrupar elementos y para aportar un espacio de nombres para los elementos agrupados. Al ser un espacio de nombres, un paquete puede importar elementos concretos de otros paquetes o todos sus elementos. Los paquetes también se pueden combinar con otros paquetes.

Perfil

Inserta el elemento `perfil`, un tipo de paquete que se puede aplicar a otros.

El paquete `perfiles` se usa para extender el metamodelo UML. La construcción de extensión principal es el estereotipo, que a su vez es parte del perfil. Los perfiles siempre deben estar relacionados con un metamodelo de referencia como UML, es decir, no pueden existir por sí mismos.

Dependencia

Inserta el elemento `dependencia`, que indica una relación de proveedor/cliente entre los elementos de modelado (en este caso paquetes o perfiles).

ImportaciónDePaquete

Inserta una relación de importación `<<import>>` que muestra que los elementos del paquete incluido se importarán en el paquete de destino. El espacio de nombres del paquete de destino obtiene acceso al espacio de nombres incluido. El espacio de nombres del paquete incluido no se ve afectado.

Nota: los elementos `private` de un paquete no se pueden combinar ni importar.

CombinaciónDePaquete

Inserta una relación de combinación `<<merge>>` que muestra que los elementos del paquete combinado (de origen) se importarán en el paquete de destino, incluido el contenido importado del paquete combinado (de origen).

Si en el paquete de destino existe el mismo elemento, las definiciones de estos elementos se expanden con las del paquete de destino. Los elementos actualizados o agregados se señalan por medio de una relación de generalización que apunta al paquete de origen.

Nota: los elementos `private` de un paquete no se pueden combinar ni importar.

AplicaciónDePerfil

Inserta una aplicación de perfil que muestra qué perfiles se aplicaron al paquete. Se trata de un tipo de importación de paquete que afirma que un perfil se aplicó a un paquete.

El perfil extiende el paquete al que se aplicó. Al aplicar un perfil (usando el icono **AplicaciónDePerfil**), todos los estereotipos que forman parte del perfil también estarán a disposición del paquete.

Los nombres de perfil aparecen en forma de líneas discontinuas con puntas de flecha que van del paquete hasta el perfil aplicado y llevan la etiqueta `<<apply>>`.

10.2.7 Diagrama de perfil y estereotipos

Sitio web de Altova: Diagramas de perfil UML

Con los diagramas de perfil de UModel puede definir estereotipos, valores etiquetados y restricciones personales en un diagrama especial.

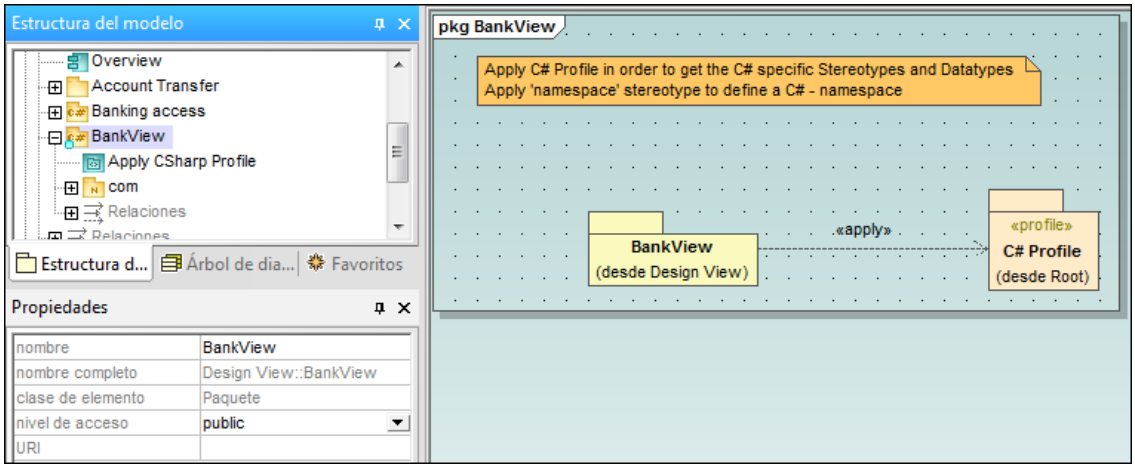
Los perfiles y los estereotipos sirven para extender el metamodelo UML. La construcción de extensión principal es el estereotipo, que forma parte del perfil. Los perfiles siempre deben estar relacionados con un metamodelo de referencia como UML porque no pueden existir por sí solos. UModel ofrece un tipo de diagrama de perfil donde puede definir estereotipos propios.

El archivo de ejemplo `Java Profile.ump` (o `C# Profile.ump` o `VB Profile.ump`) debe aplicarse cuando se creen proyectos nuevos de UModel con el comando de menú **Proyecto | Incluir un subproyecto**. Este perfil aporta los tipos de datos y estereotipos Java y es fundamental para crear código mediante ingeniería de ida y vuelta.

En esta sección explicamos:

- Cómo agregar estereotipos y definir valores etiquetados
- Los estereotipos y las enumeraciones
- Los estilos de estereotipo definidos por el usuario

Para explicar todo ello utilizamos el archivo de ejemplo `Bank_CSharp.ump` (de la carpeta `...\UModelExamples`). El perfil C# se aplicó al paquete **`BankView`**.



- Los perfiles son tipos de paquetes concretos que se aplican a otros paquetes.
- Los estereotipos son metaclases concretas que extienden las clases estándar.
- Los valores etiquetados son los valores de los atributos del estereotipo.

Una `AplicaciónDePerfil` muestra qué perfiles se aplicaron a un paquete y es un tipo de importación de paquete que afirma que un perfil se aplicó a un paquete. El perfil extiende el paquete al que se aplicó. Si se aplica un perfil a un paquete (con ayuda de **AplicaciónDePerfil**), significa que todos los estereotipos que forman parte del perfil estarán a disposición del paquete.

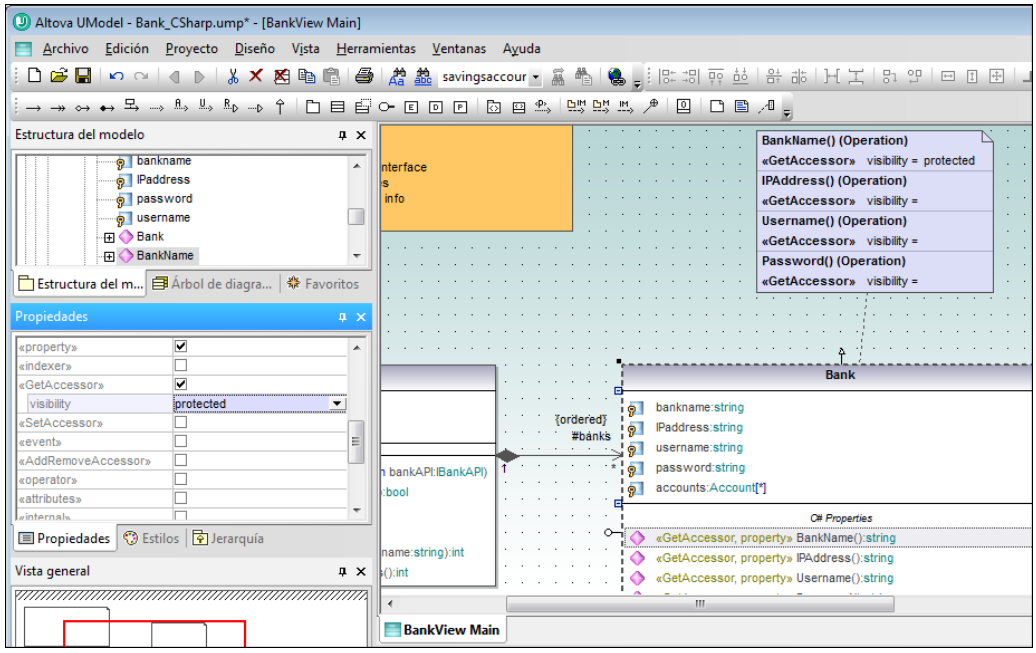
Las `AplicacionesDePerfil` se dibujan como líneas de puntos con punta de flecha que van **desde el paquete** hasta el perfil aplicado y tienen la etiqueta `<<apply>>`.

 © 2014 Altova GmbH

,

Estereotipos:
Un estereotipo define cómo se puede extender una metaclase. Se trata de un tipo de clase que extiende otras clases por medio de extensiones. Los estereotipos solamente se pueden crear en perfiles y se representan en diagramas de clases como clases estándar, pero con la palabra clave `<<stereotype>>` encima del nombre de la clase.

- Los estereotipos pueden tener **propiedades**, llamadas **definiciones de etiquetas**.
- Cuando se aplica el estereotipo a un elemento de modelado, los valores de las propiedades reciben el nombre de **valores etiquetados**.
- Cuando se aplican estereotipos que tienen propiedades, los valores etiquetados aparecen automáticamente en un comentario. Para más información sobre cómo personalizar la vista de valores etiquetados consulte este apartado.
- Los estereotipos tienen su propia familia de estilos
- Si el atributo es de tipo `enumeration`, entonces podrá seleccionar uno de los valores predefinidos. También puede insertar/seleccionar el valor concreto en el panel Propiedades (p. ej. `<<GetAccessor>> nivel de acceso = public`).

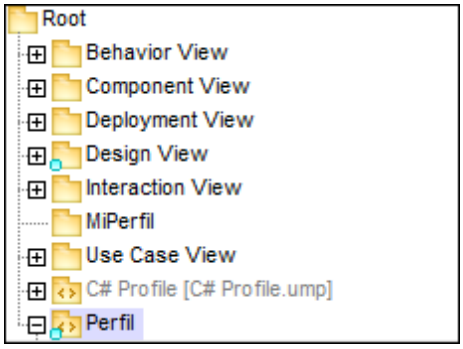


Agregar estereotipos y definir valores etiquetados

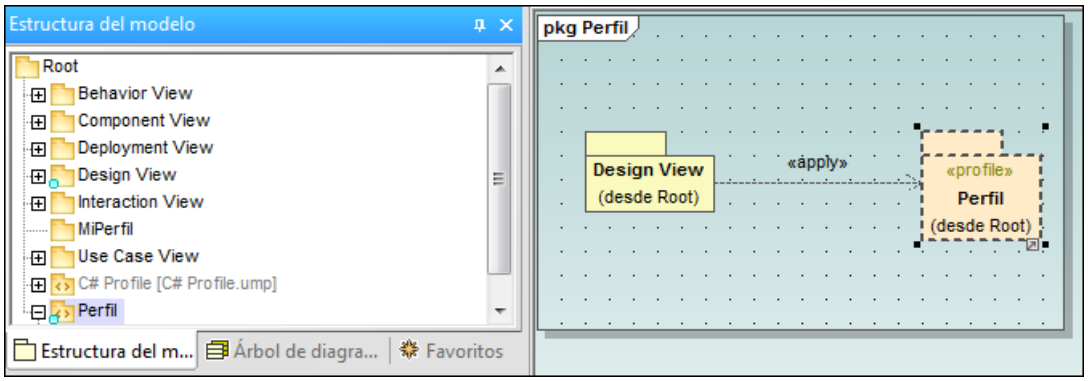
Para los ejemplos que aparecen a continuación utilizamos el archivo `Bank_MultiLanguage.ump` de la carpeta `...\UModelExamples`.

Crear un diagrama de perfil y un estereotipo

1. Cree un perfil nuevo desde el panel Estructura del modelo. Por ejemplo: haga clic con el botón secundario en el paquete **`Root`** y elija **Elemento nuevo | Perfil**. Póngale el nombre **`Perfil`** al nuevo perfil.

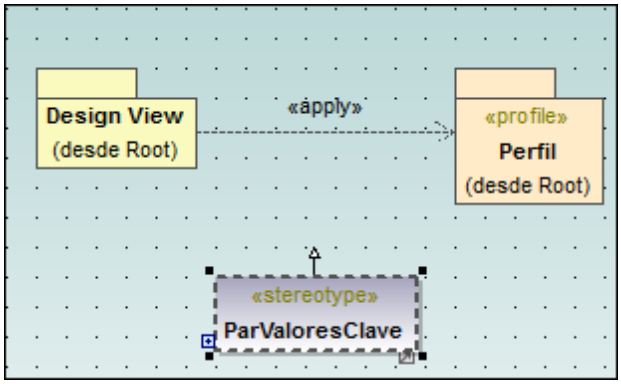


2. Ahora haga clic con el botón secundario en `Perfil` y elija **Diagrama nuevo | Diagrama de perfil**.
   Esto añade un diagrama de perfil al paquete seleccionado.
2. Arrastre el perfil recién creado `Perfil` desde la Estructura del modelo hasta el área de trabajo del diagrama de perfil.
3. Arrastre también el paquete `DesignView` hasta el área de trabajo del diagrama de perfil.
4. Ahora haga clic en el icono **AplicaciónDePerfil** de la barra de herramientas. Haga clic en el paquete `DesignView` y arrastre el puntero del mouse hasta el paquete `Perfil`.



Esto permite usar en el paquete `DesignView` y en sus subpaquetes todos los estereotipos que estén definidos en el perfil `Perfil`.

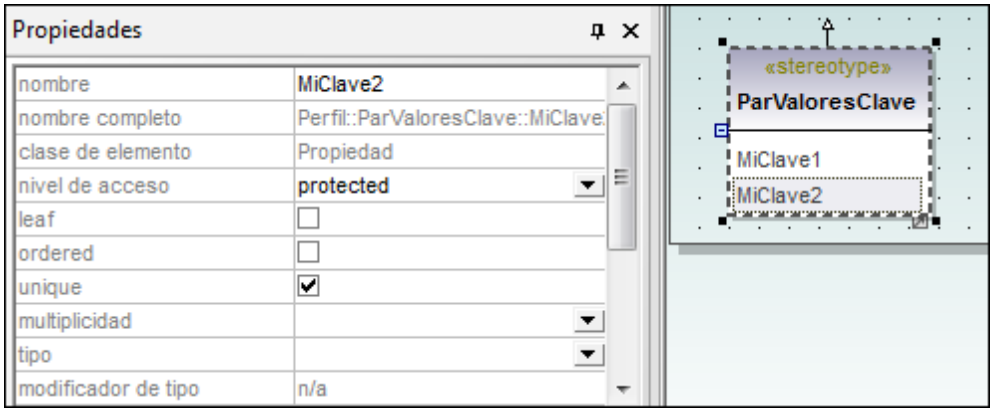
5. Haga clic en el icono **Estereotipo** de la barra de herramientas e inserte un estereotipo de clase (p. ej. `ParValoresClave`).



6. Pulse **F7** para agregar un atributo al estereotipo (p. ej. `MiClave1`). Añada otro atributo

 © 2014 Altova GmbH

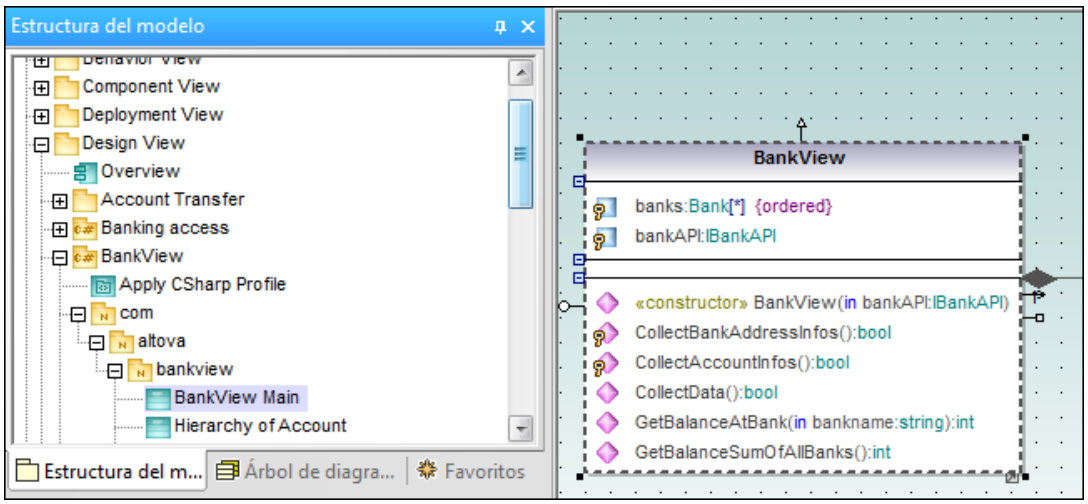
llamado `MiClave2`.



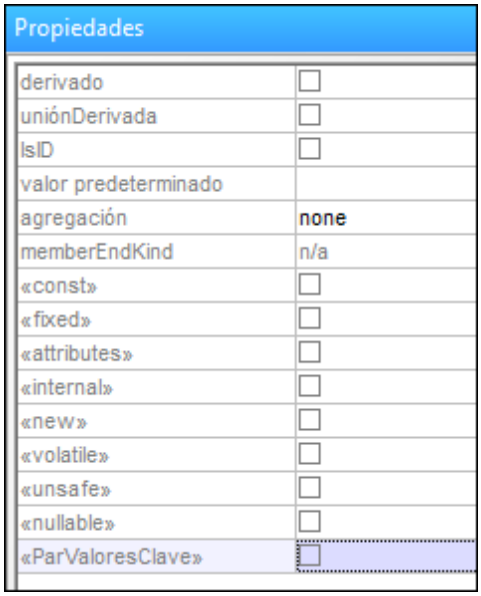
Por ahora hemos terminado de **definir** el estereotipo. En el siguiente paso podemos usar/asignar el estereotipo cuando añadamos atributos a una clase que forme parte del paquete **`BankView`**.

Usar y asignar estereotipos

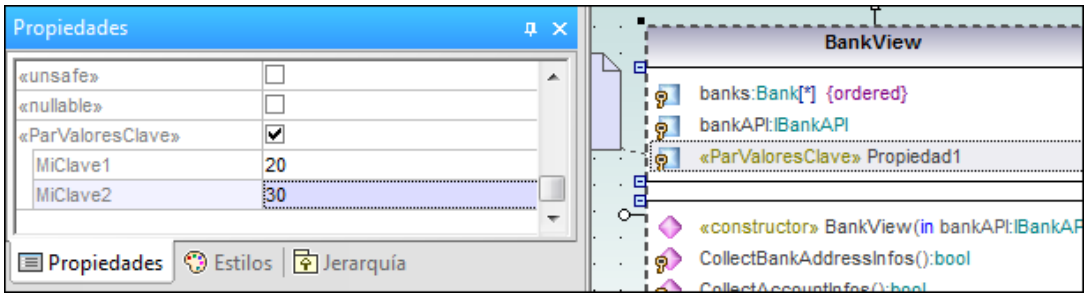
1. En la Estructura del modelo haga doble clic en el icono del diagrama de clases **`BankView Main`**.
   Esto abre el diagrama de clases y muestra las asociaciones que existen entre las clases.



   Ahora vamos a añadir un atributo a la clase **`BankView`** y asignar el estereotipo definido previamente.
2. Haga clic en la clase **`BankView`** y pulse **F7** para añadir un atributo.
3. En el panel Propiedades desplácese hasta el final de la lista. Observe que el estereotipo `ParValoresClave` está disponible en el panel de propiedades.

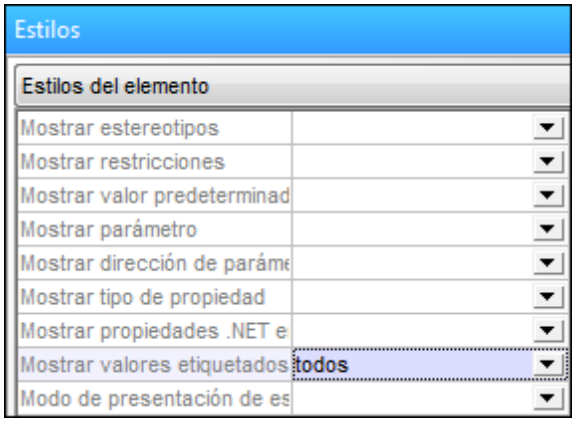


4. Marque la casilla `ParValoresClave` para aplicarlo. Los dos valores etiquetados `MiClave1` y `MiClave2` aparecen ahora bajo el estereotipo en el panel Propiedades.
5. Haga doble clic en el campo de los valores etiquetados e inserte un valor para cada uno.



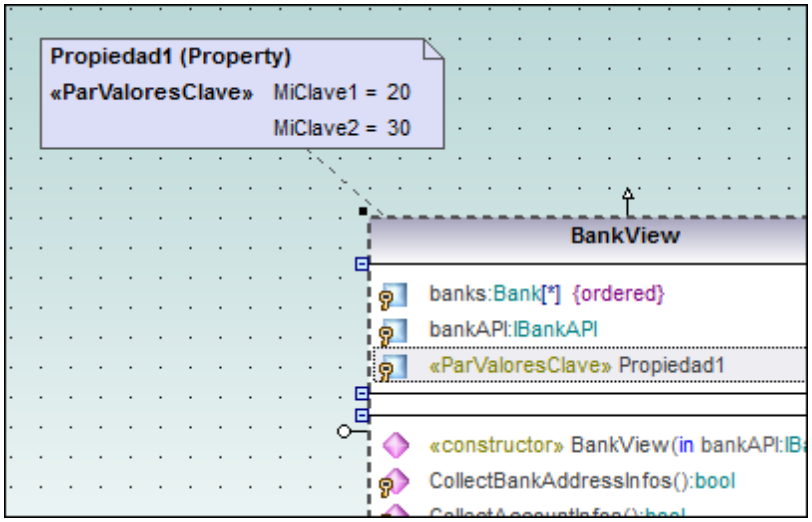
Mostrar valores etiquetados en un diagrama

1. Abra la pestaña *Estilos* y desplácese hasta la entrada `Mostrar valores etiquetados`. Seleccione el valor `todos`.



Ahora el diagrama muestra los valores etiquetados en el elemento `Nota`. Para editar un

valor del elemento `Nota`, haga doble clic.



Mostrar valores etiquetados | en compartimento
Este comando del menú contextual muestra los valores etiquetados en un compartimiento separado de la clase.

Nota: cuando oculte atributos y operaciones con el comando **Mostrar u ocultar el contenido del nodo...** del menú contextual, los valores etiquetados también se muestran/ocultan junto con el elemento de modelado.

Los extremos de asociación pueden mostrar estereotipos si asigna el valor `verdadero` a la opción `Mostrar estereotipos de memberEnd` en el panel Estilos.

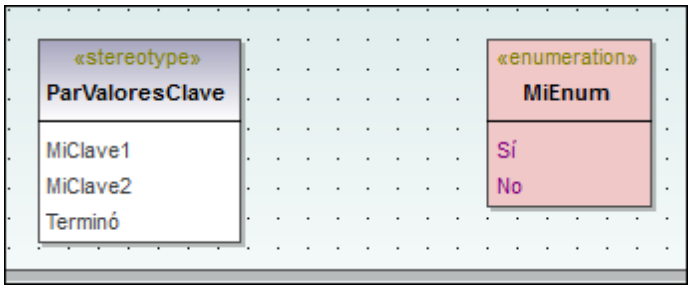
Para más información consulte el apartado Ver valores etiquetados.

Estereotipos y enumeraciones

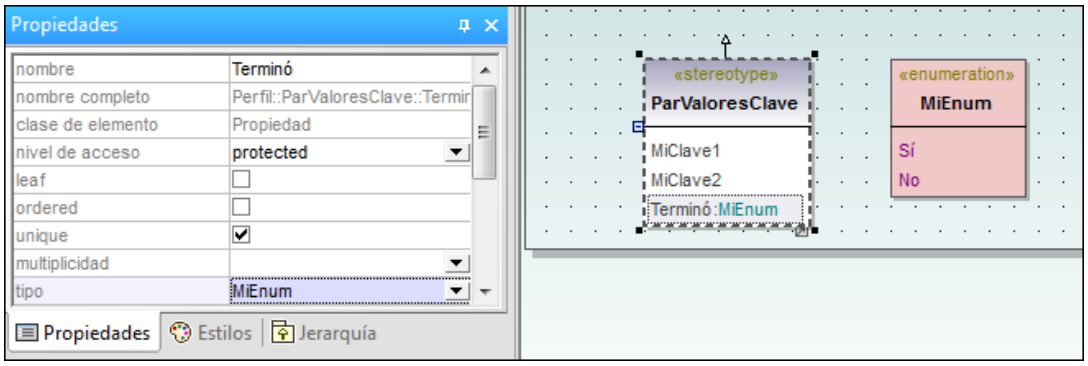
UModel ofrece un método muy eficaz para seleccionar valores enumerados de estereotipos.

Haga clic en la pestaña del diagrama que contiene la definición de estereotipo (es decir, el diagrama de perfil que añadimos en el primer paso):

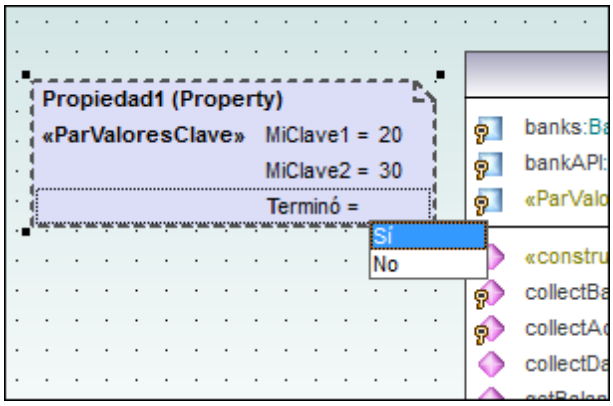
1. Haga clic en el icono **Enumeración** de la barra de herramientas para insertar una enumeración en el diagrama (que contiene el estereotipo previamente definido).
2. Pulse **Mayús+F7** para añadir los LiteralesDeEnumeración `Sí` y `No` a la enumeración. Si lo prefiere puede añadirlos desde el menú contextual.



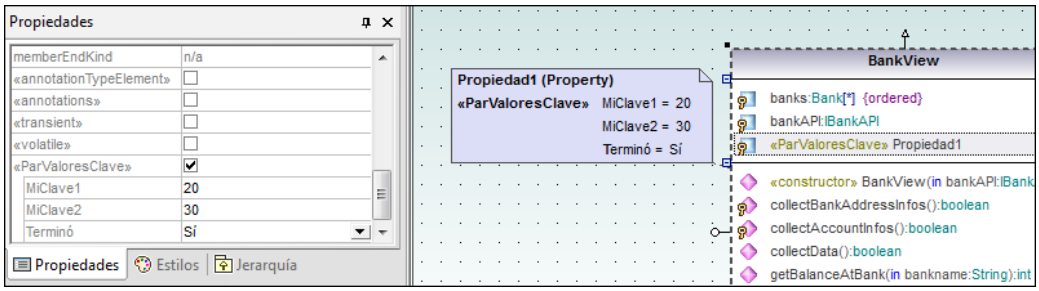
3. Haga clic en la clase estereotipo y pulse **F7** para añadir un atributo/una propiedad nueva (p. ej. `Terminó`).
4. Ahora en el panel Propiedades seleccione el tipo `MiEnum`.



5. Ahora vuelva al diagrama de clases **`BankView Main`**.
6. Observe que la propiedad `Terminó` aparece ahora como valor etiquetado en la nota.



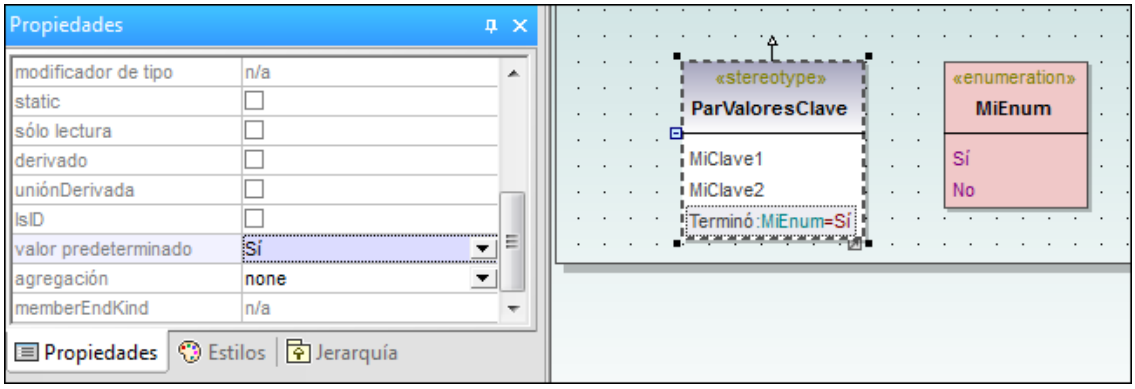
Haga doble clic en el valor etiquetado `Terminó` para ver los valores de enumeración predefinidos. Haga clic en una enumeración para seleccionarla.



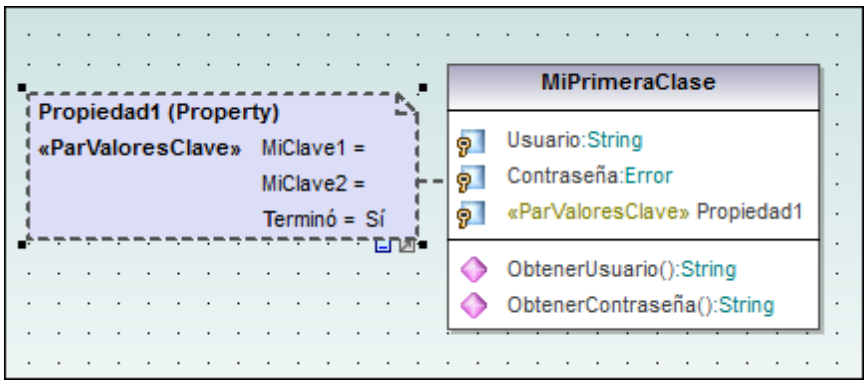
Valores de enumeración predeterminados

UModel también permite definir valores etiquetados **predeterminados**. Cuando añada un atributo al estereotipo, haga doble clic en el campo predeterminado e inserte una de las enumeraciones disponibles como valor predeterminado.

 © 2014 Altova GmbH



En el ejemplo de la imagen anterior se insertó el valor predeterminado `Sí`.
Cuando añada una propiedad a una clase y esté seleccionado el tipo `MiEnum`, el valor predeterminado se inserta automáticamente como valor etiquetado (es decir, `Terminó = Sí`).

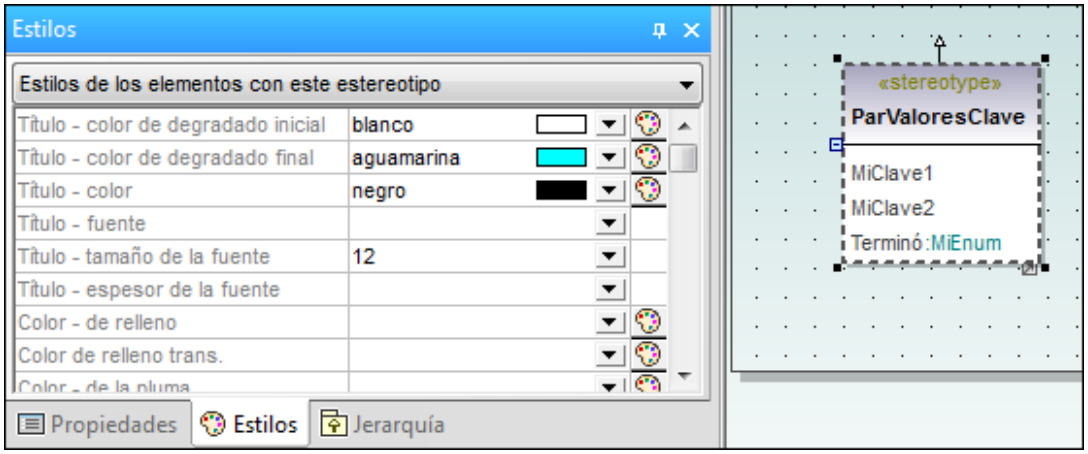


Estilos de estereotipo definidos por el usuario

En UModel puede crear estilos definidos por el usuario para cada estereotipo. Esto significa que puede tener fuentes y colores diferentes para cada estereotipo y aplicarlos a las clases que sean del tipo de estereotipo.

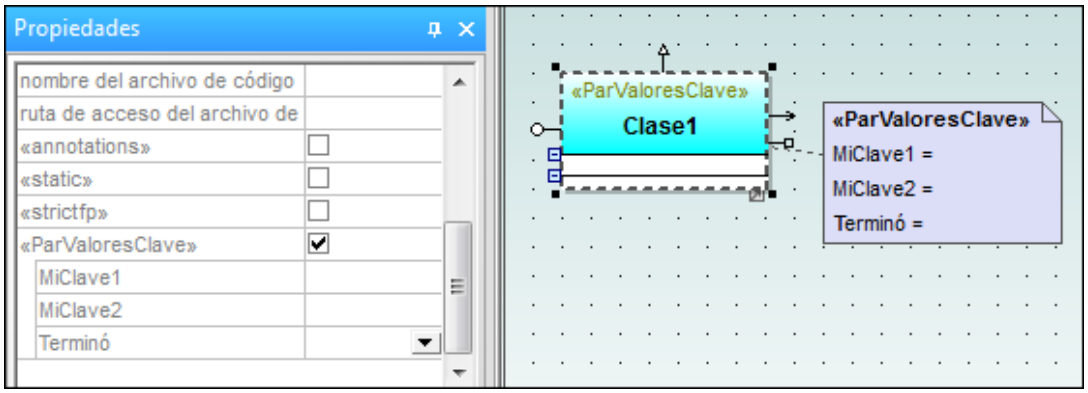
Para crear estilos de estereotipo definidos por el usuario:

1. Haga clic en un estereotipo (p. ej. `ParValoresClave`).
2. Abra la pestaña *Estilos* y, en el cuadro combinado de la parte superior, seleccione la opción **Estilos de elementos con este estereotipo**.
3. Defina los estilos de este estereotipo (p. ej. `Título - color de degradado final = aguamarina`).



Al hacer clic en la clase del estereotipo, aparecen automáticamente los estilos definidos para el estereotipo en el panel Estilos.

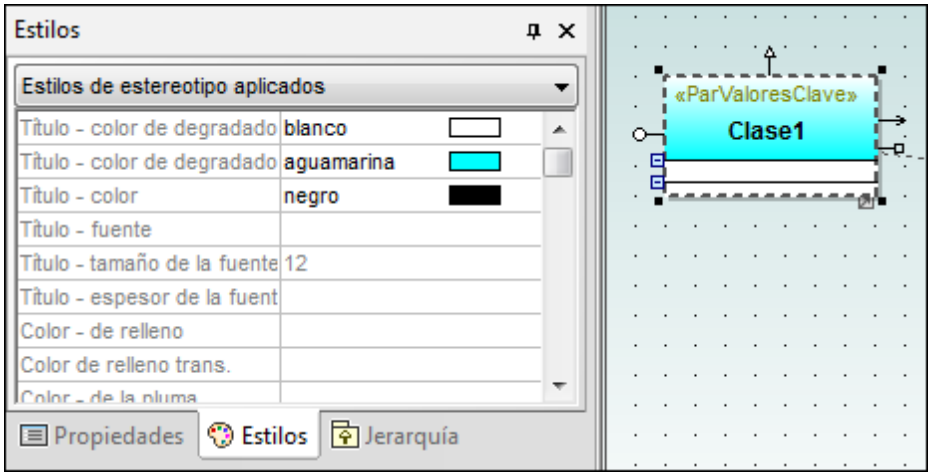
4. Ahora pase al diagrama de clases e inerte una clase nueva.
5. Haga clic en el título o encabezado de la clase y marque la casilla `ParValoresClave` en el panel Propiedades.



La clase nueva tiene los estilos que se asignaron al estereotipo (el color de degradado del título es aguamarina, p. ej.). No olvide que los estilos del estereotipo **no se aplican** si el estereotipo se aplica a una propiedad u operación de la clase.

6. Haga clic en la nueva clase estereotipo del diagrama y después abra la pestaña *Estilos*.
7. En el cuadro combinado situado en la parte superior seleccione la opción **Estilos de estereotipo aplicados**.

 © 2014 Altova GmbH



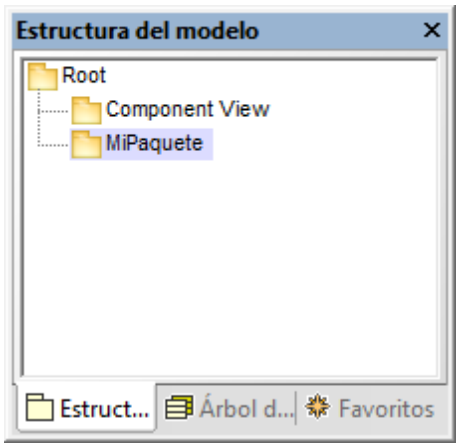
Aquí obtiene una vista previa de las opciones de estilo definidas para este estereotipo, pero no puede cambiar las opciones. Esto debe hacerse desde el diagrama de clases donde se definió el estereotipo.

Asignar iconos de estereotipo personalizados

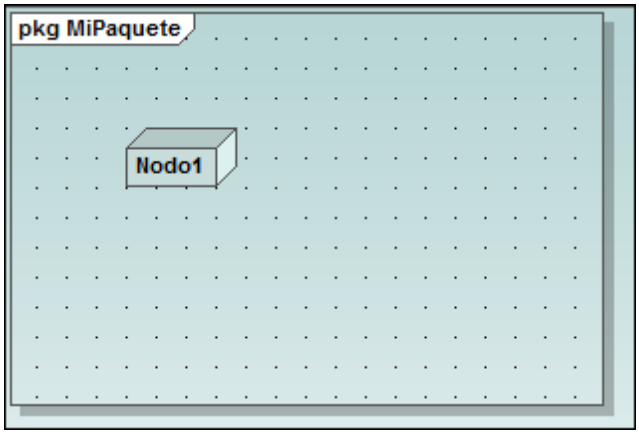
Para terminar, en UModel también puede crear iconos personalizados para sus estereotipos.

Para crear un icono de estereotipo personal:

1. Haga clic con el botón secundario en el paquete `Root` y elija **Elemento nuevo | Paquete** en el menú contextual.
2. A este paquete le ponemos el nombre `MiPaquete`.

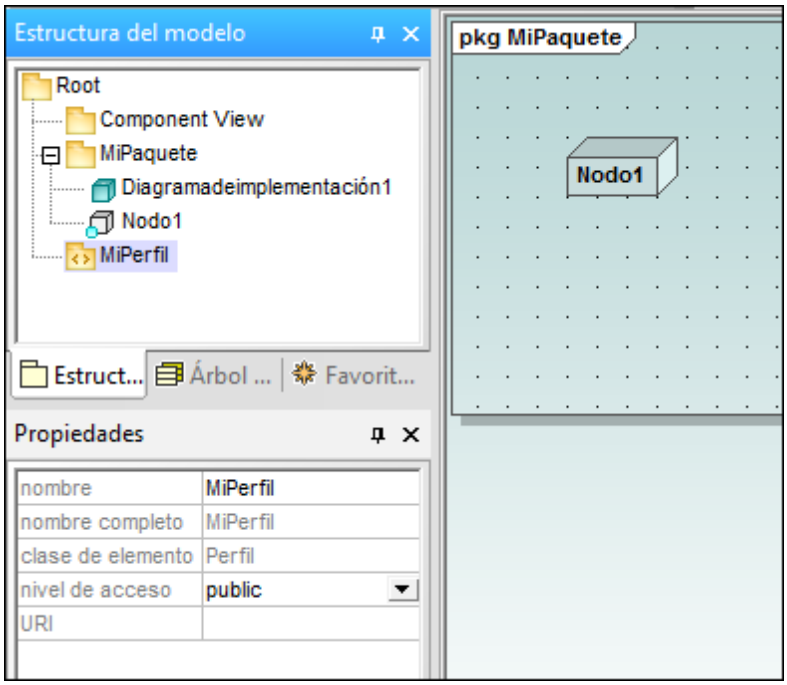


3. Haga clic con el botón secundario en `MiPaquete` y elija **Diagrama nuevo | Implementación** en el menú contextual.
   Esto crea un diagrama de implementación en `MiPaquete`.
4. Ahora haga clic con el botón secundario en `MiPaquete` y seleccione **Nuevo/a | Nodo**.



Esto añade el objeto `Nodo1` al diagrama.

5. En la Estructura del modelo haga clic con el botón secundario en el paquete `Root` y después elija **Elemento nuevo | Perfil**.
6. Ponga el nombre `MiPerfil` a este nuevo perfil.

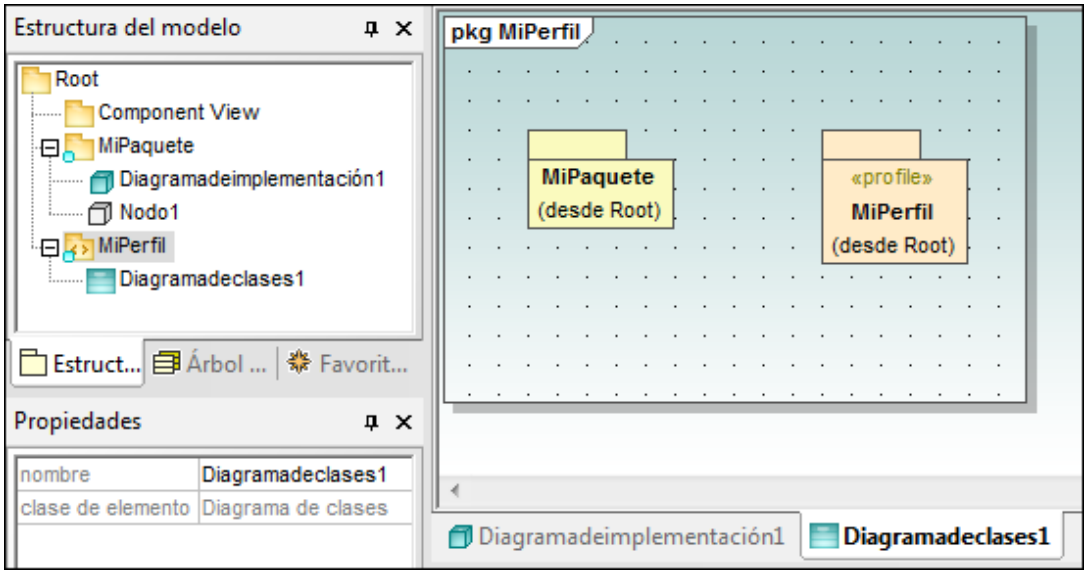


7. Haga clic con el botón secundario en `MiPerfil` y elija **Diagrama nuevo | Diagrama de clases**.

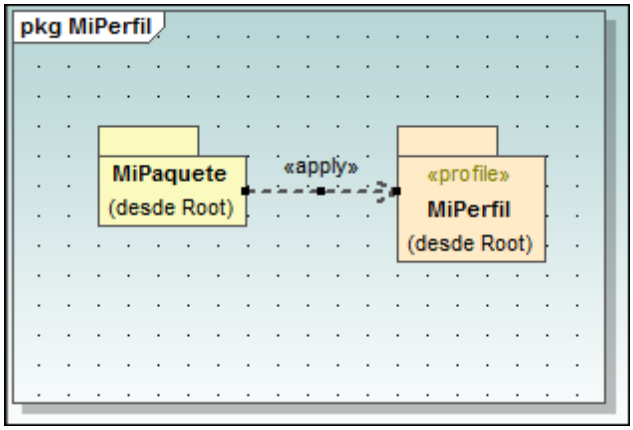
Para seleccionar el icono del estereotipo:

1. Arrastre los paquetes `MiPaquete` y `MiPerfil` desde la Estructura del modelo hasta el área de trabajo del diagrama de clases.

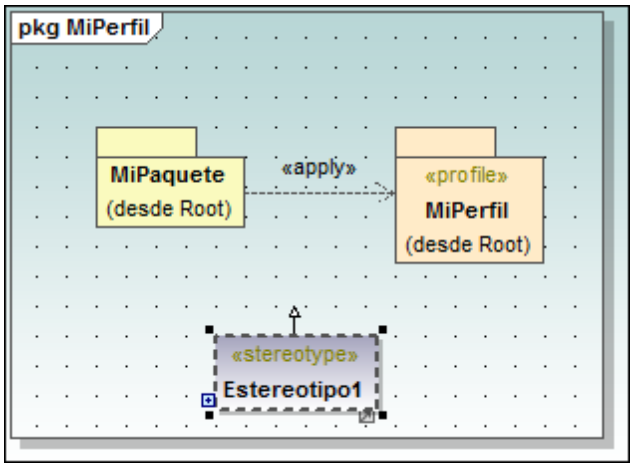
 © 2014 Altova GmbH



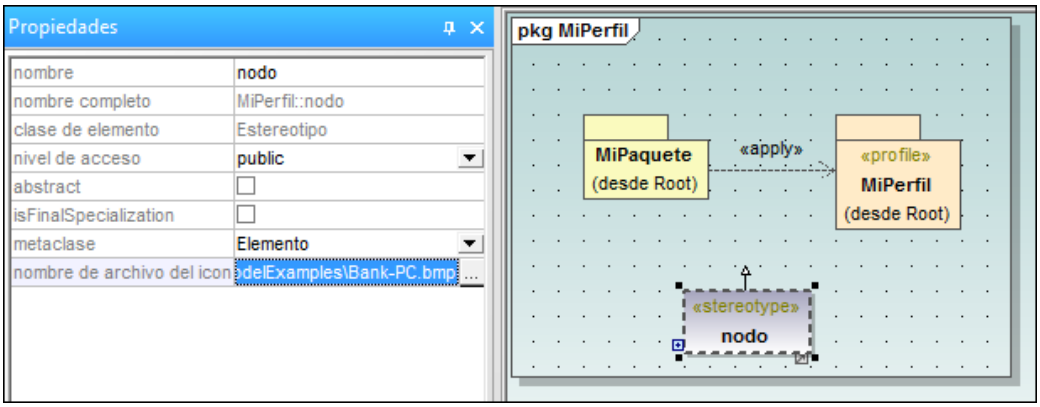
2. Haga clic en el icono **AplicaciónDePerfil** de la barra de herramientas. Haga clic en `MiPaquete` y arrastre el puntero hasta `MiPerfil`.



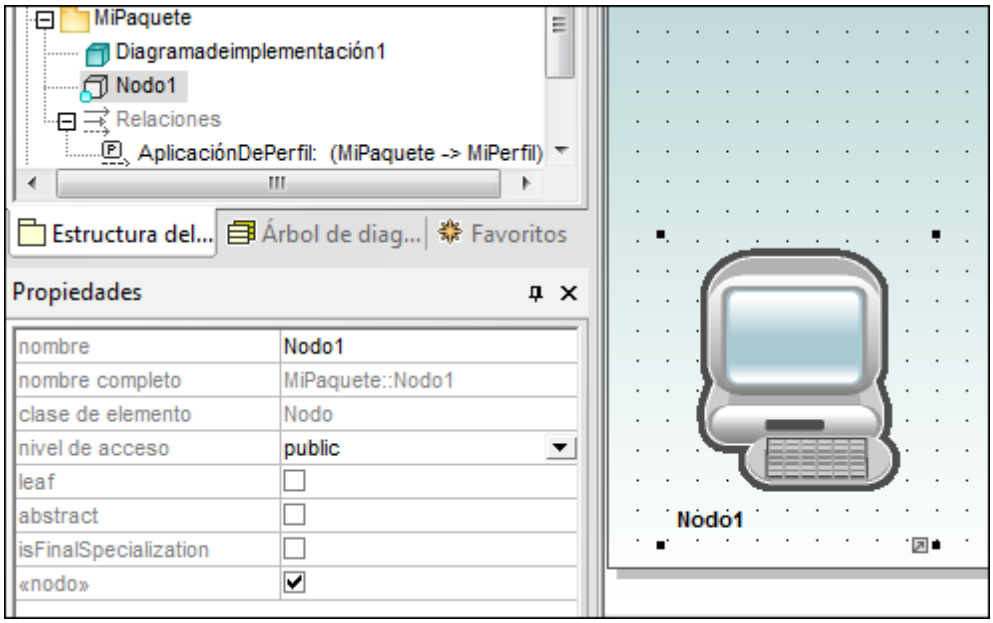
3. Haga clic en el icono **Estereotipo** de la barra de herramientas y haga clic en el paquete `MiPerfil` para insertar el estereotipo.



4. Haga clic en el estereotipo y ahora, en el panel Propiedades, cambie el nombre del elemento de `Estereotipo1` a `nodo`.
5. Cambie también el valor de la propiedad `metaclase` de `Elemento` a `Nodo`.
6. En el campo `nombre del archivo del icono` inserte la ruta de acceso de la imagen que desea usar como icono del estereotipo (o pulse el botón **Examinar** para seleccionar la ruta de acceso).



7. Vuelva al diagrama de implementación (haciendo clic en la pestaña *DiagramaDeImplementación1*).
8. Haga clic en el elemento **`Nodo1`** del diagrama.
9. En el panel Propiedades marque la casilla `<<nodo>>`.



La representación del icono del estereotipo ahora es la imagen seleccionada.

Nota: con los iconos personalizados se utiliza el color RGB 82,82,82 como color de fondo (es decir, transparente).

 © 2014 Altova GmbH

10.3 Otros diagramas

UModel ofrece funciones para importar y generar esquemas XML del W3C y puede generar código a partir de ellos mediante ingeniería directa e inversa. UModel también es compatible con el estándar BPMN (Business Processing Modeling Notation) versión 1.0, que se adoptó como estándar del OMG en febrero de 2006.

Esquemas XML
BPMN (Business Process Modeling Notation)
Diagramas SysML
Diagramas de base de datos

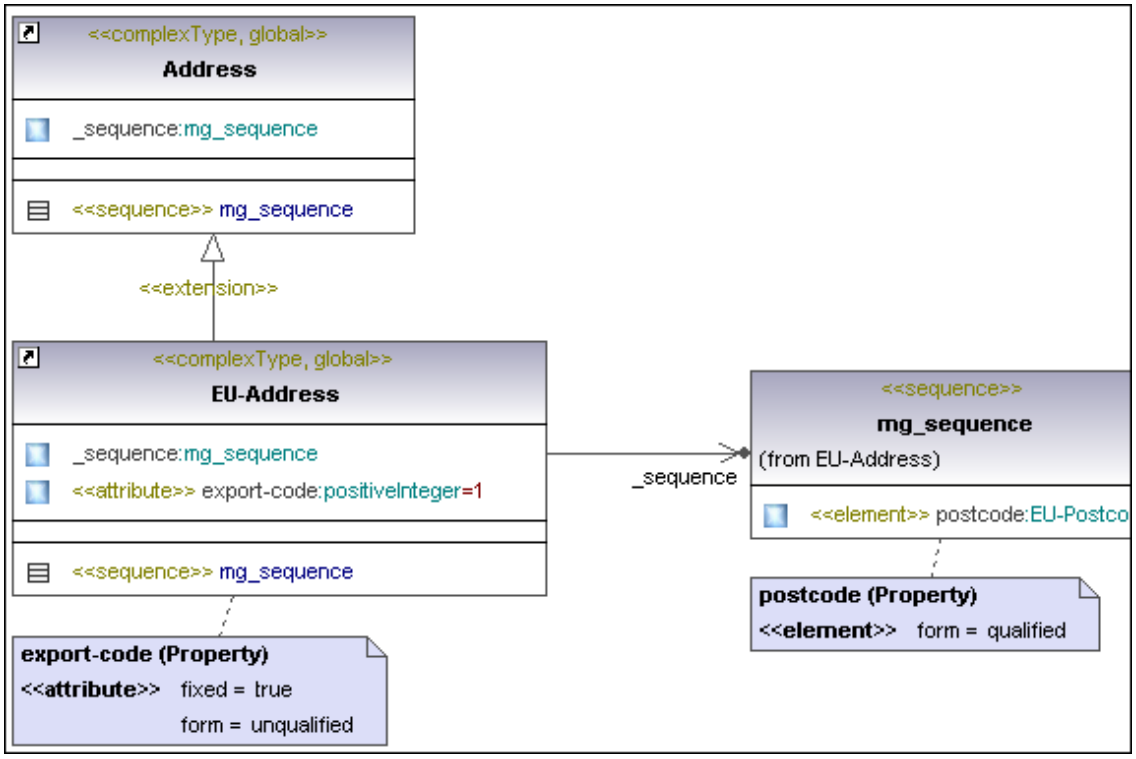
10.3.1 Diagramas de esquema XML

Sitio web de Altova: Esquemas XML en UML

Los diagramas de esquema XML dibujan los componentes de un esquema XML usando la notación UML. Los elementos globales (es decir, los elementos `element`, `simpleType`, `complexType`) se dibujan como clases o tipos de datos y los atributos se representan dentro del compartimiento de atributos.

En el compartimiento de operaciones no hay operaciones. El elemento de modelado *valor etiquetado* se utiliza para presentar los detalles del esquema.

Para ver las correspondencias entre los elementos UML y los elementos/atributos de XML Schema, consulte el apartado Correspondencia entre elementos de XML Schema y de UModel.



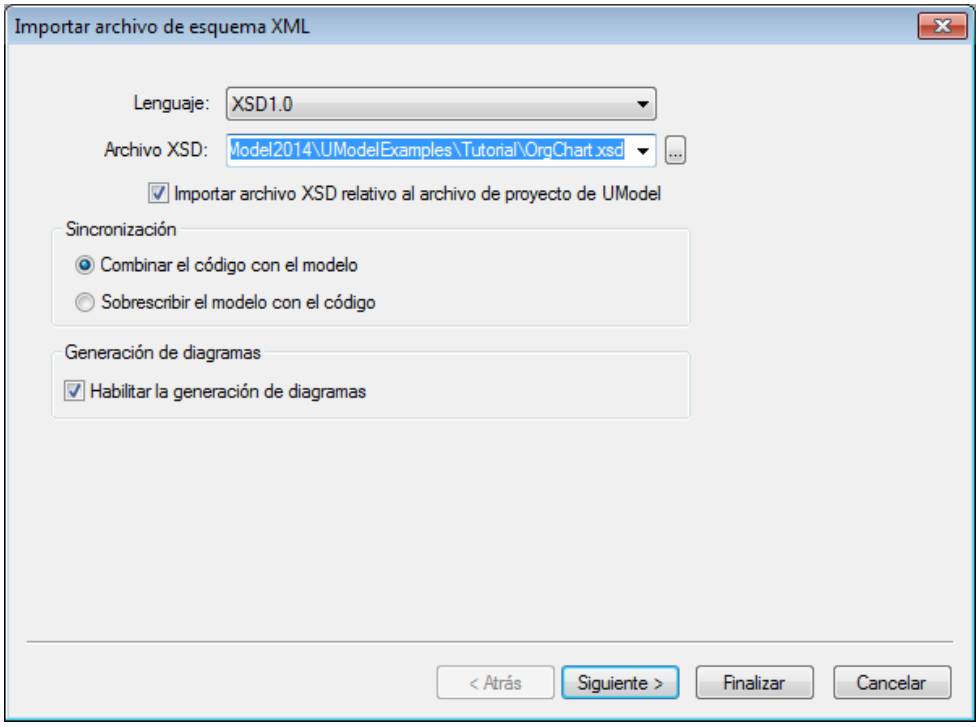
Nota: en UModel no se pueden importar esquemas XML no válidos. Los esquemas XML no se validan durante la importación ni durante su creación en UModel. Los esquemas XML tampoco se revisan durante el proceso de revisión sintáctica. Lo que sí se comprueba durante la importación es si el formato XML del esquema XML es correcto.

Importar esquemas XML

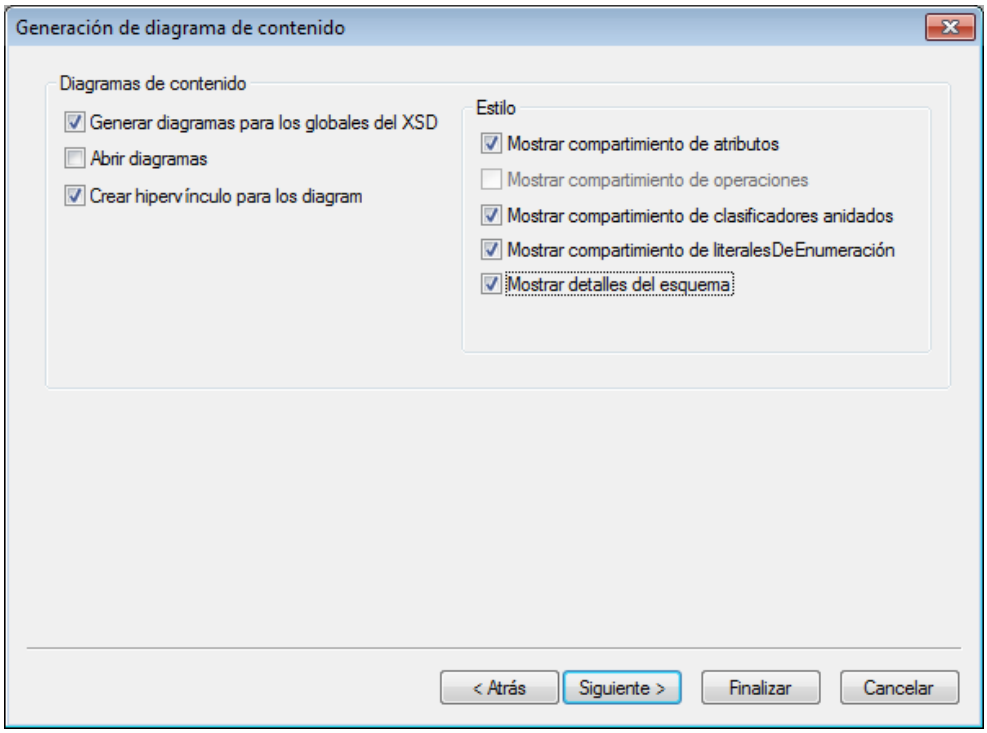
Para importar un solo esquema XML:

1. Seleccione el comando de menú **Proyecto | Importar archivo de esquema XML**.

 © 2014 Altova GmbH



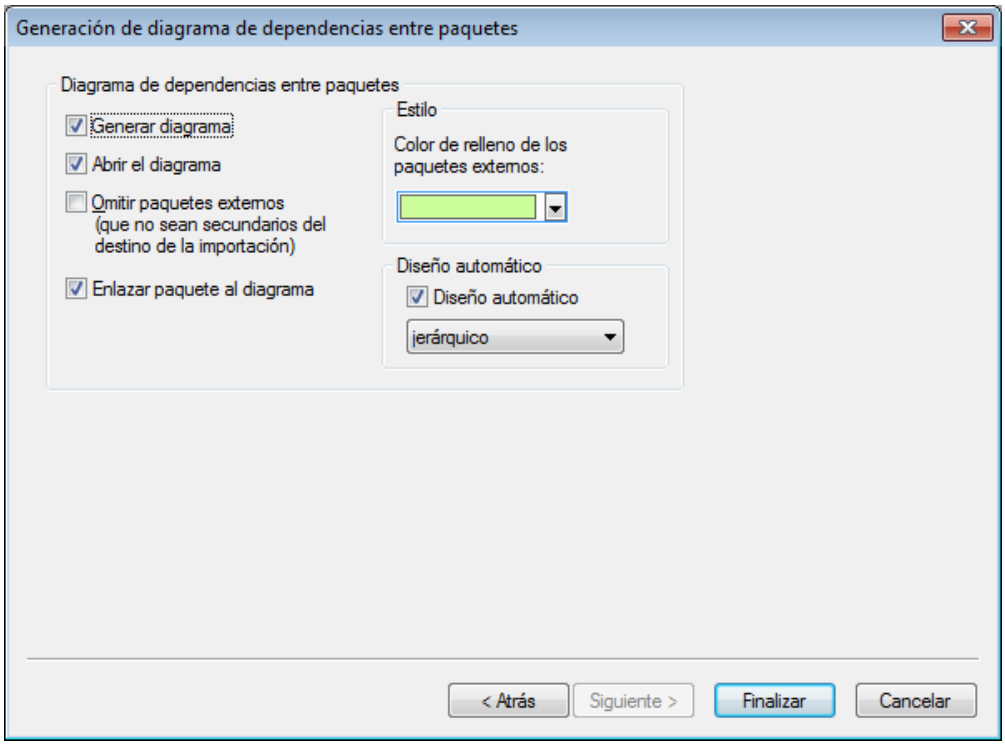
2. Asegúrese de que la casilla *Habilitar la generación de diagramas* esté activa. Haga clic en **Siguiente** para continuar.



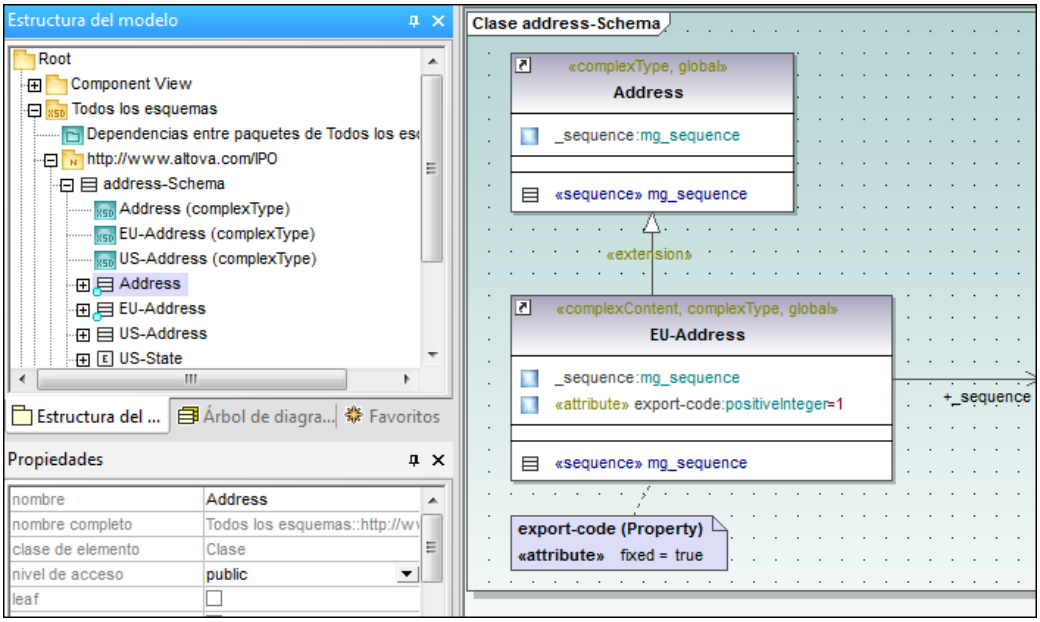
3. En la siguiente pantalla, elija las opciones en el grupo *Diagramas de contenido*. La primera opción crea un diagrama distinto por cada elemento global del esquema.
4. En el grupo de opciones *Estilo* seleccione qué compartimientos deben aparecer en los diagramas de clases. Si marca la casilla *Mostrar detalles del esquema como valores*

etiquetados, los detalles del esquema aparecerán en el elemento de modelado `ValorEtiquetado`.

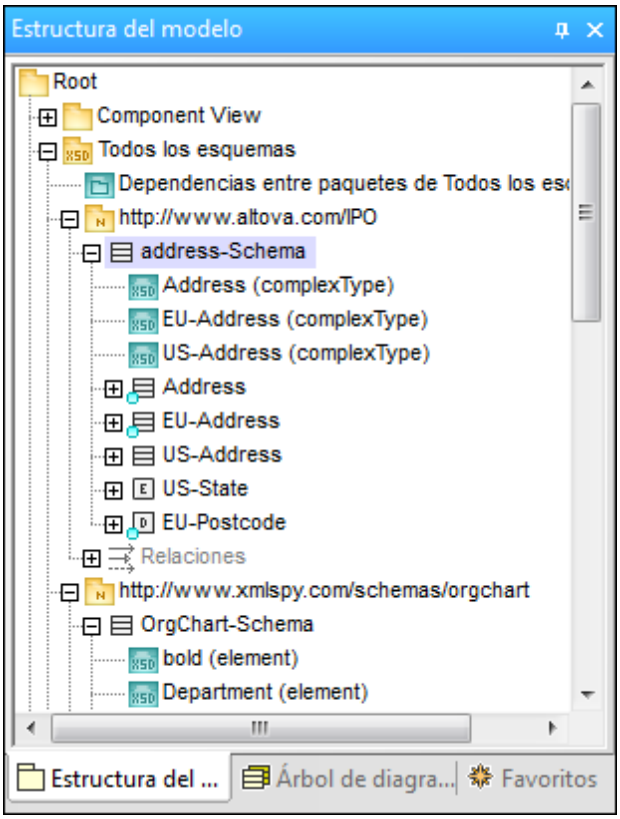
5. Haga clic en **Siguiente** para configurar la generación de diagramas de dependencias entre paquetes.



6. Haga clic en **Finalizar** para empezar a importar el esquema XML.
El esquema se importa en UModel y los diagramas generados se abren en el área de trabajo. Por ejemplo, en la imagen siguiente puede ver el contenido del diagrama **`EU-Address`** (`complexType`).

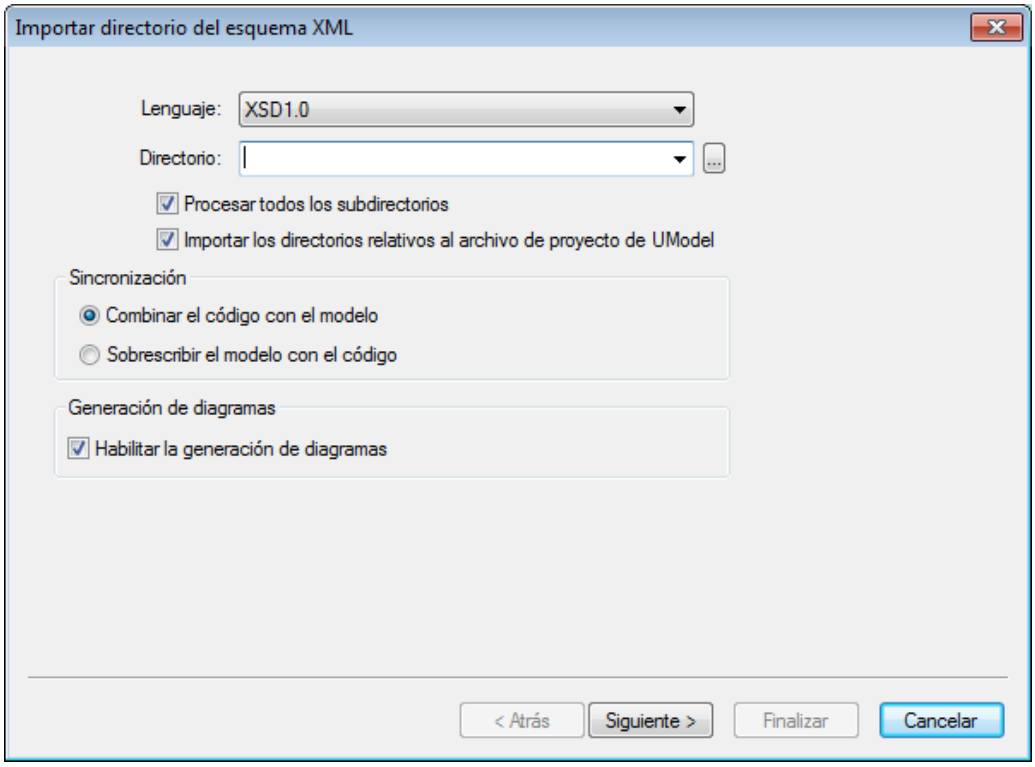


Nota: También se creó un paquete nuevo llamado `Todos los esquemas`, que se configuró como raíz del espacio de nombres XSD. Todos los elementos XSD globales generan un diagrama de esquema XML y los diagramas están bajo los paquetes de los correspondientes espacios de nombres.



Para importar varios esquemas XML:

1. Seleccione el comando de menú **Proyecto | Importar directorio del esquema XML**.



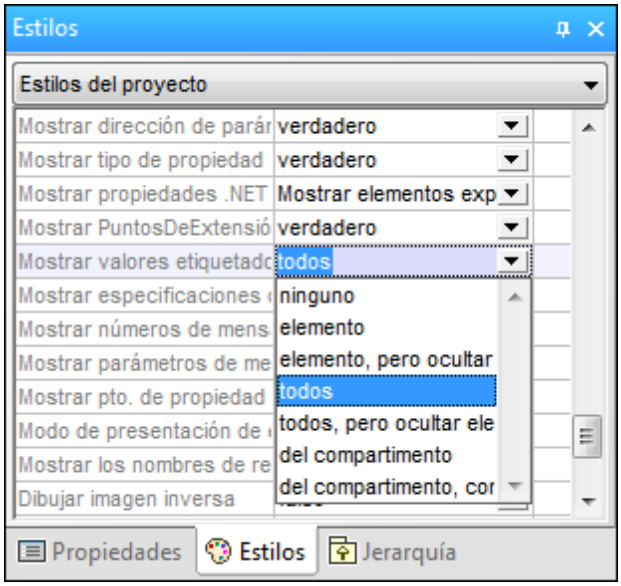
2. Marque la casilla *Procesar todos los subdirectorios* si quiere importar los esquemas de todos los subdirectorios.
   El resto del proceso de importación sigue el mismo orden que cuando se importa un solo esquema XML (ver más arriba).

   Nota: si un esquema XML incluye o importa otros esquemas, estos esquemas se importan automáticamente también.

Ver valores etiquetados: detalles del esquema

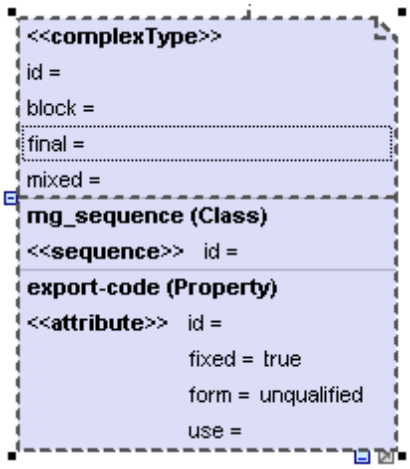
En UModel tiene la opción de configurar los detalles del esquema que se presentan como valores etiquetados en el elemento `ValorEtiquetado`. Esto se hace en el campo `Mostrar valores etiquetados` del panel Estilos o haciendo clic en el icono **Activar o desactivar el modo compacto** (situado en la esquina inferior derecha del elemento Valor etiquetado). Con este icono puede alternar entre los diferentes estados del valor etiquetado (p. ej. `todos`, `todos, pero ocultar elementos ocultos`, etc.)

 © 2014 Altova GmbH

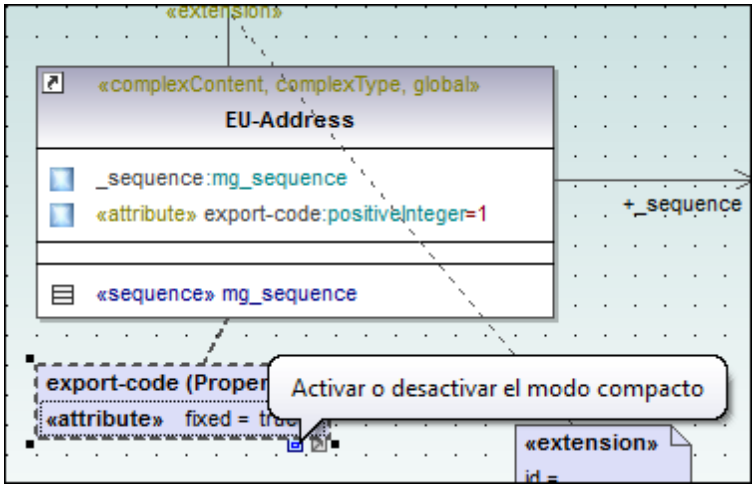


Nota: el estilo seleccionado para los valores etiquetados en la pestaña Estilos se utiliza para todo el proyecto. En cambio, si hace clic con el botón secundario en una clase y elige el comando **Valores etiquetados | Todos**, por ejemplo, esta opción solamente se aplica a la clase seleccionada.

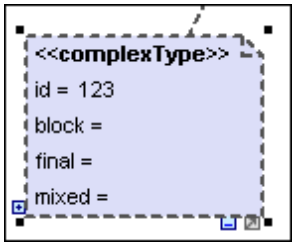
Mostrar valores etiquetados | todos
Esta opción muestra los valores etiquetados de la clase y de atributos propios y operaciones propias, etc.



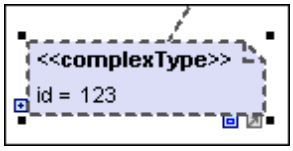
Mostrar valores etiquetados | Todos, pero ocultar elementos vacíos
Esta opción solamente muestra los valores etiquetados que tienen asignado un valor (p. ej. `fixed=true`).



Mostrar valores etiquetados | Todos
Esta opción muestra los valores etiquetados de la clase pero **no** los de los atributos y operaciones propios, etc.

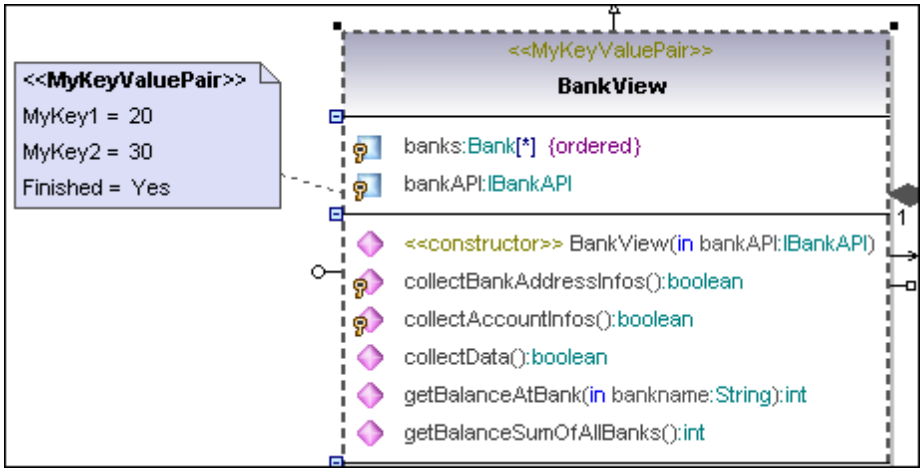


Mostrar valores etiquetados | Elemento, pero ocultar elementos vacíos
Esta opción solamente muestra los valores de elemento etiquetados de una clase, sin los atributos propios, que tengan un valor (p. ej. `id=123`).

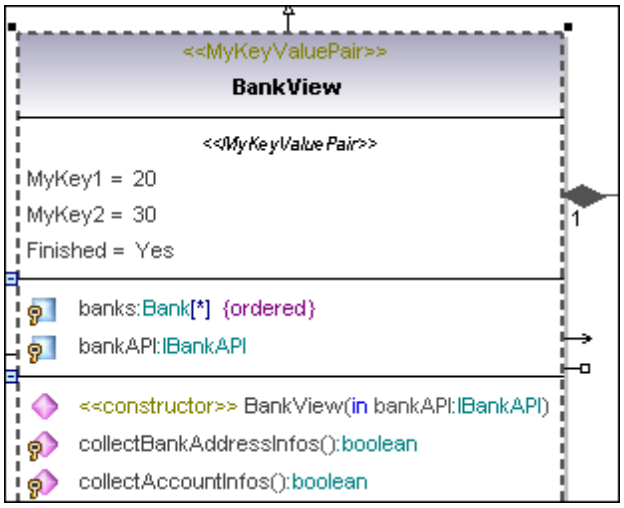


Mostrar valores etiquetados | en compartimiento

 © 2014 Altova GmbH



Esta opción muestra los valores etiquetados en un compartimiento distinto de la clase.



Mostrar valores etiquetados | En compartimiento, pero ocultar elementos vacíos
Esta opción solamente muestra los valores de elemento etiquetados de una clase, sin los atributos propios, que tengan un valor.

Anotación de XML Schema:
Cuando importe esquemas XML, tenga en cuenta que en la ventana Documentación solamente aparece la primera anotación de un elemento `complexType` o `simpleType`.

Insertar elementos

Usar iconos de la barra de herramientas:

1. Haga clic en un icono de la barra de herramientas Diagrama de esquema XML.
2. Haga clic en el área de trabajo del diagrama en el que desea insertar el elemento. Para insertar varios elementos del tipo seleccionado, mantenga pulsada la tecla **Ctrl** mientras hace clic en el área de trabajo.

Arrastrar elementos ya disponibles a un diagrama de esquema XML:
La mayoría de los elementos de un diagrama de esquema XML se pueden insertar en otro.

1. En la Estructura del modelo busque el elemento que quiere insertar en el otro diagrama (puede usar el cuadro de búsqueda o pulsar **Ctrl+F** para buscar el elemento).
2. Arrastre el elemento hasta el diagrama de esquema XML.

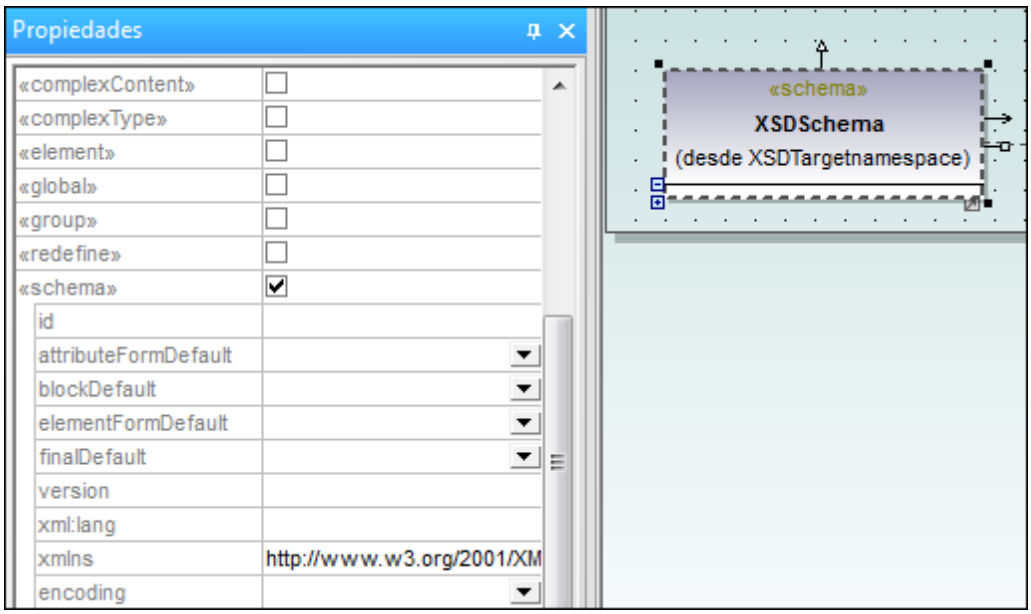
Nota: también puede usar los comandos **Copiar** y **Pegar sólo en el diagrama** para insertar elementos.

targetNamespace XSD

Inserta/define el espacio de nombres de destino para el esquema. Este espacio de nombres de destino debe pertenecer al paquete `Raíz de espacio de nombres XSD`.

schema XSD

Inserta/define un esquema XML. El esquema XML debe pertenecer a un paquete del espacio de nombres de destino XSD.



Element (global) XSD

Inserta un elemento global en el diagrama. Tenga en cuenta que también se genera automáticamente una propiedad en el compartimiento de atributos.

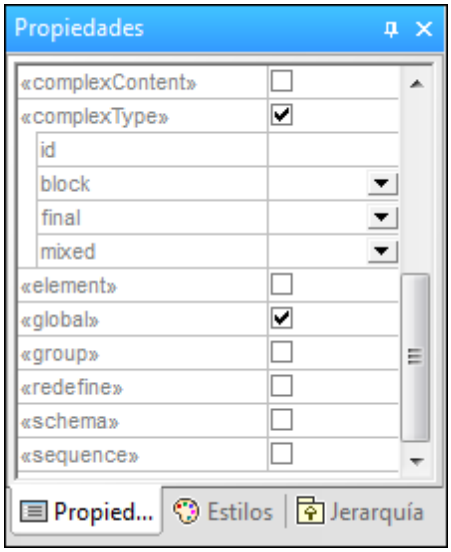
Para definir el tipo de datos de la propiedad:
1. Haga doble clic en la propiedad y ponga el cursor al final de la línea.
2. Inserte dos puntos (:) y seleccione el tipo de datos en el cuadro de diálogo emergente (p. ej. `string`).

 © 2014 Altova GmbH

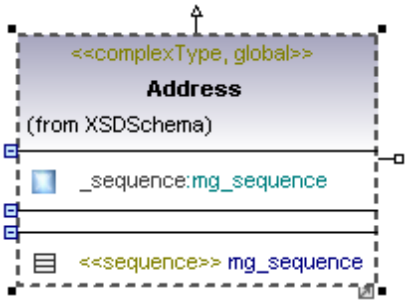
Para crear un modelo de contenido formado por un complexType con elementos obligatorios:

Esto supone insertar un elemento complexType, un elemento `sequence` y tres elementos.

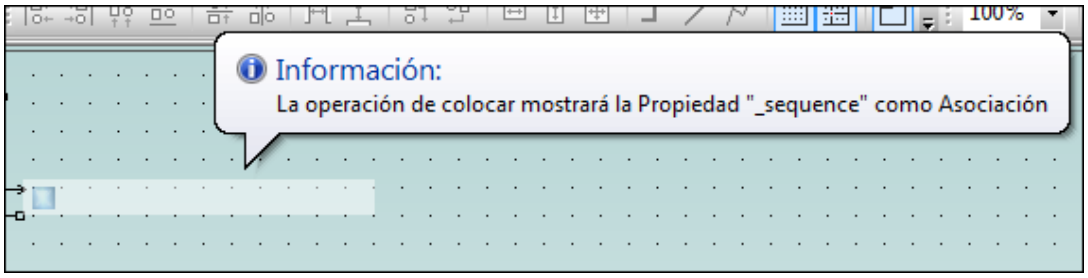
1. Haga clic en el icono **ComplexType XSD** y después haga clic en el diagrama para insertarlo.
2. Haga doble clic en el nombre y ponga el nombre nuevo **`Address`**.



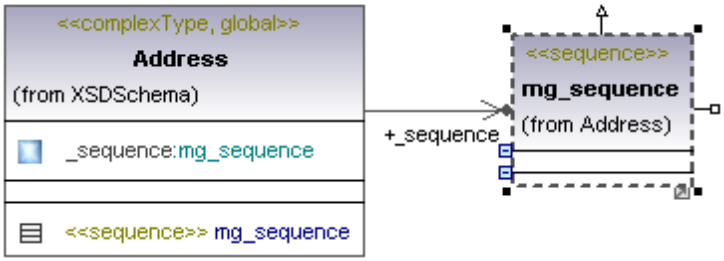
3. Haga clic con el botón secundario en **`Address`** y elija **Nuevo/a | XSD Sequence**.



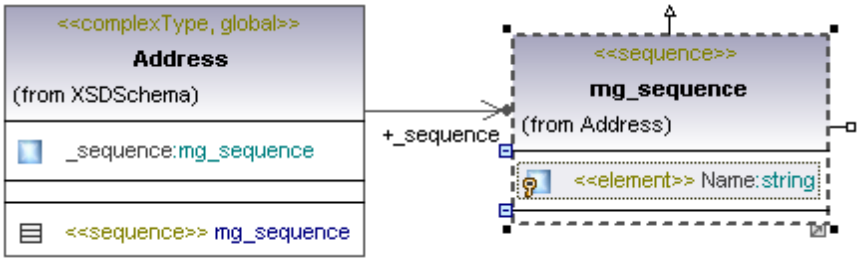
4. Haga clic en el atributo `_sequence:mg_sequence` del compartimiento de atributos y arrástrelo fuera del compartimiento.



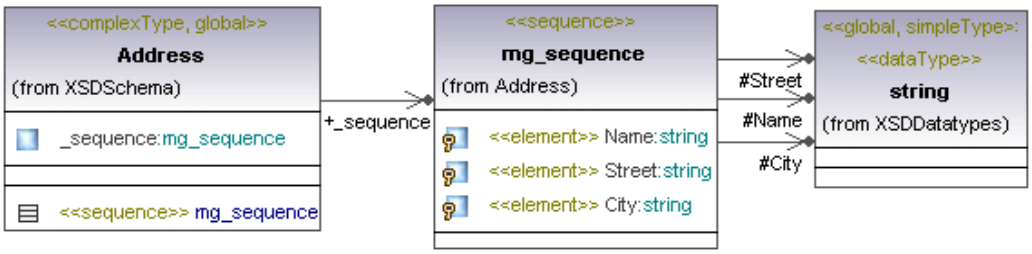
Esto crea una clase `sequence` o `compositor` al soltar el atributo.



5. Haga clic con el botón secundario en la clase `sequence` y elija **Nuevo/a | XSD Element (local)** en el menú contextual.
   Esto añade una propiedad nueva.
6. Haga doble clic en la propiedad, inserte el nombre del elemento (p. ej. `Name`), añada dos puntos (:) e inserte el tipo de datos `string`.



7. Haga lo mismo con otros dos elementos y llámelos `Street` y `City`.
8. Haga clic en la propiedad `Name` y arrástrela hasta el diagrama.



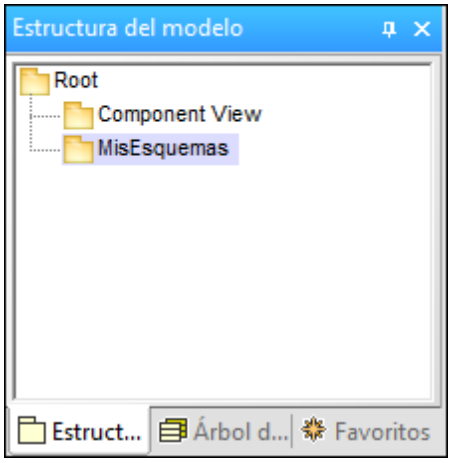
Crear y generar un esquema XML

Por lo general, el esquema se importa y se edita en UModel y después se generan los cambios. Sin embargo, en UModel también puede generar un esquema desde cero. A continuación describimos el proceso de forma muy general.

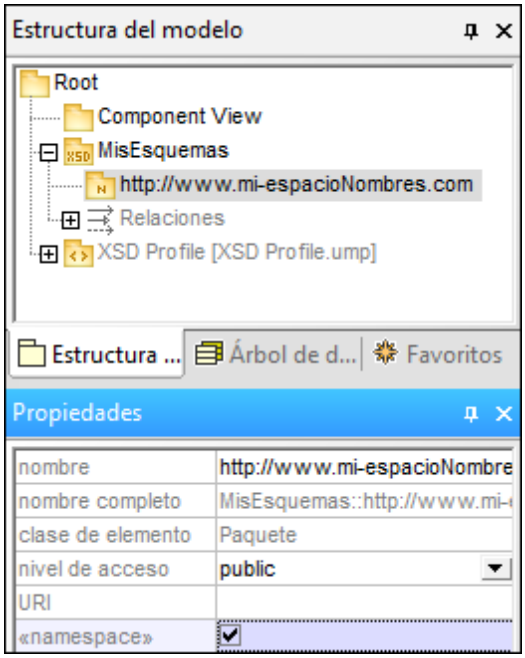
Para crear un esquema nuevo en UModel:

1. En la Estructura del modelo cree un paquete nuevo llamado `MisEsquemas`.

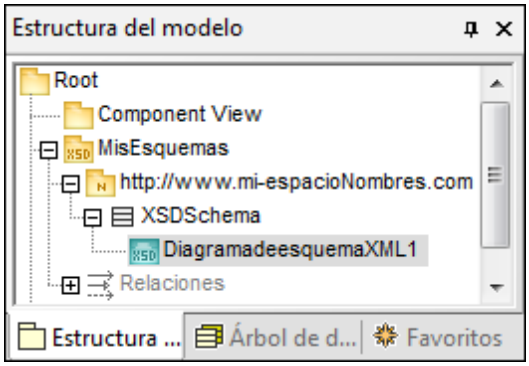
 © 2014 Altova GmbH



2. Haga clic con el botón secundario en el paquete nuevo y seleccione la opción **Ingeniería de código | Establecer como raíz de espacio de nombres XSD**. Aparece un aviso preguntando si desea asignar el perfil XSD si esta es la primera raíz de espacio de nombres XSD del proyecto.
3. Haga clic en **Aceptar** para asignar el perfil.
4. Haga clic con el botón secundario en el paquete nuevo y seleccione la opción **Elemento nuevo | Paquete**.
5. En el panel Propiedades haga doble clic en el campo `nombre` del paquete e inserte el espacio de nombres que desea usar (p. ej. `http://www.mi-espacioNombres.com`).
6. Marque la casilla `<<namespace>>` para definir este paquete como espacio de nombres de destino.



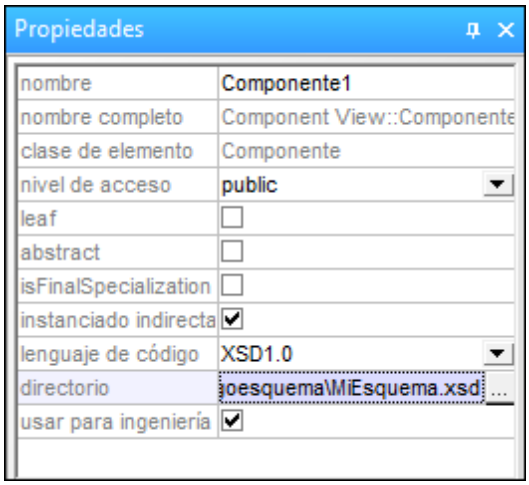
7. Haga clic con el botón secundario en el paquete de espacio de nombres y seleccione la opción **Diagrama nuevo | Diagrama de esquema XML**.
   Aparece un aviso preguntando si desea añadir el diagrama en un esquema XML nuevo.
8. Haga clic en **Sí**.



Ahora puede crear su esquema con ayuda de los iconos de la barra de herramientas Diagrama de esquema XML.

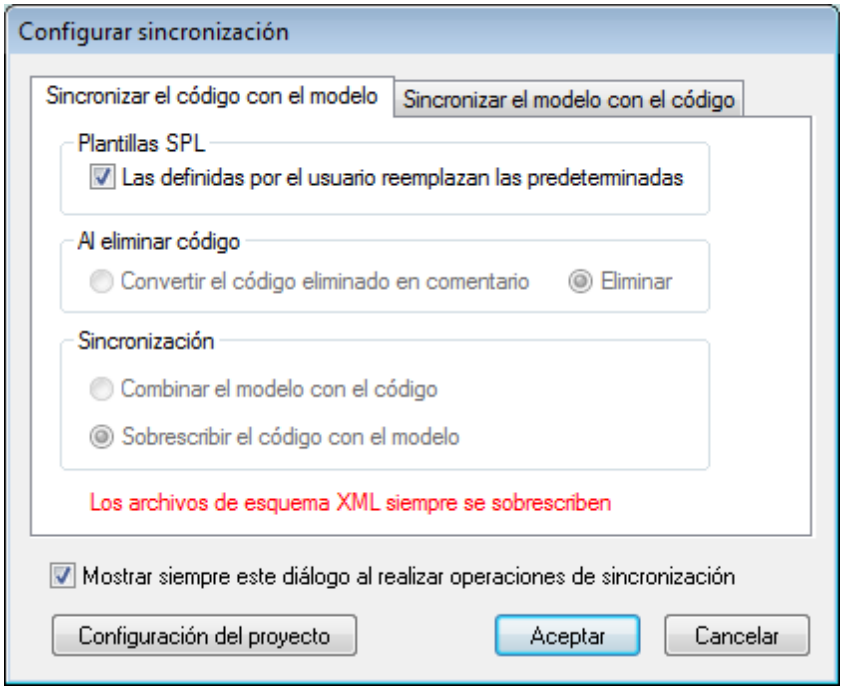
Para generar el esquema XML:

1. Arrastre el elemento `XSDSchema` hasta un componente para crear una `RealizaciónDeComponente`.
2. Defina el lenguaje de código XSD 1.0 para el componente y escriba la ruta de acceso donde se debe guardar el esquema generado (p. ej. `C:\códigoesquema\MiEsquema.xsd`).



3. Ahora seleccione el comando de menú **Proyecto | Sobrescribir el código de programa con el proyecto de UModel**. En el cuadro de diálogo haga clic en **Aceptar** para generar el esquema.

 © 2014 Altova GmbH


Configurar sincronización
Sincronizar el código con el modelo
Sincronizar el modelo con el código
Plantillas SPL
Las definidas por el usuario reemplazan las predeterminadas
Al eliminar código
Convertir el código eliminado en comentario
Eliminar
Sincronización
Combinar el modelo con el código
Sobrescribir el código con el modelo
Los archivos de esquema XML siempre se sobrescriben
Mostrar siempre este diálogo al realizar operaciones de sincronización
Configuración del proyecto
Aceptar
Cancelar

10.3.2 Diagramas BPMN 1.0 / 2.0

Sitio web de Altova: Notación de modelado de procesos de negocio (BPMN)

BPMN es una notación estándar de diagramas de flujo que muestran procesos de negocio como flujos de trabajo de fácil comprensión para todos los participantes del proceso. UModel es compatible con la versión 1.0 y 2.0 de BPMN.

UModel también es compatible con los diagramas BPMN 2.0 de coreografía, de colaboración y con los diagramas de procesos de negocio estándar BPMN 2.0.

En un proyecto de UModel puede usar diagramas BPMN 1.0 y diagramas BPMN 2.0. Además puede convertir diagramas BPMN 1.0 en diagramas BPMN 2.0 en cualquier momento del proceso de modelado.

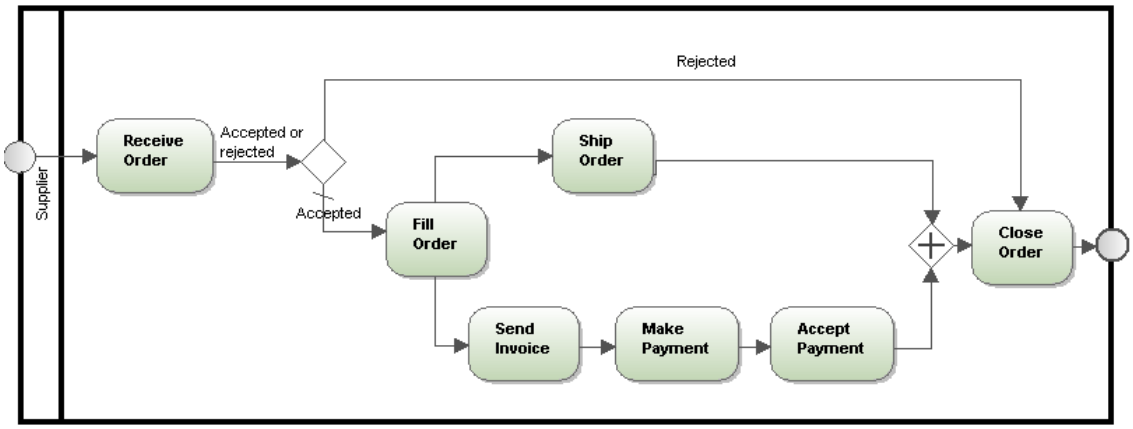
Hay cuatro categorías de elementos BPMN:

Objetos de flujo	Eventos, Actividades (Tareas o Subprocesos) y Compuertas
Objetos de conexión	Flujos de secuencia, Flujos de mensaje y Asociación
Compartimentos	Contenedores y Compartimentos
Artefactos	Objetos de datos, Grupos y Anotaciones textuales

En UModel los diagramas y objetos BPMN se insertan de la misma forma que los demás elementos de modelado.

Los objetos se pueden insertar con ayuda de los iconos de la barra de herramientas. Las asociaciones con otros objetos se pueden crear arrastrando los controladores de objeto hasta el objeto de destino. Al igual que en los demás diagramas, los elementos BPMN se pueden ver y configurar en el panel Propiedades.

Recuerde que puede usar la pestaña *Capas* para crear varias capas en cada diagrama BPMN. Para más información consulte el apartado Capas.



Para convertir diagramas BPMN 1.0 en diagramas BPMN 2.0:

1. Haga clic con el botón secundario en un diagrama BPMN 1.0 y seleccione la opción **Convertir en diagrama BPMN 2**.

 © 2014 Altova GmbH

2. Si en el mismo paquete hay varios diagramas BPMN 1.0, aparece un aviso preguntando si desea convertir todos los diagramas del paquete.
3. Después aparece otro aviso preguntando si desea incluir el perfil BPMN 2 en el proyecto.
4. Haga clic en **Aceptar** para convertir los diagramas.

Objetos de flujo

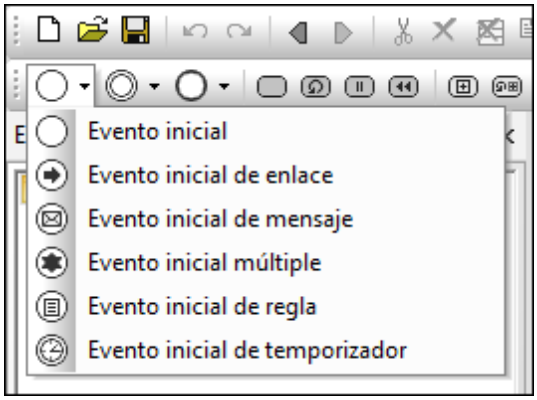
Los objetos de flujo son elementos gráficos que definen el comportamiento de un proceso de negocios. Los hay de tres tipos: eventos, actividades y compuertas.

Eventos
Un evento es algo que ocurre durante un proceso de negocio y se representa con un círculo. Los eventos afectan al flujo del proceso y por lo general tienen una causa (desencadenador) y un resultado. Hay tres tipos de eventos: inicial, intermedio y final. Cada tipo de evento tiene un menú desplegable en la barra de herramientas.

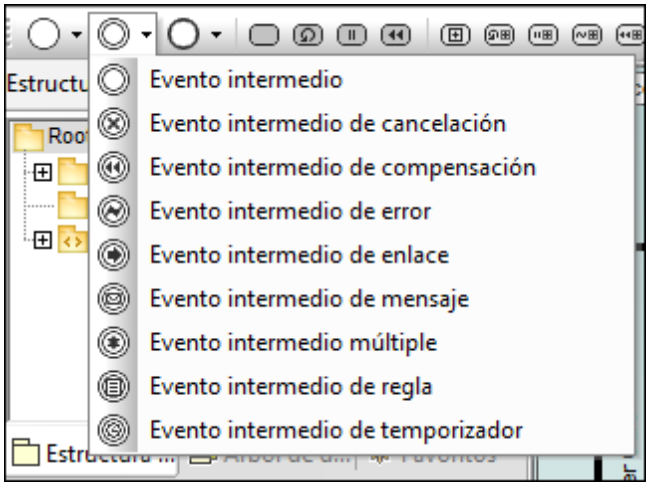
Para insertar un evento:

1. Haga clic en el menú desplegable para ver la lista de eventos del tipo que desea insertar.
2. Seleccione el evento que desea insertar y haga clic en el área de trabajo del diagrama para insertarlo.

Eventos iniciales

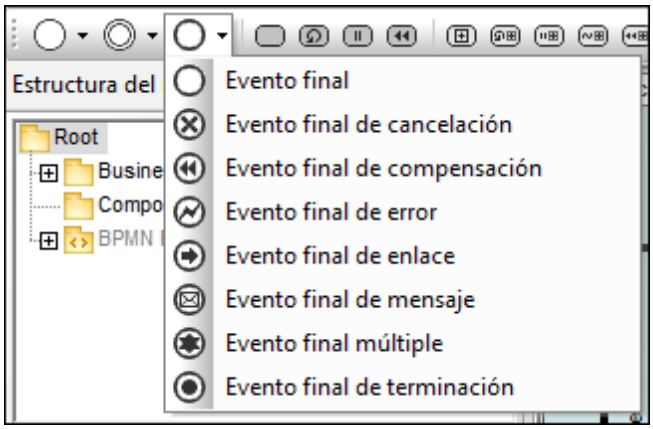


Eventos intermedios



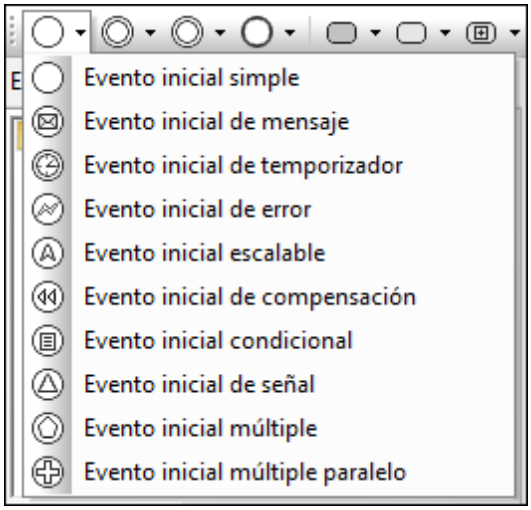
Los eventos intermedios se pueden anexar al contorno de una tarea o subproceso e indican que la actividad debe interrumpirse cuando el evento se desencadena.

Eventos finales



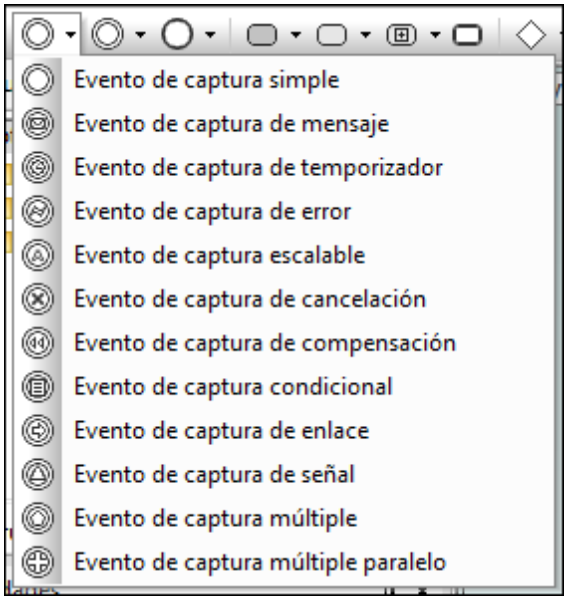
Eventos BPMN 2.0:

Eventos iniciales

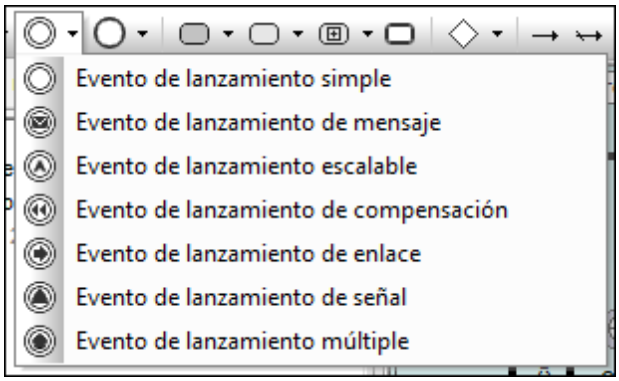


Eventos de captura

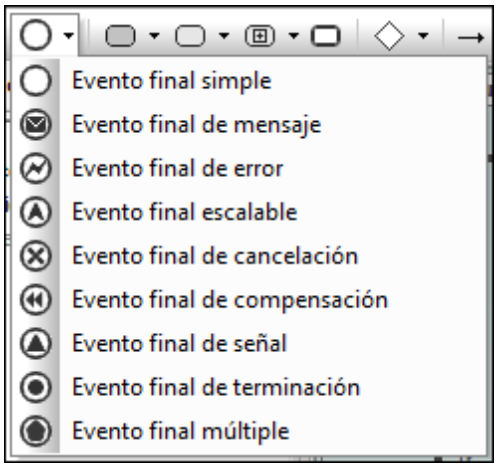
 © 2014 Altova GmbH



Eventos de lanzamiento



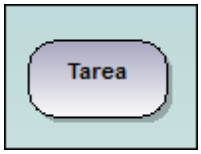
Eventos finales



Actividades

Las actividades son acciones que se desarrollan durante un proceso de negocio y se representan

con rectángulos redondeados. Las hay de tres tipos: procesos, subprocesos y tareas. Las actividades pueden ocurrir una sola vez o varias veces en un ciclo.



Para insertar una actividad:

1. Haga clic en el icono de la tarea o subproceso en la barra de herramientas.
2. Haga clic en el área de trabajo del diagrama para insertar la actividad.

Tareas

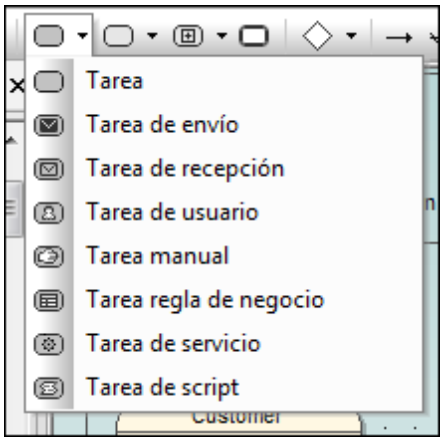
Las tareas son actividades que están incluidas en un proceso y no se pueden desglosar en subtareas porque son atómicas.

Tarea **(Ciclo)**

Tarea **(instancia múltiple)**

Tarea **(compensación)**

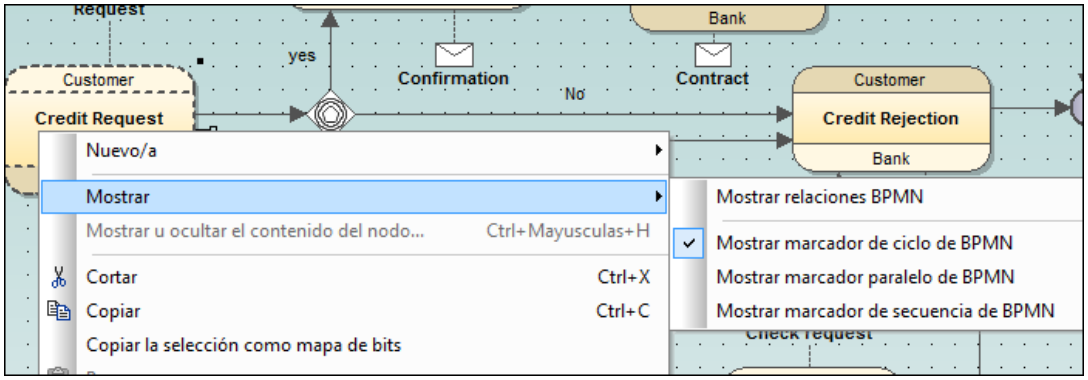
Tareas BPMN 2.0



Para definir un marcador de ciclo, paralelo, de secuencia o de compensación:

1. Haga clic con el botón secundario en la tarea insertada y seleccione un tipo de marcador (p. ej. **Mostrar | Mostrar marcador de ciclo de BPMN**).

 © 2014 Altova GmbH



Nota: también puede definir el marcador en el panel Propiedades (en la entrada `MultiInstanceLoopCharacteristics`).

Subprocesos
Un subproceso es una actividad compuesta incluida en un proceso, que permite desarrollar el modelo del proceso de negocio de forma jerárquica. Un subproceso puede desglosarse en varias actividades subordinadas.

Un subproceso **contraído** se representa como elemento de nivel superior donde los datos del subproceso no son visibles. En el elemento aparece también el icono +, lo cual indica que existe una capa más de complejidad.

Un subproceso **expandido** muestra los datos del subproceso dentro de sus límites. Recuerde que un flujo de secuencia no puede cruzar el límite de un subproceso.

Compuertas
Las compuertas sirven para determinar cómo se ramifican y cómo convergen los flujos de secuencia dentro de un proceso. Las compuertas se representan con un icono en forma de rombo.

Compuerta **inclusiva** (OR)

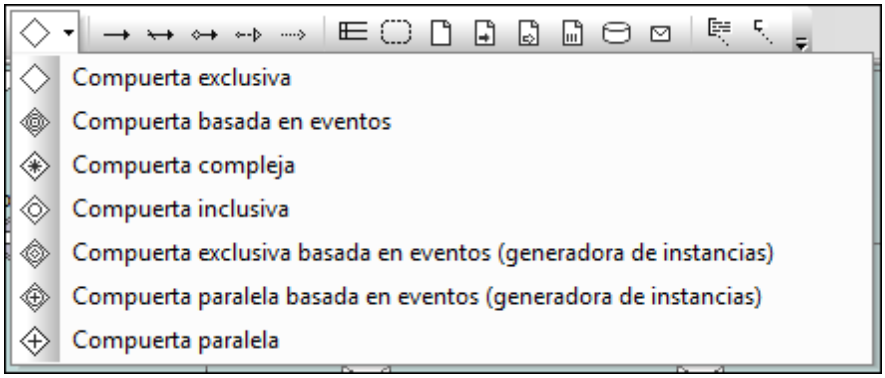
Compuerta **paralela** (AND)

Compuerta **exclusiva basada en datos** (XOR)

Compuerta **exclusiva basada en eventos** (XOR)

Compuerta **compleja** (bifurcación/convergencia)

Compuertas BPMN 2.0:



Subprocesos expandidos

Los subprocesos expandidos muestran los detalles del proceso dentro de los límites del elemento.

Subproceso expandido

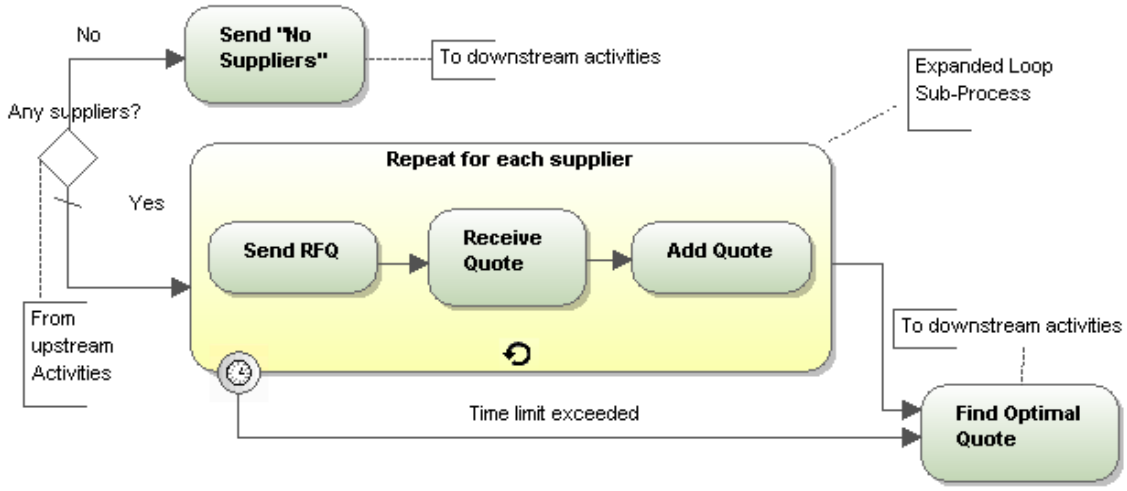
Subproceso de **ciclo** expandido

Subproceso de **instancia múltiple** expandido

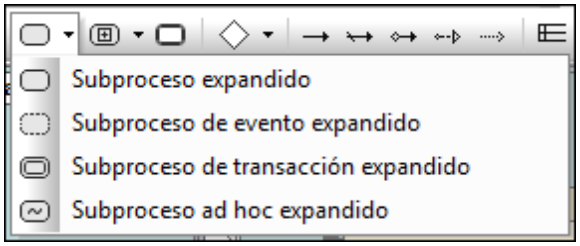
Subproceso **ad hoc** expandido

Subproceso de **compensación** expandido

 © 2014 Altova GmbH

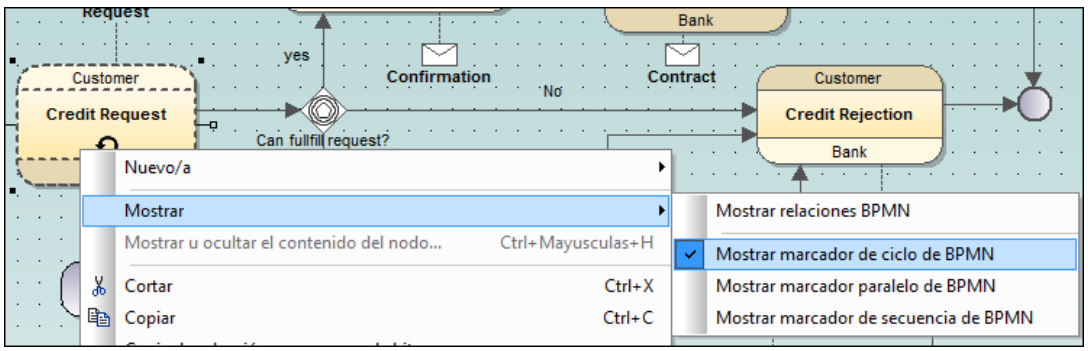


Subprocesos expandidos BPMN 2.0



Para definir un marcador de ciclo, paralelo, de secuencia o de compensación:

1. Haga clic con el botón secundario en la tarea insertada y seleccione un tipo de marcador (p. ej. **Mostrar | Mostrar marcador de ciclo de BPMN**).



Subprocesos contraídos

Los subprocesos contraídos ocultan los detalles del proceso. Cada tipo de subproceso contraído se representa con un icono diferente dentro del elemento `Subproceso`.

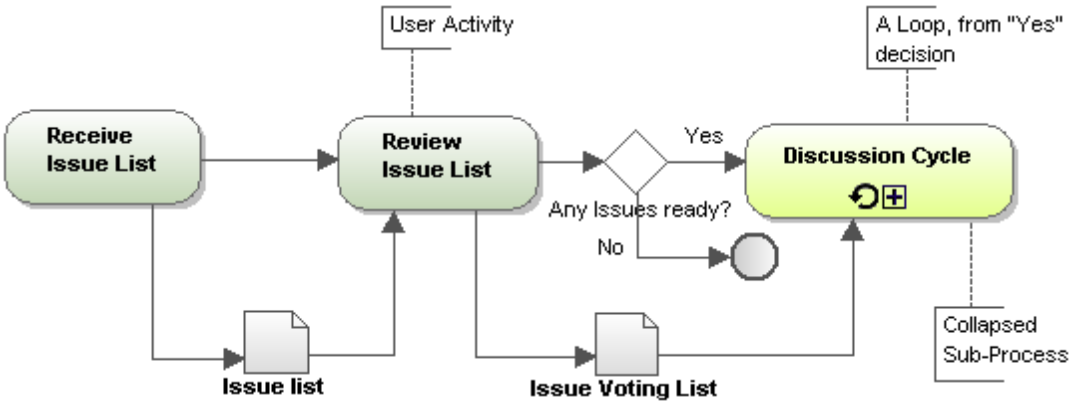
Subproceso contraído

Subproceso de **ciclo** contraído

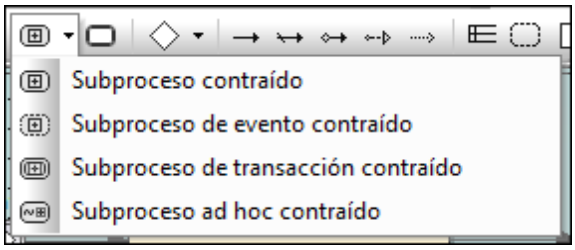
Subproceso de **instancia múltiple** contraído

Subproceso **ad hoc** contraído

Subproceso de **compensación** contraído



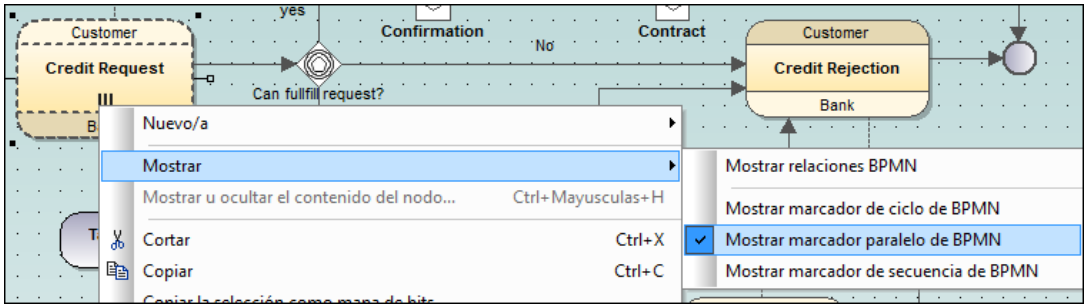
Subprocesos contraídos BPMN 2.0:



Para definir un marcador de ciclo, paralelo, de secuencia o de compensación:

1. Haga clic con el botón secundario en la tarea insertada y seleccione un tipo de marcador (p. ej. **Mostrar | Mostrar marcador paralelo de BPMN**).

 © 2014 Altova GmbH

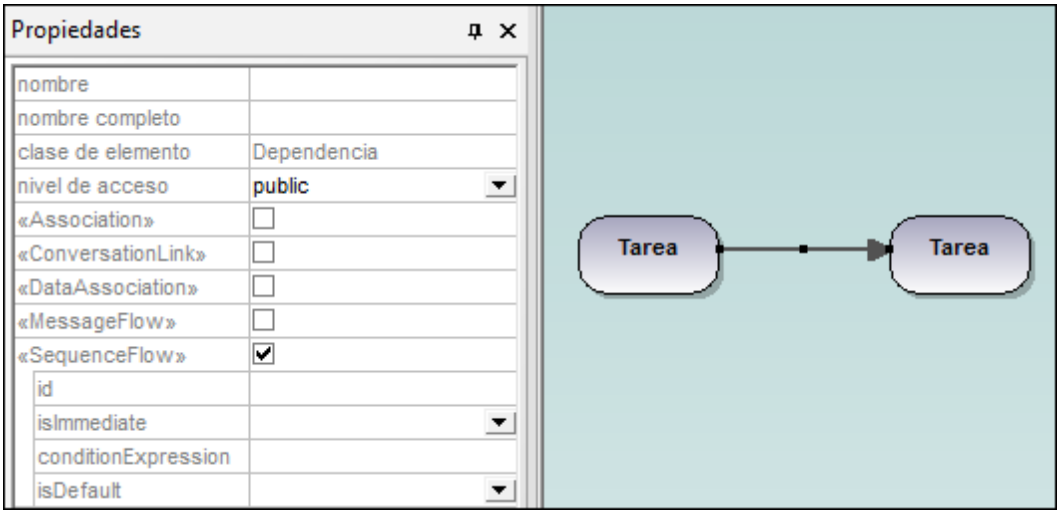


Objetos de conexión

Hay dos maneras de conectar objetos en BPMN: con flujos (usando una secuencia o un mensaje) y con asociaciones.

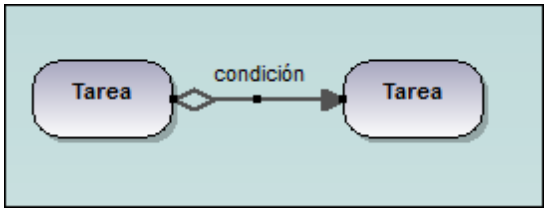
Flujo de secuencia

Un flujo de secuencia muestra en qué orden se llevan a cabo las actividades dentro de un proceso.



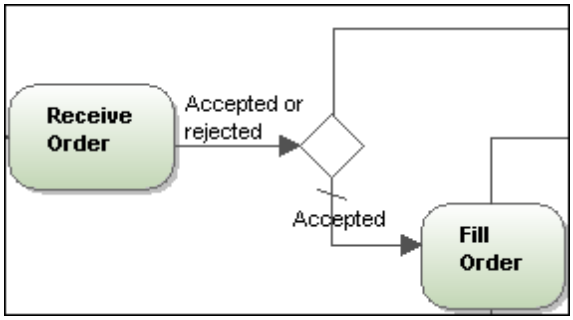
Flujo condicional

Este tipo de flujo puede tener una expresión condicional que se evalúa para determinar si el flujo se utiliza o no. Si el flujo condicional tiene su origen en una actividad, al principio de la flecha aparece un pequeño **rombo**.

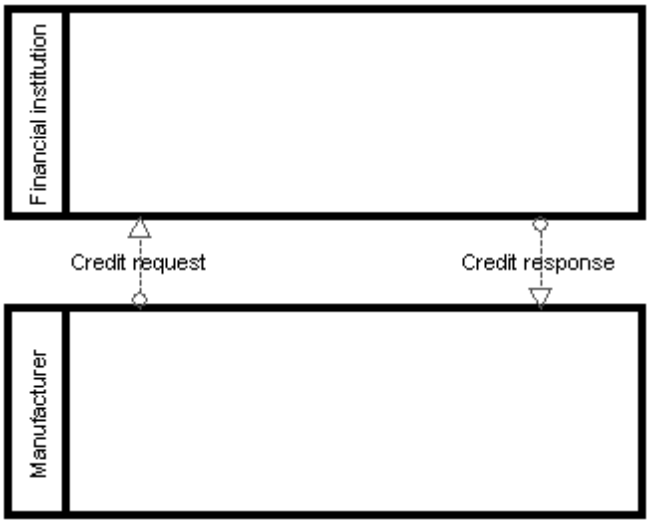


Flujo por defecto

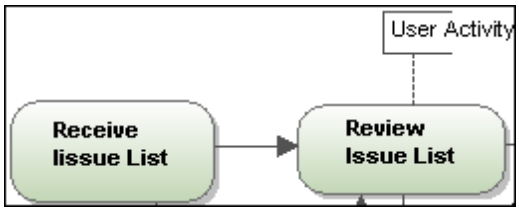
Este tipo de flujo se utiliza si todos los demás flujos condicionales dan `false` como resultado en decisiones inclusivas o exclusivas. El flujo por defecto se señala con una **barra diagonal** al principio de la flecha del flujo.



Flujo de mensaje
Un flujo de mensaje muestra el flujo de mensajes que tiene lugar entre dos participantes (entidades o roles), que pueden enviarlos y recibirlos. Cada participante se representa en un contenedor.



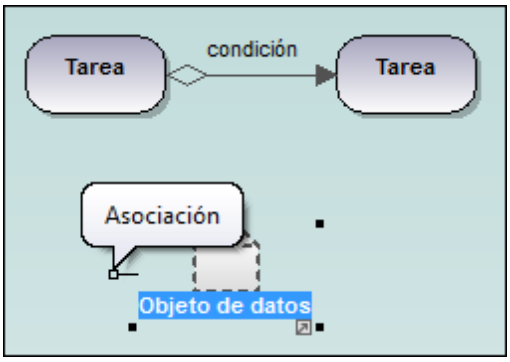
Asociaciones
Las asociaciones sirven para asociar texto o datos de objetos con objetos de flujo e indican cómo entran y salen datos de las actividades. El diagrama de la imagen siguiente, por ejemplo, muestra una anotación textual que ofrece información adicional (**User Activity**) para la tarea **Review Issue List**.



Para crear una asociación entre un objeto de datos y un control de flujo:

1. Haga clic en el controlador `Asociación` del objeto de datos (en el borde izquierdo del objeto).
2. Arrastre el conector hasta la flecha de control de flujo (que se resalta para indicar que puede soltar el conector).

 © 2014 Altova GmbH



Otra opción es hacer clic en el icono **Asociación** de la barra de herramientas, hacer clic en el objeto de datos y arrastrar el puntero del mouse hasta el control de flujo.

Contenedores y compartimentos

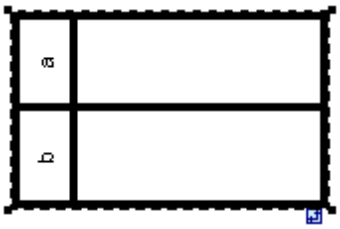
Contenedores
Los contenedores sirven para dividir y organizar las actividades. Un proceso de negocio puede mostrar la interacción que existe entre varios procesos o participantes. Cada participante se representa en un rectángulo que recibe el nombre de *contenedor.* Un participante puede ser un rol o una entidad.

- Puede arrastrar objetos BPMN hasta un contenedor y colocarlos dentro cuando el contorno del contenedor se resalte.
- Puede seleccionar los objetos que están dentro del contenedor usando las teclas **Ctrl +clic** o arrastrando el puntero para seleccionar los objetos.
- Para mover el contenedor a otra posición, haga clic en su contorno o título y arrástrelo hasta la posición nueva.

Compartimento
Los contenedores pueden dividirse en compartimentos, que sirven para organizar las actividades del contenedor por categorías. Los compartimentos pueden ser horizontales o verticales.

Para añadir un compartimento en un contenedor:

1. Haga clic con el botón secundario en el título del contenedor y elija **Nuevo/a | Compartimento**.
   Esto añade un compartimento nuevo en el contenedor. Cada compartimento puede tener un nombre distinto (haga doble clic en el campo `nombre` y escriba el nombre del compartimento).



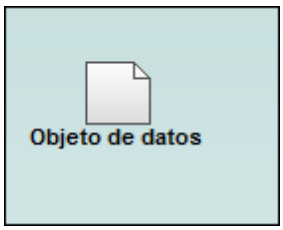
Nota: haga clic con el botón secundario en uno de los compartimentos y elija **Nuevo/a | <elemento>** para añadir elementos al contenedor.

Artefactos

Los artefactos sirven para mostrar información sobre un proceso, es decir, cómo se utilizan y actualizan los datos, los documentos y los demás objetos durante el proceso de negocio. Los artefactos no están relacionados directamente con el flujo de secuencia o de mensaje del proceso.

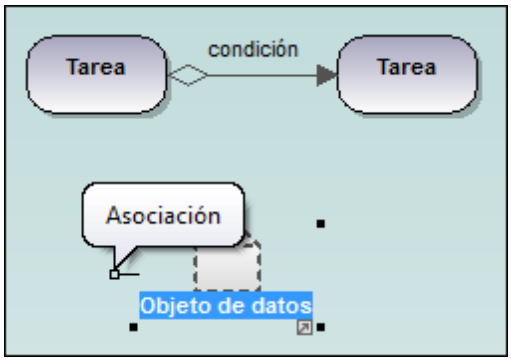
Objetos de datos

Los objetos de datos representan documentos o datos y su utilización durante el proceso de negocio. Puede usar los objetos de datos para definir cómo entran y salen datos de las actividades.



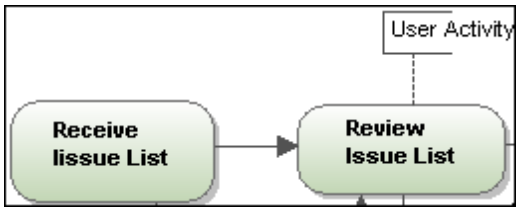
Para crear una asociación entre un objeto de datos y un control de flujo:

1. Haga clic en el controlador `Asociación` del objeto de datos (situado en el borde izquierdo del objeto).
2. Arrastre el conector hasta el control de flujo (que se resalta para indicar que puede soltar el conector).



Anotaciones textuales

Las anotaciones textuales sirven para anotar secciones del proceso de negocio y se conectan al objeto por medio de asociaciones.



 © 2014 Altova GmbH

Grupos
Los grupos se utilizan para resaltar ciertas secciones de un diagrama, incluso de contenedores distintos. Los grupos no pueden conectarse a los flujos de secuencia ni de mensaje. Suelen colocarse detrás de tareas o procesos.

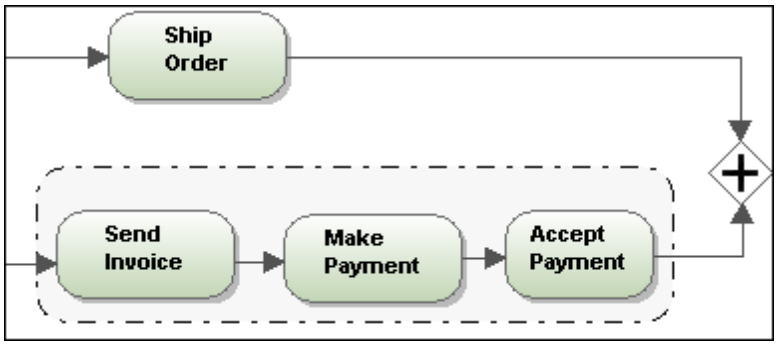
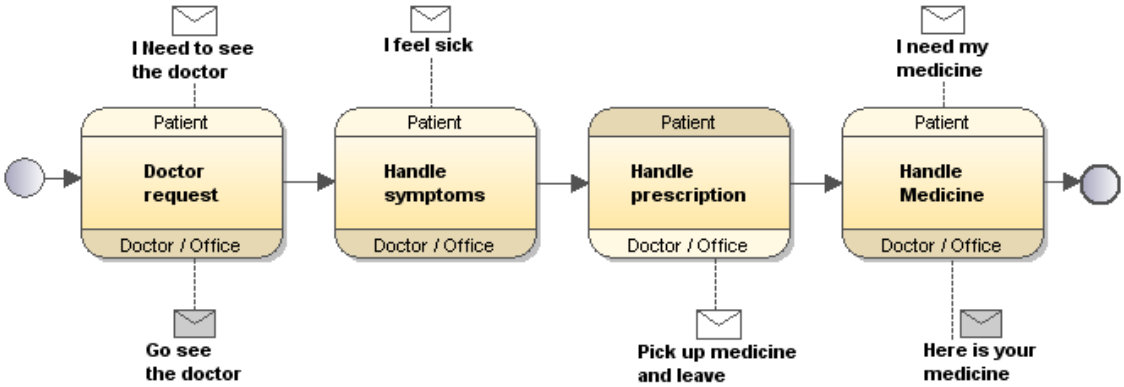


Diagrama de coreografía

Los diagramas de coreografía especifican cómo **coordinan** los participantes sus interacciones. También pueden entenderse como contratos entre los participantes, donde lo importante es el intercambio de la información (Mensajes).

Los contratos suelen tomar la forma de pedidos de compra que se envían a un proveedor, la confirmación del proveedor de que procesará el pedido y la ejecución del pedido. En las coreografías las actividades también se ordenan por medio de flujos de secuencia.

Las actividades están compuestas por interacciones que tienen lugar entre participantes. Las interacciones suelen denominarse *pauta de intercambio de mensajes* (o MEP por sus siglas en inglés). Una pauta de intercambio de mensajes es la *actividad* de la coreografía y también recibe el nombre de **tarea de coreografía**.



Tareas de coreografía

Hay cuatro tipos de tareas de coreografía:

Tarea de coreografía

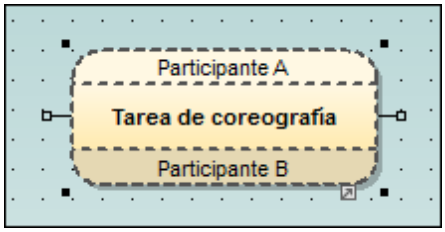
Sub-coreografía (contraída)

Sub-coreografía (expandida)

Coreografía de llamada

Para insertar una tarea de coreografía:

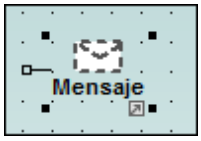
1. En la barra de herramientas haga clic en el icono del tipo de tarea que desea insertar (p. ej. **Tarea de coreografía**) y después haga clic en el área de trabajo del diagrama.



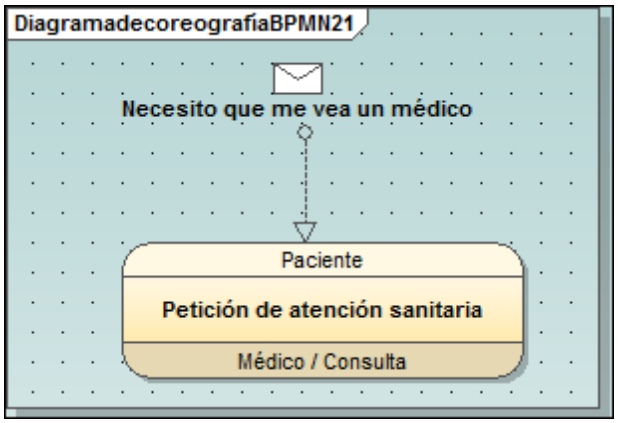
   Por ejemplo, en la imagen anterior puede ver la vista predeterminada de una tarea de coreografía. En ella se resalta automáticamente el texto predeterminado **`Tarea de coreografía`**.
2. Escriba el nombre nuevo de la tarea de coreografía.
3. Haga clic en la banda superior para escribir el nombre del **`Participante A`** y en la banda inferior para escribir el nombre del **`Participante B`**.
   Las bandas de los participantes pueden estar sombreadas y no sombreadas. El iniciador de la actividad es el participante no sombreado (en este caso el Participante A).

Para añadir/asociar mensajes a una tarea de coreografía:

1. Haga clic en el icono **Mensaje** de la barra de herramientas y después en el área de trabajo del diagrama.



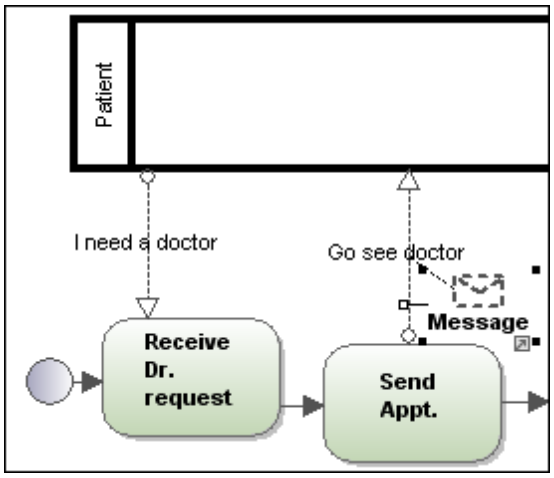
2. Escriba el nombre del mensaje (p. ej. `Necesito que me vea un médico`).
3. Haga clic en el controlador **`Asociación`** (del borde izquierdo) y arrástrelo hasta la tarea de coreografía.



 © 2014 Altova GmbH

Para añadir un mensaje a una línea (p. ej. de asociación):

1. Haga clic en la **línea** a la que dese añadir el mensaje.
2. Haga clic en el icono **Mensaje** de la barra de herramientas.
3. Haga clic otra vez en la línea.

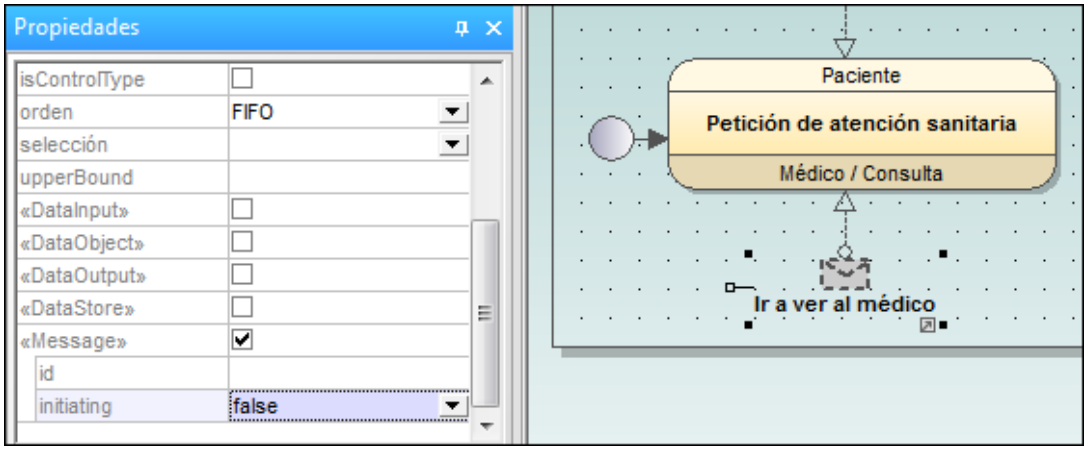


El mensaje se coloca encima de la línea y se anexa a la línea automáticamente.

Para cambiar el mensaje / participante de iniciación:

- Cuando se inserta un `Mensaje`, el mensaje se define automáticamente como mensaje de iniciación (es decir, no está sombreado).

1. Haga clic en el mensaje y seleccione el valor `false` en el cuadro combinado `initiating` del panel Propiedades.

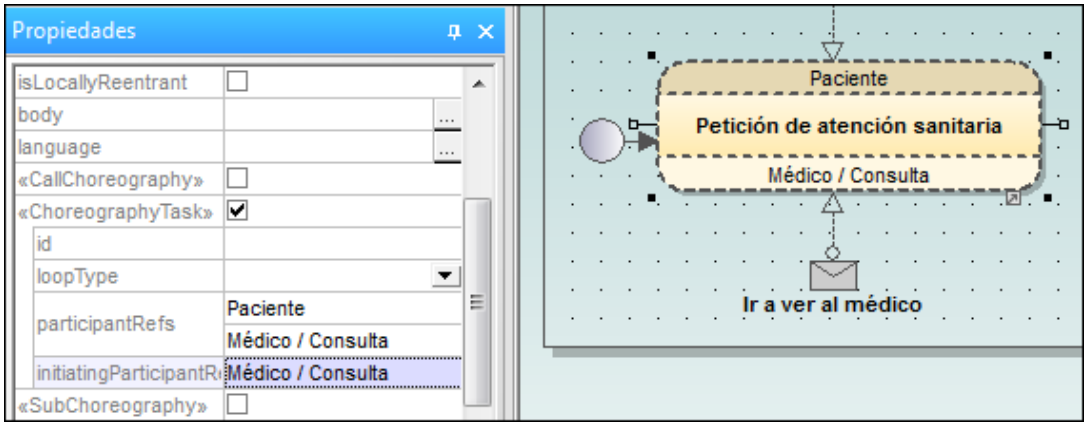


El mensaje aparece ahora sombreado.

- Cuando se inserta una tarea de coreografía, el **`Participante A`** se define automáticamente como participante iniciador.

1. Haga clic en la tarea de coreografía que contiene el participante que debe funcionar como iniciador.
2. Escriba el nombre del participante que debe ser el iniciador en el cuadro combinado

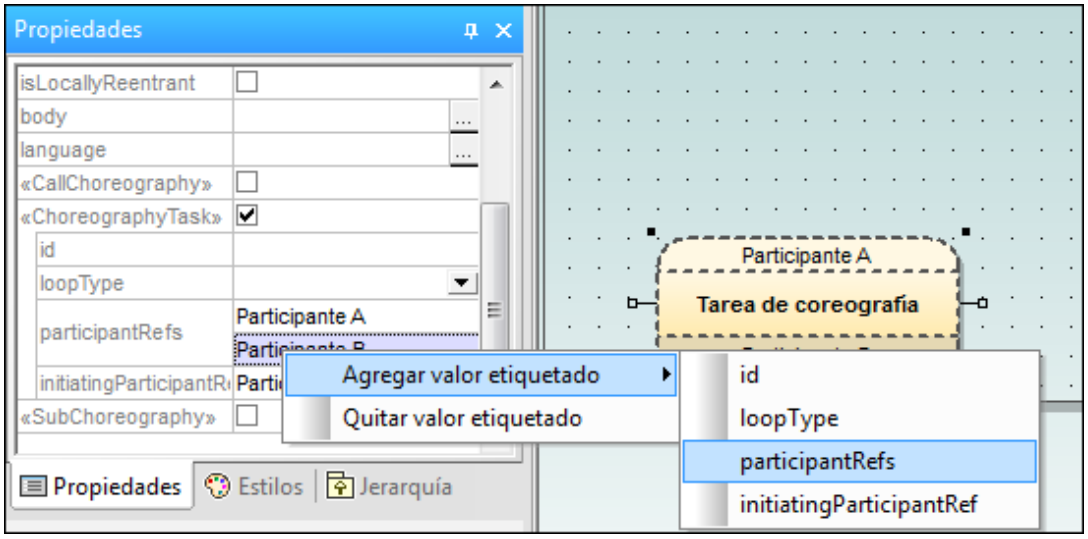
`InitiatingParticipantRef` del panel Propiedades (p. ej. `Médico / Consulta`).



La banda del participante **`Médico / Consulta`** ya no está sombreada, lo cual indica que se trata del participante iniciador. La banda del participante **`Paciente`** está sombreada.

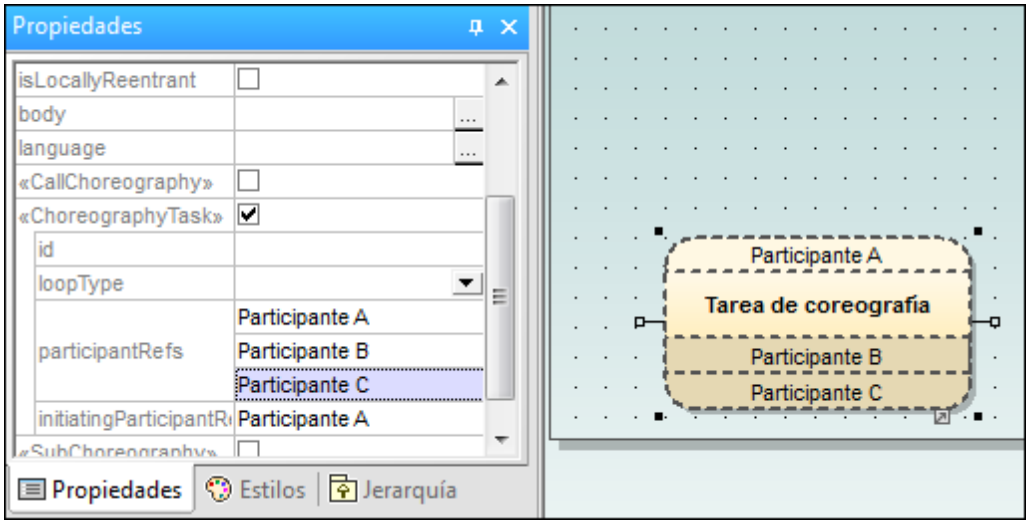
Para añadir participantes nuevos a una tarea de coreografía:

1. Haga clic en la tarea en la que desea añadir el participante.



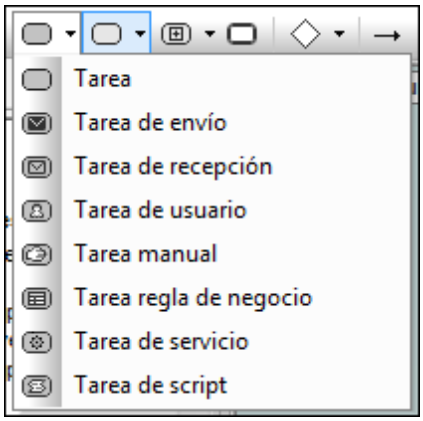
2. En el panel Propiedades haga clic con el botón secundario en el campo `participantsRefs` y seleccione **Agregar valor etiquetado | participantRefs**.
3. Escriba el nombre del nuevo participante (p. ej. **`Participante C`**).

 © 2014 Altova GmbH

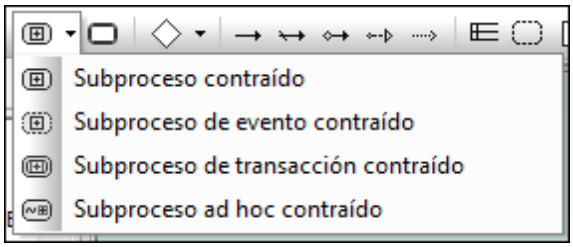


Tareas y subprocesos

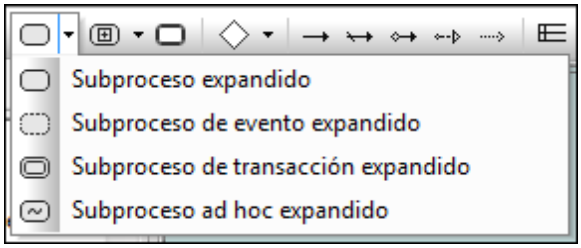
Haga clic en el icono desplegable **Tarea** para insertar una tarea determinada.



Haga clic en el icono desplegable **Subproceso contraído** para insertar un subproceso contraído determinado.



Haga clic en el icono desplegable **Subproceso expandido** para insertar un subproceso expandido determinado.



Objetos de datos

Los datos se representan por medio de cinco elementos de modelado y se insertan con ayuda de estos iconos:

Objeto de datos
Representa la información que fluye a través de los procesos (p. ej. correos electrónicos, documentos comerciales, etc.). Los objetos de datos ofrecen información sobre qué actividades deben llevarse a cabo y qué producen.

Datos de salida
Representa el resultado del proceso.

Datos de entrada
Es una entrada externa para todo el proceso. Pueden ser leídos por una actividad.

Colección de datos
Representa una colección de información (p. ej. una lista de pedidos).

Almacén de datos
Una ubicación donde el proceso puede leer o escribir datos (p. ej. una base de datos).

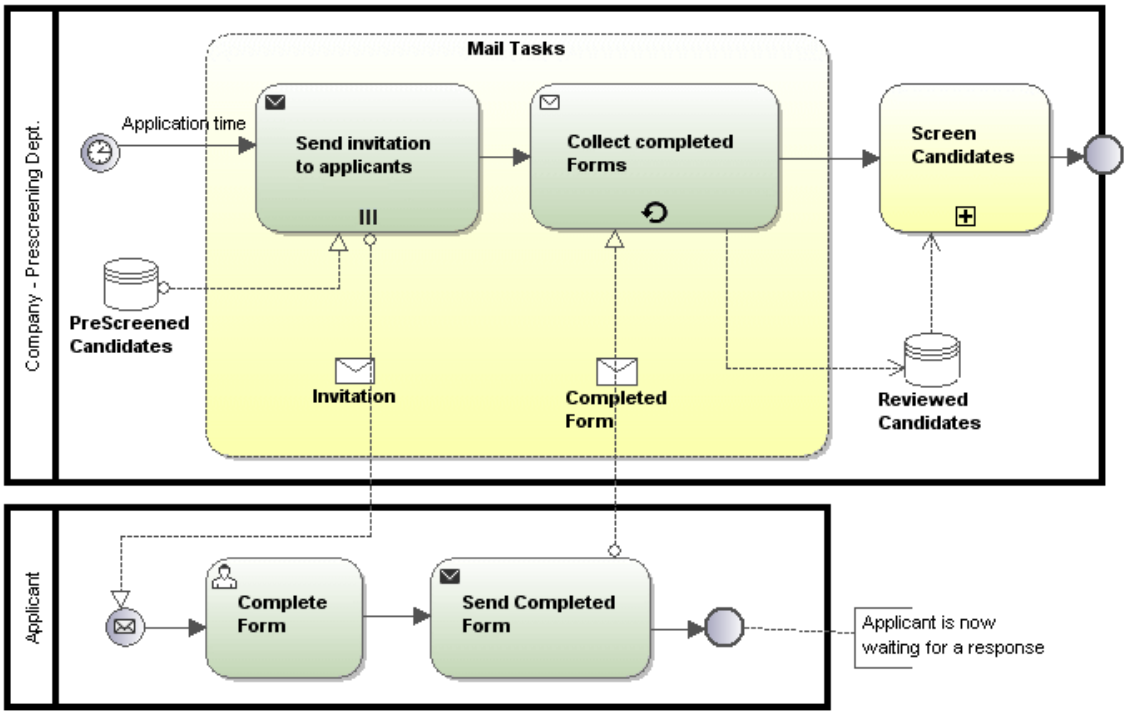
Diagrama de colaboración

Los diagramas de colaboración muestran qué interacciones existen entre los procesos.

Una colaboración suele estar compuesta por dos contenedores o más, que representan los participantes que intervienen en la colaboración. Los intercambios de mensajes que se producen entre los participantes se representan por medio de flujos de mensaje, que conectan los dos contenedores o los objetos que están dentro de los contenedores. Los contenedores también pueden estar vacíos.

En un diagrama de colaboración puede combinar contenedores procesos y coreografías.

 © 2014 Altova GmbH



Conversación

Una conversación es una versión resumida de una colaboración y tiene acceso a los mismos elementos de modelado. Una conversación define un conjunto de intercambios de mensajes relacionados lógicamente: los intercambios de mensajes se relacionan entre sí para reflejar un caso único, como por ejemplo una solicitud seguida de una respuesta.

Las conversaciones tienen dos elementos gráficos que no están disponibles en los demás diagramas BPMN:

- nodos de conversación (conversación, sub-conversación y comunicación) y
- conectores de conversación.

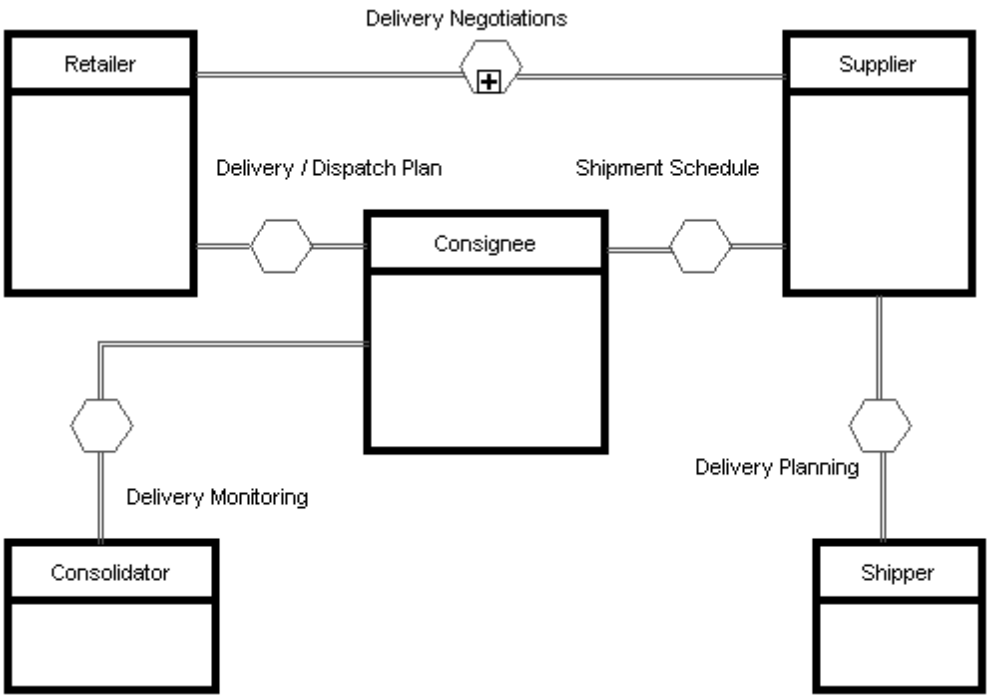
Conversación

Sub-conversación

Comunicación

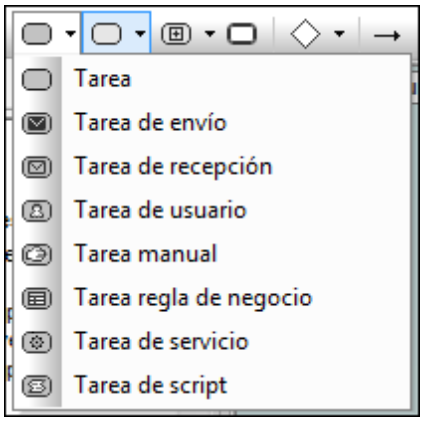
Participante / Contenedor

Conector de conversación

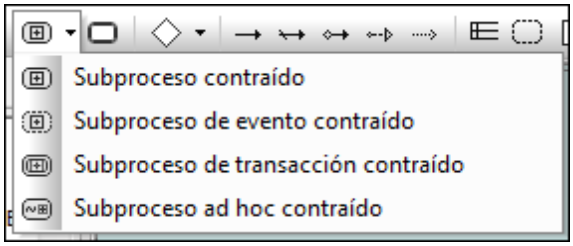


Tareas y subprocesos

Haga clic en el icono desplegable **Tarea** para insertar una tarea determinada.



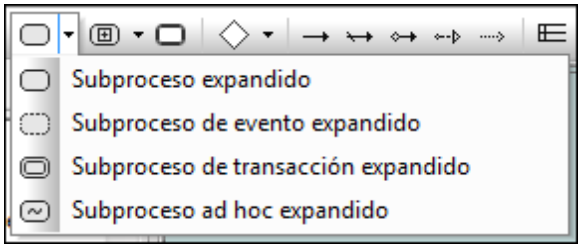
Haga clic en el icono desplegable **Subproceso contraído** para insertar un subproceso contraído determinado.



Haga clic en el icono desplegable **Subproceso expandido** para insertar un subproceso

 © 2014 Altova GmbH

expandido determinado.



Objetos de datos

Los datos se representan por medio de cinco elementos de modelado y se insertan con ayuda de estos iconos:

Objeto de datos
Representa la información que fluye a través de los procesos (p. ej. correos electrónicos, documentos comerciales, etc.). Los objetos de datos ofrecen información sobre qué actividades deben llevarse a cabo y qué producen.

Datos de salida
Representa el resultado del proceso.

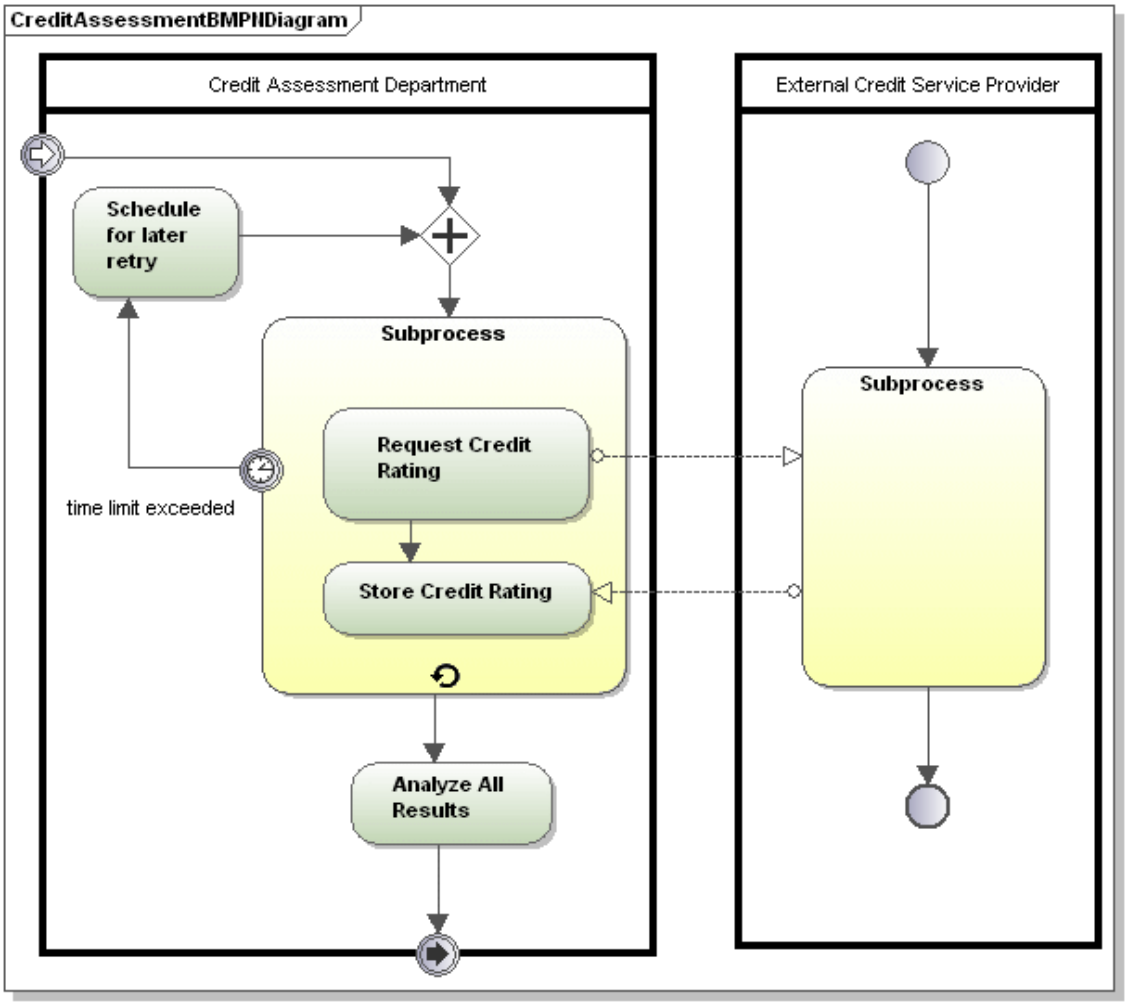
Datos de entrada
Es una entrada externa para todo el proceso. Pueden ser leídos por una actividad.

Colección de datos
Representa una colección de información (p. ej. una lista de pedidos).

Almacén de datos
Una ubicación donde el proceso puede leer o escribir datos (p. ej. una base de datos).

Diagrama de procesos de negocio estándar BPMN 2.0

Los diagramas de procesos de negocio abarcan una amplia gama de datos y emplean varios tipos de modelado diferentes para crear procesos de negocio.



Hay tres tipos de procesos de negocio:

Procesos de negocio **privados** (internos) **no ejecutables**:

- Los procesos no ejecutables son aquellos en los que no hay datos suficientes para poder ejecutar el proceso (por lo general durante el ciclo de desarrollo).

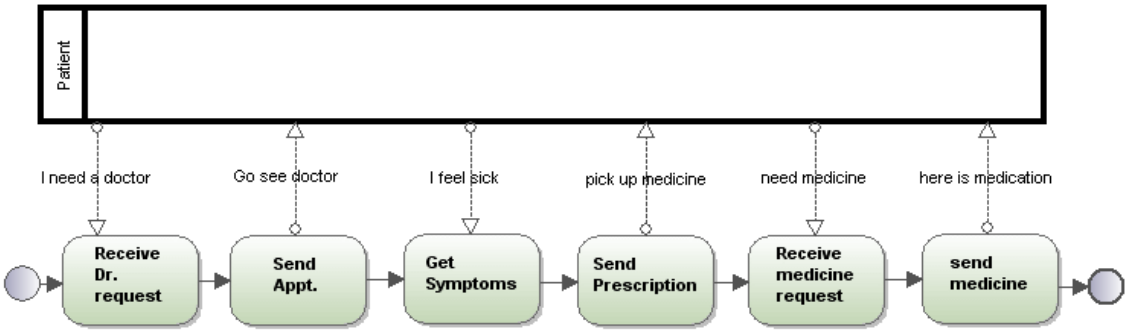
Procesos de negocio **privados** (internos) **ejecutables** (internos):

- Los procesos ejecutables son procesos que se pueden ejecutar porque se han modelado completamente siguiendo las reglas semánticas de BPMN 2.0.

Procesos **públicos**:

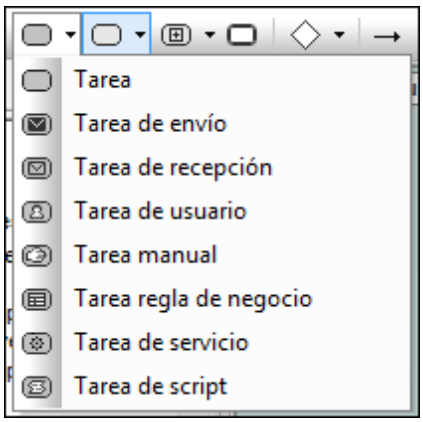
- Definen la interacción que se produce entre un proceso **privado** y otro proceso o participante.

 © 2014 Altova GmbH

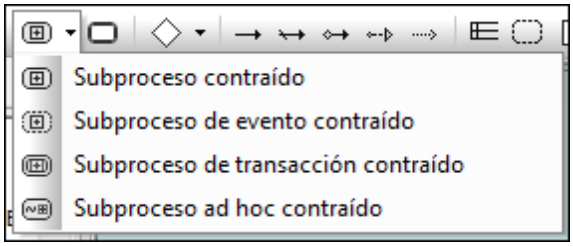


Tareas y subprocesos

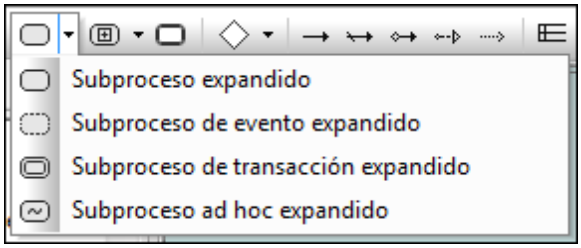
Haga clic en el icono desplegable **Tarea** para insertar una tarea determinada.



Haga clic en el icono desplegable **Subproceso contraído** para insertar un subproceso contraído determinado.



Haga clic en el icono desplegable **Subproceso expandido** para insertar un subproceso expandido determinado.



Objetos de datos

Los datos se representan por medio de cinco elementos de modelado y se insertan con ayuda de estos iconos:

Objeto de datos
Representa la información que fluye a través de los procesos (p. ej. correos electrónicos, documentos comerciales, etc.). Los objetos de datos ofrecen información sobre qué actividades deben llevarse a cabo y qué producen.

Datos de salida
Representa el resultado del proceso.

Datos de entrada
Es una entrada externa para todo el proceso. Pueden ser leídos por una actividad.

Colección de datos
Representa una colección de información (p. ej. una lista de pedidos).

Almacén de datos
Una ubicación donde el proceso puede leer o escribir datos (p. ej. una base de datos).

 © 2014 Altova GmbH

10.3.3 Diagramas SysML

Sitio web de Altova: Diagramas de modelado SysML

SysML 1.2 es un lenguaje de modelado gráfico para ingenieros de sistemas compatible con sistemas de especificación, diseño y validación (como hardware, software, procedimientos, etc.). Los modelos y datos SysML se pueden traspasar a UModel utilizando XMI.

SysML utiliza algunos diagramas UML 2.0, modifica tres diagramas UML y ofrece dos diagramas nuevos.

Diagramas nuevos:	Diagramas de requisitos y paramétricos
Diagramas UML modificados:	Diagramas de actividades, de definición de bloques y de bloques internos
Diagramas UML:	Diagramas de secuencia, de máquina de estados, de casos de uso y de paquetes

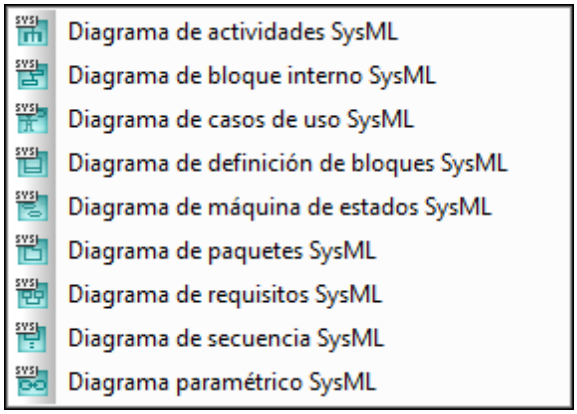
Los diagramas SysML se pueden dividir en dos categorías:

- Diagramas de estructura
- Diagramas de comportamiento

Para añadir un diagrama SysML al proyecto utilice la opción **Diagrama nuevo | Diagramas SysML** del menú contextual del panel Estructura del modelo.

Para insertar un diagrama SysML:

1. En la Estructura del modelo haga clic con el botón secundario en un paquete.
2. Elija el tipo de diagrama SysML que desea insertar (p. ej. **Diagrama nuevo | Diagrama SysML | Diagrama de actividades SysML**).



Nota: si intenta añadir diagramas SysML al paquete `Root`, los diagramas se añaden a un paquete llamado `SysML`.

Diagrama de estructura

Estos son los diagramas de estructura SysML:

- Diagrama de definición de bloques
- Diagrama de bloques internos
- Diagrama paramétrico

- Diagrama de paquetes

Los diagramas de definición de bloques y de bloques internos son diagramas UML 2.0 modificados, mientras que los diagramas paramétricos son un tipo de diagrama nuevo. El diagrama de paquetes sigue el estándar UML 2.0.

Los bloques sirven para describir la estructura de un elemento o sistema y puede tener varios compartimientos que definen sus características. Además a un bloque se le pueden asignar comportamientos.

Diagramas de definición de bloques
Este tipo de diagramas describe las relaciones entre los bloques: asociaciones, generalizaciones y dependencias.

Diagramas de bloques internos
Describe la estructura interna de un bloque, en cuanto a las propiedades y los conectores que existen entre ellas (es decir, la relación de sus partes constituyentes). Para ello se sirve de puertos, conectores y flujos.

Diagramas paramétricos
Utilizan bloques de restricción (definidos en un diagrama de definición de bloques) para restringir las propiedades de otros bloques del diagrama e integrar el análisis técnico con el modelado del diseño.

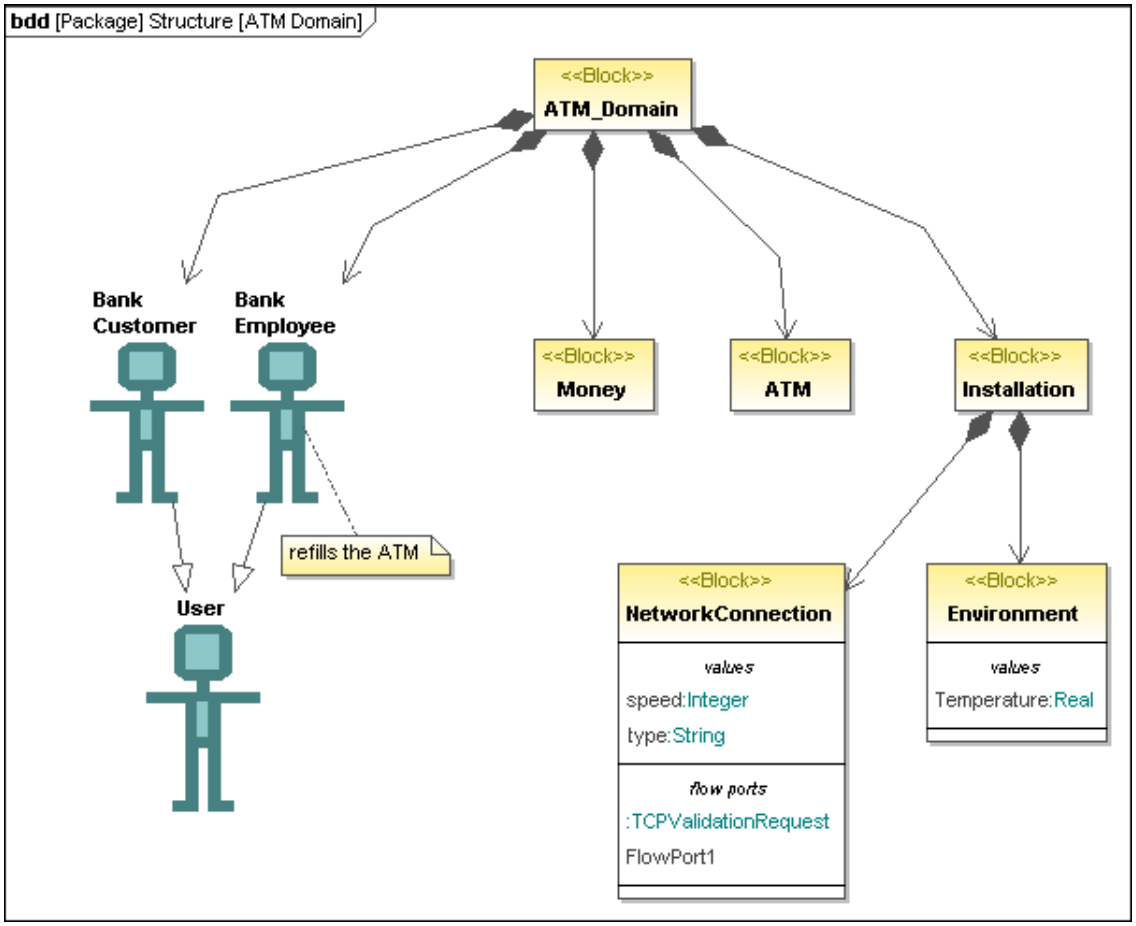
Diagramas de paquetes
Organizan los elementos de modelado en elementos empaquetables y definen dependencias entre los paquetes y entre los elementos de modelado del paquete.

Diagrama de definición de bloques

Los diagramas de definición de bloques están basados en los diagramas de clases UML pero las restricciones y extensiones vienen definidas por el estándar SysML.

Este tipo de diagramas describe la relación que existe entre varios bloques, entre sus asociaciones, generalizaciones y dependencias. Esto puede ayudar a modelar la jerarquía de un sistema o árbol de clasificación.

 © 2014 Altova GmbH

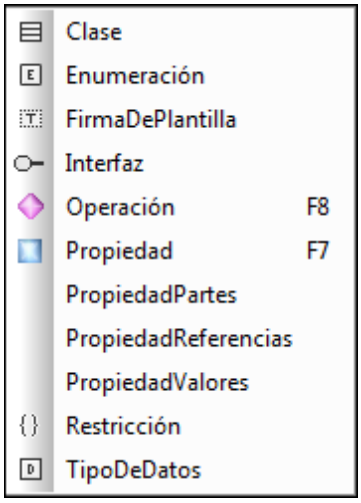


Los bloques tienen varias estructuras llamadas propiedades:

- Propiedad de partes
- Propiedad de referencias
- Propiedad de valores

Para añadir una propiedad a un bloque:

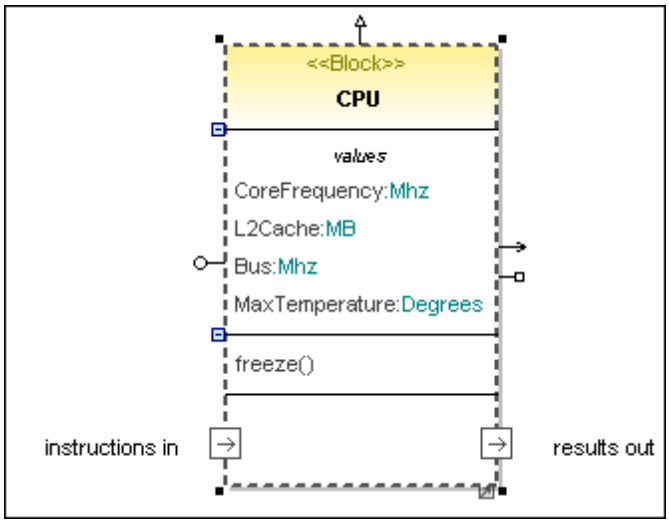
1. Haga clic con el botón secundario en un bloque y elija la opción **Nuevo/a | Propiedad** (o **PropiedadPartes**, **PropiedadReferencias** o **PropiedadValores**).



Nota: en el bloque se añade un compartimiento nuevo (p. ej. el compartimiento **`partes`** para una `PropiedadPartes`).

Para ver las propiedades de los bloques en forma de nodos:

1. Haga clic con el botón secundario en el bloque y elija la opción **Mostrar | Mostrar propiedades en forma de nodos en el nodo**.
   Esto muestra las propiedades, los puertos, etc. del bloque como objetos en un compartimiento.



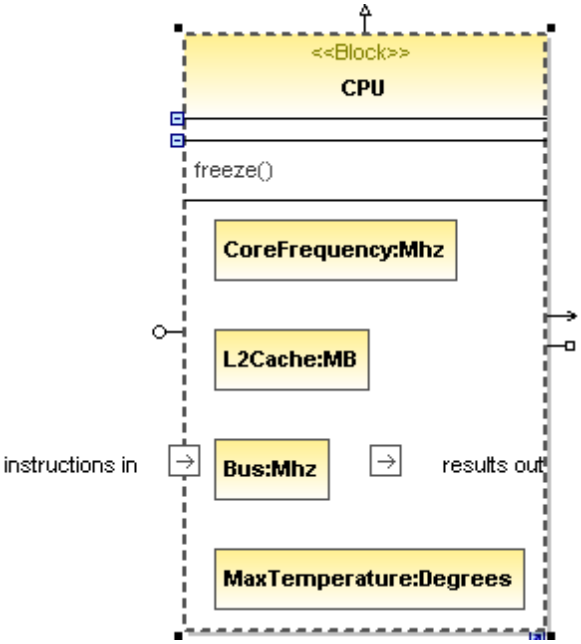
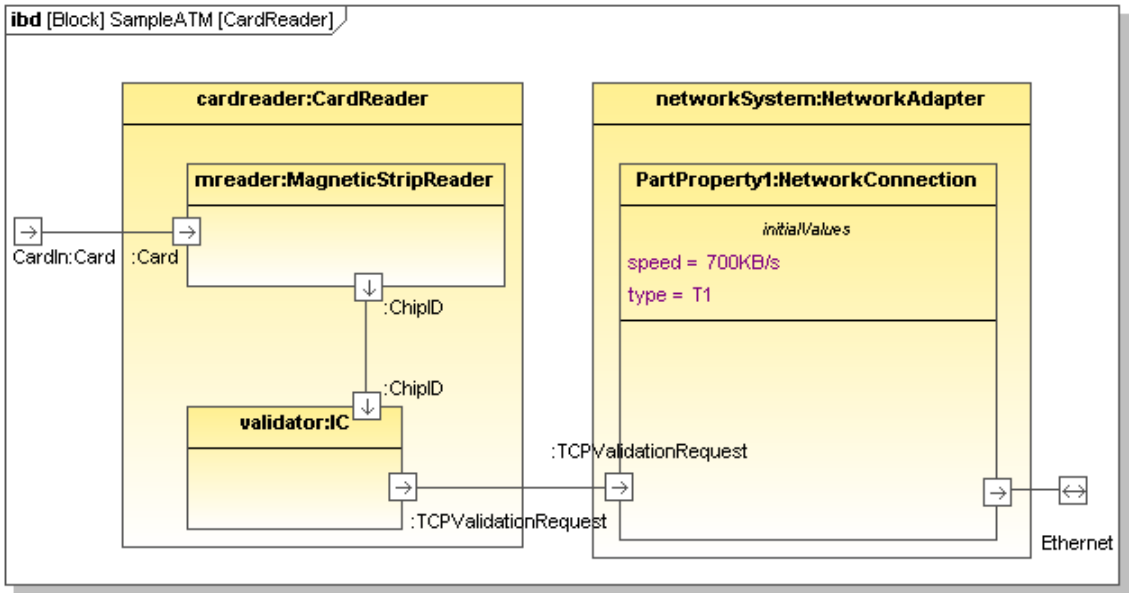


Diagrama de bloque interno

Los diagramas de bloques internos están basados en los diagramas de estructura de un compuesto pero las restricciones y extensiones vienen definidas por el estándar SysML.

Este tipo de diagramas describe la estructura interna de un bloque en cuanto a las propiedades y los conectores que existen entre ellas (es decir, la relación de sus partes constituyentes). Para ello se sirven de puertos, conectores y flujos.



Para añadir valores iniciales a una propiedad:

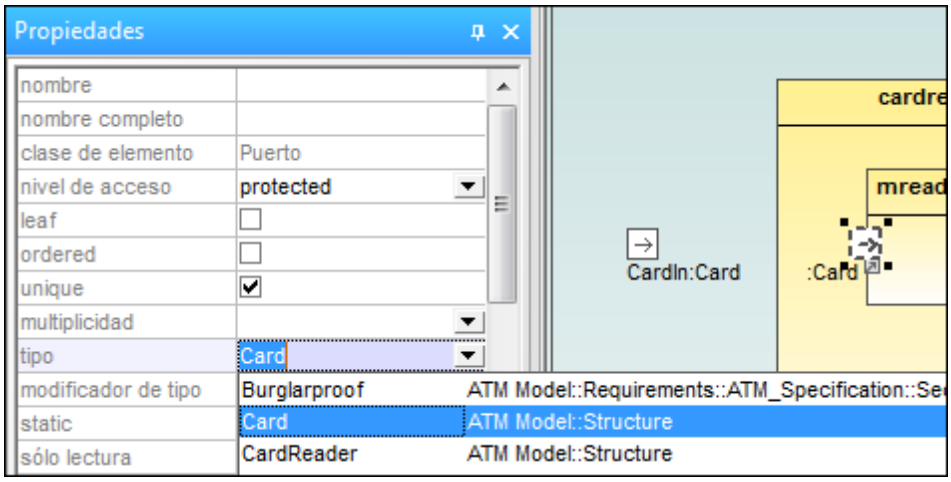
1. Haga clic con el botón secundario en la propiedad y elija la opción **Nuevo/a | Valores iniciales** en el menú contextual.
2. Haga doble clic en el marcador de posición y escriba los valores (p. ej. velocidad =

700KB/s).

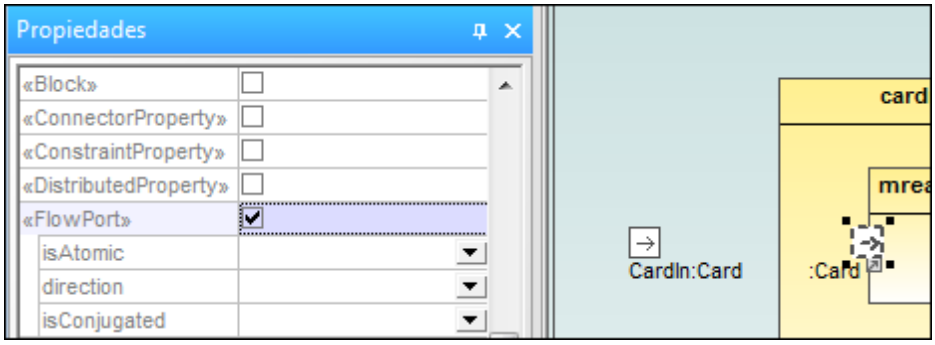
Para insertar un PuertoDeFlujo:

Primero debe comprobar que el tipo de propiedad es un bloque o un clasificador:

1. Haga clic en el icono **PuertoDeFlujo** de la barra de herramientas y después en el contorno de la propiedad donde desea añadir el puerto.
   El puerto de flujo se añade al contorno de la propiedad.



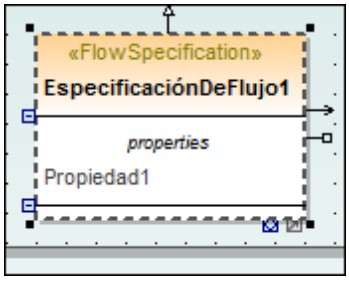
2. Escriba el nombre del puerto de flujo en el campo `nombre` del panel Propiedades.
3. Seleccione el tipo de puerto en la lista desplegable del campo `tipo` (p. ej. `Card`). Compruebe que sigue seleccionado el puerto.
4. Desplácese hasta la parte inferior del panel Propiedades y observe que ahora hay más parámetros disponibles.



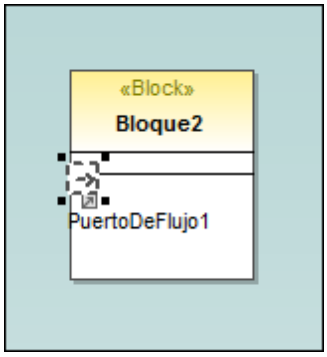
Para crear un puerto de flujo conjugado atómico:

1. Cree una `EspecificaciónDeFlujo` (interfaz) en un diagrama de definición de bloques.

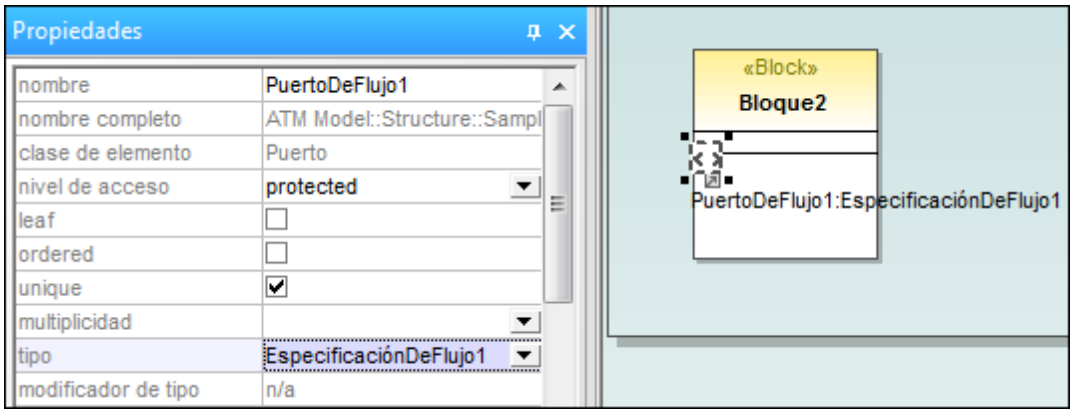
 © 2014 Altova GmbH



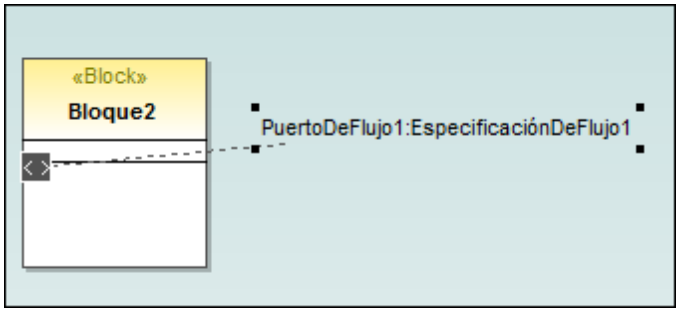
2. En el diagrama de bloques internos haga clic en el puerto de flujo que desea definir.



3. En el panel Propiedades elija el `tipo` de puerto de flujo `EspecificaciónDeFlujo`.



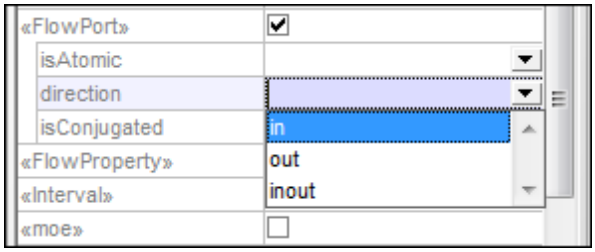
4. Elija el valor `true` para la propiedad `isConjugated`.



Este tipo de puerto aparece con un fondo oscuro.

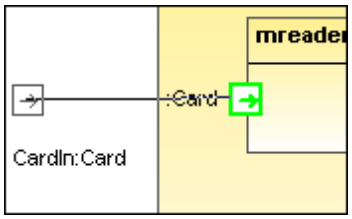
Para cambiar la dirección del puerto de flujo:

1. Haga clic en la entrada `PuertoDeFlujo` y seleccione el valor `in`, `out` o `inout` en el cuadro combinado dirección.



Para unir dos puertos:

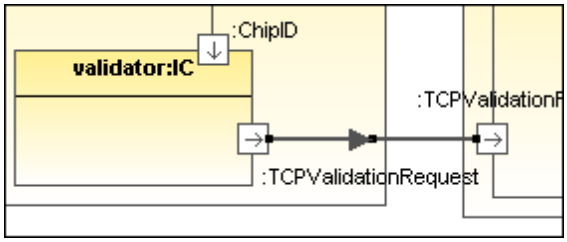
1. Haga clic en el icono **Conector** de la barra de herramientas.
2. Haga clic en el primer puerto y arrastre el puntero hasta el segundo puerto.
3. Cuando el puerto de destino aparezca resaltado, suelte el conector.



Para crear el flujo del elemento:

En los diagramas SysML puede crear flujos de elemento entre las asociaciones de bloques y en los demás conectores.

1. Haga clic con el botón secundario en un conector y elija el comando **Nuevo/a | Flujo del elemento (de derecha a izquierda) / Flujo del elemento (de izquierda a derecha)**.



Al conector se le añade una punta de flecha que indica la dirección del flujo del elemento.

Diagrama paramétrico

El diagrama paramétrico es un tipo de diagrama propio de SysML que combina análisis técnico con modelado de diseños. Los diagramas paramétricos son similares a los diagramas de bloques internos, con la diferencia de que los paramétricos solamente muestran los conectores que están conectados a parámetros de restricción en uno de sus extremos.

Los diagramas paramétricos se sirven de bloques de restricción, definidos en un diagrama de definición de bloques, para limitar las propiedades de otros bloques en el diagrama paramétrico. Los bloques de restricción se dibujan con esquinas redondeadas y no con recuadros.

 © 2014 Altova GmbH

Diagrama paramétrico Wt

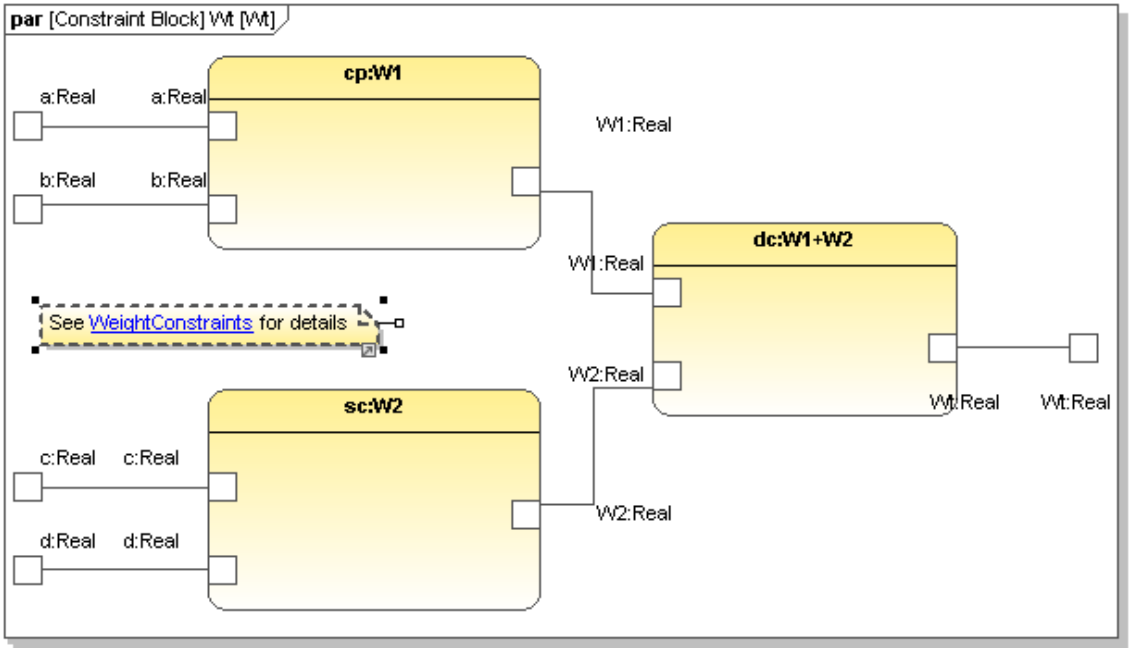


Diagrama de definición de bloques: definición de restricciones

La palabra clave «`constraint`» de una definición de bloque indica que el bloque es un bloque de restricción. Los parámetros de la restricción aparecen en el compartimiento *`parámetros`*.

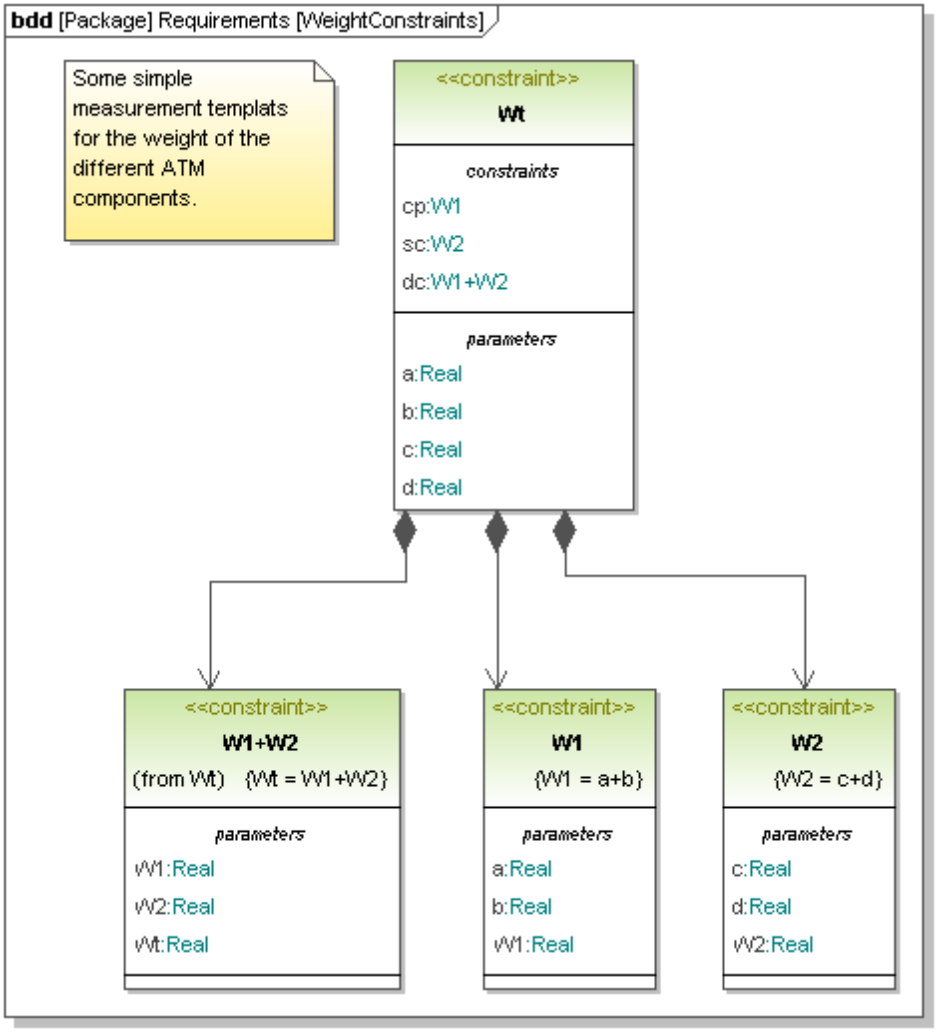
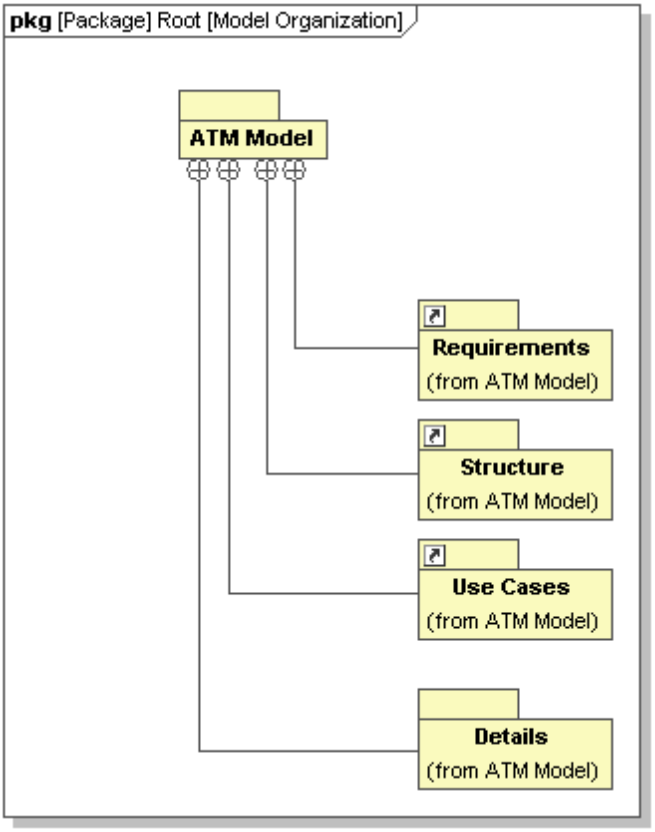


Diagrama de paquetes

Estos diagramas sirven para organizar elementos de modelado en elementos empaquetables y para definir las dependencias que existen entre los paquetes y entre sus elementos. El paquete define un espacio de nombres para los elementos empaquetables.

 © 2014 Altova GmbH



El modelo se puede organizar de varias formas y cada una de ellas muestra un aspecto único del modelo. Por ejemplo:

- jerarquía del sistema (por niveles)
- tipo de diagrama (por dominio)
- por puntos de vista (equipo de producto integrado)

Los puntos de visa permiten ver el modelo de una manera diferente (p. ej. en la vista de análisis se pueden ver los niveles jerárquicos).

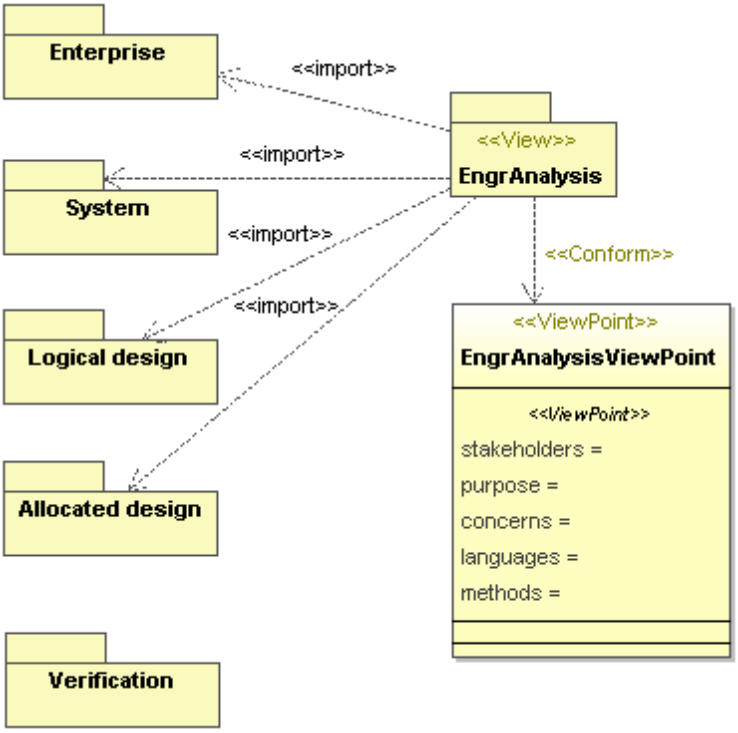


Diagrama de requisitos

El diagrama de requisitos es un tipo de diagrama diseñado para SysML que combina los modelos de comportamiento y de estructura de SysML con modelos de análisis técnico (como modelos de rendimiento, de fiabilidad, etc.).

Se trata de una construcción de modelado para requisitos textuales y para la relación entre los requisitos y los demás elementos de modelado que los cumplen.

 © 2014 Altova GmbH

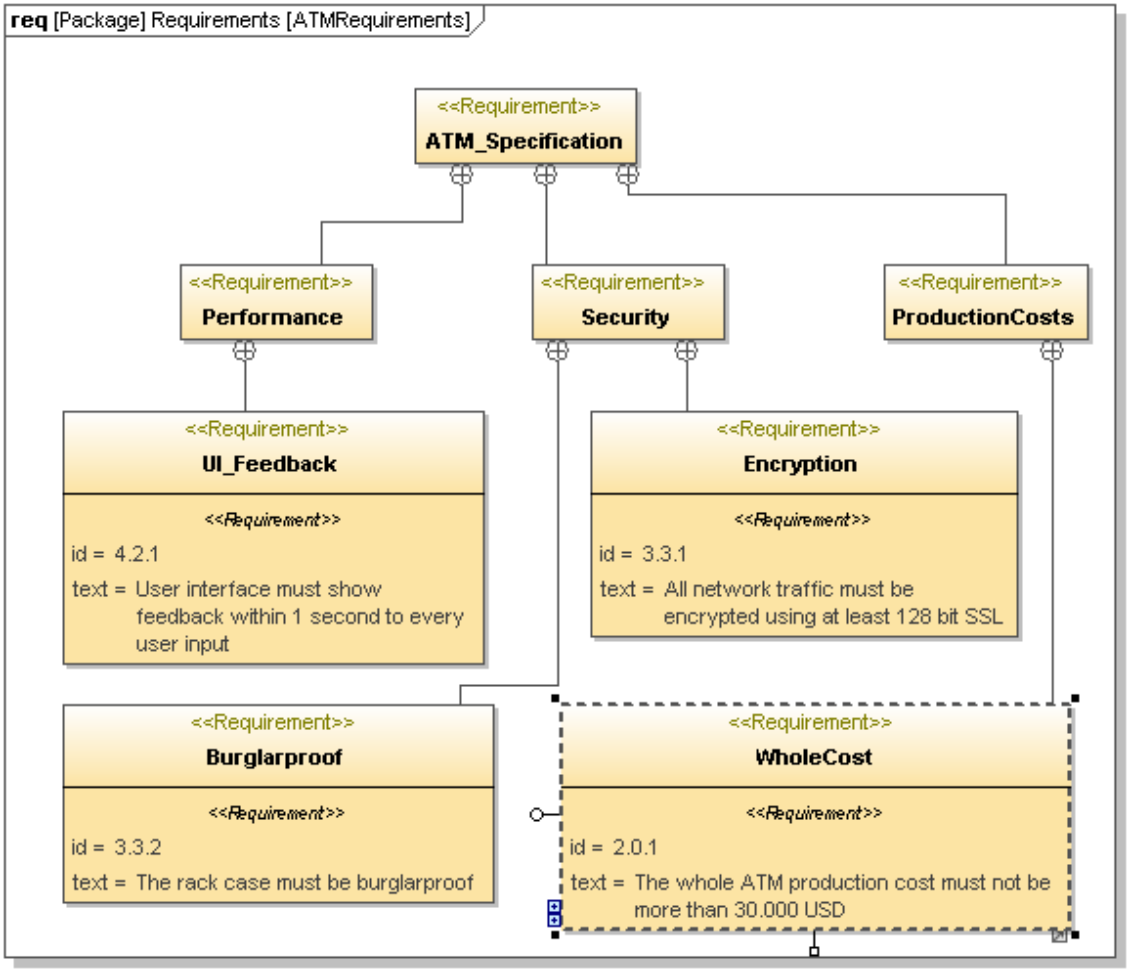


Diagrama de comportamiento

Estos son los diagramas de comportamiento SysML:

- Diagrama de actividades
- Diagrama de secuencia
- Diagrama de máquina de estado
- Diagrama de casos de uso

SysML modifica el diagrama de actividades UML 2.0, lo cual se explica en el siguiente apartado. El resto de los diagramas siguen el estándar UML 2.0.

Diagrama de actividades

Los diagramas de actividades SysML son diagramas de actividades UML 2.0 modificados.

- Para obtener **información general** sobre los diagramas de actividades, consulte el apartado Diagrama de actividades.
- Para aprender a **insertar elementos** en los diagramas de actividades, consulte el apartado Insertar elementos en diagramas de actividades.
- Para aprender a **crear bifurcaciones** y convergencias, consulte el apartado Crear bifurcaciones y convergencias.
- Para ver ejemplos de elementos de diagramas de actividades, consulte el apartado

Elementos de los diagramas de actividades.

En UML las actividades incluidas son clases y las instancias de actividad son la ejecución propiamente dicha de la actividad. Cuando se crea una instancia de actividad, se inicia la ejecución. En cambio, cuando se destruye la instancia, la ejecución termina.

Estas son las modificaciones que SysML hace en los diagramas de actividades:

- Admite el modelado de flujos continuos mediante estereotipos continuos (transmisión de actividades por secuencias).
- Alineación de actividades con el diagrama EFFBD (*Enhanced Functional Flow Block*), que no es más que un diagrama de actividades con particiones asignadas.

En SysML las actividades pueden aparecer en diagramas de definición de bloques como bloques normales y corrientes, pero con la palabra clave/el estereotipo `<<Activity>>`.

Diagrama de actividades SysML:

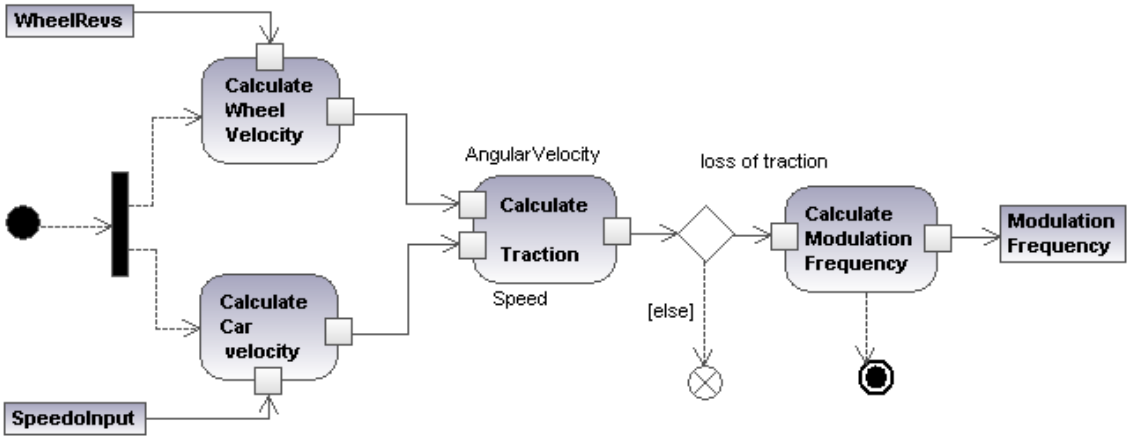
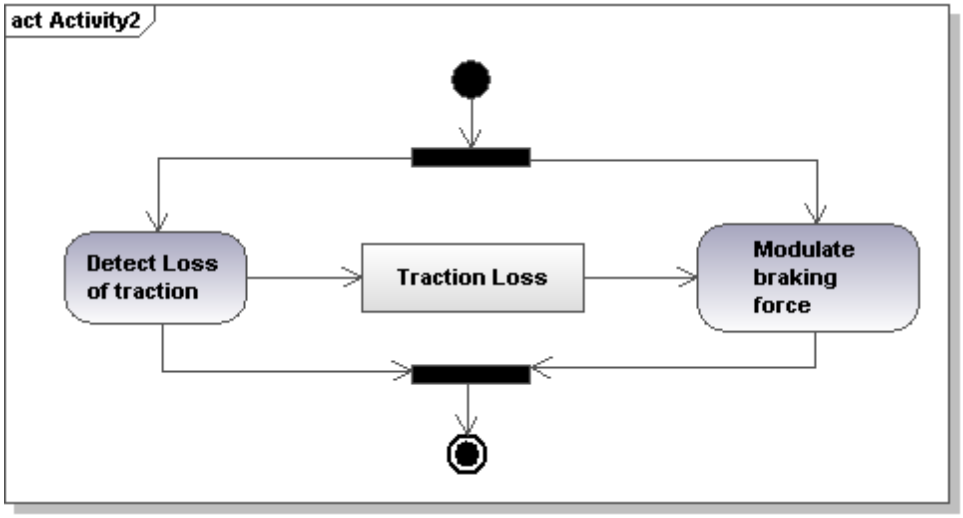


Diagrama de actividades sin compartimentos



 © 2014 Altova GmbH

Diagrama de actividades con compartimentos

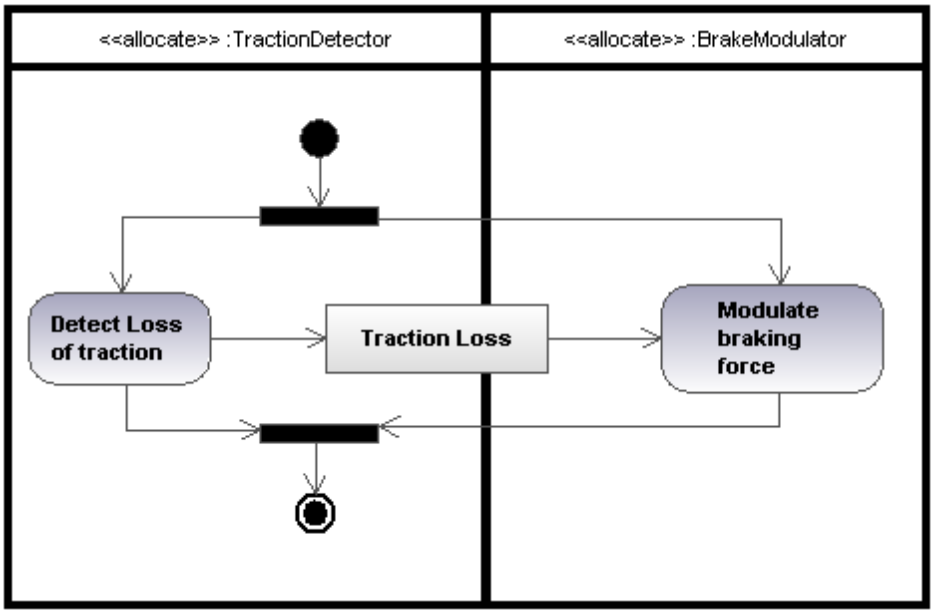


Diagrama de secuencia

Los diagramas de secuencia SysML siguen la especificación UML 2.0.

- Para obtener **información general** sobre los diagramas de secuencia, consulte el apartado Diagramas de secuencia.
- Para aprender a **insertar elementos** en estos diagramas (líneas de vida, fragmentos combinados, etc.), consulte el apartado Insertar elementos en diagramas de secuencia.
- Para aprender a **generar** diagramas de secuencia a partir de código fuente, consulte el apartado Generar diagramas de secuencia a partir de código fuente.

Diagrama de secuencia SysML:

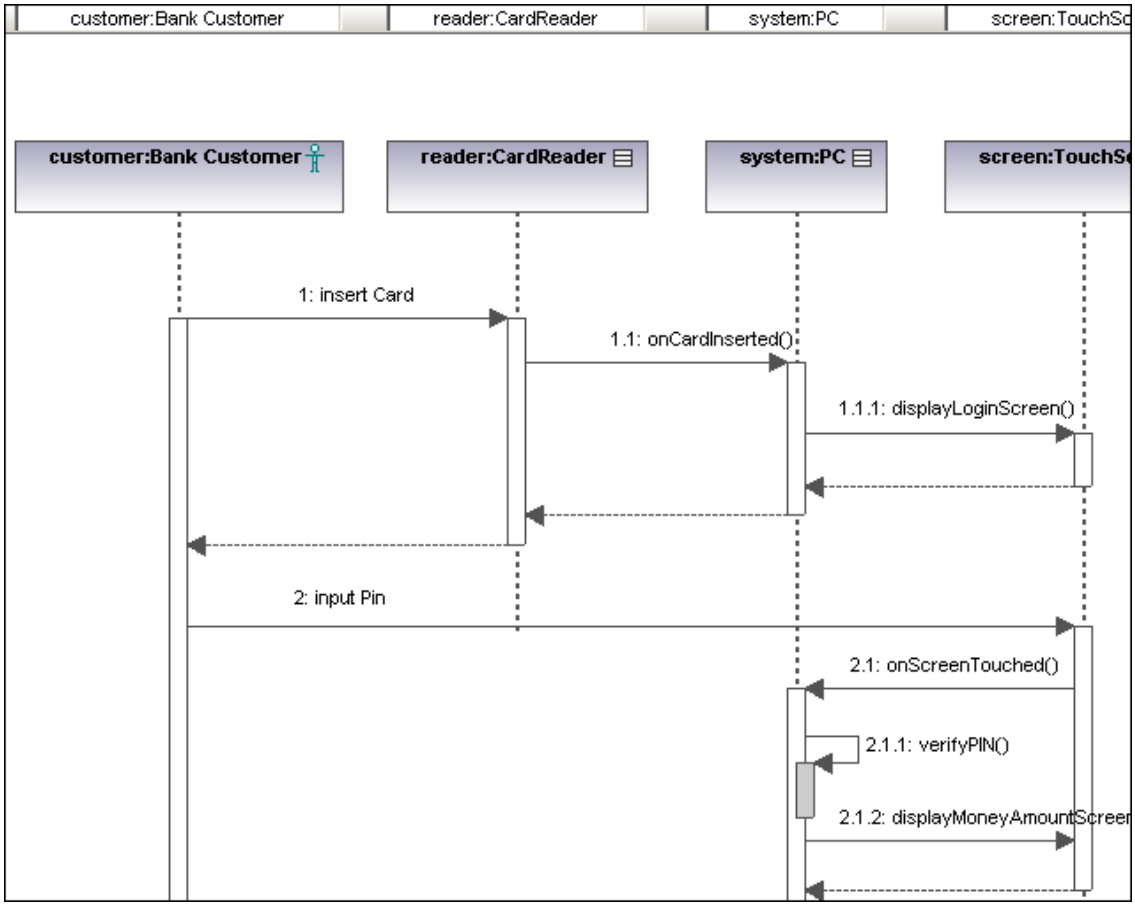


Diagrama de máquina de estados

Los diagramas de máquina de estados SysML siguen la especificación UML 2.0.

- Para obtener **información general** sobre estos diagramas, consulte el apartado Diagramas de máquina de estados.
- Para aprender a **insertar elementos** en estos diagramas, consulte el apartado Crear estados, actividades y transiciones.
- Para aprender a **crear estados compuestos**, consulte el apartado Estados compuestos.
- Para ver **ejemplos** de elementos de diagramas de máquina de estados, consulte el apartado Elementos de diagramas de máquina de estados.

Diagrama de máquina de estados SysML:

 © 2014 Altova GmbH

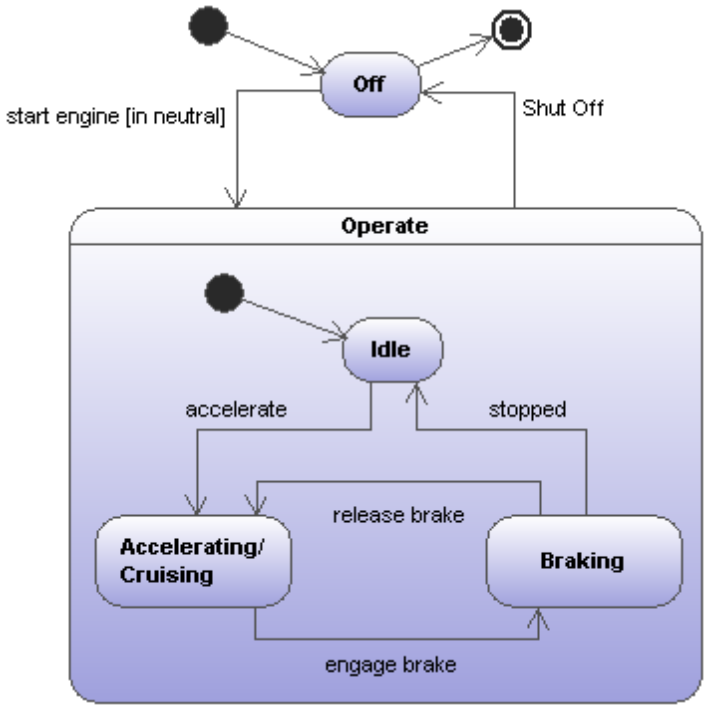
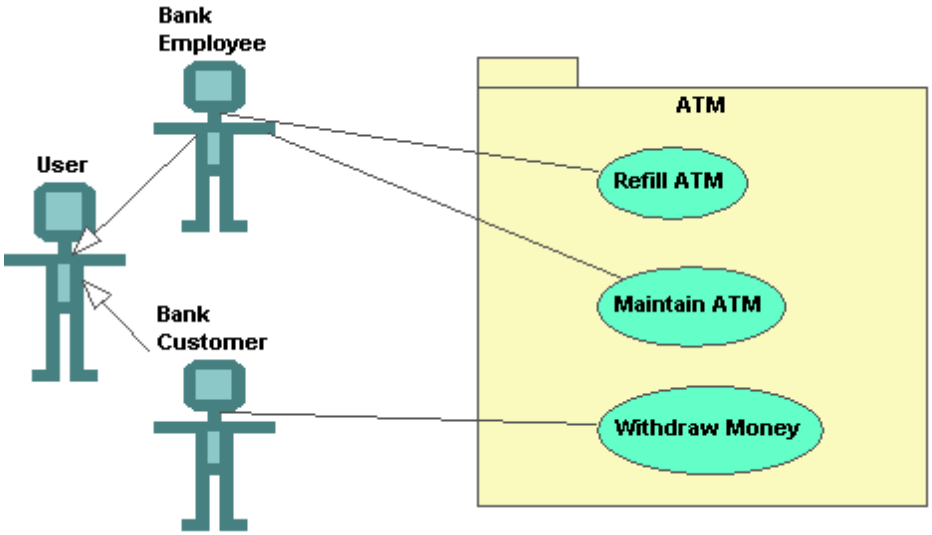


Diagrama de casos de uso

Los diagramas de casos de uso SysML siguen la especificación UML 2.0. Para más información sobre este tipo de diagramas, consulte el apartado Casos de uso del *Tutorial*.

Diagrama de casos de uso SysML:



10.3.4 Diagramas de BD (importar bases de datos SQL)

UModel puede importar y exportar bases de datos SQL y presentar la estructura de la BD como elementos de modelado de BD en un diagrama de bases de datos UML.

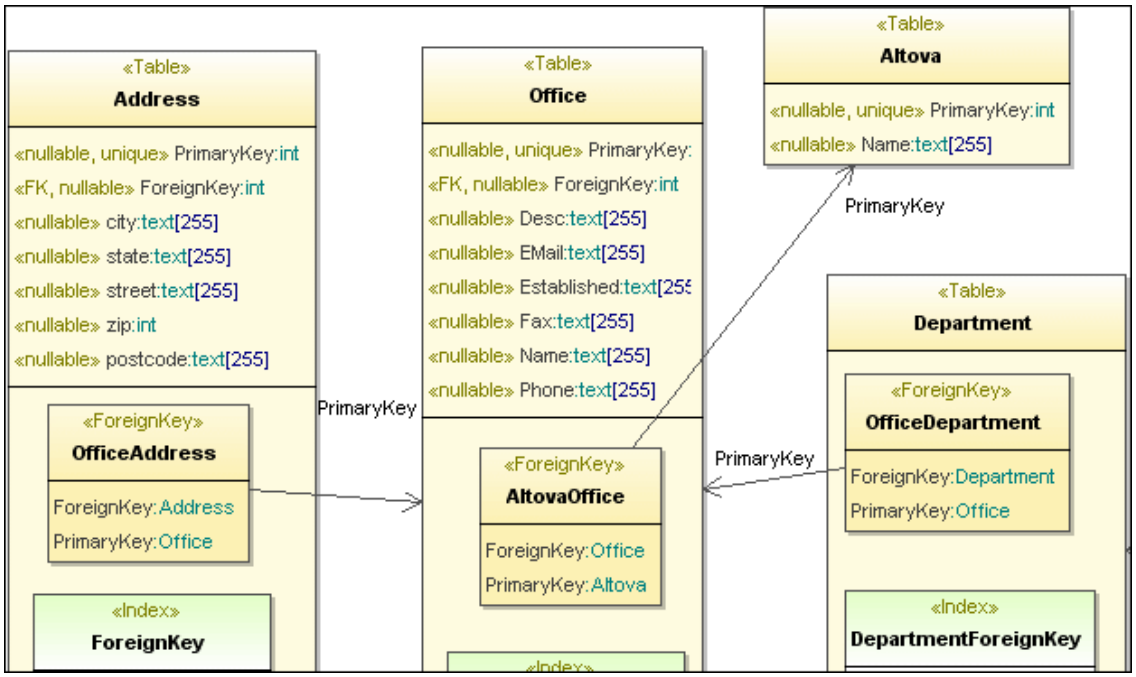
Estos son los elementos de BD que se pueden importar a un modelo de BD:

Tablas
Restricciones de comprobación
Claves principales, foráneas y únicas
Índices

Vistas
Disparadores
Procedimientos almacenados
Funciones

Nota: las vistas, los disparadores, los procedimientos almacenados y las funciones solamente se pueden importar, no se pueden añadir.

Una vez finalizado el proceso de modelado, la estructura del modelo de la BD se puede exportar a la BD con ayuda del comando de menú **Combinar el código de programa con el proyecto de UModel**. Esto crea un archivo de script de cambios que se puede ejecutar o guardar para ejecutarlo más tarde.



Importar bases de datos SQL en UModel

Las bases de datos compatibles y sus objetos raíz aparecen a continuación. Altova procura ofrecer compatibilidad con otras bases de datos ODBC/ADO, pero sólo se garantiza una conexión y un procesamiento correctos con las bases de datos de la lista. Si usa la versión de 64 bits de UModel, compruebe que tiene acceso a los controladores de BD de 64 bits de la BD a

 © 2014 Altova GmbH

la que quiere conectarse. UModel es compatible con los archivos lógicos de la base de datos IBM iSeries y los muestra como si fueran vistas.

Base de datos (con soporte nativo)	Objeto raíz
MS Access 2003, 2007, 2010, 2013	database
MS SQL Server 2005, 2008, 2012	database/schema
Oracle 9i, 10g, 11g	schema
MySQL 5.x, 5.5.28	database
PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1	database
Sybase ASE12, ASE15	database
IBM DB2 8.x, 9.5, 9.7, 10.1	schema
IBM DB2 para i 6.1, 7.1	schema
IBM Informix 11.70	database

Cuando instale la versión de **64 bits** de UModel, compruebe que tiene acceso a los controladores de BD de 64 bits necesarios para el tipo de BD a la que desea conectarse.

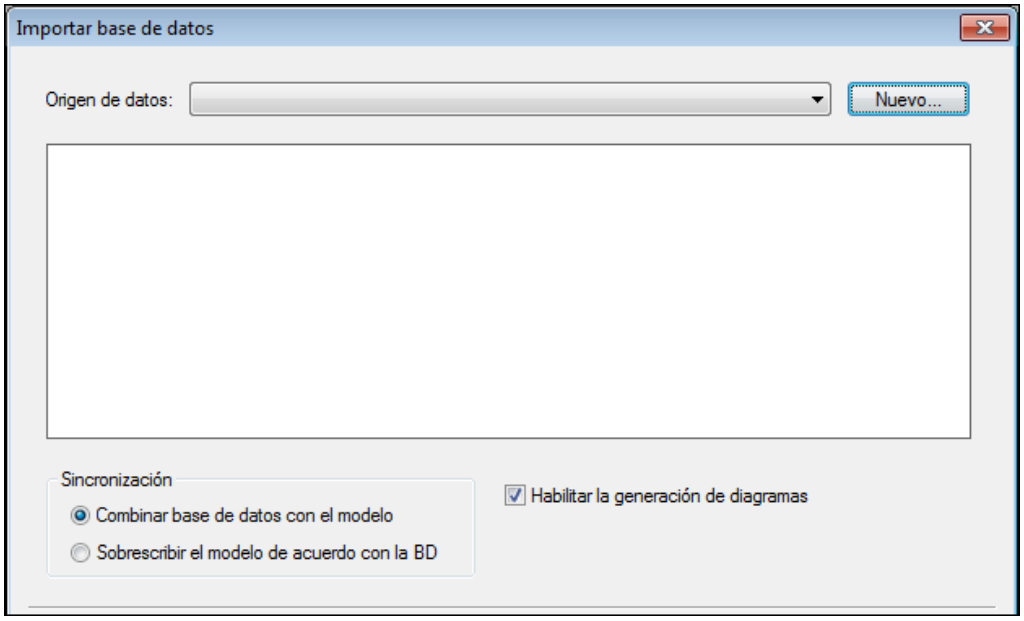
Cuando se conecte a la BD a través del cliente SQL nativo, es posible que reciba este mensaje de error:
Error de conexión de base de datos.
Motivo: no se puede conectar a la base de datos. Error de inicio de sesión. El inicio de sesión se realiza desde un dominio que no es de confianza y no se puede utilizar con autenticación de Windows.

Si esto ocurre, siga estas instrucciones:

Para una conexión ADO a un MS SQL Server con controlador SQL Server Native Client 10.0 debe definir estos valores de propiedades en la pestaña *Todas* del cuadro de diálogo "Propiedades de vínculo de datos":
(i) el valor *Seguridad integrada* debe ser un carácter de espacio
(ii) el valor de *Información de seguridad persistente* debe ser `true`.

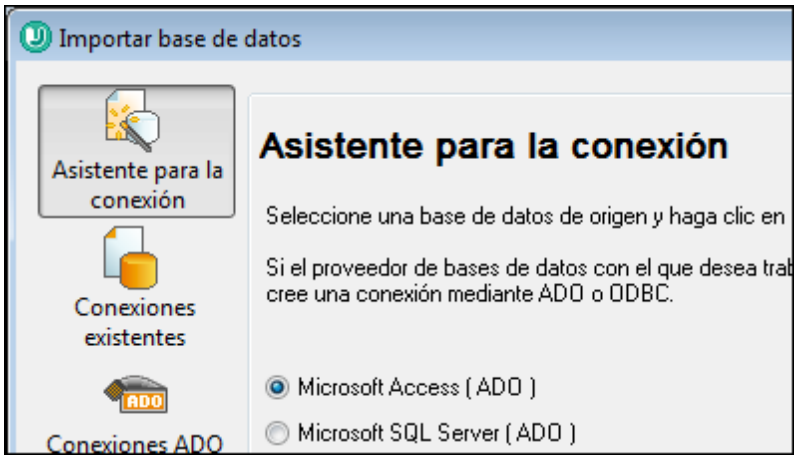
Para importar una base de datos SQL en UModel:

1. Seleccione el comando de menú **Proyecto | Importar base de datos SQL...**.
2. En el cuadro de diálogo haga clic en el botón **Nuevo...** si es la primera vez que importa una BD en UModel.



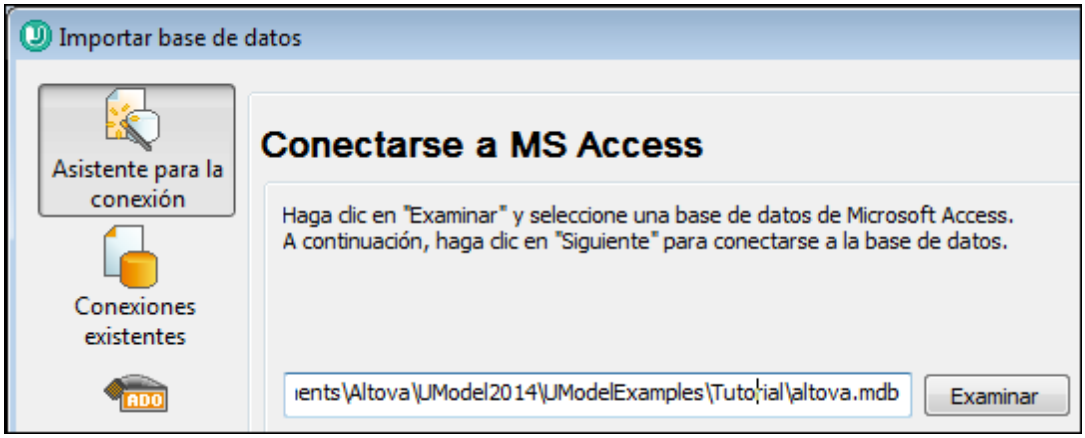
(Si ya importó una BD anteriormente, haga clic en el cuadro combinado *Origen de datos* y seleccione la BD.)

3. Ahora se abre el **Asistente para la conexión**, donde puede seleccionar la BD y las tablas que desea importar en UModel.

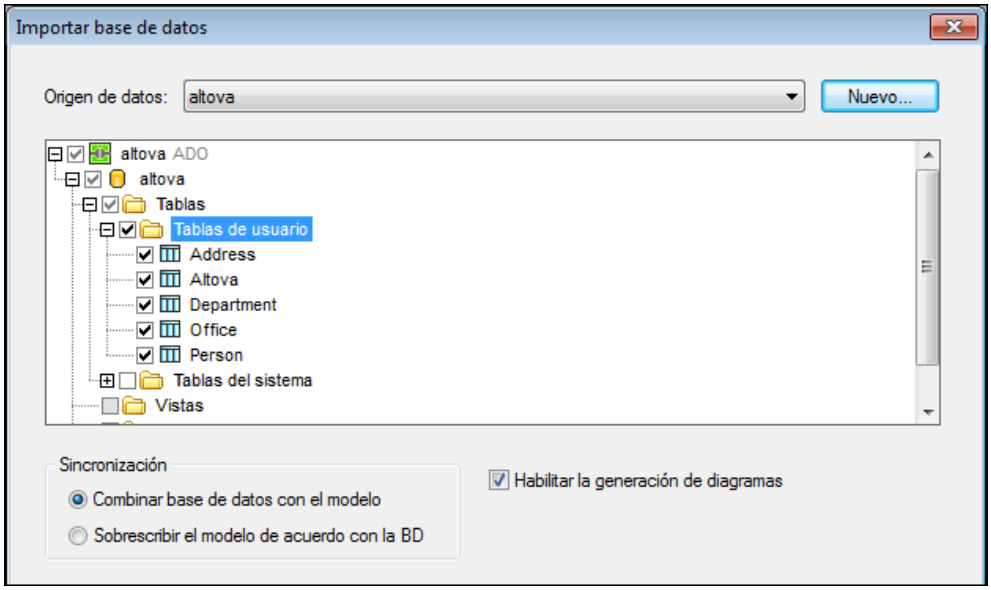


Para este ejemplo insertamos la BD MS Access de ejemplo que viene con UModel, llamada `Altova.mdb`. No olvide que puede importar todas las BD que aparecen en la lista del Asistente para la conexión.

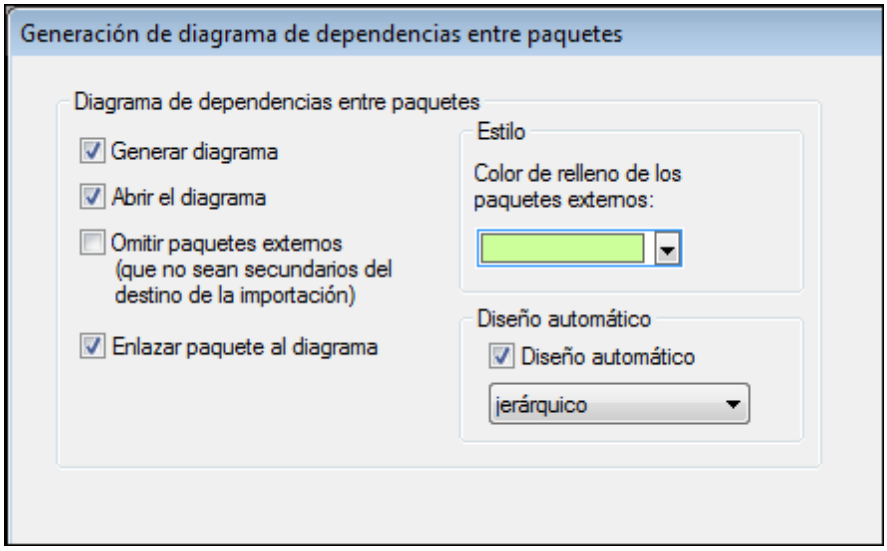
 © 2014 Altova GmbH



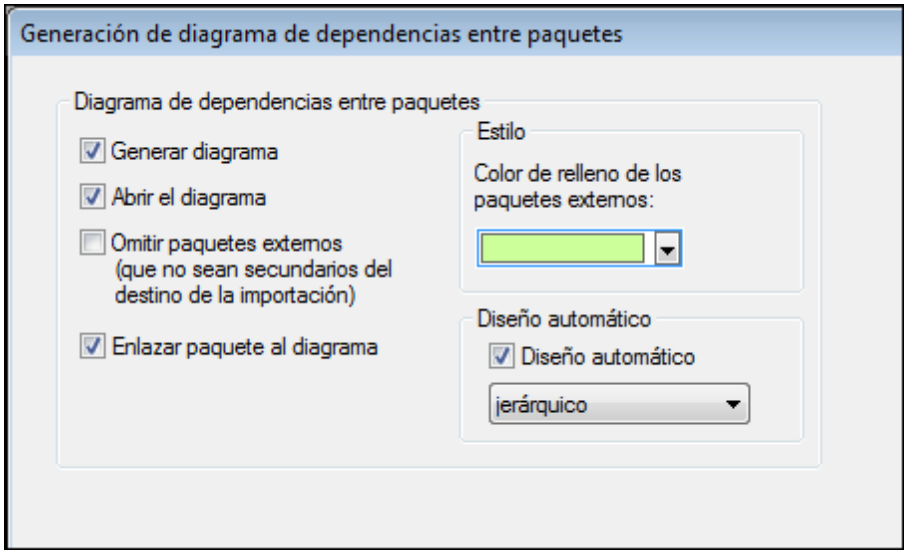
4. Haga clic en **Conectarse** para establecer la conexión con la BD. En el siguiente diálogo escriba el nombre de origen de datos DSN (p. ej. Altova).
5. En el siguiente cuadro de diálogo puede seleccionar las tablas que se deben importar en UModel.
6. Marque la casilla *Tablas de usuario* para seleccionar todas las tablas de usuario y después haga clic en **Siguiente** para continuar.



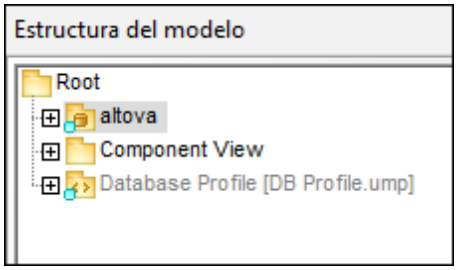
En la siguiente pantalla puede definir las opciones de configuración del diagrama. Para este ejemplo usamos las opciones de configuración predeterminadas.



7. Haga clic en **Siguiente** para definir las opciones de configuración del diagrama de dependencias entre paquetes.



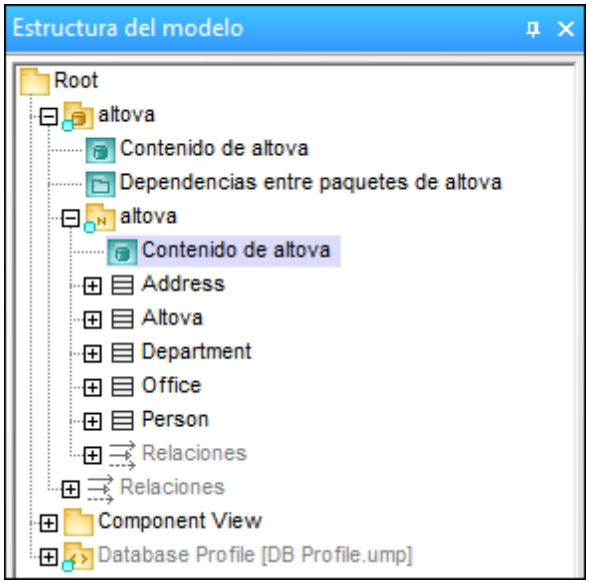
8. Por último haga clic en **Finalizar** para insertar la base de datos en UModel.



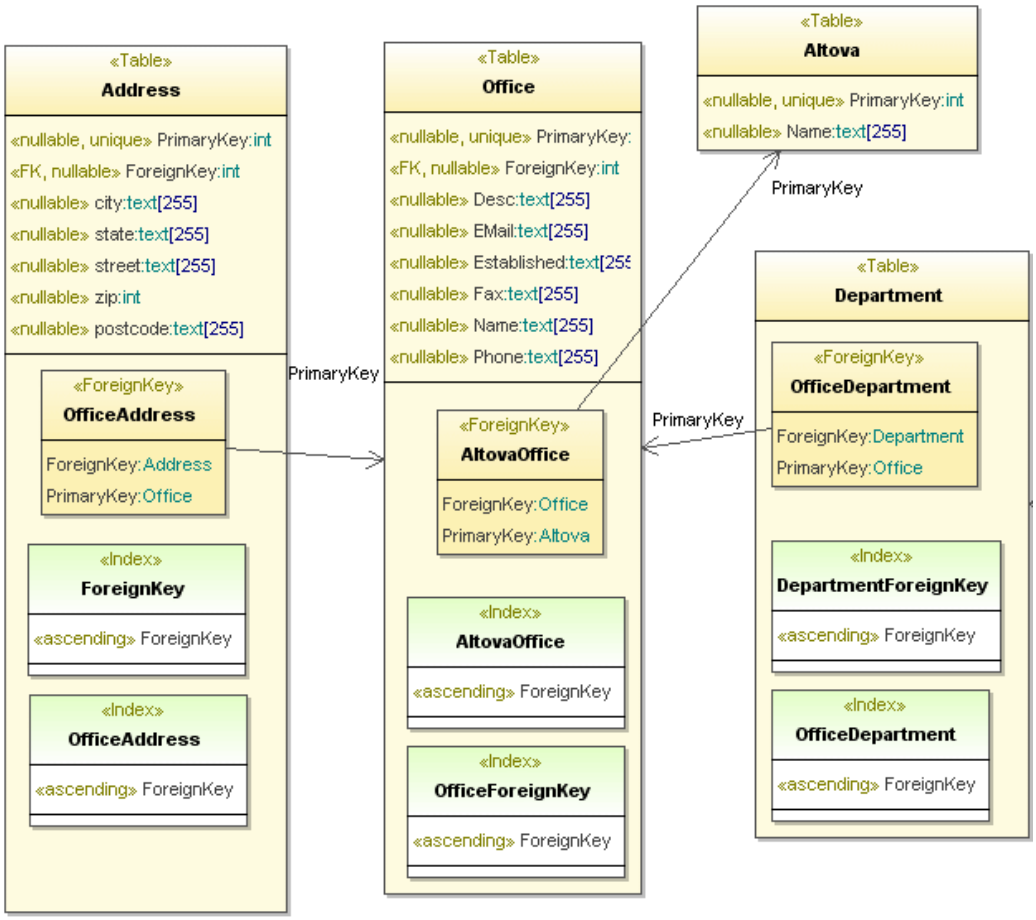
La BD se importó en dos paquetes. Observe que además se importó automáticamente el paquete **`Database Profile`**.

9. Expanda el paquete **`altova`** hasta ver el icono del diagrama **`Contenido de altova`**.

 © 2014 Altova GmbH



10. Ahora haga doble clic en el icono del diagrama `Contenido de altova` para ver los elementos de modelado de la BD.



Conexiones JDBC

En este apartado describimos cómo se establece una conexión con una BD con ayuda de un controlador JDBC. Las instrucciones que aparecen a continuación describen una conexión a una base de datos IBM DB2 pero también sirven para otros tipos de BD que admitan conexiones JDBC.

Requisitos

Antes de conectarse a la BD por JDBC:

1. Compruebe que está instalado Java Runtime Environment (JRE). Use un JRE de 32 bits si el equipo es de 32 bits o un JRE de 64 bits si el equipo es de 64 bits.
2. Instale un controlador JDBC. No hay ningún requisito de instalación especial. Basta con copiar el controlador en un directorio local, por ejemplo `c:\jdbc`. Recuerde que los controladores JDBC (que son archivos Jar) son independientes de la plataforma y los controladores funcionan en sistemas operativos de 32 y 64 bits.
3. Configure la variable `CLASSPATH` para que incluya la ubicación donde está el controlador JDBC. (La aplicación lee la variable de entorno `CLASSPATH` para localizar el controlador JDBC). Para acceder a la variable `CLASSPATH` y editarla, haga clic en **Inicio | Panel de control | Sistema | Opciones avanzadas | Variables de entorno**. En el cuadro de diálogo "Variables de entorno" que aparece seleccione la variable de entorno de usuario `CLASSPATH` o de sistema `CLASSPATH` y haga clic en el botón **Edición**. Añada la ruta de acceso del controlador JDBC a la variable `CLASSPATH`. Por ejemplo: `CLASSPATH=C:\jdbc\sqljdbc.jar; C:\jdbc\db2jcc.jar;`.
4. Cierre sesión y vuelva a iniciar sesión para que se activen los cambios.

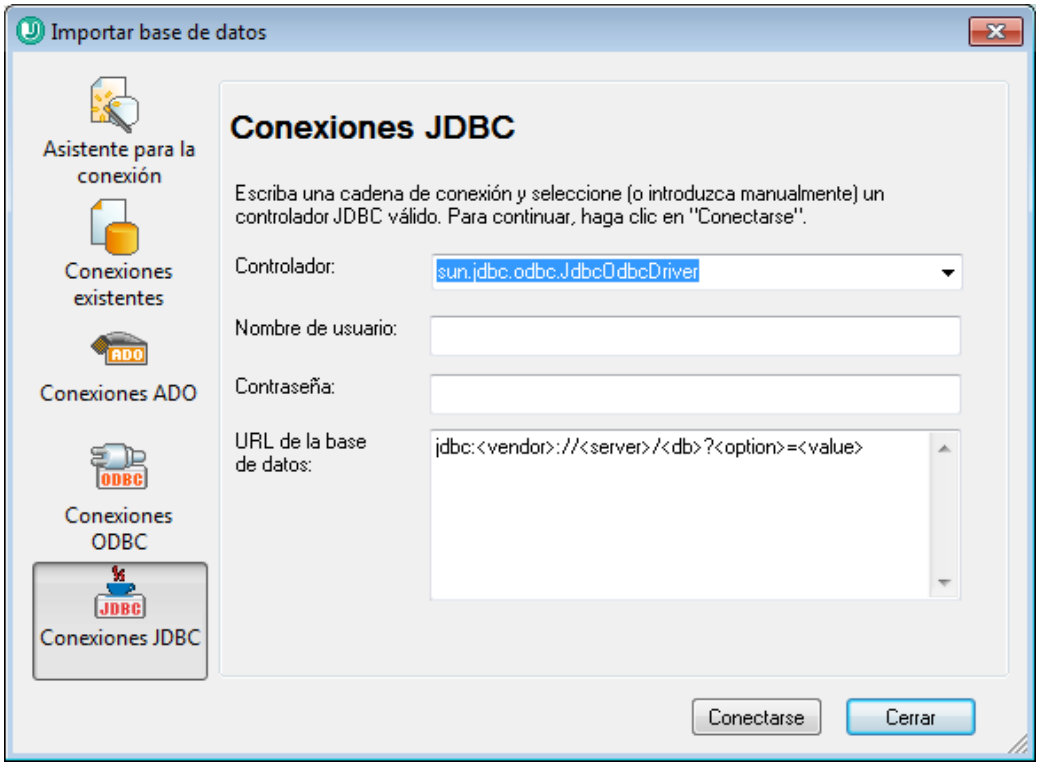
Nota: cuando se instala todo el cliente IBM DB2 u Oracle, los controladores JDBC del paquete pertinente se incluyen en la variable `CLASSPATH` automáticamente. Para más información lea el resto de este apartado.

Conexión por medio de controladores JDBC

Para conectarse a la BD usando una conexión JDBC seleccione **Proyecto | Importar base de datos SQL**:

1. Seleccione **Conexiones JDBC** en el cuadro de diálogo "Importar base de datos". Aparece el panel **Conexiones JDBC** (*imagen siguiente*).

 © 2014 Altova GmbH



2. Seleccione el controlador JDBC de la lista desplegable *Controlador:* (la lista enumera todos los controladores detectados, es decir, los de la variable de entorno `CLASSPATH`). Escriba la URL de la BD, el nombre de usuario y la contraseña. A continuación ofrecemos la sintaxis de direcciones URL para las bases de datos más utilizadas (cada una con un ejemplo).

Oracle

```
jdbc:oracle:thin:[usuario/contraseña]@//[host][:puerto]/SID
```

```
jdbc:oracle:thin:@//abcd234/ORA11
```

IBM DB2

```
jdbc:db2://nombre_host:puerto/nombreBD
```

```
jdbc:db2://MiDB2:50000/boz
```

MySQL

```
jdbc:mysql://nombre_host:puerto/nombreBD
```

```
jdbc:mysql://MiDB2:3306/moz
```

MSSQL

```
jdbc:sqlserver:// host:puerto;databasename=nombre;user=usuario;password=contraseña
```

```
jdbc:sqlserver://abcd38:1433;databasename=coz
```

```
jdbc:sqlserver://Q5;DatabaseName=coz;SelectMethod=Cursor
```

PostgreSQL	`jdbc:postgresql://host:puerto/baseDatos`
	`jdbc:postgresql://abc993:5432/qanoz`
Sybase	`jdbc:sybase:Tds:host:puerto/nombreBD`
	`jdbc:sybase:Tds:abc12:2048/QUE`

3. Haga clic en el botón **Conectarse** para establecer la conexión. Se establece la conexión y aparece un cuadro de diálogo donde puede seleccionar los datos de la BD que desea usar.

Instrucciones paso a paso para bases de datos MSSQL, MySQL, PostGre y otras BD no XML

Estas son las instrucciones para conectarse a bases de datos no XML a través de JDBC:

1. Archivos JAR JDBC (archivos de controladores): copie los archivos a una carpeta local. Después añada la ruta de acceso y el nombre de archivo a la variable de Windows `CLASSPATH`. Por ejemplo: `C:\jdbc\sqljdbc.jar; C:\jdbc\db2jcc.jar;`.
2. Cierre sesión y vuelva a iniciar sesión para que se activen los cambios realizados en `CLASSPATH`.
3. Inicie UModel y abra el cuadro de diálogo "Importar base de datos".
4. En el panel **Conexiones JDBC** los controladores JDBC detectados se enumeran en la lista desplegable del cuadro combinado *Controlador*. Si la lista está vacía, compruebe que el archivo `altovadb.jar` está en la carpeta: `C:\Archivos de programa\Altova\Common2015\jar`.
5. Conéctese a la BD siguiendo las instrucciones dadas más arriba.

Instrucciones paso a paso para bases de datos Oracle

Estas son las instrucciones para conectarse a una BD Oracle a través de JDBC. Si no necesita las características XML y XDB de la BD Oracle, siga las instrucciones del párrafo anterior para BD no XML. La carpeta de instalación del cliente de Oracle se indica con el marcador de posición: `%ORACLE_HOME%`.

1. Instale el software cliente de Oracle con las características OCI y ODBC habilitadas. Si ya tiene instalado un cliente de Oracle, compruebe si estos dos archivos JAR están en el equipo:

   ```
   %ORACLE_HOME%\LIB\xmlparserv2.jar
   %ORACLE_HOME%\RDBMS\jlib\xdb.jar
   ```

2. Añada estos archivos a la variable de entorno Windows `CLASSPATH`:

   ```
   %ORACLE_HOME%\jdbc\lib\ojdbc6.jar
   %ORACLE_HOME%\LIB\xmlparserv2.jar
   %ORACLE_HOME%\RDBMS\jlib\xdb.jar
   ```

3. Cierre sesión y vuelva a iniciar sesión para que se activen los cambios realizados en `CLASSPATH`.
4. Inicie UModel y abra el cuadro de diálogo "Importar base de datos".
5. En el panel **Conexiones JDCB** los controladores JDCB detectados se enumeran en la

 © 2014 Altova GmbH

lista desplegable del cuadro combinado *Controlador*. Si la lista está vacía, compruebe que el archivo `altovadb.jar` está presente en la carpeta: `C:\Archivos de programa\Altova\Common2015\jar`.

6. Conéctese a la BD siguiendo las instrucciones dadas al principio de este apartado.

Instrucciones paso a paso para bases de datos IBM DB2

Estas son las instrucciones para conectarse a una BD IBM DB2 a través de JDBC.

1. Si ya tienen instalado un cliente IBM DB2, no es necesario hacer nada porque la variable `CLASSPATH` ya se configuró durante el proceso de instalación del cliente.
2. Si no tiene instalado el cliente IBM DB2, añada los archivos JAR del controlador JDBC para IBM DB2 `db2jcc.jar` y `db2jcc_license_cu.jar` a la variable de entorno Windows `CLASSPATH`. Cierre sesión y vuelva a iniciar sesión para que se activen los cambios realizados en `CLASSPATH`.
3. Inicie UModel y abra el cuadro de diálogo "Importar base de datos".
4. En el panel **Conexiones JDBC** los controladores JDBC detectados se enumeran en la lista desplegable del cuadro combinado *Controlador*. Si la lista está vacía, compruebe que el archivo `altovadb.jar` está en la carpeta: `C:\Archivos de programa\Altova\Common2015\jar`.
5. Conéctese a la BD siguiendo las instrucciones dadas más arriba.

Nota: si se conecta a una BD a través de JDBC y los controladores no devuelven suficiente información, no se pueden editar los datos de las tablas que no tengan una clave principal.

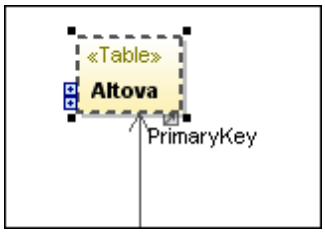
Si se conecta usando JDBC y utiliza una instrucción SELECT con el comando **Agregar/Quitar tablas para recuperar datos**, la instrucción SELECT no puede incluir el punto y coma final.

Ver elementos de modelado de BD

- Las **tablas** del perfil de datos UML se representan en forma de clases con el estereotipo **`<<Table>>`**.
- Las **columnas** se representan en forma de propiedades/atributos de la clase **`<<Table>>`**.
- Los **comportamientos** (es decir, las operaciones) se representan en forma de clases estereotipadas (p. ej. **`<<index>>`**, **`<<Check constraints>>`**, etc.)
- Las **relaciones** se representan en forma de asociaciones entre las tablas.
-

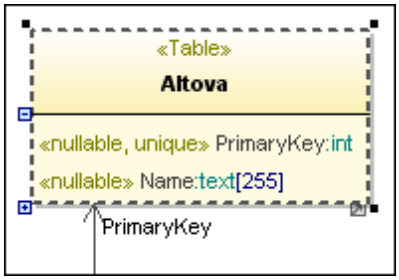
Para ver las propiedades de tabla (usando la BD de ejemplo Altova.mdb):

1. En la tabla **`Altova`** haga clic en el botón de expansión superior izquierdo.



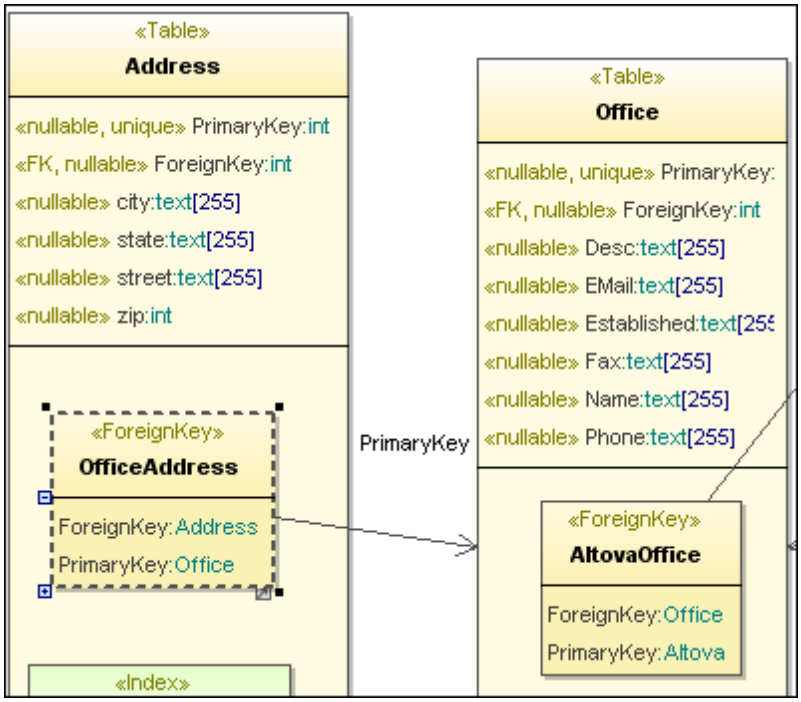
Esto abre el compartimiento de propiedades de la clase **`Table Altova`** y desvela las

columnas **PrimaryKey** y **Name**.



Clases ForeignKey

1. En la tabla **Address** haga clic en el botón de expansión superior.
   Las dos primeras columnas/campos son las entradas de clave principal y clave foránea. El resto son los campos de **Address**.



La clase **<<ForeignKey>> OfficeAddress** define las relaciones de clave principal/ foránea que existen entre las tablas.

- La entrada **ForeignKey**:***Address*** estipula que hay una columna, llamada `ForeignKey`, en la tabla propietaria **Address** que hace referencia a una columna foránea.
- La entrada **PrimaryKey**:***Office*** estipula que a la columna `PrimaryKey` de la tabla **Office** se le hace referencia como clave foránea desde la columna anterior (**(Address) ForeignKey**).

Observe que entre las tablas **Address** y **Office** existe una flecha de asociación (con la etiqueta `PrimaryKey`) que sirve para señalar la relación existente entre dichas tablas.

Cuando añada clases **ForeignKey** compruebe que:

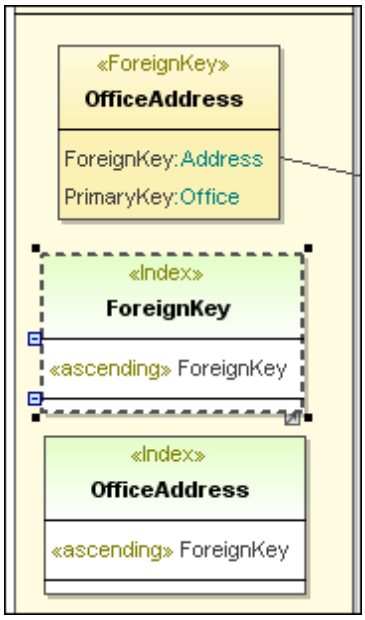
 © 2014 Altova GmbH

- La **primera** entrada/columna hace referencia a la tabla **propietaria** (en este caso, `Address`).
- La **segunda** entrada/columna hace referencia a la tabla **foránea** (en este caso, `Office`).

Índices

Las clases `Index` muestran los índices que hay definidos para cada tabla. La tabla `Address` del ejemplo tiene dos índices: `ForeignKey` y `OfficeAddress`.

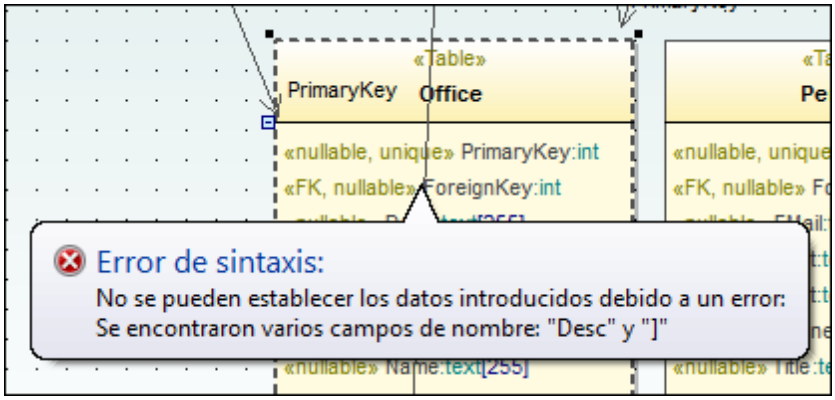
- El índice **ForeignKey** es la clave foránea de la tabla `Address` y tiene un orden ascendente.
- El índice **OfficeAddress** es la clave foránea de la clase `OfficeAddress` y también tiene un orden ascendente.



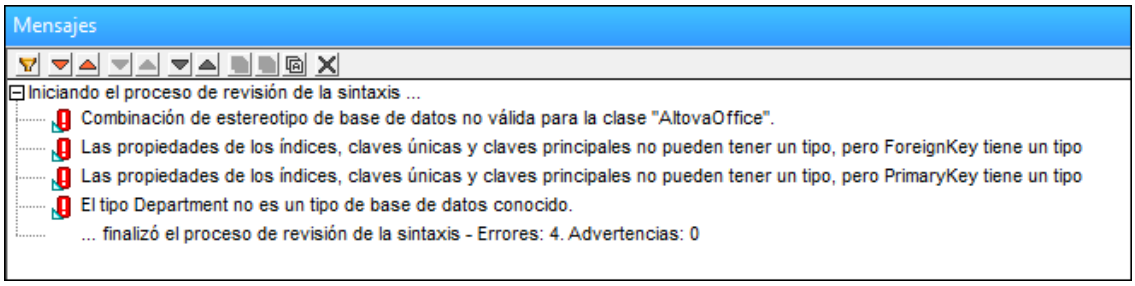
Nota: por cada clase `ForeignKey` puede definir más de una clave principal/foránea. También puede definir más de un índice en cada tabla.

Revisión sintáctica:

Cuando se añade información estructural, UModel realiza automáticamente una revisión sintáctica y, si es necesario, emite mensajes de error automáticamente.



Si quiere revisar la sintaxis de todo el proyecto, haga clic en el comando de menú **Proyecto | Revisar la sintaxis del proyecto**.



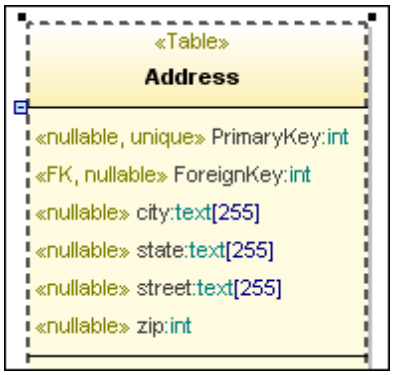
En la ventana Mensajes aparecen errores y advertencias sobre la sintaxis del proyecto.

Agregar columnas a una tabla

En UModel puede añadir elementos de modelado de BD y sincronizar el modelo con una BD. Por ejemplo, imagine que necesita añadir un campo para el código postal en la tabla **`Address`**.

Para añadir una columna/un campo en una tabla:

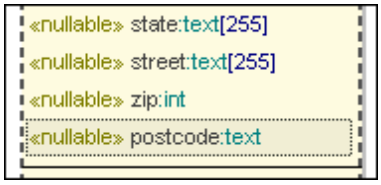
1. Haga clic con el botón secundario en la tabla **`Address`** y seleccione **Nuevo/a | Columna**.



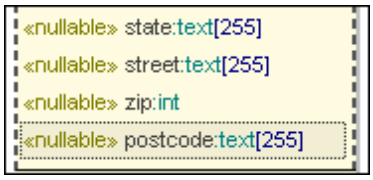
2. Reemplace el texto `Columna1` con el nombre del campo (es decir, `postcode`), inserte dos puntos (`:`) y seleccione el tipo de datos (p. ej. `text`).

3. Haga clic entre los corchetes angulares, escriba la letra `n` y seleccione **nullable** en la lista emergente de finalización automática.

 © 2014 Altova GmbH

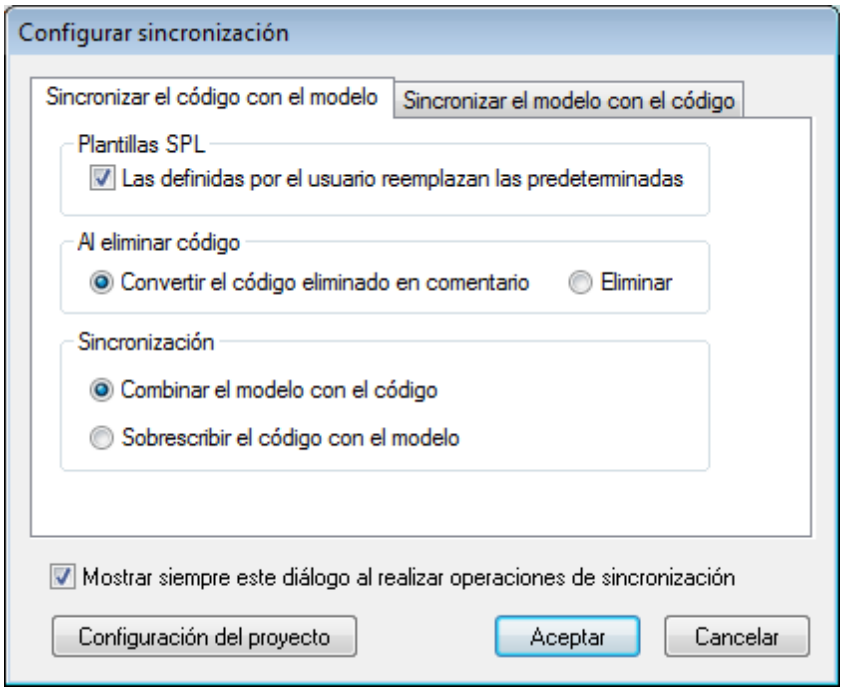


4. Inserte un corchete de apertura (`[`) y seleccione la longitud de texto que debe tener el campo (p. ej. `255`). Pulse **Entrar** para confirmar.

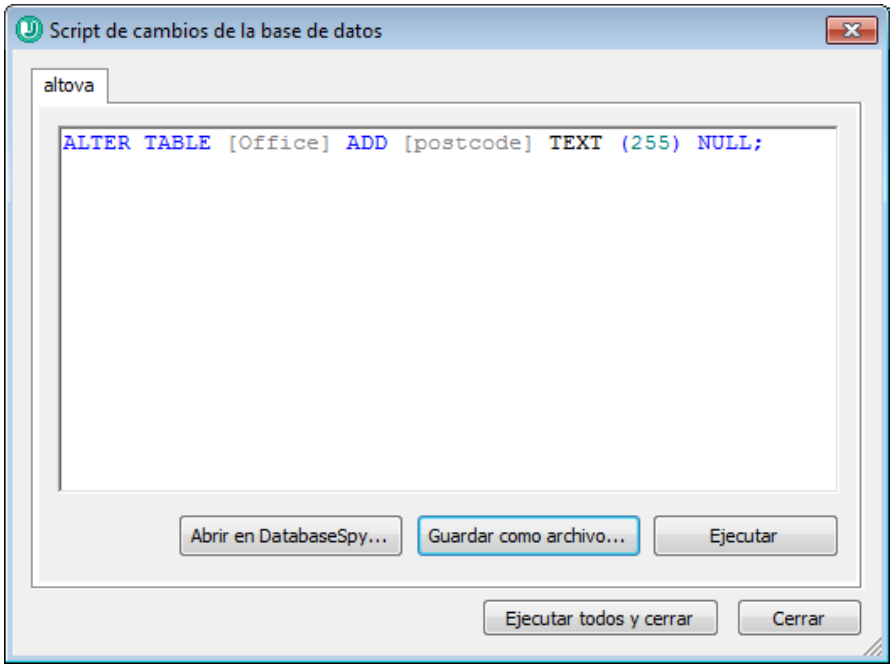


Para sincronizar el modelo actualizado con la base de datos (es decir, con el código):

1. Seleccione el comando de menú **Proyecto | Combinar el código de programa con el proyecto de UModel** para actualizar la BD con los cambios realizados en el proyecto de UModel.

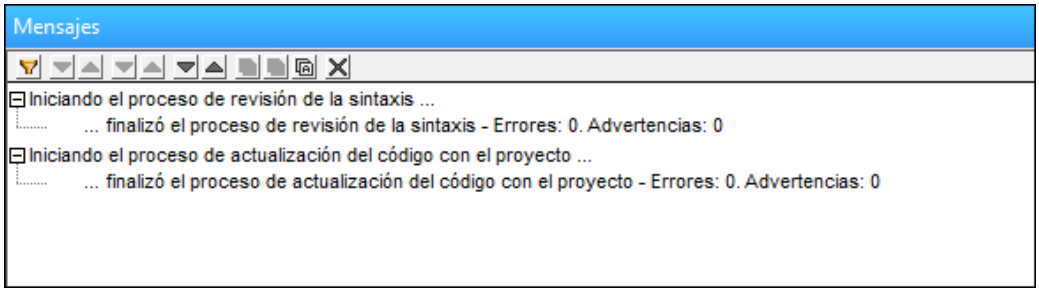


2. Si las opciones de configuración que aparecen son las correctas, haga clic en **Aceptar** para continuar.
   Aparece el cuadro de diálogo "Script de cambios de la base de datos", que muestra los comandos necesarios para actualizar la BD.

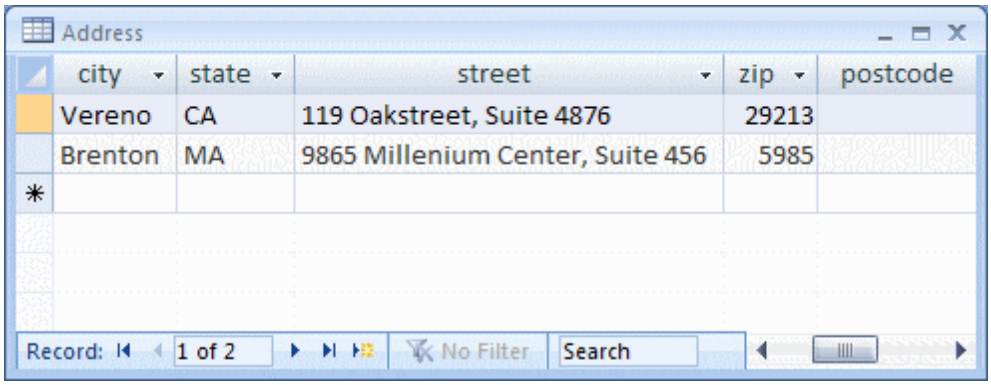


3. Si quiere ejecutar el código inmediatamente, haga clic en el botón **Ejecutar**.
   Aparece un aviso con detalles del proceso de ejecución de la actualización de la BD.

   Mire la ventana Mensajes para obtener información sobre la revisión sintáctica y sobre la actualización del código.



Nota: es necesario ejecutar el script de cambios para que el modelo y la BD estén sincronizados. El siguiente proceso de sincronización tendrá lugar como si se hubiese ejecutado el script de cambios. Si abre la BD podrá ver que se añadió la columna `postcode`.



 © 2014 Altova GmbH

Para eliminar una columna:

También puede usar el método que acabamos de describir para eliminar columnas de la BD.

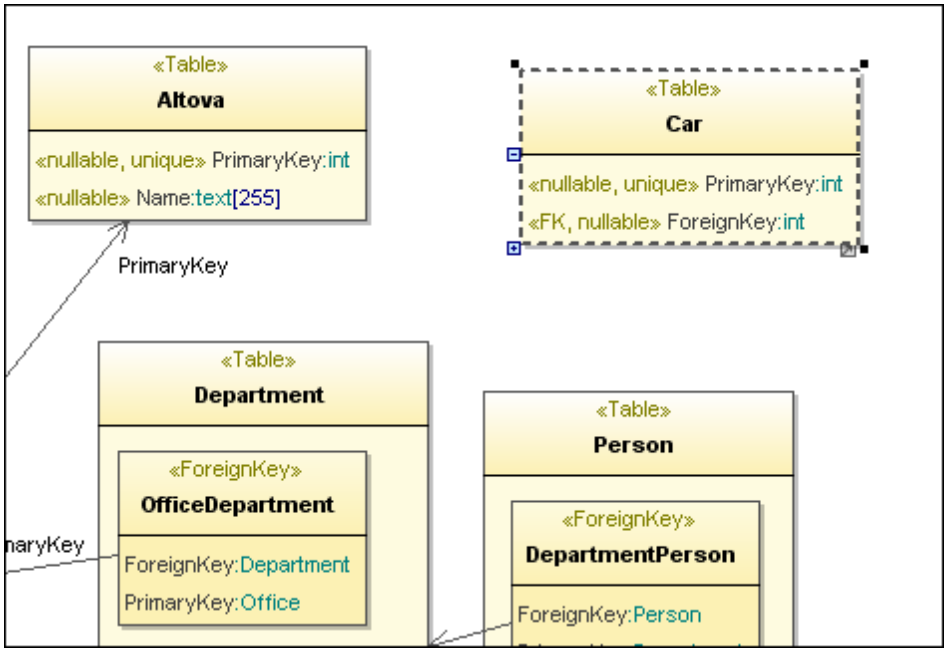
1. Seleccione la operación/columna que desea eliminar y elimínela de la tabla.
2. Sincronice el modelo con la BD.
3. El script de cambios genera un comando `DROP COLUMN`.

Nota: si ejecuta este comando en una BD, se eliminará la columna y todos sus datos.

Agregar tablas nuevas a un modelo / a una BD

Para añadir una tabla nueva al modelo/a la base de datos:

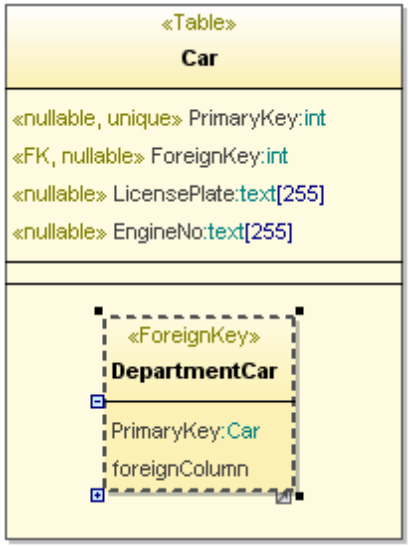
1. Haga clic con el botón secundario en el diagrama de BD y seleccione **Nuevo/a | Tabla**. Escriba `Car` como nombre de la tabla.
2. Pulse **F7** y después pulse la tecla de dirección izquierda para poner el cursor totalmente a la izquierda.
3. Ponga el cursor entre los dos corchetes angulares y escriba `<<nullable, unique>>`. Aparece la lista emergente de finalización automática. Seleccione una entrada y pulse **Entrar**.
4. Mueva el cursor hasta `Propiedad1` y sobrescriba el nombre de la propiedad con `PrimaryKey:int`. Después pulse **Entrar** para confirmar.



5. Pulse **F7** y use el mismo método para insertar **<<FK, nullable>>ForeignKey:int**. Con esto terminamos de definir las claves de la tabla.
6. Ahora pulse **F7** e introduzca los campos/las columnas **LicensePlate** y **EngineNo** tal y como aparecen en la siguiente imagen. Cuando inserte la longitud del texto (insertando el corchete de apertura `[`), observe que el corchete de cierre se inserta automáticamente. Después de insertar la longitud de texto `255`, pulse **Entrar** para confirmar.

Agregar las clases de clave foránea e índice ForeignKey y Index:

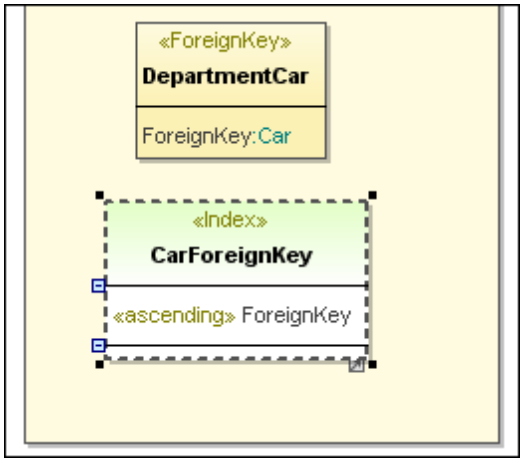
1. Haga clic con el botón secundario en la tabla `Car` y elija **Nuevo/a | ClaveForánea** y escriba el nombre de la clave `DepartmentCar`.



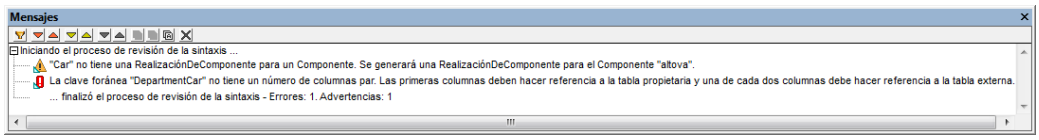
2. Haga doble clic en la entrada/propiedad superior (`PrimaryKey:Car`) de la clase y póngale el nombre `ForeignKey:Car`.

3. Elimine la segunda entrada (`foreignColumn`), que está situada debajo.
4. Haga clic con el botón secundario en la tabla `Car` y elija **Nuevo/a | Índice**. Escriba el nombre de índice `CarForeignKey`.
5. Haga doble clic en la entrada `IndexedColumn` (que se insertó automáticamente) y póngale el nombre `ForeignKey`.
6. Ponga el cursor entre los corchetes angulares y seleccione `<<ascending>>` en la lista de finalización automática.

 © 2014 Altova GmbH



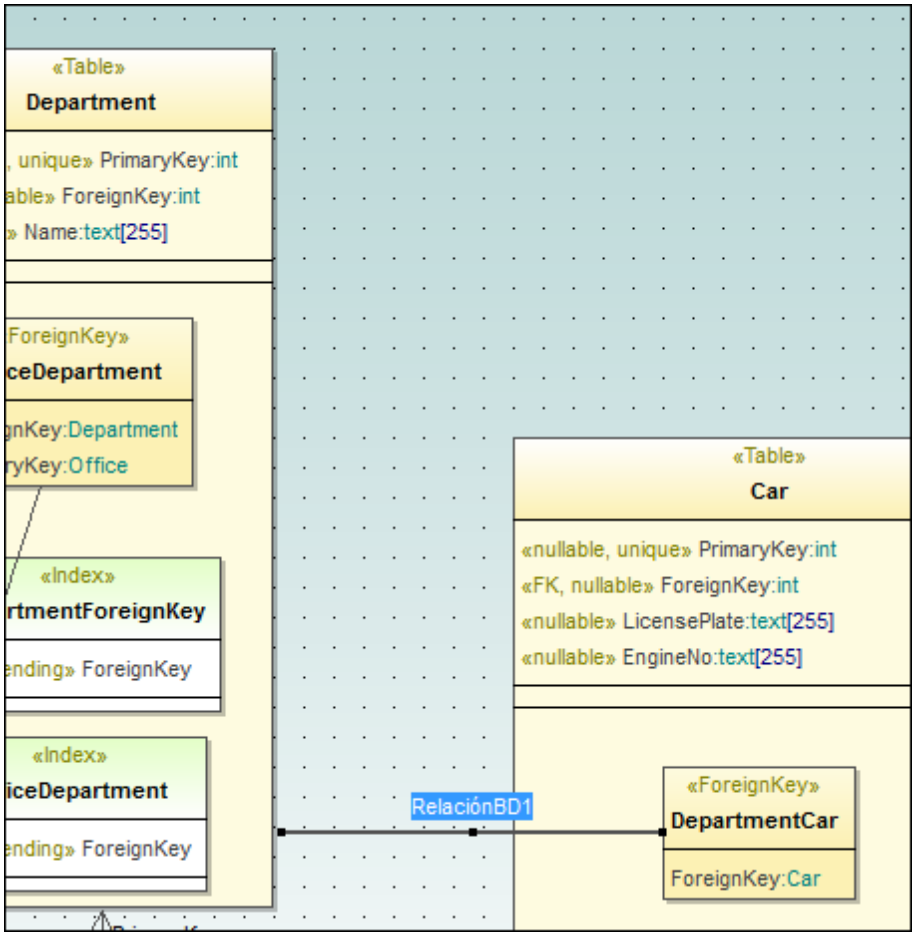
7. Seleccione el comando de menú **Proyecto | Revisar la sintaxis del proyecto** para ver si se cumplen todas las reglas de la BD.



El motivo del error que aparece en la ventana Mensajes es que en la clase `DepartmentCar` solamente se creó una operación/columna, pero se necesitan dos. En el siguiente paso añadiremos la operación/columna que falta.

Para crear relaciones entre tablas:

1. Haga clic en el icono **Asociación de relación entre bases de datos** de la barra de herramientas.
2. Haga clic en la clase `DepartmentCar` de la tabla `Car` y arrastre la línea hasta la tabla `Department`.

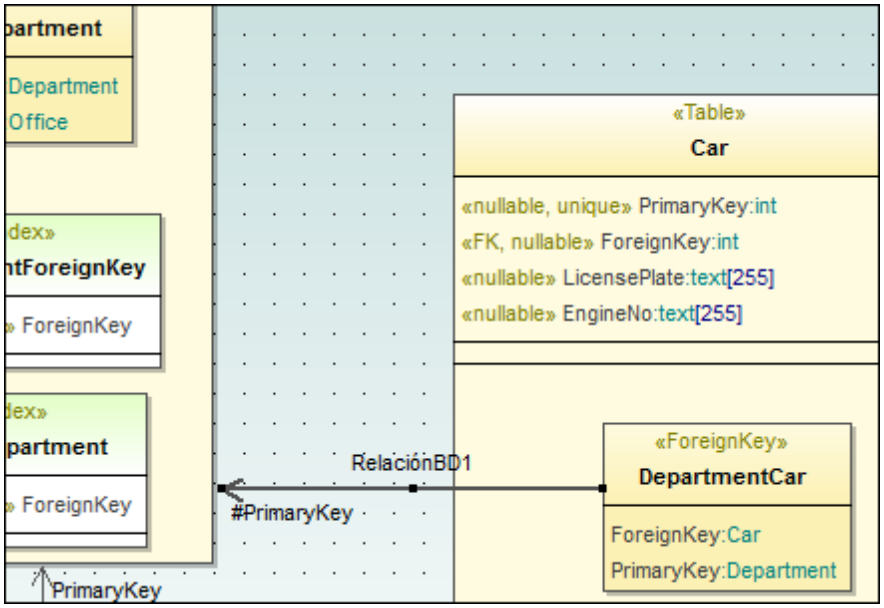


Esto crea una asociación entre las dos tablas.

3. Seleccione la asociación y, en el panel Propiedades, haga clic en el campo `A:nombre` y escriba `PrimaryKey`.
4. Ahora haga clic en el cuadro combinado `A:memberEndKind` y seleccione `memberEnd`.

 © 2014 Altova GmbH

Observe que como resultado se añade una propiedad nueva (`PrimaryKey:Department`) a la clase `ForeignKey DepartmentCar`.



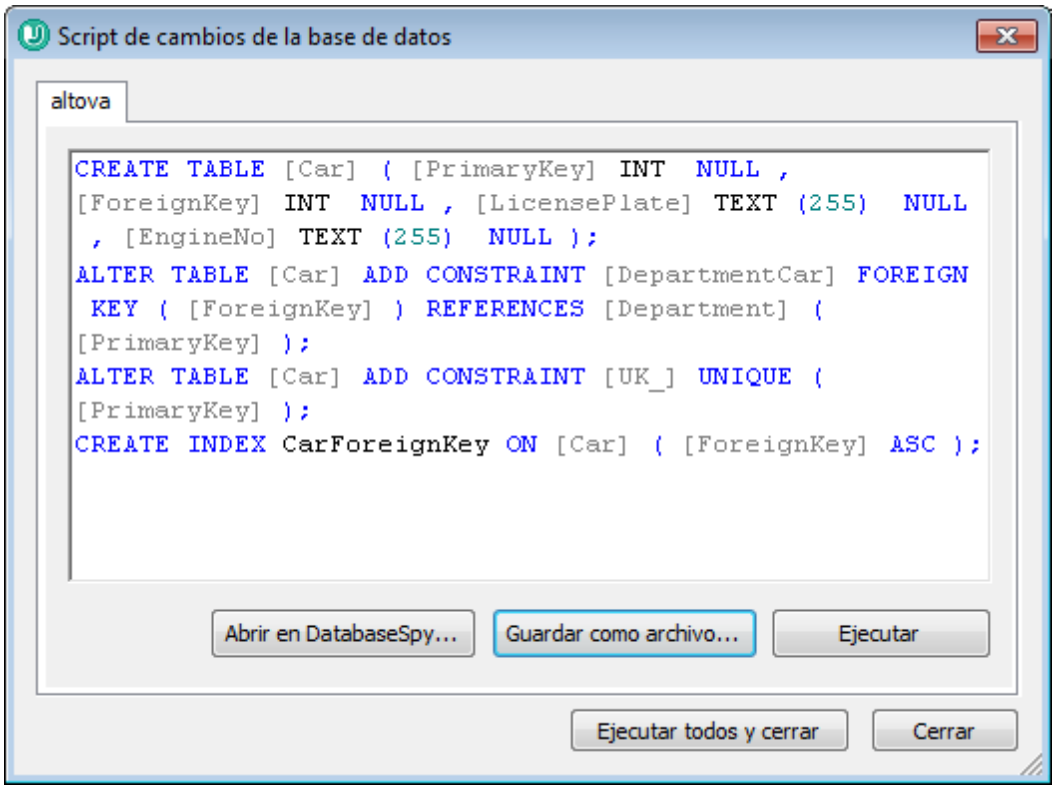
5. Ahora seleccione el comando de menú **Proyecto | Revisar la sintaxis del proyecto**. La revisión ya solo emite una advertencia.

Para sincronizar el modelo con la base de datos:

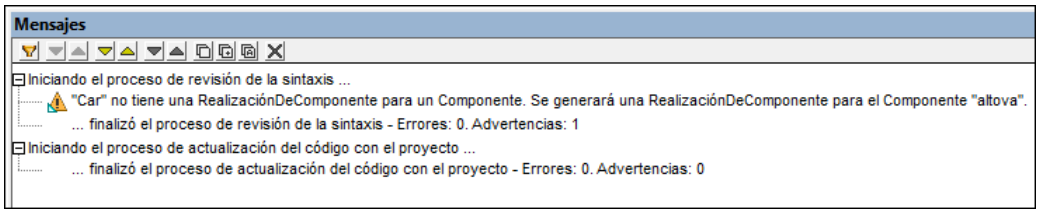
1. Seleccione el comando de menú **Proyecto | Combinar el código de programa con el proyecto de UModel**.
   Esto abre el cuadro de diálogo "Configurar sincronización".
2. Si las opciones de configuración son correctas, haga clic en **Aceptar**.

Aparece un cuadro de diálogo con el script de cambios generado por UModel. Observe que el script de cambios no se puede editar en este cuadro de diálogo y que cada **pestaña** del cuadro de diálogo corresponde a una **base de datos**.

3. Haga clic en el botón **Ejecutar todos y cerrar**.

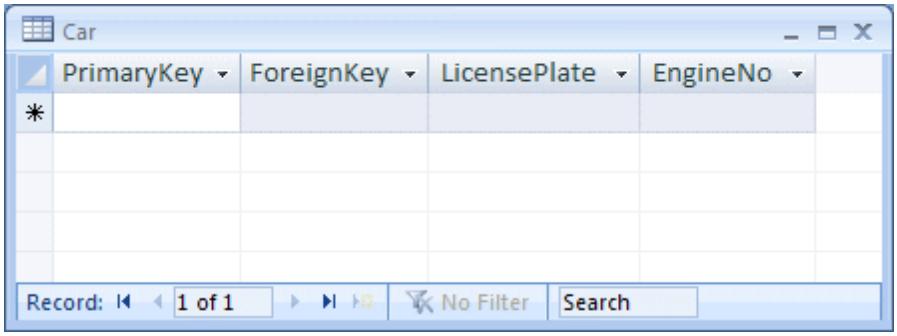


Mire la ventana Mensajes por si se detectaron errores o advertencias y haga clic en **Aceptar** para continuar.
La ventana Mensajes también ofrece información sobre el proceso de actualización en general.

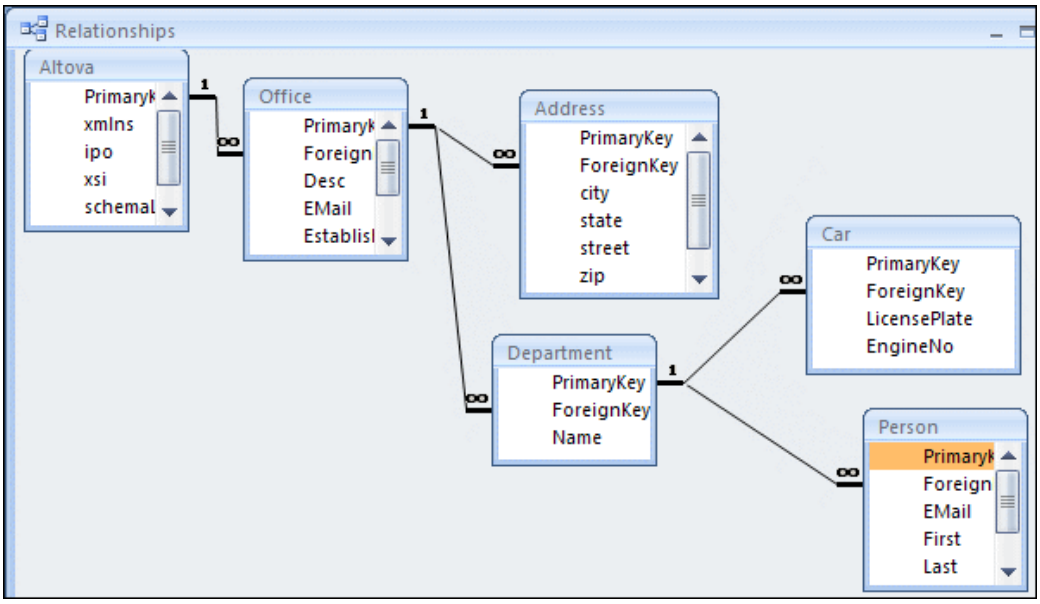


Ahora abra la BD y observe que ahora tiene una tabla nueva llamada `Car` y formada por cuatro columnas.

 © 2014 Altova GmbH



La estructura relacional de la BD se puede ver en la ventana Relaciones.



Altova UModel® 2015

XMI: intercambio de metadatos XML

11 XMI: intercambio de metadatos XML

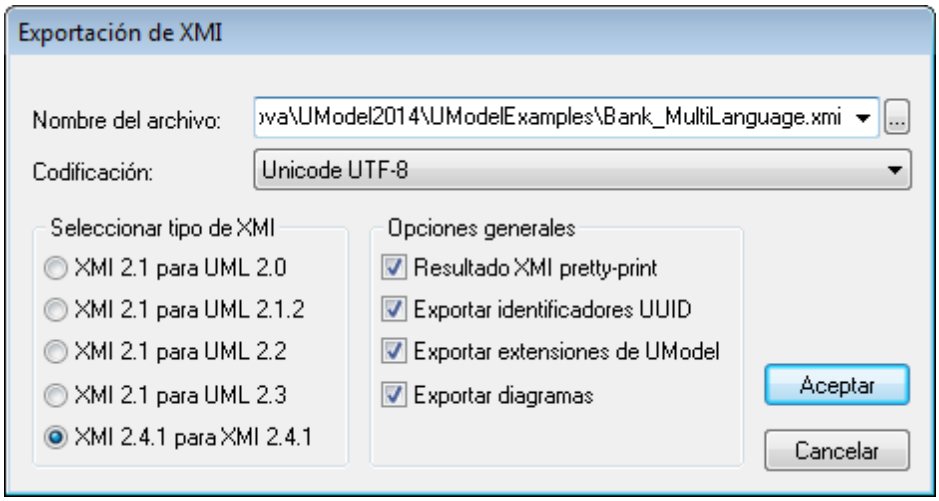
Sitio web de Altova: Intercambio de modelos UML mediante archivos XMI

UModel ofrece una función para exportar e importar datos XMI 2.4 para UML 2.0 / 2.1 / 2.1.1 y 2.1.2, 2.2, 2.3, 2.4. No utilice la función de exportación XMI para archivar proyectos de UModel. Archive los archivos de proyecto `*.ump`.

El comando **Archivo | Exportar a un archivo XMI** genera un archivo XMI a partir del proyecto de UModel. Por su parte, el comando **Archivo | Importar desde un archivo XMI** importa datos desde un archivo XMI generado con anterioridad.

El cuadro de diálogo "Exportación de XMI" sirve para seleccionar el formato XMI en el que se exporta el archivo (p. ej. XMI para UML 2.0/2.1.1). Durante el proceso de exportación también se exportan los archivos incluidos, incluso los que están *incluidos mediante referencia*.

Importante: si tiene pensado reimportar el código XMI al proyecto de UModel, asegúrese de marcar la casilla *Exportar extensiones de UModel*.



Resultado XMI pretty-print

Marque esta casilla para aplicar sangría a las etiquetas XML del archivo XMI de destino e aplicarle los retornos de carro/saltos de línea correspondientes.

Exportar identificadores UUID

El estándar XMI define tres tipos de identificadores de elementos: ID, UUID y etiquetas.

- Los **ID** son identificadores únicos en el documento XMI y son compatibles con la mayoría de las herramientas UML. UModel exporta este tipo de identificadores por defecto (es decir, no hace falta marcar ninguna casilla).
- Los **UUID** son identificadores universales únicos que sirven para asignar a cada elemento una identificación global única. Estos identificadores son únicos a nivel global (es decir, no solo en el documento XMI). Marque la casilla *exportar identificadores UUID* para generar identificadores UUID.

 © 2014 Altova GmbH

- Los UUID se almacenan en el formato estándar UUID/GUID (p. ej. "6B29FC40-CA47-1067-B31D-00DD010662DA", "550e8400-e29b-41d4-a716-446655440000",...)
- UModel admite el uso de etiquetas para identificar elementos en XMI.

Nota: el proceso de importación XMI es compatible automáticamente con los identificadores ID y UUID.

Exportar extensiones de UModel

XMI define un mecanismo que permite a cada aplicación exportar sus propias extensiones a la especificación UML. Sin embargo, algunas herramientas UML solo son capaces de importar los datos UML estándar (pasando por alto las extensiones de UModel). Sin embargo, cuando importe datos a UModel estos datos de extensión estarán disponibles.

Algunos datos de UModel, como los nombres de archivo de clases o los colores de los elementos, no forman parte de la especificación UML y, por tanto, deben eliminarse en XMI o guardarse en *extensiones*. Si se exportan como extensiones y se vuelven a importar, estos nombres de archivo y colores se importan tal y como se definieron. Si no usa las extensiones para el proceso de exportación, estos datos de UModel se perderán.

Al importar un documento XMI, UModel detecta el formato y genera un modelo automáticamente.

Exportar diagramas

Marque esta casilla para exportar los diagramas de UModel como *extensiones* en el archivo XMI. Para poder guardar los diagramas como extensiones debe marcar también la casilla *Exportar extensiones de UModel.*

Altova UModel® 2015

Trabajo en equipo con UModel

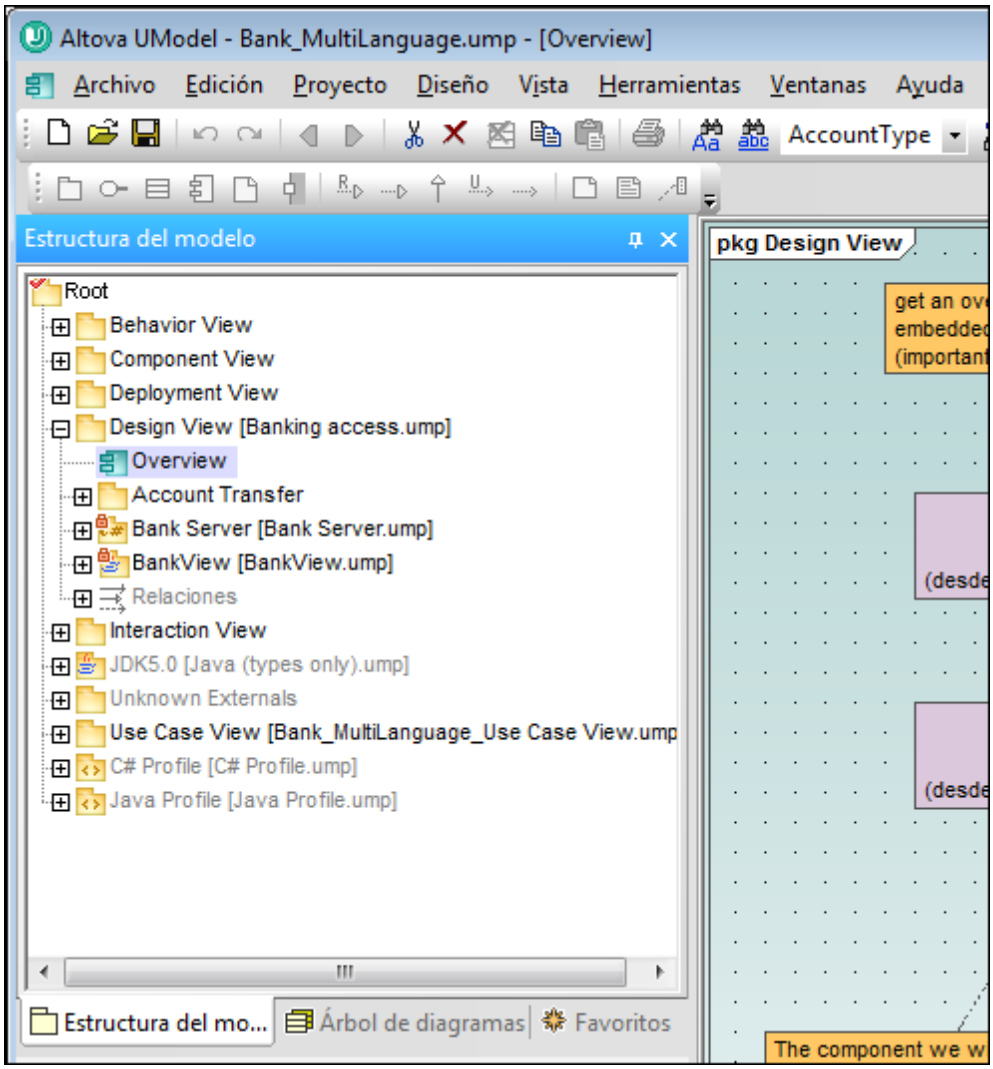
12 Trabajo en equipo con UModel

Los proyectos de UModel se pueden dividir en varios subproyectos para que varios programadores puedan editar partes diferentes del proyecto por separado. Los subproyectos pueden añadirse al sistema de control de versiones utilizado por el equipo.

El proyecto de nivel superior (es decir, el proyecto en el que se pueden incluir los subproyectos) recibe el nombre de *proyecto principal*. Los subproyectos se crean al nivel de los paquetes como archivos de proyecto UModel independientes y tienen la extensión de archivo `*.ump`.

Puede crear e incluir subproyectos de dos maneras distintas:

- un subproyecto puede ser **editable** dentro del proyecto **principal** y a nivel de subproyecto.
- un subproyecto puede ser de **sólo lectura** dentro del proyecto **principal** y editable a nivel de subproyecto.



Los subproyectos pueden estar organizados de forma jerárquica, con una estructura plana o de ambas formas. Esto permite, en teoría, dividir cada paquete de un proyecto principal en varios subproyectos editables o de sólo lectura.

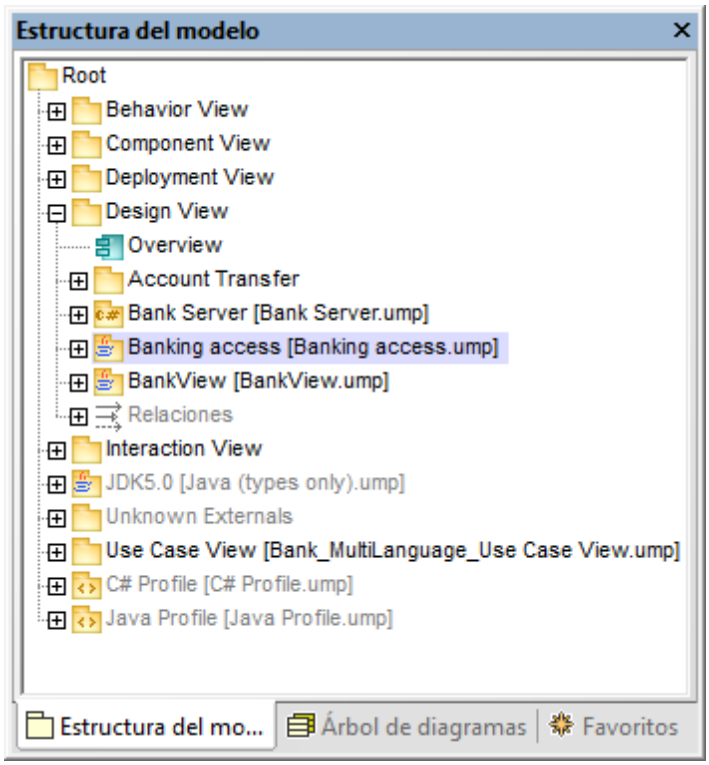
 © 2014 Altova GmbH

Durante el proceso de ingeniería de código todos los componentes subordinados de un subproyecto se tienen en cuenta. Es decir, para la ingeniería de código no hay diferencia alguna entre un archivo de proyecto sencillo y uno formado por varios subproyectos editables.

Esto también afecta a los diagramas UML, que se pueden editar a nivel de proyecto principal o de subproyecto.

12.1 Crear y editar subproyectos

A continuación usamos un ejemplo para explicar cómo se divide un proyecto en varios subproyectos. El ejemplo se sirve del archivo de muestra `Bank_MultiLanguage.ump`.

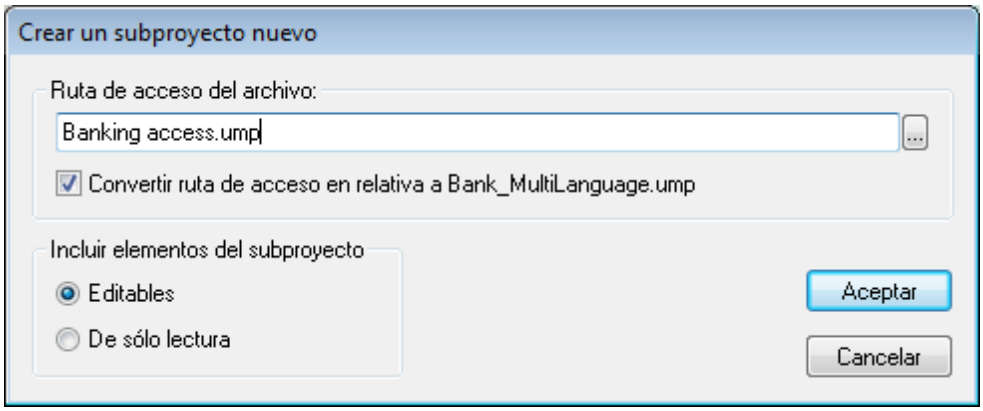


Nota: en UModel puede compartir paquetes y sus diagramas UML con varios proyectos. Los paquetes se pueden incluir en otros proyectos por medio de referencias o como copias. Para más información consulte el apartado Compartir paquetes y diagramas.

Para crear un archivo de subproyecto:

Puede crear subproyectos a partir de un proyecto principal o de otro subproyecto.

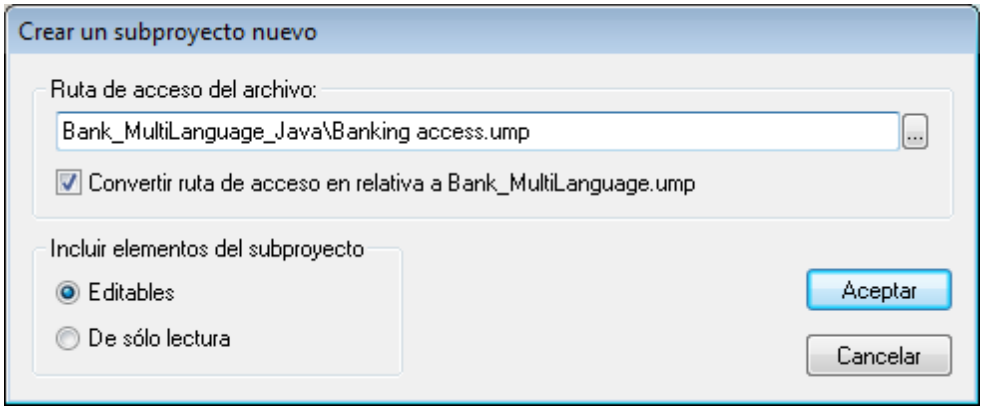
1. Haga clic con el botón secundario en un paquete (p. ej. **`Banking access`**) y seleccione **Subproyecto | Crear subproyecto nuevo**.



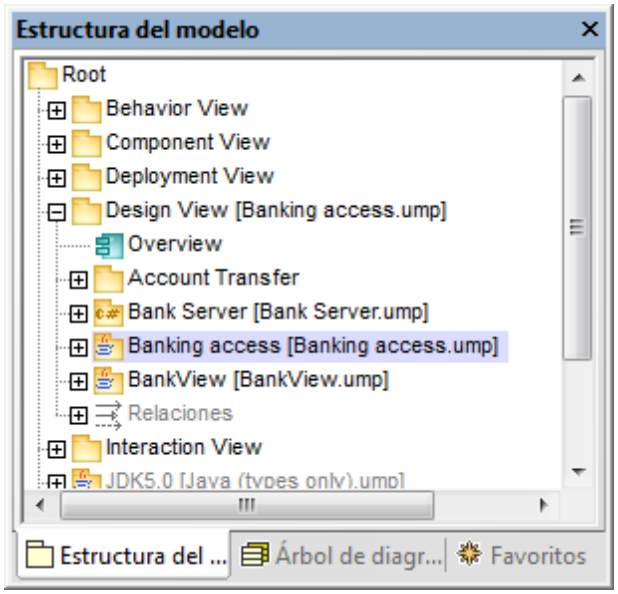
2. Haga clic en el botón **Examinar** y seleccione el subdirectorio

 © 2014 Altova GmbH

\Bank_MultiLanguage_Java.

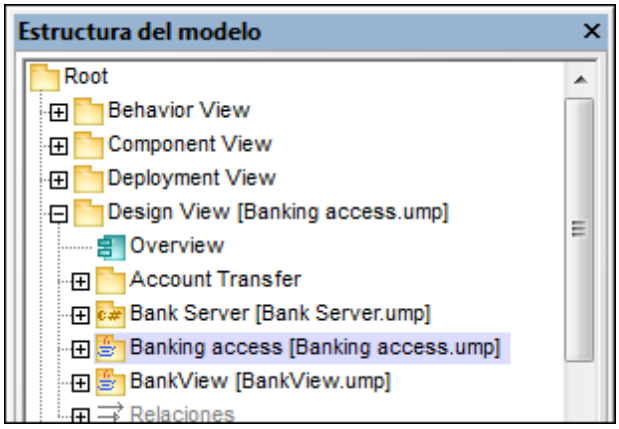


3. Seleccione el botón de opción *Editables* para poder editar el subproyecto desde el proyecto principal. (Si selecciona la opción *De sólo lectura*, el subproyecto no se podrá editar en el proyecto principal). Ahora haga clic en **Aceptar**.



El nombre del subproyecto aparece entre corchetes junto al nombre de paquete y el archivo `Banking access.ump` se guarda en la carpeta `UModelExamples \Bank_MultiLanguage_Java`.

Utilice el mismo método para crear un subproyecto de la carpeta **BankView**. El archivo `BankView.ump` se guarda en la carpeta `...\UModelExamples \Bank_MultiLanguage_Java\`.



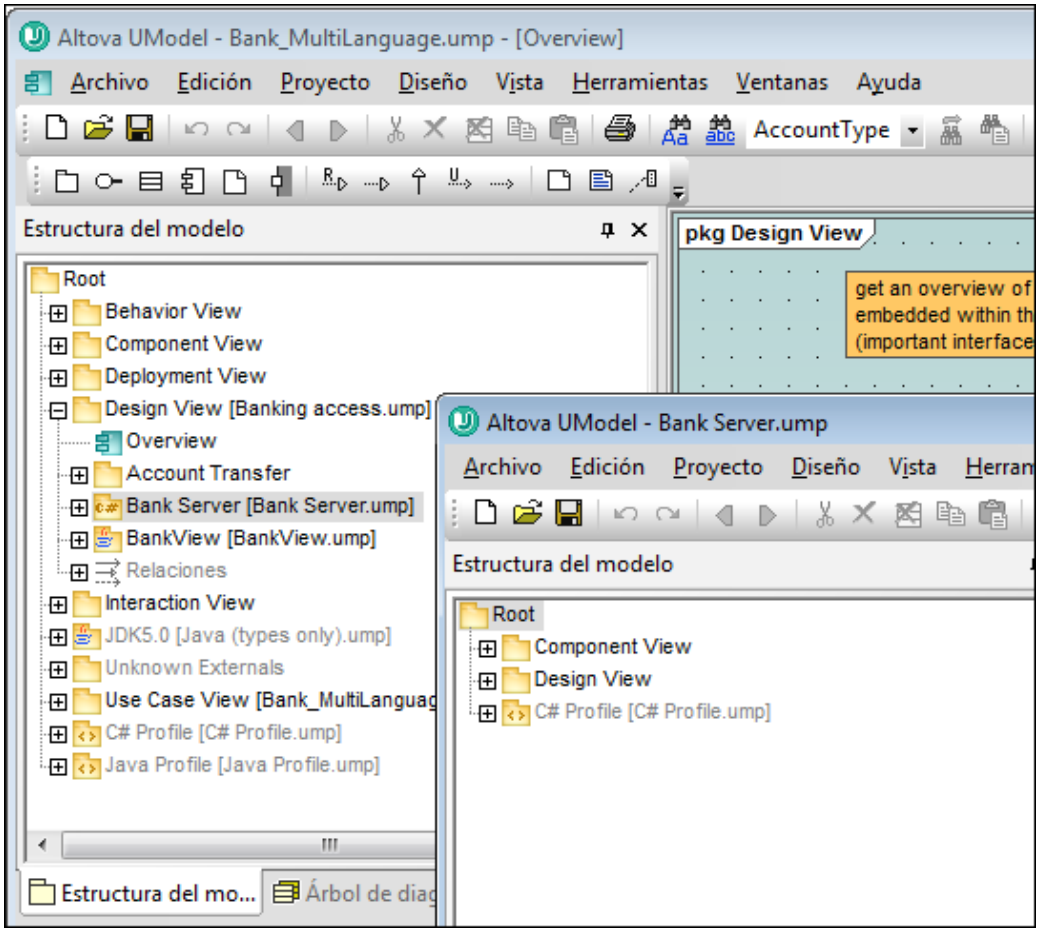
Nota: para cambiar la ruta de acceso del archivo de subproyecto haga clic con el botón secundario en el subproyecto y seleccione **Subproyecto | Editar la ruta de acceso del archivo**.

Para abrir y editar subproyectos:

Una vez creado el subproyecto, el archivo `*.ump` resultante se puede abrir y editar como si fuera un archivo de proyecto principal. Para que esto funcione no puede haber referencias sin resolver a otros elementos. UModel revisa automáticamente el archivo de subproyecto tanto cuando se crea como cuando se guarda.

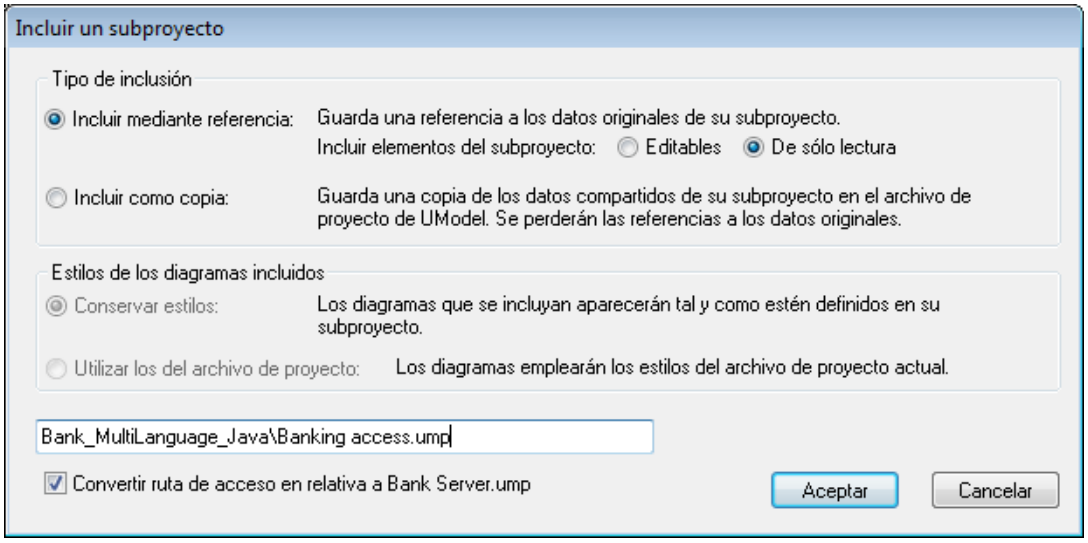
1. Haga clic con el botón secundario en el paquete de subproyecto (p. ej. `Bank Server.ump`) del proyecto principal y seleccione la opción **Subproyecto | Abrir en forma de proyecto**. Esto inicia otra instancia de UModel y abre el subproyecto como si fuera el proyecto principal. Las referencias no resueltas aparecen en la ventana *Mensajes*.

 © 2014 Altova GmbH



Para reutilizar subproyectos en un otros proyectos:
Los subproyectos en los que se dividió el proyecto principal se pueden usar en otros proyectos principales.

1. Abra el proyecto y después seleccione el comando de menú **Proyecto | Incluir un subproyecto**.
2. Haga clic en el botón **Examinar** y seleccione el archivo `*.ump` que desea incluir (p. ej. `Banking access.ump`).



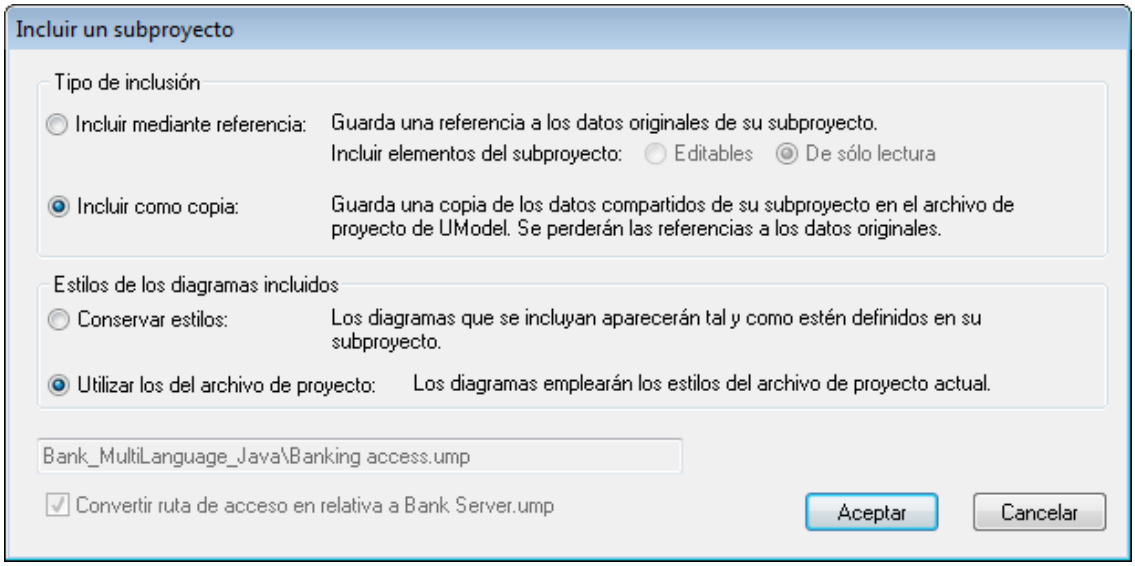
3. Elija cómo se añade el archivo de subproyecto (mediante referencia o como copia).

Guardar proyectos
Cuando se guarda el archivo de proyecto principal, también se guardan todos los subproyectos editables (es decir, se guardan todos los datos de los paquetes compartidos de los subproyectos).

Por tanto, no debería crear/agregar datos (componentes) fuera de la estructura compartida/del subproyecto, si el subproyecto se definió como editable en un proyecto principal. Si existen datos fuera de la estructura del subproyecto, UModel emite una advertencia en la ventana *Mensajes*.

Guardar subproyectos
Cuando se guardan subproyectos, se guardan también todas las referencias a subproyectos del mismo nivel y subproyectos secundarios. Por ejemplo, si tenemos los subproyectos del mismo nivel `sub1` y `sub2` y `sub1` utiliza elementos de `sub2`, entonces al guardar `sub1` se guardan automáticamente las referencias a `sub2`.

 © 2014 Altova GmbH



Si abrimos `sub1` como proyecto principal, se entiende como proyecto autónomo y se puede editar sin referencias al verdadero proyecto principal.

Para volver a integrar los subproyectos en el proyecto principal:

Los subproyectos se pueden volver a copiar dentro del proyecto principal. Si el subproyecto no contiene diagramas, la reintegración es inmediata. Por el contrario, si existen diagramas, UModel muestra un cuadro de diálogo.

1. Haga clic con el botón secundario en el subproyecto y seleccione la opción **Subproyecto | Incluir como copia**.
   Esto abre el cuadro de diálogo "Incluir un subproyecto", donde puede definir qué estilo de diagramas se deben usar al incluir el subproyecto.
2. Por último seleccione la opción de estilo y haga clic en **Aceptar**.

Altova UModel® 2015

Complemento de UModel para MS Visual Studio

13 Complemento de UModel para MS Visual Studio

UModel puede integrarse en el entorno IDE **Microsoft Visual Studio 2005/2008/2010/2012/2013**. Esto permite reunir las avanzadas funciones de modelado de UModel y el sofisticado entorno de desarrollo Visual Studio. Los requisitos para poder integrarlos son:

- Tener instalado Microsoft Visual Studio 2005/2008/2010/2012/2013.
- Tener instalado UModel Enterprise o Professional Edition.
- Descargar y ejecutar el **paquete de integración de UModel**. El paquete de integración se puede descargar del sitio web de Altova www.altova.com.

Para habilitar el complemento de UModel para Visual Studio:
Es posible que durante el proceso de instalación el complemento de UModel no se habilitara automáticamente.

1. Navegue hasta el directorio donde se instaló el ejecutable de Visual Studio (p. ej. `c:\Archivos de programa\Microsoft Visual Studio 8\Common7\IDE`).
2. Escriba este comando en la línea de comandos **`devenv.exe /setup`**.
3. Espere a que termine el proceso. Una vez terminado podrá usar la aplicación dentro de Visual Studio.

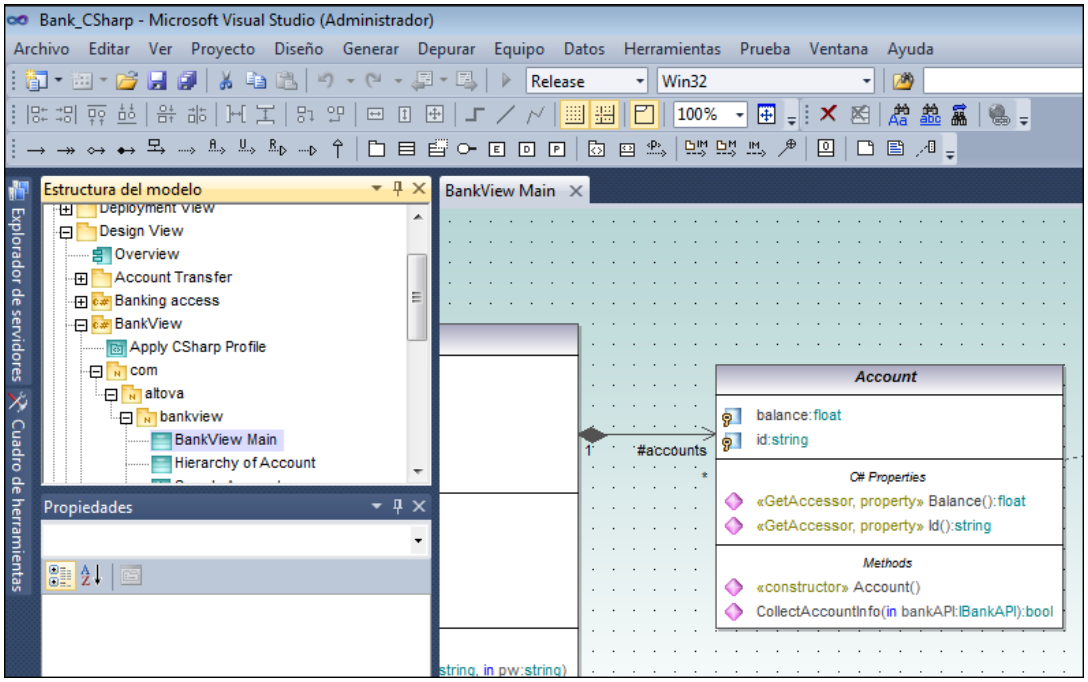
Sistemas de control de versiones:
El complemento de UModel para Visual Studio es compatible con las funciones y comandos de control de versiones disponibles en el entorno Visual Studio.

 © 2014 Altova GmbH

13.1 Abrir archivos de UModel en MS Visual Studio

Para abrir un archivo de UModel en Visual Studio:

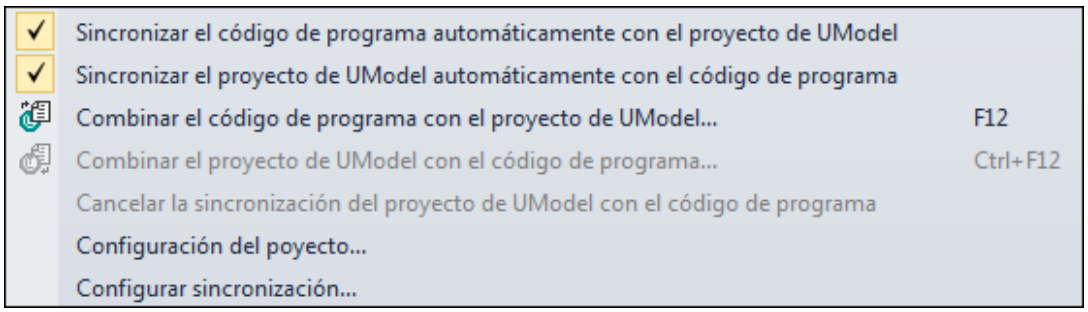
1. Seleccione la opción de menú **Archivo | Abrir | Proyecto o solución.**
2. Seleccione el archivo `Bank_CSharp.ump` de la carpeta `...\UModelExamples`.
3. En el panel Estructura del modelo haga doble clic en el icono del diagrama **`BankView`** para ver el diagrama de clases.



Nota: quizás sea necesario ajustar la posición de algunas ventanas. En el complemento para Visual Studio las ventanas están organizadas como en la versión independiente de UModel.

Para sincronizar el código de programa con el código del modelo en MS Visual Studio:

1. Seleccione la opción de menú **Proyecto | Proyecto de UModel**.
   Observe que el menú contiene dos elementos nuevos que se activan automáticamente: **Sincronizar el código de programa automáticamente** y **Sincronizar el proyecto de UModel automáticamente**.



Esto permite actualizar automáticamente el modelo y el código en cuanto se realizan

cambios en uno de los dos. Si deshabilita estas opciones, deberá generar el código de forma manual con el comando **Combinar el código de programa con el proyecto de UModel** (*imagen anterior*).

© 2014 Altova GmbH

13.2 Diferencias con la versión independiente

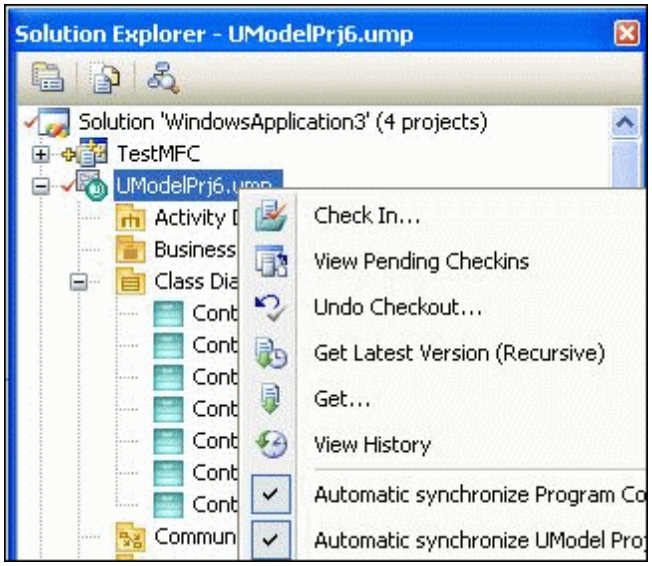
Requisitos:
Para trabajar con el complemento de UModel para Visual Studio, los archivos de código fuente deben formar parte de la misma solución que el modelo de UModel.

Por ahora en una solución puede haber un archivo de proyecto de UModel como máximo. Todos los proyectos de código C# o VB que formen parte de la solución estarán disponibles para sincronizarlos con el modelo.

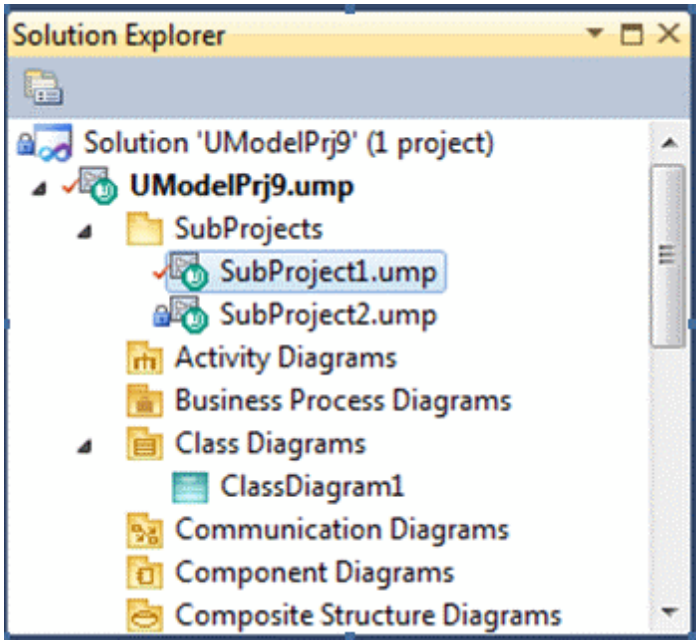
Los comandos de menú **Proyecto | Proyecto de UModel | Importar directorio de código fuente** y **Proyecto | Proyecto de UModel | Importar proyecto de código fuente** no permiten seleccionar C# ni VB.NET en el cuadro combinado *Lenguaje* del cuadro de diálogo "Importar directorio/proyecto de código fuente". El código C#/VB.NET se añade automáticamente en cuanto el proyecto C#/VB.NET se añade a la solución de Visual Studio.

El complemento de UModel para Visual Studio es compatible con las funciones nativas de control de versiones de Visual Studio. Los comandos de control de versiones para trabajar con archivos de código fuente están disponibles en los menús de Visual Studio, no en los menús propios de UModel.

Por ejemplo, al hacer clic con el botón secundario en un proyecto de UModel en la ventana Explorador de soluciones, el menú contextual incluye comandos de control de versiones.



Los proyectos de UModel se pueden dividir en subproyectos para que varios programadores puedan trabajar por separado en partes diferentes del proyecto. Cada subproyecto se puede agregar al sistema de control de código fuente. Los subproyectos aparecen en distintas carpetas en el Explorador de soluciones de Visual Studio.



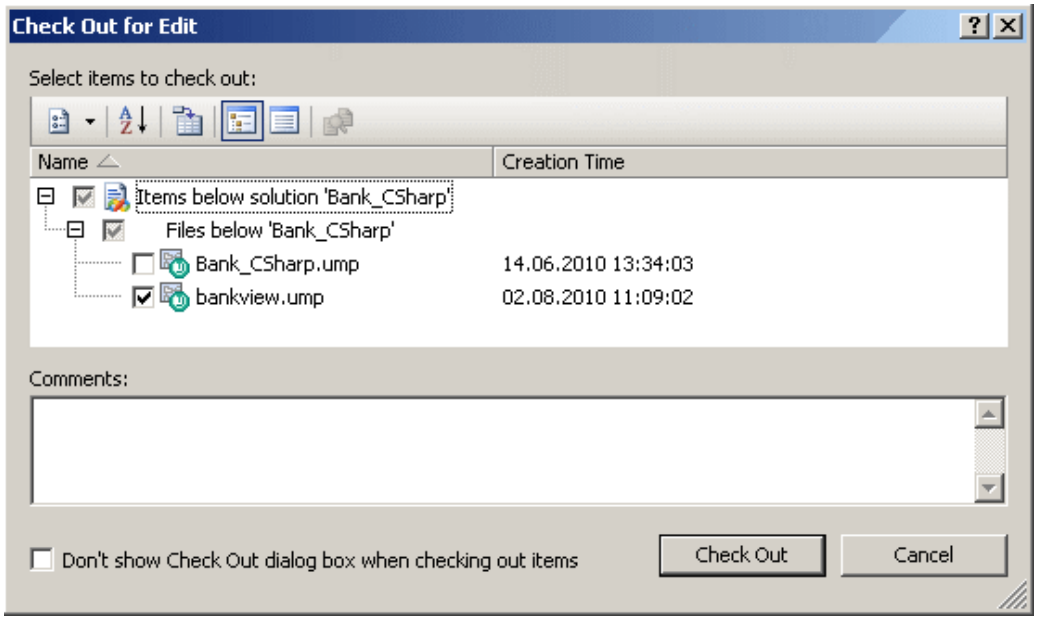
Para más información consulte el apartado Trabajar en equipo con UModel.

Fusión de proyectos a 3 bandas y control de versiones
Cuando se protegen/desprotegen archivos de proyecto, UModel genera archivos antecesores comunes automáticamente, que después se usan para el proceso de fusión a 3 bandas. Esto permite obtener un resultado mucho más preciso que con una fusión a 2 bandas. Recuerde que los archivos de instantánea utilizados para una fusión a 3 bandas en la versión independiente de UModel no se pueden crear automáticamente en Visual Studio.

Para desproteger un subproyecto:

1. Seleccione el comando **Archivo | Control de código fuente | Desproteger para editar**.

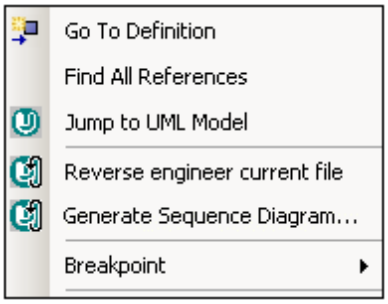
 © 2014 Altova GmbH



2. Seleccione el archivo que desea desproteger y haga clic en el botón **Desproteger**.

Opciones de menú nuevas (UModel / menú contextual)

- Ir al modelo UML / Ir al código (Visual Basic y C#).
- Aplicar ingeniería inversa al archivo actual (Visual Basic y C#).
- Generar diagrama de secuencia...



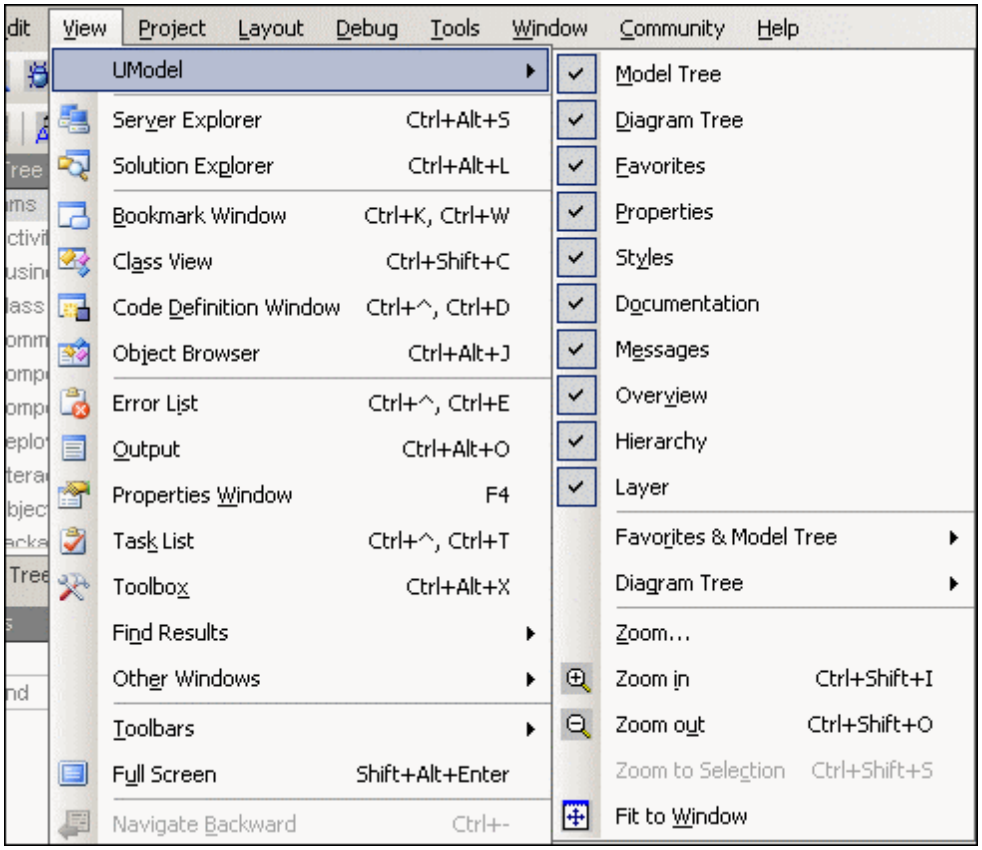
Estos comandos están disponibles en el menú contextual de la ventana correspondiente del código o del modelo.

Funciones de UModel no disponibles en el complemento para Visual Studio

- El entorno de scripting (**Herramientas | Editor de scripting...**) no está disponible en el complemento para Visual Studio.
- La opción **Herramientas | Restaurar barras de herramientas y ventanas...** tampoco está disponible en el complemento para Visual Studio.

Habilitar / deshabilitar los ayudantes de entrada de UModel:

- Seleccione la opción de menú **Ver | UModel** y después marque la casilla correspondiente.



Ingeniería de código
En el panel Estructura del modelo de Visual Studio no está disponible la opción para combinar/ actualizar una sola clase. La sincronización automática y manual actualiza todos los cambios inmediatamente.

Si el código no se puede analizar, en la barra de estado aparece un icono de estado rojo (*imagen siguiente*), que indica que el código y el modelo no están sincronizados. Este icono también aparece si se produjo un error en el último proceso de ingeniería inversa / directa. Y también cuando la revisión sintáctica encuentra errores en UModel. Al hacer clic en un error de análisis en la ventana Mensajes, se abre el archivo de código fuente correspondiente con el cursor en la línea que contiene el error.

En la barra de estado aparece una barra de progreso que indica que el modelo/código se está sincronizando.

Cuando se realicen cambios en el código en Visual Studio, en el visor de detalles de clase, etc., el código se vuelve a analizar y el modelo se actualiza.

Notas sobre la función de sincronización en el complemento de UModel para Visual Studio:

Debido a algunas limitaciones de Visual Studio, algunas modificaciones de código C# y Visual Basic en Visual Studio no desencadenan eventos internos de Visual Studio y, por tanto, no se actualizan automáticamente en UModel.

Esto significa que estos cambios realizados en el código en Visual Studio no actualizan el modelo UML inmediatamente, sino que es necesario forzar una sincronización manual o realizar cambios que sí desencadenen la actualización del archivo de código fuente.

Es necesario forzar una sincronización manual cuando se añadan/modifiquen:

- los valores predeterminados de los atributos
- los valores predeterminados de los parámetros de las operaciones
- ParámetrosDePlantilla
- EnlacesDePlantilla
- el resumen de los elementos
- la sección de comentarios de los elementos
- los cambios en el cuerpo de los métodos

Para forzar una sincronización manual:

Una vez realizados los cambios en el **código**:

1. Haga clic con el botón secundario **en el archivo de código** fuente y seleccione el comando **Aplicar ingeniería inversa al archivo actual** para forzar su actualización.

Nota: si cambia cualquiera de los elementos de modelado indicados en UModel, el código fuente se actualiza automáticamente y no es necesario forzar la sincronización.

Limitaciones en Java:

Si su proyecto de UModel contiene el perfil de Java, la sincronización automática se deshabilita automáticamente en ese proyecto. La aplicación muestra un aviso al abrir el proyecto. En estos proyectos la sincronización debe iniciarse de forma manual.

13.3 Crear un proyecto de UModel en Visual Studio

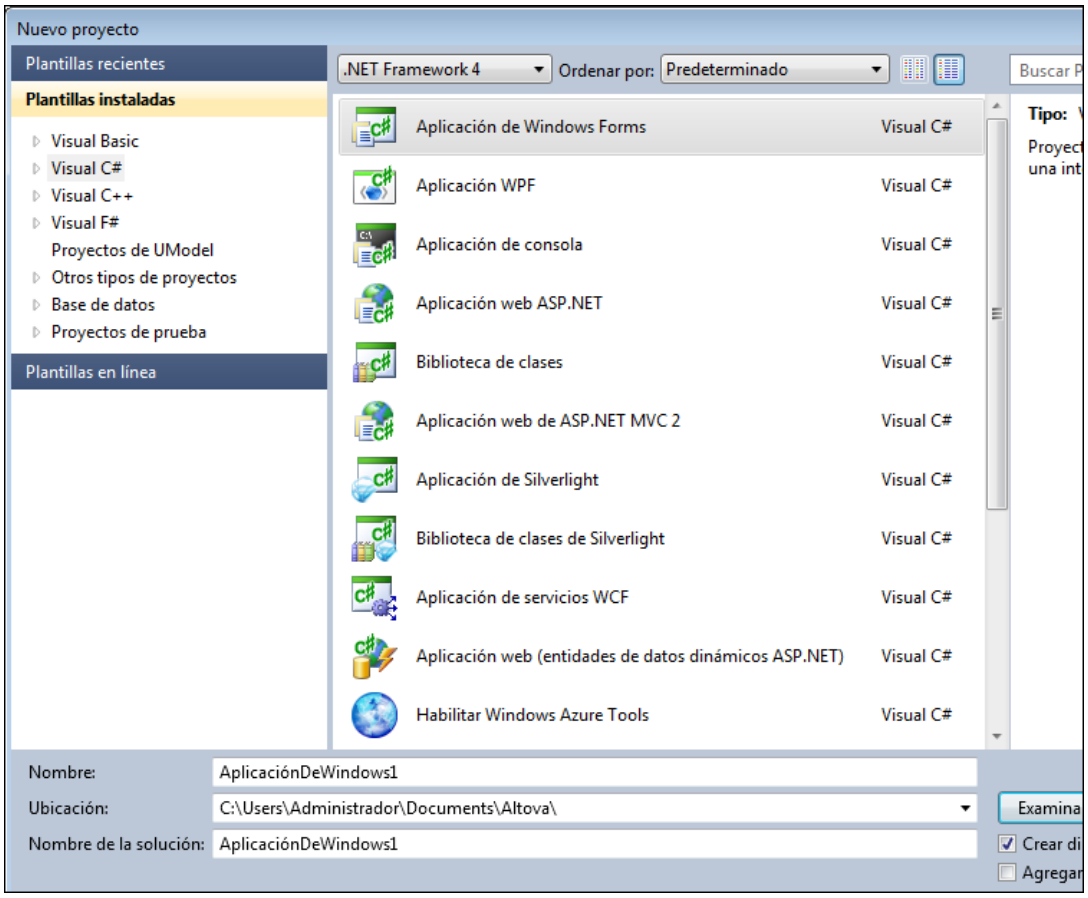
Este apartado describe la manera más rápida de crear un proyecto de UModel en Visual Studio .NET y generar código.

El proceso incluye tres fases:

1. Crear un proyecto de Visual Studio nuevo.
2. Añadir un proyecto de UModel al proyecto de Visual Studio.
3. Iniciar el proceso de ingeniería de código y generar código.

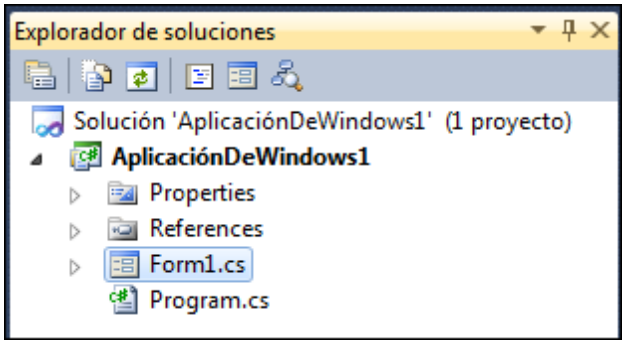
Para crear un proyecto de Visual Studio nuevo:

1. Seleccione la opción de menú **Archivo | Nuevo | Proyecto**.
2. Seleccione `Visual C#` en el panel izquierdo y después seleccione **Aplicación de Windows Forms**. Cuando termine de rellenar los cuadros de texto haga clic en **Aceptar**.



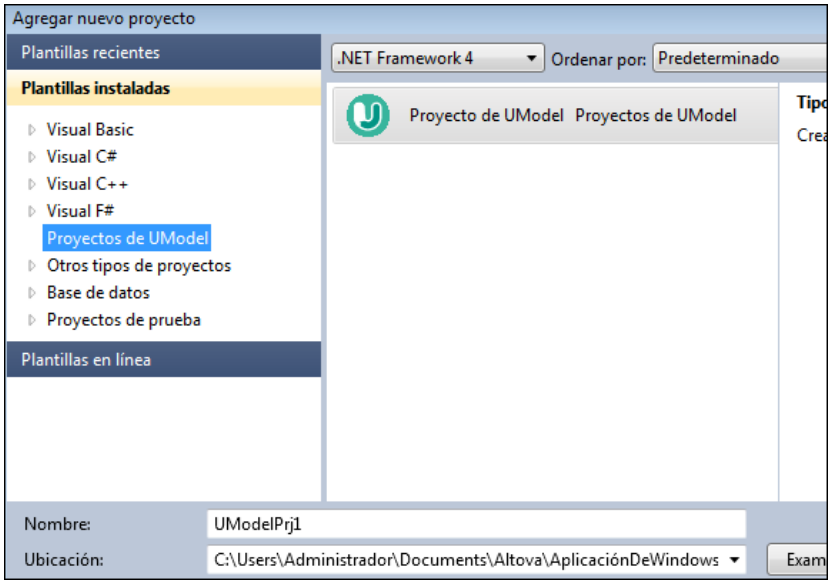
En la ventana Explorador de soluciones se crea una solución nueva: `AplicaciónDeWindows1`.

 © 2014 Altova GmbH

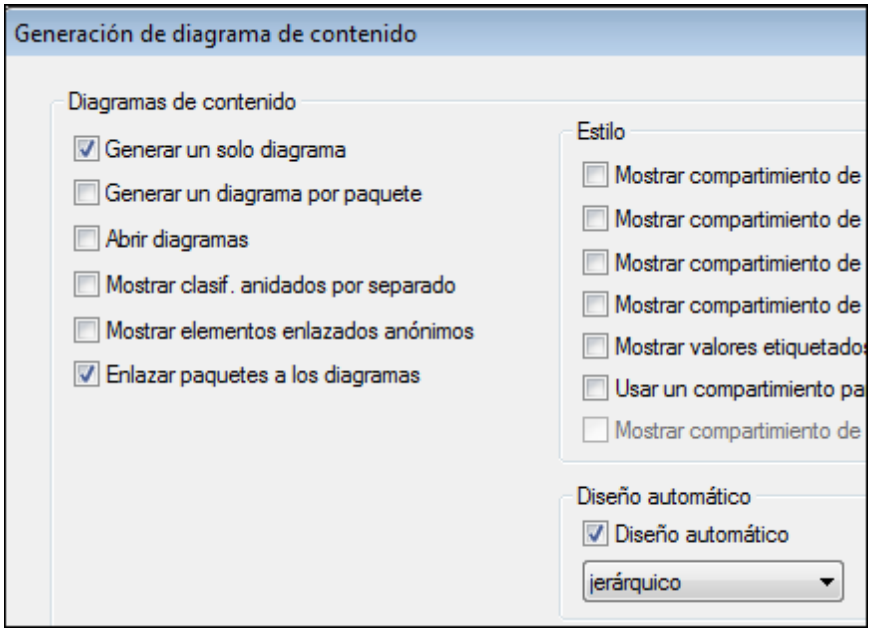


Para agregar un proyecto de UModel al proyecto de Visual Studio:

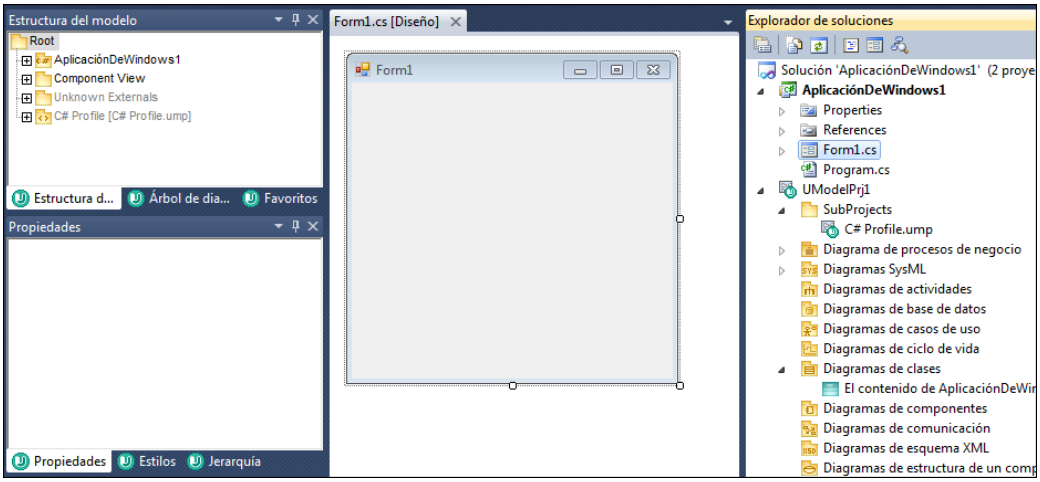
1. En la ventana Explorador de soluciones haga clic con el botón secundario en el nombre de la solución recién creada. En el menú contextual seleccione **Agregar | Nuevo proyecto | Proyectos de UModel | Proyecto de UModel**. Haga clic en **Aceptar** para confirmar.



2. La aplicación pregunta si se deben crear diagramas UML nuevos automáticamente para el proyecto `AplicaciónDeWindows1`. Haga clic en **Sí**.
3. Aparece el cuadro de diálogo "Generación de diagrama de contenido". Seleccione lo que corresponda en el grupo de opciones *Diagramas de contenido*. Si quiere crear diagramas de dependencias entre paquetes, haga clic en **Siguiente**. Si no, haga clic en **Finalizar**.



Aparece un cuadro de mensaje que confirma que se inició la sincronización de código del proyecto. El proyecto de UModel se añadió a la solución y ya puede iniciar el proceso de ingeniería de código.



Observe que el diagrama de clases "El contenido de AplicaciónDeWindows" forma parte del proyecto de UModel.

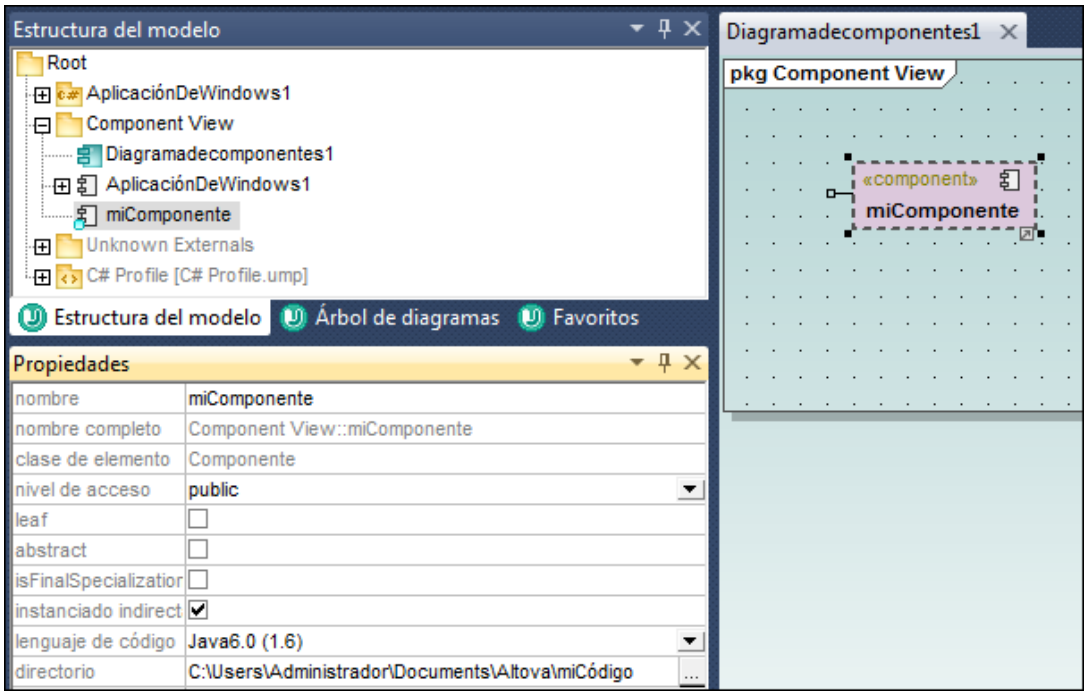
Ingeniería de código y generación de código

Para crear un componente y definir el directorio del código:

1. En el panel Estructura del modelo, haga clic con el botón secundario en el paquete `Component View` y seleccione **Diagrama nuevo | Diagrama de componentes**.
2. Arrastre la clase `MiClase` desde la Estructura del modelo hasta el diagrama de componentes .
3. Haga clic con el botón secundario en el área de trabajo del diagrama y seleccione **Nuevo/a | Componente** (p. ej. `miComponente`).

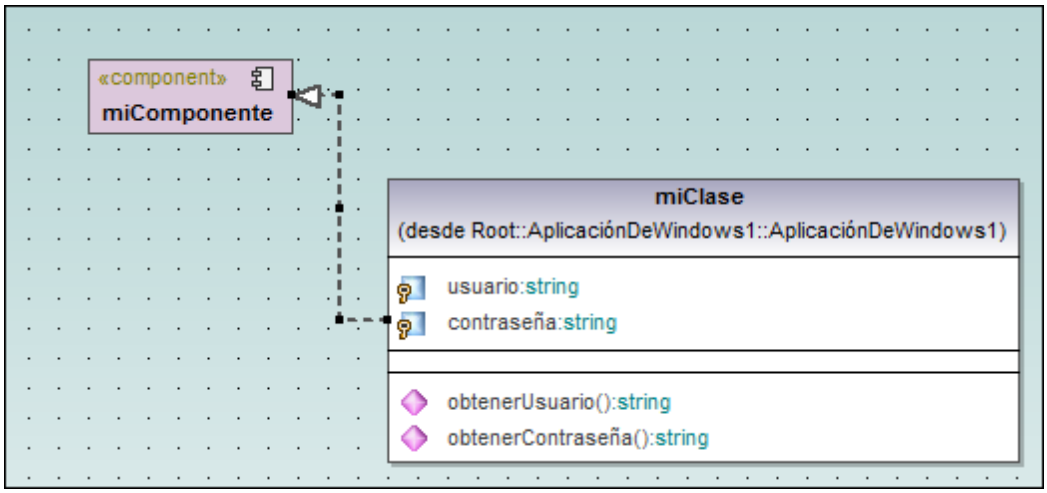
 © 2014 Altova GmbH

4. Haga clic en el componente en el diagrama para seleccionarlo y después haga clic en el campo `directorio` del panel Propiedades. Inserte el directorio donde se debe guardar el código (p. ej. `c:\miCódigo`).



Para realizar una clase:

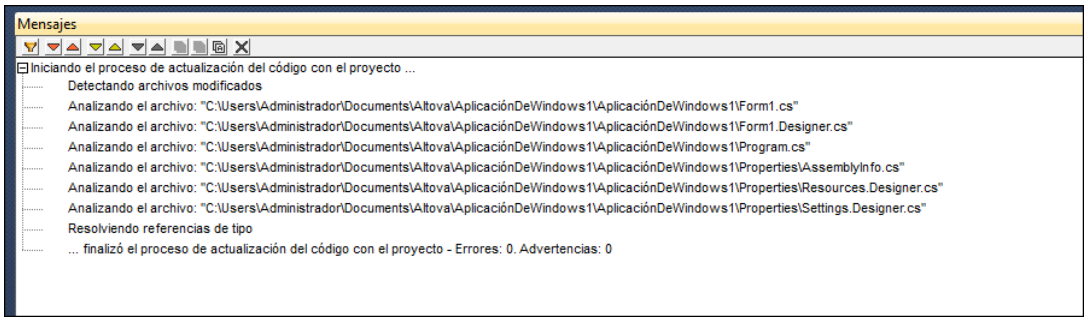
1. Haga clic en **`miClase`** y arrastre su controlador de **`RealizaciónDeComponente`** (situado en la parte inferior) hasta el componente **`miComponente`**.



Para poder generar código es necesario realizar la clase. Otra manera de hacer esto es arrastrar la clase y colocarla directamente en el componente en la Estructura del modelo.

Para revisar la sintaxis y generar código:

1. Seleccione el comando **Proyecto | Proyecto de UModel | Revisar la sintaxis del proyecto** para comprobar que todo es correcto.
2. Seleccione el comando **Proyecto | Proyecto de UModel | Combinar el código de programa con el proyecto de UModel** para generar el código C#.



En la ventana Explorador de soluciones, haga doble clic en el icono del diagrama **`El contenido de AplicaciónDeWindows`** situado debajo de la carpeta **`Diagrama de clases`**.

 © 2014 Altova GmbH

Altova UModel® 2015

Complemento de UModel para Eclipse

14 Complemento de UModel para Eclipse

Eclipse es una plataforma de código abierto que integra diferentes aplicaciones en forma de complementos. El complemento de UModel para Eclipse permite acceder a todas las funciones de UModel desde Eclipse 4.2 / 4.3 / 4.4.

El complemento de UModel para Eclipse ofrece estas funciones y características:

- Herramienta de modelado completamente funcional que permite sincronizar automáticamente el modelo y el código para Java.
- Generación de código Java, C# y Visual Basic.
- Ayuda en pantalla en el comando de menú **Ayuda | UModel | Contenido.**

Sistemas de control de versiones:
El complemento de UModel para Eclipse es compatible con las funciones de control de versiones disponibles en la plataforma Eclipse.

 © 2014 Altova GmbH

14.1 Instalar el complemento de UModel para Eclipse

Antes de instalar el complemento de UModel para Eclipse, asegúrese de que estas aplicaciones están instaladas:

- UModel Enterprise o Professional Edition.
- Java SE Runtime Environment 5.0 (JRE 5.0) o superior, que es necesario para que funcione Eclipse. Consulte el sitio web de Eclipse para obtener más información. Instale la versión de 32 o 64 bits dependiendo de la versión de UModel que tenga instalada.
- Eclipse Platform 4.2 / 4.3 / 4.4. Instale la versión de 32 o 64 bits dependiendo de la versión de UModel que tenga instalada.

Después de instalar todas estas aplicaciones puede instalar el complemento de UModel para Eclipse que incluye el paquete de integración de UModel (*ver más abajo*).

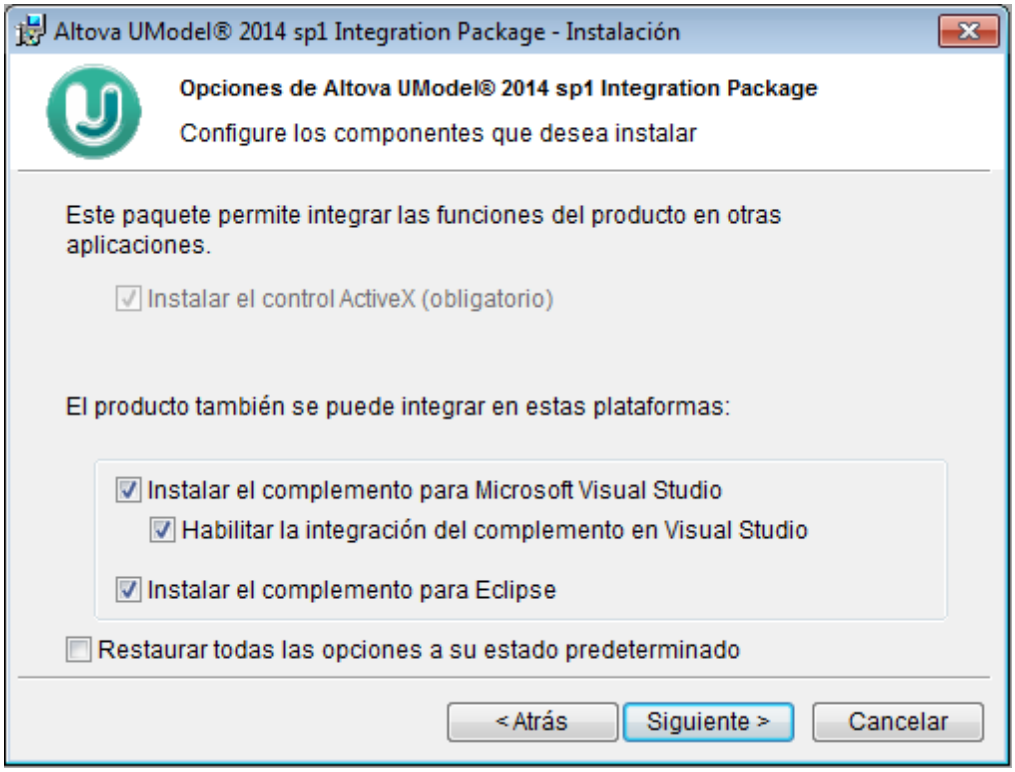
Nota sobre JRE
Si al abrir un documento en Eclipse recibe este mensaje de error:

```
java.lang.UnsupportedClassVersionError: com/altova/....
(Unsupported major.minor version 49.0)
```

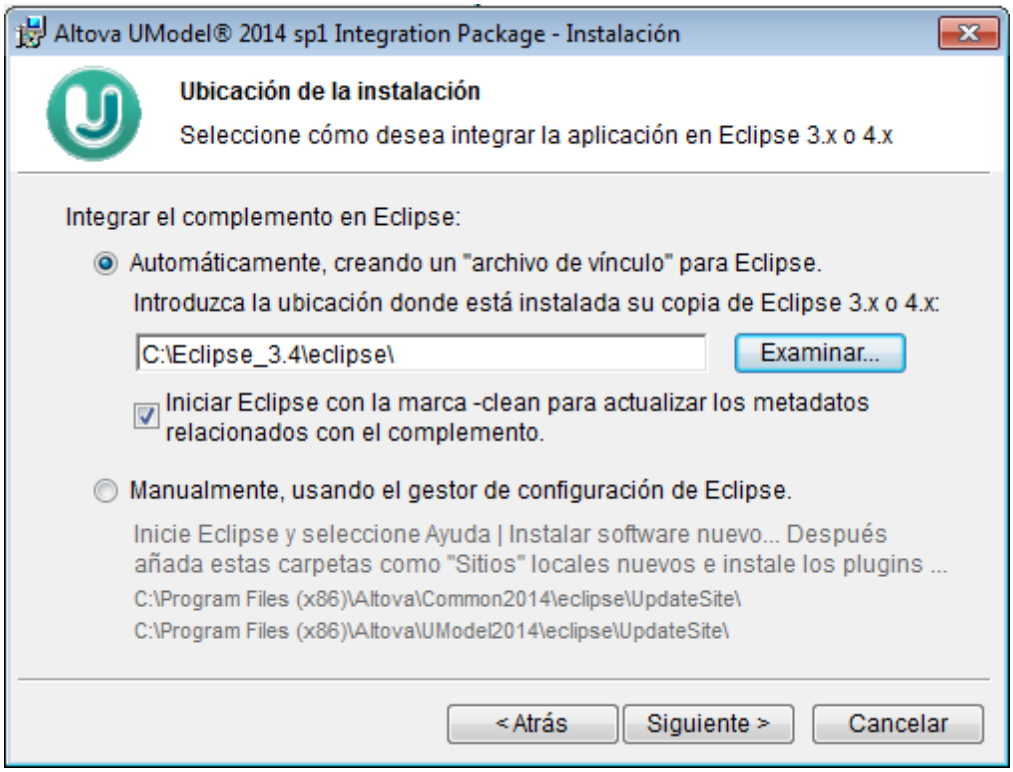
esto significa que Eclipse está usando una versión antigua de JRE presente en su equipo. Puesto que Eclipse usa la variable de entorno `PATH` para encontrar el ejecutable `javaw.exe`, el problema puede resolverse corrigiendo la variable de entorno PATH de modo que encuentre primero la versión más reciente de JRE. Otra opción es iniciar Eclipse con el parámetro de la línea de comandos `-vm`, que aporta la ruta de acceso a un ejecutable `javaw.exe` de la versión 5.0 o superior.

Paquete de integración de UModel
El complemento de UModel para Eclipse está en el paquete de integración de UModel y se instala durante la instalación del paquete de integración de UModel. Estas son las instrucciones de instalación:

1. Compruebe que tiene instalado UModel (Enterprise o Professional Edition), JRE y Eclipse (*ver más arriba*).
2. En la página de descargas de componentes del sitio web de Altova descargue e instale el paquete de integración de UModel. Durante la instalación hay dos pasos muy importantes que se describen en los puntos 3 y 4.
3. Durante la instalación del paquete de integración de UModel aparece una pantalla que le pregunta si desea instalar el complemento de UModel para Eclipse (*imagen siguiente*). Marque la casilla y haga clic en **Siguiente**.



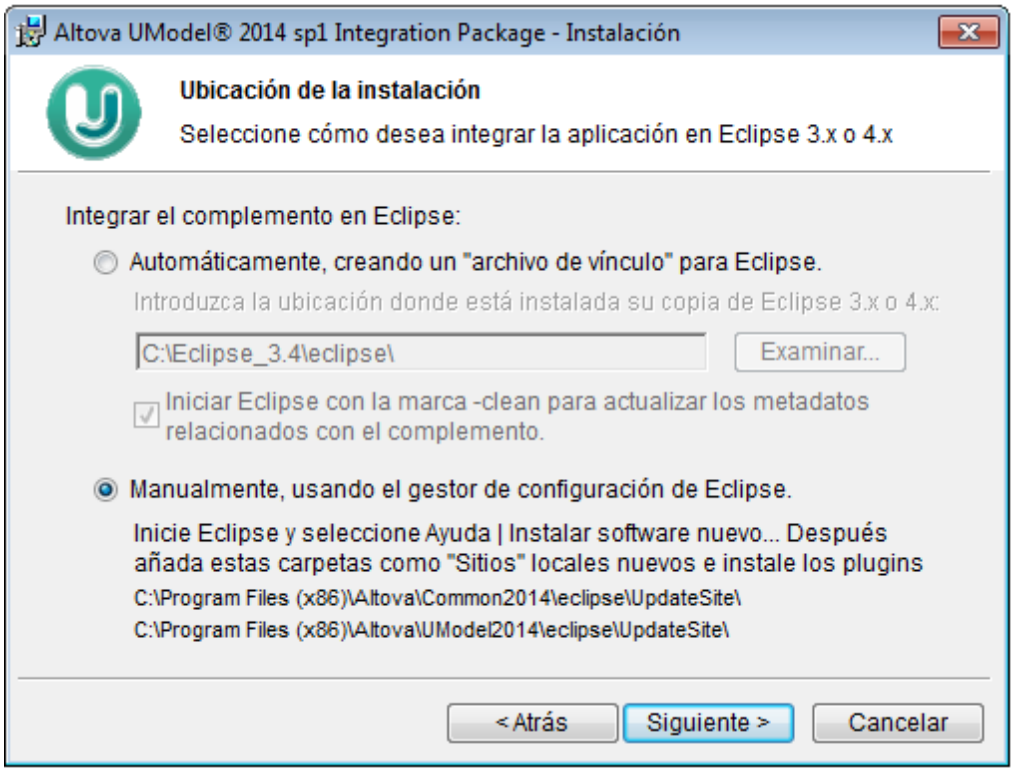
4. En la siguiente pantalla (*imagen siguiente*) puede indicar si el asistente para la instalación debe integrar el complemento de UModel para Eclipse durante la instalación (opción *Automáticamente*) o si prefiere integrarlo de forma manual más adelante.



Recomendamos que elija la opción *Automáticamente* para que el asistente para la

 © 2014 Altova GmbH

instalación se encargue de integrar el complemento. Seleccione la opción *Automáticamente* y después busque la carpeta donde se encuentra el ejecutable de Eclipse (`eclipse.exe`). Después haga clic en **Siguiente**. Si prefiere integrar el complemento de UModel para Eclipse de forma manual, seleccione la opción *De forma manual* (*imagen siguiente*). Más abajo explicamos cómo se integra el complemento de forma manual.

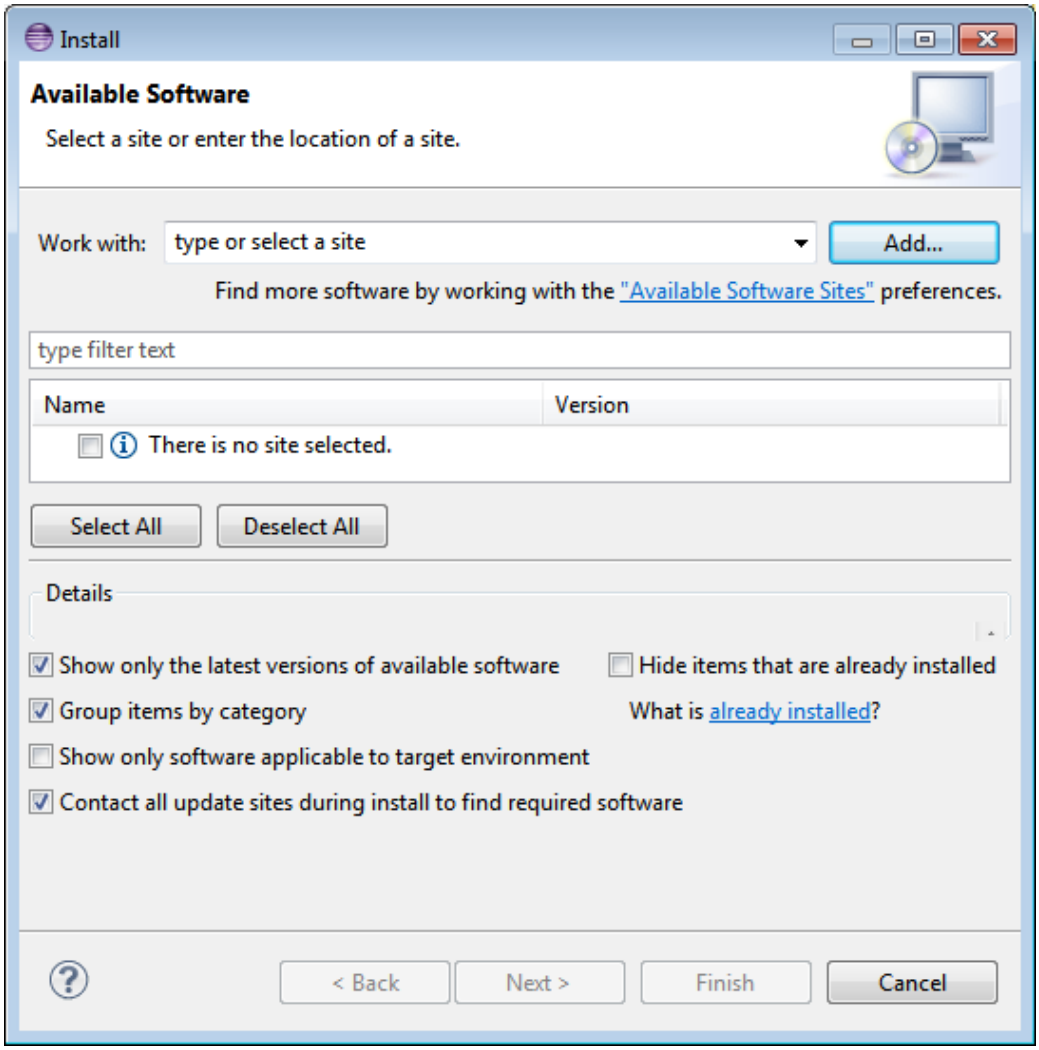


5. Finalice la instalación. Si seleccionó la integración automática, el complemento de UModel para Eclipse se integrará en Eclipse y estará disponible la próxima vez que inicie Eclipse.

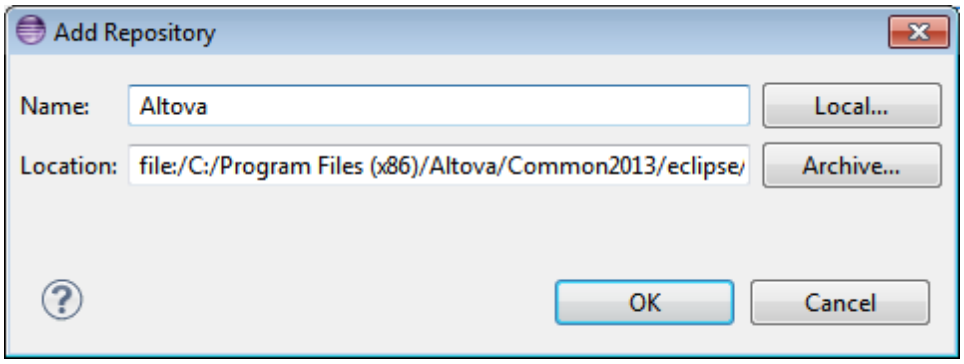
Integrar el complemento de UModel en Eclipse de forma manual

Para integrar el complemento de UModel en Eclipse de forma manual:

1. En Eclipse haga clic en el comando **Help | Install New Software** (Ayuda | Instalar software nuevo).
2. En el cuadro de diálogo que aparece (*imagen siguiente*) haga clic en el botón **Add** (Agregar).



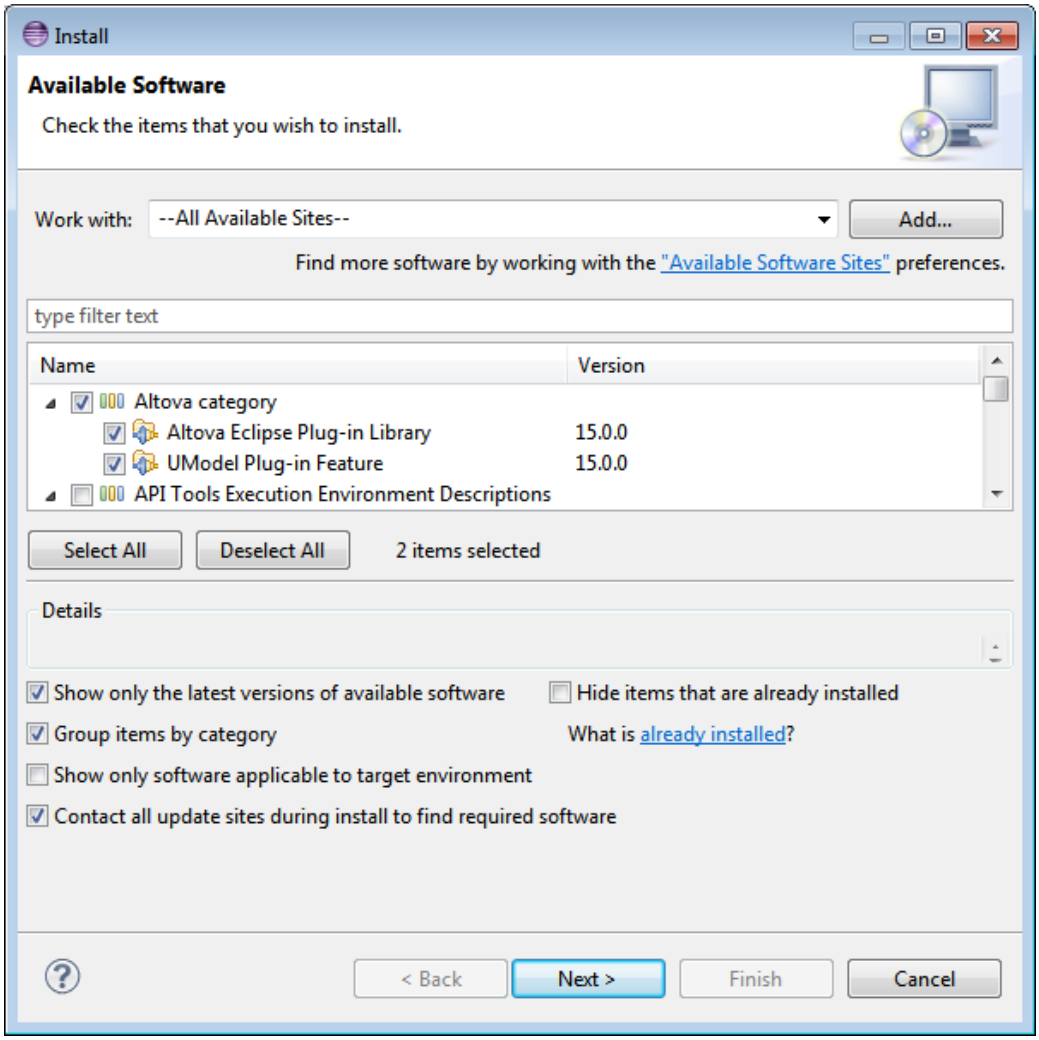
3. En el cuadro de diálogo que aparece, llamado "Add Repository" (Agregar repositorio, *imagen siguiente*) haga clic en el botón **Local**.
4. Busque la carpeta `c:\Archivos de programa\Altova\Common2015\eclipse\UpdateSite` y selecciónela. Escriba un nombre para el sitio (p. ej. Altova) y haga clic en **OK**.



5. Repita los pasos 2, 3 y 4 pero esta vez seleccione la carpeta `c:\Archivos de programa\Altova\XMLSpy2015\eclipse\UpdateSite` y escriba un nombre como "Altova UModel".

 © 2014 Altova GmbH

6. En el cuadro combinado *Work With* (Trabajar con) del cuadro de diálogo "Install" seleccione la opción *All Available Sites* (Todos los sitios disponibles, *imagen siguiente*). Como resultado aparecen todos los complementos disponibles en el panel inferior. Marque la casilla de nivel superior de la carpeta *Altova category* (*imagen siguiente*). Después haga clic en **Next** (Siguiente).



7. Aparece una pantalla llamada "Install Details" (detalles de la instalación) en la que puede repasar los elementos que se van a instalar. Haga clic en **Next** (Siguiente) para continuar.
8. Aparece la pantalla "Review Licenses" (Repasar licencias). Seleccione la opción *I accept the terms of the license agreement* (Acepto el contrato de licencia). Para el complemento de UModel no es necesario otro contrato de licencia (basta con la clave de licencia de UModel Enterprise o Professional Edition). Para terminar haga clic en el botón **Finish** (Finalizar).

Si tiene problemas con el complemento (por ejemplo, si faltan iconos), inicie Eclipse desde la línea de comandos con la marca `-clean`.

Versión instalada actualmente

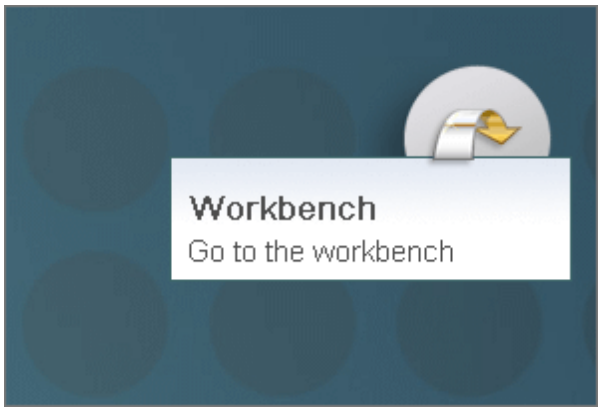
Para comprobar qué versión tiene instalada del complemento de UModel para Eclipse, en Eclipse seleccione la opción de menú **Help | About Eclipse**. Después seleccione el icono de UModel.

© 2014 Altova GmbH

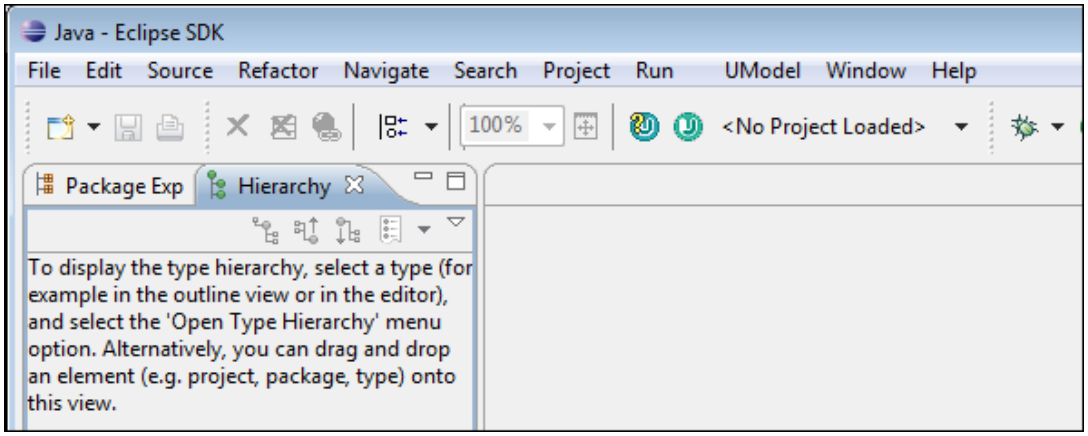
14.2 Iniciar Eclipse y usar el complemento de UModel

Para iniciar Eclipse:

1. Haga doble clic en el icono . Esto abre la pantalla de bienvenida de Eclipse.
2. Haga clic en el icono **Workbench** para cambiar al área de trabajo de Eclipse.

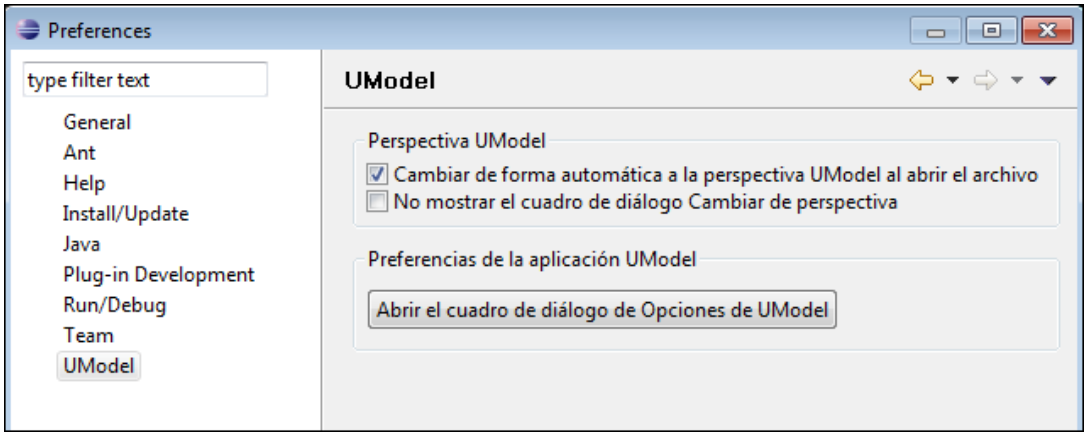


Se abre un área de trabajo vacía y se carga la barra de herramientas de UModel.



Propiedades de UModel:

1. Seleccione la opción de menú **Window | Preferences** y haga clic en **UModel**.
2. Marque la casilla *Cambiar de forma automática a la perspectiva de UModel al abrir el archivo* y haga clic en **Aceptar**.

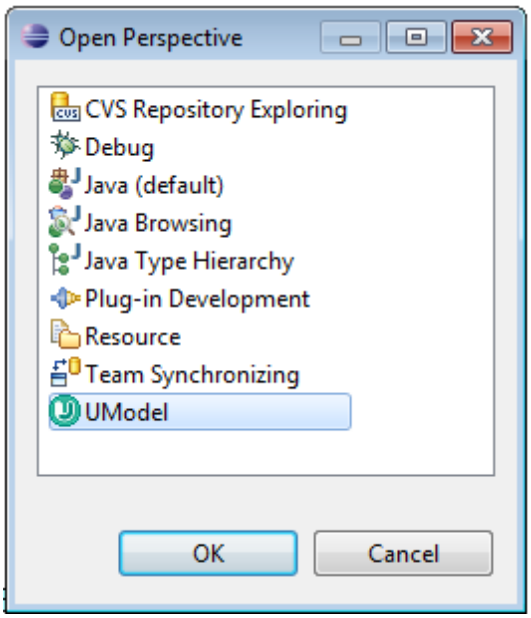


Haga clic en el botón **Abrir el cuadro de diálogo de Opciones de UModel** para definir opciones de configuración de UModel (para bibliotecas, generación de código, etc.).

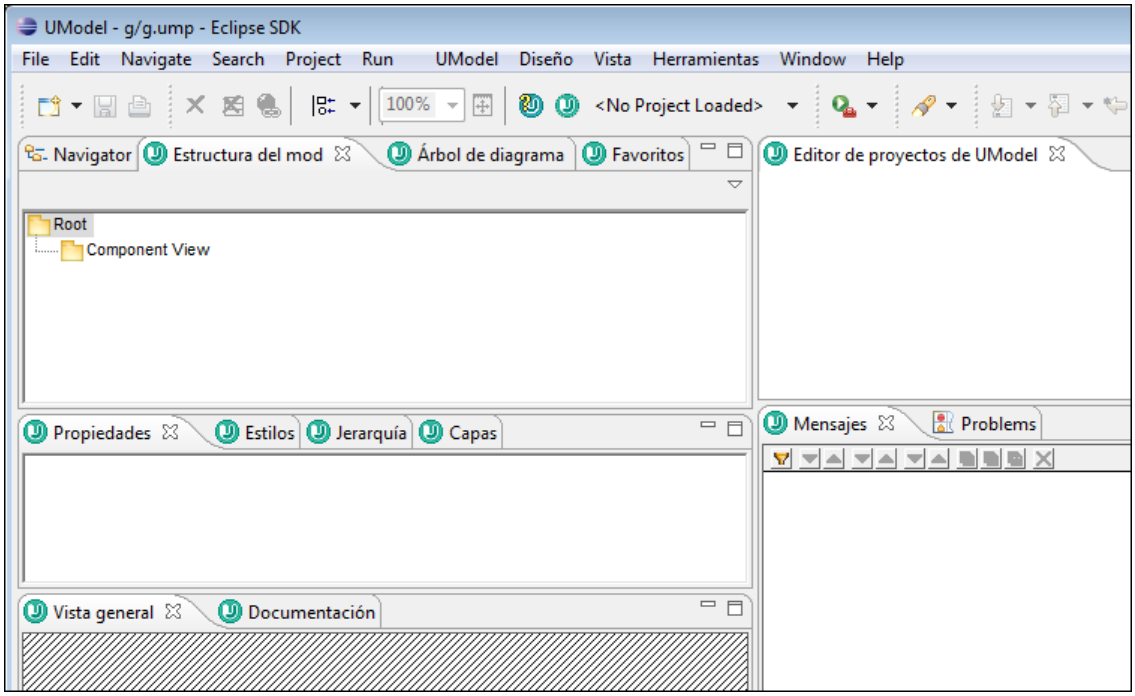
 © 2014 Altova GmbH

14.3 UModel: editor, vista y perspectiva

Para habilitar la perspectiva de UModel en Eclipse, seleccione la opción de menú **Window | Open perspective | Other | UModel**.



Los paneles y ayudantes de entrada de UModel aparecen ahora en el entorno de Eclipse:

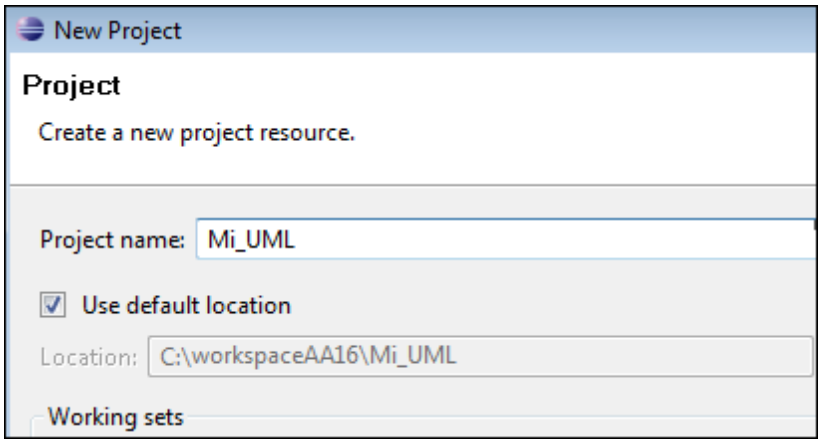


La perspectiva de UModel se puede habilitar automáticamente si marca la casilla *Cambiar de forma automática a la perspectiva de UModel al abrir el archivo* en el cuadro de diálogo Window | Preferences.

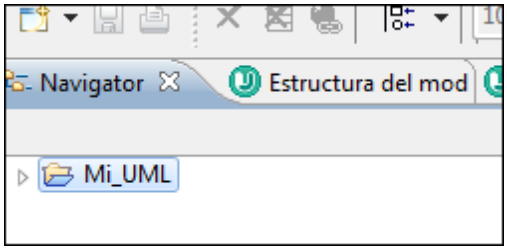
14.4 Crear/abrir un archivo/proyecto de UModel

Para abrir o crear un proyecto de UModel (`*.ump`) antes es necesario crear o abrir un proyecto de Eclipse.

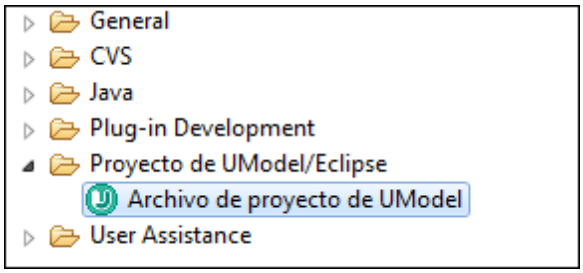
1. Haga clic con el botón secundario en el panel Navigator y seleccione **New | Project**.
2. Seleccione **General | Project** y haga clic en **Next**.
3. Escriba el nombre del proyecto en el campo *Project name* (p. ej. `Mi_UML`) y haga clic en el botón **Finish**.



Esto añade un proyecto nuevo de Eclipse al área de trabajo.

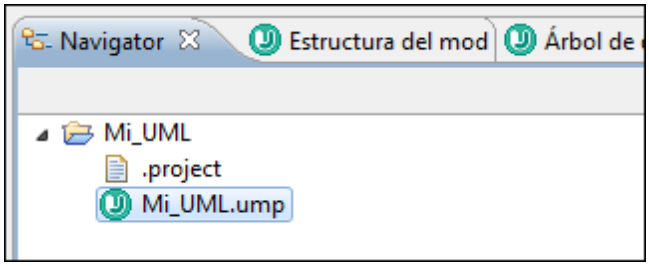


4. Haga clic con el botón secundario en el nuevo proyecto, seleccione **New | Other...** y haga clic en la opción *Archivo de proyecto de UModel*. Después haga clic en **Next**.



5. Escriba el nombre del archivo de proyecto de UModel o utilice el nombre predeterminado (p. ej. `Mi_UML.ump`). Haga clic en **Finish** para finalizar.
   Esto crea un proyecto de UModel nuevo vacío.

 © 2014 Altova GmbH

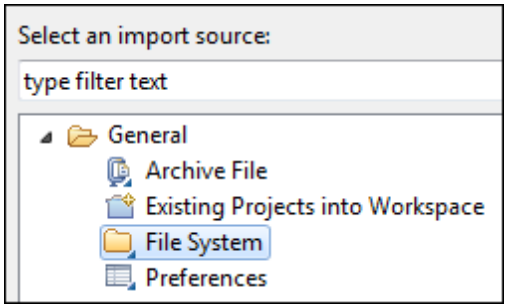


Ahora puede crear archivos de proyecto de UModel dentro de este proyecto de Eclipse o copiar archivos ya existentes. Cuando cambie uno de los modelos, el código correspondiente se genera automáticamente. Los errores y advertencias sobre la generación de código aparecen en la ventana Mensajes de UModel y en la ventana Problems de Eclipse.

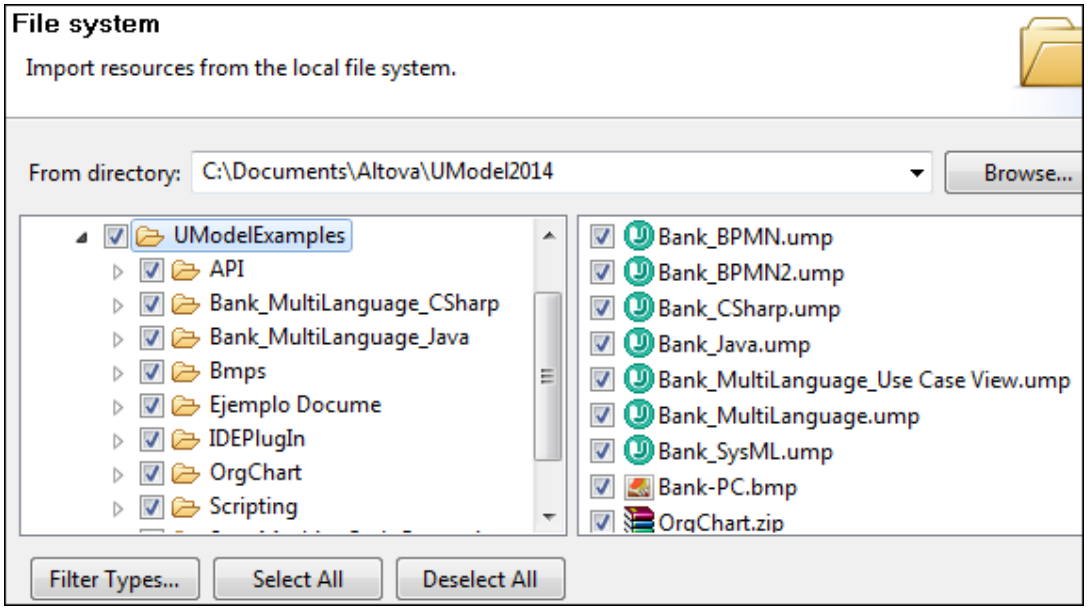
14.5 Importar/abrir ejemplos desde el navegador

Para importar la carpeta de ejemplos de UModel a la ventana Navigator:

1. Haga clic con el botón secundario en la ventana Navigator y haga clic en la opción **Import**.
2. Seleccione **General | File system** y después haga clic en **Next**.

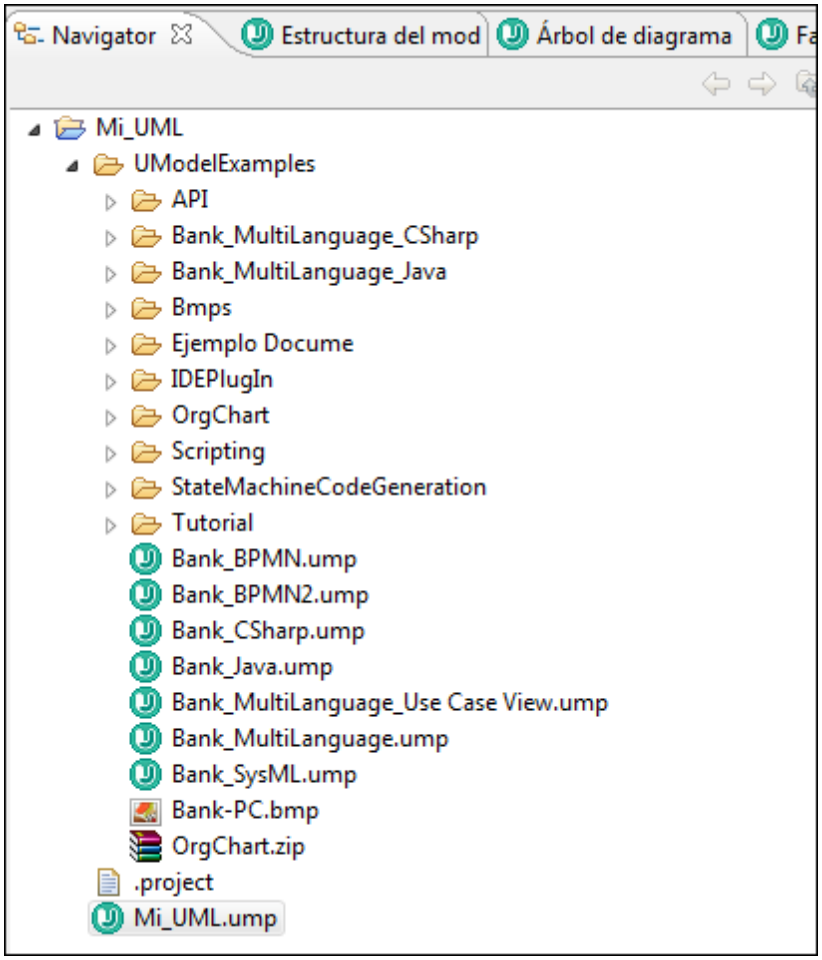


3. Haga clic en el botón **Browse** situado junto al cuadro de texto *From directory*, seleccione el directorio `UModelExamples` de la carpeta `C:\Documents and Settings\Usuario\Mis Documentos\Altova\UModel2015` y haga clic en **OK**.



5. En el campo *Into folder* haga clic en el botón **Browse** para seleccionar la carpeta de destino y después haga clic en **Finish**.

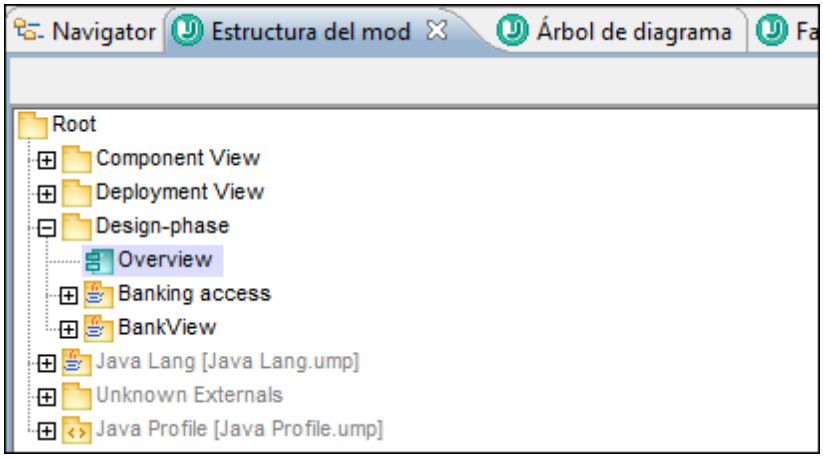
 © 2014 Altova GmbH



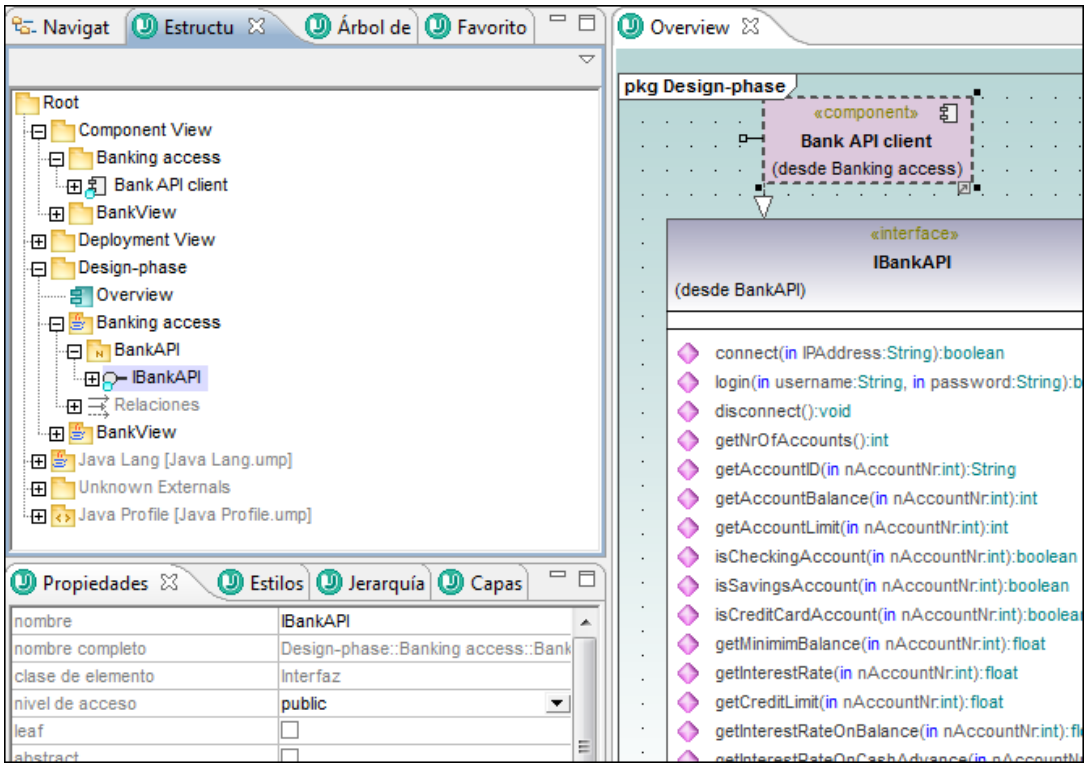
La estructura y los archivos de la carpeta seleccionada se copian al espacio de trabajo de Eclipse.

Para abrir un proyecto de UModel ya existente en el espacio de trabajo:

1. En el panel Navigator haga clic con el botón secundario en el proyecto de UModel y seleccione la opción **UModel | Cargar**.
   El archivo se abre y su estructura se puede ver en la pestaña Estructura del modelo.



2. Expanda la carpeta **`Design-phase`** y haga doble clic en el icono del diagrama **`Overview`**.



Esto abre el diagrama de componentes y muestra los componentes y elementos de interfaz.

Nota: también puede hacer clic en el cuadro combinado de UModel de la barra de herramientas (*imagen siguiente*) y seleccionar el proyecto en la lista desplegable.

14.6 Diferencias con la versión independiente

Todos los proyectos de código Java, C# y VB que forman parte del espacio de trabajo del proyecto se pueden sincronizar sin necesidad de configurarlos previamente. La sincronización automática solamente funciona con código Java.

Los comandos **Proyecto | Importar directorio de código fuente** y **Proyecto | Importar proyecto de código fuente** del complemento de UModel para Eclipse no permiten seleccionar Java en el cuadro combinado *Lenguaje* del cuadro de diálogo "Importar directorio/proyecto de código fuente". El código Java se añade automáticamente en cuanto el proyecto Java se añade al proyecto Eclipse.

Control de versiones
El complemento de UModel para Eclipse es compatible con las funciones nativas de control de versiones de Eclipse. Los comandos de control de versiones para trabajar con archivos de código fuente están disponibles en los menús de Eclipse, no en los menús propios de UModel.

Proyectos y subproyectos
Los proyectos de UModel se pueden dividir en subproyectos para que varios programadores puedan trabajar por separado en partes diferentes del proyecto. Cada subproyecto se puede agregar al sistema de control de código fuente.

Para más información consulte el apartado Trabajo en equipo con UModel.

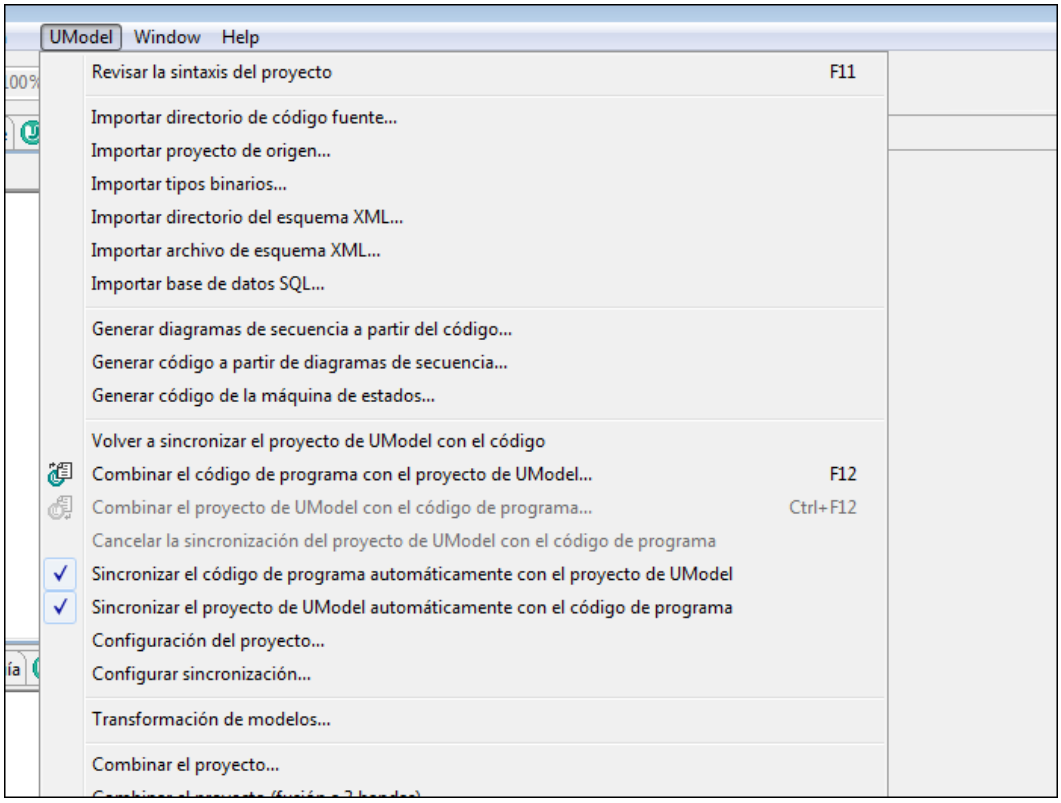
Para desproteger un subproyecto:

1. Utilice los comandos estándar de Eclipse para desproteger el archivo de subproyecto.

Fusión de proyectos a 3 bandas y control de versiones
Cuando se protegen/desprotegen archivos de proyecto, UModel genera archivos antecesores comunes automáticamente, que después se usan para el proceso de fusión a 3 bandas. Esto permite obtener un resultado mucho más preciso que con una fusión a 2 bandas. Observe que los archivos de instantánea utilizados para una fusión a 3 bandas en la versión independiente de UModel no se pueden crear automáticamente en Eclipse.

Nuevo menú:
Por lo general, la mayoría de los comandos de UModel están disponibles en el menú **UModel**.

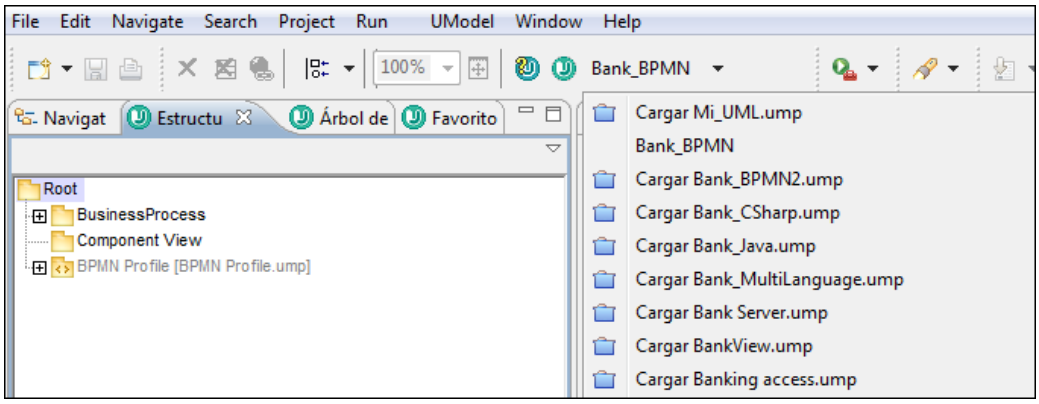
- **Sincronizar el código de programa automáticamente con el proyecto de UModel.**
- **Sincronizar el proyecto de UModel automáticamente con el código de programa.**



Nueva barra de herramientas:

El complemento de UModel para Eclipse incluye una barra de herramientas con estas funciones:

- **Estado de la ingeniería de código:** comprueba el estado actual del proceso de ingeniería de código y muestra información relevante en el panel Mensajes.
- **Ayuda de UModel:** abre el *Manual del usuario y de referencia* de UModel2015.
- **Cargar archivo de UModel:** sirve para seleccionar el proyecto de UModel que se debe abrir. La lista desplegable incluye todos los archivos `*.ump` de UModel disponibles en ese momento en el proyecto de Eclipse que está cargado. El nombre del archivo `*.ump` seleccionado aparece en el cuadro combinado.



Funciones de UModel no disponibles en el complemento para Eclipse

- El entorno de scripting (**Herramientas | Editor de scripting...**) no está disponible en el complemento para Eclipse.
- La opción **Herramientas | Restaurar barras de herramientas y ventanas...** tampoco está disponible en el complemento para Eclipse.

Ingeniería de código
En el panel Estructura del modelo de Eclipse no está disponible la opción para combinar/ actualizar una sola clase. La sincronización automática y manual actualiza todos los cambios inmediatamente.

Si el código no se puede analizar, en la barra de estado aparece un icono de estado rojo, que indica que el código y el modelo no están sincronizados. Este icono también aparece si se produjo un error en el último proceso de ingeniería inversa / directa. Y también cuando la revisión sintáctica encuentra errores en UModel.

Al hacer clic en un error de análisis en la ventana Mensajes, se abre el archivo de código fuente correspondiente con el cursor en la línea que contiene el error.

En la barra de estado aparece una barra de progreso que indica que el modelo/código se está sincronizando.

Cuando se realicen cambios en el código en Eclipse, el código se vuelve a analizar y el modelo se actualiza.

Notas sobre la función de sincronización en el complemento de UModel para Eclipse:
Algunas modificaciones de código C# y Visual Basic en Eclipse no desencadenan eventos internos y, por tanto, no se actualizan automáticamente en UModel.

Esto significa que los cambios realizados en el código en Eclipse no actualizan el modelo UML inmediatamente, sino que es necesario forzar una sincronización manual o realizar cambios que sí desencadenen la actualización del archivo de código fuente.

Limitaciones en C# y Visual Basic:
Si su proyecto de UModel contiene el perfil de C# y Visual Basic, la sincronización automática se deshabilita automáticamente en ese proyecto. La aplicación muestra un aviso al abrir el proyecto. En estos proyectos la sincronización debe iniciarse de forma manual.

14.7 Generación de código

La generación de código se puede forzar a mano seleccionando el comando **Combinar código de programa...** o **Combinar el proyecto de UModel...** del menú **UModel**.

- Puede conseguir una integración total en el proceso de compilación automática de Eclipse asignando el compilador de UModel a un proyecto de Eclipse.
- Para generar código a mano siga las instrucciones que aparecen más abajo.

Para generar código a mano:

1. En la ventana Estructura del modelo haga clic con el botón secundario en la carpeta/paquete pertinente y seleccione el comando **Ingeniería de código | Combinar el código de programa con el proyecto de UModel**. La aplicación solicita una carpeta de destino para el código generado.
2. Seleccione la carpeta y haga clic en **Aceptar** para empezar a generar el código.

 © 2014 Altova GmbH

Altova UModel® 2015

Control de código fuente

15 Control de código fuente

Dado que implementa Microsoft Source Code Control Interface (MSSCCI) v1.1 – v1.3, UModel es compatible con varios sistemas de control de código fuente, como Microsoft SourceSafe y otros repositorios.

Nota: la versión de **64 bits** de UModel es compatible automáticamente con todos los programas de control de código fuente de 32 bits citados en la presente documentación. Si usa la versión de 64 bits con un programa de control de código fuente de 32 bits, la opción *Realizar actualizaciones de estado en segundo plano cada...ms* (**Herramientas | Opciones**) aparece atenuada automáticamente y no se puede seleccionar.

Entrada del registro y complementos
Microsoft ha definido una entrada de registro en la que se pueden registrar todos los programas compatibles con control de código fuente. La entrada para UModel es esta:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders
```

Tenga en cuenta que los complementos de control de código fuente (SC) no se instalan automáticamente con todos los productos de SC. Consulte la documentación de su software de SC para obtener más información sobre los complementos.

Acceder a controles de código fuente en UModel
Para acceder a las funciones de control de código fuente use los comandos del submenú **Proyecto | Control de código fuente**.

Recuerde que un proyecto de control de código fuente no es lo mismo que un proyecto de UModel. Los proyectos de control de código fuente dependen de la estructura del directorio, mientras que los proyectos de UModel son construcciones lógicas que no dependen directamente de la estructura del directorio.

La versión de UModel para Visual Studio y Eclipse también ofrece ya funciones de control de código fuente.

En esta sección...

- Todos los comandos de control de código fuente uno por uno (los comandos del menú **Proyecto | Control de código fuente**.

Problemas relacionados con los recursos y la velocidad
Si trabaja con bases de datos de control de código fuente de gran tamaño, puede que la actualización automática de estado dure más de lo normal.

Si tiene este problema, puede acelerar el sistema desactivando la opción *Realizar actualizaciones de estado en segundo plano cada ... ms* (o aumentando su intervalo) en la pestaña *Control de código fuente* (**Herramientas | Opciones**).

 © 2014 Altova GmbH

15.1 Instalar sistemas de control de código fuente

Para empezar instale su sistema de control de código fuente preferido. Consulte la lista de sistemas de control de código fuente compatibles y las notas sobre la instalación de cada sistema. Si es necesario, configure también la BD del sistema de control para que tenga una ubicación donde pueda guardar sus proyectos.

15.1.1 Sistemas de control de código fuente compatibles

La lista que aparece a continuación enumera los sistemas de control de código fuente compatibles con UModel, junto con sus clientes de control de código fuente. Los sistemas de control de código fuente de la lista aparecen ordenados alfabéticamente. Rogamos lea también las notas sobre niveles de compatibilidad que aparecen después de la lista.

AccuRev

Versión: AccuRev 4.7.0 Windows

Clientes:
- AccuBridge for Microsoft SCC 2008.2

Bazaar

Versión: Bazaar 1.9 Windows

Clientes:
- Aigenta Unified SCC 1.0.6

Borland StarTeam 2008

Versión: StarTeam 2008 Release 2

Clientes:
- Borland StarTeam Cross-Platform Client 2008 R2

Codice Software Plastic SCM

Versión: Codice Software Plastic SCM Professional 2.7.127.10 (Server)

Clientes:
- Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)

Collabnet Subversion 1.5

Versión: 1.5.4

Clientes:
- 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)

Versión: ComponentSoftware CS-RCS (PRO) 5.1

Clientes:
- ComponentSoftware CS-RCS (PRO) 5.1

Dynamsoft SourceAnywhere for VSS

Versión: Dynamsoft SourceAnywhere for VSS 5.3.2 Standard/Professional Server

 © 2014 Altova GmbH

Clientes: • Dynamsoft SourceAnywhere for VSS 5.3.2 Client

Dynamsoft SourceAnywhere Hosted

Versión: Server hosted in a Bell Data Center

Clientes: • Dynamsoft SourceAnywhere Hosted Client (22252)

Dynamsoft SourceAnywhere Standalone

Versión: SourceAnywhere Standalone 2.2 Server

Clientes: • Dynamsoft SourceAnywhere Standalone 2.2 Client

IBM Rational ClearCase 7

Versión: 7.0.1 (LT)

Clientes: • IBM Rational ClearCase 7.0.1 (LT)

March-Hare CVSNT 2.5

Versión: 2.5.03.2382

Clientes: • Aigenta Unified SCC 1.0.6

March-Hare CVS Suite 2008

Versión: Server 2008 [3321]

Clientes:
- 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

Versión: Mercurial 1.0.2 for Windows

Clientes: • Sergey Antonov HgSCC 1.0.1

Microsoft SourceSafe 2005

Versión: 2005 with CTP

Clientes: • Microsoft SourceSafe 2005 with CTP

Microsoft Visual Studio Team System 2008 Team Foundation Server

Versión: 2008

Clientes:
- Microsoft Team Foundation Server 2008/2010 MSSCCI Provider

Perforce 2008

Versión: P4S 2008.1

Clientes:
- Perforce P4V 2008.1

PureCM

Versión: PureCM Server 2008/3a

Clientes:
- PureCM Client 2008/3a

QSC Team Coherence Version Manager

Versión: QSC Team Coherence Server 7.2.1.35

Clientes:
- QSC Team Coherence Client 7.2.1.35

Qumasoft QVCS Enterprise

Versión: QVCS Enterprise 2.1.18

Clientes:
- Qumasoft QVCS Enterprise 2.1.18

Qumasoft QVCS Pro

Versión: 3.10.18

Clientes:
- Qumasoft QVCS Pro 3.10.18

Reliable Software Code Co-Op

Versión: Code Co-Op 5.1a

Clientes:
- Reliable Software Code Co-Op 5.1a

Seapine Surround SCM

Versión: Surround SCM Client/Server for Windows 2009.0.0

Clientes:
- Seapine Surround SCM Client 2009.0.0

Serena Dimensions

Versión: Dimensions Express/CM 10.1.3 for Win32 Server

Clientes:
- Serena Dimensions 10.1.3 for Win32 Client

 © 2014 Altova GmbH

Softimage Alienbrain

Versión: Alienbrain Server 8.1.0.7300

Clientes:
- Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300

SourceGear Fortress

Versión: 1.1.4 Server

Clientes:
- SourceGear Fortress 1.1.4 Client

SourceGear SourceOffsite

Versión: SourceOffsite Server 4.2.0

Clientes:
- SourceGear SourceOffsite Client 4.2.0 (Windows)

SourceGear Vault

Versión: 4.1.4 Server

Clientes:
- SourceGear Vault 4.1.4 Client

VisualSVN Server 1.6

Versión: 1.6.2

Clientes:
- 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

Nota importante:

- Altova ha implementado Microsoft Source Code Control Interface (MSSCCI) v1.1 – v1.3 en UModel y ha probado la compatibilidad de los controladores y sistemas de control de versiones enumerados aquí. La previsión es que UModel siga siendo compatible con estos productos si se actualizan.
- Los complementos de control de código fuente que no aparezcan en la lista anterior pero que se ajusten a la especificación de MSCCI 1.1-1.3 deberían funcionar con UModel.

Para los ejemplos de este apartado usamos el proyecto de UModel **Bank_CSharp.ump** (y sus archivos de código fuente asociados), que están disponibles en la carpeta `C:\Documents and Settings\<usuario>\Mis Documentos\Altova\UModel2015\UModelExamples\`.

Hay varias maneras de usar los comandos de control de código fuente:

- Desde el menú **Proyecto | Control de código fuente**.
- Con el **menú contextual** que se abre al hacer clic con el botón secundario en la Estructura del modelo.

- Con los **iconos** de la barra de herramientas Control de código fuente. Para activar esta barra de herramientas, seleccione **Herramientas | Personalizar | Barras de herramientas**.

Recuerde que un proyecto de control de código fuente no es lo mismo que un proyecto de UModel. Los proyectos de control de código fuente dependen de la estructura del directorio, mientras que los proyectos de UModel son construcciones lógicas que no dependen directamente de la estructura del directorio.

© 2014 Altova GmbH

15.1.2 Notas sobre la instalación

Este apartado explica cómo instalar y configurar los sistemas de control de código fuente compatibles con la aplicación.

AccuBridge for Microsoft SCC 2008.2
http://www.accurev.com/

1. Instale el software cliente AccuRev, ejecute el instalador y especifique el servidor al que desea conectarse (nombre de host y puerto) y después cree una copia de trabajo.
2. Instale el proveedor de SCC AccuBridge. Extraiga el archivo ZIP al directorio <Directorio de instalación de AccuRev>\bin.

   Registre AccuRev.dll y SccAcc.dll de la siguiente manera:
3. Abra una ventana del símbolo del sistema (si trabaja con Vista, inicie el explorador de Windows, vaya a C:\Windows\System32, haga clic con el botón secundario y ejecute `cmd.exe` "como administrador").
4. Vaya al directorio <Directorio de instalación AccuRev>\bin.
5. Escriba este comando en el símbolo del sistema:
   - Regsvr32 AccuRev.dll
   - Regsvr32 SccAcc.dll
6. Ejecute el programa SwitchScc.exe y establezca AccuRev como proveedor.
7. Cierre la sesión de Windows y vuelva a iniciar sesión.

Aigenta Unified SCC 1.0.6
http://aigenta.com/products/UnifiedScc.aspx

Requisitos: cliente de control de código fuente. El cliente SCC Aigenta Unified funciona con:

- el cliente de la línea de comandos Subversion 1.5.4 (http://subversion.tigris.org)
- el cliente CVSNT 2.5 (http://www.cvsnt.org)
- Bazaar 1.9 Windows (http://bazaar-vcs.org/WindowsInstall)

La instalación estándar funciona correctamente con los productos de Altova.

Borland StarTeam Cross-Platform Client 2008 R2
http://www.borland.com/us/products/starteam

Para instalar la integración de Windows de Borland StarTeam ejecute el programa de instalación y seleccione la opción SCC API Integration. Después de la instalación los productos de Altova se pueden conectar al repositorio con solo indicar la dirección del servidor, el extremo, el usuario y la contraseña, su proyecto y la ruta de trabajo.

Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)
http://www.codicesoftware.com/xpproducts.aspx

La instalación estándar funciona correctamente con los productos de Altova. Basta con descargar los componentes "client" y "Visual Studio SCC plug-in".

ComponentSoftware CS-RCS (PRO) 5.1
http://www.componentsoftware.com/Products/RCS

1. Para instalar ComponentSoftware CS-RCS (PRO) ejecute el programa de instalación y seleccione la opción "Workstation Setup".
2. Especifique la raíz del repositorio y reinicie el equipo cuando termine la instalación siguiendo las instrucciones.
3. Use las propiedades "ComponentSoftware RCS Properties" para elegir o crear un proyecto y especificar una carpeta de trabajo.

Dynamsoft SourceAnywhere for VSS 5.3.2 Client
http://www.dynamsoft.com/Products/SAW_Overview.aspx

La instalación estándar funciona correctamente con los productos de Altova. Para integrarlo con los productos de Altova no es necesario instalar el complemento para Adobe DreamWeaver CS3. Una vez finalizada la instalación, establezca una conexión de servidor y una carpeta de trabajo.

Dynamsoft SourceAnywhere Hosted Client (22252)
http://www.dynamsoft.com/Products/SourceAnywhere-Hosting-Version-Control-Source-Control.aspx

La instalación estándar funciona correctamente con los productos de Altova. Para integrarlo con los productos de Altova no es necesario instalar el complemento para Adobe DreamWeaver CS3.

Dynamsoft SourceAnywhere Standalone 2.2 Client
http://www.dynamsoft.com/Products/SourceAnywhere-SourceSafe-VSS.aspx

La instalación estándar funciona correctamente con los productos de Altova. Para integrarlo con los productos de Altova no es necesario instalar el complemento para Adobe DreamWeaver CS3. Una vez finalizada la instalación, establezca una conexión de servidor y una carpeta de trabajo.

IBM Rational ClearCase 7.0.1 (LT)
http://www-01.ibm.com/software/awdtools/clearcase/

Para instalar IBM Rational ClearCase LT ejecute el programa de instalación.

- Si la versión del motor de scripting InstallShield es anterior a la versión 10.5, se le pedirá que la actualice. Elija la opción *Update it if necessary* (actualizar si es necesario). La actualización se ejecuta antes de que se inicie la instalación.
- Elija la opción predeterminada *Enterprise deployment, create a network release area and customize it using Siteprep.*

Para integrarlo con los productos de Altova basta con instalar el cliente. Marque sólo la casilla de cliente.

- Escriba el nombre de servidor y los elementos del servidor de licencias siguiendo los ejemplos que ofrece el programa de instalación (puerto@nombre_servidor).
- Escriba un nombre de descripción de configuración e inserte la ruta de acceso de la zona

 © 2014 Altova GmbH

de release. Esta ruta de acceso debe especificar una carpeta compartida.
- Puede crear una carpeta nueva en su equipo, compartirla y usarla como zona de release. (En Vista debe activar la detección de redes en el Centro de redes y de recursos compartidos para establecer esta ruta de acceso). A continuación se crea la zona de release, en ella se copian algunos archivos y se crea un acceso directo llamado **sitedefs.lnk**.
- Cuando se terminen de copiar todos los archivos, haga doble clic en el acceso directo desde el explorador de Windows. Se inicia otro programa de instalación para instalar el cliente.
- Cuando comience la instalación, elija la opción *Install IBM Rational ClearCase LT*.
- Haga clic en *Siguiente* hasta que aparezca el contrato de licencia de software. Acepte el contrato de licencia de software e instale el cliente.

En **Vista** la segunda instalación podría dar lugar al error interno 2739. Si así fuera, inicie el explorador de Windows y vaya a C:\Windows\System32.
- Haga clic con el botón secundario y ejecute `cmd.exe` como administrador. Aparece una ventana de comandos.
- Escriba `regsvr32 jscript.dll`.
- Ejecute de nuevo el programa de instalación.

Para trabajar con archivos almacenados en ClearCase debe crear una vista que apunte a su proyecto de ClearCase.

Jalindi Igloo 1.0.3
http://www.jalindi.com/igloo/

Para usar Jalindi Igloo con los productos de Altova basta con ejecutar el programa de instalación de Jalindi Igloo. Tenga en cuenta que si desinstala Jalindi Igloo, se eliminan también las claves de registro de los demás proveedores de SCC que estén instalados y estas claves dejan de estar disponibles.

Mientras trabaja con los productos de Altova recomendamos que active el modo *Auto Commit*.
- El modo **Auto Commit** se puede activar en las opciones avanzadas de control de código fuente.
- Después de definir la copia de trabajo ya puede empezar a trabajar.

March-Hare CVS Suite Client 2008 (3321)
http://www.march-hare.com/cvsnt/en.asp

La instalación *típica* funciona correctamente con los productos de Altova.

Mercurial
véase Sergey Antonov HgScc 1.0.1

Microsoft SourceSafe 2005 with CTP
http://msdn.microsoft.com/en-us/vstudio/aa718670.aspx

La instalación estándar de Microsoft SourceSafe 2005 funciona correctamente con los productos de Altova.

Microsoft Team Foundation Server 2008/2010 MSSCCI Provider
http://www.microsoft.com/downloads

Requisitos: Visual Studio 2008 Team Explorer o Visual Studio 2008/2010 con Team Explorer 2008/2010. La instalación estándar funciona correctamente con los productos de Altova.

Perforce P4V 2008.1
http://www.perforce.com/

Con Perforce Visual Client (P4V) puede elegir entre:

- Instalar todas las características de cliente (comportamiento predeterminado)
- O instalar la característica "Plug-in (P4SCC)" solamente.

La instalación predeterminada funciona correctamente con los productos de Altova.

Si decide instalar la característica "SCC Plug-in (P4SCC)" solamente, tenga en cuenta que:

- no funcionan las dos funciones de control de código fuente **Mostrar diferencias** y **Administrador del control de código fuente**.
- la función **Mostrar diferencias** y la opción de iniciar el administrador del control de código fuente no funcionarán porque dependen de las funciones no instaladas "Visual Merge Tool" y "Visual Client (P4V)" respectivamente.
- La función de comparación necesitará otro software, mientras que el administrador del control de código fuente sólo se puede iniciar después de la instalación explícita del “Visual Client (P4V)".
- Después de iniciar la instalación de su Perforce Visual Client, indique las opciones de configuración de su cliente (nombre de servidor, aplicación de edición de texto, nombre de usuario).
- Cuando termine la instalación, no olvide crear una carpeta de trabajo nueva o seleccionar una copia de trabajo ya existente.

PureCM Client 2008/3a
http://www.purecm.com/

La instalación estándar funciona correctamente con los productos de Altova. Cuando termine la instalación, inicie el cliente PureCM para registrar el servidor.

PushOK CVS SCC NT 2.1.2.5
http://www.pushok.com/soft_cvs.php

La instalación estándar es suficiente para usar PushOK CVS SCC NT.

- Cuando termine la instalación, asegúrese de que su copia del complemento de proxy CVS está registrada correctamente.
- Para terminar defina una copia de trabajo y puede empezar a trabajar.

 © 2014 Altova GmbH

PushOK CVS SCC x64 version 2.2.0.4
http://www.pushok.com/soft_cvs.php

La instalación estándar es suficiente para usar PushOK CVS SCC.

- Cuando termine la instalación, asegúrese de que su copia del complemento de proxy CVS está registrada correctamente.
- Para terminar defina una copia de trabajo y puede empezar a trabajar.

PushOK SVN SCC 1.5.1.1
http://www.pushok.com/soft_svn.php

La instalación estándar de PushOK SVN SCC es suficiente para trabajar con los productos de Altova. Es posible que la biblioteca COM **svncom.dll** no pueda registrarse en Vista. Si así fuera, termine la instalación y siga estos pasos para registrar la biblioteca de forma manual:

1. Inicie una ventana de comandos con la opción *Ejecutar como administrador*.
2. Escriba `cd "C:\Program Files\PushOK Software\SVNSCC\svn"`
3. Escriba el comando `> regsvr32 svncom.dll`.

PushOK SVN SCC x64 version 1.6.3.1
http://www.pushok.com/soft_svn.php

La instalación estándar de PushOK SVN SCC es suficiente para trabajar con los productos de Altova. Es posible que la biblioteca COM **svncom.dll** no pueda registrarse en Vista. Si así fuera, termina la instalación y siga estos pasos para registrar la biblioteca de forma manual:

1. Inicie una ventana de comandos con la opción *Ejecutar como administrador*.
2. Escriba `cd "C:\Program Files\PushOK Software\SVNSCC\svn"`
3. Escriba el comando `> regsvr32 svncom.dll`.

QSC Team Coherence Client 7.2.1.35
http://www.teamcoherence.com

La instalación estándar funciona correctamente con los productos de Altova.

- Si el servidor está instalado en el equipo cliente, entonces se crea una conexión predeterminada después de la instalación del cliente.
- Si el servidor reside en otro equipo, entonces necesita cambiar la propiedad HOSTNAME del cuadro de diálogo "Connection Properties" del cliente Team Coherence para que apunte al equipo pertinente.

Qumasoft QVCS Enterprise 2.1.18
http://www.qumasoft.com/

Requisitos: J2SE 1.5 o posterior http://www.oracle.com/technetwork/java/index.html

Ejecute el instalador para instalar el cliente Qumasoft QVCS-Enterprise. Si utiliza **Vista** debe cambiar el directorio de instalación predeterminado “C:\Archivos de programa\QVCS-Enterprise Client” por “C:\QVCS-Enterprise Client”. Esto se debe a que Vista no permite que las aplicaciones escriban en la zona C:\Archivos de programa. Edite el archivo **setEnv.cmd** del directorio de instalación para que la variable de entorno JAVA_HOME apunte a la ubicación de su

JVM.

Si utiliza **Vista**, puede que tenga problemas al guardar el archivo. Si así fuera, siga estas instrucciones:

1. Inicie el explorador de Windows y vaya a C:\Windows\System32.
2. Haga clic con el botón secundario en cmd.exe y ejecútelo como administrador.
3. Aparece una ventana de comandos.
4. Escriba "cd <carpeta de instalación del cliente QVCS –Enterprise>"
5. Escriba "Notepad setEnv.cmd", edite el archivo y guárdelo.
6. Ejecute el archivo por lotes gui.bat desde el directorio de instalación del cliente Qumasoft QVCS-Enterprise.
7. Añada un servidor desde el menú "Server" especificando el nombre de servidor, la dirección IP y los puertos. Inicie sesión y defina una copia de trabajo.

Qumasoft QVCS Pro 3.10.18
http://www.qumasoft.com/

Ejecute el instalador para instalar Qumasoft QVCS-Pro.

- Si utiliza **Vista**, debe modificar el directorio de instalación predeterminado "C:\Program Files\QVCSBin" por "C:\QVCSBin". Esto se debe a que Vista no permite que las aplicaciones escriban en la zona C:\Archivos de programa.
- Cuando termine la instalación, inicie el cliente QVCS 3.10, cree un usuario nuevo y habilite la integración IDE seleccionando el submenú **Ide Integration** del menú Admin y añadiendo QVCS como herramienta de control de versiones.
- Cree un proyecto y establezca una copia de trabajo.

Reliable Software Code Co-Op 5.1a
http://www.relisoft.com/co_op/index.htm

La instalación estándar funciona correctamente con los productos de Altova.

Seapine Surround SCM Client 2009.0.0
http://www.seapine.com/surroundscm.html

La instalación estándar funciona correctamente con los productos de Altova. Cuando termine la instalación es necesario establecer una conexión de servidor.

Serena Dimensions 10.1.3 for Win32 Client
http://www.serena.com/products/dimensions-cm/index.html

Versiones compatibles: Dimensions Express/CM 10.1.3 for Win32 Client

- Realice una instalación "típica" del cliente Serena Dimension.
- Especifique el nombre de host de cliente web y el número de puerto.

Sergey Antonov HgSCC 1.0.1 for Mercurial SCS

 © 2014 Altova GmbH

http://www.newsupaplex.pp.ru/hgscc_news_eng.html

La instalación estándar funciona correctamente con los productos de Altova.

Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300
http://www.alienbrain.com/

- Realice una instalación "típica" de Alienbrain Client Software y después instale Alienbrain Microsoft Visual Studio Integration. Para trabajar con los productos de Altova no hace falta instalar Microsoft Visual Studio.
- Cuando intente abrir un proyecto desde VCS o agregar un proyecto a VCS por primera vez, deberá introducir algunas opciones de configuración de usuario, como su servidor y la base de datos de proyecto a la que desea conectarse.

SourceGear Fortress 1.1.4 Client
http://www.sourcegear.com/fortress

La instalación estándar de SourceGear Fortress funciona correctamente con los productos de Altova.

SourceGear SourceOffsite Client 4.2.0 (Windows)
http://www.sourcegear.com/sos/

La instalación estándar del cliente SourceOffsite funciona correctamente con los productos de Altova.

SourceGear Vault 4.1.4 Client
http://www.sourcegear.com/vault

La instalación estándar del cliente SourceGear Vault funciona correctamente con los productos de Altova.

TamTam CVS SCC 1.2.40
http://www.daveswebsite.com/software/tamtam/

Requisitos: CVSNT 2.5 (cliente) disponible en http://www.cvsnt.org. La instalación estándar funciona correctamente con los productos de Altova.

- Para conectarse a un repositorio de CVS es necesario instalar CVSNT.
- En el producto de Altova, abra las opciones avanzadas de "Control de código fuente" y escriba la ruta de acceso al ejecutable **cvs.exe**.

TamTam SVN SCC 1.2.24
http://www.daveswebsite.com/software/tamtamsvn/

Requisitos: del cliente de línea de comandos Subversion 1.5.4 disponible en http://subversion.tigris.org . La instalación estándar funciona correctamente con los productos de Altova.

- Para conectarse al repositorio de SVN necesita instalar el cliente de línea de comandos Subversion y especificar la ruta de acceso al ejecutable **svn.exe** en las opciones de control de código fuente de la aplicación de Altova.
- Inicie UModel y registre el proveedor de control de código fuente.

Si utiliza **Vista**, puede que se produzca un error al registrar el control de código fuente. Si así fuera, siga estas instrucciones:

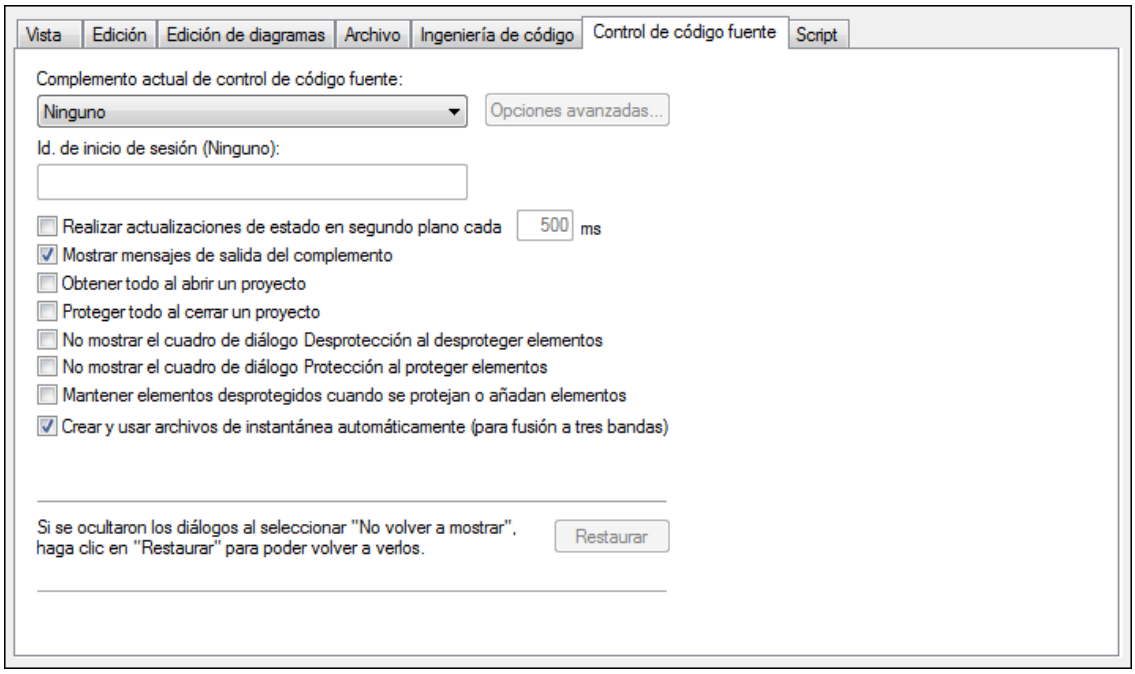
- En el explorador de Windows busque el directorio que contiene el ejecutable de la aplicación de Altova.
- Haga clic con el botón secundario en el ejecutable de Altova y seleccione "Ejecutar como administrador".
  El registro del control de código fuente se realizará correctamente.

 © 2014 Altova GmbH

15.1.3 Comparación con Altova DiffDog

Algunos sistemas SCS se pueden configurar para que usen Altova DiffDog como herramienta de comparación. Los sistemas que son compatibles con esta característica aparecen a continuación, junto con instrucciones para configurarlos. Esto se hace desde la pestaña *Control de código fuente* del cuadro de diálogo "Opciones" (**Herramientas | Opciones**, *ver imagen siguiente*).

Si usa una versión de 64 bits de UModel con un sistema SCS de 32 bits, se deshabilita la opción *Realizar actualizaciones de estado en segundo plano cada XX ms* (**Herramientas | Opciones**), que no se puede seleccionar.



La casilla *Realizar actualizaciones de estado en segundo plano cada X ms* está desactivada por defecto, es decir, no se realizan actualizaciones de estado por defecto. Marque la casilla e introduzca un valor en el campo si desea realizar actualizaciones de estado cada X milisegundos. Esta opción no afecta a las versiones de 64 bits que usan sistemas SCS de 32 bits.

En la pestaña *Control de código fuente* seleccione el sistema SCS que desea usar y después haga clic en el botón **Opciones avanzadas**. Aparece un cuadro de diálogo propio del sistema SCS seleccionado. En cuanto al proceso de instalación, tenga en cuenta que:

- Si eligió la opción estándar de instalación de Altova DiffDog, la ruta de acceso del archivo ejecutable de Altova DiffDog es:
  `C:\Archivos de programa\Altova\DiffDog2015\DiffDog.exe`
  Si Altova DiffDog se instaló en cualquier otra ubicación del sistema, inserte la ruta de acceso cuando corresponda.
- No use DiffDog para **combinar** archivos de proyecto de UModel (`*.ump`).

Configurar Altova DiffDog como herramienta de comparación del control de código fuente

A continuación aparecen instrucciones para configurar Altova DiffDog como herramienta de comparación de los sistemas SCS. En las aplicaciones de Altova que usan sistemas SCS puede configurar Altova DiffDog como herramienta de comparación del sistema SCS, bien desde la aplicación de Altova (**Herramientas | Opciones | Control de código fuente**) o desde el programa de control de código fuente.

Aigenta Unified SCC 1.0.6
http://aigenta.com/products/UnifiedScc.aspx

Siga estos pasos para integrar Altova DiffDog en Aigenta Unified:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Seleccione la pestaña *Comparación y combinación* y escriba la ruta de acceso completa de DiffDog como herramienta de comparación.

Borland StarTeam Cross-Platform Client 2008 R2
http://www.borland.com/us/products/starteam

Siga estos pasos para integrar Altova DiffDog en Borland Star Team:

1. Use las opciones personales del cliente StarTeam (**Tools | Personal options | File | Alternate applications**)
2. Utilidad de comparación: escriba la ruta de acceso completa de DiffDog.
3. Opciones de la utilidad de comparación: `$file1 $file2`.

ComponentSoftware CS-RCS (PRO) 5.1
http://www.componentsoftware.com/Products/RCS

Siga estos pasos para integrar Altova DiffDog en ComponentSoftware CS-RCS (Pro):

1. Abra el cuadro de diálogo de propiedades de ComponentSoftware CS-RCS.
2. En la pestaña Tipos de archivos seleccione una extensión de archivos y edítela.
3. Escriba/seleccione el valor *Custom Tool* para la opción “Difference Analysis Tool“ y examine el sistema para insertar la ruta de acceso completa de DiffDog.

Dynamsoft SourceAnywhere for VSS 5.3.2 Client
http://www.dynamsoft.com/Products/SAW_Overview.aspx

Siga estos pasos para integrar Altova DiffDog en Dynamsoft SourceAnywhere for VSS:

1. Abra el cuadro de diálogo de opciones del cliente Dynamic SourceAnywhere For VSS.
2. Indique la ruta de acceso completa de DiffDog como aplicación externa para comparaciones/combinaciones, con los argumentos: `%FIRST_FILE%” “%SECOND_FILE%`.

Advertencia: no configure estas opciones desde el producto de Altova ya que no se pueden insertar parámetros de aplicaciones externas.

Dynamsoft SourceAnywhere Hosted Client (22252)
http://www.dynamsoft.com/Products/SourceAnywhere-Hosting-Version-Control-Source-

 © 2014 Altova GmbH

Control.aspx

Dynamsoft SourceAnywhere Standalone 2.2 Client
http://www.dynamsoft.com/Products/SourceAnywhere-SourceSafe-VSS.aspx

Siga estos pasos para integrar Altova DiffDog en Dynamsoft SourceAnywhere Hosted y Dynamsoft SourceAnywhere Standalone:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Escriba la ruta de acceso completa de DiffDog como aplicación externa para comparación/combinación con los argumentos `%FIRST_FILE%" "%SECOND_FILE%`.

Advertencia: DiffDog sólo funciona con los archivos `ump` si la opción Tipo es Mergeable en las Propiedades. Para configurar esta propiedad desde UModel, ejecute el comando Mostrar propiedades y configúrela en la sección General. Sin embargo, esto debe hacerse para cada archivo `ump`. Por eso es mejor definir estas opciones en Dynamsoft SourceAnywhere Standalone Server Manager. Seleccione su repositorio, después seleccione "Mergeable File Types" y añada el tipo de archivo `ump`. Esta configuración se aplica a todos los archivos `ump`.

Jalindi Igloo 1.0.3
http://www.jalindi.com/igloo/

Siga estos pasos para integrar Altova DiffDog en Jalindi Igloo:

1. Inicie el comando **Mostrar diferencias** en su aplicación de Altova.
2. Abra el panel **Show Differences or Merge Files**.
3. Defina el comando *External Diff Command* insertando la ruta de acceso completa de DiffDog como ejecutable de aplicación externa de comparación.

Advertencia: si usa el editor predeterminado CvsConflictEditor, puede tener problemas al comparar archivos con líneas excesivamente largas. Recomendamos que aplique la opción *pretty print* a todos los archivos (sobre todo a los archivos `.ump`) antes de almacenarlos en el repositorio. Esta opción limita la longitud de las líneas, lo cual evita problemas con el editor CVSConflictEditor.

March-Hare CVS Suite Client 2008 (3321)
http://www.march-hare.com/cvsnt/en.asp

Siga estos pasos para integrar Altova DiffDog en March-Hare CVS Suite 2008:

1. Vaya a las preferencias de TortoiseCVS y elija la pestaña Tools.
2. Defina la ruta de acceso completa de DiffDog como aplicación de comparación y los parámetros `%1 %2` como parámetros de comparación bidireccional.

Mercurial
véase Sergey Antonov HgScc 1.0.1

Microsoft SourceSafe 2005 with CTP
http://msdn.microsoft.com/en-us/vstudio/aa718670.aspx

Siga estos pasos para integrar Altova DiffDog en Microsoft SourceSafe 2005:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.

2. Haga clic en la pestaña Custom Editors e inserte `C:\Program files\Altova \DiffDog2015\DiffDogexe %1 %2` en el campo Command Line.
3. En el cuadro combinado Operation, seleccione la opción *File Difference*.

Microsoft Team Foundation Server 2008/2010 MSSCCI Provider
http://www.microsoft.com/downloads

Requisitos: Visual Studio 2008 Team Explorer o Visual Studio 2008 con Team Explorer 2008. Siga estos pasos para integrar Altova DiffDog en Microsoft Visual Studio Team System 2008 Team Foundation Server MSSCCI Provider:

1. En las opciones del administrador (Visual Studio 2008 Team Explorer o Visual Studio 2008) configure Altova DiffDog como herramienta de usuario nueva.
2. Elija como complemento Visual Studio Team Foundation Server.
3. Defina una herramienta de usuario nueva indicando: (i) las extensiones de los archivos que desea comparar con DiffDog y (ii) la ruta de acceso completa de DiffDog.

Perforce P4V 2008.1
http://www.perforce.com/

Siga estos pasos para integrar Altova DiffDog en Perforce 2008:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Seleccione la pestaña Diff del panel Preferences.
3. Marque la opción “Other application” y escriba la ruta de acceso completa de DiffDog.

PushOK CVS SCC NT 2.1.2.5,
PushOK SVN SCC 1.5.1.1
PushOK CVS SCC x64 version 2.2.0.4
PushOK SVN SCC x64 version 1.6.3.1
http://www.pushok.com/soft_cvs.php

Siga estos pasos para integrar Altova DiffDog en PushOK CVS NT y PushOK SVN SCC:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Seleccione la pestaña CVS Executables.
3. Seleccione el valor *External merge/compare tool* en el campo Diff/Merge.
4. Inserte la ruta de acceso completa de DiffDog.
5. Escriba `%first %second` en el campo "2 way diff cmd".

Advertencia: si usa el editor predeterminado CvsConflictEditor, puede tener problemas al comparar archivos con líneas excesivamente largas. Recomendamos que aplique la opción *pretty print* a todos los archivos (sobre todo a los archivos `.ump`) antes de almacenarlos en el repositorio. Esta opción limita la longitud de las líneas, lo cual evita problemas con el editor CVSConflictEditor.

QSC Team Coherence Client 7.2.1.35
http://www.teamcoherence.com

 © 2014 Altova GmbH

Siga estos pasos para integrar Altova DiffDog en Team Coherence Version Manager:

1. Vaya a las opciones del cliente Team Coherence.
2. En el grupo de opciones “Difference Viewer”, en el campo Default Difference Viewer, escriba la ruta de acceso completa de DiffDog.
3. Especifique `"$LF $RF"` como parámetros.

Advertencia: puede que sea necesario cerrar sesión y volver a iniciar sesión en Windows para que tengan efecto las nuevas opciones de configuración.

Qumasoft QVCS Enterprise 2.1.18

http://www.qumasoft.com/

Siga estos pasos para integrar Altova DiffDog en Qumasoft QVCS-Enterprise:

1. Añada el directorio de instalación de Qumasoft QVCS-Enterprise a la variable de entorno PATH.
2. Use las preferencias de usuario de QVCS Enterprise.
3. En Utilities marque la casilla Use External Visual Compare Tool.
4. En Visual Compare Command Line escriba
   `<Ruta de acceso completa de DiffDog> “file1Name file2Name”`

Qumasoft QVCS Pro 3.10.18

http://www.qumasoft.com/

Siga estos pasos para integrar Altova DiffDog en Qumasoft QVCS-Pro:

1. Use las preferencias del cliente QVCS 3.10.
2. En Utilities, defina la ruta de acceso completa de DiffDog como utilidad de comparación visual con los parámetros "%s %s".

Seapine Surround SCM Client 2009.0.0

http://www.seapine.com/surroundscm.html

Siga estos pasos para integrar Altova DiffDog en Seapine Surround SCM:

1. Vaya a las opciones de usuario (Diff/Merge) del cliente Surround SCM.
2. En Diff/Merge defina la ruta de acceso completa de DiffDog con los parámetros "%1" "%2".
3. Reinicie el cliente Surround SCM y los productos de Altova.

Sergey Antonov HgSCC 1.0.1

http://www.newsupaplex.pp.ru/hgscc_news_eng.html

Siga estos pasos para integrar Altova DiffDog en Mercurial:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Seleccione “custom” bajo herramientas de comparación e indique la ruta de acceso completa de DiffDog.

SourceGear Fortress 1.1.4 Client
http://www.sourcegear.com/fortress

SourceGear Vault 4.1.4 Client
http://www.sourcegear.com/vault

Siga estos pasos para integrar Altova DiffDog en SourceGear Fortress and SourceGear Vault:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Configure las opciones Diff/Merge Vault definiendo la ruta de acceso completa de DiffDog como programa de comparación y usando los argumentos:
   /ro1 /ro2 /title1:"%LEFT_LABEL%" /title2:"%RIGHT_LABEL%" "%LEFT_PATH%" "%RIGHT_PATH%"

SourceGear SourceOffsite Client 4.2.0 (Windows)
http://www.sourcegear.com/sos/

Siga estos pasos para integrar Altova DiffDog en SourceGear SourceOffsite:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. En “External Programs”, “Application for comparing files”, inserte la ruta de acceso completa de DiffDog.

TamTam CVS SCC 1.2.40,
TamTam SVN SCC 1.2.24
http://www.daveswebsite.com/software/tamtam/

Siga estos pasos para integrar Altova DiffDog en TamTam CVS SCC and TamTam SVN SCC:

1. Haga clic en el botón **Opciones avanzadas** de la pestaña *Control de código fuente*.
2. Defina la ruta de acceso completa de DiffDog como herramienta externa para Diff/Merge and Conflict.

Advertencia: si usa el editor predeterminado CvsConflictEditor, puede tener problemas al comparar archivos con líneas excesivamente largas. Recomendamos que aplique la opción *pretty print* a todos los archivos (sobre todo a los archivos `.ump`) antes de almacenarlos en el repositorio. Esta opción limita la longitud de las líneas, lo cual evita problemas con el editor CVSConflictEditor.

 © 2014 Altova GmbH

15.2 Comandos de control de código fuente

En esta sección utilizamos Visual SourceSafe para explicar las funciones de control de código fuente de UModel.

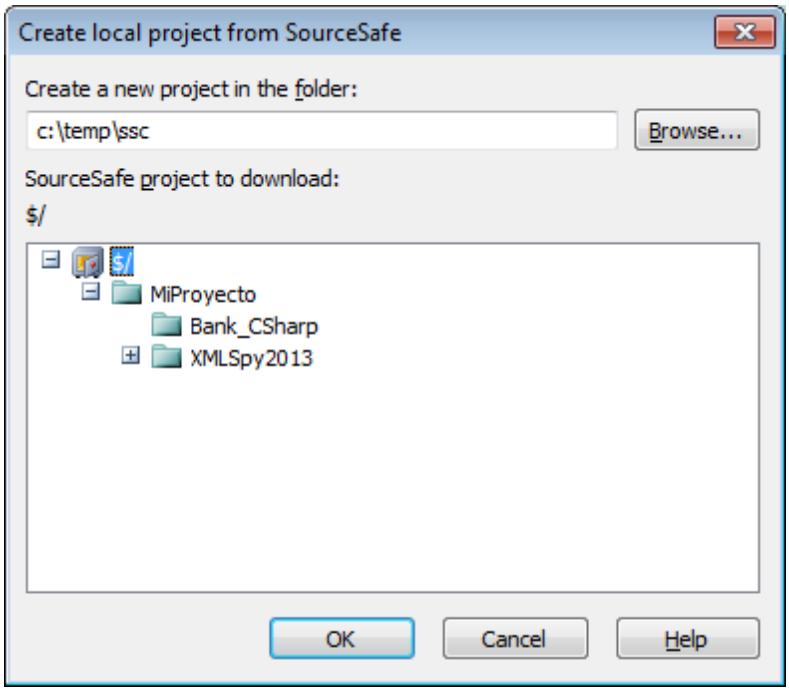
Tenga en cuenta que los comandos de control de código fuente que se describen en esta sección son los de la versión independiente de UModel. El complemento de UModel para Visual Studio y Eclipse utiliza las funciones y comandos de control de código fuente de los propios entornos IDE.

Abrir desde el control de código fuente
Habilitar control de código fuente
Obtener la versión más reciente
Obtener
Obtener carpetas
Desproteger
Proteger
Anular desprotección
Agregar al control de código fuente
Quitar del control de código fuente
Compartir desde el control de código fuente
Mostrar historial
Mostrar diferencias
Mostrar propiedades
Actualizar estado
Administrador del control de código fuente
Cambiar control de código fuente

15.2.1 Abrir desde el control de código fuente

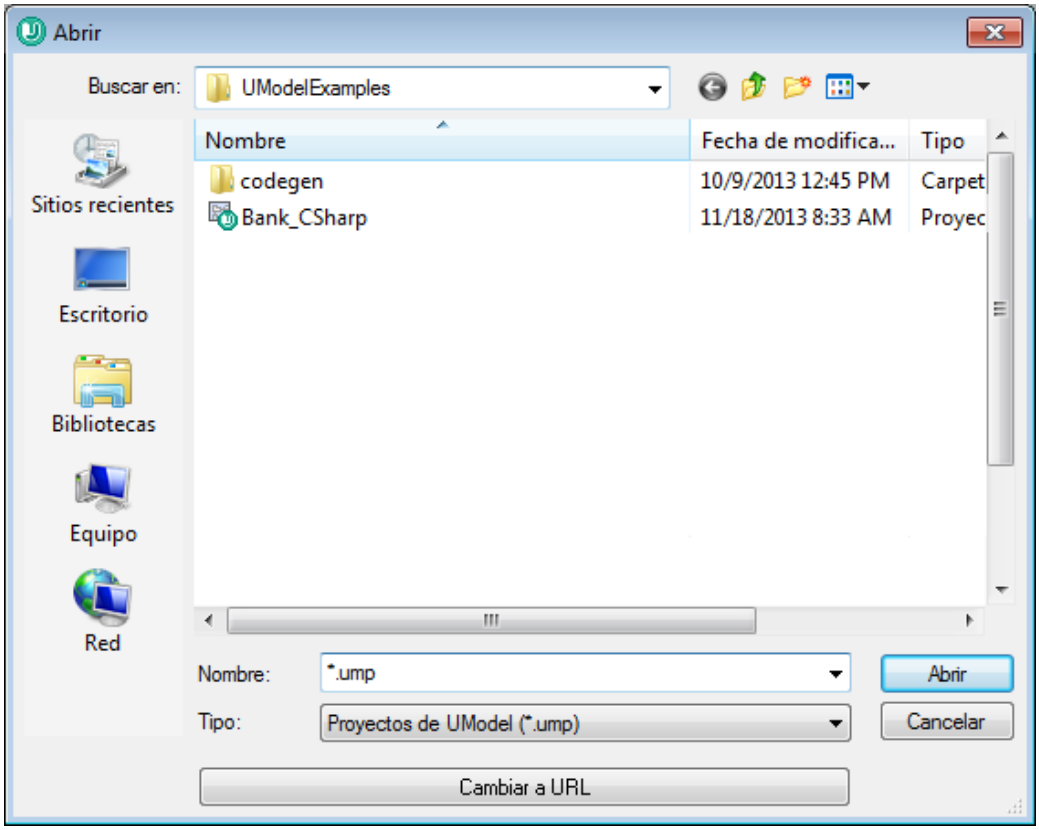
Este comando crea un proyecto local a partir de una BD de control de código fuente ya disponible y lo pone bajo control de código fuente. Para los ejemplos siguientes usamos SourceSafe.

1. Haga clic en **Proyecto | Control de código fuente | Abrir desde el control de código fuente**.
   Se abre el cuadro de diálogo de inicio de sesión. Inserte sus datos para continuar.
   Aparece el cuadro de diálogo "Create local project from SourceSafe".
2. Defina el directorio que debe contener el proyecto local nuevo (p. ej. `c:\temp\ssc`), que en adelante será el directorio de trabajo (o carpeta de desprotección).



3. Seleccione el proyecto de SourceSafe que desea descargar (p. ej. `Bank_CSharp`).
   Si la carpeta que define aquí no existe en la ubicación, deberá crearla.
4. Haga clic en **Sí** para crear el directorio nuevo. Ahora aparece el cuadro de diálogo "Abrir".

 © 2014 Altova GmbH



5. Seleccione el archivo de proyecto de UModel `Bank_CSharp.ump` y haga clic en **Abrir**.

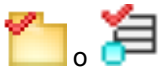
   `Bank_CSharp.ump` se abre en UModel y el archivo se pone bajo control de código fuente. Observe que junto a la carpeta **`Root`** (en la Estructura del modelo) aparece un icono en forma de candado. La carpeta **`Root`** representa tanto el archivo de proyecto como el directorio de trabajo para las operaciones de control de código fuente.

   El directorio `BankCSharp` se creó localmente y ahora puede trabajar con estos archivos de forma totalmente normal.

   Nota: para poner bajo control de código fuente los archivos de código generados cuando se sincronizó el código, consulte el apartado Agregar al control de código fuente.

Iconos del control de código fuente:

El icono en forma de candado indica que el archivo / la carpeta está bajo control de código fuente pero no está desprotegido.



La marca de verificación roja indica que el archivo / la carpeta se desprotegió para poder editarlo. Si además en la barra de título de la aplicación aparece un asterisco, significa que se realizaron cambios en el archivo y al cerrar la aplicación aparece un aviso para que guarde los cambios.

El icono en forma de flecha indica que otro usuario de la red desprotegió el archivo o que se desprotegió en otro directorio de trabajo distinto.

© 2014 Altova GmbH

15.2.2 Habilitar control de código fuente

Este comando sirve para habilitar/deshabilitar el control de código fuente para proyectos de UModel y está disponible en el menú **Proyecto | Control de código fuente**. Si se ejecuta desde un archivo/una carpeta, la acción se lleva a cabo en todo el proyecto de UModel.

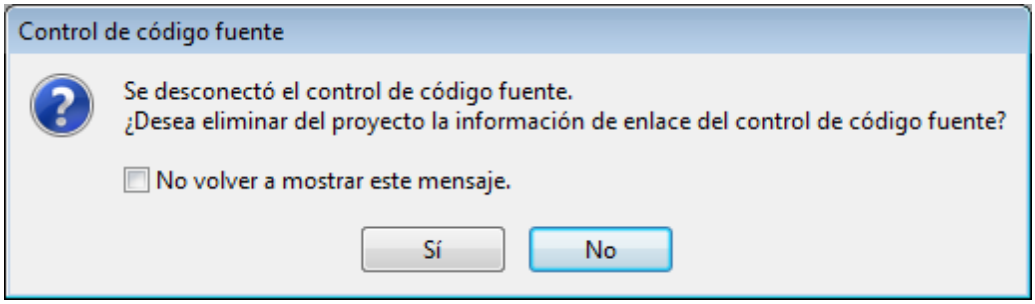
Para habilitar el control de código fuente para un proyecto:

1. Haga clic en **Proyecto | Control de código fuente** y active el comando **Habilitar control de código fuente**.

   La aplicación recupera el estado previo de protección/desprotección de los archivos, lo cual se refleja en la Estructura del modelo.

Para deshabilitar el control de código fuente para un proyecto:

1. Haga clic en el comando **Proyecto | Control de código fuente** y desactive el comando **Habilitar control de código fuente**.



   La aplicación pregunta si quiere quitar la información de enlace del proyecto.

   Haga clic en **No** para deshabilitar el control de código fuente en el proyecto de forma provisional y en **Sí** para deshabilitarlo permanentemente.

15.2.3 Obtener la versión más reciente

Este comando **recupera** la versión más reciente del archivo seleccionado del control de código fuente y la **coloca** en el directorio de trabajo. Los archivos se recuperan para solo lectura y no se desprotegen.

Si al ejecutar el comando los archivos están desprotegidos, pueden pasar tres cosas dependiendo del proveedor de control de código fuente: (i) no ocurre nada, (ii) los datos nuevos se combinan con el archivo local (iii) o los cambios se sobrescriben.

Este comando funciona igual que el comando **Obtener**, con la diferencia de que no abre el cuadro de diálogo "Control de código fuente - Obtener". Esto significa, por tanto, que con este comando no se pueden definir opciones avanzadas.

Si se ejecuta en una carpeta, este comando obtiene la versión más reciente recursivamente, es decir, abarca todos los archivos situados bajo el actual.

Para obtener la versión más reciente de un archivo:

1. En la Estructura del modelo seleccione los archivos cuya versión más reciente desea obtener.
2. Haga clic **Proyecto | Control de código fuente | Obtener la versión más reciente**.

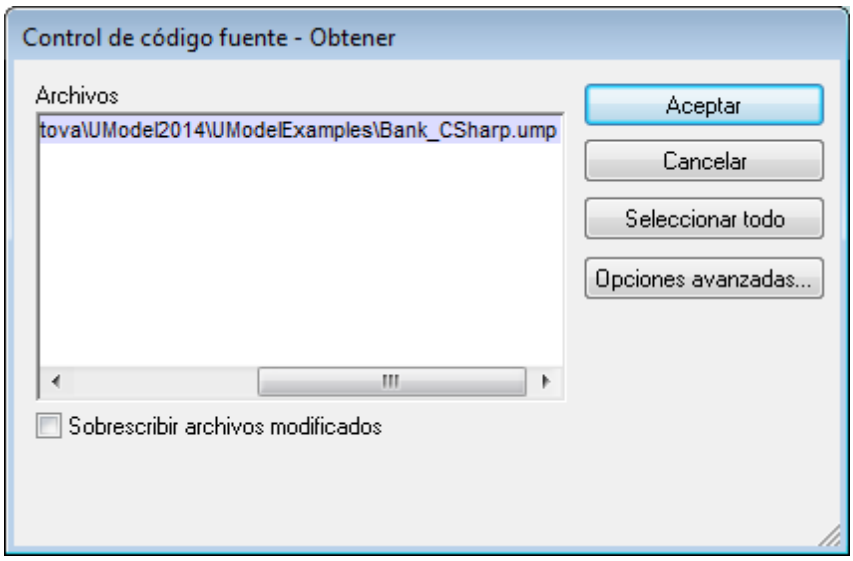
 © 2014 Altova GmbH

15.2.4 Obtener

Este comando recupera una copia de solo lectura de los archivos seleccionados y los coloca en la carpeta de trabajo. Los archivos no se desprotegen.

Para recuperar una copia de los archivos seleccionados:

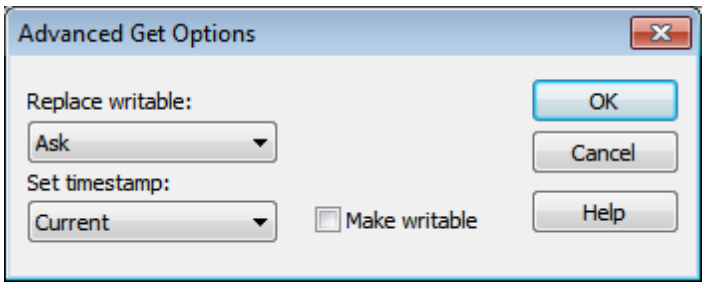
1. Seleccione los archivos en la Estructura del modelo.
2. Haga clic en **Proyecto | Control de código fuente | Obtener**.
   Aparece este cuadro de diálogo, cuyas opciones se describen más abajo:



Sobrescribir archivos modificados
Marque esta casilla si quiere sobrescribir los archivos que se modificaron localmente con los archivos de la BD del control de código fuente.

Seleccionar todo
Haga clic en este botón para seleccionar todos los archivos que aparecen en la lista del cuadro de diálogo.

Opciones avanzadas
El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están

desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

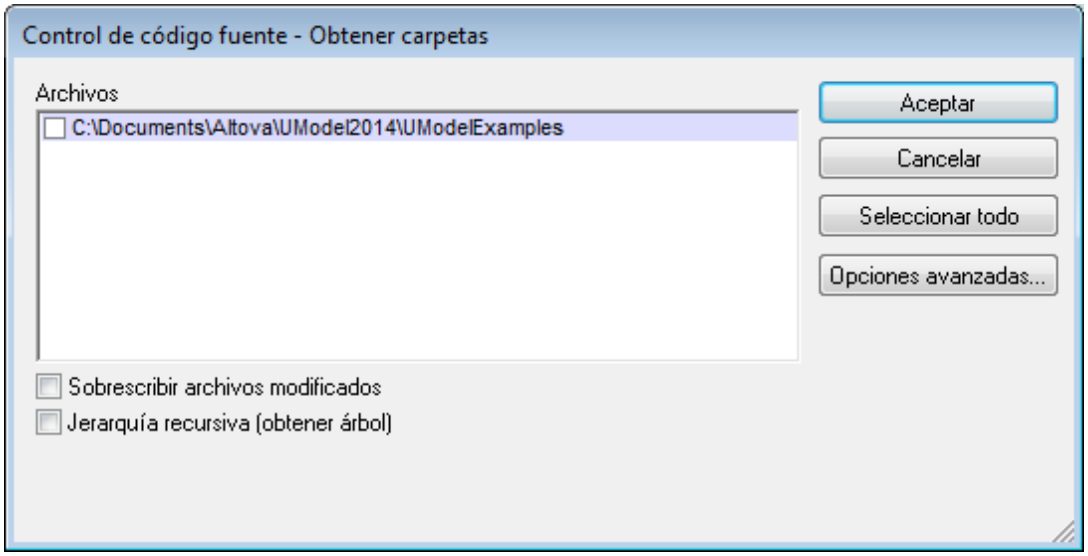
© 2014 Altova GmbH

15.2.5 Obtener carpetas

Este comando recupera una copia de solo lectura de los archivos de las carpetas seleccionadas y las coloca en la carpeta de trabajo. Los archivos no se desprotegen.

Para obtener una copia de los archivos de las carpetas seleccionadas:

1. En la Estructura del modelo seleccione qué carpetas desea obtener.
2. Haga clic en **Proyecto | Control de código fuente | Obtener**.
   Aparece este cuadro de diálogo, cuyas opciones se describen más abajo:

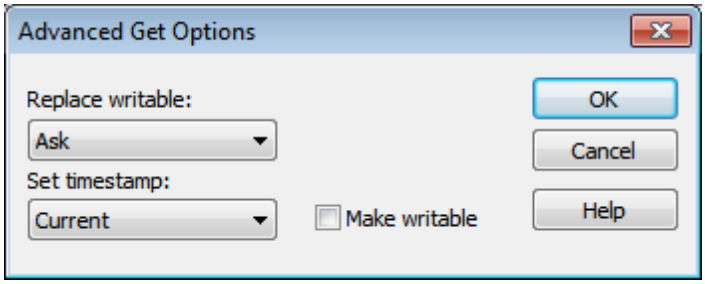


Sobrescribir archivos modificados
Marque esta casilla para sobrescribir los archivos que se modificaron localmente con los archivos de la BD del control de código fuente.

Jerarquía recursiva (obtener árbol)
Marque esta casilla para recuperar todos los archivos de la estructura de carpetas situada bajo la carpeta seleccionada.

Seleccionar todo
Haga clic en este botón para seleccionar todos los archivos que aparecen en la lista del cuadro de diálogo.

Opciones avanzadas
El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

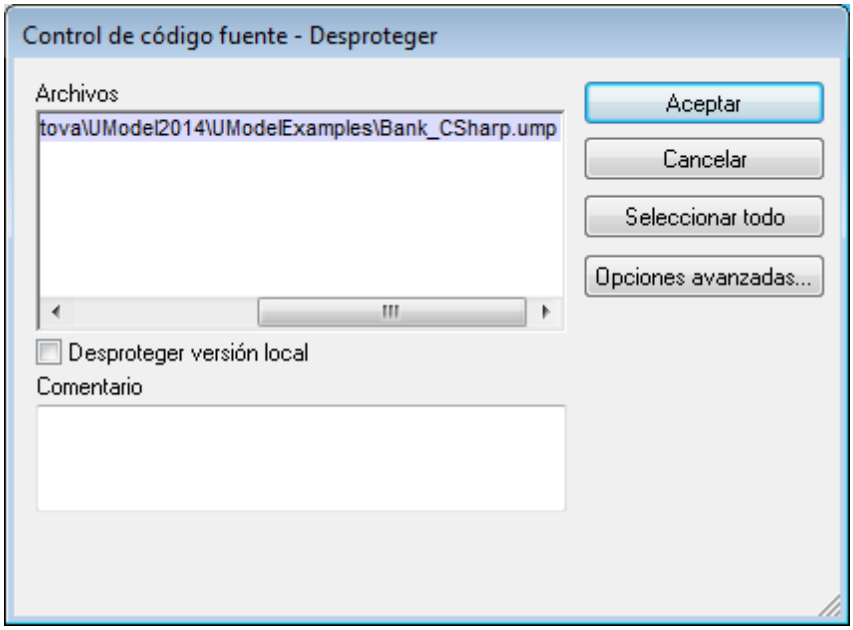
 © 2014 Altova GmbH

15.2.6 Desproteger

Este comando desprotege la versión más reciente de los archivos seleccionados y coloca una copia editable en el directorio de trabajo. Los otros usuarios ven un icono de "desprotegido" en los archivos desprotegidos.

Para desproteger archivos:

1. En la Estructura del modelo seleccione el archivo o la carpeta que desea desproteger.
2. Haga clic en **Proyecto | Control de código fuente | Desproteger**.
   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué archivos se desprotegen finalmente (marcando sus casillas). Más abajo se describen las opciones de este cuadro de diálogo.



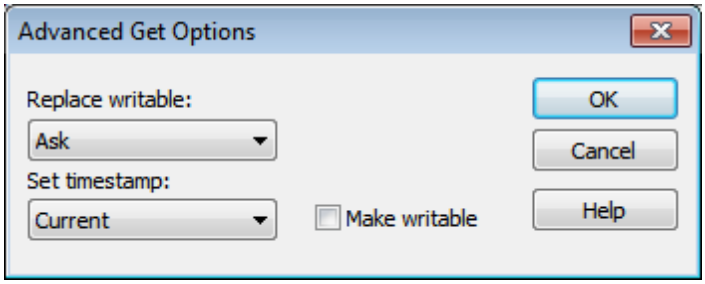
Desproteger versión local
Marque esta casilla para desproteger solamente las versiones locales de los archivos y no las versiones que están en la BD del control de código fuente.

Estos son los elementos que se pueden desproteger:

- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

La marca de verificación **roja** indica que el archivo / la carpeta **se desprotegió** para poder editarlo. Si además en la barra de título de UModel aparece un asterisco, significa que se realizaron cambios en el archivo y al cerrar la aplicación aparece un aviso para que guarde los cambios.

Opciones avanzadas
El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado se cambia para que el archivo se pueda escribir.

La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

 © 2014 Altova GmbH

15.2.7 Proteger

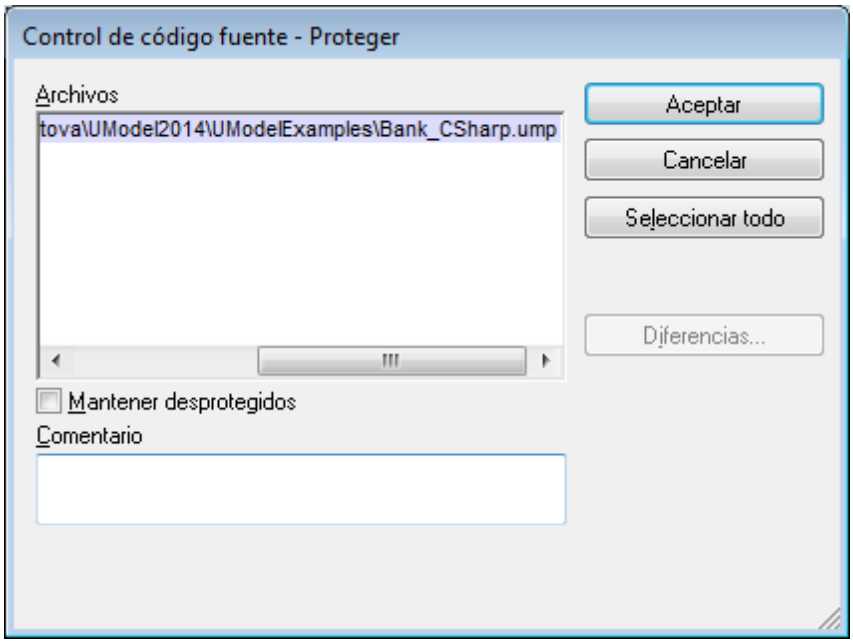
Este comando protege los archivos que estén desprotegidos (es decir, los archivos que se modificaron localmente) y los pone en la BD del control de código fuente.

Para proteger archivos:

1. En la Estructura del modelo seleccione los archivos que quiere proteger.
2. Haga clic en **Proyecto | Control de código fuente | Proteger**.

   Nota: también puede hacer clic con el botón secundario en los archivos que quiere proteger y seleccionar **Proteger** en el menú contextual.

   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué archivos se protegen finalmente (marcando sus casillas).



Estos son los elementos que se pueden proteger:

- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

o

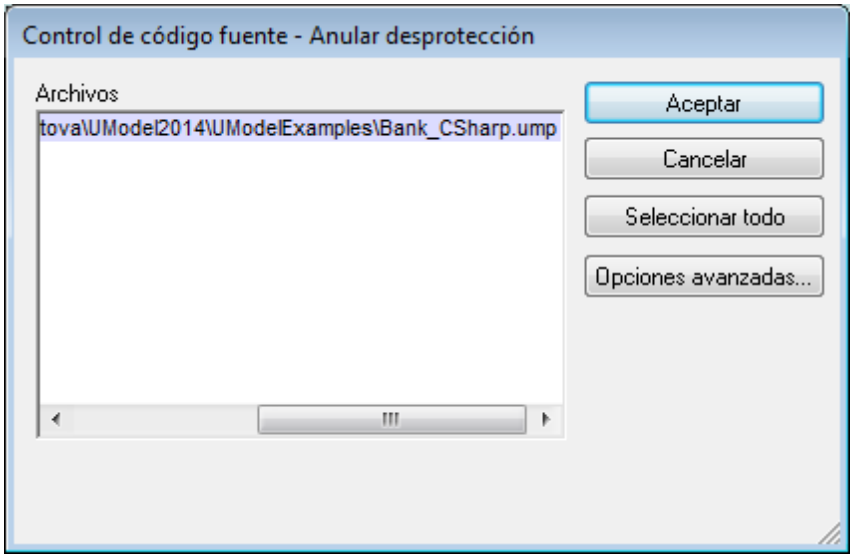
El icono en forma de candado indica que el archivo / la carpeta está **bajo control de código fuente** pero no está desprotegido.

15.2.8 Anular desprotección

Este comando rechaza los cambios realizados en archivos desprotegidos (es decir, los archivos que se modificaron localmente) y mantiene la versión previa de la BD del control de código fuente.

Para anular la desprotección:

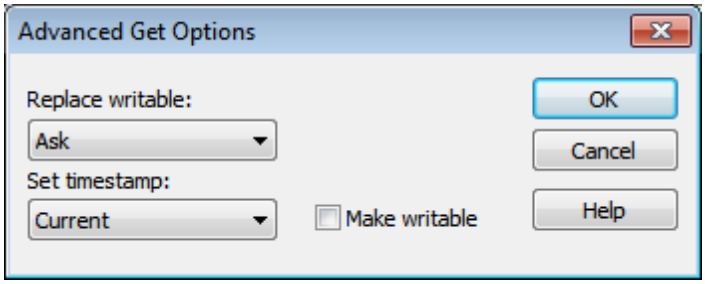
1. Seleccione los archivos correspondientes en la Estructura del modelo.
2. Haga clic en **Proyecto | Control de código fuente | Anular desprotección**. Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar para qué archivos se anula la desprotección finalmente (marcando sus casillas).



Estos son los elementos cuya desprotección se puede anular:

- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

Opciones avanzadas
El cuadro de diálogo "Opciones avanzadas" (*imagen siguiente*) se abre con el botón **Opciones avanzadas** del cuadro de diálogo "Obtener" (*primera imagen de este apartado*).



Aquí puede seleccionar (i) si se reemplazan los archivos que se pueden escribir y que están desprotegidos, (ii) la marca de tiempo y (iii) si la propiedad de solo lectura del archivo recuperado

 © 2014 Altova GmbH

se cambia para que el archivo se pueda escribir.

La casilla *Make writable* quita el atributo de solo lectura de los archivos recuperados.

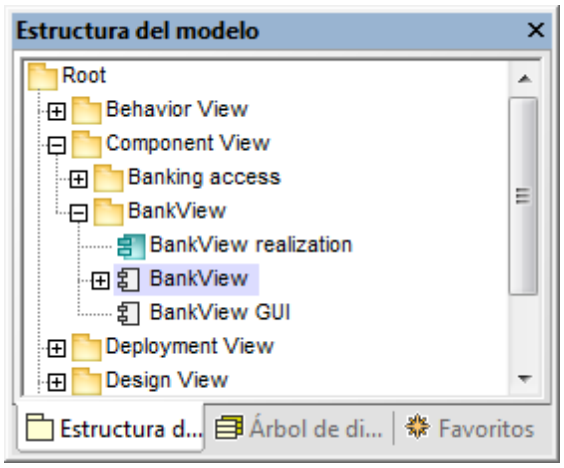
15.2.9 Agregar al control de código fuente

Este comando añade los archivos o las carpetas seleccionados a la BD del control de código fuente y los pone bajo su control. Si añade un proyecto de UModel nuevo, deberá indicar la carpeta del espacio de trabajo y la ubicación donde se debe almacenar el proyecto.

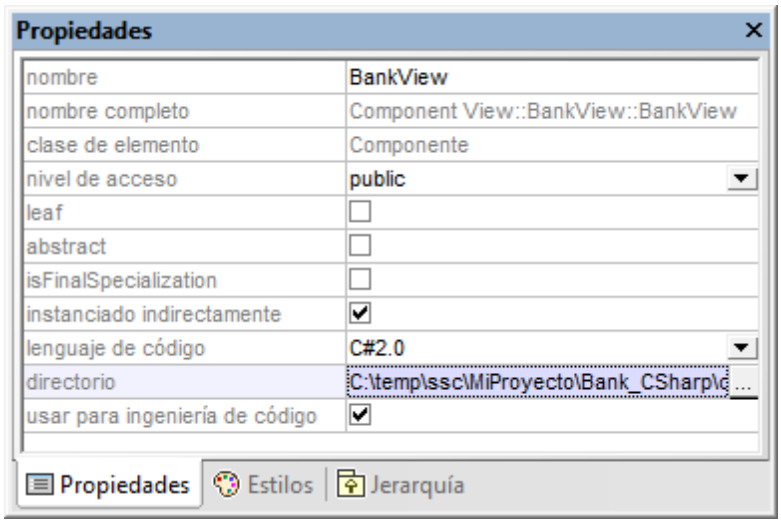
Tras poner el proyecto de UModel (`*.ump`) bajo control de código fuente, puede añadir al control de código fuente los **archivos de código** (creados durante el proceso de generación de código). Para el ejemplo que damos a continuación utilizamos el proyecto `Bank_CSharp.ump` y el directorio de trabajo `c:\temp\ssc\Bank_CSharp`.

Es importante tener en cuenta que para que esto funcione los archivos de código generados y el proyecto de UModel deben ponerse dentro/bajo el mismo **directorio de trabajo** de SourceSafe.

1. En la Estructura del modelo expanda la carpeta **`Component View`** y navegue hasta el componente **`BankView`**.



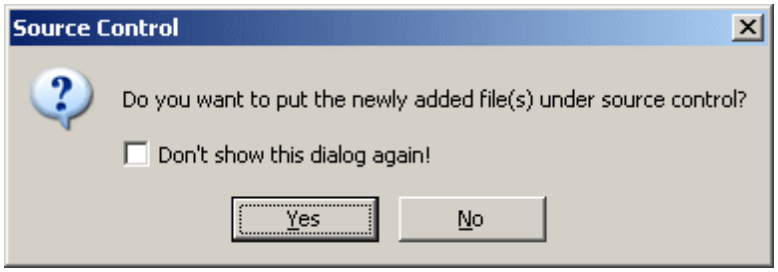
2. Haga clic en el componente **`BankView`**. En la ventana Propiedades haga clic en el icono **Examinar** del campo `directorio`.



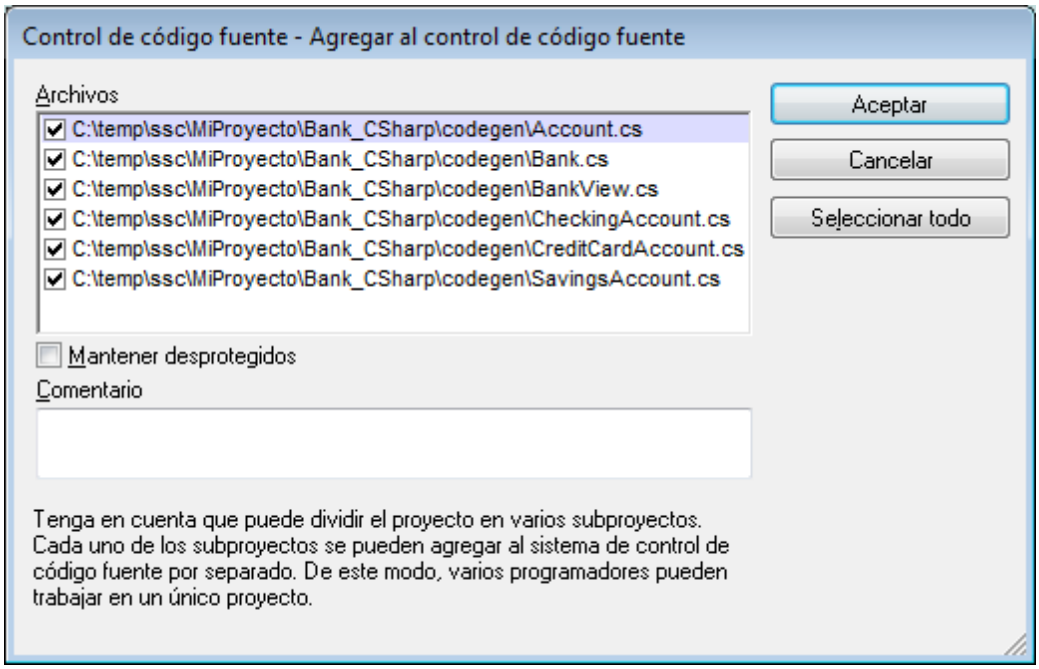
3. Cambie el directorio de ingeniería de código a `c:\temp\ssc\Bank_CSharp\codegen`
Si con anterioridad usó una carpeta distinta, aparece un cuadro de mensaje.

 © 2014 Altova GmbH

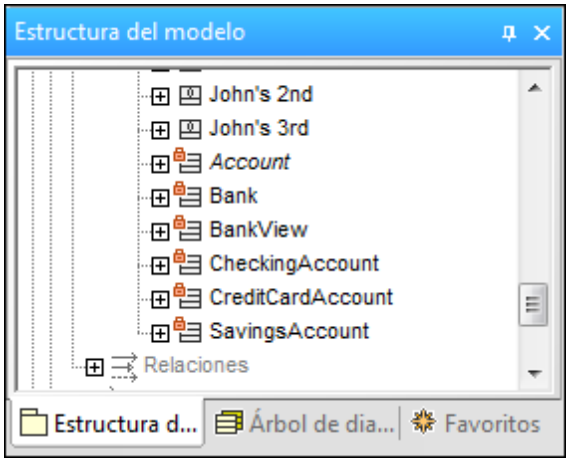
4. Haga clic en **Aceptar**.
5. Ahora haga clic en **Proyecto | Combinar el código de programa con el proyecto de UModel**.
6. Si es necesario, cambie las opciones de configuración y haga clic en **Aceptar**.
   La ventana Mensajes muestra cómo se desarrolla el proceso.
   Aparece un cuadro de mensajes que pregunta si desea poner los archivos recién creados bajo control de código fuente.



7. Haga clic en **Sí**.
8. Aparece el cuadro de diálogo "Agregar al control de código fuente" donde puede seleccionar qué archivos se ponen bajo control de código fuente.



9. Seleccione los archivos y haga clic en **Aceptar**.
   Observe que junto a las clases que ahora están bajo control de código fuente aparece un icono en forma de candado.

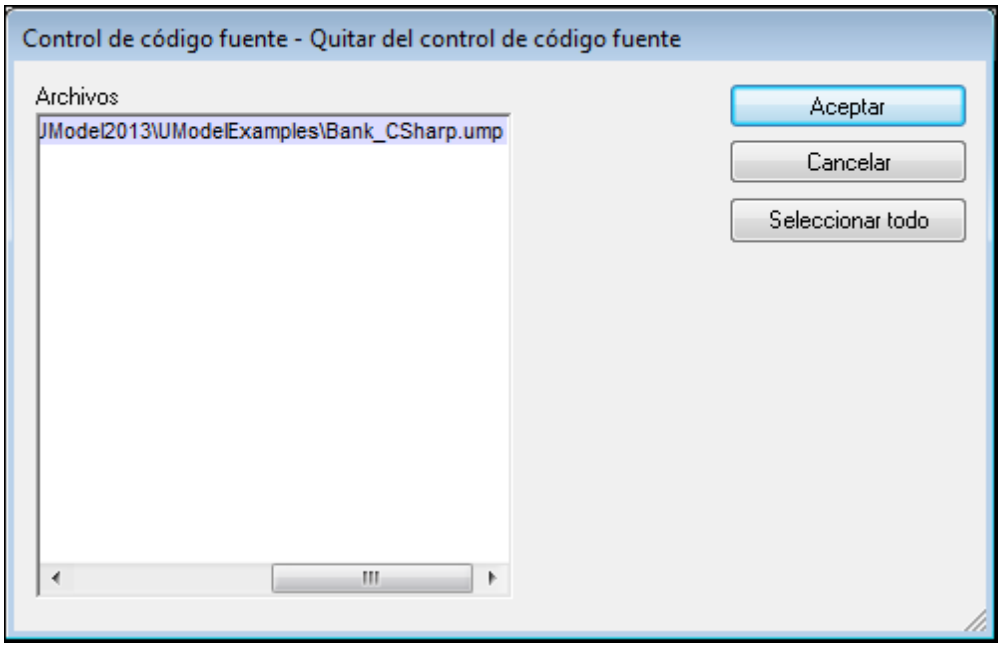

Estructura del modelo
John's 2nd
John's 3rd
Account
Bank
BankView
CheckingAccount
CreditCardAccount
SavingsAccount
Relaciones
Estructura d...
Árbol de dia...
Favoritos

15.2.10 Quitar del control de código fuente

Este comando quita de la BD del control de código fuente los archivos añadidos previamente a la BD. Este tipo de archivos siguen estando visibles en la Estructura del modelo pero no se pueden proteger ni desproteger. Para ponerlos otra vez bajo control de código fuente, utilice el comando **Agregar al control de código fuente**.

Para quitar archivos del control de código fuente:

1. Seleccione los archivos en la Estructura del modelo.
2. Haga clic en **Proyecto | Control de código fuente | Quitar del control de código fuente**.
   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué archivos se quitan del control de código fuente finalmente (marcando sus casillas).



Estos son los elementos que se pueden quitar del control de código fuente:

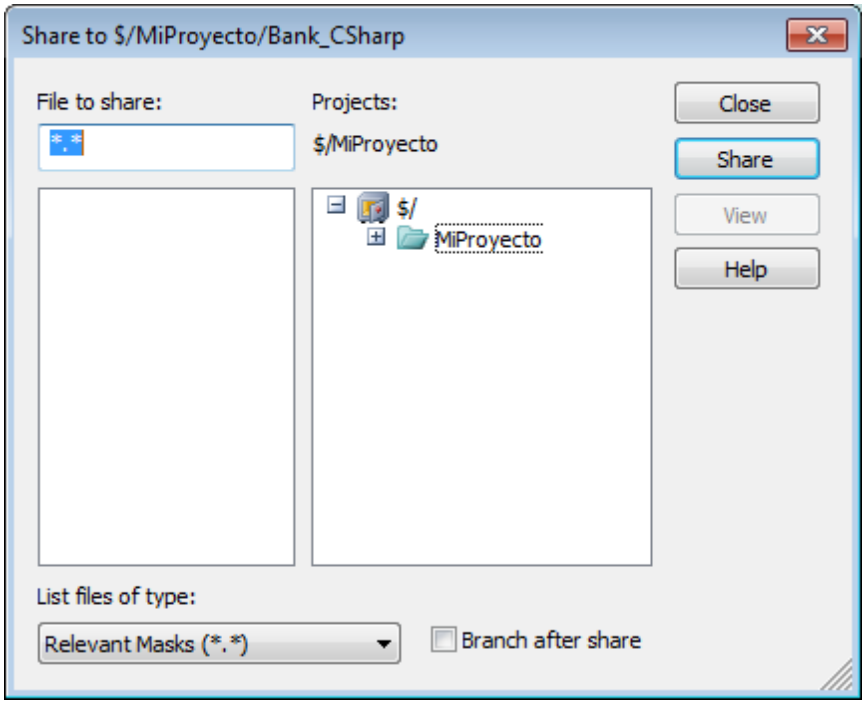
- Archivos (para seleccionar varios, pulse la tecla **Ctrl** mientras hace clic en los archivos en la Estructura del modelo).
- Carpetas (para seleccionar varias, pulse la tecla **Ctrl** mientras hace clic en las carpetas en la Estructura del modelo).

15.2.11 Compartir desde el control de código fuente

Este comando comparte/ramifica archivos de otros proyectos/carpetas del repositorio de control de código fuente con la carpeta seleccionada. Para usar este comando es necesario tener privilegios para proteger/desproteger datos en el proyecto desde el que se comparten los archivos.

Para compartir un archivo desde el control de código fuente:

1. En la Estructura del modelo seleccione la carpeta con la que quiere compartir archivos (p. ej. `BankView Component` de la carpeta `Component View`).
2. Haga clic en **Proyecto | Control de código fuente | Compartir desde el control de código fuente**.
   Aparece un cuadro de diálogo (*imagen siguiente*) donde puede seleccionar qué carpeta de proyecto contiene el archivo que desea compartir.



3. Seleccione el archivo que desea compartir y haga clic en el botón **Share**.
   El archivo se elimina de la lista *File to share*.
4. Haga clic en el botón **Close** para continuar.

   Nota: marque la casilla *Branch after share* para compartir el archivo y crear una rama nueva para crear una nueva versión.

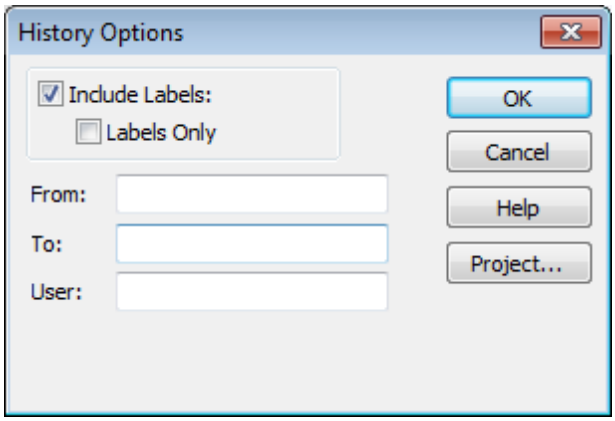
 © 2014 Altova GmbH

15.2.12 Mostrar historial

Este comando **muestra el historial** de un archivo que está bajo control de código fuente. El historial permite ver la historia de cambios por la que ha pasado el archivo, ver diferencias entre sus diferentes versiones y recuperar versiones previas.

Para ver el historial de un archivo:

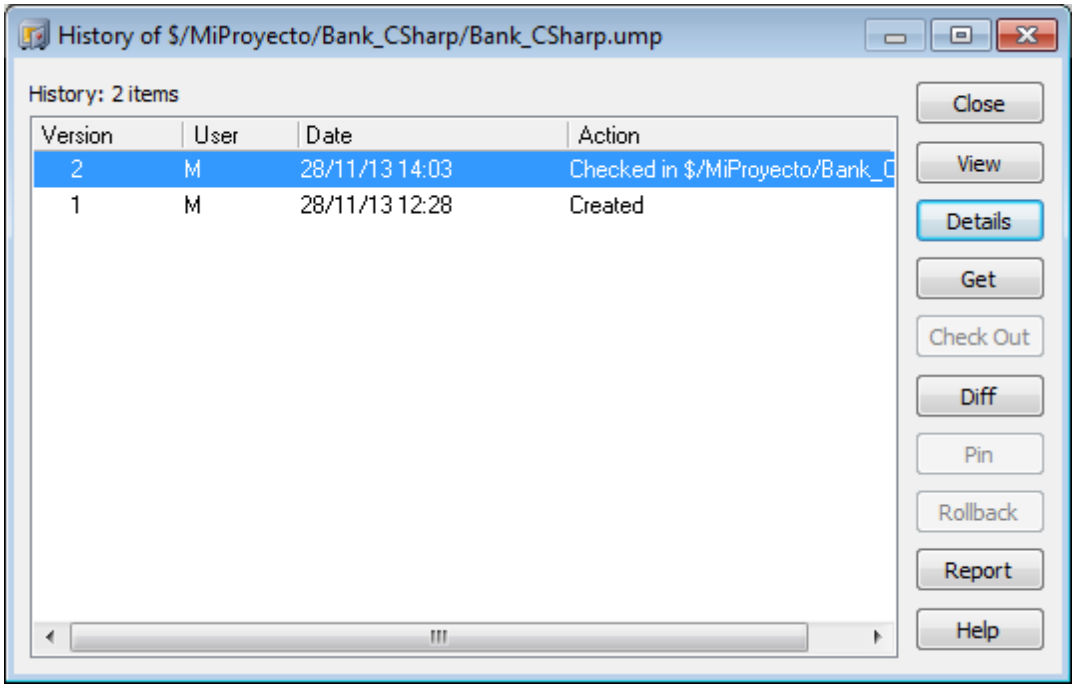
1. Seleccione el archivo en la Estructura del modelo.
2. Haga clic en el comando **Proyecto | Control de código fuente | Mostrar historial**.
   Aparece un cuadro de diálogo pidiendo más información.



3. Inserte la información pertinente y haga clic en **OK**.
4. Aparece un cuadro de diálogo (*imagen siguiente*) que muestra el historial propiamente dicho.

   En este cuadro de diálogo puede comparar versiones del archivo y obtener versiones previas.

   Para ver el historial detallado haga doble clic en una entrada de la lista.



Estos son los botones del cuadro de diálogo del historial:

Close
Cierra el cuadro de diálogo.

View
Abre otro cuadro de diálogo donde puede seleccionar en qué aplicación desea ver el archivo.

Details
Abre un cuadro de diálogo que muestra las propiedades del archivo seleccionado.

Get
Recupera una de las versiones previas del archivo y la coloca en el directorio de trabajo.

Check Out
Desprotege la versión **más reciente** del archivo.

Diff
Abre el cuadro de diálogo "Difference options" donde puede configurar la vista de las diferencias detectadas entre las dos versiones del archivo.
Para marcar dos versiones de un archivo pulse la tecla **Ctrl** mientras hace clic en las entradas. Después pulse el botón **Diff** para ver las diferencias.

Pin
Ancla/desancla una versión del archivo para que pueda definir la versión que se debe usar en la comparación de dos archivos.

Rollback
Revierte a la versión seleccionada del archivo.

 © 2014 Altova GmbH

Report
Genera un historial que puede imprimirse, guardarse en un archivo o copiarse en el portapapeles.

Help
Abre la ayuda en pantalla del cliente de control de código fuente.

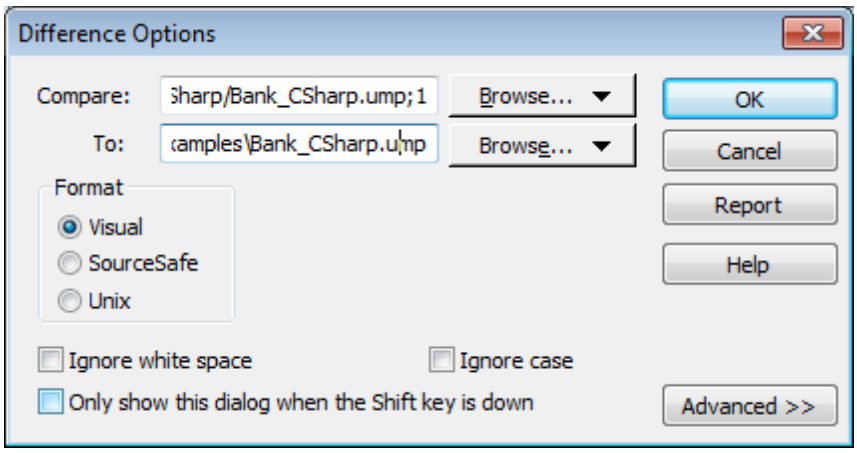
15.2.13 Mostrar diferencias

Este comando muestra las diferencias que existen entre el archivo que está en el repositorio del control de código fuente y el mismo archivo protegido/desprotegido del directorio de trabajo.

Si ancló uno de los archivos en el cuadro de diálogo del historial, el archivo anclado se inserta automáticamente en el campo de texto *Compare*. Con los botones **Browse** puede buscar los archivos que desea comparar.

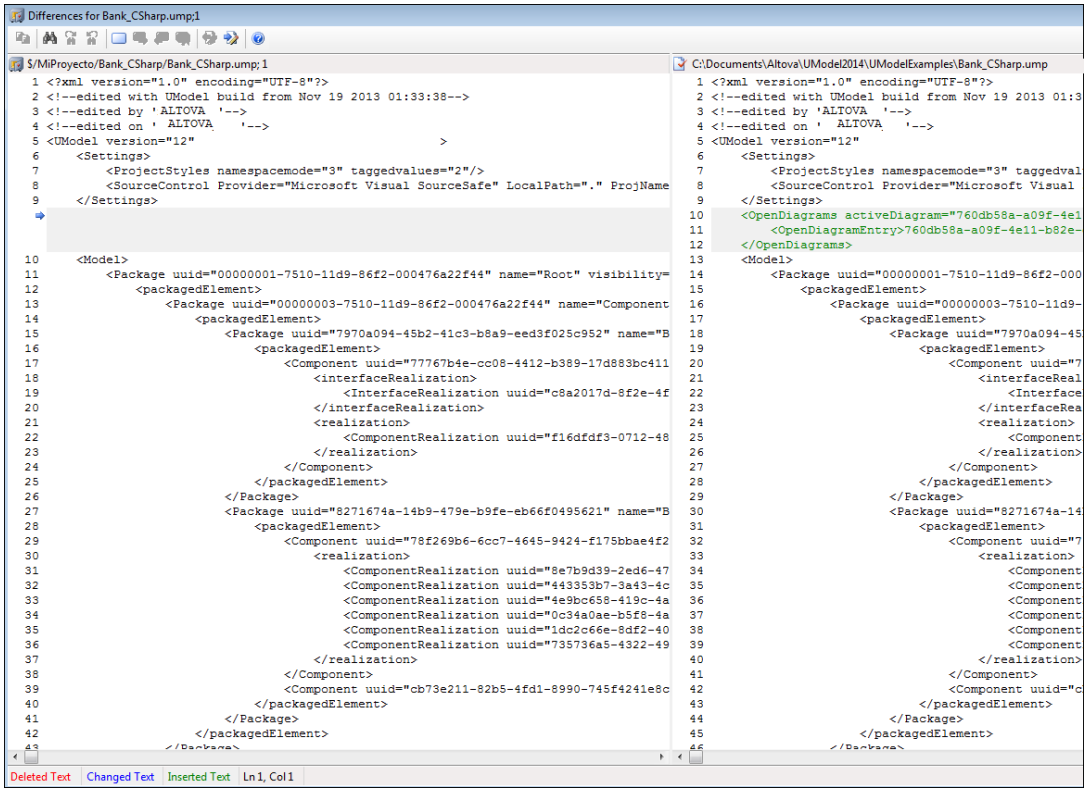
Para ver las diferencias que hay entre dos archivos:

1. En la Estructura del modelo seleccione el archivo que desea comparar.
2. Haga clic en **Proyecto | Control de código fuente | Mostrar diferencias**. Aparece un cuadro de diálogo que le pide más información.



3. Seleccione el archivo con el que desea comparar el primer archivo, elija las opciones pertinentes y haga clic en **OK**.

 © 2014 Altova GmbH



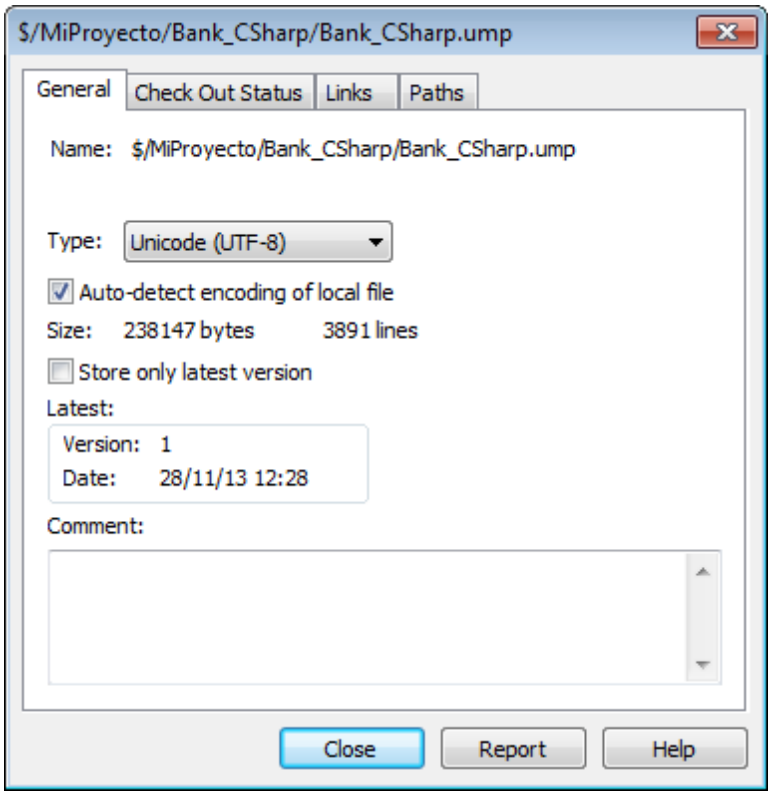
Las diferencias detectadas aparecen resaltadas. Por ejemplo, la imagen anterior muestra los resultados de la comparación en MS SourceSafe.

15.2.14 Mostrar propiedades

Este comando muestra las propiedades del archivo seleccionado y varía de un proveedor de control de código fuente a otro.

Para ver las propiedades del archivo seleccionado haga clic en **Proyecto | Control de código fuente | Mostrar propiedades**.

Tenga en cuenta que este comando no se puede ejecutar en varios archivos a la vez.



 © 2014 Altova GmbH

15.2.15 Actualizar estado

Este comando **actualiza** el estado de todos los archivos de proyecto, independientemente de cuál sea su estado actual.

15.2.16 Administrador del control de código fuente

Este comando **inicia** el software de control de código fuente, con su interfaz de usuario nativa.

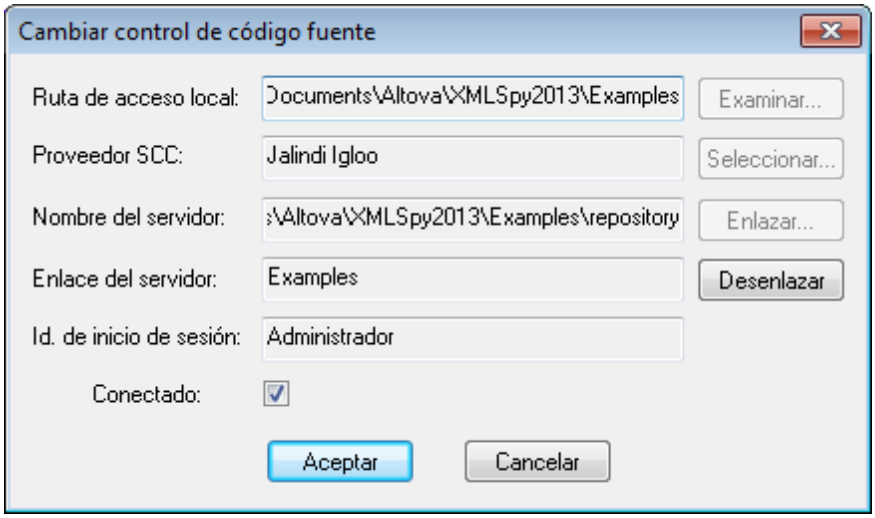
© 2014 Altova GmbH

15.2.17 Cambiar control de código fuente

Este cuadro de diálogo sirve para cambiar el enlace de control de código fuente activo.

Para cambiar de control de código fuente:

1. Haga clic en **Proyecto | Control de código fuente | Cambiar control de código fuente**.
   Aparece este cuadro de diálogo:



2. Haga clic en el botón **Desenlazar** y después (si quiere) haga clic en el botón **Seleccionar** para seleccionar un proveedor nuevo.
3. Para terminar haga clic en el botón **Enlazar** para enlazar el proyecto con una ubicación nueva del repositorio.

Altova UModel® 2015

Iconos en los diagramas de UModel

16 Iconos en los diagramas de UModel

En UModel cada tipo de diagrama tiene una barra de herramientas distinta. Cada barra de herramientas ofrece iconos para los elementos compatibles con el tipo de diagrama correspondiente.

Estos iconos pueden ser de dos tipos:

- **Agregar:** en este grupo están los iconos de todos los elementos que se pueden agregar en el diagrama.
- **Relación:** en este grupo están todos los iconos de los tipos de relación que se pueden crear entre los elementos del diagrama.

© 2014 Altova GmbH

16.1 Diagramas de actividades

Agregar:
Acción (AcciónLlamadaDeComportamiento)
Acción (AcciónOperaciónDeLlamada)
AcciónAceptarEvento
AcciónAceptarEvento (EventoDeTiempo)
AcciónEnviarSeñal

NodoDeDecisión (rama)
NodoDeCombinación
NodoInicial
NodoFinalDeActividad
NodoFinalDeFlujo
NodoDeBifurcación (vertical)
NodoDeBifurcación (horizontal)
NodoDeReunión
NodoDeReunión (horizontal)

PinDeEntrada
PinDeSalida
PinDeValor

NodoDeObjeto
NodoDeBúferCentral
NodoAlmacénDeDatos
ParticiónDeActividades (horizontal)
ParticiónDeActividades (vertical)
ParticiónDeActividades (2D)

FlujoDeControl
FlujoDeObjeto
ControladorDeExcepción

Actividad
NodoParámetroDeActividad
NodoDeActividadEstructurada
RegiónDeExpansión
NodoDeExpansión
RegiónDeActividadInterrumpible

Nota
Enlace de nota

© 2014 Altova GmbH

16.2 Diagramas de clases

Relaciones:
Asociación
Agregación
Composición
ClaseDeAsociación
Dependencia
Utilización
RealizaciónDeInterfaz
Generalización

Agregar:
Paquete
Clase
Interfaz
Enumeración
TipoDeDatos
TipoPrimitivo
Perfil
Estereotipo
AplicaciónDePerfil
EspecificaciónDeInstancia

Nota
Enlace de nota

16.3 Diagramas de comunicación

Agregar:
LíneaDeVida
Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje (Creación)
Mensaje (Destrucción)

Nota
Enlace de nota

 © 2014 Altova GmbH

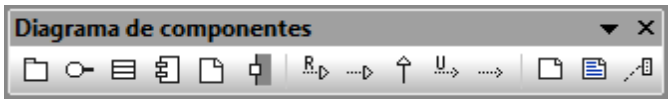
16.4 Diagramas de estructura de un compuesto

Agregar:
Colaboración
UsoDeColaboración
Parte (Propiedad)
Clase
Interfaz
Puerto

Relaciones:
Conector
Dependencia (Enlace de roles)
RealizaciónDeInterfaz
Utilización

Nota
Enlace de nota

16.5 Diagramas de componentes



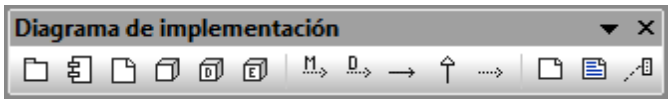
Agregar:
Paquete
Interfaz
Clase
Componente
Artefacto

Relaciones:
Realización
RealizaciónDeInterfaz
Utilización
Dependencia

Nota
Enlace de nota

 © 2014 Altova GmbH

16.6 Diagramas de implementación



Agregar:
Paquete
Componente
Artefacto
Nodo
Dispositivo
EntornoDeEjecución

Relaciones:
Manifestación
Implementación
Asociación
Generalización
Dependencia

Nota
Enlace de nota

16.7 Diagramas global de interacción

Agregar:
AcciónLlamadaDeComportamiento (Interacción)
AcciónLlamadaDeComportamiento (UsoDeInteracción)
NodoDeDecisión
NodoDeCombinación
NodoInicial
NodoFinalDeActividad
NodoDeBifurcación
NodoDeBifurcación (Horizontal)
NodoDeReunión
NodoDeReunión (Horizontal)
RestricciónDeDuración

Relaciones:
FlujoDeControl

Nota
Enlace de nota

 © 2014 Altova GmbH

16.8 Diagramas de objetos

Agregar:
Paquete
Clase
Interfaz
Enumeración
TipoDeDatos
TipoPrimitivo
EspecificaciónDeInstancia

Relaciones:
Asociación
ClaseDeAsociación
Dependencia
Utilización
RealizaciónDeInterfaz
Generalización

Nota
Enlace de nota

16.9 Diagramas de paquetes

Agregar:
Paquete
Perfil

Relaciones:
Dependencia
ImportaciónDePaquete
CombinaciónDePaquete
AplicaciónDePerfil

Nota
Enlace de nota

© 2014 Altova GmbH

16.10 Diagramas de perfil

Agregar:
Perfil
Estereotipo

Relaciones:
Generalización
AplicaciónDePerfil
ImportaciónDePaquete
ImportaciónDeElemento

Nota
Enlace de nota

16.11 Diagramas de máquina de estados de protocolos

Agregar:
Estado
Estado compuesto
Estado ortogonal
Estado de submáquina

EstadoFinal
EstadoInicial

PuntoDeEntrada
PuntoDeSalida
Elección
Unión
Terminar
Bifurcación
Bifurcación (horizontal)
Reunión
Reunión (horizontal)
ReferenciaDePuntoDeConexión

Relaciones:
TransiciónDeProtocolo

Nota
Enlace de nota

 © 2014 Altova GmbH

16.12 Diagramas de secuencia

Agregar
LíneaDeVida
FragmentoCombinado
FragmentoCombinado (Alternativos)
FragmentoCombinado (Bucle)
UsoDeInteracción
Puerta
InvarianteDeEstado
RestricciónDeDuración
RestricciónDeTiempo

Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje (Creación)
Mensaje (Destrucción)

Mensaje asíncrono (Llamada)
Mensaje asíncrono (Respuesta)
Mensaje asíncrono (Destrucción)

Mensajes sin numeración
Mensajes con numeración sencilla
Mensajes con notación decimal anidada

Nota
Enlace de nota

Activar/desactivar el movimiento de mensajes dependientes
Activar/desactivar la creación automática de respuestas para mensajes (Llamada)
Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

16.13 Diagramas de máquina de estados

Agregar:
Estado
Estado compuesto
Estado ortogonal
Estado de submáquina

EstadoFinal
EstadoInicial

PuntoDeEntrada
PuntoDeSalida
Elección
Unión
Terminar
Bifurcación
Bifurcación (horizontal)
Reunión
Reunión (horizontal)
HistorialDetallado
HistorialSuperficial
ReferenciaDePuntoDeConexión

Relaciones:
Transición

Nota
Enlace de nota

Activar/desactivar la creación automática de operaciones en el destino al escribir el nombre de la operación

 © 2014 Altova GmbH

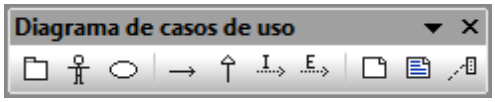
16.14 Diagramas de ciclo de vida

Agregar
LíneaDeVida (Estado o Condición)
LíneaDeVida (Valor general)
MarcaDeGraduación
Evento/estímulo
RestricciónDeDuración
RestricciónDeTiempo

Mensaje (Llamada)
Mensaje (Respuesta)
Mensaje asíncrono (Llamada)

Nota
Enlace de nota

16.15 Diagramas de casos de uso



Agregar:
Paquete
Actor
CasoDeUso

Relaciones:
Asociación
Generalización
Inclusión
Extensión

Nota

 © 2014 Altova GmbH

16.16 Diagramas de esquema XML

Agregar:
targetNamespace 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 XSD

Relaciones:
include XSD
redefine XSD
restriction XSD
extension XSD
substitution XSD

Nota
Enlace de nota

16.17 Diagramas BPMN

Agregar:
Evento inicial
Evento intermedio
Evento final

Tarea
Tarea de ciclo
Tarea (Instancia múltiple)
Tarea (Compensación)

Subproceso contraído
Subproceso de ciclo contraído
Subproceso de instancia múltiple contraído
Subproceso ad hoc contraído
Subproceso de compensación contraído

Subproceso expandido
Subproceso de ciclo expandido
Subproceso de instancia múltiple expandido
Subproceso ad hoc expandido
Subproceso de compensación expandido

Compuerta
Compuerta inclusiva (OR)
Compuerta paralela (AND)
Compuerta exclusiva basada en datos (XOR)
Compuerta exclusiva basada en eventos (XOR)
Compuerta compleja (bifurcación/convergencia)

Relaciones:
Flujo de secuencia
Flujo condicional
Flujo por defecto
Flujo de mensaje
Asociación

Contenedor
Objeto de datos
Grupo

Anotación textual
Asociación por anotación

 © 2014 Altova GmbH

16.18 Diagramas BPMN 2.0

Agregar:
Evento inicial
Evento de captura
Evento de lanzamiento
Evento final

Tarea
Subproceso expandido
Subproceso contraído
Actividad de llamada
Compuerta

Relaciones:
Flujo de secuencia
Flujo de secuencia (por defecto)
Flujo de secuencia (condicional)
Flujo de mensaje
Asociación

Contenedor
Grupo
Objeto de datos
Datos de salida
Datos de entrada
Colección de objetos de datos
Almacén de datos
Mensaje

Anotación textual
Asociación por anotación

16.19 Modelado de bases de datos

Agregar:
Tabla
RestricciónDeComprobación
ClavePrincipal
ClaveForánea
ClaveÚnica
Índice

Relaciones:
Asociación de relación entre bases de datos
Relación entre bases de datos con atributos

 © 2014 Altova GmbH

Altova UModel® 2015

Referencia del usuario

17 Referencia del usuario

Esta sección repasa uno a uno todos los menús y comandos de menú de UModel, dando una breve descripción de ellos.

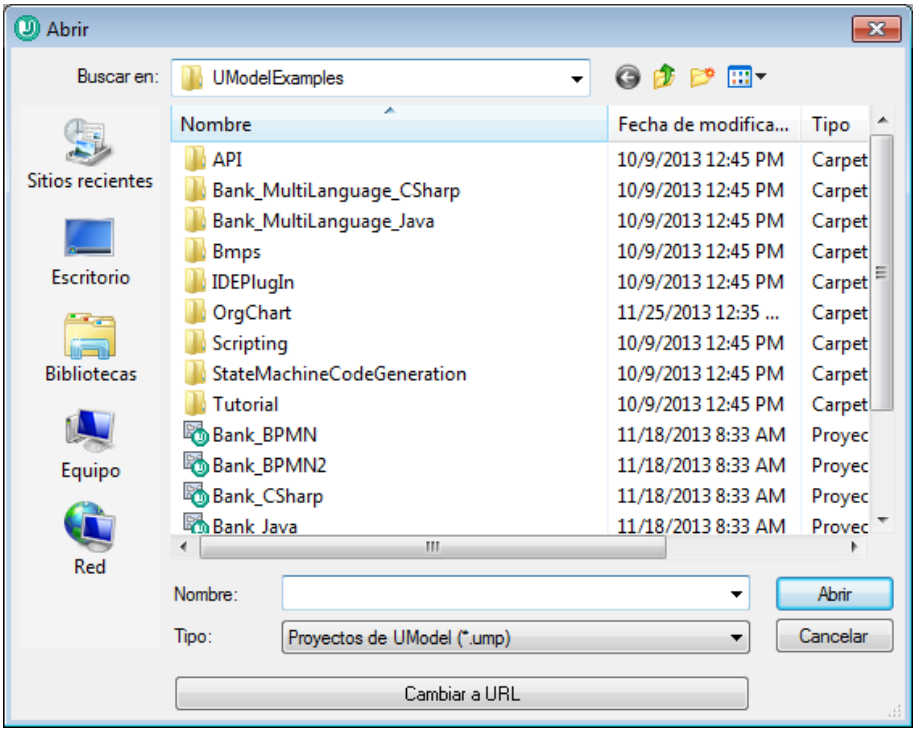
© 2014 Altova GmbH

17.1 Menú Archivo

Nuevo
Si tiene abierto un proyecto, este comando lo cierra y crea un proyecto nuevo.

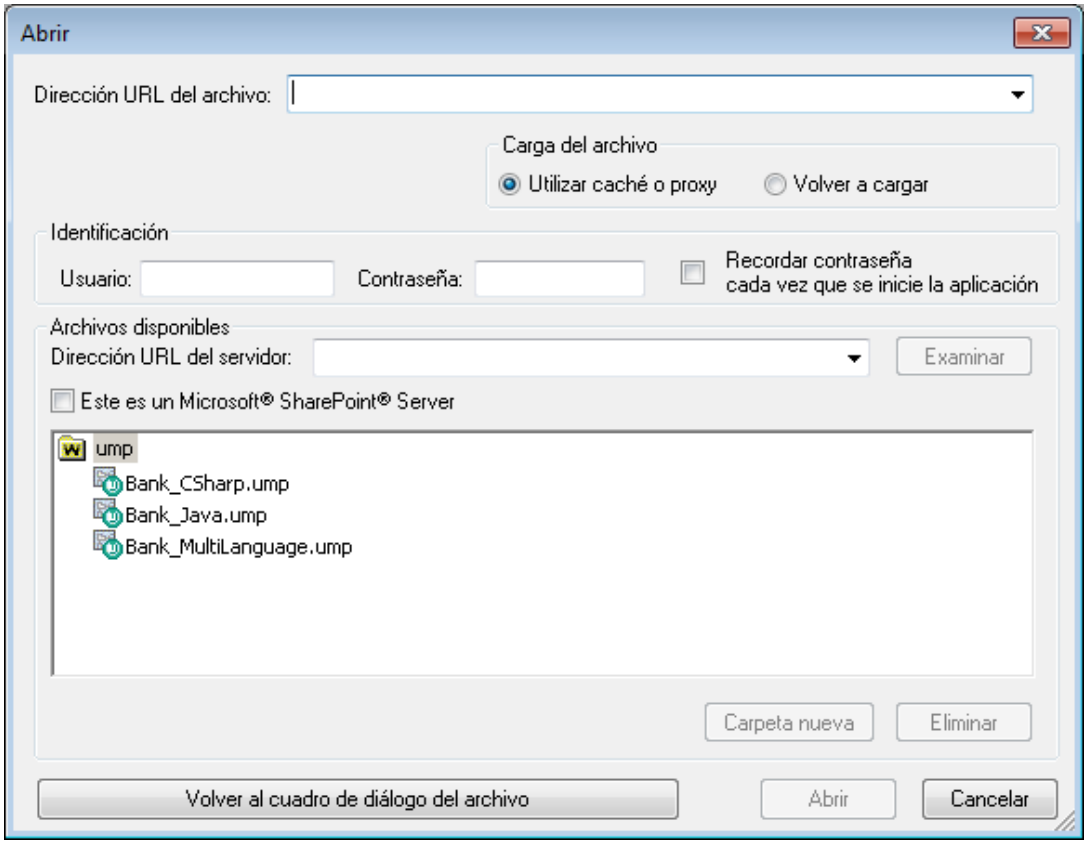
Abrir
Este comando abre un proyecto de modelado ya existente. Seleccione el archivo de proyecto `*.ump` en el cuadro de diálogo "Abrir" (*imagen siguiente*).

Observe que en la parte inferior del diálogo aparece el botón **Cambiar a URL**.



El botón **Cambiar a URL** cambia el cuadro de diálogo al modo URL, que sirve para abrir archivos de proyecto de UModel desde una dirección URL.

1. Escriba la dirección URL a la que desea acceder (en el campo *Dirección URL del servidor*).
2. Escriba el nombre de usuario y la contraseña (si el servidor está protegido con contraseña).
3. Haga clic en el archivo que desea cargar en UModel. Si el servidor es Microsoft® SharePoint®, marque la casilla *Este es un Microsoft® SharePoint® Server*. Para más información consulte las notas que aparecen más abajo.



La URL del archivo aparece en el campo *Dirección URL del archivo* (en la parte superior del cuadro de diálogo).
El botón **Abrir** se habilita en este momento.

4. Haga clic en el botón **Abrir** para cargar el archivo. El archivo aparece en la ventana principal de la aplicación.

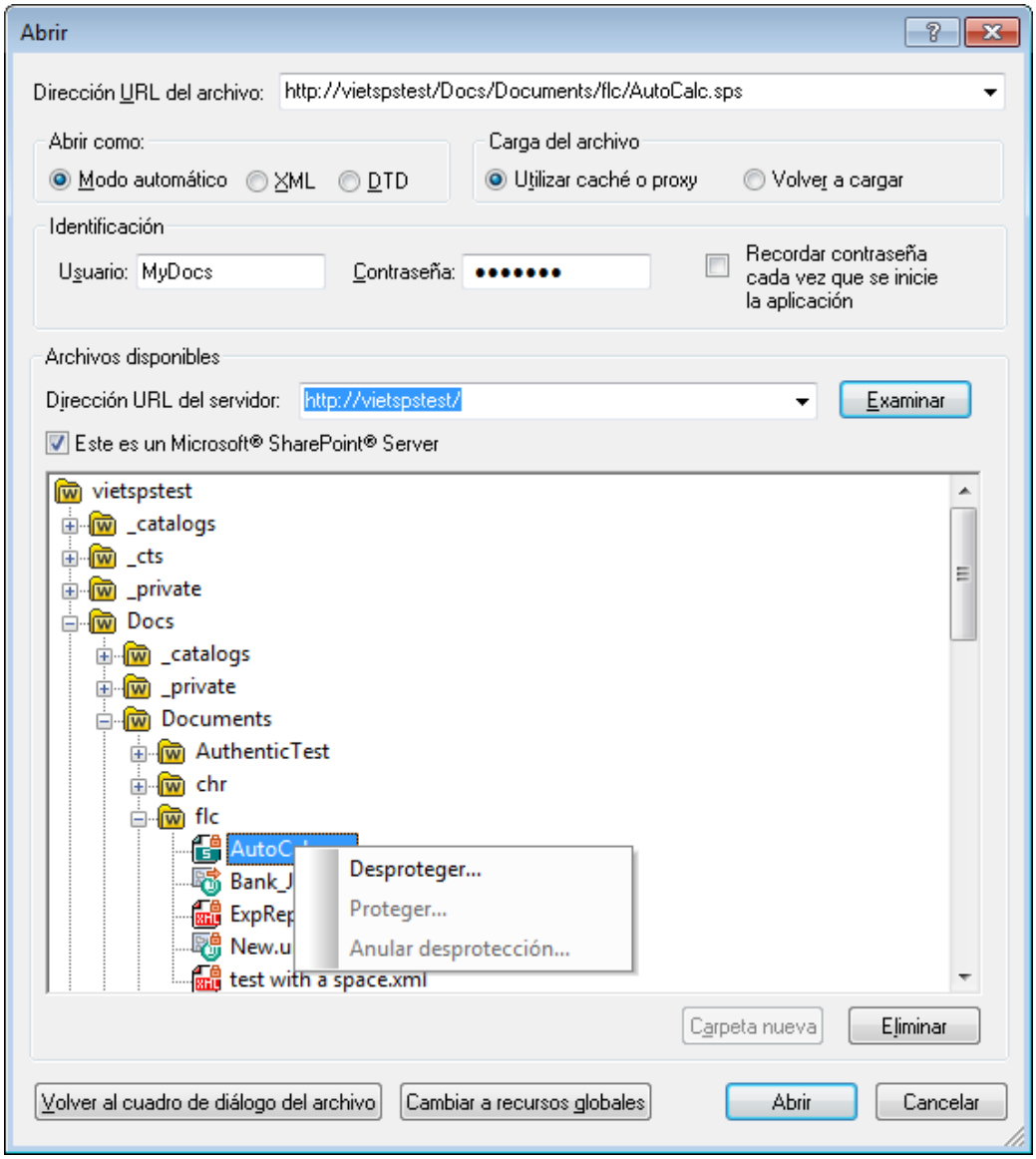
   Nota importante: el botón **Examinar** solamente se puede usar con servidores compatibles con WebDAV y con servidores Microsoft SharePoint. Los protocolos compatibles son FTP, HTTP y HTTPS.

   Además puede cargar el archivo por la memoria caché local o por un servidor proxy (lo cual acelera el proceso considerablemente si ya el archivo con anterioridad). Por último, si está trabajando en un sistema de publicación electrónica, puede volver a cargar el archivo seleccionando la opción *Volver a cargar*.

Notas sobre servidores Microsoft® SharePoint® Server
Tenga en cuenta estos aspectos de los servidores Microsoft® SharePoint® Server:

- En la estructura de directorios que aparece en el panel *Archivos disponibles* (*imagen siguiente*), los iconos de archivo tienen símbolos que indican si estos están protegidos o desprotegidos.

 © 2014 Altova GmbH



Al hacer clic con el botón secundario en un archivo aparece un menú contextual (*imagen anterior*).

- Estos son los diferentes iconos de archivo posibles:

	Protegido. Se puede desproteger.
	Desprotegido por otro usuario. No se puede desproteger.
	Desprotegido localmente. Se puede editar y después proteger.

- Después de desproteger un archivo puede editarlo en su aplicación de Altova y guardarlo con **Archivo | Guardar (Ctrl+S)**.
- Puede proteger el archivo editado con el menú contextual en el cuadro de diálogo "Abrir" (*imagen anterior*) o con el menú contextual que aparece al hacer clic con el botón secundario en la pestaña del archivo en la ventana principal de la aplicación (*imagen siguiente*).

- Cuando un archivo está desprotegido por otro usuario, no se puede desproteger.
- Cuando un archivo está desprotegido localmente (por usted), puede deshacer la desprotección con el comando **Deshacer desprotección** del menú contextual. Como resultado se devuelve el archivo al servidor sin ningún cambio.
- Si desprotege un archivo en una aplicación de Altova, no puede desprotegerlo en otra aplicación de Altova. En ese caso los comandos disponibles en la aplicación de Altova son **Proteger** y **Deshacer desprotección**.

Volver a cargar
Este comando sirve para volver a cargar el proyecto y guardar (o descartar) los cambios realizados hasta ese momento.

Guardar
Este comando guarda el proyecto de modelado activo con el nombre de archivo actual.

Guardar como
Este comando guarda el proyecto de modelado activo con otro nombre. También ofrece la opción de darle al proyecto un nombre si es la primera vez que se guarda.

Guardar copia como
Este comando guarda una copia del proyecto de modelado activo con otro nombre de archivo.

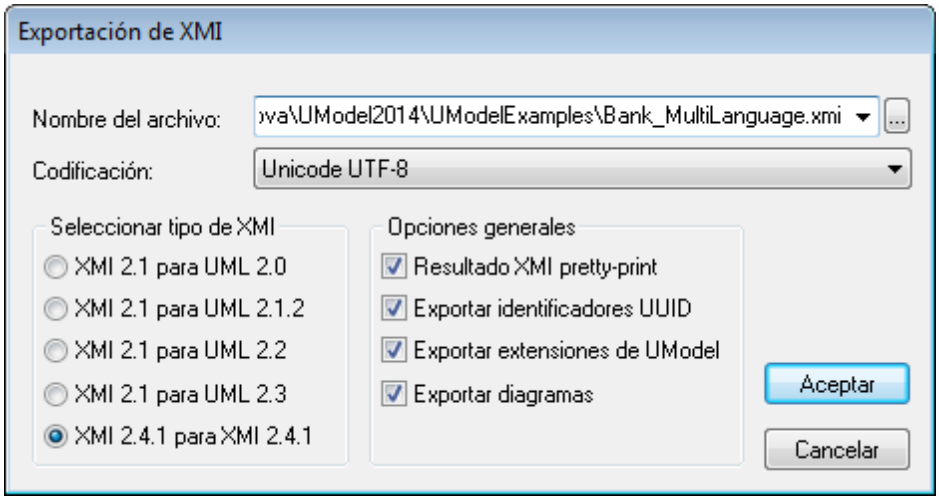
Guardar el diagrama como imagen
Este comando abre el cuadro de diálogo "Guardar como" y permite guardar el diagrama activo como archivo .PNG o .EMF (metarchivo mejorado). También puede guardar archivos PNG de gran tamaño (1GB o más).

Guardar todos los diagramas como imagen
Este comando guarda todos los diagramas del proyecto activo como archivos .PNG o .EMF (metarchivo mejorado).

Importar desde un archivo XMI
Este comando importa un archivo XMI exportado con anterioridad. Si el archivo se generó con UModel, toda las extensiones y demás elementos se conservarán.

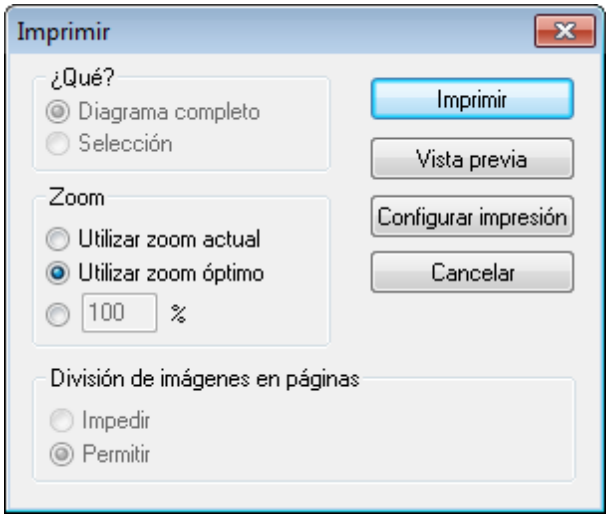
Exportar a un archivo XMI
Este comando exporta el modelo como archivo XMI (*imagen siguiente*). Puede seleccionar la versión de UML y los identificadores que desea exportar. Para más información consulte el apartado XMI: intercambio de metadatos XML.

 © 2014 Altova GmbH



Enviar por correo electrónico
Este comando abre la aplicación predeterminada de correo electrónico e inserta el proyecto de UModel actual como archivo adjunto.

Imprimir
Este comando abre el cuadro de diálogo "Imprimir" (*imagen siguiente*), desde donde puede imprimir una copia en papel del proyecto de modelado.



- *Utilizar zoom actual:* seleccione esta opción si quiere usar el factor de zoom actual del proyecto de modelado. Si selecciona esta opción, se habilita el grupo de opciones *División de imágenes en páginas.*
- *Impedir:* seleccione esta opción para que los elementos de modelado no se dividan en dos páginas y se mantengan como una unidad.
- *Utilizar zoom óptimo:* elija esta opción para ajustar el tamaño del proyecto de modelado al tamaño de la página. También puede especificar el factor de zoom numéricamente.

Imprimir todos los diagramas

Este comando abre el cuadro de diálogo "Imprimir" e imprime todos los diagramas UML que contiene el archivo de proyecto actual.

Vista previa de impresión
Este comando abre el cuadro de diálogo "Imprimir".

Configurar impresión
Este comando abre el cuadro de diálogo "Configurar impresión", donde puede definir la impresora que desea usar y seleccionar la configuración del papel.

© 2014 Altova GmbH

17.2 Menú Edición

Deshacer
UModel permite eliminar todos los cambios realizados y devolver el archivo a versiones anteriores. Todos los cambios se pueden deshacer uno por uno y no hay un límite de operaciones deshacer.

Rehacer
Permite rehacer las acciones que deshizo con el comando **Deshacer**. Esto significa que puede ir adelante y atrás en el historial de acciones con los comandos **Deshacer** y **Rehacer**.

Cortar/Copiar/Eliminar
Estos comandos de edición estándar de Windows sirven para cortar, copiar y eliminar elementos de modelado. Para más información consulte el apartado Cortar, copiar y pegar en los diagramas de UModel.

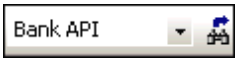
Pegar
Este comando (menú **Edición**, menú contextual o **Ctrl+V**) siempre añade un elemento de modelado nuevo al diagrama y a la Estructura del modelo. Para más información consulte el apartado Cortar, copiar y pegar en los diagramas de UModel.

Pegar sólo en el diagrama
Si usa este comando desde el menú contextual (tras hacer clic con el botón secundario en el fondo del diagrama), se añade un vínculo/una vista del elemento existente al diagrama actual y no a la Estructura del modelo. Para más información consulte el apartado Cortar, copiar y pegar en los diagramas de UModel.

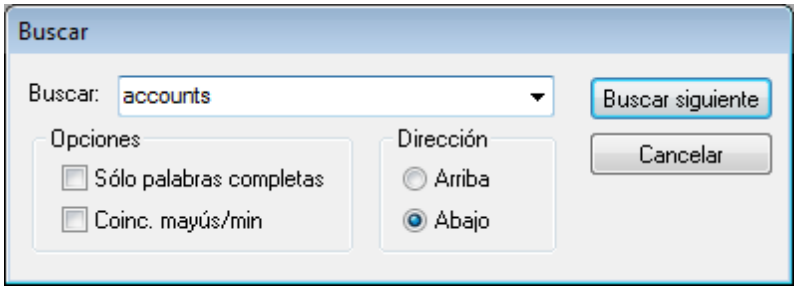
Eliminar sólo en el diagrama
Este comando elimina los elementos seleccionados del diagrama activo. Sin embargo, los elementos no se eliminan del proyecto de modelado y siguen estando disponibles en la Estructura del modelo. Recuerde que este comando no sirve para eliminar propiedades ni operaciones de clase. Las propiedades y operaciones se pueden seleccionar y eliminar en la clase directamente.

Seleccionar todo
Este comando selecciona todos los elementos de modelado del diagrama activo. Equivale a utilizar **Ctrl+A**.

Buscar
En UModel puede buscar elementos de modelado de varias formas:

- Desde el cuadro de texto de la **barra de título** principal 

- Con el comando de menú **Edición | Buscar**.
- Con la tecla de acceso rápido **Ctrl+F** (que abre el cuadro de diálogo "Buscar").



En el cuadro de diálogo "Buscar" puede buscar texto:

- En los paneles Estructura del modelo, Árbol de diagramas y Favoritos.
- En el panel Documentación.
- En cualquier diagrama activo.
- En el panel Mensajes.

Buscar siguiente **(F3)**
Este comando repite la última búsqueda realizada con el comando **Buscar** y busca la siguiente instancia del término de búsqueda en el diagrama o en la pestaña activos.

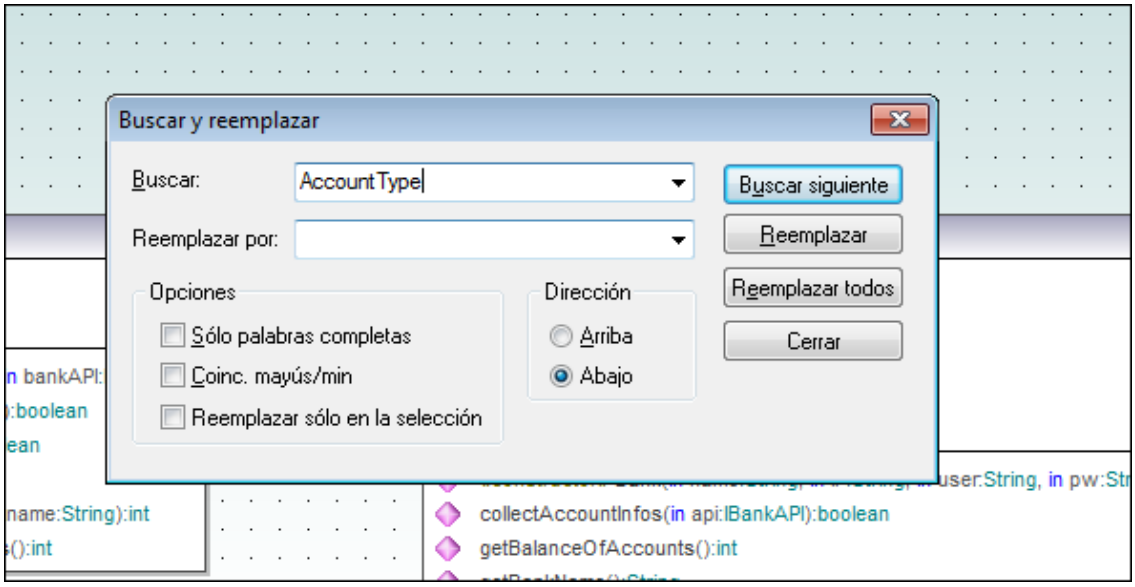
Buscar anterior (Mayús+F3)
Este comando repite la última búsqueda realizada con el comando **Buscar** y busca la instancia anterior del término de búsqueda en el diagrama o en la pestaña activos.

Reemplazar
Este comando busca y reemplaza elementos de modelado en el proyecto. Una vez encontrado, el elemento se resalta en el diagrama y en la Estructura del modelo.

Puede usar la función de búsqueda y reemplazo:

- En todos los diagramas.
- En los paneles Estructura del modelo, Árbol de diagramas y Favoritos.
- En el panel Documentación.

 © 2014 Altova GmbH



Copiar como mapa de bits
Este comando copia el diagrama activo en el portapapeles. Después podrá pegar el diagrama en cualquier aplicación.

Nota: los diagramas se copian en el portapapeles del sistema. Para verlos o acceder a ellos debe insertarlos en otra aplicación.

Copiar la selección como mapa de bits
Este comando copia los elementos de diagrama **seleccionados** en el portapapeles. Después podrá pegarlos en cualquier aplicación.

17.3 Menú Proyecto

Revisar la sintaxis del proyecto...
Este comando sirve para revisar sintaxis del proyecto de UModel. Las tablas que aparecen a continuación detallan qué aspectos del archivo de proyecto se revisan:

	Se comprueba si...	Mensaje...
A nivel de proyecto	existe una raíz de espacio de nombres Java como mínimo	Error
A nivel de componente	se estableció el archivo / directorio de proyecto	Error
	existe Realización	Error
	está desactivada la casilla *Usar para ingeniería de código*. Si es así, el componente no se revisa y se deshabilita la revisión de sintaxis en el componente.	Ninguno

Nivel	Se comprueba si...	Mensaje...
`Clase`	se estableció el nombre de archivo.	**Error** si la opción local *Generar los nombres de archivo de código que falten* no está activada. **Advertencia** si esa opción está activada.
	la clase está anidada (si lo está, no se realiza la revisión de sintaxis).	
	está en un espacio de nombres de lenguaje de código	Error
	se estableció el tipo de parámetro para las operaciones	Error
	se estableció el tipo para las propiedades	Error
	se estableció el tipo devuelto de las operaciones	Error
	hay operaciones duplicadas (nombres y tipos de parámetro)	Error
	alguna clase participa en Realización (solo si la clase no está anidada)	Advertencia
`Interfaz`	se estableció el nombre de archivo.	**Error** si no está activada la opción local *Generar los nombres de archivo de código que falten*. **Advertencia** si esa opción está activada.
	la interfaz está en un espacio de nombres de lenguaje de código	Error
	se estableció el tipo para las propiedades	Error
	se estableció el tipo de parámetro para las operaciones	Error

 © 2014 Altova GmbH

	se estableció el tipo devuelto de las operaciones	Error
	hay operaciones duplicadas (nombres y tipos de parámetro)	Error
	alguna interfaz participa en una RealizaciónDeComponente	Advertencia
Enumeración	pertenece a la raíz de espacio de nombres Java.	Advertencia (no se generará código)
	no pertenece a la raíz de espacio de nombres Java. Entonces se deshabilita la revisión de sintaxis en la enumeración. El paquete no se revisa.	Ninguno

Revisión de sintaxis en los elementos UML que intervienen en la generación de código

clase	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
propiedad de clase	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
operación de clase	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave). Se busca si existe un parámetro de devolución	Error
parámetro de operación de clase	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave). Se comprueba si el tipo tiene un nombre de tipo Java válido	Error
interfaz	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
operación de interfaz	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
parámetro de operación de interfaz	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
propiedades de la interfaz	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
paquete con espacio de nombres de estereotipo	se comprueba si el nombre es un nombre Java válido (no contiene caracteres prohibidos y el nombre no es una palabra clave)	Error
paquete sin espacio de nombres de estereotipo	no hay nada que se pueda revisar	Ninguno
clase	herencia múltiple	Error

Nota: la revisión sintáctica no revisa las restricciones de los elementos de modelado porque no intervienen en el proceso de generación de código Java. Para más información consulte el apartado Crear restricciones en los elementos de modelado.

Control de código fuente
UModel es compatible con Microsoft SourceSafe y otros sistemas de control de versiones.

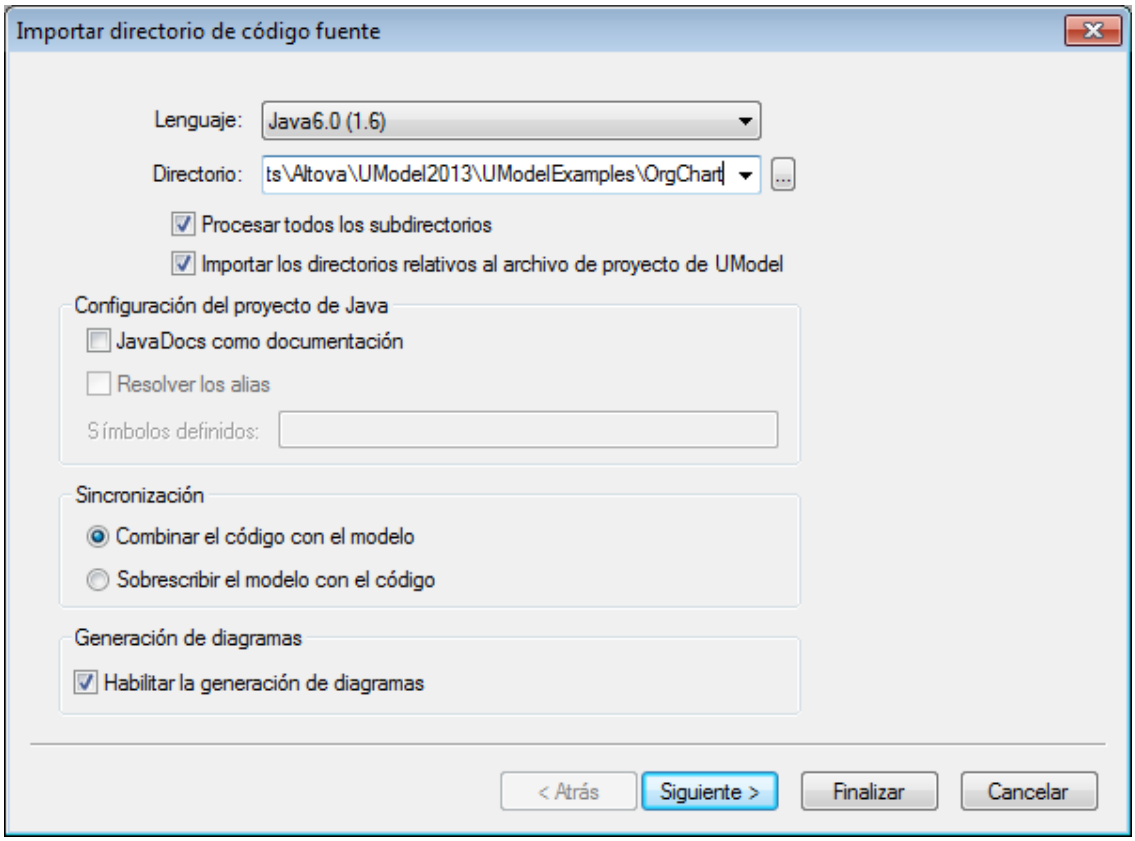
Microsoft define una entrada del registro, donde todos los programas compatibles con sistemas SCC (de control de versiones) se pueden registrar. UModel solamente lee esta entrada.

HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider \InstalledSCCProviders

Recuerde también que los complementos de control de versiones no se instalan automáticamente con todos los sistemas SCC. Para más información consulte la documentación de su software de control de versiones. También puede consultar el apartado Control de código fuente para obtener más información sobre servidores y clientes de control de código fuente y cómo utilizarlos.

 © 2014 Altova GmbH

Importar directorio de código fuente...
Este comando abre el cuadro de diálogo "Importar directorio de código fuente" (*imagen siguiente*). Para ver un ejemplo concreto consulte el apartado Ingeniería de ida y vuelta (código - modelo - código).

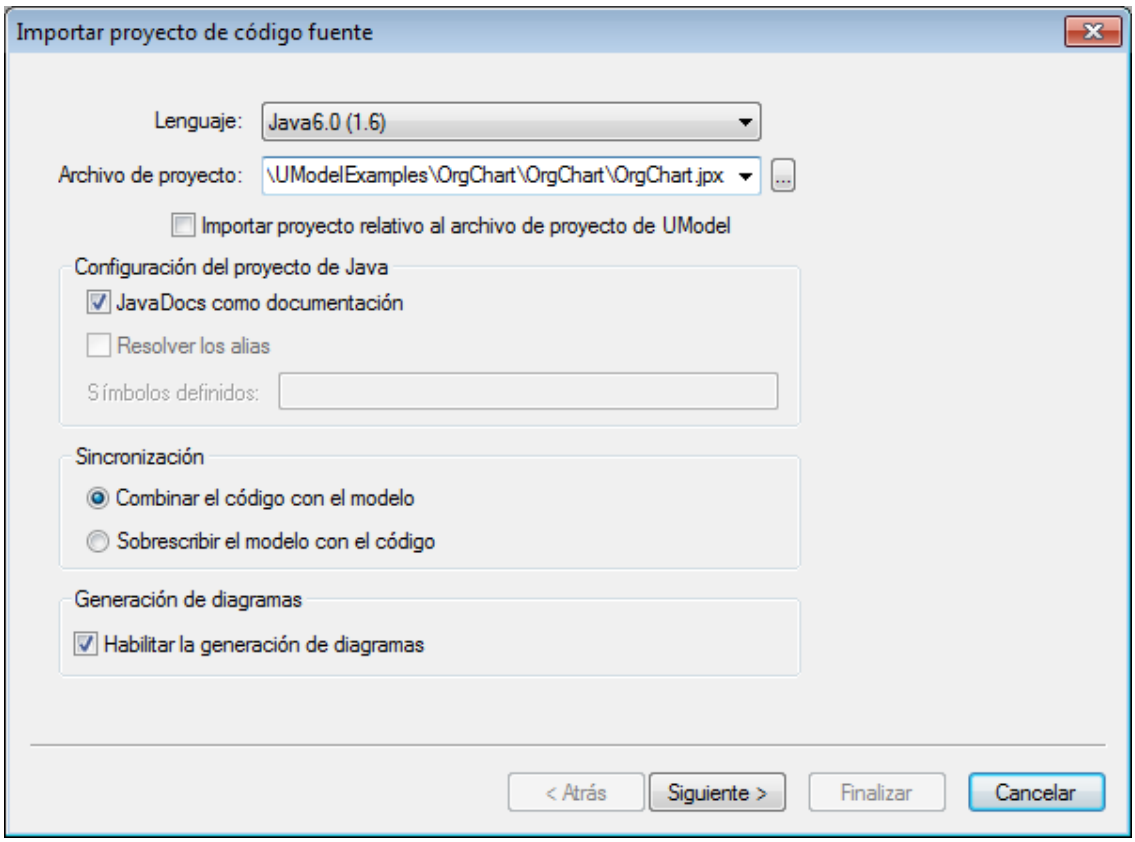


Importar proyecto de código fuente...
Este comando abre el cuadro de diálogo "Importar proyecto de código fuente" (*imagen siguiente*).

Haga clic en el botón **Examinar** para seleccionar el archivo de proyecto y el tipo de proyecto. Para ver un ejemplo concreto consulte el apartado Importar código fuente a los proyectos.

Proyectos Java:

- UModel admite proyectos de JBuilder **.jpx**, Eclipse **.project** y NetBeans (p. ej.: proyecto**.xml**).



Proyectos C#:

- Proyectos de MS Visual Studio (**csproj**, **csdprj**...)
- Proyectos de Borland **.bdsproj**

Proyectos de VB.NET:

- Proyectos de MS Visual Studio (**vbproj**, **vbdproj**.)

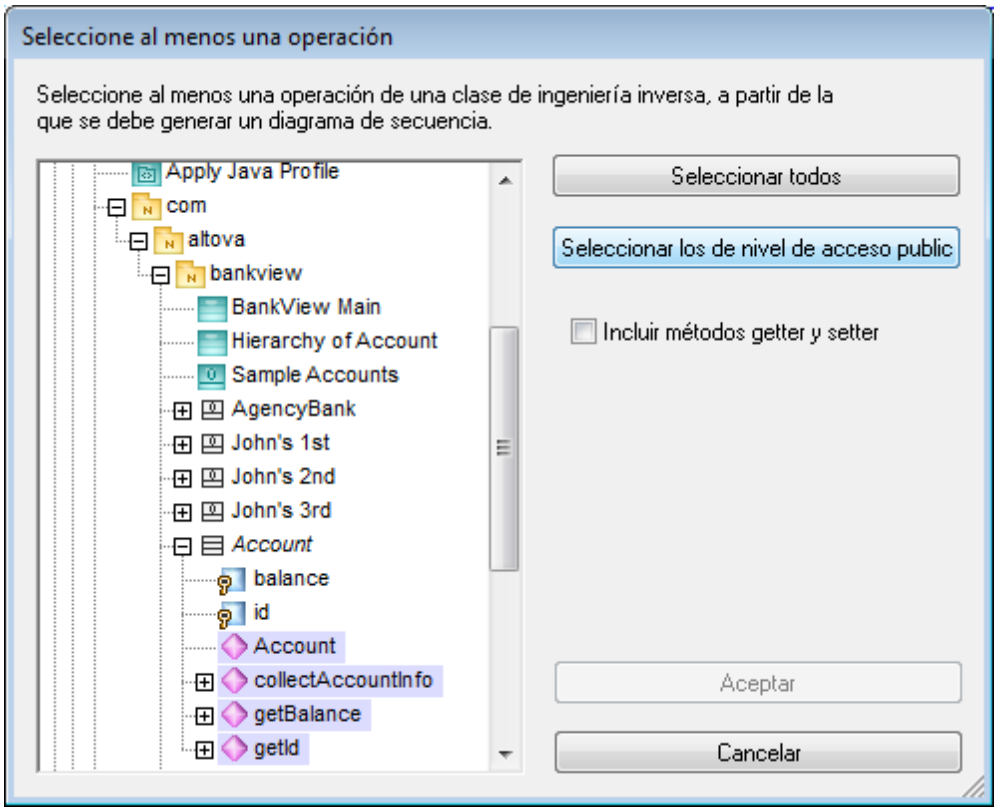
Importar tipos binarios
Abre el cuadro de diálogo "Importar tipos binarios", que sirve para importar archivos Java, C# y VB binarios. Para más información consulte el apartado Importar binarios Java, C# y VB.

Importar directorio del esquema XML
Abre el cuadro de diálogo "Importar el directorio del esquema XML", que sirve para importar todos los esquemas XML del directorio seleccionado y también los de sus subdirectorios.

 © 2014 Altova GmbH

Importar archivo de esquema XML
Abre el cuadro de diálogo "Importar archivo de esquema XML", que sirve para importar archivos de esquema. Consulte el apartado Diagramas de esquema XML para obtener más información.

Generar diagramas de secuencia a partir del código
Abre el cuadro de diálogo "Seleccione al menos una operación" (*imagen siguiente*), donde puede seleccionar las operaciones que deben utilizarse como base del diagrama de secuencia generado.



Tras seleccionar las operaciones, haga clic **Aceptar**. Se abre el cuadro de diálogo "Generación de diagrama de secuencia", donde puede configurar la generación.

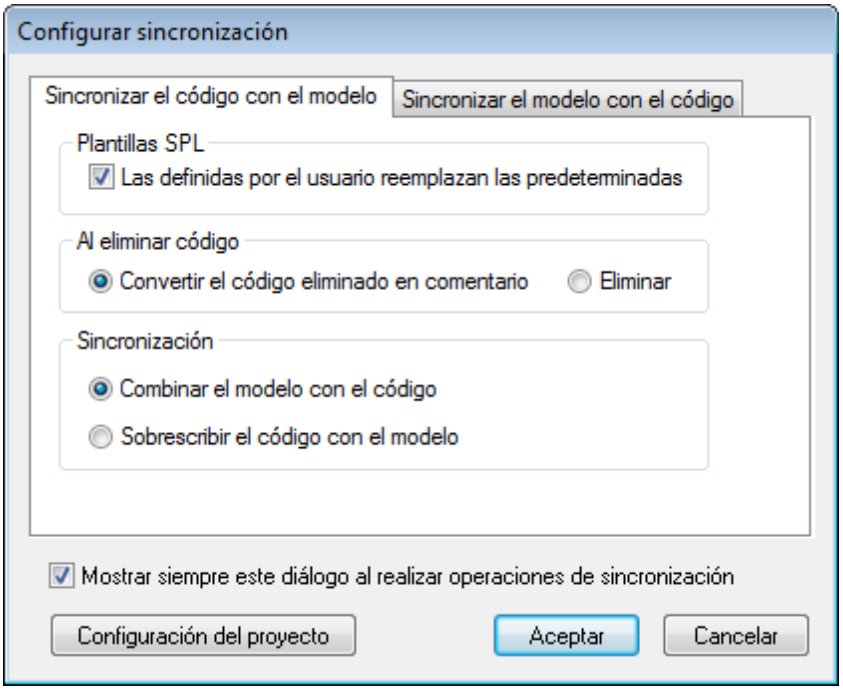
Combinar el código de programa con el proyecto de UModel
Abre el cuadro de diálogo "Configurar sincronización" por la pestaña *Sincronizar el código con el modelo*. Haga clic en el botón **Configuración del proyecto** para seleccionar opciones de configuración para el lenguaje de programación elegido.

Combinar o sobrescribir código
Supongamos que el código se generó a partir de un modelo y que desde entonces se realizaron cambios tanto en el modelo como en el código. Por ejemplo, supongamos que se realizaron estos cambios:

- En UModel se añadieron elementos de modelado nuevos (p. e. se añadió una clase nueva llamada `X`).
- Al código externo se añadió una clase nueva llamada `Y`.

Combinar (el modelo con el código) significa que:
- la clase nueva Y que se añadió en el código externo **se conserva**.
- la clase nueva X que se añadió en UModel **se añade al código**.

Sobrescribir (el código con el modelo) significa que:
- la clase nueva Y que se añadió en el código externo **se elimina**.
- la clase nueva X que se añadió en UModel **se añade al código**.



Combinar el proyecto de UModel con el código de programa
Abre el cuadro de diálogo "Configurar sincronización" por la pestaña *Sincronizar el modelo con el código*. Haga clic en el botón **Configuración del proyecto** para seleccionar opciones de configuración para el lenguaje de programación elegido.

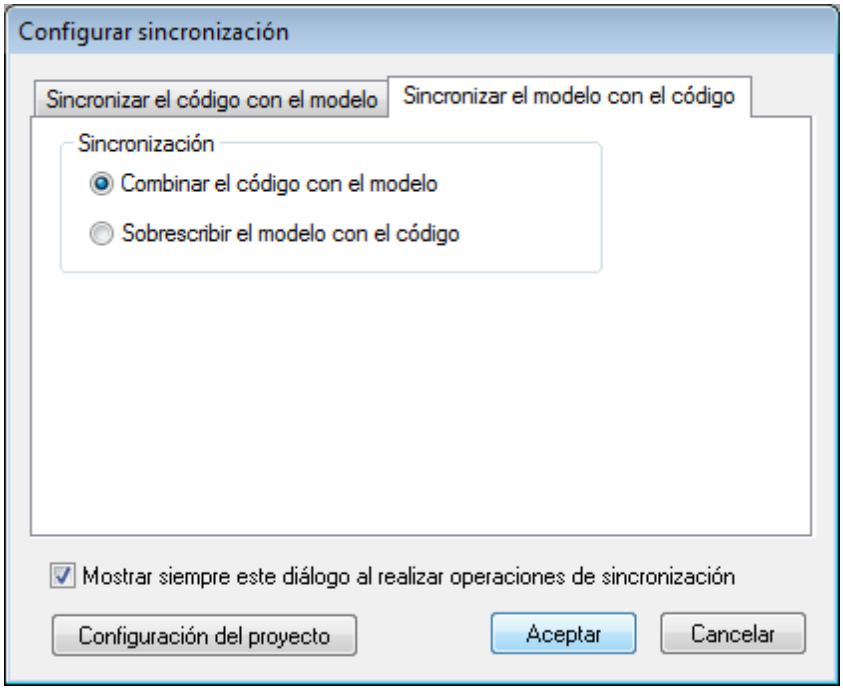
Combinar o sobrescribir el código
Supongamos que el código se generó a partir de un modelo y que desde entonces se realizaron cambios tanto en el modelo como en el código. Por ejemplo, supongamos que se realizaron estos cambios:

- En UModel se añadieron elementos de modelado nuevos (p. e. se añadió una clase nueva llamada `X`).
- Al código externo se añadió una clase nueva llamada `Y`.

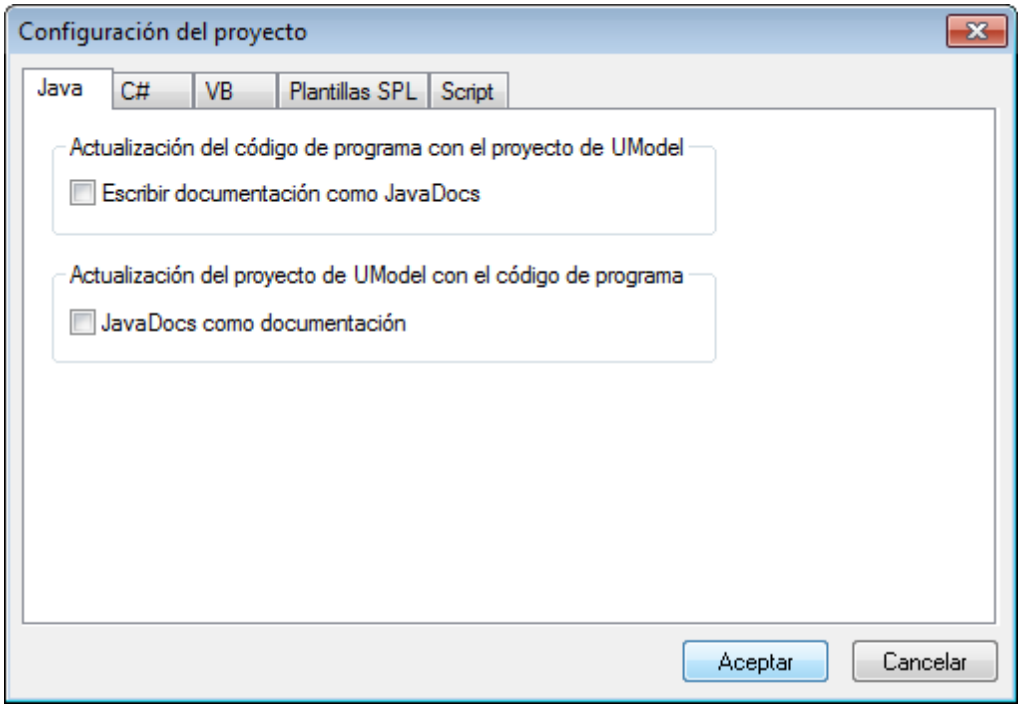
Combinar (el código con el modelo) significa que:
- la clase nueva X que se añadió en UModel **se conserva**.
- la clase nueva Y que se añadió en el código externo **se añade al modelo**.

Sobrescribir (el modelo con el código) significa que:
- la clase nueva X que se añadió en UModel **se elimina**.
- la clase nueva Y que se añadió en el código externo **se añade al modelo**.

 © 2014 Altova GmbH



Configuración del proyecto
Este comando sirve para definir opciones de configuración para cada lenguaje de programación.



Configurar sincronización...
Abre el cuadro de diálogo "Configurar sincronización" (*ver imágenes más arriba*).

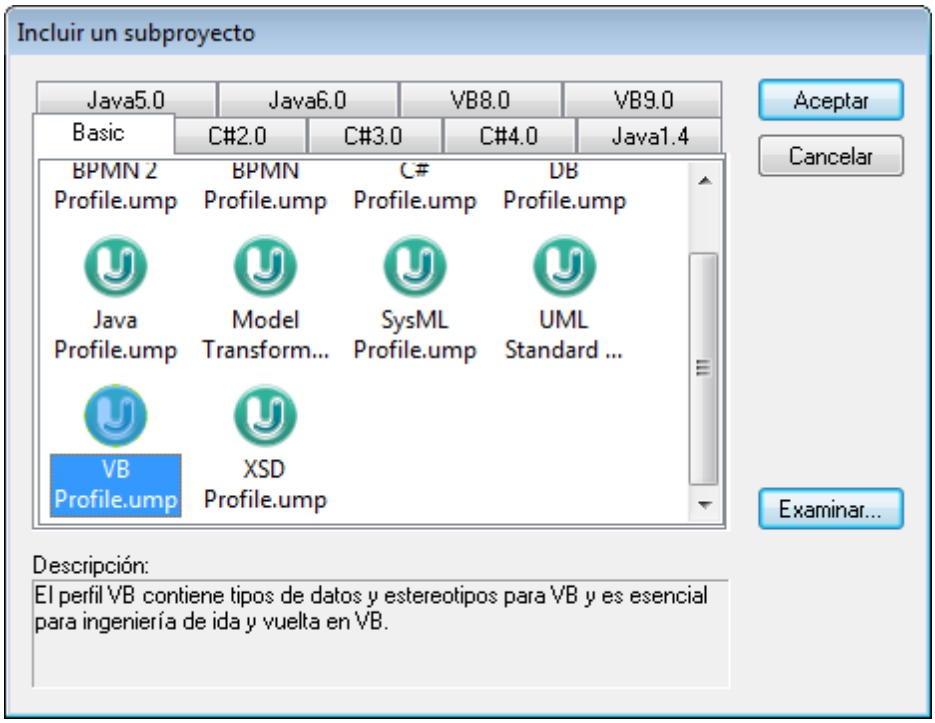
Combinar el proyecto...
Combina dos proyectos de UModel en uno solo modelo. El primer archivo que se abre es con el que se combina el segundo archivo. Consulte el apartado Combinar proyectos de UModel para obtener más información.

Incluir un subproyecto...
Puede incluir varios subproyectos en un proyecto de UModel. En el cuadro de diálogo "Incluir un subproyecto" haga clic en la pestaña *Java* para incluir clases, interfaces y paquetes del lenguaje Java en el proyecto.

1. Seleccione el comando **Proyecto | Incluir un subproyecto** para abrir el cuadro de diálogo "Incluir un subproyecto".
2. Haga clic en el archivo de proyecto de UModel que desea incluir y haga clic en **Aceptar**.

Los proyectos de UModel se pueden incluir dentro de otros proyectos de UModel. Para incluir proyectos debe poner los archivos `*.ump` correspondientes en estas carpetas:

- ...\UModel2015**UModelInclude** (para que aparezcan en la pestaña *Basic*) o
- ...\UModel2015**UModelInclude\Java** (para que aparezcan en la pestaña *Java*).



Nota: en la pestaña *C# 2.0* encontrará un archivo de inclusión, que contiene todos los tipos de Microsoft .NET Framework 2.0.

Para crear una carpeta/pestaña definida por el usuario:

1. Haga clic en el botón **Examinar**, y navegue hasta la carpeta `...\UModel2015\UModelInclude` y añada una carpeta nueva (p. ej. `...\UModelInclude\miCarpeta=)`.

Para crear una descripción para cada archivo de proyecto de UModel:

1. Cree un archivo de texto en la misma carpeta que el archivo `*.ump` y con el mismo

nombre.
P. ej. para el archivo `MiModelo.ump` debería crear el archivo `MiModelo.txt`. Asegúrese de usar la codificación UTF-8 para este archivo de texto.

Para quitar un proyecto ya incluido:

1. Haga clic en el paquete incluido en el panel Estructura del modelo y pulse la tecla **Supr**.
2. La aplicación le pregunta si desea continuar con la eliminación.
3. Haga clic en **Aceptar** para eliminar el archivo incluido del proyecto.

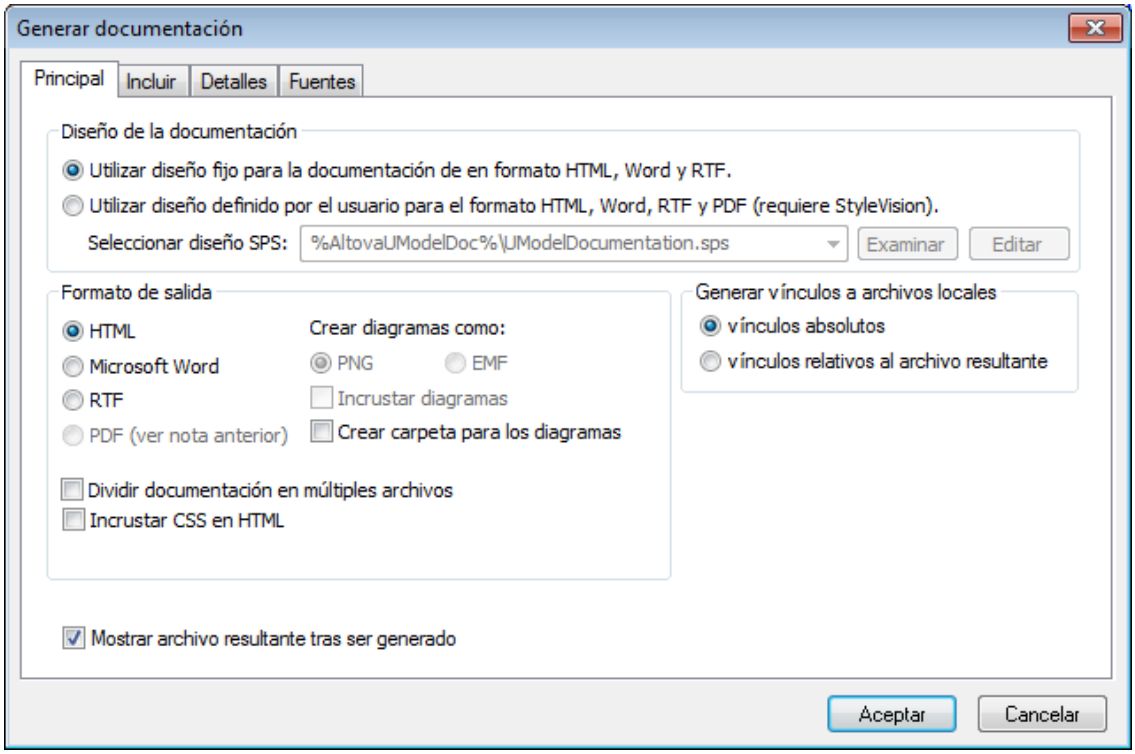
Nota: para eliminar o quitar un proyecto del cuadro de diálogo "Incluir un subproyecto" debe eliminar el proyecto `*.ump` en la correspondiente carpeta.

Abrir en forma de proyecto
Este comando abre el subproyecto seleccionado como un proyecto nuevo.

Borrar mensajes
Este comando borra los mensajes, errores y advertencias de revisión de sintaxis y de combinación de código de la ventana Mensajes.

Nota: los errores informan de problemas que deben solucionarse inmediatamente para poder generar código o para poder actualizar el código del modelo. Las advertencias, sin embargo, se pueden dejar para más tarde. En UModel los errores y advertencias los genera la función de revisión de sintaxis, el compilador de cada lenguaje de programación, el analizador de UModel que lee los archivos fuente generados y la función de importación XMI.

Generar documentación
Este comando sirve para generar documentación para el archivo activo en formato HTML, Word y RTF. Para más información consulte el apartado Generar documentación UML.



Mostrar elementos no utilizados en ningún diagrama
Este comando crea una lista con todos los elementos que no se utilizan en ningún diagrama del proyecto.

Mostrar paquetes compartidos
Este comando crea una lista con todos los paquetes compartidos del proyecto actual.

Mostrar paquetes incluidos
Este comando crea una lista con todos lo paquetes incluidos en el proyecto actual. Los paquetes `Java Profile.ump` y `Java Lang.ump` están incluidos automáticamente en el proyecto de muestra `Bankview` que viene con UModel.

 © 2014 Altova GmbH

17.4 Menú Diseño

Los comandos del menú **Diseño** sirven para alinear y poner en fila los elementos de los diagramas de modelado.

Cuando utilice el recuadro de selección (arrastrando el puntero del mouse mientras pulsa el botón principal del mouse) para seleccionar elementos, el elemento con una línea de puntos como contorno es el elemento activo (es decir, el último elemento seleccionado por el recuadro). Los comandos de alineación toman este elemento como referencia/base.

Alinear
Este grupo de comandos sirve para alinear los elementos de modelado con el borde elegido o en el centro, dependiendo de la opción elegida.

Espaciar uniformemente
Este grupo de comandos sirve para espaciar los elementos seleccionados uniformemente, en horizontal o en vertical.

Igualar tamaño
Este grupo de comandos sirve para ajustar el ancho y el alto de los elementos seleccionados al ancho/alto del elemento activo.

Poner en fila
Este grupo de comandos sirve para poner en fila vertical u horizontal los elementos seleccionados.

Estilo de la línea
Este grupo de comandos permite elegir el tipo de línea que se utiliza para conectar los diferentes elementos de modelado. Las líneas pueden ser líneas de dependencia o de asociación.

Tamaño automático
Este comando ajusta el tamaño de los elementos seleccionados hasta alcanzar un tamaño óptimo.

Aplicar diseño automático a todo
Este comando sirve para elegir el tipo de presentación para los elementos de modelado del diagrama UML activo:

- **Diseño dirigido por fuerzas**
  Organiza los elementos de forma céntrica.
- **Diseño jerárquico**
  Organiza los elementos en función de las relaciones que existen entre ellos.
- **Diseño por bloques**
  Organiza los elementos según su tamaño y formando un rectángulo.

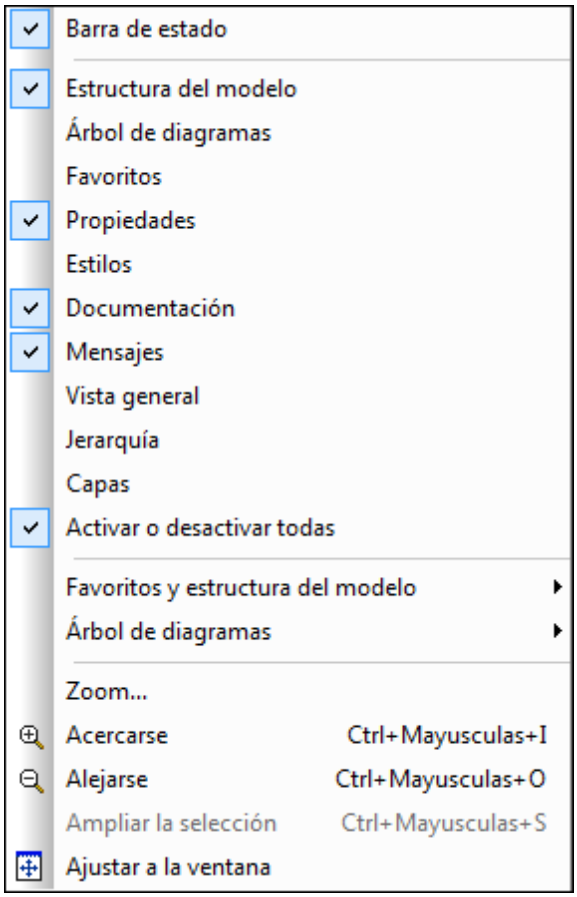
Ajustar la posición de las etiquetas de texto
Este comando pone el nombre de los elementos (de todos o de los seleccionados) en su posición predeterminada.

© 2014 Altova GmbH

17.5 Menú Vista

Los comandos de este menú sirven para:

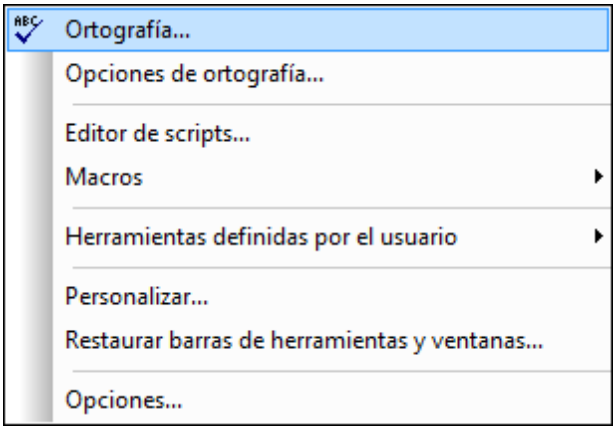
- Activar las pestañas de los diferentes paneles.
- Definir el criterio de ordenación de los elementos de modelado en las pestañas *Estructura del modelo* y *Favoritos*.
- Definir el criterio de agrupación de los diagramas en la pestaña *Árbol de diagramas*.
- Mostrar/ocultar determinados elementos UML en las pestañas *Estructura del modelo* y *Favoritos*.
- Definir el factor de zoom del diagrama actual.



17.6 Menú Herramientas

Los comandos del menú **Herramientas** sirven para:

- Revisar la ortografía del proyecto de UModel y configurar la revisión ortográfica.
- Abrir el entorno de scripting de UModel. En el entorno de scripting puede crear, gestionar y almacenar formularios, macros y controladores de eventos.
- Ver y ejecutar las macros que ya están definidas.
- Personalizar la aplicación definiendo barras de herramientas, teclas de acceso rápido, menús y macros personales.
- Restaurar las barras de herramientas y ventanas de la aplicación a su estado predeterminado de instalación.
- Definir opciones de configuración globales para la aplicación.

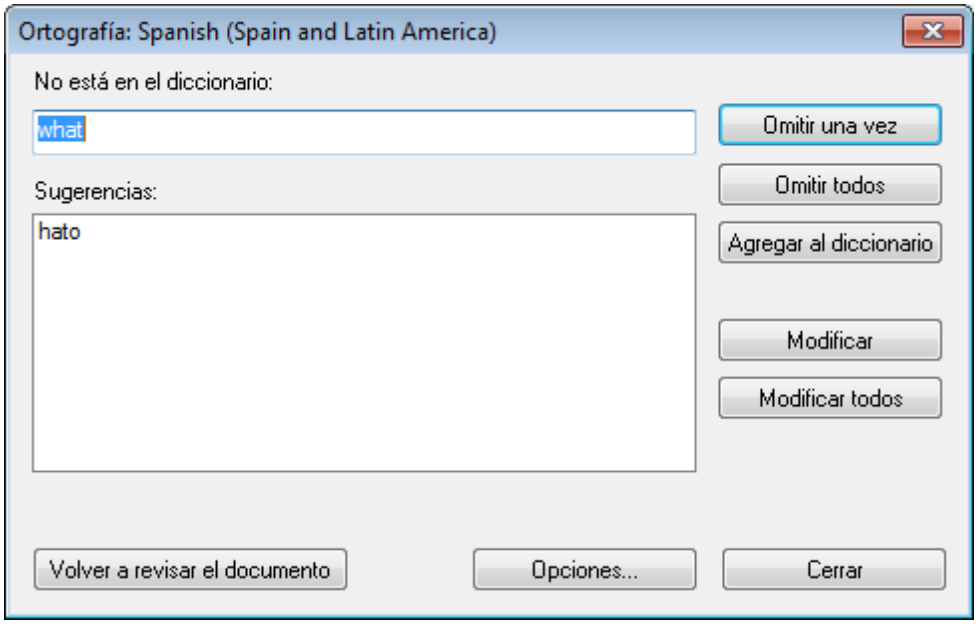


 © 2014 Altova GmbH

17.6.1 Ortografía

Este comando del menú **Herramientas** sirve para revisar la ortografía del proyecto. Al hacer clic en él, aparece el cuadro de diálogo "Ortografía", que incluye opciones de revisión ortográfica estándar. Para configurar opciones más específicas, haga clic en el botón **Opciones...** o seleccione el comando **Herramientas | Opciones de ortografía**.

Nota: puede revisar la ortografía de las entradas de la Estructura del modelo y de los diagramas UML. Si hace clic con el botón secundario en la Estructura del modelo y selecciona **Ortografía de la documentación** en el menú contextual, se revisa la ortografía de los comentarios y las notas de la Estructura del modelo.



No está en el diccionario:
Este cuadro de texto contiene la palabra que no se encontró ni en el diccionario del idioma seleccionado ni en el diccionario del usuario.

Sugerencias
Este cuadro de lista muestra palabras similares a la palabra desconocida. Haga doble clic en una palabra de esta lista para insertarla automáticamente en el documento y continuar la revisión ortográfica.

Omitir una vez
Pulse este botón para continuar revisando la ortografía del documento y pasar por alto la primera instancia de la palabra desconocida. Si esa palabra vuelve aparecer, la revisión ortográfica la señalará.

Omitir todos
Pulse este botón para pasar por alto todas las instancias de la palabra desconocida en todo el documento.

Agregar al diccionario
Pulse este botón para añadir la palabra desconocida al diccionario del usuario. Puede acceder al diccionario del usuario desde el cuadro de diálogo "Opciones".

Modificar
Pulse este botón para reemplazar la palabra que está resaltada en el documento con la palabra editada del cuadro *No está en el diccionario*.

Modificar todos
Pulse este botón para reemplazar todas las instancias de la palabra que está resaltada con la palabra editada del cuadro *No está en el diccionario*.

Volver a revisar el documento
Pulse este botón para reiniciar la revisión ortográfica desde el principio del documento.

© 2014 Altova GmbH

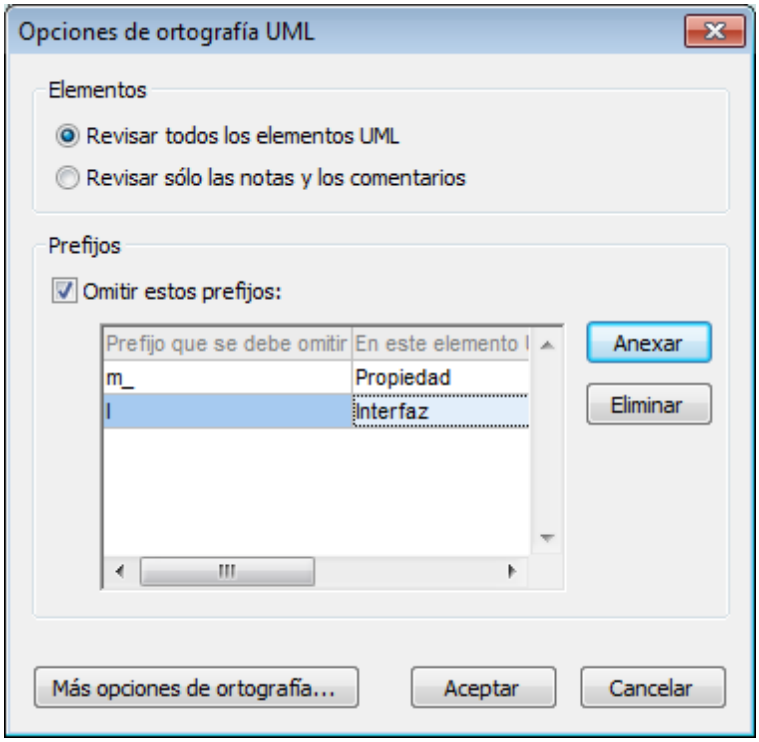
17.6.2 Opciones de ortografía...

Este comando del menú **Herramientas** abre el cuadro de diálogo "Opciones de ortografía UML" (*imagen siguiente*). Este cuadro de diálogo sirve para configurar la revisión ortográfica de elementos UML y sus opciones se describen a continuación.

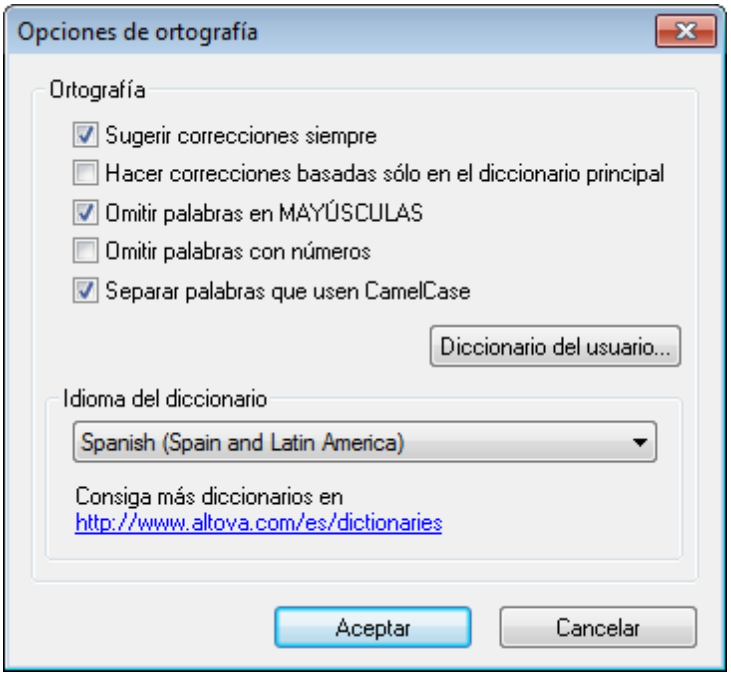
Elementos
En este grupo de opciones puede elegir si se revisan todos los elementos UML o solamente las notas y los comentarios.

Prefijos
Haga doble clic en la columna *Prefijo que se debe omitir* para insertar los prefijos de los elementos UML que desea pasar por alto durante la revisión ortográfica (p. ej. `m_` para las propiedades y `I_` para las interfaces).

El botón **Anexar** añade una fila nueva a la tabla de prefijos. El botón **Eliminar** elimina la fila seleccionada de la tabla.



Más opciones de ortografía...
Este botón abre el cuadro de diálogo "Opciones de ortografía" (*imagen siguiente*), que sirve para modificar la configuración global del corrector ortográfico. Sus opciones se describen más abajo.



Sugerir correcciones siempre
Si marca esta casilla, el cuadro *Sugerencias* siempre mostrará opciones del diccionario integrado elegido y del diccionario del usuario. Si desactiva esta opción, el corrector no ofrecerá ninguna sugerencia.

Hacer correcciones basadas sólo en el diccionario principal
Si marca esta casilla, solamente se usan sugerencias del diccionario integrado elegido (diccionario principal). El diccionario del usuario no se utilizará para ofrecer sugerencias. Si marca esta casilla, se deshabilita el botón **Diccionario del usuario**. Es decir, el diccionario del usuario no se puede editar mientras esté activa esta opción.

Omitir palabras en MAYÚSCULAS
Si marca esta casilla, el corrector ortográfico pasa por alto las palabras que están en mayúsculas.

Omitir palabras con números
Si marca esta casilla, el corrector ortográfico pasa por alto las palabras que contienen números.

Separar palabras que usen CamelCase
Las palabras que usan CamelCase están formadas por palabras que empiezan por mayúsculas, unidas unas a otras sin espacios. Por ejemplo, la palabra *CamelCase* está formada por las palabras *Camel* y *Case*, ambas en mayúsculas y unidas sin espacios. Este tipo de palabras no suele aparecer en los diccionarios y, por tanto, el corrector ortográfico las marcará como erróneas. Marque la casilla *Separar palabras que usen CamelCase* para que el corrector revise por separado cada una de las palabras que forman la palabra CamelCase. Esta opción está marcada por defecto.

Idioma del diccionario
Use este cuadro combinado para seleccionar el idioma del diccionario integrado del corrector ortográfico. La opción predeterminada es "US English" (inglés EE UU). Para descargar gratis más diccionarios en otros idiomas, visite el sitio web de Altova.

 © 2014 Altova GmbH

¿Cómo agregar diccionarios para el corrector ortográfico?
Por cada idioma hay dos archivos de diccionario Hunspell que funcionan conjuntamente: un archivo `.aff` y un archivo `.dic`. Los diccionarios se instalan en la carpeta `Lexicons` de este directorio:

Windows 7 y Windows 8:	`C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons`
Windows XP:	`C:\Documents and Settings\All Users\Datos de programa\Altova\SharedBetweenVersions\SpellChecker\Lexicons`

Dentro de la carpeta `Lexicons` se crea una carpeta por idioma: `<nombre del idioma>\<archivos del diccionario>`. Por ejemplo, en Windows 7 y Windows 8 los archivos de los dos diccionarios de español (de España y de Hispanoamérica y España) se almacenan así:

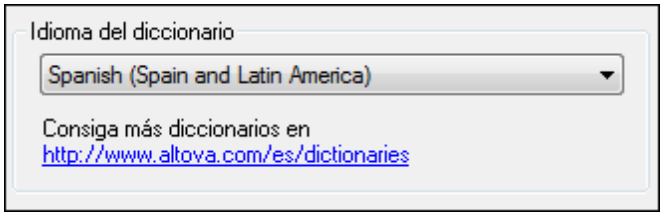
```
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\Spanish (Spain and Latin America)\es_ANY.aff
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\Spanish (Spain and Latin America)\es_ANY.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\Spanish (Spain)\es_ES.dic
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\Spanish (Spain)\es_ES.dic
```

La lista desplegable *Idioma del diccionario* del cuadro de diálogo "Opciones de ortografía" muestra los diccionarios disponibles. Se trata de los diccionarios de la carpeta `Lexicons` y reciben el nombre de las subcarpetas de la carpeta `Lexicons`. Por ejemplo, los dos diccionarios de inglés del ejemplo anterior aparecerían así en el cuadro combinado: *Spanish (Spain)* y *Spanish (Spain and Latin America)*.

Todos los diccionarios instalados son compartidos por los diferentes usuarios del equipo y por las diferentes versiones de los productos de Altova (tanto en 64 como en 32 bits).

Hay dos maneras de agregar diccionarios nuevos para el corrector ortográfico. En ninguno de los dos casos es necesario registrar los archivos en el sistema:

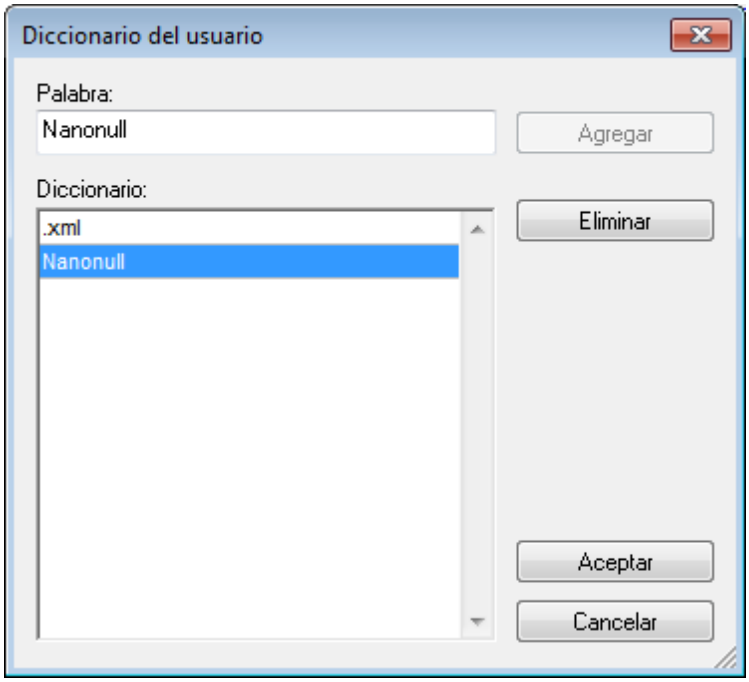
- Puede añadir diccionarios Hunspell a una subcarpeta nueva de la carpeta `Lexicons`. Los diccionarios Hunspell se pueden descargar desde http://wiki.services.openoffice.org/wiki/Dictionaries o desde http://extensions.services.openoffice.org/en/dictionaries, por ejemplo. (Recuerde que OpenOffice utiliza el formato comprimido `OXT`. Cambie la extensión a `.zip` y descomprima los archivos `.aff` y `.dic` en las subcarpetas correspondientes de la carpeta `Lexicons`. También puede usar diccionarios Myspell, ya que los diccionarios Hunspell están basados en Myspell.)
- Puede usar el instalador de diccionarios de Altova, que instala un paquete con varios diccionarios en el directorio adecuado del equipo. En el cuadro de diálogo "Opciones de ortografía", bajo el panel *Idioma del diccionario*, aparece un enlace a la página de Altova de descarga de diccionarios (*imagen siguiente*). Si no usa derechos de administrador para instalar los diccionarios, se producirá un error de instalación.



Nota: recuerde que es decisión suya aceptar o no las condiciones de uso de la licencia del diccionario elegido. También es responsabilidad suya comprobar si el diccionario puede utilizarse en su equipo o no.

Trabajar con el diccionario del usuario

Cada usuario tiene un diccionario del usuario propio, donde se almacenan las palabras aprobadas por el usuario. Durante la revisión ortográfica, el corrector compara la ortografía con una lista compuesta por palabras del diccionario integrado y del diccionario del usuario. Puede añadir o eliminar palabras del diccionario del usuario en el cuadro de diálogo "Diccionario del usuario" (*imagen siguiente*) Para abrir este cuadro de diálogo pulse el botón **Diccionario del usuario** del cuadro de diálogo "Opciones de ortografía" (*segunda imagen de este apartado*).



Para añadir una palabra al diccionario del usuario escriba la palabra en el recuadro *Palabra* y pulse el botón **Agregar**. La palabra se añade a la lista alfabética del panel *Diccionario.* Para eliminar una palabra del diccionario, seleccione la palabra del panel *Diccionario* y pulse el botón **Eliminar**. La palabra se elimina del panel *Diccionario*. Cuando termine de editar el cuadro de diálogo "Diccionario del usuario" haga clic en el botón **Aceptar**. Los cambios se guardan en el diccionario del usuario.

También puede añadir palabras al diccionario del usuario durante la revisión ortográfica. Si el corrector encuentra una palabra desconocida, aparece el cuadro de diálogo Ortografía. Pulse el botón **Agregar al diccionario** para añadir la palabra desconocida al diccionario del usuario.

 © 2014 Altova GmbH

El diccionario del usuario se encuentra en este directorio:

Windows 7 y Windows 8: `C:\Users\<usuario>\Mis Documentos\Altova\SpellChecker\Lexicons\user.dic`
Windows XP: `C:\Documents and Settings\<usuario>\Mis Documentos\Altova\SpellChecker\Lexicons\user.dic`

17.6.3 Editor de scripts...

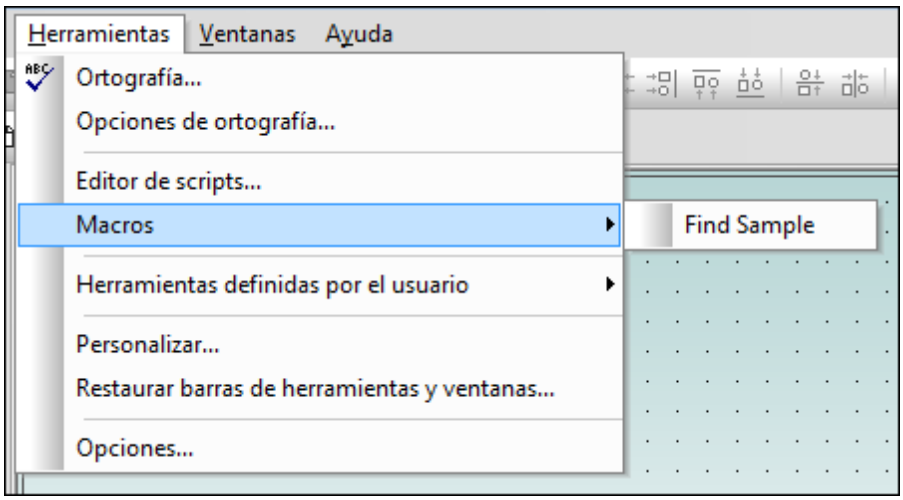
Este comando del menú **Herramientas** abre la ventana del editor de script. Para más información sobre cómo trabajar con el editor de script, consulte la sección Editor de scripts de la presente documentación.

Nota: es necesario tener instalado .NET Framework versión 2.0 (o superior) para poder ejecutar el editor de scripts.

© 2014 Altova GmbH

17.6.4 Macros

Al hacer clic en el comando **Macros** del menú **Herramientas** aparece un submenú (*imagen siguiente*) con todas las macros definidas en el proyecto de script activo. El proyecto de script activo se define en la pestaña *Script* del cuadro de diálogo "Opciones locales".



17.6.5 Herramientas definidas por el usuario

Al hacer clic en el comando **Herramientas definidas por el usuario** aparece un submenú con comandos hechos a medida que usan aplicaciones externas. Para crear estos comandos, use la pestaña *Herramientas* del cuadro de diálogo "Personalizar". Al hacer clic en uno de estos comandos personalizados, se ejecuta la acción asociada al comando.

© 2014 Altova GmbH

17.6.6 Personalizar

El comando **Herramientas | Personalizar** abre el cuadro de diálogo "Personalizar", que sirve para personalizar varios aspectos de la interfaz gráfica de UModel.

El cuadro de diálogo "Personalizar" está compuesto por estas pestañas:

Comandos
Barras de herramientas
Herramientas
Teclado
Menú
Macros
Complementos
Opciones

Comandos

En la pestaña *Comandos* puede personalizar sus menús y barras de herramientas y añadir comandos a los menús y a las barras de herramientas, dependiendo de lo que necesite. No obstante, tenga en cuenta que no puede crear comandos ni menús nuevos para la aplicación.

Para añadir un comando a una barra de herramientas o menú:

1. Seleccione el comando **Herramientas | Personalizar**. Se abre el cuadro de diálogo "Personalizar".
2. Seleccione la pestaña ***Comandos***. En el cuadro de lista *Categorías* seleccione la opción **Todos los comandos**. Todos los comandos disponibles aparecen en el cuadro de lista *Comandos*.
3. Haga clic en un comando del cuadro de lista *Comandos* y arrástrelo a un menú o barra de herramientas ya existente. Al pasar el puntero por encima de una posición donde se puede colocar el comando aparece el icono **I**.
4. Cuando encuentre la posición donde desea colocar el comando, suelte el botón del mouse.

Tenga en cuenta que:

- Mientras arrastra el comando, aparece un pequeño botón al final del puntero del mouse. Esto indica que el comando está siendo arrastrado.
- Si el comando no se puede colocar en la posición actual del cursor, debajo del puntero aparece una **X**.
- Si el cursor está en una posición donde se puede colocar el comando (en una barra de herramientas o en un menú), la **X** desaparece y el icono **I** indica que la posición es válida.
- Los comandos se pueden colocar en menús o barras de herramientas. Si creó una barra de herramientas nueva, puede usar este mecanismo de personalización para rellenar la barra de herramientas con comandos.
- Si pasa el cursor por un menú que está cerrado, el menú se abre y puede insertar el comando en cualquier parte del menú.

Para agregar comandos a menús contextuales

También puede añadir comandos a menús contextuales arrastrando comandos del cuadro de lista *Comandos* hasta el menú contextual:

1. Haga clic en la pestaña *Menú* del cuadro de diálogo "Personalizar".
2. En el cuadro combinado del panel *Menús contextuales* seleccione un menú contextual. El menú contextual seleccionado aparece en pantalla.
3. Vuelva a la pestaña *Comandos* del cuadro de diálogo "Personalizar".
4. Seleccione un comando en el cuadro de lista *Comandos* y arrástrelo hasta la posición deseada del menú contextual.

Para eliminar un comando o menú

Por último, puede eliminar un comando de un menú, menú contextual (ver párrafo anterior) o barra de herramientas o eliminar un menú entero:

1. Abra el cuadro de diálogo "Personalizar" (**Herramientas | Personalizar**).
2. Seleccione cualquier pestaña del cuadro de diálogo "Personalizar". Haga clic con el botón secundario en un menú o comando de menú y seleccione **Eliminar** en el menú contextual que aparece. Si lo prefiere, también puede arrastrar el menú o comando de menú hasta que aparezca el icono **X** debajo del puntero del mouse y suelte el menú o comando de menú. Como resultado se elimina el menú o comando de menú.

Para volver a instalar los comandos de menú eliminados, utilice los mecanismos descritos en este apartado. Para restablecer un menú eliminado, seleccione **Herramientas | Personalizar | Menú** y pulse el botón **Restaurar** del panel *Menús del marco de la aplicación*. Otra opción es seleccionar **Herramientas | Personalizar | Barras de herramientas**, hacer clic en la barra de herramientas pertinente y pulsar el botón **Restaurar**.

Barras de herramientas

En la pestaña *Barras de herramientas* puede: (i) activar o desactivar barras de herramientas (es decir, decidir qué barras de herramientas aparecen en la interfaz), (ii) definir qué iconos aparecen en cada barra de herramientas y (iii) crear barras de herramientas personalizadas.

Las barras de herramientas incluyen iconos para los comandos de menú más utilizados. Además, al pasar el puntero sobre un icono, se ofrece información rápida sobre el icono en un mensaje emergente y en la barra de estado de la aplicación. Las barras de herramientas se pueden colocar en cualquier posición de la pantalla, donde aparece como ventana flotante.

Para activar/desactivar una barra de herramientas:

Marque la casilla de la barra de herramientas en el cuadro de lista *Barras de herramientas*.

Para aplicar los cambios a todas las vistas:

Marque la casilla situada al final de la pestaña. De lo contrario, los cambios realizados afectan solamente a la vista activa.
Recuerde que los cambios realizados **después** de marcar la casilla *Aplicar cambios en todas las vistas* afectarán a todas las vistas.

Para añadir una barra de herramientas nueva:

1. Pulse el botón **Nueva...** y escriba el nombre de la barra de herramientas nuevas en el cuadro de diálogo "Nombre de la barra de herramientas" que aparece.
2. Arrastre comandos desde la pestaña *Comandos* hasta la barra de herramientas nueva.

Para cambiar el nombre de una barra de herramientas nueva:

1. Seleccione la barra de herramientas en el panel *Barra de herramientas* y pulse el botón **Cambiar de nombre**.

 © 2014 Altova GmbH

2. Edite el nombre en el cuadro de diálogo "Nombre de la barra de herramientas" que aparece.

Para restaurar la barra de menús:
Seleccione `Barra de menús` en el panel *Barras de herramientas* y pulse el botón **Restaurar**.
La barra de menús vuelve a su estado original de instalación.

Para restaurar todas las barras de herramientas y comandos de menú:
Pulse el botón **Restaurar todo**.
Todas las barras de herramientas y menús vuelven a su estado original de instalación.

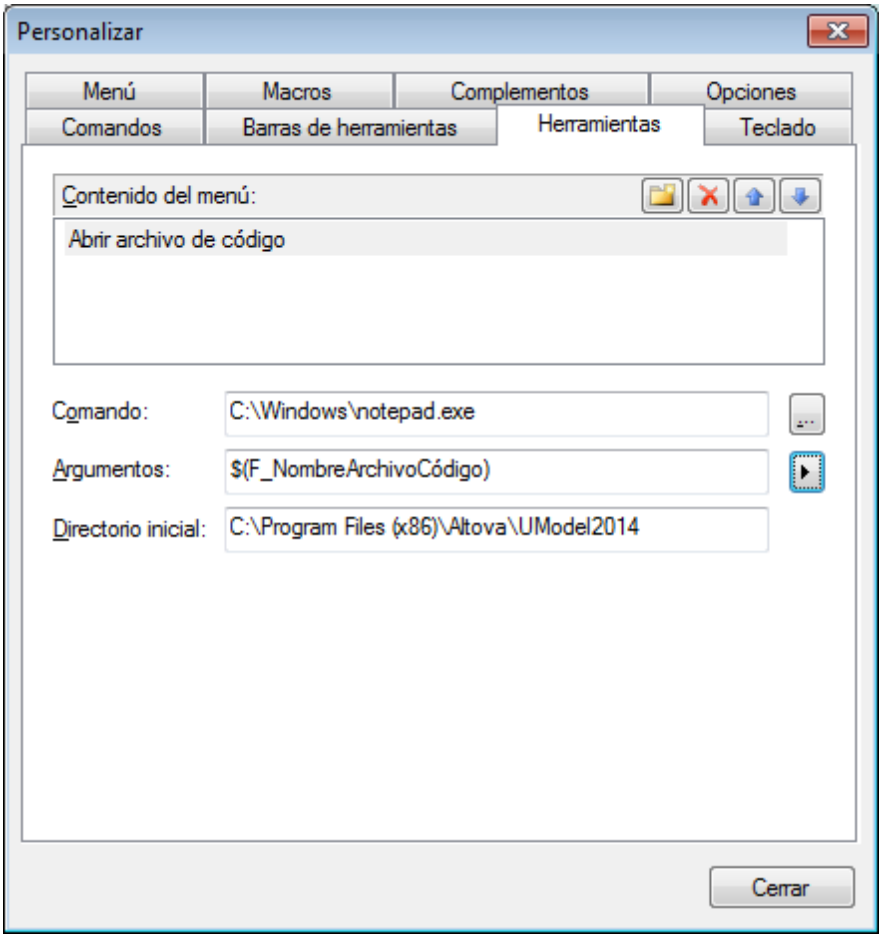
Para eliminar una barra de herramientas:
Seleccione la barra de herramientas en el panel *Barras de herramientas* y pulse el botón **Eliminar**.

Para mostrar las etiquetas de texto de una barra de herramientas:
Seleccione la barra de herramientas y marque la casilla *Mostrar etiquetas de texto*.
Recuerde que debe activar las etiquetas de texto de cada barra de herramientas por separado.

Herramientas

En la pestaña *Herramientas* puede crear comandos para poder usar aplicaciones externas desde UModel. Estos comandos se añaden al menú **Herramientas | Herramientas definidas por el usuario**.

Haga clic en el icono en forma de carpeta (de la barra de iconos *Contenido del menú)* para añadir una entrada nueva y use el campo *Comando* para asociarlo a una aplicación.

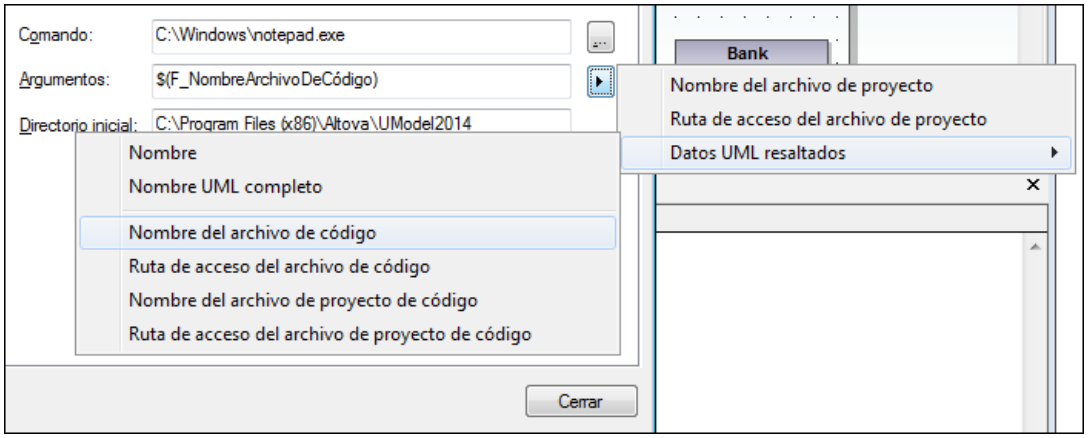


La pestaña *Herramientas* también permite definir argumentos. Los argumentos son variables a las que se asignan valores concretos cuando se inicia una herramienta externa desde el comando de menú.

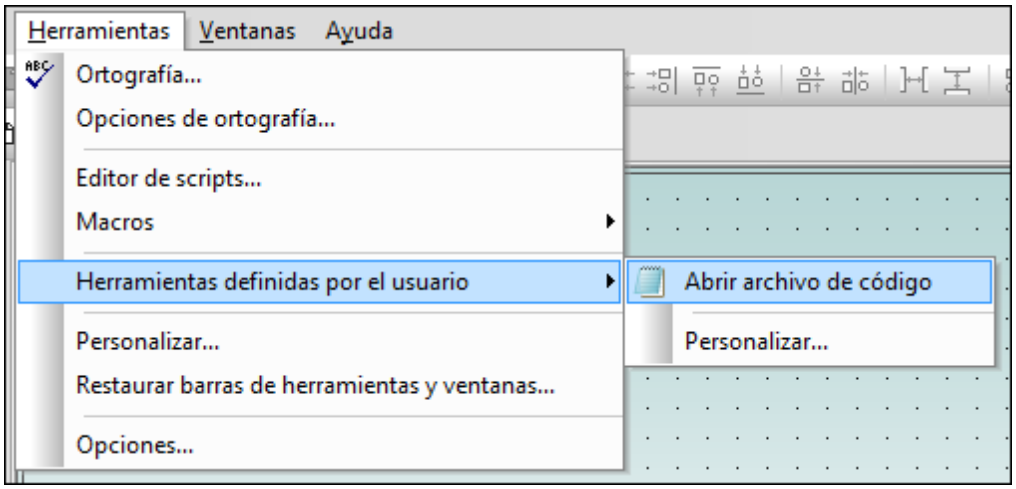
P. ej. supongamos que queremos abrir en Notepad el archivo de código fuente de la clase UML seleccionada.

1. Haga clic en **Herramientas | Personalizar** y abra la pestaña *Herramientas* del cuadro de diálogo "Personalizar".
2. Escriba el nombre y la ruta de acceso de la aplicación externa (p. ej. `C:\...\notepad.exe`).
3. Haga clic en el botón flotante del campo *Argumentos* y seleccione el argumento que desea utilizar (p. ej. *Nombre del archivo de código*).

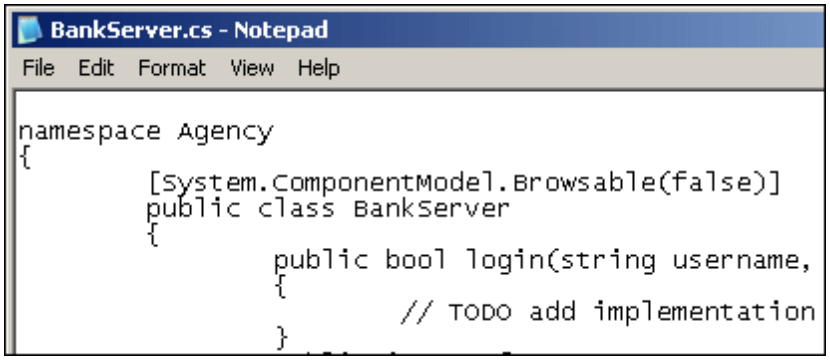
 © 2014 Altova GmbH



4. Haga clic en el botón **Cerrar** para finalizar.
5. Haga clic en el menú **Herramientas | Herramientas definidas por el usuario** y seleccione **Abrir archivo de código**.



Se abre el archivo `BankServer.cs` en Notepad.



Argumentos de UModel

Nombre del archivo de proyecto

El nombre de archivo del archivo de proyecto de UModel activo (p. ej. `Prueba.ump`).

Ruta de acceso del archivo de proyecto

La ruta de archivo absoluta del archivo de proyecto de UModel activo (p. ej. `c:\MiDirectorio\Prueba.ump`).

Datos UML resaltados – Nombre
El nombre del elemento UML resaltado (p. ej. `Clase1`).

Datos UML resaltados – Nombre UML completo
El nombre completo del elemento UML resaltado (p. ej. `Paquete1::Paquete2::Clase1`).

Datos UML resaltados – Nombre del archivo de código
El nombre del archivo de código de la clase, interfaz o enumeración UML resaltada, tal y como aparece en la ventana Propiedades (p. ej. `Clase1.cs` o `MiEspacioDeNombres\Clase1.Java`).

Datos UML resaltados – Ruta de acceso del archivo de código
La ruta de acceso del archivo de código de la clase, interfaz o enumeración UML resaltada, tal y como aparece en la ventana Propiedades (p. ej. `C:\Temp\MiCódigo\Clase1.cs`).

Datos UML resaltados – Nombre del archivo de proyecto de código
El nombre de archivo del proyecto de código al que pertenece la clase, interfaz o enumeración UML resaltada.

El nombre de archivo del proyecto puede ser relativo al archivo de proyecto de UModel y es el mismo que aparece en la ventana Propiedades del componente (p. ej. `C:\Temp\MiCódigoFuente\MiProyecto.vcproj` o `MiCódigoFuente\MiProyecto.vcproj`)

Datos UML resaltados – Ruta de acceso del archivo de proyecto de código
La ruta de acceso de archivo del proyecto de código al que pertenece la clase, interfaz o enumeración UML resaltada (p. ej. `c:\Temp\MiCódigoFuente\MiProyecto.vcproj`").

Teclado

En la pestaña *Teclado* puede crear teclas de acceso rápido nuevas o cambiar las teclas de acceso rápido ya existentes para cualquier comando de la aplicación.

Para asignar una tecla de acceso rápido nueva a un comando o cambiar una tecla de acceso rápido ya existente:

1. En el cuadro combinado *Categoría* seleccione la opción *Todos los comandos*.
2. En el cuadro de lista *Comandos* seleccione el comando al que desea asignar una tecla de acceso rápido nueva o el comando cuya tecla de acceso rápido desea cambiar.
3. Haga clic dentro del cuadro *Pulsar tecla de acceso rápido nueva* y pulse la tecla de acceso rápido que desea asignar al comando. La tecla de acceso rápido aparece en el cuadro *Pulsar tecla de acceso rápido nueva.* Si la tecla de acceso rápido no se asignó todavía a ningún comando, se habilita el botón **Asignar**. Si la tecla ya se asignó a un comando, el comando aparece debajo del cuadro y el botón **Asignar** está deshabilitado. (Para borrar el contenido del cuadro *Pulsar tecla de acceso rápido nueva* pulse **Ctrl**, **Alt** o **Mayús**).
4. Haga clic en el botón **Asignar**. La tecla de acceso rápido aparece ahora en el cuadro de lista *Teclas actuales*. Puede asignar varias teclas de acceso rápido al mismo comando si lo desea.
5. Para confirmar los cambios pulse el botón **Cerrar**.

 © 2014 Altova GmbH

Para eliminar una tecla de acceso rápido:

1. En el cuadro de lista *Teclas actuales* seleccione la tecla de acceso rápido que desea eliminar.
2. Pulse el botón **Quitar**.
3. Para confirmar los cambios pulse el botón **Cerrar**.

Menú

En la pestaña *Menú* puede personalizar las dos barras de menú principales (la barra de menú predeterminada y la barra de menú de la aplicación) así como los menús contextuales de la aplicación.

Para personalizar la barra de menú predeterminada y la barra de menú de la aplicación:

La barra de menú predeterminada es la barra de menú que aparece cuando no hay ningún documento abierto en la ventana principal. La barra de menú de la aplicación es la barra que aparece cuando hay un documento abierto en la ventana principal. Cada una de estas barras de menú se puede personalizar y los cambios realizados en una de las barras de menú no afecta a la otra.

Para personalizar una barra de menú, selecciónela en el cuadro combinado *Mostrar menús para:* de la pestaña *Menú* (*imagen anterior*). Después cambie a la pestaña *Comandos* del cuadro de diálogo "Personalizar" y arrastre comandos desde el cuadro de lista *Comandos* hasta la barra de menú.

Para eliminar comandos de menús y restaurar las barras de menú:

Para eliminar un menú entero o un comando de un menú, seleccione el menú o el comando de menú y después (i) haga clic con el botón secundario y seleccione **Eliminar** o (ii) arrastre el comando fuera de la barra de menú o del menú.

Para restaurar estas dos barras de menú (la barra de menú predeterminada y la de la aplicación) a su estado original de instalación seleccione el menú en el cuadro combinado *Mostrar menús para:* y haga clic en el botón **Restaurar** situado bajo el cuadro combinado.

Para personalizar los menús contextuales de la aplicación:

Los menús contextuales son los menús que aparecen cuando se hace clic con el botón secundario en determinados objetos de la interfaz de la aplicación. Siga estos pasos para personalizar un menú contextual:

1. Seleccione el menú contextual en el cuadro combinado *Seleccionar menú contextual*. Aparece el menú contextual.
2. Pase a la pestaña *Comandos* del cuadro de diálogo Personalizar.
3. Arrastre un comando del cuadro de lista *Comandos* al menú contextual.
4. Si desea eliminar un comando del menú contextual, haga clic en él con el botón secundario y seleccione **Eliminar**. También puede seleccionar el comando y arrastrarlo fuera del menú contextual.

Para restaurar un menú contextual a su estado original de instalación seleccione el menú en el cuadro combinado *Seleccionar menú contextual* y después pulse el botón **Restaurar**, situado bajo el cuadro combinado.

Sombras de menú
Marque la casilla *Sombras de menú* para dar sombra a todos los menús.

Macros

En la pestaña *Macros* puede elegir los macros definidos en el proyecto de script que está activo en UModel.

El proyecto de script activo aparece en la pestaña *Script* del cuadro de diálogo "Opciones locales" o en la pestaña *Script* del cuadro de diálogo "Configuración del proyecto".

Complementos

En la pestaña *Complementos* puede seleccionar un archivo de complemento (`.dll`).

Para más información consulte el apartado Probar el complemento de UModel.

Opciones

En la pestaña *Opciones* puede definir la configuración general del entorno.

Marque las casillas para activar estas opciones:

- *Mostrar información en pantalla en las barras de herramientas:* al pasar el puntero del mouse sobre los iconos de las barras de herramientas aparece un mensaje emergente con una breve descripción de la función del icono, así como su tecla de acceso rápido (si le asignó una y si marcó la casilla *Mostrar teclas de acceso rápido*).
- *Mostrar teclas de acceso rápido en la información en pantalla:* la información en pantalla de los iconos de las barras de herramientas incluye la tecla de acceso rápido del comando.
- *Iconos grandes*: alterna el tamaño de los iconos de las barras de herramientas entre tamaño estándar y tamaño grande.

 © 2014 Altova GmbH

17.6.7 Restaurar barras de herramientas y ventanas

El comando **Restaurar barras de herramientas y ventanas** cierra UModel y lo reinicia con su configuración predeterminada. Antes de cerrarse, UModel le pregunta si desea cerrar o no la aplicación.

Este comando es muy práctico si movió ventanas o barras de herramientas de sitio, si las ocultó o si ajustó su tamaño y desea poner todas estas barras de herramientas y ventanas como estaban en un principio.

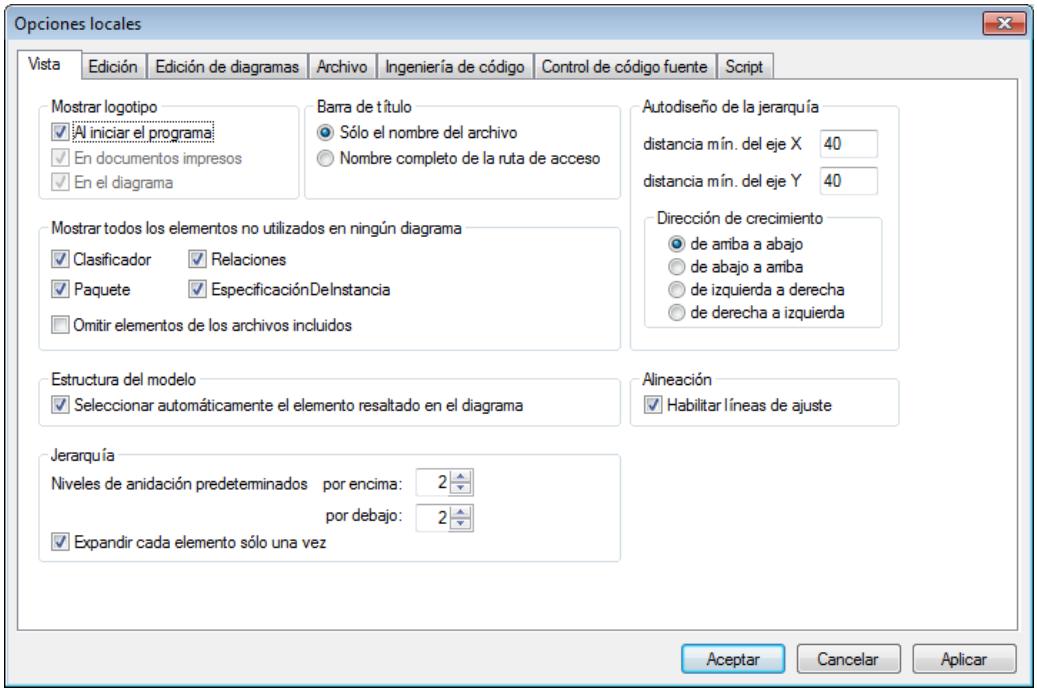
17.6.8 Opciones

El comando **Herramientas | Opciones** abre el cuadro de diálogo "Opciones locales".

Pestaña *Vista*

En esta pestaña puede definir:

- donde debe aparecer el logotipo del programa.
- el contenido de la barra de título de la aplicación.
- el tipo de elementos que deben aparecer en la lista generada por el comando *Mostrar elementos no utilizados en ningún diagrama* del menú contextual de los paneles Estructura del modelo y Favoritos. También tiene la opción de omitir los elementos de los archivos incluidos.
- si un elemento seleccionado de un diagrama se selecciona/sincroniza automáticamente o no en la Estructura del modelo.
- la profundidad predeterminada de la vista jerárquica generada por el comando **Mostrar en forma de diagrama** en el panel Jerarquía.
- la profundidad de los niveles del panel Jerarquía (con las opciones del grupo *Autodiseño de la jerarquía*).
- que se expanda solo uno de los clasificadores del mismo tipo de la misma imagen / del mismo diagrama (con la opción *Expandir cada elemento solo una vez*).
- si se habilitan las líneas de ajuste para ayudarle durante la alineación de elementos en el diagrama.



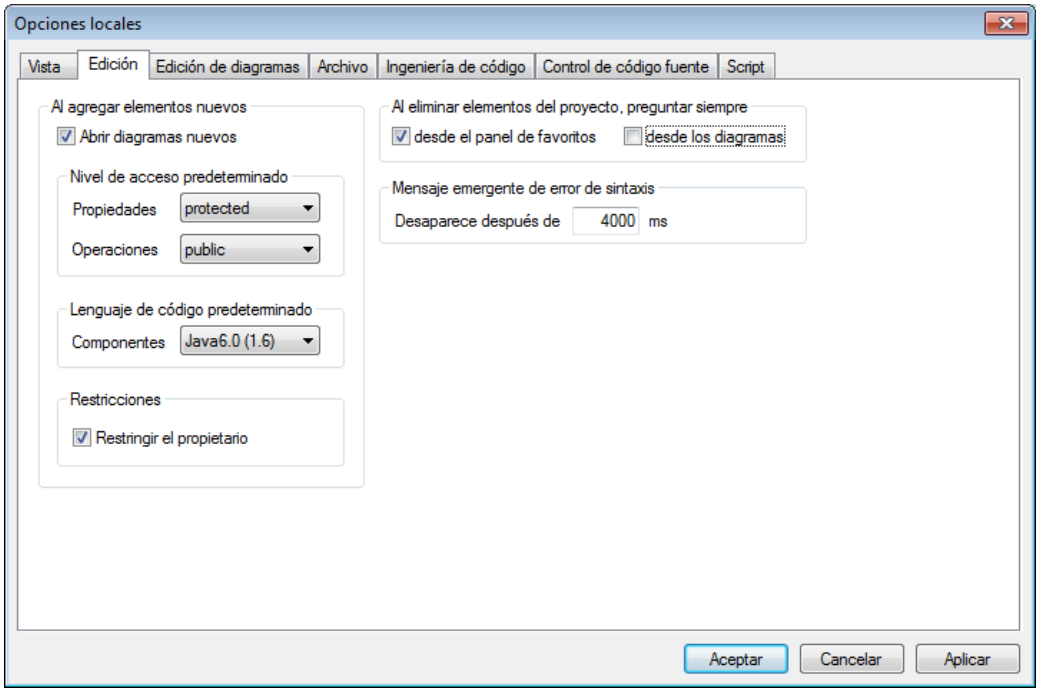
Pestaña *Edición*

En esta pestaña puede definir:

- que cuando se cree un diagrama nuevo en la Estructura del modelo, el diagrama se abra

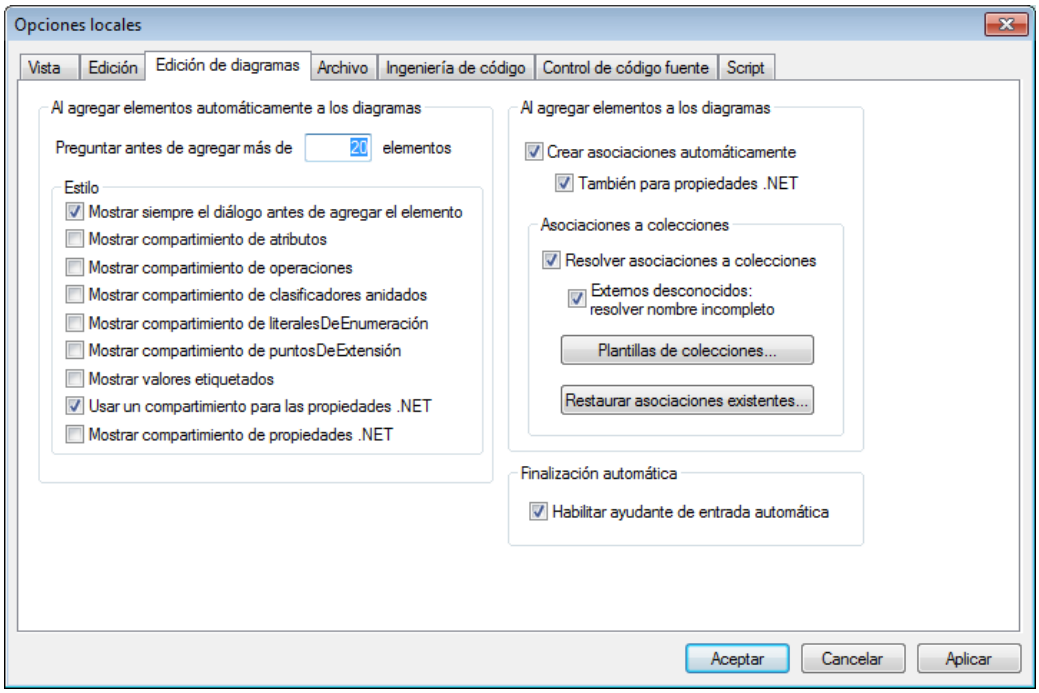
automáticamente en el área de trabajo.

- el nivel de acceso predeterminado de los elementos nuevos (propiedades y operaciones).
- el lenguaje de código predeterminado para los componentes nuevos.
- si las restricciones también deben restringir automáticamente su propietario.
- si debe aparecer un aviso cuando se **eliminen** elementos del proyecto, del panel Favoritos o de un diagrama. Este aviso se puede desactivar cuando se eliminen elementos.
- cuánto tardan en cerrarse los mensajes emergentes de error de sintaxis.



Pestaña *Edición de diagramas*:
En esta pestaña puede definir:

- cuántos elementos se pueden añadir automáticamente a un diagrama sin que aparezca un aviso.
- la presentación de los estilos cuando se añaden automáticamente a un diagrama.
- si se debe crear automáticamente asociaciones entre los elementos de modelado cuando se añaden elementos a un diagrama.
- si se deben resolver las asociaciones a colecciones.
- si las plantillas de elementos externos desconocidos se deben resolver como plantillas no completas.
- o si se deben usar plantillas de colecciones ya disponibles o definir plantillas nuevas. Debe definir las plantillas de colecciones como plantillas completas (es decir, `a.b.c.Lista`). Si la plantilla tiene este espacio de nombres, UModel crea una asociación de colecciones automáticamente. Excepción: si la plantilla pertenece al paquete **`Elementos externos desconocidos`** y está habilitada la opción Externos desconocidos: resolver nombre incompleto, entonces se tiene en cuenta el nombre de la plantilla solamente (es decir, `Lista` en lugar de `a.b.c.Lista`).
- si la ventana de finalización automática está disponible mientras se editan atributos u operaciones en los diagramas de clases.



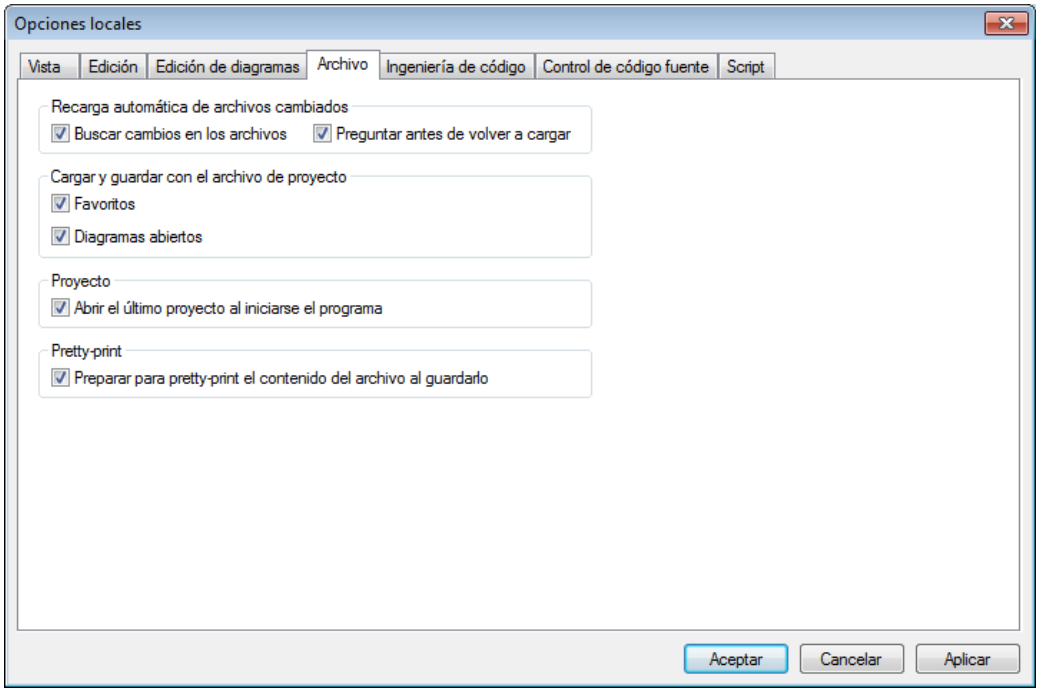
Pestaña *Archivo*:
En esta pestaña puede definir:

- qué ocurre cuando los archivos cambian.
- si el contenido del panel Favoritos y los diagramas abiertos deben cargarse y guardarse con el proyecto actual.
- si el último proyecto que se abrió se debe abrir automáticamente cuando se inicia la

 © 2014 Altova GmbH

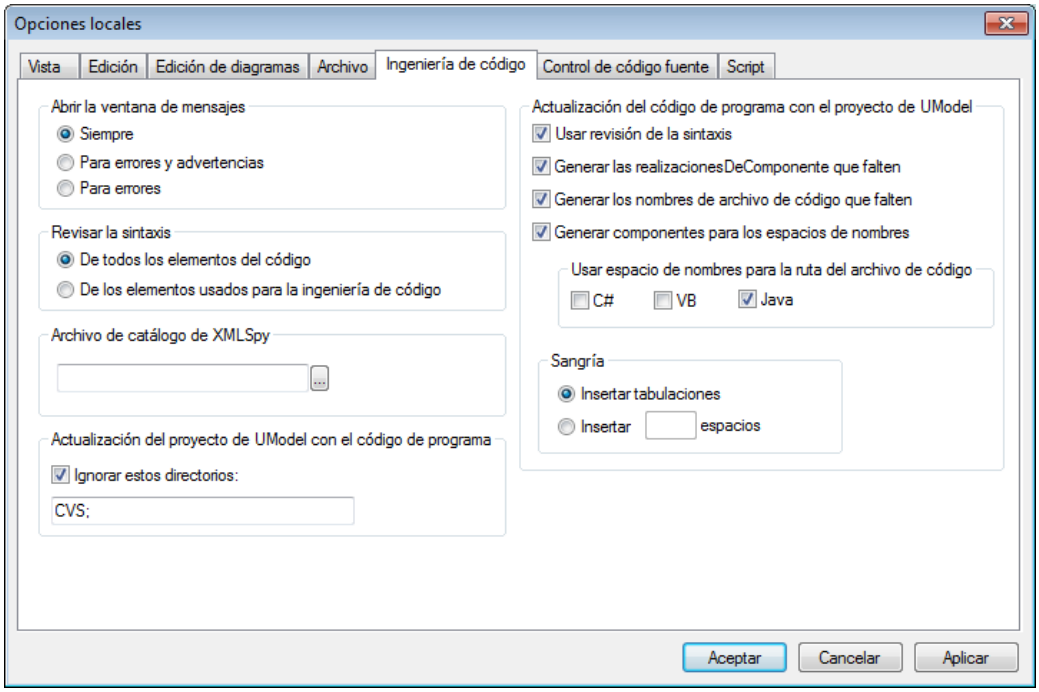
aplicación.

- si se añaden retornos de carro/saltos de línea y tabulaciones al archivo de proyecto para darle formato **pretty-print**.



Pestaña *Ingeniería de código*:
En esta pestaña puede definir:

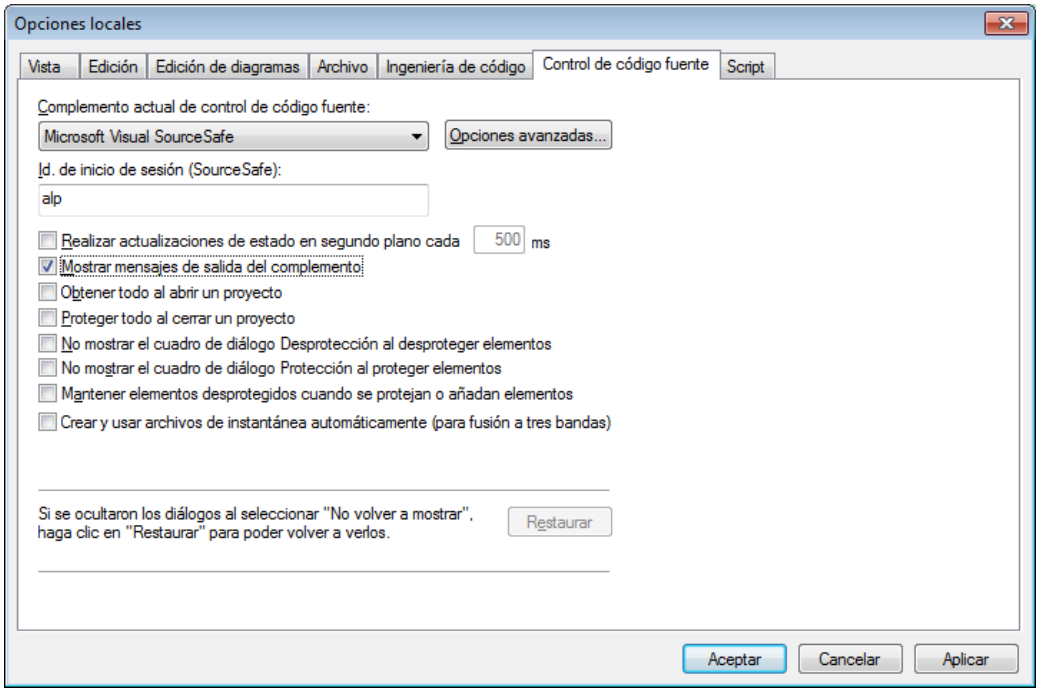
- en qué circunstancias se abre la ventana Mensajes.
- si se revisan **todos los elementos** de código (es decir, los que están en una raíz de espacio de nombres Java / C# / VB y los que están asignados a un componente Java / C# / VB) o **solo los elementos utilizados para ingeniería de código** (es decir, los que tienen activa la casilla `Usar para ingeniería de código`).
- si durante la actualización del código de programa:
  - se revisa la sintaxis o no.
  - se generan automáticamente las `RealizacionesDeComponente` que faltan o no.
  - se generan los nombres de archivo de código que faltan en el código combinado o no.
  - se utilizan espacios de nombres en la ruta de acceso del archivo de código o no.
- el tipo de sangría que se aplica al código (es decir, tabulaciones o espacios).
- qué directorios se omiten cuando se actualice un proyecto de UModel con código. Separe los directorios con un punto y coma. Los directorios secundarios que se llamen igual también se pasarán por alto.
- la ubicación del archivo de catálogo de XMLSpy `RootCatalog.xml`, que permite a UModel y a XMLSpy recuperar esquemas, hojas de estilos y otros archivos utilizados con frecuencia desde carpetas de usuario locales. Esto aumenta la velocidad de procesamiento y permite al usuario trabajar sin conexión.



 © 2014 Altova GmbH

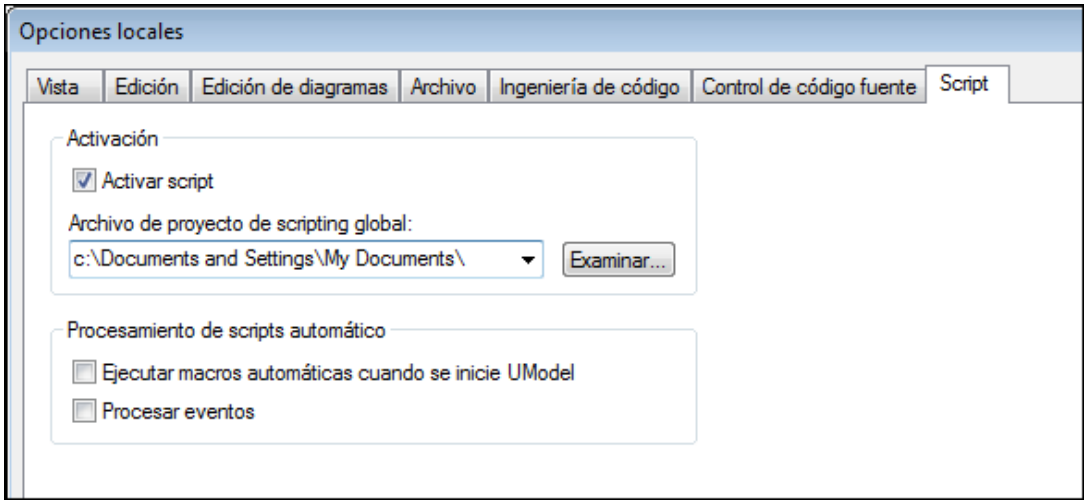
Pestaña *Control de código fuente*:
En esta pestaña puede elegir:

- el complemento de control de código fuente actual. Con el botón **Opciones avanzadas** puede configurar algunas opciones avanzadas del sistema de control de código seleccionado. Estas opciones avanzadas dependen del control de código fuente elegido.
- el id. de inicio de sesión para el proveedor de control de código fuente.
- otras opciones relacionadas con la protección y desprotección de archivos.
- el botón **Restaurar** se habilita si el usuario marca la casilla *No volver a mostrar* en un cuadro de diálogo. Con el botón **Restaurar** puede volver a habilitar los avisos.



Pestaña *Script*:
En esta pestaña puede definir:

- si el entorno de scripting debe activarse para el proyecto actual.
- qué archivo de scripting global se utiliza.
- si se deben ejecutar las macros automáticas cuando se inicia UModel.
- si se deben procesar los eventos de scripting.



 © 2014 Altova GmbH

17.7 Menú Ventanas

En cascada:
Este comando reorganiza todos las ventanas de documento que están abiertas en forma de **cascada** (es decir, las ventanas se apilan una encima de otra).

En mosaico horizontal:
Este comando reorganiza todas las ventanas de documento que están abiertas en forma de **mosaico horizontal** (es decir, se pueden ver todas las ventanas a la vez y se distribuyen de forma horizontal).

En mosaico vertical:
Este comando reorganiza todas las ventanas de documento que están abiertas en forma de **mosaico vertical** (es decir, se pueden ver todas las ventanas a la vez y se distribuyen de forma vertical).

Ordenar iconos:
Este comando reorganiza los elementos que estén dispersos por el diagrama y los coloca en la parte inferior del área de trabajo.

Cerrar:
Este comando cierra la pestaña del diagrama activo.

Cerrar todas:
Este comando cierra todas las pestañas de diagrama que están abiertas.

Cerrar ventanas inactivas:
Este comando cierra todas las pestañas de diagrama que están abiertas excepto la pestaña activa.

Adelante:
Este comando abre la siguiente pestaña de diagrama o el siguiente elemento con hipervínculo.

Atrás:
Este comando abre la pestaña de diagrama anterior o el elemento con hipervínculo anterior.

Lista de ventanas abiertas:
Esta lista muestra todas las ventanas que están abiertas en cada momento y permite cambiar de una ventana a otra rápidamente.
También puede usar las teclas de acceso rápido **Ctrl+Tabulador** o **Ctrl+F6** para recorrer todas las ventanas que están abiertas.

17.8 Menú Ayuda

El menú **Ayuda** contiene todos los comandos necesarios para obtener ayuda o información sobre UModel, así como algunos enlaces a páginas del sitio web de Altova que pueden serle de utilidad.

Desde este menú también puede abrir el cuadro de diálogo "Activación del software", donde puede introducir su código clave de licencia para activar UModel.

Contenido

Descripción
Abre la ayuda en pantalla por la tabla de contenido. Esta tabla de contenido ofrece una vista general de todo el documento de ayuda. Al hacer clic en una entrada del contenido se abre el tema correspondiente de la ayuda.

Índice

Descripción
Abre la ayuda en pantalla por el índice de palabras clave. Este índice temático enumera todas las palabras claves de la ayuda y permite navegar a un tema con solo hacer doble clic en la palabra clave correspondiente. Una palabra clave puede estar asociada a varios temas.

Buscar

Descripción
Abre la ayuda en pantalla por la función de búsqueda. Para buscar un término en el documento de ayuda escriba el término de búsqueda en el campo de consulta y pulse la tecla **Entrar**. Después haga doble clic en uno de los temas de la lista para ver su contenido.

 © 2014 Altova GmbH

Activación del software

Descripción

Después de descargar el software de Altova puede activarlo con un código clave de prueba gratis o con un código clave de licencia permanente.

- ***Código de prueba gratis:*** Cuando inicie el software por primera vez, aparece el cuadro de diálogo "Activación del software". En este diálogo hay un botón para solicitar un código clave de prueba gratis. Escriba su nombre, el nombre de su compañía y su dirección de correo electrónico en el diálogo y después pulse el botón **Enviar solicitud**. Nuestro servidor envía un código clave de prueba a la dirección de correo electrónico proporcionada (que suele llegar en solo unos minutos). Copie y pegue el código clave de prueba en el campo *Código clave* y pulse el botón **Aceptar** para empezar a trabajar con el software de Altova. El software permanecerá desbloqueado 30 días.
- ***Código clave permanente:*** El cuadro de diálogo "Activación del software" incluye un botón para comprar un código clave permanente. Al hacer clic en este botón se abre la página de la tienda en línea de Altova. Hay dos tipos de licencias permanentes: licencias de un solo usuario o de varios usuarios. Nuestro servidor envía la licencia a su correo electrónico. La *licencia de un solo usuario* contiene los datos de la licencia e incluye su nombre, el de su compañía, su correo electrónico y el código clave. La *licencia de varios usuarios* contiene los datos de la licencia e incluye el nombre de su compañía y el código clave. Recuerde que, según lo dispuesto en el contrato de licencia, no puede instalar más del número máximo de copias del software de Altova en equipos de su organización (licencia por número de usuarios). Asegúrese de que los datos introducidos en el cuadro de diálogo de activación del software coinciden con los datos del correo electrónico que incluye la licencia.

  Nota: cuando introduzca los datos de la licencia en el cuadro de diálogo "Activación del software", compruebe que los datos introducidos coinciden con los datos del correo electrónico que incluye la licencia. En caso de usar una licencia de varios usuarios, cada usuario debe introducir su nombre en el campo *Nombre*.

Para abrir el cuadro de diálogo "Activación del software" haga clic en el comando **Ayuda | Activación del software**.

Formulario de pedido

Descripción

Cuando desee comprar un código clave permanente, haga clic en el botón **Comprar un código clave permanente** del cuadro de diálogo "Activación del software" (*ver más arriba*) o seleccione el comando **Ayuda | Formulario de pedido**, que le llevará a la tienda en línea de Altova.

Registro del software

Descripción

Abre la página de registro del software en una pestaña del explorador web. Registro su software de Altova para recibir información sobre novedades del producto.

Buscar actualizaciones

Descripción

Busca una versión más reciente del producto en el servidor de Altova y emite un mensaje a tal efecto.

© 2014 Altova GmbH

Centro de soporte técnico

Descripción

Es un enlace al centro de soporte técnico del sitio web de Altova. El centro de soporte técnico incluye preguntas frecuentes, foros de debate y un formulario para ponerse en contacto con el equipo de soporte técnico de Altova.

Preguntas más frecuentes

Descripción

Es un enlace a la página de preguntas frecuentes del sitio web de Altova. Esta página se actualiza constantemente con las preguntas que recibimos de nuestros clientes.

Descargar herramientas gratis y componentes

Descripción

Es un enlace al centro de descargas de componentes del sitio web de Altova. Aquí puede descargar software adicional para usarlo con los productos de Altova, como procesadores XSLT y XSL-FO y paquetes de integración. Estos componentes suelen ser totalmente gratis.

UModel en Internet

Descripción

Es un enlace al sitio web de Altova, donde encontrará más información sobre UModel, otros productos de Altova y tecnologías relacionadas.

Acerca de UModel

Descripción

Abre la pantalla de presentación de la aplicación, que incluye el número de versión del producto e información sobre copyright.

Altova UModel® 2015

Generador de código

18 Generador de código

UModel incluye un generador de código integrado que puede generar archivos Java, C#, Visual Basic o de XML Schema automáticamente a partir de modelos UML.

Nota sobre el uso de **alias**:
Cuando aplique ingeniería inversa a código que contiene alias de espacio de nombres o alias de clase, ahora puede decidir si se resuelve el alias o no (casilla *Resolver los alias* del cuadro de diálogo "Importar proyecto/directorio de origen").

Cuando actualice el código (p. ej. cuando actualice el código con el modelo) las definiciones de alias se mantienen en el código tal y como están. El ámbito de las definiciones de alias son los archivos en los que aparecen.

P. ej.

```
using Q = System.Collections.Generic.Queue<String>;
Q myQueue;
```

Los alias que estén en conflicto y cuyo uso no esté del todo claro se añaden a la lista de "Elementos externos desconocidos".

Nota: puede cambiar la opción de configuración *Resolver los alias* siempre que quiera en el cuadro de diálogo "Configuración del proyecto".

 © 2014 Altova GmbH

18.1 Códigos de error

Códigos de error del sistema operativo

201 No se encontró el archivo: "%s"
202 No se puede crear el archivo "%s"
203 No se puede abrir el archivo "%s"
204 No se puede copiar el archivo "%s" en "%s"

Códigos de error de sintaxis

401 Se esperaba una palabra clave
402 Se esperaba "%s"
403 No se especificó el archivo de salida
404 Fin inesperado del archivo
405 Palabra clave no permitida

Códigos de error de tiempo de ejecución

501 Variable desconocida "%s"
502 Redefinición de la variable "%s"
503 La variable "%s" no es un contenedor
504 Propiedad desconocida "%s"
505 No se puede convertir %s en %s
507 Función desconocida
508 Ya se definió esta función
509 Parámetro no válido
510 División por cero
511 Método desconocido
512 Número incorrecto de parámetros
513 Desbordamiento de pila

Altova UModel® 2015

Referencia del programador

19 Referencia del programador

UModel como servidor de automatización
UModel es un servidor de automatización. Esto significa que es una aplicación que expone objetos programables a otras aplicaciones (llamadas clientes de automatización). Como resultado, el cliente de automatización tiene acceso directo a los objetos y a las funciones que el servidor de automatización pone a su disposición. Esto es una ventaja para el cliente de automatización porque puede usar todas las funciones de UModel, como la función de ingeniería inversa. Por tanto, los programadores pueden mejorar sus propias aplicaciones usando las funciones de UModel.

Los objetos programables de UModel se ponen a disposición de los clientes de automatización mediante la API de UModel, que es una API de COM. El modelo de objetos de la API se describe en la sección API de UModel, que también describe todos los objetos disponibles.

Entornos de ejecución
Puede acceder a la API de la aplicación desde estos entornos:

- Desde el editor de scripts
- Desde complementos creados en entornos IDE
- Desde programas externos

A continuación ofrecemos una breve descripción de estos tres entornos.

Editor de scripts: personalizar y modificar las funciones de UModel
Si quiere puede personalizar su versión de UModel modificando y añadiéndole funciones. También puede crear formularios y modificar la interfaz del usuario añadiéndole comandos de menú nuevos e iconos nuevos en las barras de herramientas. Para ello basta con escribir scripts que interactúen con objetos de la API de la aplicación. Para ayudarle a escribir estos scripts, UModel le ofrece un editor de scripts integrado. Las funciones de este editor se describen detalladamente en el apartado Editor de scripts de este manual. Los lenguajes de programación compatibles son **JScript** y **VBScript**.

Complementos creados en entornos IDE: cómo crear complementos para trabajar con UModel
Gracias a la interfaz especial de UModel para complementos, podrá crear complementos propios e integrarlos en UModel. Esto se describe detalladamente en el apartado Crear complementos en entornos IDE.

El objeto de aplicación se pasa a la mayoría de los métodos que el complemento debe implementar y se le llama desde la aplicación. Los principales lenguajes utilizados para implementar un complemento creado en un entorno IDE son **C#** y **C++**. Consulte el apartado Crear complementos en entornos IDE para obtener más información.

Programas externos

 © 2014 Altova GmbH

También puede manipular UModel usando scripts externos. Por ejemplo, puede escribir un script para abrir UModel en un momento determinado, después abrir un proyecto de UModel, generar su documentación e imprimirla. Al igual que los entornos anteriores, los scripts externos utilizan la API de la aplicación para realizar estas tareas. Para más información consulte el apartado API de UModel.

Para poder usar la API de UModel desde programas externos es necesario iniciar una instancia de UModel primero. Esto se puede hacer de varias formas, dependiendo del lenguaje de programación utilizado (consulte el apartado Lenguajes de programación).

Básicamente UModel se iniciará a través de su registro de COM. Después se devuelve el objeto `Application` asociado con la instancia de UModel. Dependiendo de la configuración de COM, se puede devolver un objeto asociado con una instancia que ya está en ejecución. Además se puede usar cualquier lenguaje de programación que permita crear e invocar objetos COM. Los más frecuentes son:

- Los archivos JScript y VBScript tienen una sintaxis sencilla y están diseñados para acceder a objetos COM. Se pueden ejecutar desde la línea de comandos directamente o haciendo doble clic en ellos desde el explorador de Windows. Son muy útiles para tareas de automatización sencillas.
- C# es un lenguaje de programación de pleno derecho con una amplísima gama de funciones. El acceso a objetos COM se puede encapsular automáticamente usando C#..
- Java: los productos de Altova vienen con clases Java nativas que encapsulan la API de la aplicación y ofrecen un aspecto Java total
- También puede usar Visual Basic for Applications, Perl y Python.

Información sobre la referencia del programador de UModel

Esta sección de la documentación se divide en varios apartados:

- Notas sobre la versión: enumera cambios realizados desde la versión anterior.
- Editor de scripts: referencia del usuario para el entorno de scripts integrado de UModel.
- Complementos creados en entornos IDE: explica cómo crear complementos para UModel.
- API de UModel: referencia para la API de la aplicación.

19.1 Notas sobre la versión

A continuación se enumeran todos los cambios importantes incorporados en cada versión de la API de UModel.

Si en la versión principal de la biblioteca de tipos se produce un cambio (p. ej. 1.0 => 2.0 => 3.0), significa que no se debería volver a compilar clientes como los complementos creados en C#, VB.NET, C++, etc. desde entornos IDE.

Interfaz de automatización para UModel 2013: biblioteca de tipos versión 5.0
Estos son los cambios incorporados desde la versión anterior (UModel 2012, biblioteca de tipos versión 4.1):

API de UModel

- IDocument tiene un método nuevo: **GenerateSequenceDiagramsForAllOperations**

API de UModel - UMLData

- ENUMExportXMIType tiene una entrada nueva para UML2.4 (`eXMI24ForUML24`).
- ENUMUMLGuiTextLabelKind tiene un literal nuevo `eTextLabel_DotNetPropertyName`
- ENUMUMLPredefinedElement tiene literales nuevos para SysML 1.2.
- IUMLGuiSequenceDiagram tiene propiedades nuevas relacionadas con la generación de código: **UseForForwardEngineering** y **CodeOperation**.
- IUMLExecutionEvent, IUMLCreationEvent, IUMLDestructionEvent, IUMLSendOperationEvent, IUMLSendSignalEvent, IUMLReceiveOperationEvent y IUMLReceiveSignalEvent se han eliminado porque sus clases correspondientes ya no forman parte de UML2.4.

Interfaz de automatización para UModel 2012: biblioteca de tipos versión 4.1
Estos son los cambios incorporados desde la versión anterior (UModel 2011r3, biblioteca de tipos versión 4.0):

- Se introducen IModelTransformationDlg, IModelTransformationTypeMappings y IModelTransformationTypeMapping para las transformaciones de modelos. IDocument tiene un método nuevo: **ModelTransformation**.
- ILocalOptionsView tiene una propiedad nueva: **EnableSnapLines**.

API de UModel - UMLData

- ENUMCodeLang tiene un literal nuevo: `eCodeLang_UML`.
- ENUMUMLPredefinedElement tiene varios literales nuevos para las transformaciones de modelos.

Interfaz de automatización para UModel 2011r3: biblioteca de tipos versión 4.0
Estos son los cambios incorporados desde la versión anterior (UModel 2011r2, biblioteca de tipos versión 3.2):

API de UModel

- ILocalOptionsDiagramEditing tiene propiedades nuevas para controlar propiedades .NET:

 © 2014 Altova GmbH

UseDotNetPropertyCompartment y **ShowDotNetPropertyCompartment**.
- IDialog tiene dos propiedades nuevas: **Application** y **Parent**.
- IImportSourceDlg tiene propiedades nuevas para controlar propiedades .NET: **Content_UseDotNetPropertyCompartment** y **Content_ShowDotNetPropertyCompartment**.

API de UModel - UMLData
- Se puede establecer la propiedad **BehaviorSpecification** para IUMLBehavior
- ENUMUMLGuiStyleKind tiene un literal nuevo para controlar propiedades .NET: **eUMLGuiStyle_ShowDotNetPropertyCompartment.**

Interfaz de automatización para UModel 2011r2: biblioteca de tipos versión 3.2
Estos son los cambios incorporados desde la versión anterior (UModel 2011, biblioteca de tipos versión 3.1):

API de UModel
- Se introduce IGenerateStateMachineCodeDlg para la generación de código de máquina de estados y IDocument tiene un método nuevo: **GenerateStateMachineCode**.
- IGenerateDocumentationDlg tiene propiedades nuevas para la generación de documentación con archivos SPS (**UseFixedDesign** y **SPSFile**), una propiedad nueva (**Include_IncludedPredefinedSubprojects**) y un método nuevo (**Fonts_SetDefaults**).
- ENUMDocumentationOutputFormat tiene un literal nuevo para generar documentación en formato PDF.
- ENUMUMLPredefinedElement tiene varios literales nuevos para funciones relacionadas con BPMN2.

API de UModel - UMLData
- IUMLGuiBPMN2Diagram, IUMLGuiBPMN2ChoreographyDiagram y IUMLGuiBPMN2CollaborationDiagram se introducen para BPMN2

Interfaz de automatización para UModel 2011: biblioteca de tipos versión 3.1
Estos son los cambios incorporados desde la versión anterior (UModel 2010r3, biblioteca de tipos versión 3.0):

API de UModel
- IImportDatabaseDlg se introduce para la importación de bases de datos.
- IDocument tiene un método nuevo para importar bases de datos (**ImportDatabase**) y otro para fusiones de proyecto a 3 bandas (**MergeProject3Way**).

API de UModel - UMLData
- ENUMCodeLangVersion y ENUMUMLPredefinedElement tienen literales nuevos para funciones relacionadas con bases de datos.
- ENUMUMLDBDataSourceMethod se introduce para funciones relacionadas con bases de datos.

Interfaz de automatización para UModel 2010r3: biblioteca de tipos versión 3.0
Estos son los cambios incorporados desde la versión anterior (UModel 2010r2, biblioteca de tipos versión 2.1):

API de UModel

- IGenerateSequenceDiagramDlg tiene una propiedad nueva para omitir operaciones diferentes cuando se generen diagramas de secuencia a partir de código fuente (**OperationIgnoreList**).
- La interfaz IGenerateDocumentationDlg tiene la propiedad nueva **Details_Constraints**.
- ENUMDiagramLayoutKind tiene una entrada nueva para el tipo de diseño “Block” (por bloques).
- ENUMCodeLangVersion tiene una entrada nueva para C# 4.0.

API de UModel - UMLData

- IUMLElement tiene un método nuevo para recuperar todos los elementos propios de un tipo determinado (**GetOwnedElementsOfKind**).
- IUMLClass, IUMLEnumeration y IUMLInterface tienen un método nuevo para obtener la ruta de acceso completa del archivo de código (**GetCodeFilePath**). Véase también GetCodeFileName, que devuelve el nombre del archivo solamente).
- IUMLInterface tiene la propiedad nueva **Protocol** y el método nuevo **SetNewProtocol** para remitir a un IUMLProtocolStateMachine.
- IUMLConstraint tiene dos propiedades nuevas: **OwningTransition** y **OwningState**.
- IUMLState tiene la propiedad nueva **StateInvariant** y el método nuevo **SetNewStateInvariant**.
- IUMLPort tiene una propiedad nueva: **Protocol.**
- IUMLStructuredClassifier tiene un método nuevo: **InsertOwnedPortAt**.
- ENUMUMLPredefinedElement tiene varios literales nuevos para C# 4.0.
- Además se introdujeron estas nuevas interfaces: IUMLValueSpecificationAction, IUMLProtocolStateMachine, IUMLProtocolTransition, IUMLGuiProtocolStateMachineDiagram

Interfaz de automatización para UModel 2010r2: biblioteca de tipos versión 2.1

Estos son los cambios incorporados desde la versión anterior (UModel 2010, biblioteca de tipos versión 2.0):

API de UModel

- IGenerateSequenceDiagramDlg tiene una propiedad nueva para la nueva división de diagramas de secuencia (**SplitIntoSmallerDiagrams**).

API de UModel - UMLData

- ENUMExportXMIType tiene una entrada nueva para UML2.3.
- IUMLAction tiene una propiedad nueva: **IsLocallyReentrant**.
- IUMLPort tiene una propiedad nueva: **IsConjugated**.
- La propiedad **ConnectorKind** de IUMLConnector ahora es de solo lectura.
- IUMLClassifier tiene una propiedad nueva: **IsFinalSpecialization**.
- IUMLActivityGroup ahora se deriva de IUMLNamedElement.

Interfaz de automatización para UModel 2010: biblioteca de tipos versión 2.0

Estos son los cambios incorporados desde la versión anterior (UModel 2009, biblioteca de tipos versión 1.0):

Complementos para UModel

- Compatibilidad con controles ActiveX. Los complementos que también sean controles

 © 2014 Altova GmbH

ActiveX aparecen en una barra de control de diálogos de UModel (véase también Controles ActiveX o el ejemplo StatisticsActiveX)

API de UModel

- IApplication tiene una propiedad nueva (**ServicePackVersion**) un método nuevo (**RunMacro**) para iniciar una macro de un proyecto de script ya cargado.
- IDocument tiene métodos nuevos: **SaveCopyAs**, **CanFocusUMLDataInModelTree**, **FocusUMLDataInModelTree**, **Reload**.
- La propiedad **FocusedUMLDataNotifier** de IDocument se puede usar para obtener la interfaz nueva IFocusedUMLDataEvents.
- El método **OnModifiedFlagChanged** de IDocumentEvents tiene la interfaz IDocument como segundo parámetro.
- La interfaz IDiagramWindow tiene estos métodos nuevos para la función de diseño automático: **Autolayout**, **AutolayoutSelection**.
- La interfaz IProjectSettingsDlg tiene dos propiedades nuevas: **CSharp_ResolveAliases** y **VBasic_ResolveAliases**.
- La interfaz IGenerateDocumentationDlg tiene dos propiedades nuevas: **EmbedCSSinHTML** y **CreateFolderForDiagrams**.
- ILocalOptionsCodeEngineering tiene estas propiedades nuevas: **CodeFromModel_Indentation_InsertTabs**, **CodeFromModel_Indentation_InsertNSpaces**.

API de UModel - UMLData

- La nueva interfaz IUMLHyperlink2Model permite crear hipervínculos a elementos de modelado (en la Estructura del modelo). IUMLNamedElement tiene **InsertOwnedHyperlink2ModelAt**, IUMLGuiTextHyperlink tiene **SetHyperlinkModelElementAddress** para definir vínculos con elementos de modelado.
- IUMLCommentTextHyperlink es nueva y habilita los hipervínculos para IUMLComments (véase también Cómo crear y usar hipervínculos). IUMLComment se amplía con **InsertOwnedCommentTextHyperlinkAt** y **OwnedHyperlinks** para insertar y acceder a estos hipervínculos.
- Igualmente IUMLGuiDiagram se amplía con **InsertOwnedGuiTextHyperlinkAt** y **OwnedHyperlinks**.
- La interfaz IUMLElement tiene dos propiedades nuevas (**OwnedDocComment** y **OwnedDocCommentBody**) para acceder directamente al cuerpo del comentario, que aparece en el panel Documentación cuando el elemento está resaltado.
- IUMLGuiDiagram tiene un método nuevo (**AddUMLGuiContainmentLink**) para insertar líneas de contención en los diagramas.
- ENUMUMLGuiTextLabelKind tiene un literal nuevo: **eTextLabel_InformationFlow**.
- ENUMUMLPredefinedElement tiene literales nuevos para tipos de datos XSD y el perfil SysML.
- Además se añadieron estas nuevas interfaces: IUMLInformationFlow, IUMLGuiContainmentLink, IUMLGuiSysMLActivityDiagram, IUMLGuiSysMLBlockDefinitionDiagram, IUMLGuiSysMLInternalBlockDiagram, IUMLGuiSysMLPackageDiagram, IUMLGuiSysMLParametricDiagram, IUMLGuiSysMLRequirementDiagram, IUMLGuiSysMLSequenceDiagram, IUMLGuiSysMLStateMachineDiagram, IUMLGuiSysMLUseCaseDiagram

19.2 Editor de scripts

El Editor de scripts de UModel usa los componentes del Editor de formularios de Microsoft .NET Framework y, por tanto, ofrece acceso a Microsoft .NET Framework. Esto significa que además de usar JScript y VBScript con la API de UModel (que es un API de COM), también puede acceder a las clases de Microsoft .NET Framework y utilizarlas.

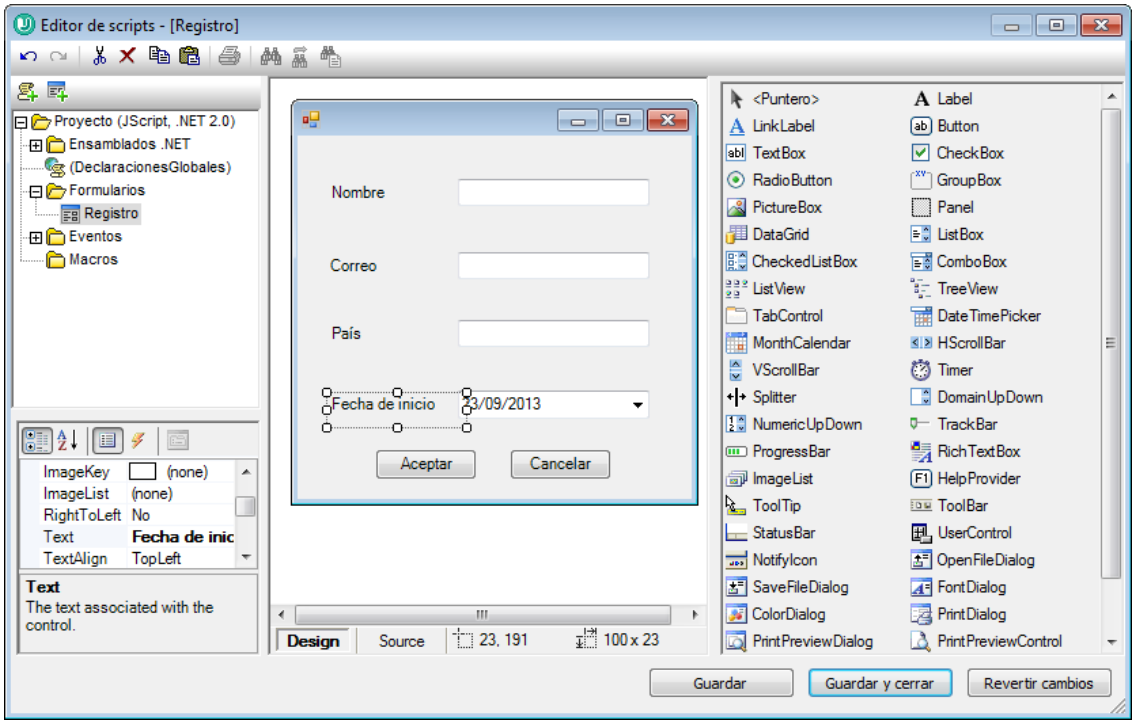
Por tanto, dentro de UModel podrá crear y usar sus propias macros y formularios para modificar y agregar funciones a la aplicación.

Nota: el entorno de scripting **no es compatible** con el lenguaje Visual Basic. Sí lo es con VBScript y JavaScript. Por tanto, asegúrese de utilizar la sintaxis VBScript y no la sintaxis Visual Basic en el entorno de scripting.

Nota: para que funcione el Editor de scripts es necesario instalar Microsoft **.NET Framework 2.0 o superior** antes de instalar UModel.

El Editor de scripts

El Editor de scripts (*imagen siguiente*) se abre en una ventana nueva haciendo clic en el comando de menú **Herramientas | Editor de scripts** de UModel. Los lenguajes de programación que se pueden usar en el entorno de scripting son **JScript** y **VBScript**. El lenguaje de scripting del proyecto se puede cambiar: haga clic con el botón secundario en el nodo `Proyecto` del panel superior izquierdo del Editor y, en el menú contextual, seleccione **Configuración del proyecto**. Ahora seleccione el lenguaje pertinente en el cuadro de diálogo.



¿Qué puede hacer con el editor de scripts?

En el Editor de scripts puede crear formularios, controladores de eventos y macros para construir un proyecto de scripting. Los proyectos de scripting pueden ser proyectos globales para UModel, lo cual permite usar los scripts del proyecto en la aplicación. Pero los proyectos también se

 © 2014 Altova GmbH

pueden asignar a proyectos distintos de UModel, lo cual permite usar scripts distintos en proyectos de UModel diferentes.

En cada proyecto de scripting puede definir la versión del Runtime de .NET que se debe utilizar. Una aplicación puede trabajar con varios proyectos de scripting con versiones diferentes del Runtime de .NET siempre y cuando las versiones correspondientes de .NET estén instaladas. Por ejemplo: los proyectos de scripting con .NET 4.0 solo funcionan en equipos que tengan .NET 4.0 instalado.

Documentación sobre el Editor de scripts

La presente documentación del entorno de scripting se divide en varias secciones:

- Información general: resumen sobre las características del Editor de scripts y los proyectos de scripting
- Instrucciones para crear un proyecto de scripting
- Qué son las declaraciones globales
- Instrucciones para crear formularios
- Controladores de eventos de UModel
- Instrucciones para usar macros en el Editor de scripts y en UModel.

19.2.1 Información general

El Editor de scripts ofrece una interfaz gráfica donde puede (i) diseñar formularios de forma gráfica asignando scripts a los componentes del formulario, (ii) crear controladores de eventos y (iii) crear macros.

Estos formularios, controladores de eventos y macros se organizan en proyectos de scripting, que se asignan a los proyectos de UModel y se pueden usar en la aplicación.

También puede definir variables y funciones en un script de declaraciones globales, que siempre se ejecuta antes que los scripts de macros y controladores de eventos.

Esta sección ofrece información general sobre el Editor de scripts y los proyectos de scripting y se divide en varios apartados:

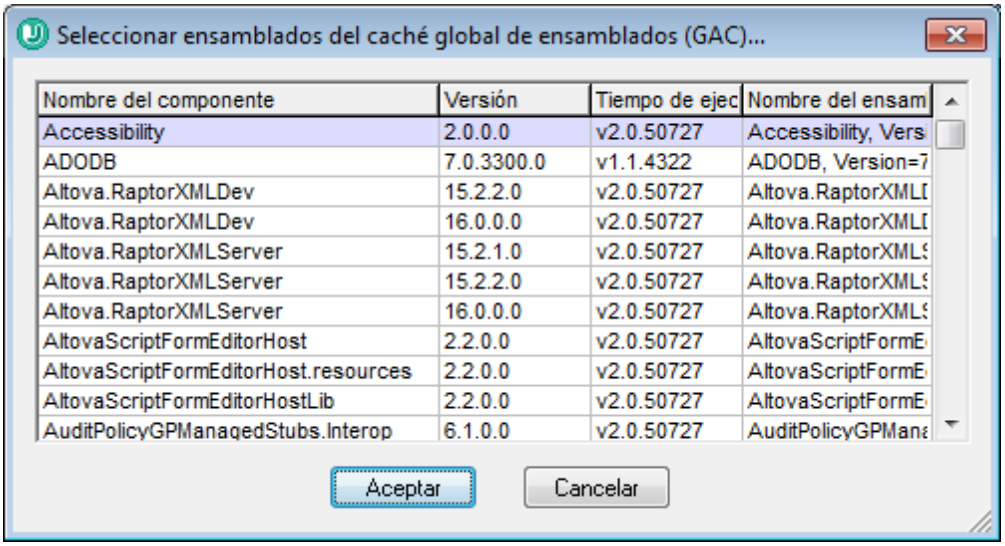
- Proyectos de scripting en UModel: describe cómo se usan en UModel los proyectos de scripting creados con el Editor de scripts.
- Interfaz gráfica del usuario del Editor de scripts: información detallada sobre la interfaz gráfica del Editor de scripts y sus componentes.
- Componentes de un proyecto de scripting: explica qué componentes constituyen un proyecto de scripting.

Para aprender a crear los diferentes componentes del proyecto de scripting, consulte los apartados correspondientes de las secciones Declaraciones globales, Formularios, Controladores de eventos y Macros.

Ensamblados .NET
Los proyectos de scripting pueden tener referencias a ensamblados .NET (además de referencias predeterminadas). Los ensamblados .NET se pueden añadir a todo el proyecto de scripting o a las macros (usando el nuevo comando `CLR.LoadAssembly` en el código fuente, como se explica en el apartado Comandos integrados). Por ejemplo, puede añadir ensamblados desde el caché global de ensamblados.

Para añadir un ensamblado haga clic con el botón secundario en el proyecto o en una macro y, en el menú contextual, seleccione **Agregar ensamblado .NET | Caché global de ensamblados (GAC)**.

 © 2014 Altova GmbH



Esto funciona igual que en Visual Studio y no solo ofrece acceso a todo Microsoft .NET Framework sino también a cualquier ensamblado definido por el usuario.

Proyectos de scripting en UModel

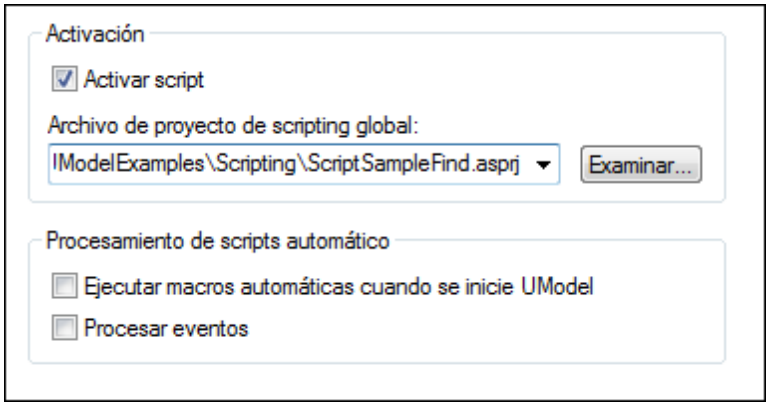
Todos los scripts creados en el Editor de scripts se almacenan en proyectos de scripting de Altova (archivos **`.asprj`**) y puede crear tantos proyectos de scripting de Altova como quiera. Una vez creado, el archivo .asprj puede tener varios usos:

- Se puede definir como proyecto global de scripting para UModel. A los scripts de un proyecto global de scripting se les puede llamar desde dentro de la aplicación y las macros del proyecto global de scripting se pueden usar para todos los proyectos de UModel.
- Se puede asignar a un proyecto de UModel (como proyecto de aplicación). Cuando se abre un proyecto de UModel en UModel, se pueden invocar los scripts del proyecto de scripting asociado.

UModel viene con un ejemplo de proyecto de scripting llamado `ScriptSampleFind.asprj`. Este archivo está en la carpeta: `C:\Documents and Settings\<usuario>\Mis Documentos\Altova\UModel2015\Examples\` y contiene declaraciones globales para varias tareas estándar.

Definir el proyecto de scripting global de una aplicación

El proyecto de scripting global de una aplicación se define en la pestaña *Script* del cuadro de diálogo "Opciones" de UModel (*imagen siguiente*).

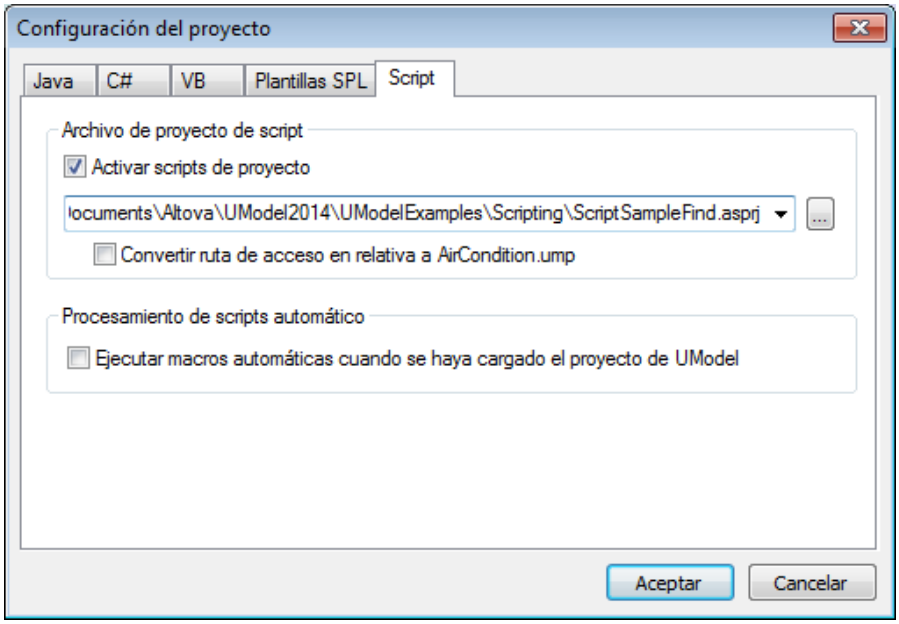


Para definir un proyecto de scripting global para UModel, marque la casilla *Activar script* y después busque el proyecto de scripting (archivo `.asprj`) que desea activar. En esta pestaña también puede indicar: (i) si las macros automáticas del proyecto de scripting se ejecutan automáticamente cuando se inicie UModel y (ii) si los scripts de control de eventos de la aplicación del proyecto se deben ejecutar automáticamente o no.

Nota: se pueden ejecutar scripts anidados, es decir, las macros pueden llamar a otras macros y los eventos se reciben durante la ejecución de macros o de eventos.

Para asignar un proyecto de scripting a un proyecto de UModel:

1. En UModel abra el proyecto de aplicación pertinente.
2. Seleccione el comando de menú **Proyecto | Configuración del proyecto | Script** y haga clic en la pestaña *Script*.



3. Marque la casilla *Activar scripts de proyecto* y seleccione el proyecto de scripting pertinente (archivo `.asprj`). Si quiere ejecutar las macros automáticas cuando se cargue el proyecto de UModel, marque la casilla *Ejecutar macros automáticas*.
4. Para terminar haga clic en **Aceptar**.

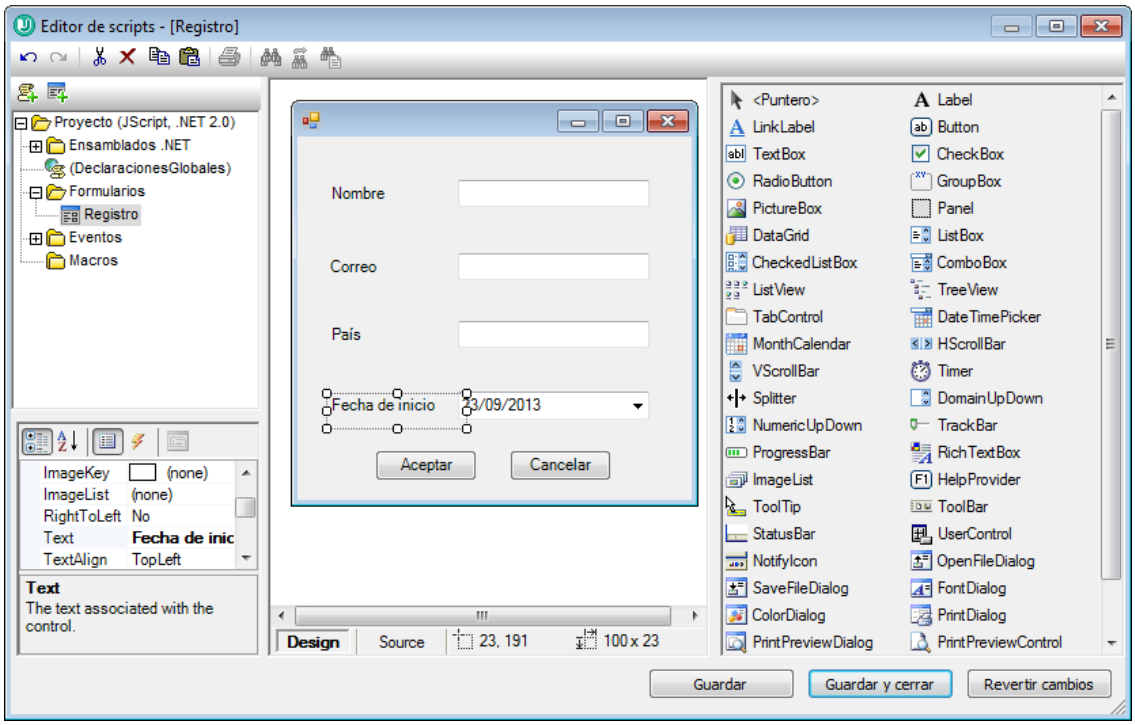
   Nota: para desactivar el proyecto de scripting de un proyecto de UModel (es decir, para anular la asignación), desactive la casilla *Activar scripts de proyecto*.

 © 2014 Altova GmbH

La interfaz gráfica del editor de scripts

A continuación puede ver una imagen de la interfaz gráfica del usuario del Editor de scripts, que se compone de:

- Una barra de herramientas
- El panel Estructura del proyecto de scripting (arriba a la izquierda)
- El panel Propiedades y eventos (abajo a la izquierda)
- Una ventana principal con las pestañas *Design* y *Source* para el diseño y el código fuente respectivamente
- Una paleta de objetos de formulario (en el lateral derecho)



Barra de herramientas del Editor de scripts

La barra de herramientas del Editor de scripts incluye iconos para:

- Comandos de archivo como **Nuevo**, **Abrir**, **Guardar** e **Imprimir**. Estos comandos sirven para crear proyectos de scripting nuevos, abrir proyectos de scripting, guardarlos e imprimirlos.
- Comandos de edición estándar como **Copiar**, **Pegar**, **Deshacer**, **Rehacer**, **Buscar** y **Reemplazar**. Recuerde que los comandos **Buscar** y **Reemplazar** afectan al código de la pestaña *Source* del Editor de scripts.

Estructura del proyecto de scripting

La estructura del proyecto de scripting se representa en el panel superior izquierdo (*imagen siguiente*) y muestra los componentes que constituyen el proyecto de scripting. Estos componentes pueden dividirse en cuatro categorías principales: (i) **`Declaraciones globales`**, (ii) **`Formularios`**, (iii) **`Eventos`** y (iv) **`Macros`**.

Desde el panel Estructura del proyecto puede acceder a todos los componentes del proyecto de scripting. Por ejemplo, si quiere ver y editar un formulario determinado, expanda la carpeta **`Formularios`** (*imagen anterior*), haga clic con el botón secundario en el formulario que le interesa y, en el menú contextual, seleccione **Abrir**.

La forma más rápida de abrir un formulario, evento, macro o script de declaraciones globales es hacer doble clic en el icono correspondiente. Para eliminar un formulario o una macro del proyecto de scripting, haga clic con el botón secundario en el componente y, en el menú contextual, seleccione **Eliminar**.

El panel Estructura del proyecto de scripting también incluye una barra de herramienta con varios iconos (*imagen siguiente*).

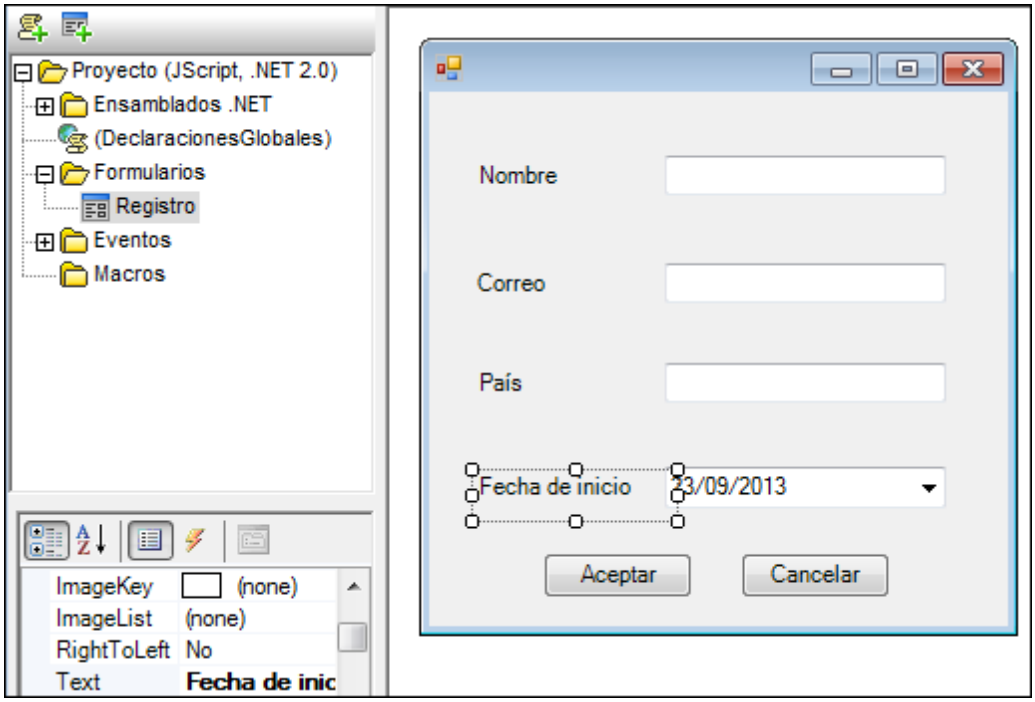
Estos iconos (de izquierda a derecha) sirven para (i) crear una macro nueva, (ii) crear un formulario nuevo, (iii) ejecutar una macro y (iv) depurar una macro. Estos comandos también están disponibles en el menú contextual que aparece al hacer clic con el botón secundario en cualquier componente de la Estructura del proyecto de scripting.

Propiedades y eventos

El panel Propiedades y eventos (*imagen siguiente*) muestra:

- Propiedades de formularios (cuando se selecciona un formulario)
- Propiedades de objetos (cuando se selecciona un objeto de un formulario, como en la imagen siguiente)
- Eventos de formularios (cuando se selecciona un formulario)
- Eventos de objetos (cuando se selecciona un objeto de un formulario)

 © 2014 Altova GmbH



Para cambiar entre las propiedades y los eventos del componente seleccionado, haga clic en el icono **Propiedades** (el tercero empezando por la izquierda de la barra de herramientas del panel Propiedades y eventos) o en el icono Eventos (el cuarto icono empezando por la izquierda).

El primer y el segundo icono de la barra de herramientas empezando por la izquierda son los iconos **Por categorías** y **Alfabético** que muestran las propiedades y eventos ordenados por categorías o por orden alfabético ascendente respectivamente.

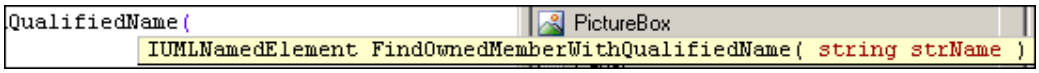
Cuando se selecciona una propiedad o evento, en la parte inferior del panel aparece una breve descripción.

Ventana principal

La ventana principal muestra el componente seleccionado y tiene una o dos pestañas, dependiendo del tipo de componente. Si se trata de un script de declaraciones globales, de un evento o de una macro, entonces aparece solo la pestaña *Source* que muestra el código fuente del componente seleccionado.

La pestaña *Source* para el código fuente ofrece estas características:

- color de sintaxis
- plegamiento de código
- creación y eliminación de marcadores con **Ctrl+F2**
- finalización automática con información sobre los parámetros (*imagen siguiente*)
- extender e ir a la llave
- método completo o firma de la propiedad junto al ayudante de entrada de finalización automática (*imagen anterior*)



- resaltado de llaves durante la entrada de código

```
if ( x == y.GetName( a, b, c() ) )
```

- información rápida (firma y documentación) al pasar el cursor del mouse sobre métodos o propiedades conocidas:

```
var sDir = objApplication.InstallationDirectory;
        string IApplication.InstallationDirectory
        installation directory of the application, e.g. C:\Program Files\Altova\UModel2009
```

Si la ventana principal muestra un **formulario**, entonces incluye dos pestañas: la pestaña *Design* (que muestra el formulario y permite diseñarlo) y la pestaña *Source* (que contiene el código fuente del formulario). El contenido de ambas pestañas se puede editar.

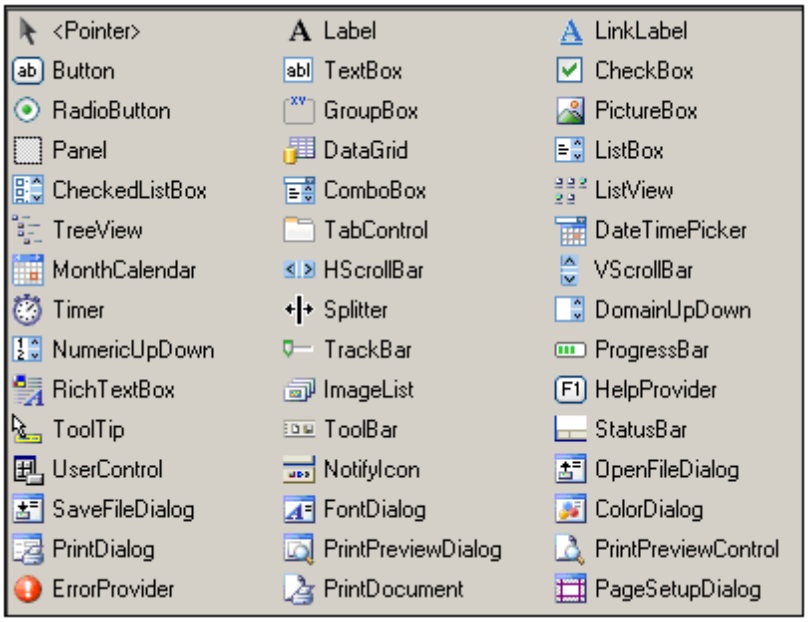
Notas:
JScript y VBScript son lenguajes sin tipo y por eso los ayudantes de entrada y la función de finalización automática solamente funciona en caso de "construcciones completas" como `Application.ActiveDocument.RootPackage.xxx` y nombres "predefinidos" como `objDocument123.RootPackage.xxx`.

Si los nombres empiezan por `objUMLxxx`, aparecerán los miembros de la interfaz `IUMLxxx` correspondiente. Por ejemplo, la API de COM de UModel tiene una interfaz: `IUMLClass`. Si mientras escribe el código utiliza `objUMLClass`, `objUMLClass123` o `objUMLClassParent`, aparecerán los miembros de la clase `IUMLClass` correspondiente.

Si los nombres empiezan por `objApplication`, `objDocument` o `objDiagramWindow`, entonces aparecerán los miembros de la interfaz correspondiente. Esto también afecta a todas las demás interfaces definidas en la API de UModel. Durante la edición aparece el ayudante de entrada de finalización automática y la información sobre parámetros. No obstante, puede invocar a estos ayudantes de entrada en cualquier momento con solo pulsar **Ctrl+Barra espaciadora**.

Paleta de objetos de formulario
La paleta de objetos de formulario contiene todos los objetos que están disponibles para el diseño de formularios (*imagen siguiente*). También puede añadir controles ActiveX registrados a la paleta. Para ello haga clic con el botón secundario en la paleta y seleccione el comando **Agregar control ActiveX**.

 © 2014 Altova GmbH



Para insertar un objeto de la paleta en el diseño de la ventana principal, haga clic en el objeto y después en la parte del formulario donde desea insertar el objeto. El objeto se inserta en la posición elegida y en muchos casos deberá definir algunas de sus propiedades (en el panel Propiedades y eventos). Los objetos se pueden arrastrar de una parte a otra del formulario y su tamaño se puede ajustar. Además si hace clic con el botón secundario en el objeto aparece un menú contextual con varios comandos de edición (p. ej. comandos para centrar el objeto, para alinearlo, etc.).

Algunos objetos de formulario, como el temporizador `Timer`, no se añaden al formulario sino que se insertan en una bandeja situada debajo de la ventana principal del Editor de scripts. Después puede seleccionar el objeto en la bandeja y definir sus propiedades y controladores de eventos en el panel Propiedades y evento.

Componentes de un proyecto de scripts

Los proyectos de scripting de Altova están formados por cuatro componentes principales:

- *Declaraciones globales:* este componente contiene las definiciones de variables y funciones que los scripts de formularios, macros y controladores de eventos del proyecto de scripting pueden utilizar.
- *Formularios*: este componente contiene todos los formularios definidos en el proyecto de scripting.
- *Eventos:* este componente contiene los scripts de control de eventos para todos los eventos de la aplicación (no del formulario).
- *Macros:* este componente contiene todas las macros definidas en el proyecto de scripting.

Estos componentes aparecen representados en el panel Estructura del proyecto de scripting del Editor de scripts (*imagen siguiente*) y se pueden abrir desde allí.

A continuación ofrecemos una breve descripción de cada uno de los componentes.

Declaraciones globales
El componente `DeclaracionesGlobales` es un script que contiene variables y funciones que pueden ser utilizadas por formularios, controladores de eventos y macros. Las funciones se sirven de la para acceder a las funciones de UModel. Si crea una función o variable en el módulo `DeclaracionesGlobales` los formularios, controladores de eventos y macros del proyecto de scripting podrán acceder a la variable o función.

Para agregar una variable o función, abra el componente `DeclaracionesGlobales` (clic con el botón secundario en `DeclaracionesGlobales` en la Estructura del proyecto de scripting y después clic en **Abrir**) y edite el script de declaraciones globales en la ventana principal.

Formularios
En el Editor de scripts puede construir un formulario de forma gráfica con ayuda de la paleta de objetos de formulario, que contiene objetos como campos de entrada de texto y botones. Por ejemplo: puede crear un formulario para aceptar la entrada de un nombre de elemento para después eliminar todas las instancias de dicho elemento en el activo.

En este formulario puede asociar un script de función a un cuadro de texto que tome una variable de entrada. Y puede asociar un controlador de eventos a un botón para que empiece a ejecutar la función de eliminación, que está disponible en la. El formulario se invoca con una llamada desde dentro de una función (en el script Declaraciones globales) o desde una macro directamente. Para más información sobre cómo crear y editar formularios consulte el apartado Formularios.

Control de eventos
Los scripts de control de eventos se pueden asociar a una gran variedad de eventos. Puede controlar eventos que tengan lugar dentro de formularios (eventos de formulario) y dentro de la interfaz de la aplicación (eventos de aplicación). El script asociado a un evento se ejecuta inmediatamente en cuanto el evento se desencadena.

La mayoría de los eventos tienen parámetros que ofrecen información detallada sobre el evento. El valor devuelto por el script suele indicar a la aplicación cómo debe continuar con su procesamiento (por ejemplo, puede que la aplicación no permita la edición).

Un controlador de eventos se ejecuta cuando el evento tiene lugar dentro del formulario o dentro de UModel. Para más información sobre cómo crear controladores de eventos consulte el apartado Eventos.

 © 2014 Altova GmbH

Macros
Las macros se utilizan para implementar tareas complejas y repetitivas. Las macros no usan parámetros ni valores devueltos.

En una macro se puede acceder a todas las variables y funciones declaradas en el script de declaraciones globales y también se pueden mostrar formularios.

Para ver un ejemplo de cómo crear una macro consulte el apartado Escribir una macro. También puede consultar el apartado para aprender a llamar a una macro. Las macros se ejecutan desde la interfaz de UModel.

19.2.2 Crear un proyecto de scripting

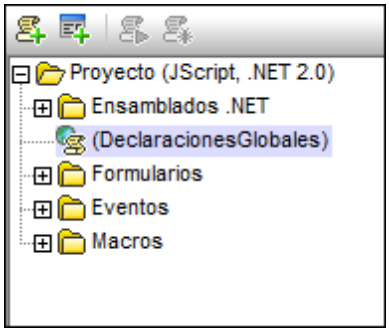
Los formularios, controladores de eventos y macros se crean en el Editor de scripts. No obstante, no todos ellos se invocan y ejecutan de la misma manera. Esto debe tenerse en cuenta a la hora de crear el proyecto de scripting.

- Los formularios se invocan con una llamada desde dentro de una función del script de declaraciones globales o de una macro directamente.
- Los controladores de eventos se ejecutan cuando el evento correspondiente ocurre en Si el controlador de un evento está definido tanto en el proyecto de scripting global como en el proyecto de scripting de un proyecto de UModel, primero se ejecuta el controlador de eventos del proyecto de scripting del proyecto y después el del proyecto de scripting global.
- Las macros se ejecutan desde la interfaz de UModel. En una macro se puede tener acceso a todas las funciones y variables declaradas en el script de declaraciones globales y mostrar los formularios de entrada del usuario.

© 2014 Altova GmbH

19.2.3 Declaraciones globales

El componente **DeclaracionesGlobales** está presente por defecto en todos los proyectos de scripting (*imagen siguiente*) y, por tanto, no hace falta crearlo. Para poder agregar variables y funciones al script de declaraciones globales, debe abrir el script de declaraciones globales y añadirle el fragmento de código correspondiente. Para más información consulte el apartado Componentes de un proyecto de scripting y Crear un proyecto de scripting.



Para abrir un script de declaraciones globales de un proyecto de scripting, primero haga clic con el botón secundario en el nodo **DeclaracionesGlobales** de la Estructura del proyecto de scripting (*imagen anterior*). Después seleccione **Abrir** y el script de declaraciones globales se abre en la ventana principal del Editor de scripts.

Nota: cada vez que se ejecuta una macro o que se llama a un controlador de eventos, las declaraciones globales se vuelven a inicializar.

19.2.4 Formularios

El proceso de creación y edición de formularios en el Editor de scripts comprende estos pasos:

1. Crear un formulario nuevo: el formulario nuevo se crea, se le pone un nombre y se definen sus propiedades.
2. Diseñar el formulario: el formulario se diseña añadiéndole objetos de formulario y asignando valores a estos objetos.
3. Crear scripts de eventos de formulario: se asignan scripts a los eventos del formulario.

Crear un formulario nuevo

El proceso de creación de formularios en el Editor de scripts comprende estos pasos:

1. Crear un formulario nuevo y darle un nombre
2. Definir las propiedades del formulario

Crear un formulario nuevo y darle un nombre

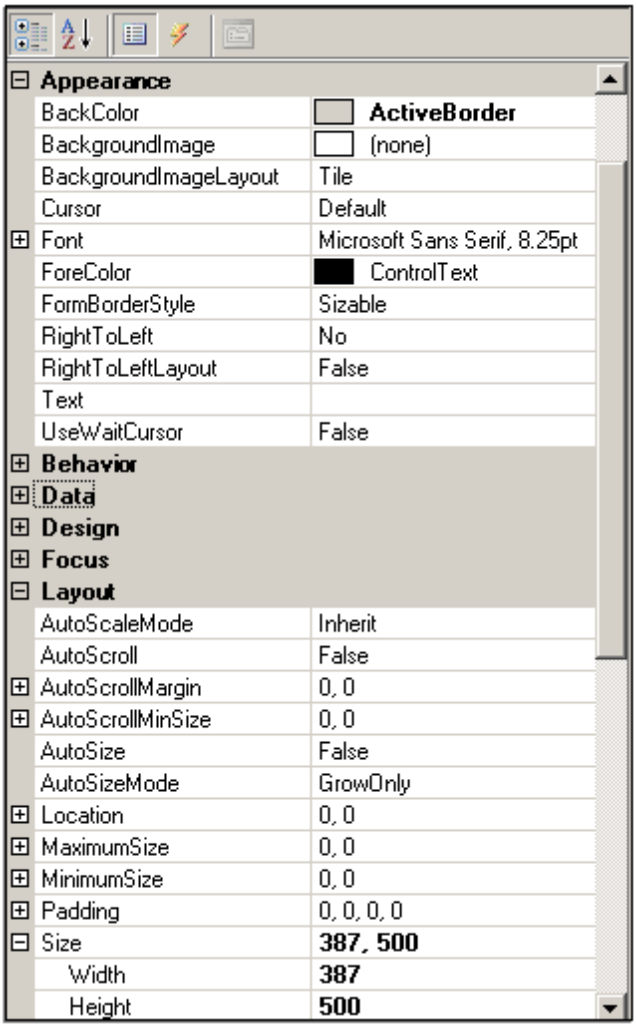
Para agregar un formulario nuevo a un proyecto de scripting, haga clic en el icono **Agregar formulario** (*imagen siguiente*) de la barra de herramientas del panel Estructura del proyecto. El formulario nuevo se crea y puede escribir el nombre del formulario.

El formulario nuevo se añade al proyecto y aparece en la ventana principal. En el panel Estructura del proyecto aparece el nuevo formulario bajo la carpeta `Formularios`. Pulse la tecla **F2** para cambiar el nombre al formulario. También puede hacer clic con el botón secundario en el formulario y después en **Cambiar de nombre** en el menú contextual. En el ejemplo de la imagen siguiente creamos un formulario nuevo llamado .

Propiedades de los formularios

Las propiedades del formulario (p. ej. su tamaño, color de fondo, fuente, etc.) se pueden definir en el panel Propiedades. En la imagen siguiente, por ejemplo, puede ver las propiedades de las categorías *Layout* (disposición) y *Appearance* (aspecto).

 © 2014 Altova GmbH



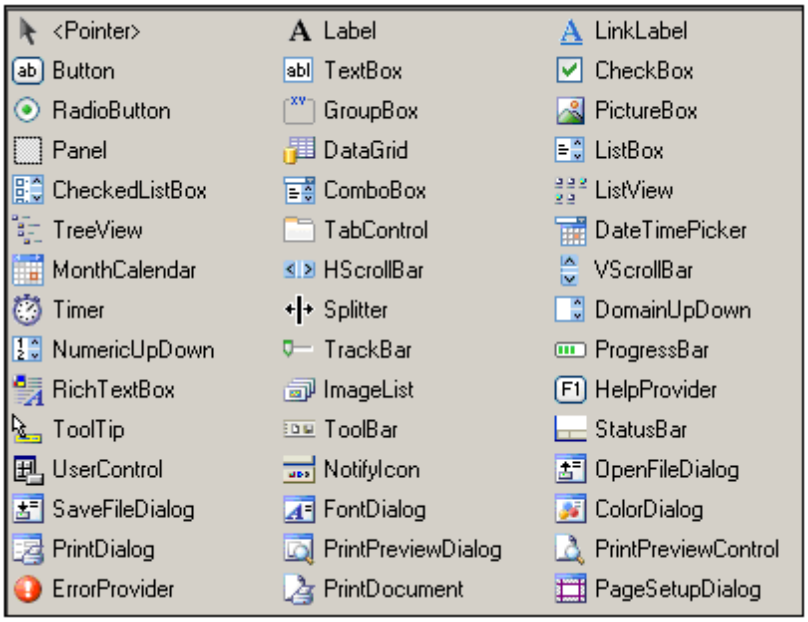
Probar un formulario
Para probar un formulario haga clic con el botón secundario en el formulario en el panel Estructura del proyecto. En el menú contextual seleccione el comando **Probar formulario**.

Diseño de formularios y objetos

El proceso de diseño de formularios en el Editor de scripts comprende estos pasos:

- Primero se colocan objetos de la paleta de objetos de formulario en el diseño del formulario.
- Después se asignan valores a las propiedades de los diferentes objetos del formulario.
- Por último se asignan scripts a los eventos del formulario.

La paleta de objetos de formulario
La paleta de objetos de formulario contiene todos los objetos que están disponibles para el diseño de formularios (*imagen siguiente*). También puede añadir controles ActiveX registrados a la paleta. Para ello haga clic con el botón secundario en la paleta y seleccione el comando **Agregar control ActiveX**.



Para insertar un objeto de la paleta en el diseño de la ventana principal, haga clic en el objeto y después en la parte del formulario donde desea insertar el objeto. El objeto se inserta en la posición elegida y en muchos casos deberá definir algunas de sus propiedades (en el panel Propiedades y eventos). Los objetos se pueden arrastrar de una parte a otra del formulario y su tamaño se puede ajustar. Además si hace clic con el botón secundario en el objeto aparece un menú contextual con varios comandos de edición (p. ej. comandos para centrar el objeto, para alinearlo, etc.).

Algunos objetos de formulario, como el temporizador `Timer`, no se añaden al formulario sino que se insertan en una bandeja situada debajo de la ventana principal del Editor de scripts. Después puede seleccionar el objeto en la bandeja y definir sus propiedades y controladores de eventos en el panel Propiedades y evento.

Estos son los objetos más utilizados:

***Label* (etiqueta):** añade campos de texto que se usan como título o descripción

***Button* (botón):** añade un botón. A estos botones puede asignar mapas de bit como imágenes de fondo.

***Check Box* (casilla):** añade una casilla de verificación que permite realizar selecciones de tipo `Sí/No`.

***Combo Box* (cuadro combinado):** añade un cuadro combinado, que permite al usuario seleccionar una opción de un menú desplegable.

***List Box* (cuadro de lista):** añade un cuadro de lista, que enumera todos los elementos que se pueden seleccionar.

***TextBox* (cuadro de texto):** permite al usuario escribir una sola línea de texto.

 © 2014 Altova GmbH

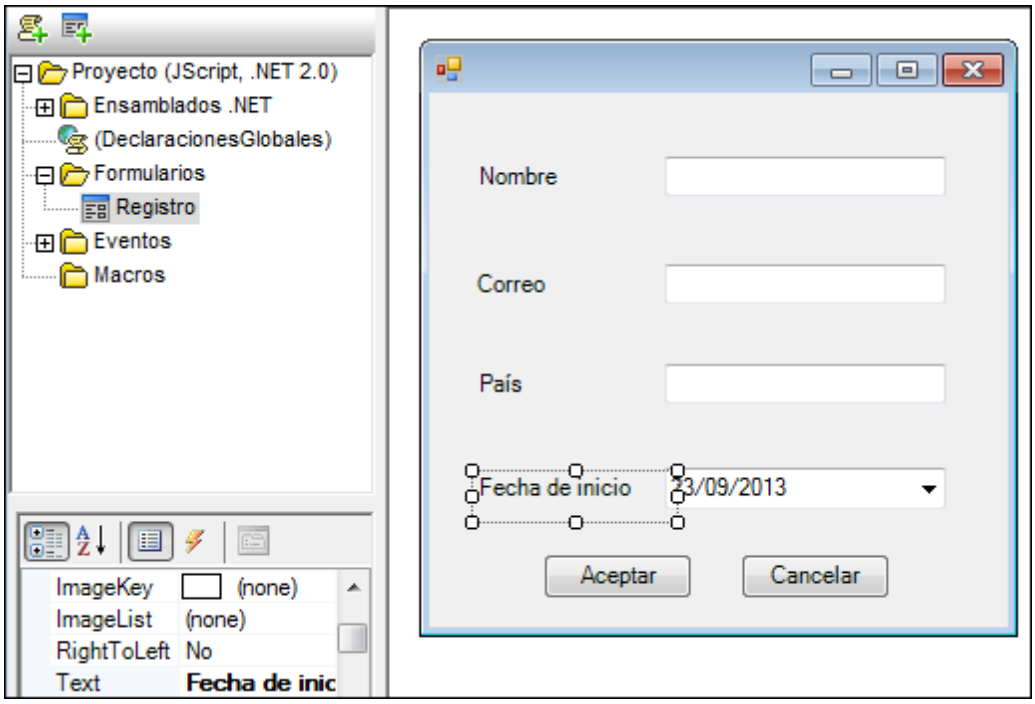
***Rich TextBox* (cuadro de texto enriquecido):** permite al usuario escribir varias líneas de texto.

Crear objetos y configurar sus propiedades
Para crear un objeto en el formulario primero debe seleccionar el objeto en la paleta de objetos y después debe hacer clic en la parte del formulario donde desea insertar el objeto. Una vez insertado, el objeto puede arrastrarse a otra posición del formulario o puede ajustar su tamaño.

Cuando se selecciona un objeto del formulario sus propiedades aparecen en el panel Propiedades y eventos, donde las puede configurar. Puede hacer clic en el icono **Propiedades** de la barra de herramientas de este panel para ver una lista de propiedades del objeto.

Por ejemplo, en la imagen siguiente seleccionamos el objeto **Etiqueta** que tiene el texto . En el panel Propiedades y eventos el nombre del objeto (el que se usa para identificar el objeto en el código: `Etiqueta1`) se define en la categoría de propiedades *Design*. En este caso el objeto se llama `Etiqueta1`.

El texto de la etiqueta (que es lo que aparece en el formulario) debe introducirse como valor de la propiedad *Texto* de la categoría de propiedades *Appearance*.

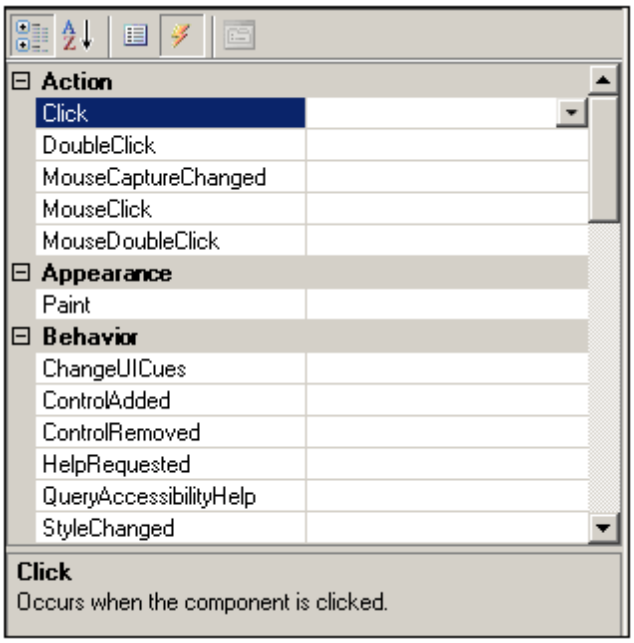


Para asignar otras propiedades al objeto, introduzca el valor correspondiente en las propiedades del panel Propiedades y eventos.

Probar un formulario
Para probar un formulario haga clic con el botón secundario en el formulario en el panel Estructura del proyecto. En el menú contextual seleccione el comando **Probar formulario**.

Eventos de formularios

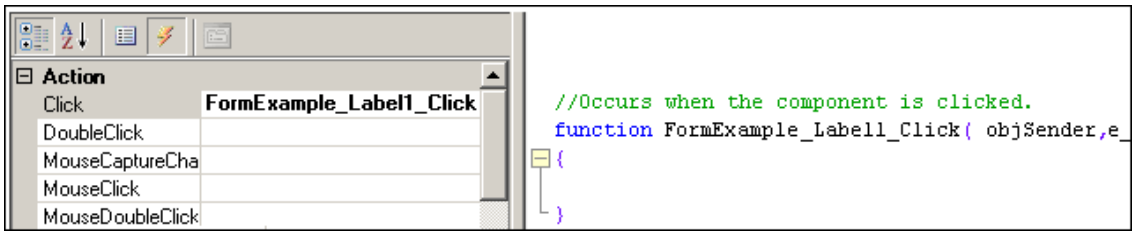
Cuando está seleccionado un objeto en el diseño, haga clic en el icono **Eventos** de la barra de herramientas del panel Propiedades y eventos (cuarto icono empezando por la izquierda) para ver todos los eventos que están disponibles para el objeto (*imagen siguiente*). Los eventos pueden aparecer ordenados por categoría (*imagen siguiente*) o alfabéticamente.



Por cada evento puede introducir el nombre de un controlador de eventos o función ya existente. También tiene estas dos opciones:

- Puede hacer doble clic en un evento para crear (i) un script de función vacío en la pestaña *Source* de la ventana principal y (ii) una asociación entre la función recién creada y el evento seleccionado.
- Puede hacer doble clic en un botón de la pestaña *Design* para generar código auxiliar vacío para el evento en la pestaña *Source*.

La captura de pantalla siguiente, por ejemplo, se tomó después de hacer doble clic en el evento **`Click`**. Observe que en la ventana principal se creó una función de controlador de eventos vacía llamada `FormExample_Label1_Click` y que en el panel Propiedades y eventos esta función se asoció al evento **`Click`**.



Ahora podemos insertar el código necesario y guardar el proyecto.

Escribir scripts

Cuando termine de diseñar el formulario de forma visual, los objetos del formulario estarán

 © 2014 Altova GmbH

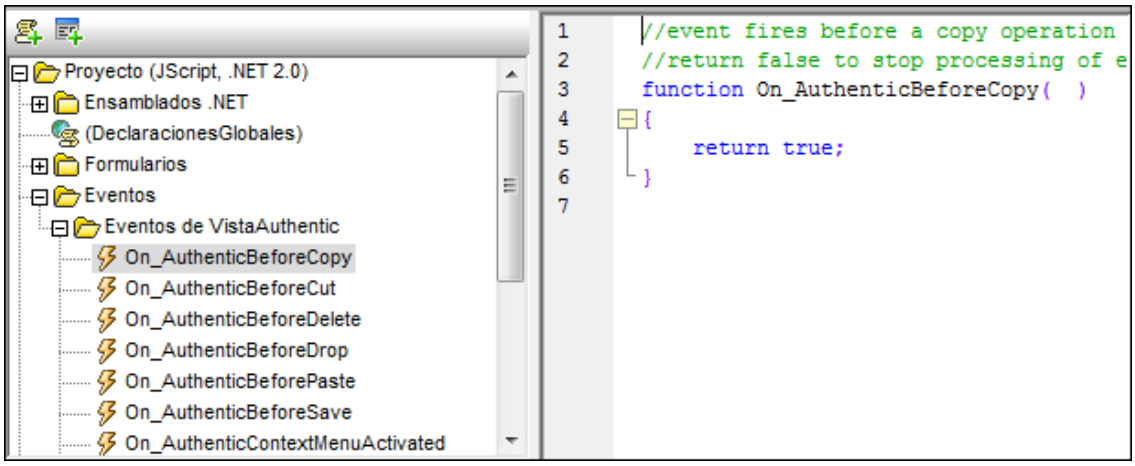
asociados a los correspondientes scripts. El ejemplo de script que aparece a continuación añade colores cuando se hace clic en un botón. El script se inserta como controlador de eventos para el evento `Click` del botón `Botón1` (el evento está disponible en el panel Propiedades y eventos cuando se selecciona el botón en el diseño):

```
function FormExample_Botón1_Click( objSender, e_EventArgs )
{
       // Establece el color de primer plano (rojo) del botón
       objSender.ForeColor = CLR.Static( "System.Drawing.Color" ).Red;
       // Establece el color de fondo (azul) del botón
       objSender.BackColor = CLR.Static( "System.Drawing.Color" ).Blue;
       // Establece el color de fondo (verde) del formulario
       objSender.FindForm().BackColor =
CLR.Static( "System.Drawing.Color" ).Green;
}
```

19.2.5 Eventos

El componente **Eventos** del proyecto de scripting (*imagen siguiente*) contiene

Para acceder al script de control de cualquiera de estos eventos, haga clic con el botón secundario en el evento y seleccione el comando **Abrir** del menú contextual. El script aparece en la ventana principal (*imagen siguiente*), donde puede editarlo. Cuando termine de editar el script guarde los cambios haciendo clic en el icono **Guardar** de la barra de herramientas del Editor de scripts.



Además debe tener en cuenta que:

- El nombre del evento debe estar correctamente escrito en el encabezado de función. De lo contrario no se podrá llamar al controlador de eventos.
- Dentro de las macros y de los controladores de eventos se pueden definir variables locales y funciones auxiliares.

 © 2014 Altova GmbH

19.2.6 Macros

Las macros sirven para automatizar tareas repetitivas o complejas. En el entorno de scripting puede crear un script que llame a funciones de la aplicación y a otras funciones personalizadas definidas por el usuario. Es decir, las macros ofrecen un potente método de automatizar tareas dentro de UModel. Esta sección se divide en varios apartados:

- Crear y editar una macro: instrucciones para crear una macro nueva y editar macros ya existentes.
- Depurar una macro: instrucciones para depurar una macro.

Nota importante
Es muy importante tener en cuenta estos puntos:

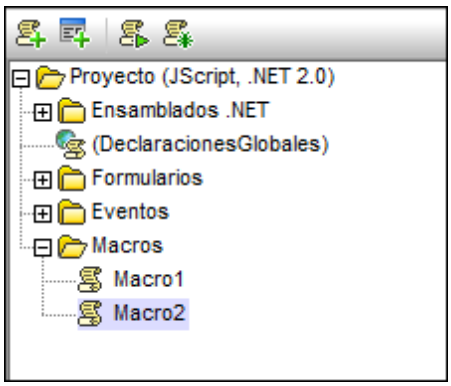
- En el proyecto de scripting activo puede incluir tantas macros como quiera. Todas ellas se guardan en el archivo de proyecto de scripting de Altova (archivo `.asprj`).
- Las funciones que se usan en una macro se pueden guardar como declaración global. Todas las declaraciones globales se guardan en el archivo de proyecto de scripting de Altova (archivo `.asprj`).
- Puede probar las macros ejecutándolas dentro del Editor de scripts. En el editor también puede depurar las macros.
- UModel puede tener un proyecto de scripting global, un proyecto de scripting adicional asignado al proyecto activo y todas las macros que estén disponibles en ambos proyectos. Consulte el apartado para obtener más información.

Crear y editar una macro

A continuación describimos las operaciones necesarias para crear una macro nueva y editar macros ya disponibles.

Crear una macro nueva
Haga clic con el botón secundario en la carpeta Macros del panel Estructura del proyecto de scripting y seleccione **Agregar macro** en el menú contextual. En realidad este comando siempre está disponible en el menú contextual que aparece al hacer clic con el botón secundario en cualquier elemento del panel. Otra opción es hacer clic en el icono **Macro nueva** de la barra de herramientas del panel Estructura del proyecto de scripting.

El documento de macro recién creado (y vacío) aparece en la ventana principal del editor y el nombre de la macro aparece en la barra de título (*imagen siguiente*).



Poner un nombre a la macro o cambiarle el nombre
Para dar un nombre a una macro o cambiar su nombre, haga clic en el nombre de la macro en el panel Estructura del proyecto de scripting y pulse la tecla **F2**. También puede hacer clic con el botón secundario en la macro y seleccionar **Cambiar el nombre de la macro** en el menú contextual.

Abrir una macro
Para abrir una macro haga clic con el botón secundario en la macro en el panel Estructura del proyecto (*imagen anterior*) y seleccione **Abrir** en el menú contextual. La macro se abre en la ventana principal y su nombre aparece en la barra de título del Editor de scripts (*imagen siguiente*). También puede abrir la macro haciendo doble clic en el nombre de la macro en el panel Estructura del proyecto.

Editar la macro
Para editar una macro escriba o edite su código en la ventana principal del editor.

Nota: en las macros no se pueden usar parámetros ni valores devueltos.

Ejecutar una macro

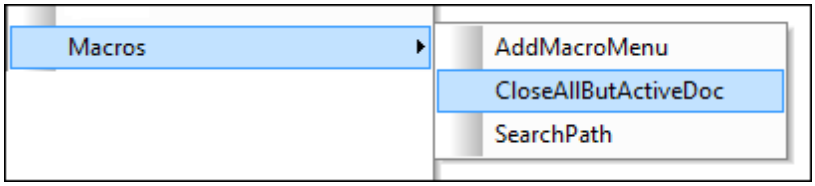
Para ejecutar una macro en el Editor de scripts haga clic con el botón secundario en la macro (en el panel Estructura del proyecto de scripting) y seleccione el comando **Ejecutar macro**.

Hay varias maneras de ejecutar una macro desde UModel:

- Desde el menú Herramientas | Macros de UModel.
- Creando y usando un botón de la barra de herramientas para la macro.

Recuerde que no se puede ejecutar más de una macro a la vez. Cuando se termina de ejecutar la macro (o el evento), el script se cierra y las variables globales pierden sus valores.

El comando de UModel para ejecutar macros
El comando de menú **Herramientas | Macros** (*imagen siguiente*) abre un submenú que enumera todas las macros definidas en el proyecto de scripting que está activo en ese momento en UModel. El proyecto de scripting activo es el indicado en la pestaña "Script" del cuadro de diálogo "Opciones" o en el cuadro de diálogo "Scripts".



Ahora seleccione la macro del submenú que desea ejecutar.

Icono de la barra de herramientas
Puede crear un icono de barra de herramientas o un comando de menú que ejecute la macro elegida. Para ello empiece por hacer clic en **Herramientas | Personalizar**. Esto abre el cuadro

 © 2014 Altova GmbH

de diálogo "Personalizar". Haga clic en la pestaña *Macros* (*imagen siguiente*).

Ahora siga estas instrucciones:

1. En la pestaña "Macros" seleccione la macro pertinente en el panel *Macros*. Este panel muestra todas las macros que están disponibles en el proyecto de scripting activo.
2. En el campo de entrada *Mostrar texto* escriba el nombre del icono. Este nombre aparecerá cuando el usuario pase el puntero del mouse sobre el icono.
3. Ahora haga clic en el botón **Agregar comando** para añadir el comando a la lista del panel *Comandos asociados*.
4. Seleccione el comando y haga clic en el botón **Editar icono** para crear un icono nuevo.
5. Arrastre el icono recién creado desde la lista del panel *Comandos asociados* hasta la barra de herramientas o el menú donde desea colocarlo. Cuando el cursor tenga forma de línea o de **I**, suelte el icono.
6. Si quiere, también puede asignar un acceso directo a la macro desde la pestaña *Teclado* del cuadro de diálogo "Personalizar".

Para eliminar el icono de la barra de herramientas o el comando del menú, abra la pestaña *Macros* del cuadro de diálogo "Personalizar" y arrastre el icono desde la barra de herramientas o desde el menú hasta el panel *Comandos asociados*. Ahora seleccione el comando en el panel *Comandos asociados* y haga clic en el botón **Quitar**.

Depurar una macro

Las macros se pueden depurar usando el depurador que tenga instalado en el equipo.

Esto abre el cuadro de diálogo "Depurador Just-In-Time" (*imagen siguiente*), que enumera todos los depuradores que están instalados en el equipo. Seleccione uno de ellos y haga clic en el botón **Sí**.

Entonces se inicia el depurador seleccionado.

19.2.7 Notas sobre programación

Es necesario tener en cuenta todos estos puntos:

- En Microsoft .NET Framework se puede acceder por defecto a todos los espacios de nombres y tipos de estos ensamblados .NET:

```
System
System.Data
System.Design
System.Drawing
System.Windows.Forms
System.XML
```

  Al proyecto de scripting se pueden añadir más ensamblados con ayuda del menú contextual del panel Estructura del proyecto del Editor de scripts o en el código fuente de forma dinámica (en tiempo de ejecución) usando `CLR.LoadAssembly`.

- Los parámetros out de los métodos de .NET Framework exigen variables especiales en JScript. Ejemplos:

```
var dictionary =
CLR.Create( "System.Collections.Generic.Dictionary< System.String,
System.String >" );
dictionary.Add("1", "A");
dictionary.Add("2", "B");

// usar el método JScript para acceder a los parámetros out
var strOut = new Array(1);
if ( dictionary.TryGetValue("1", strOut) ) // TryGetValue
establecerá el parámetro out
        alert( strOut[0] ); // usar el parámetro out
```

- Los métodos de .NET que exigen argumentos enteros no se deberían invocar directamente con objetos Number de JScript que sean valores de punto flotante.

  Por ejemplo, en lugar de

```
var objCustomColor = CLR.Static( "System.Drawing.Color" ). FromArgb( 128,
128, 128 );
```

  recomendamos usar

```
var objCustomColor = CLR.Static( "System.Drawing.Color" ).
FromArgb( Math.floor( 128 ), Math.floor( 128 ), Math.floor( 128 ) );
```

- Para iterar en colecciones .NET puede usar el objeto Enumerator de JScript y el iterador .NET. Ejemplo:

```
// iterar usando el iterador de JScript
var itr = new Enumerator( coll );
for ( ; !itr.atEnd(); itr.moveNext() )
        alert( itr.item() );
```

 © 2014 Altova GmbH

```
// iterar usando el iterador .NET
var itrNET = coll.GetEnumerator();
while( itrNET.MoveNext() )
       alert( itrNET.Current );
```

- Puede crear instancias de las plantillas .NET de esta forma:

```
var coll =
CLR.Create( "System.Collections.Generic.List<System.String>" );
```

o

```
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String,
String > >" );
```

- Puede acceder a los valores de .NET Enum de esta forma:

```
       var enumValStretch =
CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch;
```

- Los literales de enumeración, tal y como se definen en las bibliotecas de tipos de Altova, ahora se pueden usar en lugar de valores numéricos.

```
       objExportXMIFileDlg.XMIType = eXMI21ForUML23;
```

Comandos integrados

En este apartado se explican:

- Comandos integrados
  alert
  conform
  doevents
  CreateForm
  lastform
  prompt
  ShowForm
  watchdog

- Comandos de interoperabilidad de .NET
  CLR.Create
  CLR.Import
  CLR.LoadAssembly
  CLR.ShowImports
  CLR.ShowLoadedAssemblies
  CLR.Static

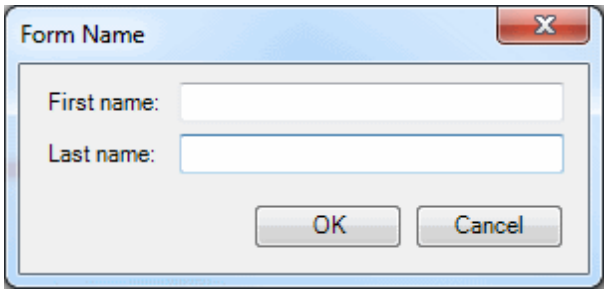
Comandos integrados
Estos son los comandos integrados disponibles:

ShowForm(strFormName : cadenaDeTexto)
Crea una instancia de un objeto FormularioNuevo a partir del nombre de formulario dado y lo muestra inmediatamente en forma de cuadro de diálogo.
Valor devuelto: un número que representa el DialogResult generado (`System.Windows.Forms.DialogResult`).

Ejemplo:

```
var dialogResult = ShowForm( "FormName" );
```

Muestra el formulario "FormName" como cuadro de diálogo:



DialogResult se puede evaluar con:

```
if ( dialogResult ==
CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
        alert( "ok" );
else
        alert( "cancel" );
```

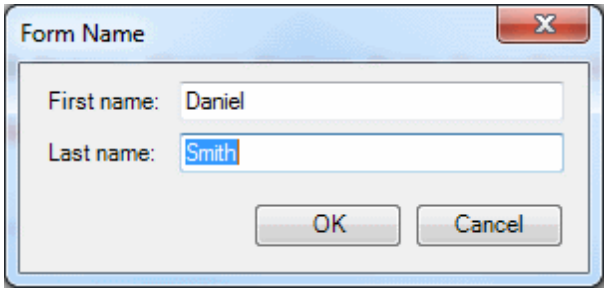
CreateForm(strFormName : cadenaDeTexto)
Crea una instancia de un objeto FormularioNuevo a partir del nombre de formulario dado.
Valor devuelto: el objeto Form (`System.Windows.Forms.Form`) con el nombre dado o `null` si existe ningún objeto Form con ese nombre.

Ejemplo:

```
var myForm = CreateForm( "FormName" );
if ( myForm != null )
{
        myForm.textboxFirstName.Text = "Daniel";
        myForm.textboxLastName.Text = "Smith";
        var dialogResult = myForm.ShowDialog();
}
```

Muestra el formulario "FormName" como cuadro de diálogo. Los cuadros de texto ya están inicializados:

 © 2014 Altova GmbH



DialogResult se puede evaluar con:

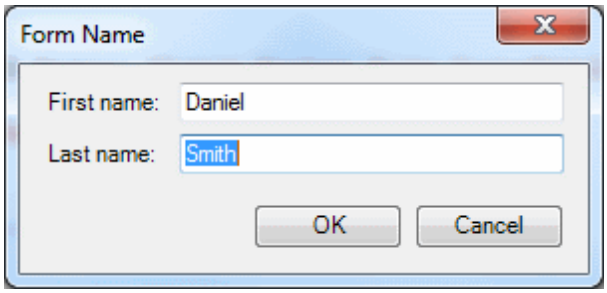
```
if ( dialogResult ==
CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
        alert( "ok" );
else
        alert( "cancel" );
```

lastform

Este campo global se puede usar para acceder fácilmente al último objeto Form que se creó.
Valor devuelto: devuelve una referencia al último objeto Form (`System.Windows.Forms.Form`) que se creó mediante `CreateForm()` o `ShowForm()`.

```
Ejemplo:
        CreateForm( "FormName" );
        if ( lastform != null )
        {
                lastform.textboxFirstName.Text = "Daniel";
                lastform.textboxLastName.Text = "Smith";
                var dialogResult = lastform.ShowDialog();
        }
```

Muestra el formulario "FormName" como cuadro de diálogo. Los cuadros de texto ya están inicializados (como en el ejemplo CreateForm anterior):



doevents()

Procesa todos los mensajes de Windows que estén en la cola de mensajes.
Valor devuelto: Ninguno

```
Ejemplo:
        for ( i=0; i < nLongLastingProcess; ++i )
        {
```

```
        // realizar procesos de larga duración

        doevents(); // procesar mensajes de Windows; dar a la interfaz del
usuario la oportunidad de actualizarse
}
```

watchdog(bEnable : booleano)
Los scripts de larga ejecución que consumen mucha CPU hacen que el guardián pregunte al usuario si el script se debe interrumpir. El método guardián `watchdog()` se utiliza para deshabilitar o habilitar este comportamiento. El guardián está habilitado por defecto.
Valor devuelto: Ninguno

```
Ejemplo:
        watchdog( false );   // deshabilitar guardián. Sabemos que la siguiente
        instrucción consume mucha CPU pero finalizará con toda seguridad
        doCPUIntensiveScript();
        watchdog( true );    // volver a habilitar el guardián
```

Consejo práctico:
También puede restaurar el guardián llamando a `watchdog(true)`. Esto puede ser muy práctico antes de ejecutar tareas de larga ejecución (que consumen mucha CPU) para garantizar que tienen asignada la cuota máxima de procesamiento permitida.

alert(strMessage : cadenaDeTexto) o MsgBox(strMessage : cadenaDeTexto)
Se usa un cuadro de alerta para mostrar un mensaje dado. El usuario debe hacer clic en **Aceptar** para continuar.
Valor devuelto: Ninguno

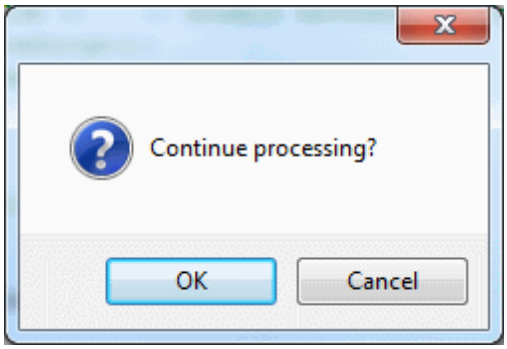
```
Ejemplo:
        alert( "Hello World" );
```

confirm(strMessage : cadenaDeTexto)
Abre un cuadro de diálogo que muestra el mensaje de confirmación dado. Los cuadros de confirmación se usan a menudo para verificar o aceptar algo. El usuario debe hacer clic en **Aceptar** o **Cancelar** para continuar.
Valor devuelto: un booleano que representa la respuesta del usuario. Si el usuario hace clic en **Aceptar**, el cuadro de diálogo devuelve `true`. Si hace clic en **Cancelar**, el cuadro de diálogo devuelve `false`.

 © 2014 Altova GmbH

Ejemplo:

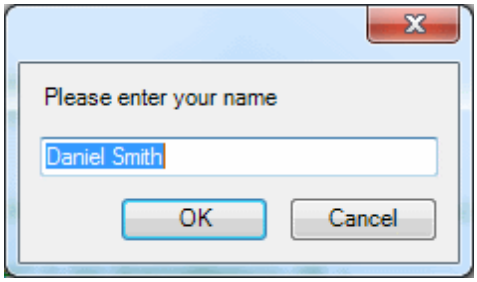
```
if ( confirm( "Continue processing?" ) == false )
        return;
```



prompt(strMessage : cadenaDeTexto, strDefault : cadenaDeTexto)
Abre un cuadro de diálogo que muestra el mensaje de solicitud dado y un cuadro de texto con una respuesta predeterminada. Los cuadros de solicitud suelen usarse para introducir un valor de cadena sencillo.
Valor devuelto: una cadena que contiene el valor del cuadro de texto o `null` si el usuario hizo clic en **Cancelar**.

Ejemplo:

```
var name = prompt( "Please enter your name", "Daniel Smith" );
if ( name != null )
        alert( "Hello " + name + "!" );
```



Comandos de interoperabilidad de .NET
Bajo CLR se ofrecen algunas funciones .NET más para permitir una mayor interoperabilidad.

CLR.Import(strNamespaceCLR : cadenaDeTexto)
Este es el equivalente a la palabra clave *using* de C# e *imports* de VB.Net. Permite obviar los espacios de nombres dados en las llamadas sucesivas, como en `CLR.Create()` y `CLR.Static()`.
Valor devuelto: ninguno

Ejemplo:

En lugar de tener que usar siempre nombres completos:

```
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 );
}
```

se pueden importar los espacios de nombres y usar la forma abreviada:

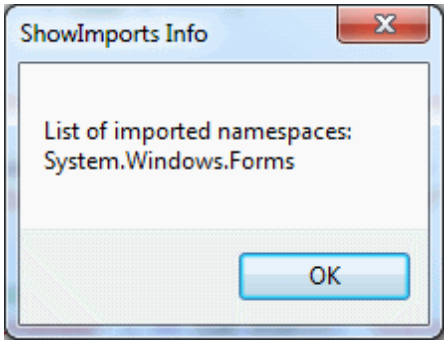
```
CLR.Import( "System.Windows.Forms" );
if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
        var sName = lastform.textboxFirstName.Text + " " +
lastform.textboxLastName.Text;
        CLR.Static( "MessageBox" ).Show( "Hello " + sName );
}
```

Nota importante:
Al importar un espacio de nombres no se añade ni se carga el ensamblado correspondiente al proyecto de scripting.
Los ensamblados se pueden agregar al proyecto de scripting de forma dinámica (en tiempo de ejecución) en el código fuente con ayuda de `CLR.LoadAssembly`.

`CLR.ShowImports()`
Abre un cuadro de diálogo MessageBox que muestra los espacios de nombres importados actualmente. El usuario debe hacer clic en **Aceptar** para continuar.
Valor devuelto: Ninguno

Ejemplo:

CLR.ShowImports();



`CLR.LoadAssembly(strAssemblyNameCLR : cadenaDeTexto)`
Carga el ensamblado .NET con el nombre de ensamblado dado o con la ruta de acceso de archivo dada.
Valor devuelto: un valor booleano. `True` si se cargó el ensamblado. `False` si no se cargó.

Ejemplo:

```
// establecer el texto del portapapeles (si es posible)
// System.Windows.Clipboard es parte del ensamblado PresentationCore, así
que queremos cargar este ensamblado primero:
```

 © 2014 Altova GmbH

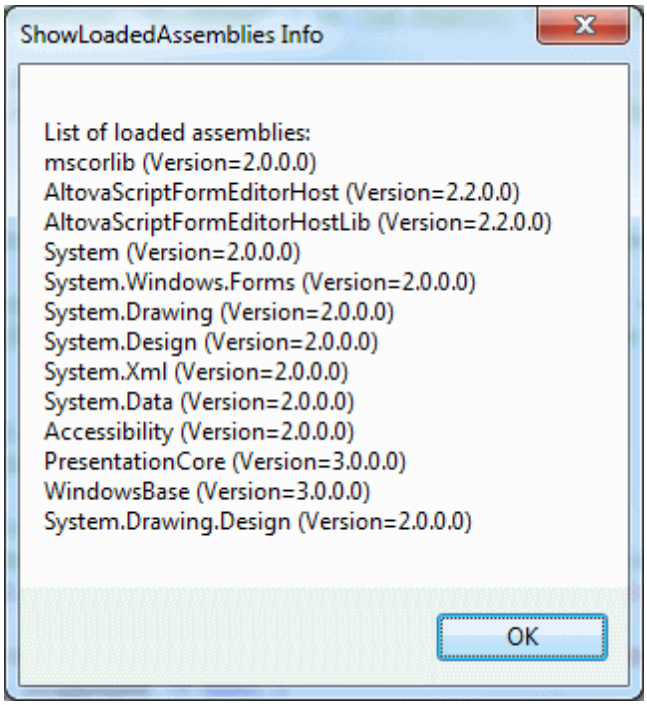
```
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35", true ) )
{
        var clipboard = CLR.Static( "System.Windows.Clipboard" );
        if ( clipboard != null )
                clipboard.SetText( "HelloClipboard" );
}
```

CLR.ShowLoadedAssemblies()

Abre un cuadro de diálogo MessageBox que enumera los ensamblados que están cargados actualmente. El usuario debe hacer clic en **Aceptar** para continuar.
Valor devuelto: Ninguno

Ejemplo:

```
CLR.ShowLoadedAssemblies();
```



CLR.Create(strTypeNameCLR : cadenaDeTexto, argumentos del constructor ...)

Crea una nueva instancia de objeto .NET para el typeName dado. Si se pasa más de un argumento los argumentos sucesivos se interpretan como argumentos para el constructor del objeto .NET.
Valor devuelto: una referencia al objeto .NET creado

Ejemplos:

```
var objArray = CLR.Create("System.Collections.ArrayList");

var newItem = CLR.Create( "System.Windows.Forms.ListViewItem",
"NewItemText" );

var coll =
```

```
CLR.Create( "System.Collections.Generic.List<System.String>" );

CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String,
String > >" );
```

CLR.Static(strTypeNameCLR : cadenaDeTexto)
Da acceso a los tipos de .NET que no tienen instancias y que solo contienen miembros estáticos.
Valor devuelto: una referencia al objeto .NET estático

```
Ejemplo:
       var enumValStretch =
       CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch

       var clipboard = CLR.Static( "System.Windows.Clipboard" );
       clipboard.SetText( "HelloClipboard" );

       if ( ShowForm( "FormName" ) ==
       CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
              alert( "ok" );
       else
              alert( "cancel" );
```

Uso de los formularios y comandos

Este es el uso de los formularios:

Con los objetos Form se puede acceder a la estructura de componentes mediante acceso de campo:

Por ejemplo, imagine que hay designado este objeto Form:

```
MiFormulario
    ButtonPanel
        OkButton
        CancelButton
    TextEditor
    AxMediaPlayer1

TrayComponents:
    MiTemporizador
```

Desde el script se puede crear una instancia del objeto Form:

```
var objForm = CreateForm("MiFormulario");
```

Para acceder a uno de sus componentes puede usar el acceso de campo:

```
objForm.ButtonPanel.OkButton.Enabled = false;
```

o

```
objForm.TextEditor.Text = "Hola mundo";
```

 © 2014 Altova GmbH

Para acceder a Tray Components use este método en el objeto Form:

```
var objTrayComponent = <A form object>.GetTrayComponent(strComponentName :
String);
```

Para obtener una referencia al componente Timer y habilitarlo puede utilizar:

```
var objTimer = objForm.GetTrayComponent("MiTemporizador");
objTimer.Enabled = true;
```

Para los controles ActiveX puede acceder al objeto COM subyacente mediante la propiedad OCX:

```
var ocx = lastform.AxMediaPlayer1.OCX; // obtener objeto COM subyacente
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";
```

19.2.8 Ejemplo de uso (script de muestra de UModel)

Este ejemplo está en la carpeta `...\UModelExamples`. Se trata del archivo **ScriptSampleFind.asprj.**

Declaraciones globales:

```
// importar el espacio de nombres System.Windows.Forms para todas las macros,
todos los formularios y todos los eventos:
CLR.Import( "System.Windows.Forms" );
```

Formulario FindSample

```
//aparece cada vez que el usuario carga el formulario.
function FindKinds_Load( objSender, e_EventArgs )
{
	// inicializar todos los controles:
	objSender.btnFind.Enabled = false;

	objSender.gbOptions.cbMatchCase.Checked = bMatchCase;
	objSender.gbOptions.cbWholeName.Checked = bMatchWholeName;

	objSender.gbSearchKinds.cbClasses.Checked 		=
bSearchClasses;
	objSender.gbSearchKinds.cbInterfaces.Checked 		=
bSearchInterfaces;
	objSender.gbSearchKinds.cbEnumerations.Checked 		=
bSearchEnumerations;
	objSender.gbSearchKinds.cbPackages.Checked 		=
bSearchPackages;
	objSender.gbSearchKinds.cbProperties.Checked 		=
bSearchProperties;
	objSender.gbSearchKinds.cbOperations.Checked 		=
bSearchOperations;
	objSender.gbSearchKinds.cbEnumerationLiterals.Checked 	=
bSearchEnumLiterals;
	objSender.gbSearchKinds.cbOthers.Checked 		=
bSearchOthers;
}

//Evento generado cuando el valor de la propiedad Text se cambia en Control.
function FindKinds_tbFindText_TextChanged( objSender, e_EventArgs )
{
	// objSender es la casilla de texto "Buscar"; además, obtener el
formulario:
	var objForm = objSender.FindForm();

	// oara habilitar el botón Buscar, searchstring debería tener como mínimo
un carácter
	objForm.btnFind.Enabled = ( objSender.Text.length > 0 );
}

//Aparece cuando se hace clic en el componente
```

 © 2014 Altova GmbH

```
function FindType_btnFind_Click( objSender, e_EventArgs )
{
	// objSender es el botón "Busar"; además, obtener el formulario:
	var objForm = objSender.FindForm();

	// guardar opciones elegidas en las variables de macro globales:
	bMatchCase		= objForm.gbOptions.cbMatchCase.Checked;
	bMatchWholeName		= objForm.gbOptions.cbWholeName.Checked;

	bSearchClasses		= objForm.gbSearchKinds.cbClasses.Checked;
	bSearchInterfaces	= objForm.gbSearchKinds.cbInterfaces.Checked;
	bSearchEnumerations	= objForm.gbSearchKinds.cbEnumerations.Checked;
	bSearchPackages		= objForm.gbSearchKinds.cbPackages.Checked;
	bSearchProperties	= objForm.gbSearchKinds.cbProperties.Checked;
	bSearchOperations	= objForm.gbSearchKinds.cbOperations.Checked;
	bSearchEnumLiterals	=
objForm.gbSearchKinds.cbEnumerationLiterals.Checked;
	bSearchOthers		= objForm.gbSearchKinds.cbOthers.Checked;

	var sFindText = objForm.tbFindText.Text;

	if ( !bMatchCase )
		sFindText = sFindText.toUpperCase();

	// mostrar el cursor de espera durante la búsqueda; cargar clase estática
"Cursors":
	var objCursors = CLR.Static( "Cursors" );
	objForm.Cursor = objCursors.WaitCursor;
	// iniciar actualización de ListView:
	objForm.lvResults.BeginUpdate();
	// restaurar Items:
	objForm.lvResults.Items.Clear();
	// iniciar la búsqueda con el paquete raíz:
	Find( Application.ActiveDocument.RootPackage, sFindText );
	// ListView se actualizó:
	objForm.lvResults.EndUpdate();
	// restaurar el cursor predeterminado:
	objForm.Cursor = objCursors.Default;

	// informar al usuario cuando no se encuentre nada:
	if ( objForm.lvResults.Items.Count == 0 )
		alert( "No se encontró nada." );
}

//Aparece cuando cambia la propiedad 'SelectedIndex' para esta ListView.
function FindKinds_lvResults_SelectedIndexChanged( objSender, e_EventArgs )
{
	// resaltar el elemento UML seleccionado en la Estructura del esquema:
	if ( objSender.SelectedItems.Count == 1 )
	{
		// obtener el nombre QName del elemento seleccionado:
		var selItem = objSender.SelectedItems.Item( 0 );
		var sQualifiedName = selItem.SubItems.Item( 2 ).Text;
```

```
            var objDocument = Application.ActiveDocument;
            // buscar el elemento UML con ayuda del nombre QName:
            var objUMLElement =
objDocument.RootPackage.FindOwnedMemberWithQualifiedName( sQualifiedName );
            // resaltar el elemento UML en la Estructura del esquema:
            objDocument.FocusUMLDataInModelTree( objUMLElement, false, true );
        }
}
```

Macro Find Sample:

```
// ##########################################
// Macro 'Find Sample'
// Buscar elementos del modelo UML por nombre y tipo
// ##########################################

// variables globales de la macro
var bMatchCase              = false;
var bMatchWholeName = false;

var bSearchClasses          = true;
var bSearchInterfaces       = true;
var bSearchEnumerations     = true;
var bSearchPackages         = true;
var bSearchProperties       = true;
var bSearchOperations       = true;
var bSearchEnumLiterals     = true;
var bSearchOthers           = false;

// comprobar si el tipo de elemento UML debe tenerse en cuenta
function MatchKind( objUMLElement )
{
        if ( objUMLElement.KindName == "Class"                ) return
bSearchClasses;
        if ( objUMLElement.KindName == "Interface"            ) return
bSearchInterfaces;
        if ( objUMLElement.KindName == "Enumeration"          ) return
bSearchEnumerations;
        if ( objUMLElement.KindName == "Package"              ) return
bSearchPackages;
        if ( objUMLElement.KindName == "Property"             ) return
bSearchProperties;
        if ( objUMLElement.KindName == "Operation"            ) return
bSearchOperations;
        if ( objUMLElement.KindName == "EnumerationLiteral"   ) return
bSearchEnumLiterals;
        return bSearchOthers;
}

// comprobar si el nombre del elemento UML coincide con las opciones de búsqueda
function MatchName( objUMLNamedElement, sFindText )
{
        var sName = objUMLNamedElement.Name;
        if ( !bMatchCase )
```

 © 2014 Altova GmbH

```
                sName = sName.toUpperCase();

        if ( bMatchWholeName )
                return ( sName == sFindText )
        else
                return ( sName.indexOf( sFindText ) != -1 )
}

// buscar recursivamente todos los elementos
function Find( objUMLElement, sFindText )
{
        // solo nos interesan los elementos que tengan nombre:
        if ( objUMLElement.IsKindOf( "NamedElement" ) )
        {
                var objUMLNamedElement = objUMLElement;

                // ver solamente los elementos cuyo nombre y tipo coincida:
                if ( MatchKind( objUMLNamedElement ) &&
                     MatchName( objUMLNamedElement, sFindText ) )
                {
                        // crear una clase ListViewItem nueva
                        var newItem = CLR.Create( "ListViewItem",
objUMLNamedElement.KindName );
                        // y agregarla a los elementos listview:
                        lastForm.lvResults.Items.Add( newItem );
                        // anexar el nombre y el QName en subelementos:
                        newItem.SubItems.Add( objUMLNamedElement.Name );
                        newItem.SubItems.Add( objUMLNamedElement.QualifiedName );
                }
        }

        // iterar en todos los secundarios (ownedElements):
        var ownedElements = objUMLElement.OwnedElements;
        var itr = new Enumerator( ownedElements );
        for ( ; !itr.atEnd(); itr.moveNext() )
                Find( itr.item(), sFindText );
}

// crear y ver el formulario:
ShowForm( "FindSample" );
```

19.3 Crear complementos en entornos IDE

Si quiere puede crear sus propios complementos e integrarlos en UModel.

Estos complementos pueden utilizarse para:

- Configurar UModel con nuevos comandos, menús, iconos, botones, etc.
- Reaccionar ante determinados eventos desde UModel.
- Ejecutar código dentro de UModel con total acceso a la API de la aplicación.
- Integrar controles ActiveX propios en UModel.

UModel espera que el complemento implemente la interfaz IUModelPlugIn y en este ejemplo puede ver una implementación básica de esta interfaz. Además en la carpeta `UModelExamples\UModelAPI` encontrará otros ejemplos que usan C#.

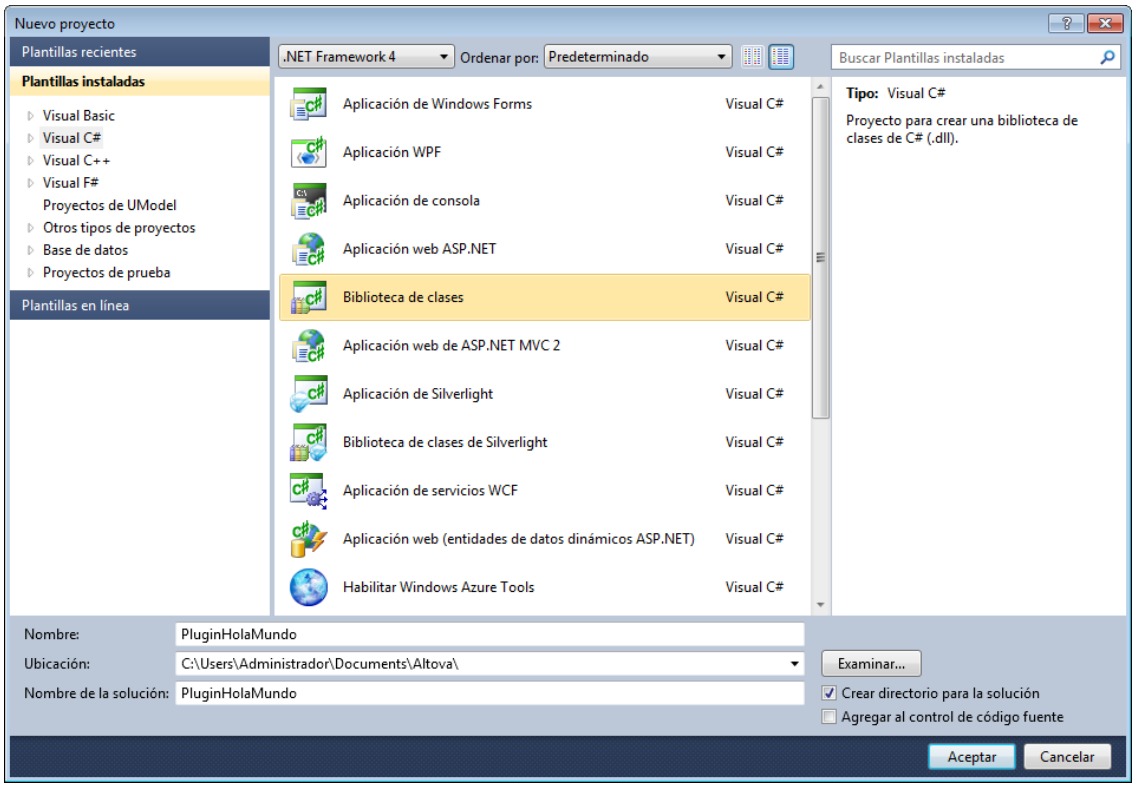
© 2014 Altova GmbH

19.3.1 Cómo crear un complemento en MS Visual Studio .NET

Este apartado explica cómo usar C# para crear un DLL para UModel.

Tras instalar UModel y Microsoft Visual Studio .NET:

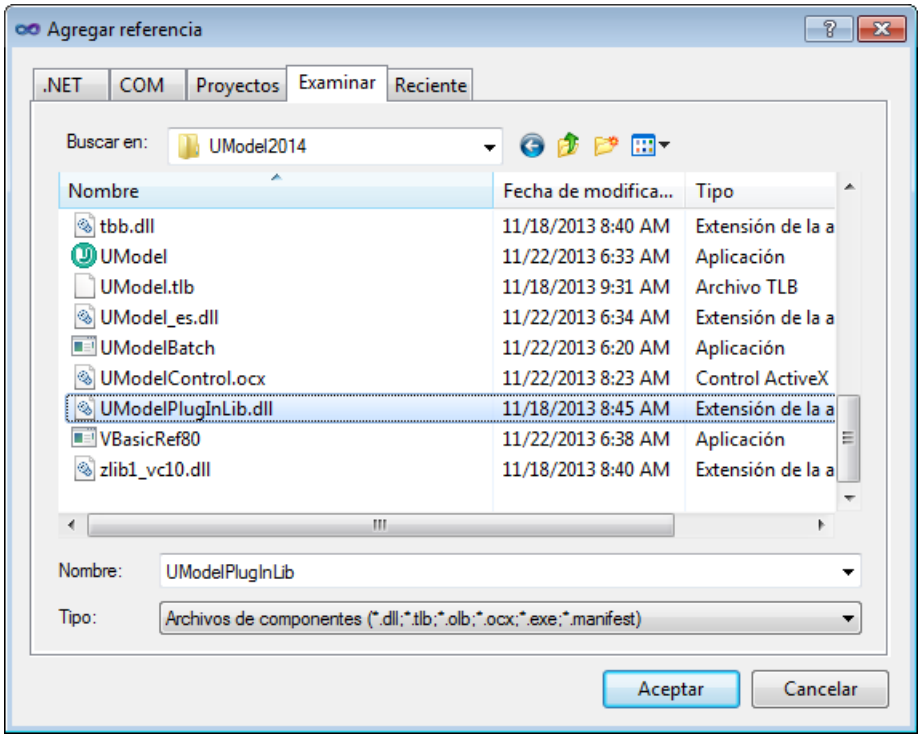
1. Inicie MS Visual Studio .NET y cree un proyecto nuevo vacío en el lenguaje pertinente (en este caso C#).
   Elija la plantilla **Biblioteca de clases** para crear una biblioteca de clases (.dll).



Agregar una referencia a la biblioteca del complemento

Agregar una referencia a la biblioteca del complemento para UModel
Si se añade un DLL a UModel en forma de complemento, es necesario que el DLL implemente la interfaz IUModelPlugIn, que se define en `UModelPlugInLib.dll`.

1. Haga clic con el botón secundario en **References** en el Explorador de soluciones y seleccione el comando **Agregar referencia...** en el menú contextual.
2. En el cuadro de diálogo seleccione la pestaña *Examinar* y busque el directorio de instalación de UModel.
3. Seleccione el archivo `UModelPlugInLib.dll`:



`UModelPlugInLib.dll` es un ensamblado .NET creado a partir de `IUModelPlugIn.tlb`. También puede crear su propio ensamblado usando el importador de bibliotecas de tipos que quiera.

Nota: el archivo `UModelPlugInLib.dll` que está instalado se generó con Microsoft .NET Framework v2.0. Si quiere instalar su complemento en un .NET Framework anterior a la versión 2.0 (p. ej. 1.1), debe crear su propio `UModelPlugInLib.dll`.

Cómo crear su propio archivo UModelPlugInLib.dll

Puede crear su propio archivo `UModelPlugInLib.dll` usando, por ejemplo, el importador de bibliotecas de tipos del kit de desarrollo Microsoft .NET Framework:

```
tlbimp.exe IUModelPlugIn.tlb
```

o para crear el ensamblado con un par de claves con nombre seguro y una versión determinada:

```
tlbimp.exe IUModelPlugIn.tlb /keyfile:UModelPlugIn.snk /
asmversion:1.0.0.0
```

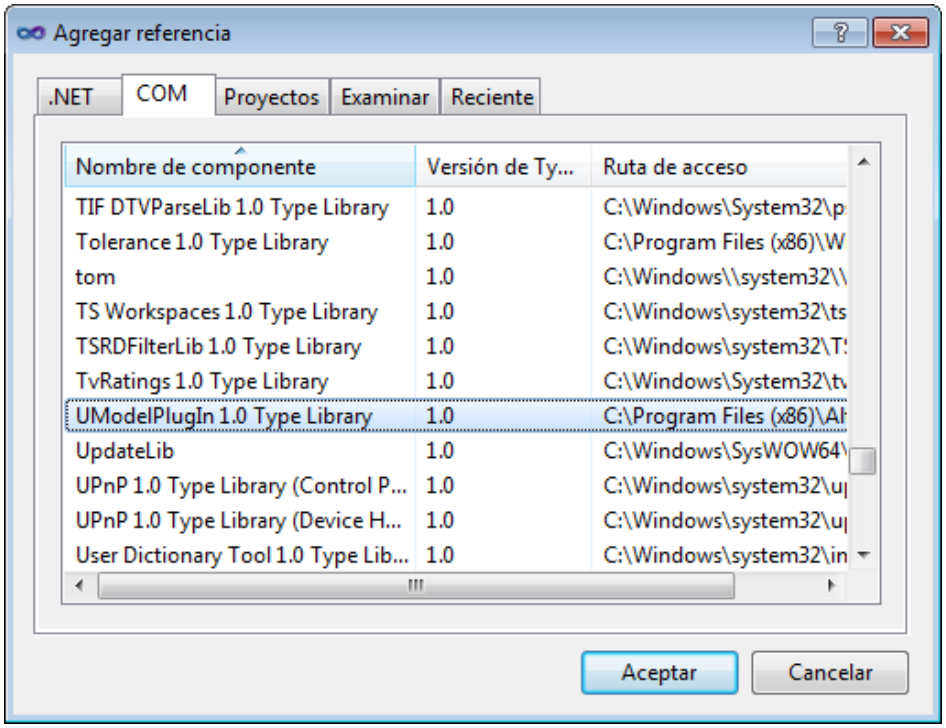
donde `UModelPlugIn.snk` es un archivo de clave creado con sn.exe (p. ej. "`sn.exe -k UModelPlugIn.snk`")

Nota: `sn.exe` forma parte del kit de desarrollo Microsoft .NET Framework.

Agregar una referencia a la biblioteca de tipos de UModel

Para usar todas las interfaces de la API de UModel es necesario añadir una referencia a la biblioteca de tipos de UModel:

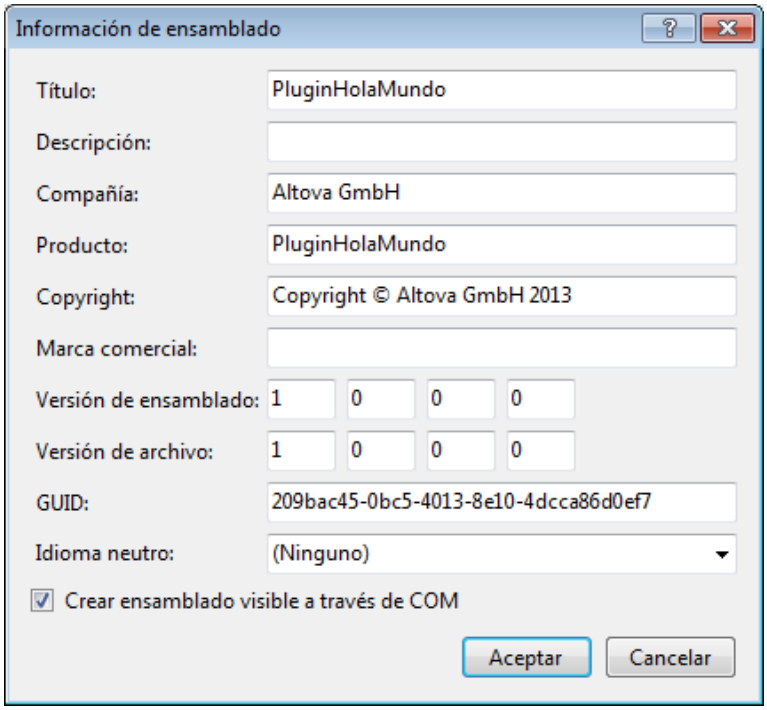
1. Haga clic con el botón secundario en **Referencias** en el Explorador de soluciones y después seleccione **Agregar referencias...** en el menú contextual.
2. En el cuadro de diálogo seleccione la pestaña *COM* y elija la biblioteca de tipos de UModel:



Cambiar la configuración del compilador (crear ensamblado visible)

Para que sea posible acceder a su código desde COM debe cambiar la configuración de su compilador.

1. Haga clic con el botón secundario en su proyecto de C# y seleccione el comando **Propiedades**.
2. En la pestaña *Aplicación* haga clic en el botón **Información de ensamblado**.
3. En el cuadro de diálogo "Información de ensamblado" marque la casilla *Crear ensamblado visible a través de COM* situada en la parte inferior.



Cambiar la configuración del compilador (exponer contenedor COM)

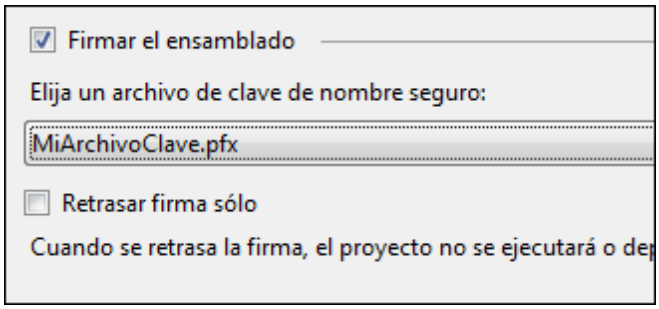
Para exponer un contenedor al que se puede llamar desde COM que pueda interactuar con objetos COM:

1. Haga clic con el botón secundario en su proyecto de C# y seleccione el comando **Propiedades**.
2. En la pestaña *Generar* marque la casilla *Registrar para interoperabilidad COM*.

Firmar el componente con un nombre seguro (opcional)

Para firmar el ensamblado con un par de claves con nombre seguro (p. ej. para la implementación):

1. Haga clic con el botón secundario en su proyecto de C# y seleccione el comando **Propiedades**.
2. En la pestaña *Firma* marque la casilla *Firmar el ensamblado* y elija un archivo de clave (bien con la opción **Examinar** para buscar un archivo de clave ya disponible, bien con la opción **Nuevo** para crear uno nuevo).



 © 2014 Altova GmbH

Implementar IUModelPlugIn

Los complementos deben implementar la interfaz IUModelPlugIn.

El código que aparece más abajo muestra una sencilla implementación de esta interfaz. Añade un elemento de menú y un separador al menú **Edición** de UModel.

Nota: dentro del método `OnUpdateCommand()` siembre está habilitado el comando `new`. El comando abre un cuadro de mensaje sencillo (compruebe que su proyecto de C# también hace referencia a `System.Windows.Forms`).

```
using System;
using System.Collections.Generic;
using System.Text;
using UModelPlugInLib;

namespace HelloWorldPlugIn
{
    public class MyHelloWorldUModelPlugIn : IUModelPlugIn
    {
        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "HelloWorldPlugIn;HelloWorldPlugIn demuestra una
sencilla implementación de un complemento para UModel";
        }

        public string GetUIModifications()
        {
            return  "<ConfigurationData>"                          +
                        "<Modifications>"                          +
                            // agregar "Hello World..." al menú
Edición
                            "<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>:Edición</Parent>"      +
                                "</UIElement>"                     +
                            "</Modification>"                      +
                        // agregar separador al menú Edición
                            "<Modification>"                       +
                                "<Action>Add</Action>"             +
                                "<UIElement type=\"MenuItem\">"    +
                                    "<ID>0</ID>"                   +
                                    "<Place>1</Place>"             +
                                    "<MenuID>101</MenuID>"         +
```

```
                                    "<Parent>:Edición</Parent>"      +
                                "</UIElement>"                       +
                            "</Modification>"                        +
                        // fin de la descripción de la modificación
                        "</Modifications>"                           +
                    "</ConfigurationData>";
        }

        public void OnInitialize(object pUModel)
        {
            // antes de procesar comandos DDE o de procesamiento por
lotes
        }

        public void OnRunning(object pUModel)
        {
            // se procesan los comandos DDE o de procesamiento por
lotes. La aplicación está totalmente inicializada
        }

        public void OnShutdown(object pUModel)
        {
            // la aplicación se cerrará; liberar todos los objetos no
utilizados
        }

        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
    }
}
```

NOTA: Las versiones de Microsoft Visual Studio anteriores a la versión 2005 generan automáticamente identificadores exclusivos globales (GUID) nuevos durante la compilación. Para usar siempre el miso GUID para su implementación de IUModelPlugIn, defina el GUID usando la clase GuidAttribute del espacio de nombres System.Runtime.InteropServices. Por ejemplo:

```
[ComVisible(true), GuidAttribute("332B1F8B-4415-44d7-BB6F-
176E2A52DFBD")]
public class MyHelloWorldUModelPlugIn : IUModelPlugIn
{
```

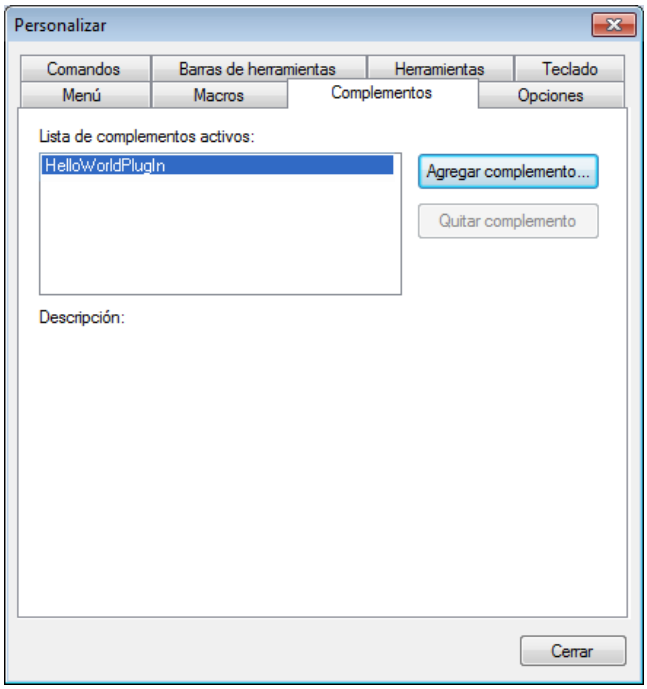
```
        ...
}
```

Probar el complemento de UModel

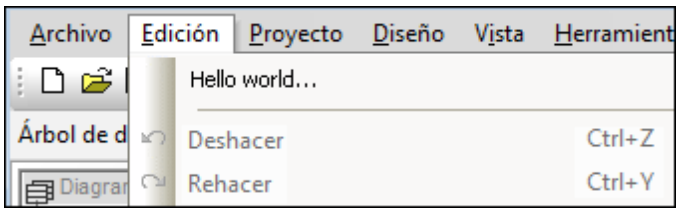
Una vez creado el proyecto de C#, puede probar el complemento de UModel.

Recuerde que no hace falta registrar/instalar el complemento antes de probarlo:

1. Inicie UModel y seleccione el comando **Herramientas | Personalizar...**.
2. En el cuadro de diálogo "Personalizar" abra la pestaña *Complementos* y haga clic en el botón **Agregar complemento**.
   Elija el archivo DLL que implementa su complemento (`PluginHolaMundo.dll`):



3. Cierre el cuadro de diálogo "Personalizar" y compruebe si el menú **Edición** se modificó según lo indicado (`GetUIModifications()`):



4. Compruebe si el comando se ejecuta tal y como especifica el complemento.

19.3.2 Implementación de complementos

Requisitos para la implementación:
Recuerde que el complemento se registra automáticamente en el equipo de desarrollo durante el proceso de generación. Además no hace falta implementar el complemento.

Esto es lo que necesita para instalar correctamente el **complemento en el sistema de destino/equipo de implementación**:

- UModel Professional y Enterprise Edition y
- Microsoft .NET Framework (su el complemento está escrito en .NET).

Implementar complementos para UModel con un instalador
Los complementos para UModel creados en entornos IDE se pueden implementar creando un instalador.

El ejemplo SetupStyles explica cómo crear un proyecto de instalación para instalar y registrar el complemento en el sistema de equipo.

A continuación le ofrecemos algunos consejos para el registro/instalación manual:

Consejos para registrar UModelPlugInLib.dll
Registre el ensamblado para COM:

```
regasm.exe UModelPlugInLib.dll /codebase
```

e instálelo en el caché global de ensamblados (GAC):

```
gacutil.exe /i UModelPlugInLib.dll
```

Nota: `regasm.exe` y `gacutil.exe` forman parte del kit de desarrollo de Microsoft .NET Framework SDK.

El archivo `UModelPlugInLib.dll` que está instalado se generó con Microsoft .NET Framework v2.0. Si tiene pensado instalar el complemento en un .NET Framework anterior a la versión 2.0, debe crear su propio archivo `UModelPlugInLib.dll` (consulte también el apartado Cómo crear el archivo UModelPlugInLib.dll)

Consejos para implementar complementos escritos en .NET para UModel
Compruebe que `UModelPlugInLib.dll` está registrado (*ver más arriba*).

- **Cómo firmar el ensamblado (complemento) con un par de claves con nombre seguro**
  Puede firmar el ensamblado con assembly linker (esto es lo recomendado). Consulte Firmar el componente con un nombre seguro) o con AssemblyKeyFileAttribute:

  cree un archivo de claves, por ejemplo:
  ```
  sn.exe -k MiArchivoDeClaves.snk
  ```

  y añada esta línea al archivo de ensamblado (p. ej. `assembly.cs`) del proyecto:
  ```
  [assembly: AssemblyKeyFile("MiArchivoDeClaves.snk")]
  ```

- **Cómo registrar el ensamblado (complemento) para COM**

 © 2014 Altova GmbH

```
regasm.exe MiPlugin.dll /codebase /tlb
```

- **Cómo instalar el ensamblado (complemento) en el caché global de ensamblados (GAC)**

```
gacutil.exe /i MiPlugin.dll
```

Nota:
`sn.exe`, `regasm.exe` y `gacutil.exe` forman parte del kit de desarrollo Microsoft .NET Framework

Cuando implemente un complemento para UModel creado en versiones de .NET Framework anteriores a 2.0, el archivo DLL debe estar instalado en el mismo directorio que UModel.exe o estar firmado con una clave de nombre seguro y registrado en el caché global de ensamblados (GAC).

Registro de complementos en UModel
UModel se reserva una clave concreta del registro, donde almacena todos sus complementos registrados:

```
HKEY_CURRENT_USER\Software\Altova\UModel\PlugIns
```

Todos los valores de esta clave se consideran referencias a complementos registrados y deben tener este formato:

Nombre del valor:	Id. de programa del complemento
Tipo de valor:	debe ser REG_SZ
Datos de valor:	CLSID del componente

Cada vez que se inicia la aplicación, los valores de la clave PlugIns se escanean y los complementos registrados se cargan.

- **Registrar el complemento a mano**
  Para registrar a mano un complemento, utilice el cuadro de diálogo "Personalizar" de UModel. Con el botón **Agregar complemento...** puede seleccionar el archivo DLL que implementa el complemento. UModel registra el archivo DLL como servidor COM y añade la entrada correspondiente en su clave PlugIns.

  Si el registro manual no funciona bien, compruebe si el CLSID del componente está registrado correctamente en la clave PlugIns. Si ese no es el problema, es probable que el nombre del archivo DLL del complemento no sea un nombre exclusivo. Utilice otro nombre o realice un registro directo.

  Aquí puede ver un ejemplo de cómo registrar un complemento a mano.

- **Registrar el complemento directamente**
  Los complementos se pueden registrar directamente como complementos. Para ello primero debe registrar el archivo DLL y después añadir el valor apropiado a la clave PlugIns de UModel durante la instalación del complemento. El nuevo complemento se activará la próxima vez que se inicie UModel.

19.3.3 XML de configuración

Con el complemento puede cambiar la interfaz del usuario (IU) de UModel. Para ello es necesario describir cada modificación con una secuencia de datos XML. Los datos XML de configuración se pasan a UModel con el método GetUIModifications de la interfaz IUModelPlugIn.

El archivo XML que contiene las modificaciones de la IU para el complemento debe seguir esta estructura:

```
<ConfigurationData>
   <ImageFile>ruta de acceso del archivo de imagen</ImageFile>
   <Modifications>
      <Modification>
         ...
      </Modification>
      ...
   </Modifications>
</ConfigurationData>
```

Puede definir iconos o botones de barra de herramientas para los nuevos elementos de menú que el complemento añade a la IU de UModel. La ruta de acceso del archivo que contiene las imágenes se establece con el elemento `ImageFile`. Las imágenes deben tener cada una un tamaño de 16 x 16 píxels. Las referencias de imagen deben hacerse de izquierda a derecha en una sola línea (<ImageFile>...). El valor de índice de la imagen más a la derecha es cero.

El elemento `Modifications` puede tener un número ilimitado de secundarios `Modification`. Cada elemento `Modification` define un cambio concreto en la IU estándar de UModel. También se pueden eliminar elementos de la IU de UModel.

Estructura de los elementos Modification

Todos los elementos `Modification` están formados por estos dos elementos secundarios:

```
<Modification>
   <Action>Type of action</Action>
   <UIElement Type="type of UI element">
   </UIElement>
</Modification>
```

Estos son los valores válidos para el elemento `Action`:

Add - para agregar el elemento de interfaz gráfica en UModel.
Hide - para ocultar el elemento de interfaz gráfica en UModel.
Remove - para quitar el elemento de interfaz gráfica del cuadro de lista *Comandos* del cuadro de diálogo "Personalizar" de UModel.

Estos valores del elemento `Action` se pueden combinar (p. ej. "Hide Remove").

El elemento `UIElement` describe elementos nuevos o ya existentes de la IU de UModel. Estos son los elementos posibles: barras de herramientas, botones, menús y elementos de menú. El atributo `type` define qué elemento de la IU describe el elemento XML.

Elementos secundarios de UIElement

 © 2014 Altova GmbH

Los elementos `ID` y `Name` son válidos para todos los tipos de fragmentos de `UIElement`. Sin embargo, se puede ignorar uno de los valores para un tipo de `UIElement` determinado (p. ej. Name se ignora en los separadores).

```
<ID></ID>
<Name></Name>
```

Si `UIElement` describe un elemento ya existente de la IU, el valor de `ID` viene ya predefinido por UModel. Lo normal es que estos valores no sean públicos. Si, por el contrario, el fragmento XML describe una parte nueva de la IU, entonces el valor de `ID` es aleatorio y el valor debe ser menor que 1000.

El elemento `Name` establece el valor textual. Los elementos ya existentes de la IU se pueden identificar por el nombre (p. ej. menús y elementos de menú con submenús asociados). Para elementos nuevos de la IU, el elemento `Name` establece el título (p. ej. el título de una barra de herramientas o el texto para un elemento de menú).

Barras de herramientas y menús

Para poder definir una barra de herramientas es necesario especificar el identificador o el nombre de la barra de herramientas. Puede referirse a barras de herramientas ya existentes con su nombre (o con su identificador, si lo conoce). Para crear una barra de herramientas **nueva**, deben definirse ambos valores. El atributo **type** debe ser igual a "ToolBar".

```
<UIElement Type="ToolBar">
   <ID>1</ID>
   <Name>Styles</Name>
</UIElement>
```

Para referirse a un menú de UModel es necesario usar dos parámetros:

- El identificador de la barra de menú que incluye el menú. (El identificador de la barra de menú principal de UModel es 101).
- El nombre del menú. Los menús no tienen asociado ningún identificador. Por ejemplo, a continuación puede ver cómo se define el menú **Edición** de la barra de menú:

```
<UIElement Type="Menu">
   <ID>101</ID>
   <Name>Edición</Name>
</UIElement>
```

Si quiere crear un elemento nuevo, debe añadir otro elemento: el elemento `Place`, que define la posición del nuevo menú en la barra de menú:

```
<UIElement Type="Menu">
   <ID>101</ID>
   <Name>PlugIn Menu</Name>
   <Place>12</Place>
</UIElement>
```

Si asigna el valor -1 al elemento `Place`, el nuevo botón o elemento de menú se coloca al final del menú o de la barra de herramientas.

Comandos

Si añade un comando nuevo (mediante un botón o un elemento de menú nuevo), el fragmento `UIElement` puede incluir estos subelementos:

```
<Info></Info>
<ImageID></ImageID>
```

El elemento `Info` contiene una breve cadena de texto que aparece en la barra de estado cuando se pasa el puntero del mouse por encima del comando (del botón o del elemento de menú). El elemento `ImageID` define el índice del icono en el archivo de imagen externo. Recuerde que todos los iconos se almacenan en un solo archivo de imagen.

Para definir un botón de barra de herramientas debe crear un fragmento `UIElement` con esta estructura:

```
<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>
```

También puede declarar un botón de barra de herramientas con estos elementos: `Place`, `ToolBarID` y `ToolBarName`. Los elementos `ToolBarID` y `ToolBarName` se utilizan para identificar la barra de herramientas que contiene el botón nuevo o ya existente. El valor textual del elemento `ToolBarName` distingue entre mayúsculas y minúsculas. El atributo **type** debe ser igual a `ToolBarItem`.

Para definir un elemento de menú, además de los elementos estándar utilizados para declarar un comando, están disponibles los elementos `MenuID`, `Place` y `Parent`. El elemento `MenuID` debe ser 101. Para más información consulte el subapartado anterior sobre barras de herramientas y menús.

El elemento `Parent` sirve para identificar el **menú** donde debe insertarse la entrada de menú nueva. Como los elementos de los submenús no tienen un identificador de Windows único, hay que encontrar otra manera de identificar al submenú.

El valor del elemento `Parent` es una ruta de acceso al elemento de menú.

El valor textual del elemento `Parent` debe equivaler al **nombre del menú primario** del submenú (y el nombre del submenú debe estar separado por dos puntos). Si el menú no tiene un primario (porque no es un submenú) añada dos puntos (:) al principio del nombre. El atributo **type** debe tener el valor `MenuItem`.

Por ejemplo, este fragmento `UIElement` define un elemento de menú:

```
<UIElement Type="MenuItem">
   <!--este elemento es un Id de comando local-->
   <ID>3</ID>
   <Name>Fill red</Name>
```

 © 2014 Altova GmbH

```
        <Place>-1</Place>
        <MenuID>101</MenuID>
        <Parent>:PlugIn Menu1</Parent>
        <ImageID>0</ImageID>
    </UIElement>
```

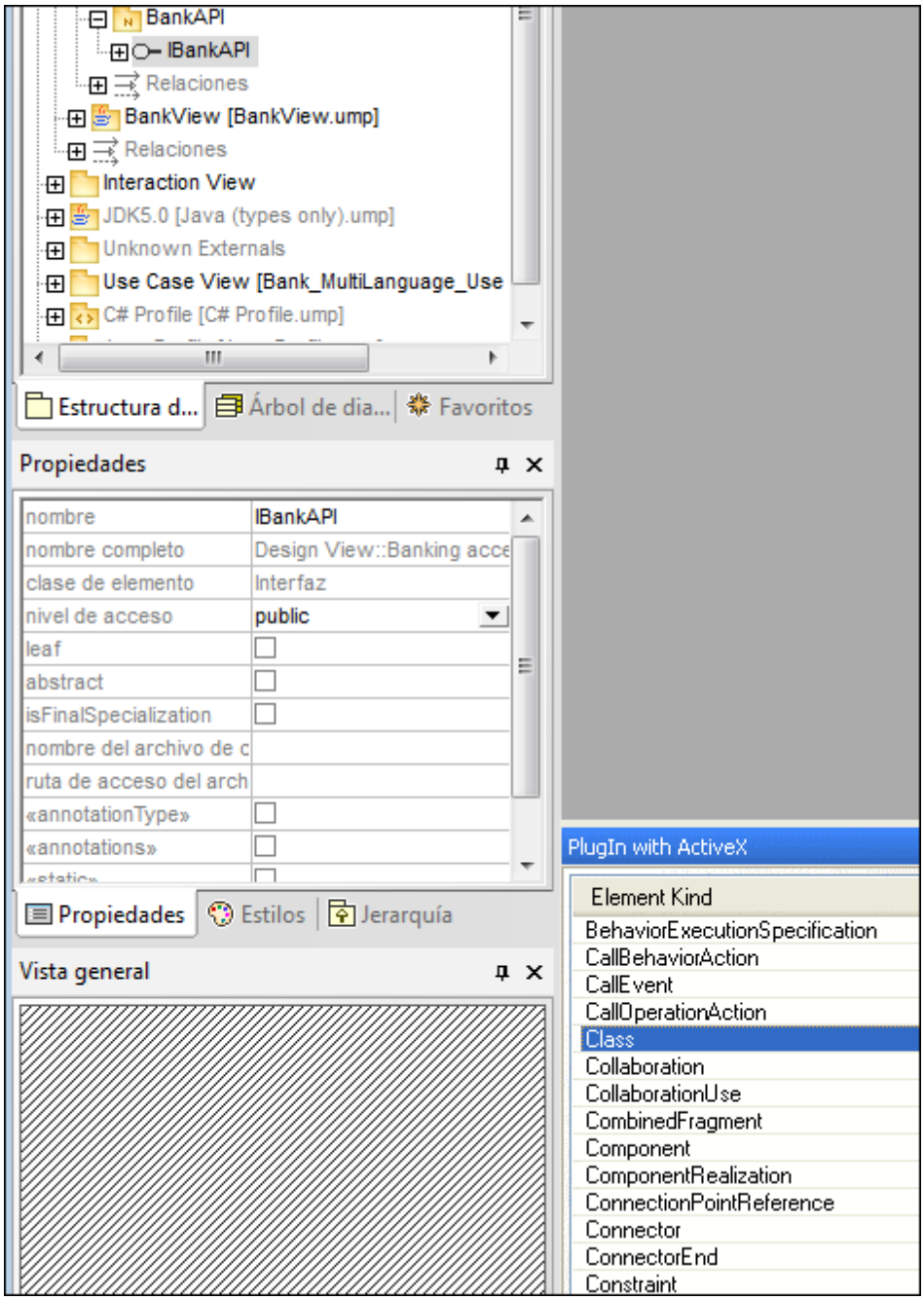
UModel permite añadir separadores de barra de herramientas y menús si el valor del elemento `ID` es 0.

19.3.4 Controles ActiveX

En la barra de control de diálogos aparecen los complementos que también sean controles ActiveX.

Para que funcione como control ActiveX, el complemento debe, por ejemplo, implementar la interfaz `IOleControl` (C++) o derivarse de `System.Windows.Forms.UserControl` (C#, VB.NET).

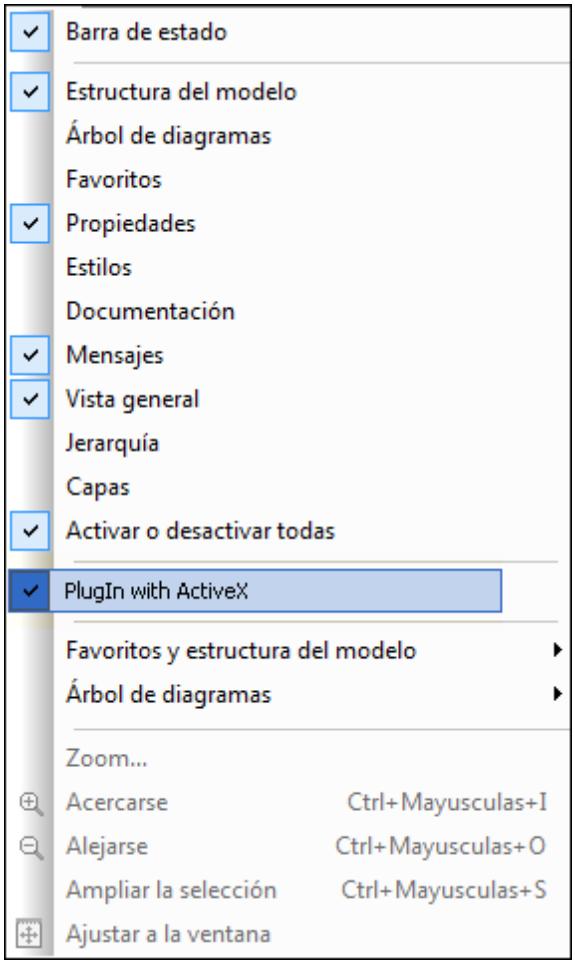
Consulte también el ejemplo StatisticsActiveX:



Cada complemento ActiveX recibe un comando de menú nuevo en el menú **Vista**. Ese comando

 © 2014 Altova GmbH

permite mostrar/ocultar la barra de control de diálogos correspondiente (panel):



19.3.5 IUModelPlugIn

Métodos

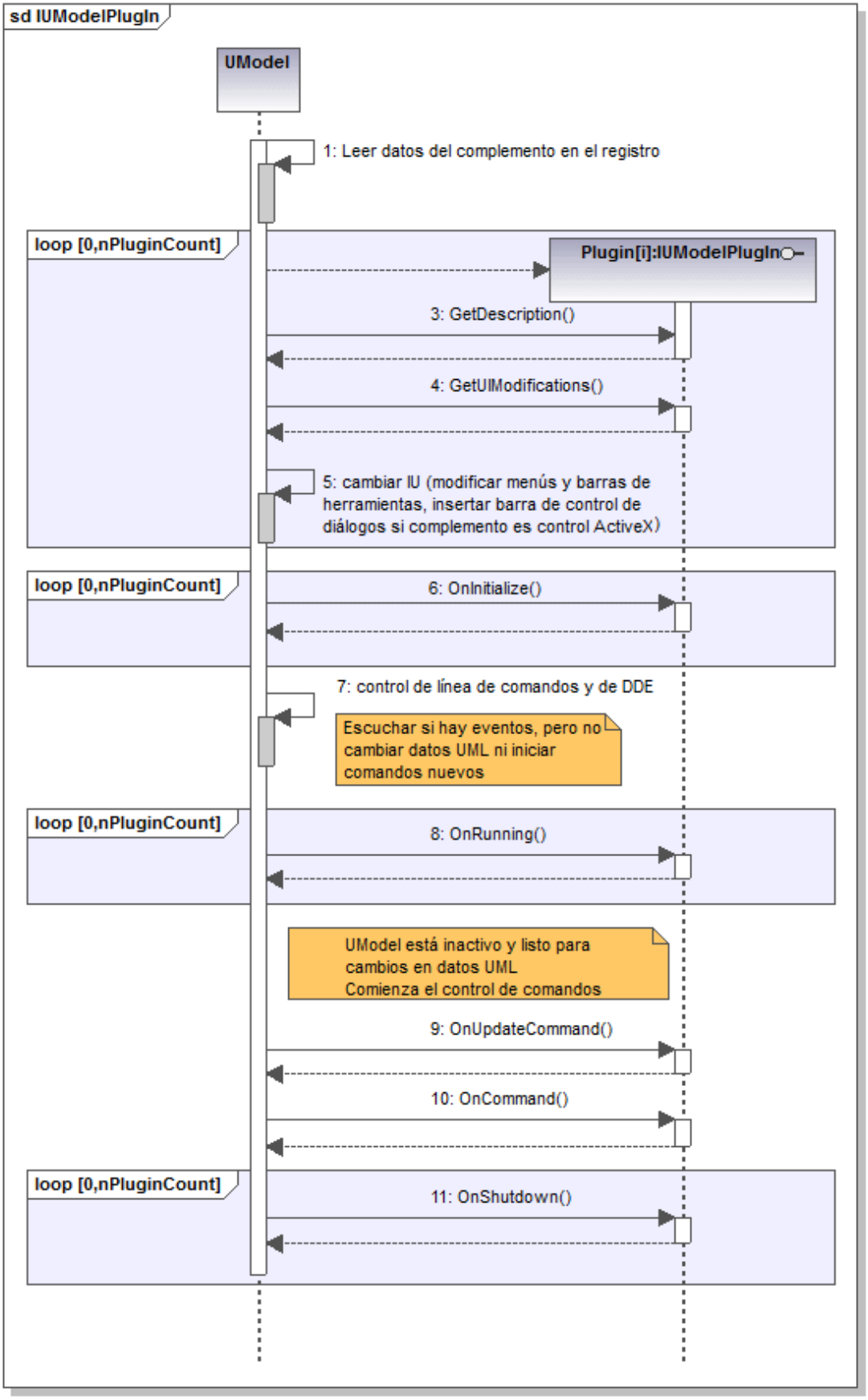
OnInitalize
OnRunning
OnShutdown
GetUIModifications
GetDescription
OnCommand
OnUpdateCommand

Descripción

Si se añade un archivo DLL como complemento en UModel, para que el DLL se reconozca como complemento es necesario que registre un componente de COM que responda a una interfaz IUModelPlugIn con el identificador UUID reservado (6B893B43-BEDD-4eea-B6D2-C0F3A6402119).

El diagrama de secuencia que aparece a continuación explica cómo interactúa UModel con IUModelPlugIn:

 © 2014 Altova GmbH


sd IUModelPlugIn
UModel
1: Leer datos del complemento en el registro
loop [0,nPluginCount]
Plugin[i]:IUModelPlugIn
3: GetDescription()
4: GetUIModifications()
5: cambiar IU (modificar menús y barras de herramientas, insertar barra de control de diálogos si complemento es control ActiveX)
loop [0,nPluginCount]
6: OnInitialize()
7: control de línea de comandos y de DDE
Escuchar si hay eventos, pero no cambiar datos UML ni iniciar comandos nuevos
loop [0,nPluginCount]
8: OnRunning()
UModel está inactivo y listo para cambios en datos UML
Comienza el control de comandos
9: OnUpdateCommand()
10: OnCommand()
loop [0,nPluginCount]
11: OnShutdown()

OnInitialize

Declaration: `OnInitialize(pUModel as IDispatch)`

Descripción
Al método `OnInitialize` de la implementación de interfaz se le llama cuando se inicializa el complemento y **antes** de que se procesen los comandos DDE o de procesamiento por lotes.

Puede adjuntar notificadores y escuchar si hay eventos de UModel, pero no debería iniciar comandos / cambios nuevos hasta que se invoque el método `OnRunning`.

`pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.

Ejemplo
```
public void IUModelPlugIn_OnInitialize(object pUModel)
{
      // antes de procesar comandos DDE o de procesamiento por lotes
}
```

OnRunning

Declaration: `OnRunning(pUModel as IDispatch)`

Descripción
Al método `OnRunning` de la implementación de interfaz se le llama cuando se inicializa el complemento y **después** de que se procesen los comandos DDE o de procesamiento por lotes.

Ahora la aplicación está completamente inicializada y puede iniciar nuevos comandos / cambios y modificar los datos UML.

`pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.

Ejemplo
```
public void IUModelPlugIn_OnRunning(object pUModel)
{
      // se procesan los comandos DDE o de procesamiento por lotes. La
aplicación está totalmente inicializada
      // y podemos adjuntar UModelApplicationEvents
      AttachAppEvents( (IApplication)pUModel );
}
```

OnShutdown

Declaration: `OnShutdown(pUModel as IDispatch)`

Descripción
Al método `OnShutdown` de la implementación de interfaz se le llama **inmediatamente antes** de que se descargue el complemento (por ejemplo cuando se va a cerrar la aplicación).

`pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.

Ejemplo
```
public void IUModelPlugIn_OnShutdown(object pUModel)
```

 © 2014 Altova GmbH

```
{
	// separar UModelApplicationEvents; dejar de recibir eventos
	DetachAppEvents();

	// la aplicación se cerrará; liberar todos los objetos no
utilizados
	GC.Collect();
}
```

GetUIModifications

Declaration: `GetUIModifications()` as String

Descripción

Al método `GetUIModifications()` se le llama durante la inicialización del complemento para obtener los datos XML de configuración que definen los cambios en la interfaz de usuario de UModel.

Al método se le llama cuando se carga el complemento por primera vez y cada vez que se inicia UModel.

Para más información sobre cómo cambiar la interfaz de usuario consulte el apartado XML de configuración.

Ejemplo

```
protected string GetPlugInPath()
{
	string sDLLPath =
System.Reflection.Assembly.GetExecutingAssembly().Location;
	return System.IO.Path.GetDirectoryName(sDLLPath);
}

public string IUModelPlugIn_GetUIModifications()
{
	try
	{
		string sPath = GetPlugInPath();

		System.IO.StreamReader myFile = new
System.IO.StreamReader(sPath + "\\config.xml");
		string sRet = myFile.ReadToEnd();
		myFile.Close();

		// esto reemplaza el símbolo "**path**" del archivo XML
con
		// la ruta real de instalación del complemento para
obtener el archivo de imagen
		return sRet.Replace("**path**", sPath);
	}
	catch (System.Exception ex)
	{
		System.Windows.Forms.MessageBox.Show("Error in
GetUIModifications:" + ex.Message);
```

```
            throw ex;
        }
}
```

GetDescription

Declaration: `GetDescription()` as String

Descripción

`GetDescription()` se utiliza para definir la cadena de descripción para las entradas del complemento que son visibles en el cuadro de diálogo "Personalizar".

Ejemplo

```
public string IUModelPlugIn_GetDescription()
{
      return "Complemento con muestra de estilos para UModel. Este
complemento demuestra cómo cambiar varios estilos de los elementos de
diagrama seleccionados.";
}
```

OnCommand

Declaration: `OnCommand(nID` as long, `pUModel` as `IDispatch)`

Descripción

Al método `OnCommand()` de la implementación de interfaz se le llama cada vez que se procesa un comando añadido por el complemento (ya sea un comando de menú o un botón de una barra de herramientas).

`nID` almacena el identificador del comando definido por el elemento `ID` del correspondiente `UIElement`.

`pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.

Ejemplo

```
public void IUModelPlugIn_OnCommand(int nID, object pUModel)
{
      // rellenar en rojo
      if (nID == 3 || nID == 6)
            OnSetStyles((IApplication)pUModel, "red");

      // rellenar en verde
      if (nID == 4 || nID == 7)
            OnSetStyles((IApplication)pUModel, "green");

      // liberar objetos no utilizados
      GC.Collect();
}
```

 © 2014 Altova GmbH

OnUpdateCommand

Declaration: `OnUpdateCommand(nID` as long, `pUModel` as `IDispatch)` as `UModelUpdateAction`

Descripción
Al comando `OnUpdateCommand()` se le llama cada vez que se necesita establecer el estado visible de un botón u opción de menú.

`nID` almacena el identificador del comando definido por el elemento `ID` del correspondiente `UIElement`.

`pUModel` almacena una referencia a la interfaz de envío del objeto `Application` de UModel.

Los valores devueltos posibles (según lo definido en UModelUpdateAction) para establecer el estado de actualización son:

`UModelUpdateAction_Enable`	= 1
`UModelUpdateAction_Disable`	= 2
`UModelUpdateAction_Check`	= 4
`UModelUpdateAction_Uncheck`	= 8

Puede combinar varios valores usando el operador OR bit a bit (p. ej. `UModelUpdateAction_Enable | UModelUpdateAction_Check`).

Ejemplo

```
public UModelUpdateAction IUModelPlugIn_OnUpdateCommand(int nID, object
pUModel)
{
	UModelUpdateAction action =
UModelUpdateAction.UModelUpdateAction_Disable;

	// buscar "rellenar en rojo"
	if (nID == 3 || nID == 6)
		action = OnUpdateSetStyles((IApplication)pUModel);

	// buscar "rellenar en verde"
	if (nID == 4 || nID == 7)
		action = OnUpdateSetStyles((IApplication)pUModel);

	// liberar objetos sin utilizar
	GC.Collect();

	return action;
}
```

19.4 The UModel API

The COM-based API of UModel enables clients to easily access the functionality of UModel. As a result, it is now possible to automate a wide range of tasks.

UModel follows the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the UModel API from common development environments, such as those using C, C++, C# and VisualBasic, and with scripting languages like JavaScript and VBScript.

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of UModel may still remain in the system.
- See Error handling for details of how to avoid annoying error messages.
- Free references explicitly, if using languages such as C or C++; "`GC.Collect()`" can be used in C# and VisualBasic to force garbage collection.
- Umodel API collections are zero-based (e.g. `myPackage.InsertPackagedElementAt( 0, "Interface"`); will insert a new interface as first child of the package)

 © 2014 Altova GmbH

19.4.1 Overview

This overview of the UModel API provides you with the object model for the API and a description of the most important API concepts.
The following topics are covered:

- The object model
- How to...
- Error handling

Object model

The starting point for every application which uses the UModel API is the IApplication interface.

To create an instance of the application object, call `WScript.GetObject("", "UModel.Application")` from JScript, or a similar function from your preferred development environment, to create a COM object.

There is no need (or other way) to create any objects other than to use the complete UModel API. All interfaces are accessed through other objects, with the application object (IApplication) as the starting point.

Note: When implementing a UModel IDE plugin, there is no need to create an instance of the application object, because UModel is already running and the current instance of the application object is provided by IApplication as parameter for all important methods of IUModelPlugIn.

The application object consists of the following parts (each indentation level indicates a child–parent relationship with the level directly above):

```
IApplication
   IDocument
      IDiagramWindows
         IDiagramWindow
      IFocusedUMLDataNotifier
      ITransactionNotifier
      IUMLData (and all other derived UML data interfaces)
         IUMLDataList
   IDialogs
      IExportXMIFileDlg
      IGenerateDocumentationDlg
         IKindSelectionList
            IKindSelection
      IGenerateSequenceDiagramDlg
      IGenerateStateMachineCodeDlg
      IImportBinaryTypesDlg
         IImportBinaryTypeEntries
            IImportBinaryTypeEntry
      IImportDatabaseDlg
      IImportSourceDirectoryDlg
      IImportSourceProjectDlg
      IImportXMLSchemaDirectoryDlg
      IImportXMLSchemaFileDlg
      IIncludeSubprojectDlg
         IModelTransformationDlg
```

```
            IModelTransformationTypeMappings
               IModelTransformationTypeMapping
      IProjectSettingsDlg
      ISaveAllDiagramsAsImagesDlg
      ISynchronizationSettingsDlg
         IMatchRenamedDlg
            IMatchRenamedEntries
               IMatchRenamedEntry
      IURLDlg
   ILocalOptions
      ILocalOptionsCodeEngineering
      ILocalOptionsDiagramEditing
         ICollectionTemplates
            ICollectionTemplate
      ILocalOptionsEditing
      ILocalOptionsFile
      ILocalOptionsView
```

Enumerations
Events

Once you have created an application object, you can start using the functionality of UModel. You will generally either open an existing document, create a new one, or access the active document (IDocument).

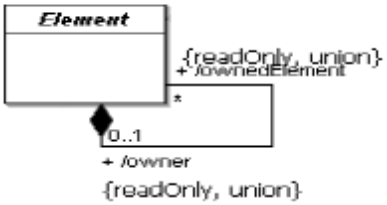
IDocument corresponds to a UModel project and can be used to include sub-projects, generate documentation, synchronize model and code..., while also giving access to the main UMLData objects (root package, GUI root element, also see Object model UMLData).

Object model UMLData

The starting point to access UML elements is the root package (IUMLPackage), which is a property of the IDocument interface.

All children of the root package are a subtype of IUMLElement and are stored as defined by the OMG in the UML 2.4 Superstructure Specification (also see http://www.uml.org).

E.g. the UML Superstructure Specification defines the following relationship for UML Element:

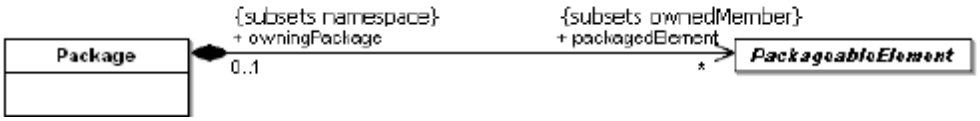


Which means that every UML element can have a list of owned elements, and every UML element (apart from the root-package) has an owner:

In the UModel API this is mapped to IUMLElement having the properties "OwnedElement" and "Owner". Since these relationships are "read only" in the UML specification, both properties cannot be modified in the UModel API.

 © 2014 Altova GmbH

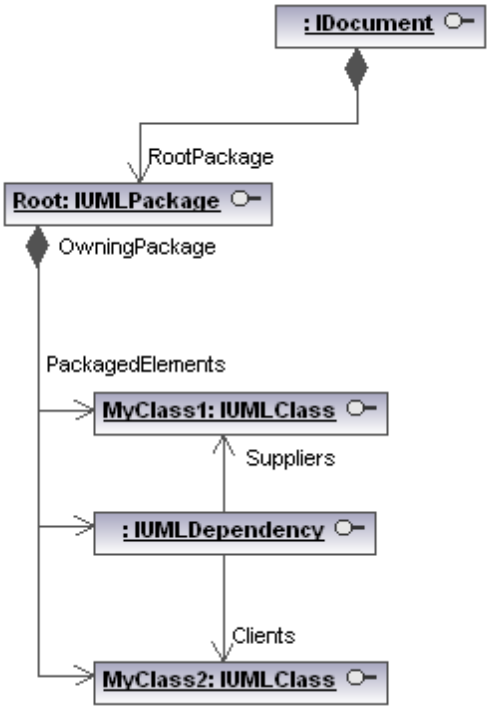
The UML superstructure specification also defines the following relationship between Package and PackageableElement:



This is mapped to IUMLPackageableElement having a property "OwningPackage" and an IUMLPackage, which not only has a property "PackagedElements", but also a method "InsertPackagedElementAt" to insert new IUMLPackageableElements (at the specified position).

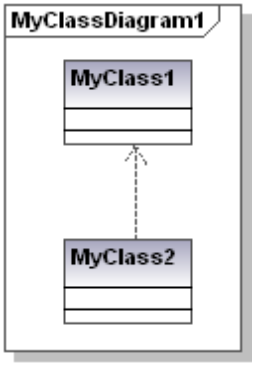
Note: "EraseFromModel" deletes any IUMLElement (and all its children) from the model.

The sample below shows the mapping of a project which consists of two classes (IUMLClass) with a dependency (IUMLDependency) between them:



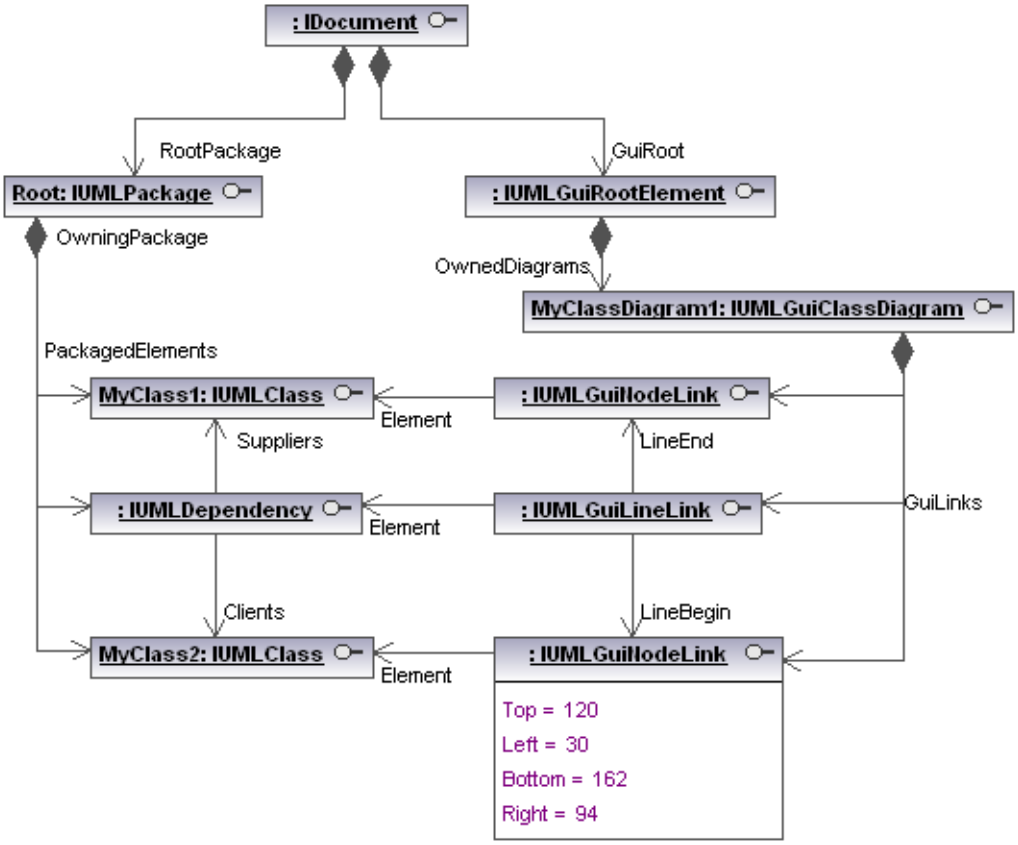
This structure is independent of whether these elements are shown on any diagram or not.

The representation of graphical objects on diagrams (as shown in the image below) is stored in a second structure with elements of kind IUMLGuiElement (also see Graphical Objects).



The starting point to access UML GUI elements is the GuiRoot (IUMLGuiRootElement), which is a property of the IDocument interface.

Lines are handled by IUMLGuiLineLinks, most other objects (like classes, interfaces, packages,...) by IUMLGuiNodeLinks.

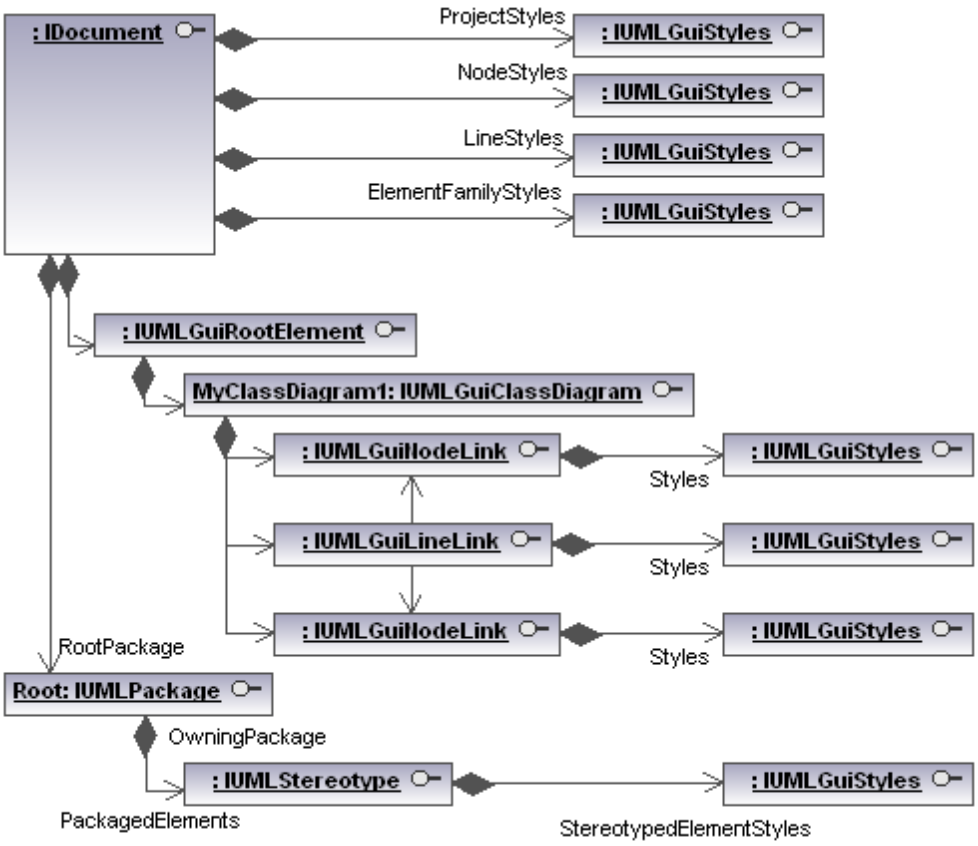


Object model UMLData styles

UModel has various styles allowing you to adapt the diagram appearance (i.e. font size, weight, color, visibility options,...).

 © 2014 Altova GmbH

The following picture shows how the different styles (IUMLGuiStyles) can be accessed using the UModel API:



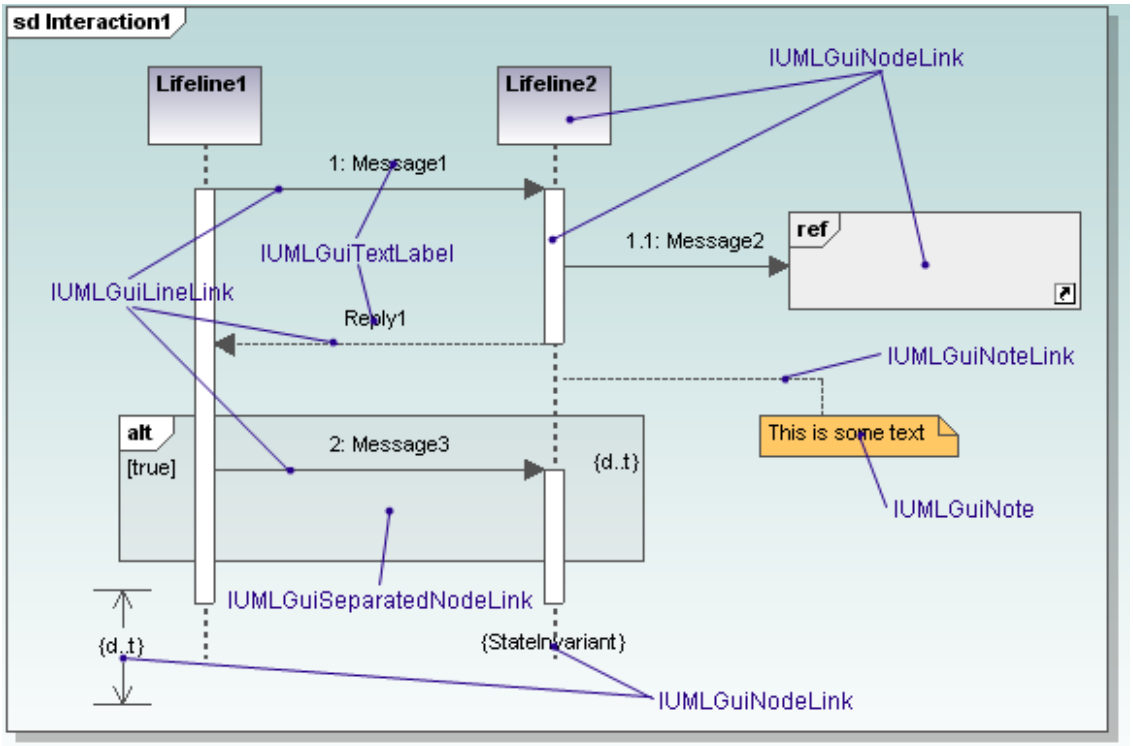
The different styles can be identified by ENUMUMLGuiStyleKind.

Graphical Objects

In the UModel API, graphical objects on diagrams are represented by objects derived from the IUMLGuiElement interface. Most of them can be accessed using the IUMLGuiDiagram property 'GuiLinks'.

For most diagrams, most objects which are lines are instances of IUMLGuiLineLink and most other, solid objects or 'nodes' are instances of IUMLGuiNodeLink. These interfaces have properties and methods for manipulating the basic properties of these graphical objects, such as position, color and style.

There are of course more specialized interfaces derived from these general interfaces which provide access to special properties. The following image shows a sequence diagram and the interface representing each graphical object on it:



How to...

How to create Sequence Diagrams

There are two ways to create sequence diagrams programmatically using the UModel API:

- Generating a sequence diagram from existing source code when there is code available that you want to be reverse engineered and displayed as UML diagram
- Manually create a sequence diagram from scratch using IUMLGuiElements directly

How to generate Sequence Diagrams from code

Sequence diagrams in UModel can be generated programmatically from an IUMLOperation element. The operation needs to exist in the model and have some source code associated to it.

The operation could possibly have been previously "read" by the reverse engineering functionality of UModel. Creating new Sequence Diagrams programmatically by reverse engineering source code using the UModel API involves two short steps:

- Setting up the options for diagram generation
- Invoking the diagram generation function

The following C# code shows how to set up the options and start the generation of the sequence diagram:

```
// 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);
}
```

How to create Sequence diagrams manually

Creating new Sequence Diagrams programmatically from scratch using the UModel API is basically nothing more than placing interaction fragments, such as Lifelines on a diagram and connecting them with messages.

Messages can easily be created using the **`AddUMLLineElement()`** `method` of IUMLGuiLineLink, which removes the necessity of creating multiple underlying UML Elements such as MessageEnds, ExecutionOccurrences and similar manually.

To make it simple to create Messages between two interaction fragments such as Lifelines, create a small helper function which calls `AddUMLLineElement()` and layouts the created line:

```
// 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;
}
```

As you can see, IUMLDiagram.AddUMLLineElement() accepts as a parameter not only the string "Message", to create a Message Line; but also "Reply", "Create" and "Destruct", for Reply Messages, Creation Messages and Destruction Messages.

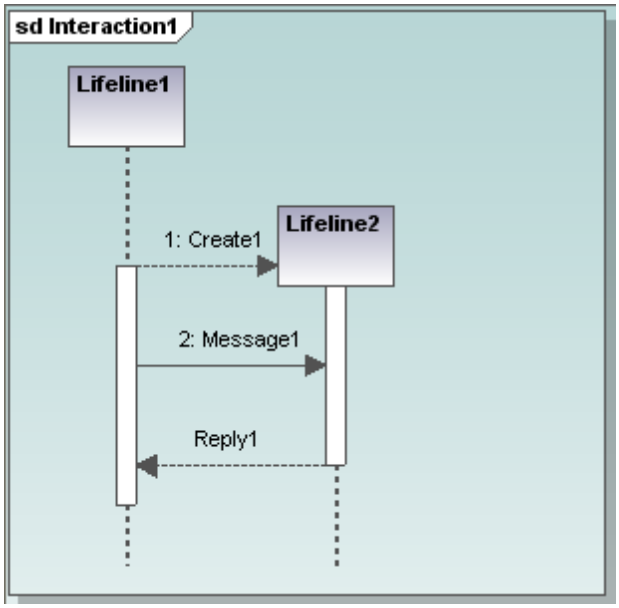
In order to create a simple diagram it is only necessary to create a Sequence Diagram in the GuiRoot object, open the diagram, add a handful of lifelines and connect them with messages using this helper function:

```
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);
```

 © 2014 Altova GmbH

```
// connect these lifelines using some messages

addMessage(100, "Create",  lifeline1, lifeline2, wnd);
addMessage(150, "Message", lifeline1, lifeline2, wnd);
addMessage(200, "Reply",   lifeline2, lifeline1, wnd);
```

The resulting created Diagram will look like this:



Setting the Type of a Lifeline

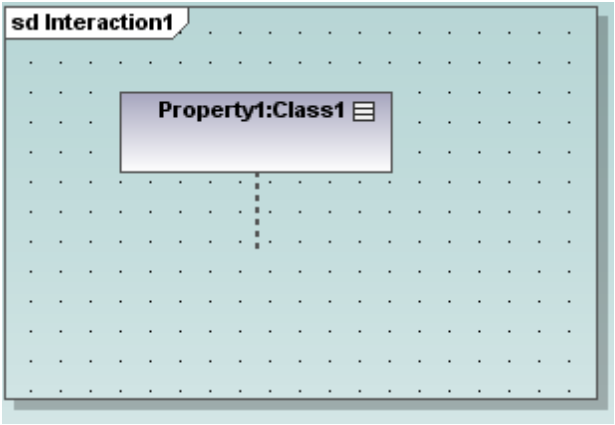
To display the Type represented by a Lifeline, be it be a Class, Interface, DataType or similar, this is done by using the IUMLLifeline.Represents property which references a IUMLProperty. If the Type of this property is set, the Type will show up on the diagram as well.

The following code creates a Lifeline which references a class:

```
// 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);
```

The resulting lifeline would then look like this:



Setting the Operation of a Message

Messages usually represent the invocation of an operation of an object. Note: based on the type of the Message (normal Message, Creation, Deletion or Reply) and the existence, or absence of underlying UML elements, such as MessageOccurenceSpecifications or CallEvents, it is not always possible for a Message to represent an Operation, and getting the cprrect UML element to point to the Operation is not that trivial.

This is why the IUMLMessage interface in the UModel API, offers the method **SetOperation()** with makes it possible to let a Message refer an Operation if it is able to do so:

```
// 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);
```

 © 2014 Altova GmbH

Undo / Redo and UMLData transaction handling

When modifying the UML data structure using the UModel API, there is no need to take care of Undo/Redo or transactions.

The following code makes three modifications:

```
        public void ChangeClass( IUMLClass iClass )
        {
            iClass.SetName("NewName");
            iClass.Visibility =
ENUMUMLVisibilityKind.eVisibility_Public;
            iClass.IsAbstract = true;
        }
```

and for every modification, a new undo-step is created, in other words: the user will have to press the "Undo" button three times in UModel to undo these three changes.

This is not always the required behavior so the UModel API supports "transaction-handling" making it possible to execute multiple modifications in one step.

IDocument has the functionality to define when a group of modifications starts ("BeginModification") and when it ends ("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
            }
        }
```

This kind of transaction handling may only be used for UML data modifications.
Other functions, such as e.g. 'synchronize model from code', will create one single Undo step anyway.

How to use predefined UModel elements

UModel defines several important elements as "predefined". This includes several internal elements (Root, Component View and Unknown Externals package) as well as the elements of all profiles installed with UModel (e.g. the C#, VB and Java profile).

Predefined elements can be uniquely identified by using ENUMUMLPredefinedElement, which allows direct and easy access to these elements for several functionalities, e.g.

- Find a predefined element:

```
// get the CSharp profile
IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc
.RootPackage
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement.ePredefined_CSharp_Profile, false);
```

- Apply a predefined stereotype

```
// set the CSharp 'delegate' stereotype
iClass
.ApplyPredefinedStereotype
( ENUMUMLPredefinedElement
.ePredefined_CSharp_delegateStereotypeOfClass );
```

- Check if a predefined stereotype is applied:

```
// check if package is a CSharp - namespace (if 'namespace' stereotype
is applied)
bool bIsCSharpNamespace =
iPackage
.IsPredefinedStereotypeApplied
( ENUMUMLPredefinedElement
.ePredefined_CSharp_namespaceStereotypeOfPackage );
```

- Set the tagged value of a predefined tag definition:

```
// set attribute-section "STAThread"
// ...
iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement
.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
iSTAThread.Name);
```

How to work with Stereotypes and Tagged Values

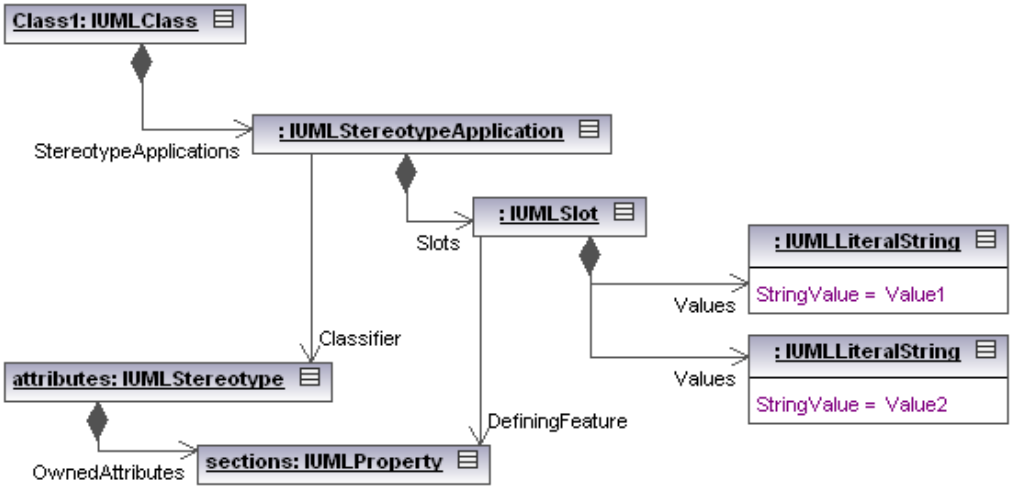
Stereotypes and tagged values are quite complex as defined in the UML Superstructure Specification.

UModel has simplified their handling and treats them similar to IUMLInstanceSpecifications and

IUMLSlots in UML.

In the following sample the stereotype "attributes" is applied to "Class1", and the tag definition "sections" has the tagged values "Value1" and "Value2":

UModel API introduces IUMLStereotypeApplications and maps the sample above to the following UMLData structure:



Applying stereotypes and setting tagged values using the UModel API is quite simple:

```
IUMLStereotype iStereotypeAttributes = ...;
IUMLProperty iTagDefSections = ...;
IUMLClass iClass = ...;

IUMLStereotypeApplication iStereotypeApp =
iClass.ApplyStereotype(iStereotypeAttributes);
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value1");
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value2");
```

Also see section Predefined UModel elements how to deal with predefined stereotypes, tag definitions and tagged values.

How to use UMLData events and eventfilters

Event receivers must implement the _IUMLDataEvents interface in order to receive one or more of following possible events from IUMLData objects:

- OnBeforeErase: sent immediately before the UML data is erased from the model.
  If multiple data are erased, this event is sent for every IUMLData (not only for the topmost one)
- OnAfterAddChild: sent when the UML data is added to the model tree.
  If multiple data are added in one step (e.g. a class with multiple attributes is added to a package) only the topmost IUMLData event is sent
- OnChanged: sent when the UML data has been modified (e.g. when a class name is changed)
- OnMoveData: sent when the UML data has been moved to a new parent (e.g. when a class is moved to another package in the ModelTree).

  This event always occurs twice: once when detaching from the old parent, and once when the UML data is attached to the new parent.

Eventfilter can be set with (combinations of) ENUMUMLDataEventFilter in order to specify which events should be sent by the UModel API.

To keep performance high and the overhead as low as possible, event receiver should only register for those events, they are really interested in.

E.g. the following code registers "OnAfterAddChild" events when exactly the root-package gets a new child (no event will arrive if a child of the root-package gets a new child):

```
// 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 work hierarchically, so the event filter can be set to receive events from the attached IUMLData only, or from the attached IUMLData and any of its children (grandchildren,...).

```
// ensure we get "OnBeforeErase" events also for *any* erased child
(grandchild,...) of the rootpackage
m_RootPackage.EventFilter |= (int)
ENUMUMLDataEventFilter.eUMLDataEvent_EraseDataOrChild;
```

UMLData events are also sent when UML data is modified by Undo / Redo, but beware that **no** UML data modification may be made during Undo / Redo:

```
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)
	{
		// ...
	}
}
```

 © 2014 Altova GmbH

How to create and use Hyperlinks

UModel allows hyperlinks between most modeling elements (except for lines) and:

- any diagram in the current ump project
- any element on a diagram
- any element in the Model Tree
- external documents, e.g. PDF, Excel or Word documents
- web pages

Also see Hyperlinking modeling elements.

Hyperlinks are not part of the UML specification and the UModel API introduces following interfaces for hyperlinks on

IUMLNamedElements:

- IUMLHyperlink is the common base interface and can be used to open links as well as to retrieve the default- and user-defined link name
- IUMLHyperlink2File to handle external documents and web pages
- IUMLHyperlink2GuiElement to handle any diagram in the current ump project or any element on a diagram
- IUMLHyperlink2Model for hyperlinks to model elements (in the Model Tree)

Examples:

Insert a hyperlink to the Altova homepage:

```
IUMLHyperlink2File iHyperlink = iMyClass.InsertOwnedHyperlink2FileAt(-
1, "http://www.altova.com");
```

Insert a hyperlink to a diagram of the current ump project:

```
IUMLGuiDiagram iDiagram = ...;
IUMLHyperlink2GuiElement iHyperlink =
iMyClass.InsertOwnedHyperlink2GuiElementAt(-1, iDiagram, null);
```

Insert a hyperlink to the representation of a class on a diagram:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLHyperlink2GuiElement iHyperlink =
iMyClass.InsertOwnedHyperlink2GuiElementAt(-1, iNodeLink, null);
```

Insert a hyperlink to an attribute of a class on a diagram:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLProperty iAttribute = ...;
IUMLHyperlink2GuiElement iHyperlink =
iMyClass.InsertOwnedHyperlink2GuiElementAt(-1, iNodeLink, iAttribute);
```

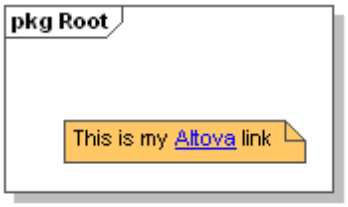
Insert a hyperlink to the same attribute (from above) in the Model Tree:

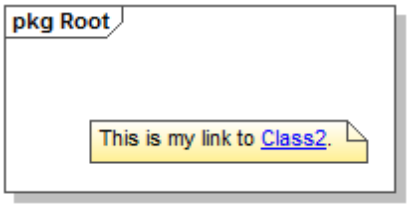
```
IUMLHyperlink2Model iHyperlink = iMyClass.InsertOwnedHyperlink2ModelAt(-1,
iAttribute);
```

Open all hyperlinks of an IUMLNamedElement:

```
foreach (IUMLHyperlink iHyperlink in iMyClass.OwnedHyperlinks)
      iHyperlink.OpenLink();
```

UModel also allows hyperlinks in notes (IUMLGuiNote) and comments (IUMLComment):





These are handled by IUMLGuiTextHyperlinks (respectively IUMLCommentTextHyperlinks)and the start- and end-character position of the hyperlink must be specified, e.g:

```
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");
```

Similar for hyperlinks in comments:

```
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 );
```

Error handling

The UModel API returns errors in two different ways. Every API method returns an `HRESULT`. This return value informs the caller about any errors during the execution of the method. If the call was successful, the return value is equal to `S_OK`. C/C++ programmers generally use `HRESULT` to detect errors.

VisualBasic, scripting languages, and other high-level development environments do not give the programmer access to the returning `HRESULT` of a COM call. They use the second error-raising mechanism supported by the UModel API, the `IErrorInfo` interface. If an error occurs, the API creates a new object that implements the `IErrorInfo` interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

 © 2014 Altova GmbH

The following text describes how to deal with errors raised from the UModel API in different development environments.

VisualBasic

A common way to handle errors in VisualBasic is to define an error handler. This error handler can be set with the `On Error` statement. Usually the handler displays an error message and performs cleanup functions to avoid spare references and any kind of resource leaks.

VisualBasic fills its own `Err` object with the information from the `IErrorInfo` interface.

Example:

```
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

The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, in that you also declare an error object containing the necessary information.

Example:

```
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++
C/C++ gives you easy access to the `HRESULT` of the COM call and to the `IErrorInterface`.

```
eobpri q= êX

// Call DoSomeWork() from the UModel API
if(FAILED(hr = ipDocument->DoSomeWork()))
{
 IErrorInfo *ipErrorInfo = Null;

 if(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
 {
  BSTRbstrDescr;
  ipErrorInfo->GetDescription(&bstrDescr);

  // handle Error information
  wprintf(L"Error message:\t%s\n",bstrDescr);
  ::SysFreeString(bstrDescr);

  // release Error info
  ipErrorInfo->Release();
 }
}
```

Programming languages

Programming languages differ in the way they support COM access. A few examples for the most frequently used languages (*links below*) will help you get started. The code listings in this section are available in files in the API Examples folder and can be tested straight away. The path to the API Examples folder is given below:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/`
Windows Vista, Windows 7	`C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/`

C#
C# can be used to access the Application API functionality. The code listings show how to access the API for certain basic functionality.

Java
The UModel API can be accessed from Java code. This section explains how some basic UModel functionality can be accessed from Java code. It is organized into the following sub-sections.

JScript
The JScript listings demonstrate the following basic functionality.

 © 2014 Altova GmbH

C#

The C# programming language can be used to access the Application API functionality. You could use Visual Studio 2008 or Visual Studio 2010 to create the C# code, saving it in a Visual Studio project. Create the project as follows:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the UModel Type Library by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed COM components. Select the UModel Type Library component from the list to add it.
3. Enter the code you want.
4. Compile the code and run it.

Example C# project
Your UModel package contains an example C# project, which is located in the C# folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/`
Windows Vista, Windows 7	`C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/`

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Platform configuration
If you have a 64-bit operating system and are using a 32-bit installation of UModel, you must add the x86 platform in the solution's Configuration Manager and build the sample using this configuration.

A new x86 platform (for the active solution in Visual Studio) can be created in the New Solution Platform dialog (Build | Configuration Manager | Active solution platform | <New…>)

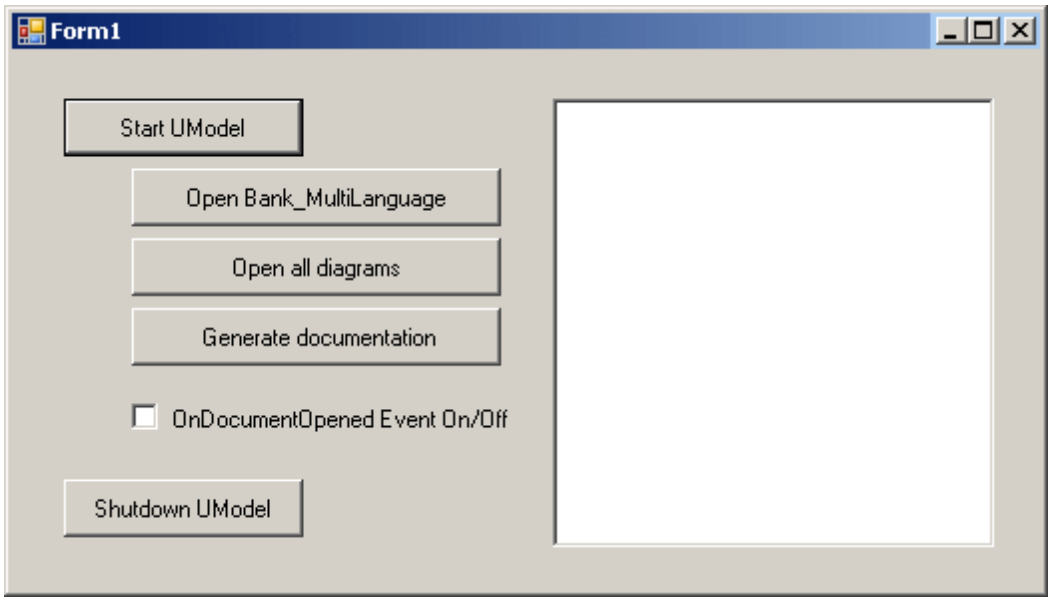
File List:

Readme.txt	This file
AutomateUModel.csproj	Visual Studio 2008 and 2010 project file
AutomateUModel_VS2008.sln	Visual Studio 2008 solution file
AutomateUModel_VS2010.sln	Visual Studio 2010 solution file
Program.cs	C# example source code

Properties:
Form1.cs
Form1.Designer.cs
Form1.resx

What the example does
The example shows a simple user interface (*screenshot below*) with buttons that invoke basic UModel operations:

- Start UModel
- Open Bank_MultiLanguage.ump
- Open All Diagrams
- Generate documentation for the currently active document
- Shows how to listen to UModel events (OnDocumentOpened Event On/Off)
- Shutdown UModel



Compiling and running the example
Double-click the file `AutomateUModel_VS2008.sln` (for Visual Studio 2008) or the file `AutomateUModel_VS2010.sln` (for Visual Studio 2010). Alternatively the file can be opened from within Visual Studio (with **File | Open | Project/Solution**). To compile and run the example, select **Debug | Start Debugging** or **Debug | Start Without Debugging**.

Code listing of Form1.cs
The listing is commented for ease of understanding. Parts of the code are also presented separately in the sub-sections of this section, according to the Application API functionality they access.

The code essentially consists of a series of handlers for the buttons in the user interface shown in the screenshot above.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
```

```
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // An instance of UModel accessed via its automation interface.
        UModelLib.Application UModel;
        UModelLib.Document UModelDocument;
        String docName = "Bank_MultiLanguage.ump";
        // Location of examples installed with UModel
        String strExamplesFolder;

        private void Form1_Load(object sender, EventArgs e)
        {
            // locate examples installed with UModel.
            // REMARK: You might need to adapt this if you have a different
major version of the product.
            strExamplesFolder =
Environment.GetEnvironmentVariable("USERPROFILE") + "\\My Documents\\Altova\
\UModel2012\\UModelExamples\\";
        }

        // handler for the "Start UModel" button
        private void StartUModel_Click(object sender, EventArgs e)
        {
            if (UModel == null)
            {
                Cursor.Current = Cursors.WaitCursor;

                // if we have no UModel instance, we create one a nd make it
visible.
                UModel = new UModelLib.Application();
                UModel.Visible = true;

                Cursor.Current = Cursors.Default;
            }
            else
            {
                // if we have already an UModel instance running we toggle its
visibility flag.
                UModel.Visible = !UModel.Visible;
            }
        }

        // handler for the "Open Bank_Multilanguage" button
        private void openBank_MultiLanguage_Click(object sender, EventArgs e)
        {
            if (UModel == null)
                StartUModel_Click(null, null);

            // Open one of the sample files installed with the product.
            UModelDocument = UModel.OpenDocument(strExamplesFolder + docName);
```

```
        }

        // handler for the "Open All Diagrams" button
        private void openAllDiagrams_Click(object sender, EventArgs e)
        {
            if (UModel == null)
                StartUModel_Click(null, null);

            if (UModelDocument != null)
            {
                UModelDocument.OpenAllDiagrams();
                UModelLib.DiagramWindows diagramWindows =
UModelDocument.DiagramWindows;
                System.Collections.IEnumerator diagramWindowsEnum =
diagramWindows.GetEnumerator();

               if ( diagramWindows.Count == 0 )
                   listBoxMessages.Items.Add("No opened diagrams.");
               else
                   listBoxMessages.Items.Add("Opened diagrams:");

                int count = 0;

                while ( diagramWindowsEnum.MoveNext() )
                {
                    String sName = "";
                    Object obj = diagramWindowsEnum.Current;

                    if (obj is UModelLib.DiagramWindow)
                    {
                        sName = ((UModelLib.DiagramWindow)obj).Name;
                        listBoxMessages.Items.Add("\t" + ++count + " " + sName);
                    }
                }
            }
        }

        // handler for the "Shutdown UModel" button
        // shut-down application instance by explicitly releasing the COM
object.
        private void shutdownUModel_Click(object sender, EventArgs e)
        {
            if (UModel != null)
            {
                // allow shut-down of UModel by releasing UI
                UModel.Visible = false;

                // explicitly release COM object
                try
                {
                    while
(System.Runtime.InteropServices.Marshal.ReleaseComObject(UModel) > 0) ;
                }
                finally
                {
                    // avoid later access to this object.
                    UModel = null;
```

```
                }
            }
        }

        // handler for button "Generate Documentation"
        private void generateDocumentation_Click(object sender, EventArgs e)
        {
            if (UModel == null)
                listBoxMessages.Items.Add("start UModel first.");
            // COM errors get returned to C as exceptions. We use a try/catch
block to handle them.
            try
            {
                // generate documentation
             UModelLib.Dialogs dlgs = UModel.Dialogs;
                UModelLib.ENUMDocumentationOutputFormat docFormat =
UModelLib.ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML;
             UModelLib.GenerateDocumentationDlg docDlg =
dlgs.GenerateDocumentationDlg;
                docDlg.OutputFormat = docFormat;
                String strDocOutputFolder = strExamplesFolder +
"GeneraredDocFromCExample\\";
                if (!Directory.Exists(strDocOutputFolder))
                {
                    Directory.CreateDirectory(strDocOutputFolder);
                    listBoxMessages.Items.Add("Created folder \"" +
strDocOutputFolder + "\"");
                }
                String strResultFile = strDocOutputFolder +
docName.Replace(".ump", ".html"); ;
                listBoxMessages.Items.Add("Generating documentation...");
                UModelDocument.GenerateDocumentation(docDlg, strResultFile);
                listBoxMessages.Items.Add("Generated document: " +
strResultFile);
            }
            catch (Exception ex)
            {
                // The COM call was not successful.
                // Probably no application instance has been started or no
document is open.
                MessageBox.Show("COM error: " + ex.Message);
            }
        }

        delegate void addListBoxItem_delegate(string sText);
        // called from the UI thread
        private void addListBoxItem(string sText)
        {
            listBoxMessages.Items.Add(sText);
        }
        // wrapper method to allow to call UI controls methods from a worker
thread
        void syncWithUIthread(Control ctrl, addListBoxItem_delegate
methodToInvoke, String sText)
        {
            // Control.Invoke: Executes on the UI thread, but calling thread
waits for completion before continuing.
```

```
            // Control.BeginInvoke: Executes on the UI thread, and calling
thread doesn't wait for completion.
            if (ctrl.InvokeRequired)
                ctrl.BeginInvoke(methodToInvoke, new Object[] { sText });
        }

        // event handler for OnDocumentOpened event
        private void handleOnDocumentOpened(UModelLib.Document i_ipDocument)
        {
            String sText = "";

            if (i_ipDocument.Name.Length > 0)
                sText = "Document " + i_ipDocument.Name + " was opened!";
            else
                sText = "An empty document was created.";

            // we need to synchronize the calling thread with the UI thread
because
            // the COM events are triggered from a working thread
            addListBoxItem_delegate methodToInvoke = new
addListBoxItem_delegate(addListBoxItem);
            // call syncWithUIthread with the following arguments:
            // 1 - listBoxMessages - list box control to display messages from
COM events
            // 2 - methodToInvoke  - a C delegate which points to the method
which will be called from the UI thread
            // 3 - sText           - the text to be displayed in the list box
            syncWithUIthread(listBoxMessages, methodToInvoke, sText);
        }

        private void checkBoxEventOnOff_CheckedChanged(object sender, EventArgs
e)
        {
            if (UModel != null)
            {
                if (checkBoxEventOnOff.Checked)
                    UModel.OnDocumentOpened += new
UModelLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpene
d);
                else
                    UModel.OnDocumentOpened -= new
UModelLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpene
d);
            }
        }
    }
}
```

Java

The Application API can be accessed from Java code. To allow accessing the UModel automation server directly from Java code, the libraries listed below must reside in the `classpath`. They are installed in the folder: `JavaAPI` in the UModel application folder.

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers

 © 2014 Altova GmbH

- `UModelAPI.jar`: Java classes that wrap the UModel automation interface
- `UModelAPI_JavaDoc.zip`: a `Javadoc` file containing help documentation for the Java API

Note: In order to use the Java API, the DLL and Jar files must be on the Java Classpath.

Example Java project
An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. For more details of the example Java project, see the section, Example Java Project.

Rules for mapping the Application API names to Java
The rules for mapping between the Application API and the Java wrapper are as follows:

- ***Classes and class names***
  For every interface of the UModel automation interface a Java class exists with the name of the interface.

- ***Method names***
  Method names on the Java interface are the same as used on the COM interfaces but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with `get` and `set` can be used. If a property does not support write-access, no setter method is available. Example: For the `Name` property of the `Document` interface, the Java methods `getName` and `setName` are available.

- ***Enumerations***
  For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.

- ***Events and event handlers***
  For every interface in the automation interface that supports events, a Java interface with the same name plus '`Event`' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus '`DefaultHandler`'. For example:
  `Application`: Java class to access the application
  `ApplicationEvents`: Events interface for the Application
  `ApplicationEventsDefaultHandler`: Default handler for `ApplicationEvents`

The following section explains how some basic UModel functionality can be accessed from Java code.

- Example Java Project

Example Java Project

The UModel installation package contains an example Java project, located in the Java folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/

This folder contains Java examples for the UModel API. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list
The Java examples folder contains all the files required to run the example project. These files are listed below:

`AltovaAutomation.dll`	Java-COM bridge: DLL part
`AltovaAutomation.jar`	Java-COM bridge: Java library part
`UModelAPI.jar`	Java classes of the UModel API
`RunUModel.java`	Java example source code
`BuildAndRun.bat`	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
`.classpath`	Eclipse project helper file
`.project`	Eclipse project file
`UModelAPI_JavaDoc.zip`	`Javadoc` file containing help documentation for the Java API
`Readme.txt`	This file

What the example does
The example starts up UModel and performs a few operations, including opening and closing documents. When done, UModel stays open. You must close it manually.

Running the example from the command line
To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a JDK 1.5 or later installation on your computer.

Press the **Return** key. The Java source in `RunUModel.java` will be compiled and then executed.

Loading the example in Eclipse
Open Eclipse and use the **File | Import... | General | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `RunUModel` will then appear in your Package Explorer or Navigator.

 © 2014 Altova GmbH

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1to get help for that class or method.

Java source code listing

The Java source code in the example file `RunUModel.java` is listed below with comments.

```
// access general JAVA-COM bridge classes
import java.io.File;
import java.util.Iterator;

import com.altova.automation.libs.*;

// access XMLSpy Java-COM bridge
import com.altova.automation.UModel.*;
import com.altova.automation.UModel.Enums.ENUMDocumentationOutputFormat;

/**
 * A simple example that starts UModel COM server and performs a few operations
on it.
 * Feel free to extend.
 */
public class RunUModel
{
   public static void main(String[] args)
   {
      // an instance of the application.
      Application umodel = null;

      // instead of COM error handling use Java exception mechanism.
      try
      {
         // Start UModel as COM server.
         umodel = new Application();
         // COM servers start up invisible so we make it visible
         umodel.setVisible(true);

         String docName = "Bank_MultiLanguage.ump";

         // The following lines attach to the application events using a default
implementation
         // for the events and override one of its methods.
         // If you want to override all application events it is better to
derive your listener class
         // from ApplicationEvents and implement all methods of this interface.
         umodel.addListener(new ApplicationEventsDefaultHandler()
         {
            @Override
            public void onDocumentOpened(Document i_ipDoc) throws
AutomationException
            {
               String name = i_ipDoc.getName();
```

```
                System.out.println("Document " + name + " was opened.");
            }
        });

        // Locate samples installed with the product.
        String strExamplesFolder = System.getenv("USERPROFILE") + "\\My
Documents\\Altova\\UModel2012\\UModelExamples\\";
        Document doc = umodel.openDocument(strExamplesFolder + docName);

        // The following lines attach to the document events using a default
implementation
        // for the events and override one of its methods.
        // If you want to override all document events it is better to derive
your listener class
        // from DocumentEvents and implement all methods of this interface.
        doc.addListener(new DocumentEventsDefaultHandler()
        {
            @Override
            public void onDiagramWindowOpened(DiagramWindow ipDiagram) throws
AutomationException
            {
                String name = ipDiagram.getName();

                System.out.println("Diagram " + name + " was opened.");
            }
        });
        // open all diagrams
        doc.openAllDiagrams();

        DiagramWindows diagramWindows = doc.getDiagramWindows();

        Iterator<DiagramWindow> itr = diagramWindows.iterator();

        if (diagramWindows.getCount() == 0)
            System.out.print("No opened diagrams.\n");
        else
            System.out.print("Opened diagrams:\n");

        int count = 0;
        while (itr.hasNext())
        {
            // iterate through the opened diagram windows
            String sName = "";
             Object element = itr.next();
             if (element instanceof DiagramWindow)
             {
                sName = ((DiagramWindow)element).getName();
                System.out.print("\t" + ++count + " " + sName + "\n");
             }
        }

        // generate documentation
        Dialogs dlgs = umodel.getDialogs();
        ENUMDocumentationOutputFormat docFormat =
ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML;
        GenerateDocumentationDlg docDlg = dlgs.getGenerateDocumentationDlg();
        docDlg.setOutputFormat(docFormat);
```

 © 2014 Altova GmbH

```
         String strDocOutputFolder = strExamplesFolder + "Generared
documentation\\Java\\";
         if ((new File(strDocOutputFolder)).mkdirs())
            System.out.println("Created folder \"" + strDocOutputFolder + "\"");
         String strResultFile = strDocOutputFolder + docName.replace(".ump",
".html");
         System.out.print("Generating documentation...\n");
         doc.generateDocumentation(docDlg, strResultFile);
         System.out.print("Generated document: " + strResultFile + "\n");

         System.out.println("Watch UModel!");
      }
      catch (AutomationException e)
      {
         // e.printStackTrace();
      }
      finally
      {
         // Make sure that UModel can shut down properly.
         if (umodel != null)
            umodel.dispose();

         // Since the COM server was made visible and still is visible, it will
keep running
         // and needs to be closed manually.
         System.out.println("Now close UModel!");
      }
   }
}
```

JScript

This section contains listings of JScript code that demonstrate the following basic functionality:

Example files

The code is available in example files that you can test as is or modify to suit your needs. The JScript example files are located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/Altova/UModel2015/UModelExamples/API/`
Windows Vista, Windows 7	`C:/Users/<username>/Documents/Altova/UModel2015/UModelExamples/API/`

The example files can be run in one of two ways:

- *From the command line:*
  Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script.
- *From Windows Explorer:*
  In Windows Explorer, browse for the JScript file and double-click it. The Windows

Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script. After the script is executed, the command console gets closed automatically.

`DocumentAccess.js`	Shows how to open, iterate and close documents
`GenerateDoc.js`	Shows how to generate documentation using JScript
`Start.js`	Start UModel as a registered automation server, or connects to a running instance.
`UModelCreateCode.js`	Shows forward engineering to create code elements in a new UModel project and then generate code.
`UModelUpdateDocumentation.js`	reverse engineers all C# UModelAPI samples, and creates HTML and RTF documentation.
`Readme.txt`	This file.

Start application

The JScript below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

Script listing - Start.js

The JScript listing below is explained with comments in the code.

```
// 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)
{
   Exit("Can't access or create UModel.Application");
}

// 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
```

 © 2014 Altova GmbH

Running the script
The JScript code listed above is available in the file `Start.js` located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/`
Windows Vista, Windows 7	`C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/`

To run the script, start it from a command prompt window or from Windows Explorer.

Document Access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

Script listing - DocumentAccess.js
The JScript listing below is explained with comments in the code.

```
// 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)
{
   Exit("Can't access or create UModel.Application");
}

// 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 via USERPROFILE shell variable. The path needs to be adapted
to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\UModel2012\\UModelExamples\\";

objDoc = objUModel.OpenDocument(strExampleFolder + "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
```

Generate documentation

The JScript listing below shows how to generate documentation for the Bank_MultiLanguage.ump file in the UModelExamples folder.

Script listing - GenerateDoc.js
The JScript listing below is explained with comments in the code.

```
// 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)
{
   Exit("Can't access or create UModel.Application");
}

// if newly started, the application will start without its UI visible. Set it
to visible.
objUModel.Visible = true;
```

 © 2014 Altova GmbH

```
// Locate examples via USERPROFILE shell variable. The path needs to be adapted
to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExamplesFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\
\My Documents\\Altova\\UModel2012\\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 + "GeneraredDocWithJScriptExample\\";

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
```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/`
Windows Vista, Windows 7	`C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/`

To run the script, start it from a command prompt window or from Windows Explorer.

Generate code

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and generate code.

Script listing - UModelCreateCode.js

The JScript listing below is explained with comments in the code.

```
// ##########################################
// 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)
   {
      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)
   {
      Exit("Can't access or create UModel.Application");
   }
}
```

 © 2014 Altova GmbH

```
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 an 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
      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
```

 © 2014 Altova GmbH

```
      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_UserDefinedSPLTemplatesOver
rideDefault = 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
```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the `JScript` folder of the API Examples folder:

Windows XP	`C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/`
Windows Vista, Windows 7	`C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/`

To run the script, start it from a command prompt window or from Windows Explorer.

Update Documentation

The JScript listing below shows how to:

- reverse engineer all C# UModelAPI samples — If a UMP file does not exist
- create HTML and RTF documentation
- Export as XMI and save the *.UMP file

or,

- open the UMP file and synchronize model from code — If a UMP file exists
- create HTML and RTF documentation only if some data has been changed

Script listing - UModelUpdateDocumentation.js

The JScript listing below is explained with comments in the code.

```
// ##########################################
// access runing UModel.Application or
// launch new one and access it
// ###########################################
```

 © 2014 Altova GmbH

```
// ##########################################
// 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;
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)
   {
      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;
```

 © 2014 Altova GmbH

```
      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;
}

// ////////////////////// 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
   {
      // 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
```

 © 2014 Altova GmbH

```
      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
```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2015/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2015/UModelExamples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

© 2014 Altova GmbH

19.4.2 Usage Examples

The automation interface of UModel can be used in many different ways and accessed from different programming languages. In this section you will find sample code that show the capabilities of the new UModel API interfaces.

These samples should make it easier for you to get started with your own project. Additional code snippets can be found in the How to... section of the Overview.

The listed code samples are:

UModel IDE plug-in samples (written in C#):

- Styles sets multiple styles for selected diagram elements
- CSharpDelegate inserts a new C# delegate at the current scroll position on the active diagram window
- DefaultPrefix automatically sets a prefix when new attributes or enumerationliterals are added to the UModel project

UModel IDE plug-in ActiveX sample (written in C#):

- StatisticsActiveX counts the elements of the different element kinds after every data modification (transaction) and shows the result in an ActiveX control

Scripting samples (written in JScript):

- CreateCode creates a new UModel project, creates some classes and generates code
- UpdateDocumentation reverse engineers all UModel API samples and creates HTML and RTF documentation as well as an XMI export of the project

Deployment sample:

- SetupStyles creates an installer for deploying the Styles IDE plug-in sample

Styles (C#)

The following C# IDE plug-in sample sets multiple styles for selected diagram elements (if style is available and not already set).

Keywords: UModel IDE plug-in, active document, active diagram, diagram selection, elements on diagrams, undo-transaction handling, styles, styles enumeration, verification of API version

File: UModelExamples\IDEPlugIn\Styles\Styles.cs:

```
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()
        {
            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;
```

 © 2014 Altova GmbH

```
            }
        }

        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 != 2) // this
plug-in was compiled for API major version '2'!
            {
                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 )
            {
                // 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
    }
}
```

SetupStyles (Deployment)

The SetupStyles sample creates an installer for deploying the Styles IDE plug-in sample.

The setup installs all necessary files, registers the IDE plug-in for COM and UModel on your target system, that the Styles plug-in is automatically loaded when UModel is started the next time.

File: UModelExamples\IDEPlugIn\Styles\SetupStyles\SetupStyles.sln

CSharpDelegate (C#)

The following C# IDE plug-in sample inserts a new C# delegate at the top/left corner of the active diagram window (if this diagram is inside a C# namespace root)

Keywords: UModel IDE plug-in, active diagram; diagram scroll positions, setting diagram selection, check for C# namespace root; add new data on diagram; applying stereotypes and setting tagged values; predefined UModel elements, undo-transaction handling

File: UModelExamples\IDEPlugIn\CSharpDelegate\UModelCSharpDelegate.cs:

```
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();
        }
```

 © 2014 Altova GmbH

```
        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;

            // 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
```

 © 2014 Altova GmbH

```
            IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc
.RootPackage
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement.ePredefined_CSharp_Profile, 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_attributesStereotypeOfClass);
                IUMLEnumerationLiteral iSTAThread =
(IUMLEnumerationLiteral)
iCSharpProfile
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement
.ePredefined_CSharp_AttributePresetsEnumeration_STAThreadEnumerationLi
teral, 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_voidPrimitiveType, 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
            }
        }

        #endregion
    }
}
```

DefaultPrefix (C#)

The following C# IDE plug-in sample automatically sets a prefix when new attributes or enumerationliterals are added to your UModel project.

Keywords: UModel IDE plug-in, application events, UMLData events, UMLData event filter, connection points and delegates

File: UModelExamples\IDEPlugIn\DefaultPrefix\DefaultPrefix.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Runtime.InteropServices.ComTypes;
using UModelLib;
using UModelPlugInLib;

/*
 * DefaultPrefix sample
```

 © 2014 Altova GmbH

```
 * 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)
        {
            // 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;
            }
        }
```

 © 2014 Altova GmbH

```
        #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
            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 children) 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);
                        }
                    }
                }
                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 "";
        }
    }
}
```

CreateCode (JScript)

The following JScript sample creates a new UModel project, creates some classes and generates code.

Topics: loading include files, saving UModel projectfiles, namespace root handling, using predefined elements and stereotypes, applying profiles and stereotypes, setting tagged values, creating new UML elements, code from model synchronization (forward engineering)

File: UModelExamples\API\CreateCode\UModelCreateCode.js:

```
// ##########################################
// 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)
	{
		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)
	{
		Exit("Can't access or create UModel.Application");
	}
}

function GetSourceCodeDirectory()
{
	// get directory for source code
	var path =  objUModel.PersonalDataDirectory + "\\UModelExamples\
```

```
\API\\CreateCode";
      var codeDirectory = objFSO.BuildPath( path, "SampleCode" );
      return codeDirectory;
}

function GetUMPFilePath()
{
      // get file path to save UModel projectfile
      return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\
\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 an 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_STAThreadEnumerationLit
eral = 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
            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_UserDefinedSPLTemplatesOverrideDefault = 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
```

UpdateDocumentation (JScript)

The following JScript sample reverse engineers all UModel API samples and creates HTML and RTF documentation as well as an XMI export of the UModel project.

Topics: start Umodel from script; reverse engineering dialog; documentation generation dialog

File: UModelExamples\API\UpdateDocumentation\UModelUpdateDocumentation.js:

```
// ##########################################
```

```
// 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;
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)
	{
		Exit("Can't access or create UModel.Application");
	}
}

// ////////////////////////// get different filepathes / ensure folders
are created  ////////////////////////////////
function GetScriptPath()
{
	return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\
\UpdateDocumentation";
}

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;
}

// ////////////////////////// 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                    =
objFSO.GetParentFolderName( GetScriptPath() );
            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
      {
            // 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 =
```

 © 2014 Altova GmbH

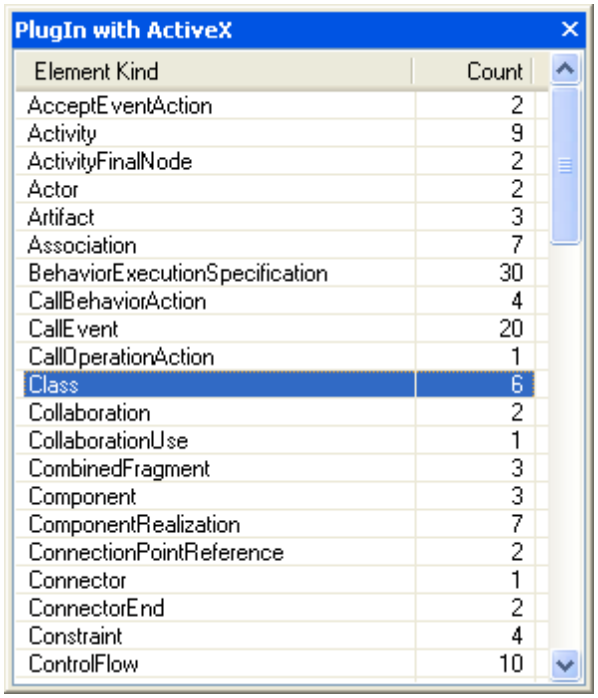
```
true;
            objIDocumentationGenerationDlg.Include_UnknownExternals =
false;

            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
```

StatisticsActiveX (C#)

The following C# IDE plug-in sample listens for data modifications and counts the elements of the different element kinds.
Since the plug-in derives from `System.Windows.Forms.UserControl` it also acts as an ActiveX control and the results can be shown in a Dialog Control Bar inside UModel:

 © 2014 Altova GmbH



PlugIn with ActiveX

Element Kind	Count
AcceptEventAction	2
Activity	9
ActivityFinalNode	2
Actor	2
Artifact	3
Association	7
BehaviorExecutionSpecification	30
CallBehaviorAction	4
CallEvent	20
CallOperationAction	1
Class	6
Collaboration	2
CollaborationUse	1
CombinedFragment	3
Component	3
ComponentRealization	7
ConnectionPointReference	2
Connector	1
ConnectorEnd	2
Constraint	4
ControlFlow	10

Keywords: UModel IDE plug-in, ActiveX, active document, transaction events

File: UModelExamples\IDEPlugIn\StatisticsActiveX\StatisticsActiveX.cs:

```
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;

/*
 * 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()
        {
            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)
```

 © 2014 Altova GmbH

```
                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)
                    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
    }
}
```

 © 2014 Altova GmbH

19.5 UModel API Reference

This section describes all interfaces, operations, enumerations and events of UModel Plugins and of the UModel API.

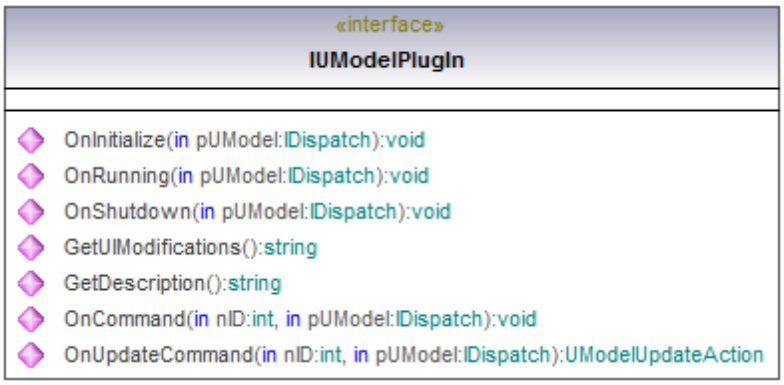
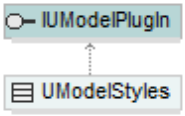
19.5.1 UModel Plugins

UModel allows you to create your own plug-ins and integrate them into UModel (also see how UModel interacts with IUModelPlugin).

A detailed description about how to create a plug-in can be found in section How to create a plug-in in MS Visual Studio .NET.

UModelAPI - IUModelPlugIn

Interface **IUModelPlugIn**

diagram	
hierarchy	

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**

 © 2014 Altova GmbH

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**

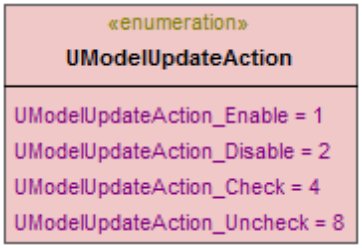
parameter	name	direction	type	type modifier	multiplicity	default
	nID	**in**	**int**			
	pUModel	**in**	**IDispatch**			
	return	**return**	**UModelUpdateAction**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - UModelUpdateAction

Enumeration **UModelUpdateAction**

diagram



typedElements		
	Interface **IUModelPlugIn**	Operation **OnUpdateCommand**
	Class **UModelStyles**	Operation **OnUpdateCommand**
		OnUpdateSetStyles

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

19.5.2 UModel API

This chapter contains the reference of the UModel 4.1 Type Library.

Errors

1000 The application object is no longer valid.
1001 Invalid parameter or invalid address for the return parameter was specified.
1002 UModel API is not available in the current edition.
1003 Model Transformations are not supported in the current edition.

1050 Macro not found
1051 Invalid (nested) macro execution

1100 Error when saving file, probably the file name is invalid.
1101 Invalid (duplicate) call to BeginModification.
1102 EndModification called without BeginModification

1200 Error deleting file at URL.
1201 Error creating directory at URL.

Errors of UMLData can be found here.

 © 2014 Altova GmbH

UModelAPI - IApplication

Interface **IApplication**

diagram	«interface» **IApplication** Quit():void OpenDocument(in strModelFilePath:string):IDocument OpenDocumentFromURL(in IURLDlg:IURLDlg):IDocument ImportFromXMIFile(in strXMIFile:string):IDocument ImportFromXMIFileFromURL(in IURLDlg:IURLDlg):IDocument NewDocument():IDocument RunMacro(in strName:string):void «GetAccessor, property» Name():string «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» Visible():bool «GetAccessor, property» WindowHandle():int «GetAccessor, property» ActiveDocument():IDocument «GetAccessor, property» Status():ENUMApplicationStatus «GetAccessor, property» MajorVersion():int «GetAccessor, property» MinorVersion():int «GetAccessor, property» Edition():string «GetAccessor, property» IsAPISupported():bool «GetAccessor, property» APIMajorVersion():int «GetAccessor, property» APIMinorVersion():int «GetAccessor, property» InstallationDirectory():string «GetAccessor, property» PersonalDataDirectory():string «GetAccessor, property» Dialogs():IDialogs «GetAccessor, property» Options():ILocalOptions «GetAccessor, property» ServicePackVersion():int
typedElements	Class **UModelStyles** Operation **OnSetStyles** **OnUpdateSetStyles**

Operation **IApplication::ActiveDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDocument**			

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**			

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**			

Operation **IApplication::ImportFromXMIFileFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**IDocument**			

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::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**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDocument**			

Operation **IApplication::OpenDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	strModelFilePath		**string**			
	return	**return**	**IDocument**			

Operation **IApplication::OpenDocumentFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**IDocument**			

Operation **IApplication::Options**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptions**			

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**			

Operation **IApplication::Visible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IApplication::WindowHandle**

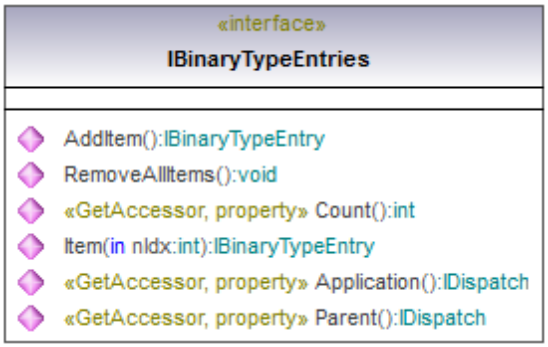
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IBinaryTypeEntries

Interface **IBinaryTypeEntries**

diagram



typedElements	Interface	Operation
	IImportBinaryTypesDlg	**CSharp_BinaryTypes** **Java_BinaryTypes** **VBasic_BinaryTypes**

Operation **IBinaryTypeEntries::AddItem**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IBinaryTypeEntry**			

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**			

Operation **IBinaryTypeEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

 © 2014 Altova GmbH

Operation **IBinaryTypeEntries::RemoveAllItems**

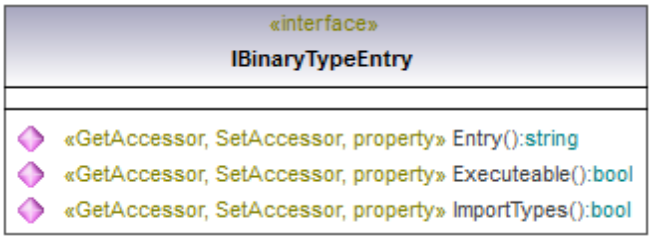
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IBinaryTypeEntry

Interface **IBinaryTypeEntry**

diagram



typedElements: Interface **IBinaryTypeEntries** Operation **AddItem** **Item**

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**

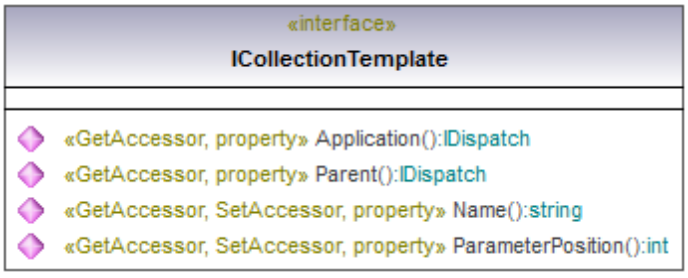
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ICollectionTemplate

Interface **ICollectionTemplate**

diagram



typedElements	Interface **ICollectionTemplates**	Operation **InsertItemAt** **Item**

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**			

Operation **ICollectionTemplate::Parent**

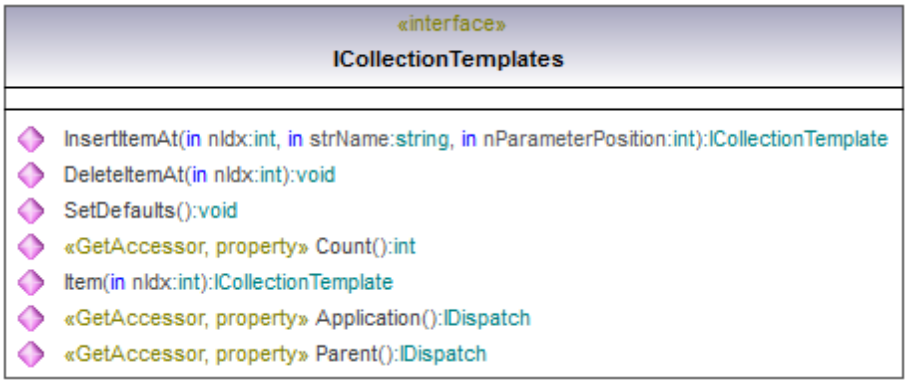
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ICollectionTemplates

Interface **ICollectionTemplates**

diagram



typedElements	Interface **ILocalOptionsDiagramEditing**	Operation **CollectionTemplates_CSharp** **CollectionTemplates_Java** **CollectionTemplates_UML** **CollectionTemplates_VBasic**

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**			

Operation **ICollectionTemplates::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**ICollectionTemplate**			

Operation **ICollectionTemplates::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ICollectionTemplates::SetDefaults**

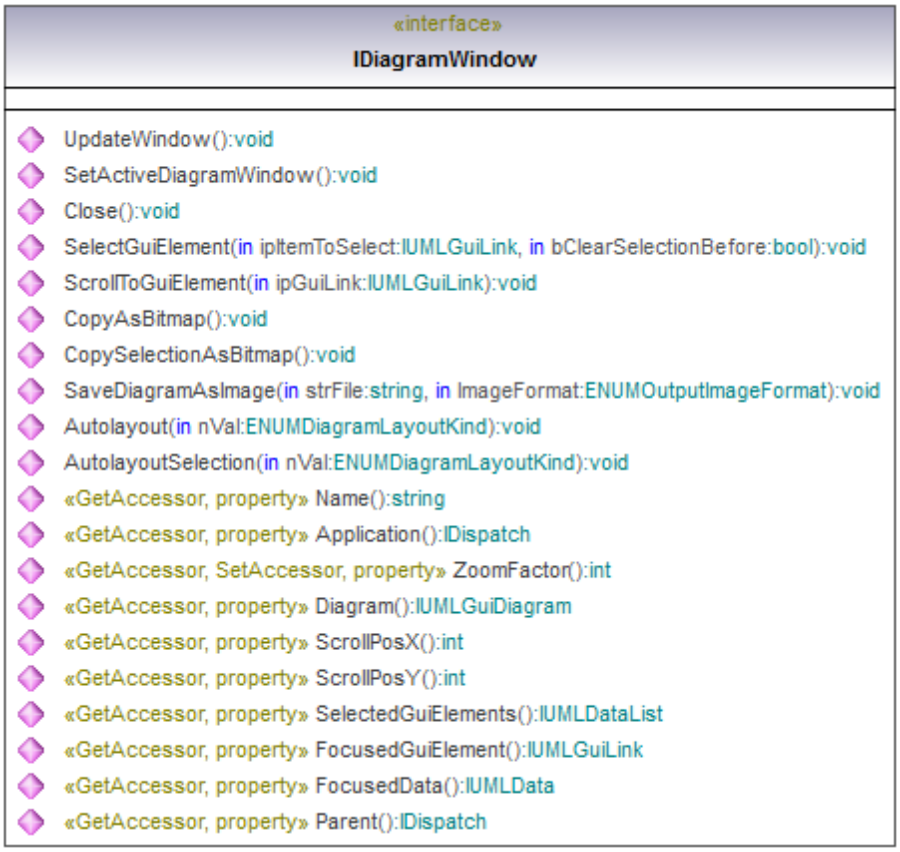
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IDiagramWindow

Interface **IDiagramWindow**

diagram



typedElements		
	Interface **_IDiagramWindowEvents**	Operation **OnDiagramWindowClosed**
	Interface **_IDocumentEvents**	Operation **OnActivateDiagramWindow** **OnDiagramWindowClosed** **OnDiagramWindowOpened**
	Interface **IDiagramWindows**	Operation **Item**
	Interface **IDocument**	Operation **ActiveDiagramWindow** **OpenDiagram**

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**			
	return	**return**	**void**			

Operation **IDiagramWindow::AutolayoutSelection**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**nVal**	**in**	**ENUMDiagramLayoutKind**			
	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**			

Operation **IDiagramWindow::FocusedData**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			

Operation **IDiagramWindow::FocusedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**			

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**			
	return	**return**	**void**			

Operation **IDiagramWindow::ScrollPosX**

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**			
	return	**return**	**void**			

Operation **IDiagramWindow::SelectedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGuiElement.					

Operation **IDiagramWindow::SelectGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	ipItemToSelect	**in**	**IUMLGuiLink**			
	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**			

UModelAPI - IDiagramWindows

Interface **IDiagramWindows**

diagram	«interface» IDiagramWindows «GetAccessor, property» Count():int Item(in nIdx:int):IDiagramWindow «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch
typedElements	Interface **IDocument** Operation **DiagramWindows**

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**			

Operation **IDiagramWindows::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IDialog

Interface **IDialog**

diagram	«interface» IDialog «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» ShowDialog():bool

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**

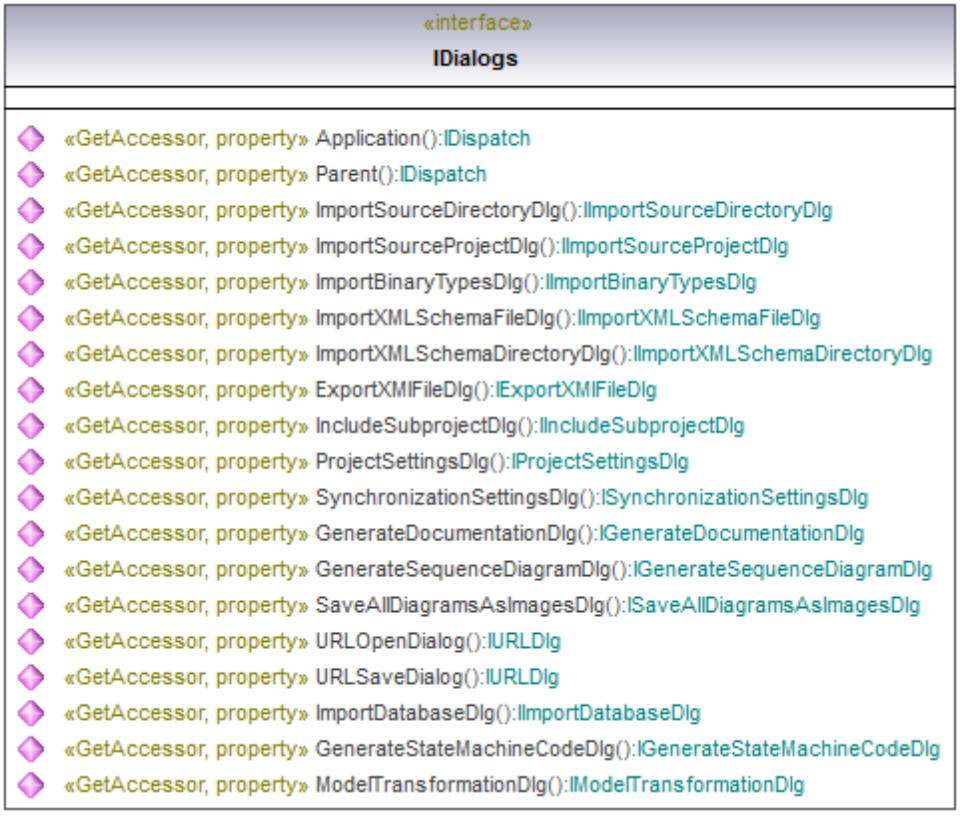
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IDialogs

Interface **IDialogs**

diagram



typedElements	Interface **IApplication**	Operation **Dialogs**

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**			

Operation **IDialogs::GenerateDocumentationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IGenerateDocumentationDlg**			

Operation **IDialogs::GenerateSequenceDiagramDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IGenerateSequenceDiagramDlg**			

Operation **IDialogs::GenerateStateMachineCodeDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IGenerateStateMachineCodeDlg**			

Operation **IDialogs::ImportBinaryTypesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IImportBinaryTypesDlg**			

Operation **IDialogs::ImportDatabaseDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IImportDatabaseDlg**			

Operation **IDialogs::ImportSourceDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IImportSourceDirectoryDlg**			

Operation **IDialogs::ImportSourceProjectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IImportSourceProjectDlg**			

Operation **IDialogs::ImportXMLSchemaDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IImportXMLSchemaDirectoryDlg**			

Operation **IDialogs::ImportXMLSchemaFileDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IImportXMLSchemaFileDlg**			

Operation **IDialogs::IncludeSubprojectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IIncludeSubprojectDlg**			

Operation **IDialogs::ModelTransformationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IModelTransformationDlg**			

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**			

Operation **IDialogs::SaveAllDiagramsAsImagesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ISaveAllDiagramsAsImagesDlg**			

Operation **IDialogs::SynchronizationSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ISynchronizationSettingsDlg**			

Operation **IDialogs::URLOpenDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg**			

Operation **IDialogs::URLSaveDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IURLDlg**			

© 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IDocument

Interface **IDocument**

© 2014 Altova GmbH

diagram

«interface»
IDocument

- Save():void
- SaveAs(in strFileName:string):void
- OpenAllDiagrams():void
- BeginModification():bool
- AbortModification():void
- EndModification():void
- OpenDiagram(in ipUMLDiagram:IUMLGuiDiagram):IDiagramWindow
- ElementFamilyStyles(in strKind:string):IUMLGuiStyles
- CheckProjectSyntax():bool
- ImportSourceDirectory(in ipDlg:IImportSourceDirectoryDlg):bool
- ImportSourceProject(in ipDlg:IImportSourceProjectDlg):bool
- ImportBinaryTypes(in ipDlg:IImportBinaryTypesDlg):bool
- ImportXMLSchemaFile(in ipDlg:IImportXMLSchemaFileDlg):bool
- ImportXMLSchemaDirectory(in ipDlg:IImportXMLSchemaDirectoryDlg):bool
- ExportToXMIFile(in ipDlg:IExportXMIFileDlg):bool
- SynchronizeModelFromCode(in ipDlg:ISynchronizationSettingsDlg):bool
- SynchronizeCodeFromModel(in ipDlg:ISynchronizationSettingsDlg):bool
- IncludeSubproject(in ipDlg:IIncludeSubprojectDlg):bool
- MergeProject(in strProjectFile:string):bool
- MergeProjectFromURL(in IURLDlg:IURLDlg):bool
- SynchronizationSettings(in ipDlg:ISynchronizationSettingsDlg):void
- ProjectSettings(in ipDlg:IProjectSettingsDlg):void
- GenerateDocumentation(in ipDlg:IGenerateDocumentationDlg, in strResultFile:string):bool
- SaveAllDiagramsAsImages(in ipDlg:ISaveAllDiagramsAsImagesDlg):bool
- GenerateSequenceDiagram(in ipDlg:IGenerateSequenceDiagramDlg, in ipOp:IUMLOperation):bool
- SaveToURL(in IURLDlg:IURLDlg):void
- SaveCopyAs(in strFileName:string):void
- CanFocusUMLDataInModelTree(in ipUMLData:IUMLData):bool
- FocusUMLDataInModelTree(in ipUMLData:IUMLData, in bFocusModelTree:bool, in bEnsureModelTreeVisible:bool):void
- Reload():bool
- ImportDatabase(in ipDlg:IImportDatabaseDlg):bool
- MergeProject3Way(in strProjectFile:string, in strCommonAncestorFile:string):bool
- GenerateStateMachineCode(in ipDlg:IGenerateStateMachineCodeDlg, in ipStateMachine:IUMLStateMachine):bool
- ModelTransformation(in ipDlg:IModelTransformationDlg):bool
- GenerateSequenceDiagramsForAllOperations(in bAllPublicOnly:bool, in bIncludeGettersAndSetters:bool):void
- «GetAccessor, property» Application():IDispatch
- «GetAccessor, property» Parent():IDispatch
- «GetAccessor, SetAccessor, property» FullName():string
- «GetAccessor, property» Name():string
- «GetAccessor, property» Path():string
- «GetAccessor, property» IsLoadedFromPreviousFileFormat():bool
- «GetAccessor, property» Saved():bool
- «GetAccessor, property» DiagramWindows():IDiagramWindows
- «GetAccessor, property» TransactionNotifier():ITransactionNotifier
- «GetAccessor, SetAccessor, property» ModifiedFlag():bool
- «GetAccessor, property» RootPackage():IUMLPackage
- «GetAccessor, property» GuiRoot():IUMLGuiRootElement
- «GetAccessor, property» IsInUndoRedo():bool
- «GetAccessor, property» FocusedUMLData():IUMLData
- «GetAccessor, property» ActiveDiagramWindow():IDiagramWindow
- «GetAccessor, property» ProjectStyles():IUMLGuiStyles
- «GetAccessor, property» NodeStyles():IUMLGuiStyles
- «GetAccessor, property» LineStyles():IUMLGuiStyles
- «GetAccessor, property» FocusedUMLDataNotifier():IFocusedUMLDataNotifier

typedElements	Interface _IApplicationEvents	Operation	OnDocumentClosed OnDocumentOpened OnNewDocument
	Interface _IDocumentEvents	Operation	OnAfterReloadDocument OnBeforeReloadDocument OnDocumentClosed OnDocumentSaved OnDocumentSavedAs OnModifiedFlagChanged
	Interface _ITransactionEvents	Operation	OnBeginDataModification OnEndDataModification
	Interface IApplication	Operation	ActiveDocument ImportFromXMIFile ImportFromXMIFileFromURL NewDocument OpenDocument OpenDocumentFromURL

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			

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			
	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			

Operation **IDocument::ElementFamilyStyles**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiStyles**			

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**			
	return	**return**	**bool**			

Operation **IDocument::FocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			
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**			

Operation **IDocument::FocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	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**			
	strResultFile	**in**	**string**			
	return	**return**	**bool**			

Operation **IDocument::GenerateSequenceDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IGenerateSequenceDiagramDl**			

			g			
	ipOp	in	IUMLOperation			
	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			
	ipStateMachine	in	IUMLStateMachine			
	return	return	bool			

Operation **IDocument::GuiRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiRootElement			

Operation **IDocument::ImportBinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportBinaryTypesDlg			
	return	return	bool			

Operation **IDocument::ImportDatabase**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportDatabaseDlg			
	return	return	bool			

Operation **IDocument::ImportSourceDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportSourceDirectoryDlg			
	return	return	bool			

Operation **IDocument::ImportSourceProject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportSourceProjectDlg			
	return	return	bool			

Operation **IDocument::ImportXMLSchemaDirectory**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportXMLSchemaDirectoryDlg**			
	return	**return**	**bool**			

Operation **IDocument::ImportXMLSchemaFile**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IImportXMLSchemaFileDlg**			
	return	**return**	**bool**			

Operation **IDocument::IncludeSubproject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IIncludeSubprojectDlg**			
	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**			

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**			
	return	**return**	**bool**			

Operation **IDocument::MergeProjectFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
	IURLDlg	**in**	**IURLDlg**			
	return	**return**	**bool**			

Operation **IDocument::ModelTransformation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**IModelTransformationDlg**			
	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**			

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**			
	return	**return**	**IDiagramWindow**			

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**			
	return	**return**	**void**			

Operation **IDocument::ProjectStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IDocument::Reload**

 © 2014 Altova GmbH

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**			

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**			
	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**			
	return	**return**	**void**			

Operation **IDocument::SynchronizationSettings**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg**			
	return	**return**	**void**			

Operation **IDocument::SynchronizeCodeFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg**			
	return	**return**	**bool**			

Operation **IDocument::SynchronizeModelFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	**in**	**ISynchronizationSettingsDlg**			
	return	**return**	**bool**			

Operation **IDocument::TransactionNotifier**

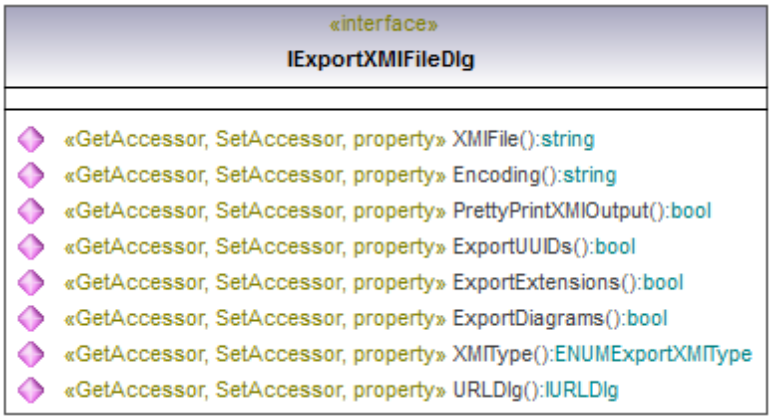
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ITransactionNotifier**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

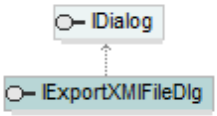
UModelAPI - IExportXMIFileDlg

Interface **IExportXMIFileDlg**

diagram



hierarchy



typedElements		
	Interface **IDialogs**	Operation **ExportXMIFileDlg**
	Interface **IDocument**	Operation **ExportToXMIFile**

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**

 © 2014 Altova GmbH

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**			

Operation **IExportXMIFileDlg::XMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

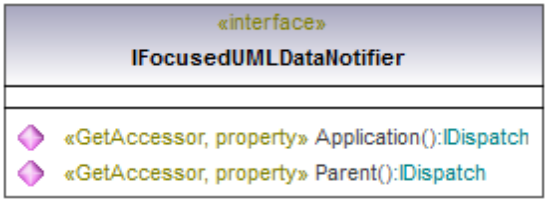
Operation **IExportXMIFileDlg::XMIType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMExportXMIType**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IFocusedUMLDataNotifier

Interface **IFocusedUMLDataNotifier**

diagram	
typedElements	Interface **IDocument** Operation **FocusedUMLDataNotifier**

Operation **IFocusedUMLDataNotifier::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

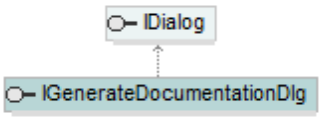
Operation **IFocusedUMLDataNotifier::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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 (http://manual.altova.com/UModel/umodelprofessional/).]***
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **GenerateDocumentationDlg** Operation **GenerateDocumentation**

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**			

 © 2014 Altova GmbH

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
	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**			

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

© 2014 Altova GmbH

r	**return**	**return**	**bool**

Operation **IGenerateDocumentationDlg::Fonts_GetFace**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**string**			

Operation **IGenerateDocumentationDlg::Fonts_GetSize**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Fonts_GetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**int**			

Operation **IGenerateDocumentationDlg::Fonts_IsBold**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_IsItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_IsUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	return	**return**	**bool**			

Operation **IGenerateDocumentationDlg::Fonts_SetBold**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	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**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetSize**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	nNewVal	**in**	**int**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::Fonts_SetUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	**in**	**ENUMDocumentationFontSetting**			
	bVal	**in**	**bool**			
	return	**return**	**void**			

Operation **IGenerateDocumentationDlg::GenerateLinksToLocalFiles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDocumentationFilePathKind**			

© 2014 Altova GmbH

Operation **IGenerateDocumentationDlg::Include_Diagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IKindSelectionList**			

Operation **IGenerateDocumentationDlg::Include_Elements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IKindSelectionList**			

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
	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**	**ENUMDocumentationOutputFormat**			

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**			

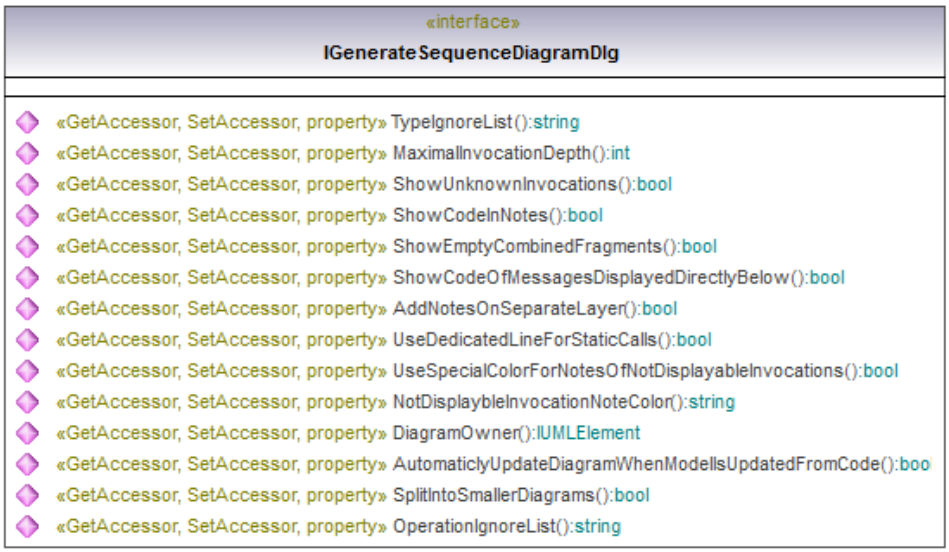
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

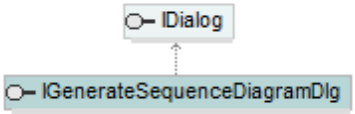
UModelAPI - IGenerateSequenceDiagramDlg

Interface **IGenerateSequenceDiagramDlg**

diagram



hierarchy



typedElements

Interface **IDialogs**
Interface **IDocument**
Operation **GenerateSequenceDiagramDlg**
Operation **GenerateSequenceDiagram**

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**			

Operation **IGenerateSequenceDiagramDlg::MaximalInvocationDepth**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IGenerateSequenceDiagramDlg::NotDisplayableInvocationNoteColor**

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**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - IGenerateStateMachineCodeDlg

Interface **IGenerateStateMachineCodeDlg**

diagram	«interface» IGenerateStateMachineCodeDlg «GetAccessor, SetAccessor, property» GenerateDebugMessages():bool «GetAccessor, SetAccessor, property» GetCallEvents():bool «GetAccessor, SetAccessor, property» IRegion_GetName():bool «GetAccessor, SetAccessor, property» IRegion_GetStates():bool «GetAccessor, SetAccessor, property» IState_GetName():bool «GetAccessor, SetAccessor, property» IState_GetId():bool «GetAccessor, SetAccessor, property» IState_GetRegions():bool «GetAccessor, SetAccessor, property» AdditionalImportsAndDeclarations():string «GetAccessor, SetAccessor, property» AutomaticallyUpdateStateMachineCode():boo
hierarchy	IDialog IGenerateStateMachineCodeDlg
typedElements	Interface **IDialogs** Interface **IDocument** Operation **GenerateStateMachineCodeDlg** Operation **GenerateStateMachineCode**

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** 09/04/12 14:02:42

© 2014 Altova GmbH

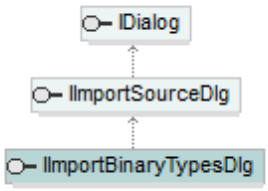
UModelAPI - IImportBinaryTypesDlg

Interface **IImportBinaryTypesDlg**

diagram

«interface»
IImportBinaryTypesDlg

- «GetAccessor, SetAccessor, property» Runtime():string
- «GetAccessor, SetAccessor, property» Content_GenerateSingleDiagram():bool
- «GetAccessor, SetAccessor, property» Content_GenerateDiagramPerPackage():bool
- «GetAccessor, SetAccessor, property» Content_ShowNestedClassifiersSeparately():bool
- «GetAccessor, SetAccessor, property» Content_ShowAnonymousBoundElements():bool
- «GetAccessor, SetAccessor, property» Content_HyperlinkPackagesToDiagrams():bool
- «GetAccessor, SetAccessor, property» Java_OverridePathForNativeLibraries():string
- «GetAccessor, SetAccessor, property» Java_ImportReferencedTypes():bool
- «GetAccessor, SetAccessor, property» Java_ImportReferencedTypesRestriction():string
- «GetAccessor, SetAccessor, property» Java_ImportTypesOnly():bool
- «GetAccessor, SetAccessor, property» Java_ImportVisibility():bool
- «GetAccessor, SetAccessor, property» Java_ImportVisibilityRestriction():string
- «GetAccessor, SetAccessor, property» Java_SuppressAnnotationModifiers():bool
- «GetAccessor, property» Java_BinaryTypes():IBinaryTypeEntries
- «GetAccessor, SetAccessor, property» CSharp_ImportReferencedTypes():bool
- «GetAccessor, SetAccessor, property» CSharp_ImportReferencedTypesRestriction():string
- «GetAccessor, SetAccessor, property» CSharp_ImportTypesOnly():bool
- «GetAccessor, SetAccessor, property» CSharp_ImportVisibility():bool
- «GetAccessor, SetAccessor, property» CSharp_ImportVisibilityRestriction():string
- «GetAccessor, SetAccessor, property» CSharp_SuppressAttributeSections():bool
- «GetAccessor, SetAccessor, property» CSharp_OneAttributePerAttributeSection():bool
- «GetAccessor, SetAccessor, property» CSharp_SuppressAttributeSuffix():bool
- «GetAccessor, SetAccessor, property» CSharp_OverridePathForNativeLibraries():string
- «GetAccessor, SetAccessor, property» CSharp_ReflectionOnly():bool
- «GetAccessor, property» CSharp_BinaryTypes():IBinaryTypeEntries
- «GetAccessor, SetAccessor, property» VBasic_OverridePathForNativeLibraries():string
- «GetAccessor, SetAccessor, property» VBasic_ReflectionOnly():bool
- «GetAccessor, SetAccessor, property» VBasic_ImportReferencedTypes():bool
- «GetAccessor, SetAccessor, property» VBasic_ImportReferencedTypesRestriction():string
- «GetAccessor, SetAccessor, property» VBasic_ImportTypesOnly():bool
- «GetAccessor, SetAccessor, property» VBasic_ImportVisibility():bool
- «GetAccessor, SetAccessor, property» VBasic_ImportVisibilityRestriction():string
- «GetAccessor, SetAccessor, property» VBasic_SuppressAttributeSections():bool
- «GetAccessor, SetAccessor, property» VBasic_OneAttributePerAttributeSection():bool
- «GetAccessor, SetAccessor, property» VBasic_SuppressAttributeSuffix():bool
- «GetAccessor, property» VBasic_BinaryTypes():IBinaryTypeEntries

hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportBinaryTypesDlg** Operation **ImportBinaryTypes**

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**			

Operation **IImportBinaryTypesDlg::CSharp_ImportReferencedTypes**

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

 © 2014 Altova GmbH

r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_ImportVisibilityRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_OneAttributePerAttributeSection**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**string**			

Operation **IImportBinaryTypesDlg::CSharp_ReflectionOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_SuppressAttributeSections**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::CSharp_SuppressAttributeSuffix**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_BinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IBinaryTypeEntries**			

Operation **IImportBinaryTypesDlg::Java_ImportReferencedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IImportBinaryTypesDlg::Java_ImportReferencedTypesRestriction**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**string**			

Operation **IImportBinaryTypesDlg::Java_ImportTypesOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	**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**			

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**

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**			

© 2014 Altova GmbH

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**

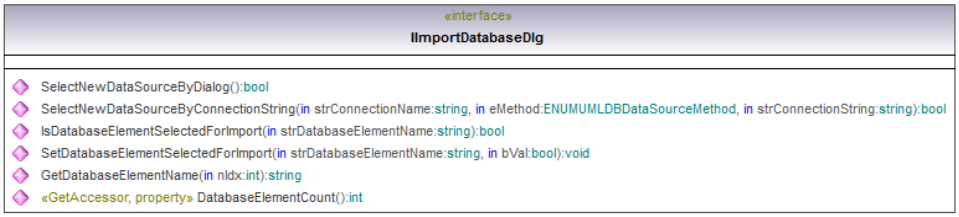
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

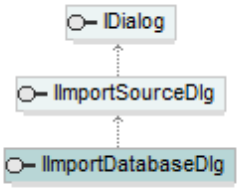
UModelAPI - IImportDatabaseDlg

Interface **IImportDatabaseDlg**

diagram



hierarchy



typedElements		
	Interface **IDialogs**	Operation **ImportDatabaseDlg**
	Interface **IDocument**	Operation **ImportDatabase**

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**			
	strConnectionString	**in**	**string**			
	return	**return**	**bool**			

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**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - IImportSourceDirectoryDlg

Interface **IImportSourceDirectoryDlg**

diagram	«interface» **IImportSourceDirectoryDlg** «GetAccessor, SetAccessor, property» Directory():string «GetAccessor, SetAccessor, property» ProcessSubdirectories():bool «GetAccessor, SetAccessor, property» ImportDirectoriesRelative():bool «GetAccessor, SetAccessor, property» Content_GenerateSingleDiagram():bool «GetAccessor, SetAccessor, property» Content_GenerateDiagramPerPackage():bool «GetAccessor, SetAccessor, property» Content_ShowNestedClassifiersSeparately():bool «GetAccessor, SetAccessor, property» Content_ShowAnonymousBoundElements():bool «GetAccessor, SetAccessor, property» Content_HyperlinkPackagesToDiagrams():bool
hierarchy	IDialog IImportSourceDlg IImportSourceDirectoryDlg
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportSourceDirectoryDlg** Operation **ImportSourceDirectory**

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**

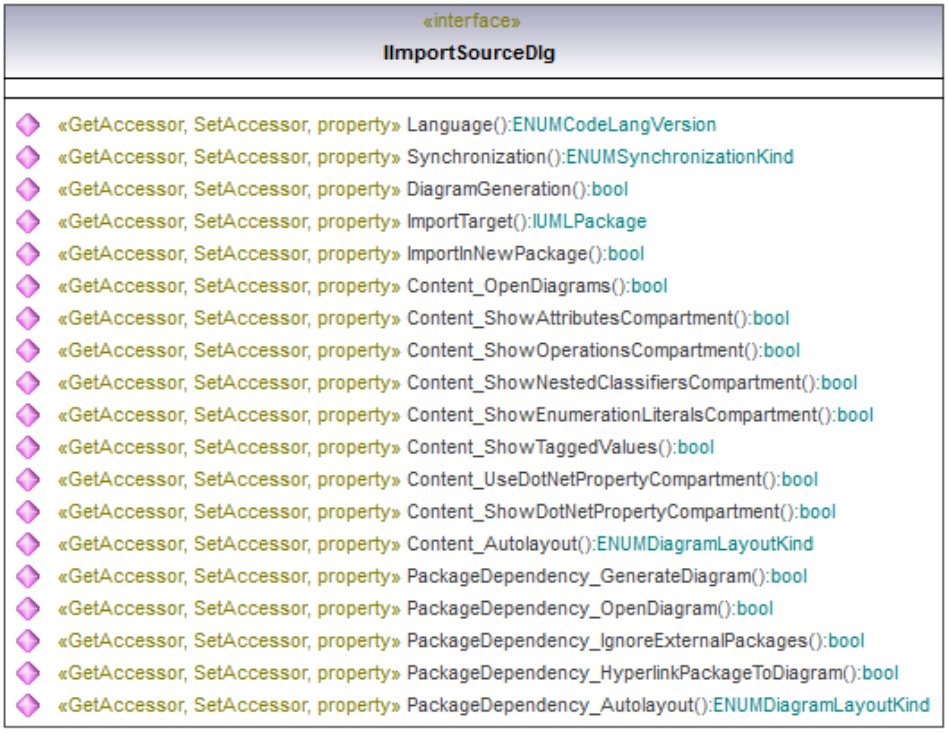
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

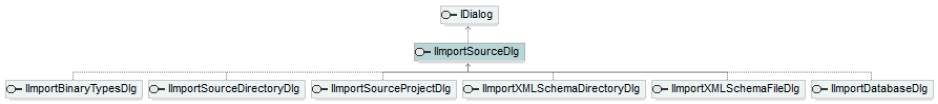
UModelAPI - IImportSourceDlg

Interface **IImportSourceDlg**

diagram



hierarchy



Operation **IImportSourceDlg::Content_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDiagramLayoutKind**			

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**			

Operation **IImportSourceDlg::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

Operation **IImportSourceDlg::PackageDependency_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMDiagramLayoutKind**			

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**

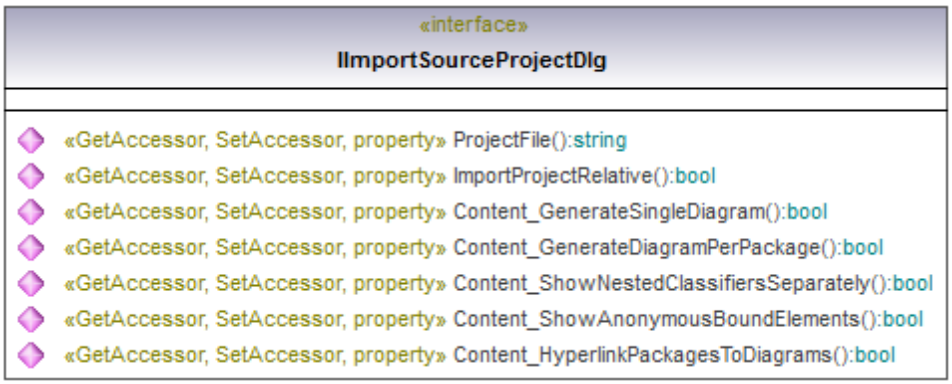
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

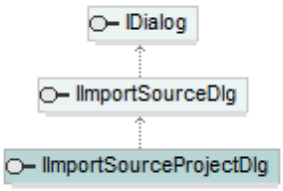
UModelAPI - IImportSourceProjectDlg

Interface **IImportSourceProjectDlg**

diagram



 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IDialogs** Operation **ImportSourceProjectDlg** Interface **IDocument** Operation **ImportSourceProject**

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**			

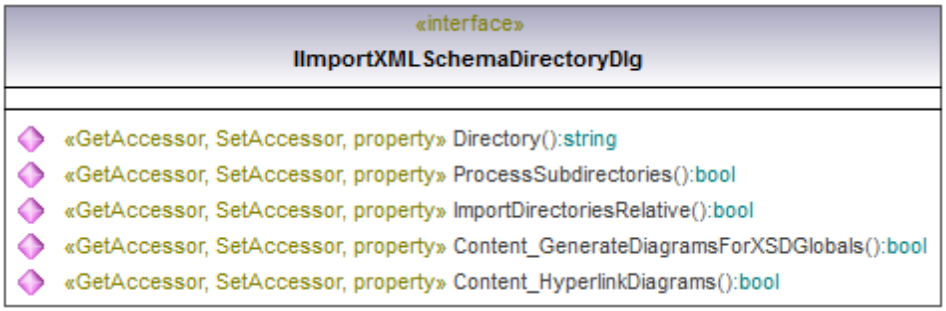
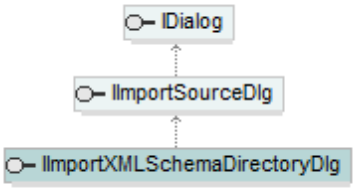
Operation **IImportSourceProjectDlg::ProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IImportXMLSchemaDirectoryDlg

Interface **IImportXMLSchemaDirectoryDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportXMLSchemaDirectoryDlg** Operation **ImportXMLSchemaDirectory**

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**

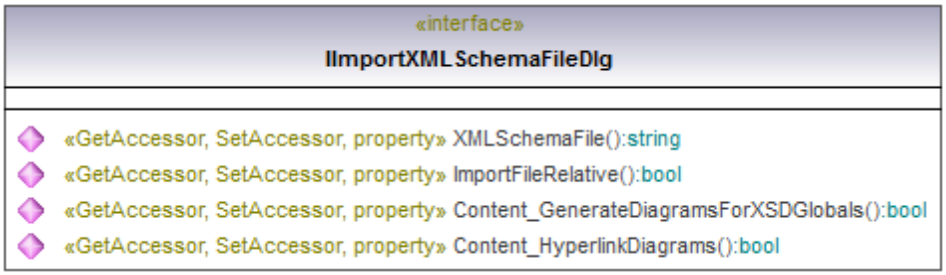
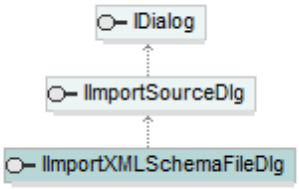
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IImportXMLSchemaFileDlg

Interface **IImportXMLSchemaFileDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ImportXMLSchemaFileDlg** Operation **ImportXMLSchemaFile**

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**			

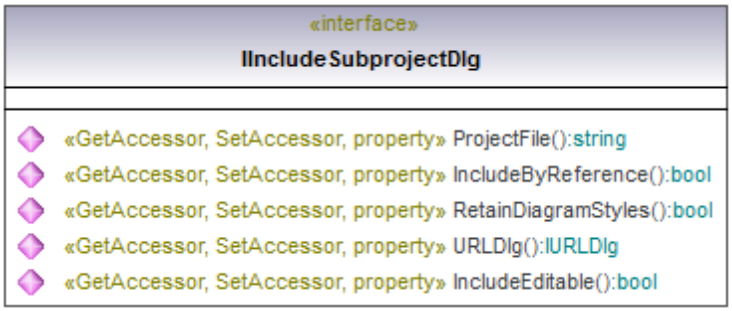
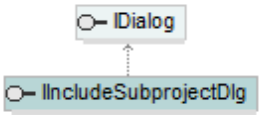
Operation **IImportXMLSchemaFileDlg::XMLSchemaFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IIncludeSubprojectDlg

Interface **IIncludeSubprojectDlg**

diagram	
hierarchy	
typedElements	Interface **IDialogs** Interface **IDocument** Operation **IncludeSubprojectDlg** Operation **IncludeSubproject**

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**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IKindSelection

Interface **IKindSelection**

diagram	«interface» IKindSelection IsKindOf(in strKindName:string):bool «GetAccessor, property» KindName():string «GetAccessor, SetAccessor, property» Selection():bool
typedElements	Interface **IKindSelectionList** Operation **Item**

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** 09/04/12 14:02:42

UModelAPI - IKindSelectionList

Interface **IKindSelectionList**

diagram	«interface» IKindSelectionList «GetAccessor, property» Count():int Item(in nIdx:int):IKindSelection «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch
typedElements	Interface **IGenerateDocumentationDlg** Operation **Include_Diagrams** **Include_Elements**

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**			

Operation **IKindSelectionList::Parent**

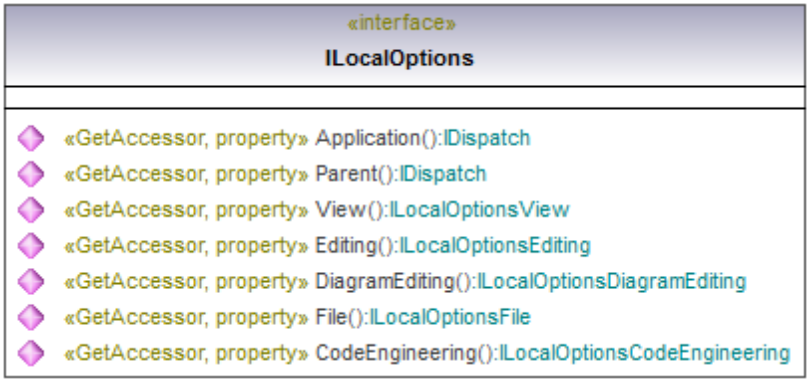
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ILocalOptions

Interface **ILocalOptions**

diagram



typedElements	Interface **IApplication**	Operation **Options**

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**			

Operation **ILocalOptions::DiagramEditing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsD**			

 © 2014 Altova GmbH

	iagramEditing

Operation **ILocalOptions::Editing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsEditing**			

Operation **ILocalOptions::File**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsFile**			

Operation **ILocalOptions::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptions::View**

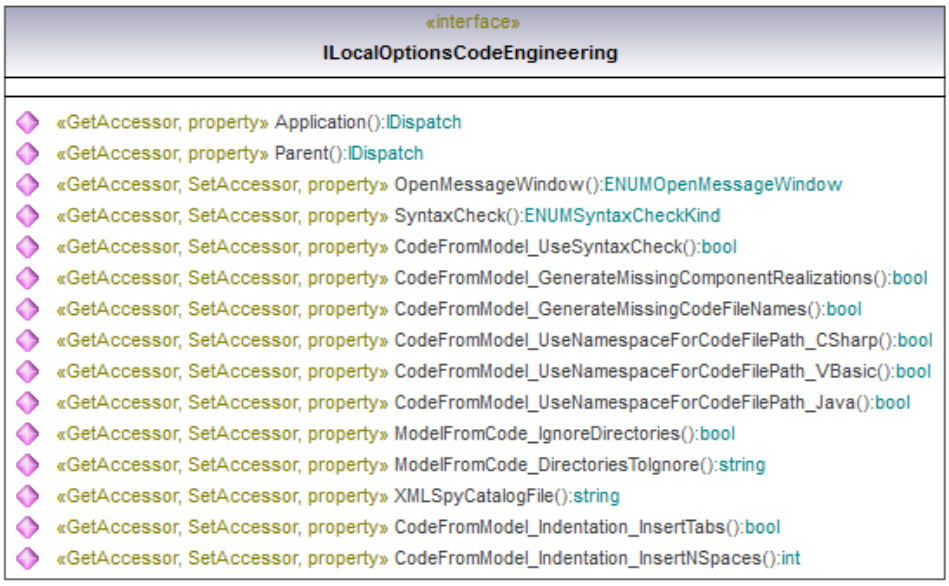
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ILocalOptionsView**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ILocalOptionsCodeEngineering

Interface **ILocalOptionsCodeEngineering**

diagram



typedElements	Interface **ILocalOptions**	Operation **CodeEngineering**

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**			

 © 2014 Altova GmbH

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**			

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**			

Operation **ILocalOptionsCodeEngineering::XMLSpyCatalogFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ILocalOptionsDiagramEditing

Interface **ILocalOptionsDiagramEditing**

diagram	«interface» **ILocalOptionsDiagramEditing** ResetExistingAssociations():void «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» AskBeforeAddingMoreThanItemsCount():int «GetAccessor, SetAccessor, property» AlwaysShowStyleDialogBeforeAddingItems():bool «GetAccessor, SetAccessor, property» ShowAttributesCompartment():bool «GetAccessor, SetAccessor, property» ShowOperationsCompartment():bool «GetAccessor, SetAccessor, property» ShowNestedClassifierCompartment():bool «GetAccessor, SetAccessor, property» ShowEnumerationLiteralsCompartment():bool «GetAccessor, SetAccessor, property» ShowExtensionPointsCompartment():bool «GetAccessor, SetAccessor, property» ShowTaggedValues():bool «GetAccessor, SetAccessor, property» UseDotNetPropertyCompartment():bool «GetAccessor, SetAccessor, property» ShowDotNetPropertyCompartment():bool «GetAccessor, SetAccessor, property» AutomaticallyCreateAssociations():bool «GetAccessor, SetAccessor, property» ResolveCollections():bool «GetAccessor, SetAccessor, property» ResolveCollectionsToUnknownExternalsUnqualified():bool «GetAccessor, property» CollectionTemplates_Java():ICollectionTemplates «GetAccessor, property» CollectionTemplates_CSharp():ICollectionTemplates «GetAccessor, property» CollectionTemplates_VBasic():ICollectionTemplates «GetAccessor, property» CollectionTemplates_UML():ICollectionTemplates «GetAccessor, SetAccessor, property» EnableAutomaticEntryHelper():bool
typedElements	Interface **ILocalOptions** Operation **DiagramEditing**

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**

© 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_Java**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_UML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_VBasic**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ICollectionTemplates**			

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**			

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**

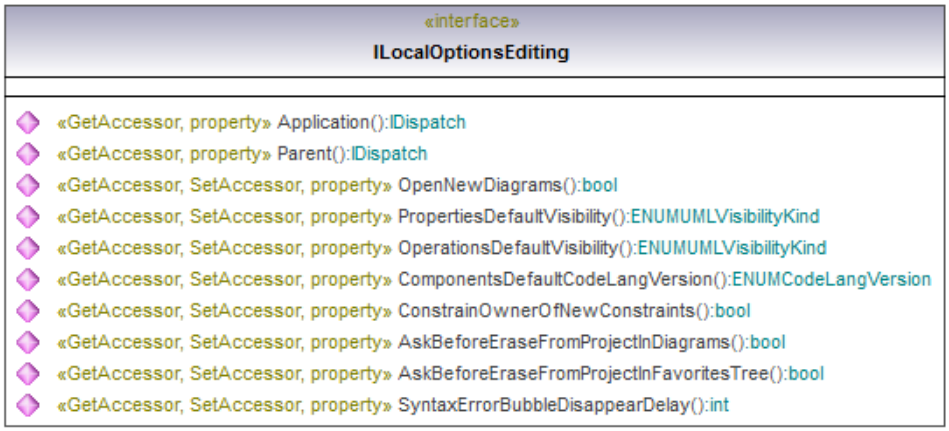
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ILocalOptionsEditing

Interface **ILocalOptionsEditing**

diagram



typedElements: Interface **ILocalOptions** Operation **Editing**

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**			

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**			

Operation **ILocalOptionsEditing::OperationsDefaultVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

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**			

Operation **ILocalOptionsEditing::SyntaxErrorBubbleDisappearDelay**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

© 2014 Altova GmbH

UModelAPI - ILocalOptionsFile

Interface **ILocalOptionsFile**

diagram	«interface» **ILocalOptionsFile** «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» WatchForFileChanges():bool «GetAccessor, SetAccessor, property» AskBeforeReloadChangedFile():bool «GetAccessor, SetAccessor, property» LoadAndSaveFavoritesWithProjectFile():bool «GetAccessor, SetAccessor, property» LoadAndSaveOpenDiagramsWithProjectFile():bool «GetAccessor, SetAccessor, property» OpenLastProjectOnProgramStart():bool «GetAccessor, SetAccessor, property» PrettyPrintFileContentWhenSaving():bool
typedElements	Interface **ILocalOptions** Operation **File**

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**			

© 2014 Altova GmbH

Operation **ILocalOptionsFile::WatchForFileChanges**

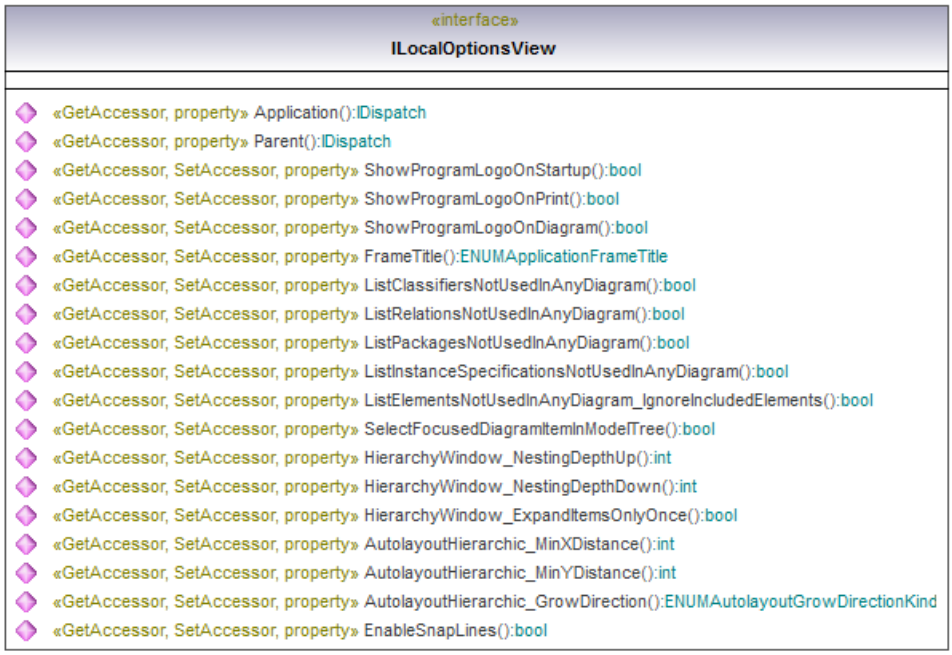
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ILocalOptionsView

Interface **ILocalOptionsView**

diagram



typedElements: Interface **ILocalOptions** Operation **View**

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**			

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**			

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**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **ILocalOptionsView::SelectFocusedDiagramItemInModelTree**

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**			

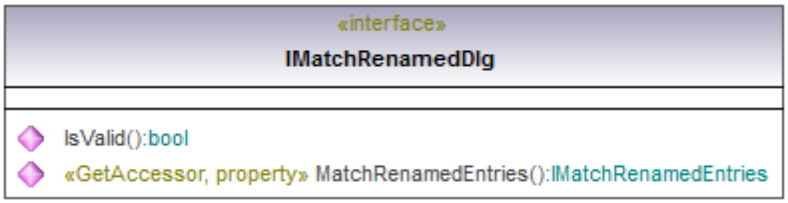
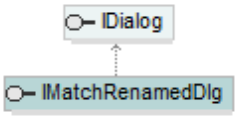
Operation **ILocalOptionsView::ShowProgramLogoOnStartup**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IMatchRenamedDlg

Interface **IMatchRenamedDlg**

diagram	
hierarchy	
typedElements	Interface **_ISynchronizationEvents** Operation **OnMatchRenamed**

Operation **IMatchRenamedDlg::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IMatchRenamedDlg::MatchRenamedEntries**

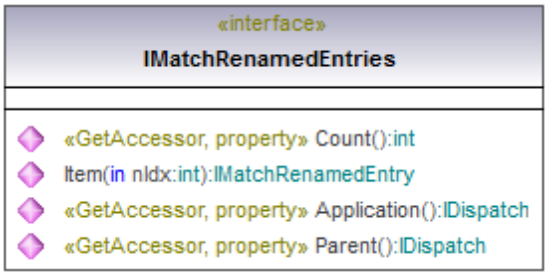
parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IMatchRenamedEntries**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IMatchRenamedEntries

Interface **IMatchRenamedEntries**

diagram	
typedElements	Interface **IMatchRenamedDlg** Operation **MatchRenamedEntries**

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**			

Operation **IMatchRenamedEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IMatchRenamedEntry

Interface **IMatchRenamedEntry**

diagram	«interface» **IMatchRenamedEntry** IsValid():bool «GetAccessor, property» KindName():string «GetAccessor, property» Name():string «GetAccessor, property» Namespace():string «GetAccessor, property» PossibleMatchingNames():IDispatch «GetAccessor, SetAccessor, property» MatchingName():string
typedElements	Interface **IMatchRenamedEntries** Operation **Item**

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** 09/04/12 14:02:42

UModelAPI - IModelTransformationDlg

Interface **IModelTransformationDlg**

diagram	«interface» IModelTransformationDlg «GetAccessor, SetAccessor, property» SourcePackage():IUMLPackage «GetAccessor, SetAccessor, property» TargetPackage():IUMLPackage «GetAccessor, SetAccessor, property» TransformInNewPackage():bool «GetAccessor, SetAccessor, property» TransformFromLanguage():ENUMCodeLang «GetAccessor, SetAccessor, property» TransformToLanguage():ENUMCodeLang «GetAccessor, SetAccessor, property» Synchronization():ENUMSynchronizationKind «GetAccessor, SetAccessor, property» TransformClassDiagrams():bool «GetAccessor, SetAccessor, property» OpenNewDiagrams():bool «GetAccessor, SetAccessor, property» GenerateComponentsAndComponentRealizations():bool «GetAccessor, SetAccessor, property» AutomaticallyUpdateTransformationAfterUpdateModelFromCode():bool «GetAccessor, SetAccessor, property» AutomaticallyUpdateTransformationBeforeUpdateCodeFromModel():boo «GetAccessor, property» TypeMappings():IModelTransformationTypeMappings
hierarchy	IDialog IModelTransformationDlg
typedElements	Interface **IDialogs** Interface **IDocument** Operation **ModelTransformationDlg** Operation **ModelTransformation**

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
	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

© 2014 Altova GmbH

r	**return**	**return**	**IUMLPackage**

Operation **IModelTransformationDlg::Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

Operation **IModelTransformationDlg::TargetPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

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**			

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**			

Operation **IModelTransformationDlg::TypeMappings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IModelTransformationTypeMappings**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IModelTransformationTypeMapping

Interface **IModelTransformationTypeMapping**

diagram	«interface» **IModelTransformationTypeMapping** «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» FromType():string «GetAccessor, SetAccessor, property» ToType():string
typedElements	Interface **IModelTransformationTypeMappings** Operation **InsertItemAt** **Item**

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** 09/04/12 14:02:42

UModelAPI - IModelTransformationTypeMappings

Interface **IModelTransformationTypeMappings**

diagram	«interface» **IModelTransformationTypeMappings** InsertItemAt(in nIdx:int, in strFromType:string, in strToType:string):IModelTransformationTypeMapping DeleteItemAt(in nIdx:int):void SetDefaults():void «GetAccessor, property» Count():int Item(in nIdx:int):IModelTransformationTypeMapping «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch

 © 2014 Altova GmbH

typedElements	Interface	Operation
	IModelTransformationDlg	TypeMappings

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**			

Operation **IModelTransformationTypeMappings::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IModelTransformationTypeMapping**			

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**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IProjectSettingsDlg

Interface **IProjectSettingsDlg**

diagram	«interface» **IProjectSettingsDlg** «GetAccessor, SetAccessor, property» Java_WriteDocumentationAsJavaDocs():bool «GetAccessor, SetAccessor, property» Java_JavaDocsAsDocumentation():bool «GetAccessor, SetAccessor, property» CSharp_WriteDocumentationAsDocComments():bool «GetAccessor, SetAccessor, property» CSharp_DocCommentsAsDocumention():bool «GetAccessor, SetAccessor, property» CSharp_DefinedSymbols():string «GetAccessor, SetAccessor, property» VBasic_WriteDocumentationAsDocComments():bool «GetAccessor, SetAccessor, property» VBasic_DocCommentsAsDocumention():bool «GetAccessor, SetAccessor, property» VBasic_DefinedSymbols():string «GetAccessor, SetAccessor, property» CSharp_ResolveAliases():bool «GetAccessor, SetAccessor, property» VBasic_ResolveAliases():bool
hierarchy	IDialog IProjectSettingsDlg
typed Elements	Interface **IDialogs** Interface **IDocument** Operation **ProjectSettingsDlg** Operation **ProjectSettings**

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**			

© 2014 Altova GmbH

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**			

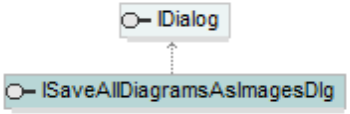
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ISaveAllDiagramsAsImagesDlg

Interface **ISaveAllDiagramsAsImagesDlg**

diagram

hierarchy



typedElements

Interface **IDialogs**
Interface **IDocument**
Operation **SaveAllDiagramsAsImagesDlg**
Operation **SaveAllDiagramsAsImages**

Operation **ISaveAllDiagramsAsImagesDlg::Folder**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **ISaveAllDiagramsAsImagesDlg::ImageFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMOutputImageFormat**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ISynchronizationSettingsDlg

Interface **ISynchronizationSettingsDlg**

diagram	«interface» ISynchronizationSettingsDlg «GetAccessor, SetAccessor, property» ModelFromCode_Synchronization():ENUMSynchronizationKind «GetAccessor, SetAccessor, property» CodeFromModel_Synchronization():ENUMSynchronizationKind «GetAccessor, SetAccessor, property» CodeFromModel_UserDefinedSPLTemplatesOverrideDefault():bool «GetAccessor, SetAccessor, property» CodeFromModel_DeletingCode():ENUMSynchronizationDeleteKind
hierarchy	IDialog ISynchronizationSettingsDlg
typedElements	Interface **IDialogs** Interface **IDocument** Operation **SynchronizationSettingsDlg** Operation **SynchronizationSettings** **SynchronizeCodeFromModel** **SynchronizeModelFromCode**

Operation **ISynchronizationSettingsDlg::CodeFromModel_DeletingCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationDeleteKind**			

Operation **ISynchronizationSettingsDlg::CodeFromModel_Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSynchronizationKind**			

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**	**ENUMSynchron**			

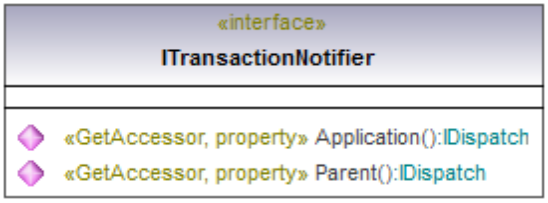
 © 2014 Altova GmbH

	izationKind

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ITransactionNotifier

Interface **ITransactionNotifier**

diagram	
typedElements	Interface **IDocument** Operation **TransactionNotifier**
documentation	Use this interface to register for _ITransactionEvents.

Operation **ITransactionNotifier::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

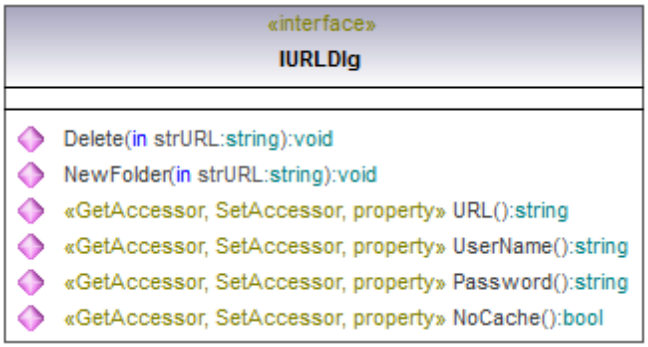
Operation **ITransactionNotifier::Parent**

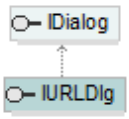
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IURLDlg

Interface **IURLDlg**

diagram	

hierarchy	
typedElements	Interface **IApplication** Interface **IDialogs** Interface **IDocument** Interface **IExportXMIFileDlg** Interface **IIncludeSubprojectDlg** Operation **ImportFromXMIFileFromURL** **OpenDocumentFromURL** Operation **URLOpenDialog** **URLSaveDialog** Operation **MergeProjectFromURL** **SaveToURL** Operation **URLDlg** Operation **URLDlg**

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**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

Events

This is a list of all events sent by the UModel API.

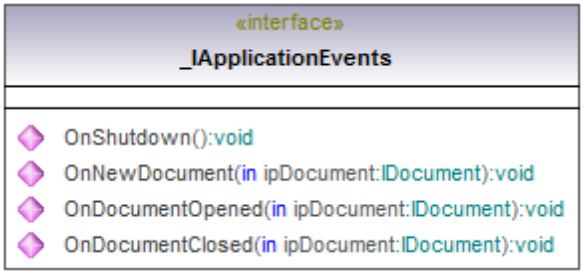
A list of events sent on UMLData level can be found here.

 © 2014 Altova GmbH

UModelAPI - _IApplicationEvents

Interface **_IApplicationEvents**

diagram



Operation **_IApplicationEvents::OnDocumentClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnDocumentOpened**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnNewDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IApplicationEvents::OnShutdown**

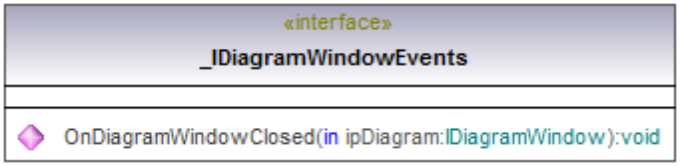
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - _IDiagramWindowEvents

Interface **_IDiagramWindowEvents**

diagram



Operation **_IDiagramWindowEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindo**			

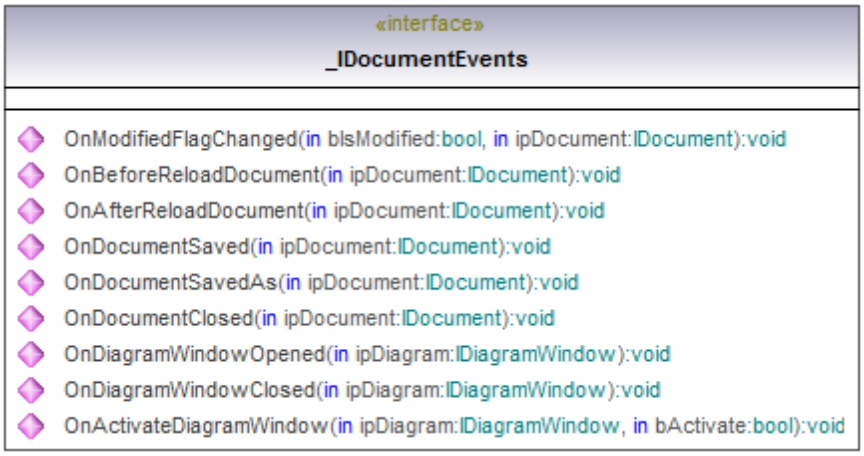
	return	return	w void

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - _IDocumentEvents

Interface **_IDocumentEvents**

diagram



Operation **_IDocumentEvents::OnActivateDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindow**			
	bActivate	**in**	**bool**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnAfterReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnBeforeReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	**in**	**IDiagramWindow**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDiagramWindowOpened**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**ipDiagram**	**in**	**IDiagramWindow**
	return	**return**	**void**

Operation **_IDocumentEvents::OnDocumentClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentSaved**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnDocumentSavedAs**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_IDocumentEvents::OnModifiedFlagChanged**

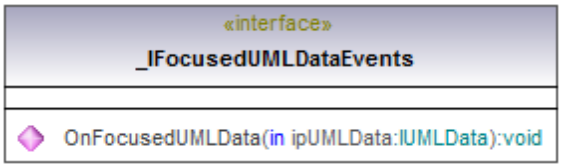
parameter	name	direction	type	type modifier	multiplicity	default
	bIsModified	**in**	**bool**			
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - _IFocusedUMLDataEvents

Interface **_IFocusedUMLDataEvents**

diagram



Operation **_IFocusedUMLDataEvents::OnFocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - _ISynchronizationEvents

Interface **_ISynchronizationEvents**

diagram	«interface» **_ISynchronizationEvents** OnMatchRenamed(in ipMatchRenamedDlg:IMatchRenamedDlg):void

Operation **_ISynchronizationEvents::OnMatchRenamed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipMatchRenamedDlg	**in**	**IMatchRenamedDlg**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - _ITransactionEvents

Interface **_ITransactionEvents**

diagram	«interface» **_ITransactionEvents** OnBeginDataModification(in ipDocument:IDocument):void OnEndDataModification(in ipDocument:IDocument):void
documentation	Use ITransactionNotifier to register for these events

Operation **_ITransactionEvents::OnBeginDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

Operation **_ITransactionEvents::OnEndDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	**in**	**IDocument**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

Enumerations

This is a list of all enumerations used by the UModel API. If your scripting environment does not support enumerations use the number-values instead.

A list of enumerations defined on UMLData level can be found here.

 © 2014 Altova GmbH

UModelAPI - ENUMApplicationFrameTitle

Enumeration **ENUMApplicationFrameTitle**

diagram	«enumeration» **ENUMApplicationFrameTitle** eApplicationFrameTitle_FileNameOnly = 0 eApplicationFrameTitle_FullPathName = 1
typedElements	Interface **ILocalOptionsView** Operation **FrameTitle**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMApplicationStatus

Enumeration **ENUMApplicationStatus**

diagram	«enumeration» **ENUMApplicationStatus** eApplicationRunning = 0 eApplicationAfterLicenseCheck = 1 eApplicationBeforeLicenseCheck = 2 eApplicationConcurrentLicenseCheckFailed = 3 eApplicationProcessingCommandLine = 4
typedElements	Interface **IApplication** Operation **Status**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMAutolayoutGrowDirectionKind

Enumeration **ENUMAutolayoutGrowDirectionKind**

diagram	«enumeration» **ENUMAutolayoutGrowDirectionKind** eAutolayoutGrowDirection_TopDown = 0 eAutolayoutGrowDirection_BottomUp = 1 eAutolayoutGrowDirection_LeftToRight = 2 eAutolayoutGrowDirection_RightToLeft = 3
typedElements	Interface **ILocalOptionsView** Operation **AutolayoutHierarchic_GrowDirection**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMCodeLang

Enumeration **ENUMCodeLang**

diagram	«enumeration» **ENUMCodeLang** eCodeLang_UML = -1 eCodeLang_Java = 0 eCodeLang_CSharp = 1 eCodeLang_XSD = 2 eCodeLang_VBasic = 3 eCodeLang_DB = 4
typedElements	Interface **IModelTransformationDlg** Interface **IUMLComponent** Interface **IUMLDataAll** Interface **IUMLPackage** Operation **TransformFromLanguage** **TransformToLanguage** Operation **CodeLang** Operation **CodeLang** **IsCodeLangNamespace** **IsCodeLangNamespaceRoot** Operation **IsCodeLangNamespace** **IsCodeLangNamespaceRoot**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - ENUMCodeLangVersion

Enumeration **ENUMCodeLangVersion**

diagram	«enumeration» **ENUMCodeLangVersion** eCodeLang_Java_1_4 = 0 eCodeLang_Java_5_0 = 1 eCodeLang_Java_6_0 = 2 eCodeLang_CSharp_1_2 = 3 eCodeLang_CSharp_2_0 = 4 eCodeLang_CSharp_3_0 = 5 eCodeLang_XSD_1_0 = 6 eCodeLang_VBasic_7_1 = 7 eCodeLang_VBasic_8_0 = 8 eCodeLang_VBasic_9_0 = 9 eCodeLang_CSharp_4_0 = 10 eCodeLang_DB_MSAccess = 11 eCodeLang_DB_MSSQLServer = 12 eCodeLang_DB_Oracle = 13 eCodeLang_DB_MySQL = 14 eCodeLang_DB_IBMDB2 = 15 eCodeLang_DB_Sybase = 16 eCodeLang_DB_PostgreSQL = 17
typedElements	Interface **IImportSourceDlg** Interface **ILocalOptionsEditing** Interface **IUMLComponent** Interface **IUMLDataAll** Operation **Language** Operation **ComponentsDefaultCodeLangVersion** Operation **CodeLangVersion** Operation **CodeLangVersion**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMDiagramLayoutKind

Enumeration **ENUMDiagramLayoutKind**

diagram	«enumeration» **ENUMDiagramLayoutKind** eDiagramLayout_None = -1 eDiagramLayout_ForceDirected = 0 eDiagramLayout_Hierarchic = 1 eDiagramLayout_Block = 2 eDiagramLayout_XSDGlobals = 3
typedElements	Interface **IDiagramWindow** Interface **IImportSourceDlg** Operation **Autolayout** **AutolayoutSelection** Operation **Content_Autolayout**

	PackageDependency_Autolayout

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMDocumentationFilePathKind

Enumeration **ENUMDocumentationFilePathKind**

diagram	«enumeration» **ENUMDocumentationFilePathKind** eDocumentationFilePath_Absolute = 0 eDocumentationFilePath_RelativeToResultFile = 1
typedElements	Interface IGenerateDocumentationDlg Operation GenerateLinksToLocalFiles

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMDocumentationFontSetting

Enumeration **ENUMDocumentationFontSetting**

diagram	«enumeration» **ENUMDocumentationFontSetting** eDocumentationFontSetting_Header = 0 eDocumentationFontSetting_Header2 = 1 eDocumentationFontSetting_ElementHeader = 2 eDocumentationFontSetting_ElementHeader2 = 3 eDocumentationFontSetting_TableLineTitle = 4 eDocumentationFontSetting_TableLineData = 5 eDocumentationFontSetting_SubTableLineTitle = 6 eDocumentationFontSetting_SubTableLineData = 7 eDocumentationFontSetting_Footer = 8 eDocumentationFontSetting_Footer2 = 9
typedElements	Interface IGenerateDocumentationDlg Operation Fonts_GetFace Fonts_GetSize Fonts_GetTextColor Fonts_IsBold Fonts_IsItalic Fonts_IsUnderline Fonts_SetBold Fonts_SetFace Fonts_SetItalic Fonts_SetSize Fonts_SetTextColor Fonts_SetUnderline

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - ENUMDocumentationImageFormat

Enumeration **ENUMDocumentationImageFormat**

diagram	«enumeration» **ENUMDocumentationImageFormat** eDocumentationImageFormat_PNG = 0 eDocumentationImageFormat_EMF = 1

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMDocumentationOutputFormat

Enumeration **ENUMDocumentationOutputFormat**

diagram	«enumeration» **ENUMDocumentationOutputFormat** eDocumentationOutputFormat_HTML = 0 eDocumentationOutputFormat_Word = 1 eDocumentationOutputFormat_RTF = 2 eDocumentationOutputFormat_PDF = 3	
typedElements	Interface **IGenerateDocumentationDlg**	Operation **OutputFormat**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMExportXMIType

Enumeration **ENUMExportXMIType**

diagram	«enumeration» **ENUMExportXMIType** eXMI21ForUML20 = 0 eXMI21ForUML212 = 1 eXMI21ForUML22 = 2 eXMI21ForUML23 = 3 eXMI24ForUML24 = 4	
typedElements	Interface **IExportXMIFileDlg**	Operation **XMIType**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMOpenMessageWindow

Enumeration **ENUMOpenMessageWindow**

diagram	«enumeration» **ENUMOpenMessageWindow** eOpenMessageWindow_Always = 0 eOpenMessageWindow_ForErrorsAndWarnings = 1 eOpenMessageWindow_ForErrors = 2
typedElements	Interface **ILocalOptionsCodeEngineering** Operation **OpenMessageWindow**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMOutputImageFormat

Enumeration **ENUMOutputImageFormat**

diagram	«enumeration» **ENUMOutputImageFormat** eOutputImageFormat_PNG = 0 eOutputImageFormat_EMF = 1
typedElements	Interface **IDiagramWindow** Operation **SaveDiagramAsImage** Interface **IGenerateDocumentationDlg** Operation **DiagramImageFormat** Interface **ISaveAllDiagramsAsImagesDlg** Operation **ImageFormat**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMSynchronizationDeleteKind

Enumeration **ENUMSynchronizationDeleteKind**

diagram	«enumeration» **ENUMSynchronizationDeleteKind** eSynchronizationDelete_CommentOut = 0 eSynchronizationDelete_Delete = 1
typedElements	Interface **ISynchronizationSettingsDlg** Operation **CodeFromModel_DeletingCode**

© 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMSynchronizationKind

Enumeration **ENUMSynchronizationKind**

diagram	«enumeration» **ENUMSynchronizationKind** eSynchronization_Merge = 0 eSynchronization_Overwrite = 1
typedElements	Interface **IImportSourceDlg** Interface **IModelTransformationDlg** Interface **ISynchronizationSettingsDlg** Operation **Synchronization** Operation **Synchronization** Operation **CodeFromModel_Synchronization** **ModelFromCode_Synchronization**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMSyntaxCheckKind

Enumeration **ENUMSyntaxCheckKind**

diagram	«enumeration» **ENUMSyntaxCheckKind** eSyntaxCheck_AllCodingElements = 0 eSyntaxCheck_ElementsUsedForCodeEngineering = 1
typedElements	Interface **ILocalOptionsCodeEngineering** Operation **SyntaxCheck**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UMLData

The UMLData interfaces allow direct UML-level access to a document. You can read and directly modify the UML representation of the document.

IUMLData is the common base interface of IUMLElement and IUMLGuiElement.

IUMLElements contains elements as defined by OMG in the UML 2.4 Superstructure Specification (also see http://www.uml.org)

IUMLGuiElements contains elements from Altova for diagrams and to show IUMLElements on diagrams

Also see Object model UMLData how to read and modify UML elements and GUI elements.

Errors

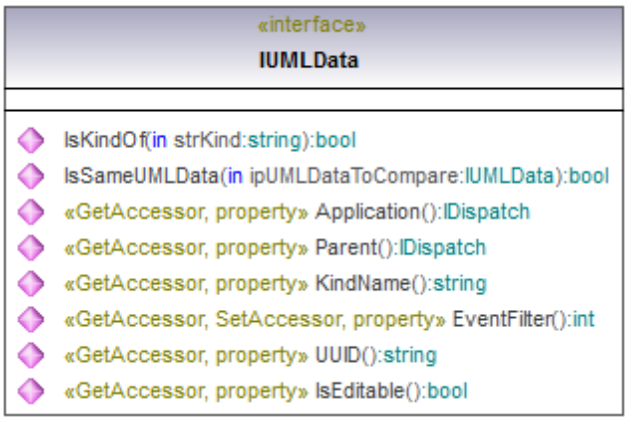
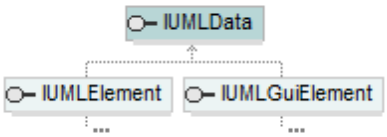
1000 The application object is no longer valid.
1001 Invalid parameter or invalid address for the return parameter was specified.
1002 UModel API is not available in the current edition.

1400 Invalid UMLData modification.
1401 Invalid Waypoint modification.
1402 No changes allowed.
1403 No changes allowed during Undo/Redo.
1404 Element is hidden by Element Style (visibility).
1405 Predefined element not found.
1406 Predefined element is of invalid kind.

Errors of the UModel API can be found here.

UModelAPI - IUMLData

Interface **IUMLData**

diagram	
hierarchy	
typedElements	Interface **_IFocusedUMLDataEvents** Interface **_IUMLDataEvents** Interface **IDiagramWindow** Interface **IDocument** Interface **IUMLCommentTextHyperlink** Operation **OnFocusedUMLData** Operation **OnAfterAddChild** **OnBeforeErase** **OnChanged** **OnMoveData** Operation **FocusedData** Operation **CanFocusUMLDataInModelTree** **FocusedUMLData** **FocusUMLDataInModelTree** Operation **SetHyperlinkModelElementAddress**

 © 2014 Altova GmbH

	Interface **IUMLData**	Operation **IsSameUMLData**
	Interface **IUMLDataAll**	Operation **AddUMLGuiNodeLink** **FindPredefinedOwnedElement** **InsertOwnedDiagramAt** **InsertOwnedHyperlink2ModelAt** **IsSameUMLData** **LinkedModelElement** **SetHyperlinkModelElementAddress**
	Interface **IUMLDataList**	Operation **ContainsUMLData** **Item**
	Interface **IUMLElement**	Operation **FindPredefinedOwnedElement**
	Interface **IUMLGuiDiagram**	Operation **AddUMLGuiNodeLink**
	Interface **IUMLGuiRootElement**	Operation **InsertOwnedDiagramAt**
	Interface **IUMLGuiTextHyperlink**	Operation **SetHyperlinkModelElementAddress**
	Interface **IUMLHyperlink2Model**	Operation **LinkedModelElement**
	Interface **IUMLNamedElement**	Operation **InsertOwnedHyperlink2ModelAt**

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**			
	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

r	**return**	**return**	**IDispatch**

Operation **IUMLData::UUID**

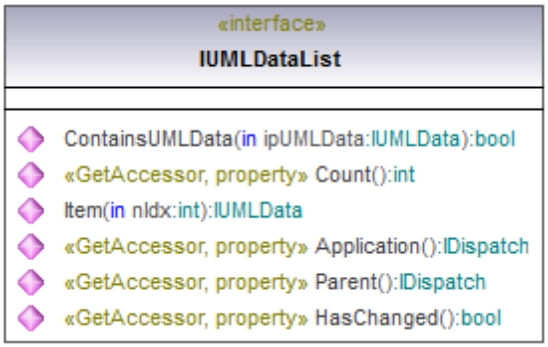
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDataList

Interface **IUMLDataList**

diagram



typedElements

Interface **IDiagramWindow**	Operation **SelectedGuiElements**
Interface **IUMLAcceptEventAction**	Operation **ActionTriggers** **EventActionResults**
Interface **IUMLAction**	Operation **InputPins** **LocalPostConditions** **LocalPreConditions** **OutputPins**
Interface **IUMLActivity**	Operation **ActivityEdges** **ActivityGroups** **ActivityNodes**
Interface **IUMLActivityEdge**	Operation **ActivityPartitions** **InterruptibleActivityRegions** **StructuredActivityNodes**
Interface **IUMLActivityGroup**	Operation **ContainedEdges** **ContainedNodes** **SubGroups**
Interface **IUMLActivityNode**	Operation **IncomingEdges** **OutgoingEdges**
Interface **IUMLActivityPartition**	Operation **Edges** **Nodes** **SubPartitions**
Interface **IUMLArtifact**	Operation **Manifestations** **NestedArtifacts** **OwnedAttributes** **OwnedOperations**
Interface **IUMLAssociation**	Operation **EndTypes** **MemberEnds** **NavigableOwnedEnds** **OwnedEnds**
Interface **IUMLBehavior**	Operation **OwnedParameters** **Postconditions** **Preconditions**
Interface **IUMLBehavioralFeature**	Operation **Methods** **OwnedParameters** **RaisedExceptions**
Interface **IUMLBehavioredClassifier**	Operation **InterfaceRealizations** **OwnedBehaviors**
Interface **IUMLCallAction**	Operation **Results**
Interface **IUMLClass**	Operation **NestedClassifiers**

 © 2014 Altova GmbH

	Interface IUMLClassifier	Operation	OwnedOperations SuperClasses Attributes CollaborationUses Features Generalizations Generals InheritedMembers OwnedUseCases Specifics UseCases
	Interface IUMLClassifierTemplateParameter	Operation	ConstrainingClassifiers
	Interface IUMLCollaboration	Operation	CollaborationRoles
	Interface IUMLCollaborationUse	Operation	RoleBindings
	Interface IUMLCombinedFragment	Operation	Operands
	Interface IUMLComment	Operation	AnnotatedElements OwnedHyperlinks
	Interface IUMLComponent	Operation	Realizations
	Interface IUMLConnectionPointReference	Operation	Entries Exits
	Interface IUMLConstraint	Operation	ConstrainedElements
	Interface IUMLDataAll	Operation	ActionTriggers ActivityEdges ActivityGroups ActivityNodes ActivityPartitions ActualGates AllAppliableStereotypes AllWaypoints AnnotatedElements AppliedStereotypes Arguments AttachedNodes Attributes ClientDependencies Clients CollaborationRoles CollaborationUses ConnectionPoints Connections ConstrainedElements ConstrainingClassifiers ContainedEdges ContainedNodes Conveyed DeployedElements Deployments Edges ElementImports EndTypes Entries EventActionResults ExceptionHandlers ExceptionTypes Exits Extends ExtensionLocations ExtensionPoints Features FeaturingClassifiers FormalGates Fragments Generalizations Generals GetOwnedElementsOfKind GuiLinks Handlers ImportedMembers Includes IncomingEdges Incomings InformationFlowRealizations InformationSources InformationTargets InheritedMembers InputElements InputPins

		InputValues InStates InterfaceRealizations InterruptibleActivityRegions InterruptingEdges Layers Lifelines LineConnectionWaypoints LineLinks LocalPostConditions LocalPreConditions LowerValues Manifestations MemberEnds Members Messages Methods NavigableOwnedEnds NestedArtifacts NestedClassifiers NestedNodes NestedPackages Nodes Observations Operands OutgoingEdges Outgoings OutputElements OutputPins OutputValues OwnedArguments OwnedAttributes OwnedBehaviors OwnedComments OwnedConnectors OwnedDiagrams OwnedElements OwnedEnds OwnedGuiElements OwnedGuiNodeLinks OwnedHyperlinks OwnedLiterals OwnedMembers OwnedOperations OwnedParameters OwnedPorts OwnedRules OwnedStereotypes OwnedTemplateBindings OwnedTemplateParameters OwnedTypes OwnedUseCases PackagedElements PackageImports PackageMerges ParameterSubstitutions Postconditions Preconditions ProfileApplications Qualifiers RaisedExceptions Realizations RealizingConnectors Referred Regions RelatedElements RelativeNodes Results RoleBindings Slots Sources Specifics StereotypeApplications StructuredActivityNodes SubGroups Subjects SubmachineStates SubPartitions SubVertices

© 2014 Altova GmbH

		SuperClasses
		SupplierDependencies
		Suppliers Targets TextLabels
		Transitions Triggers
		UpperValues UseCases
		Values Waypoints
	Interface IUMLDataType	Operation OwnedAttributes
		OwnedOperations
	Interface IUMLDependency	Operation Clients Suppliers
	Interface IUMLDeploymentTarget	Operation DeployedElements
		Deployments
	Interface IUMLDirectedRelationship	Operation Sources Targets
	Interface IUMLDuration	Operation Observations
	Interface IUMLElement	Operation AllAppliableStereotypes
		AppliedStereotypes
		GetOwnedElementsOfKind
		OwnedComments
		OwnedElements
		StereotypeApplications
	Interface IUMLEncapsulatedClassifier	Operation OwnedPorts
	Interface IUMLEnumeration	Operation OwnedLiterals
	Interface IUMLExceptionHandler	Operation ExceptionTypes
	Interface IUMLExecutableNode	Operation Handlers
	Interface IUMLExpansionRegion	Operation InputElements
		OutputElements
	Interface IUMLExpression	Operation Operands
	Interface IUMLExtend	Operation ExtensionLocations
	Interface IUMLFeature	Operation FeaturingClassifiers
	Interface IUMLGuiDiagram	Operation GuiLinks Layers
		OwnedHyperlinks
	Interface IUMLGuiElement	Operation OwnedGuiElements
	Interface IUMLGuiLineLink	Operation AllWaypoints
		LineConnectionWaypoints
		Waypoints
	Interface IUMLGuiLink	Operation AttachedNodes
		RelativeNodes
	Interface IUMLGuiNodeLink	Operation OwnedGuiNodeLinks
	Interface IUMLGuiNote	Operation OwnedHyperlinks
	Interface IUMLGuiRootElement	Operation OwnedDiagrams
	Interface IUMLGuiTextLabelWaypoint	Operation TextLabels
	Interface IUMLGuiWaypoint	Operation LineLinks
	Interface IUMLInformationFlow	Operation Conveyed
		InformationFlowRealizations
		InformationSources
		InformationTargets
		RealizingConnectors
	Interface IUMLInstanceSpecification	Operation Slots
	Interface IUMLInteraction	Operation FormalGates Fragments
		Lifelines Messages
	Interface IUMLInteractionUse	Operation ActualGates
	Interface IUMLInterface	Operation NestedClassifiers
		OwnedAttributes
		OwnedOperations
	Interface IUMLInterruptibleActivityRegion	Operation InterruptingEdges Nodes

	Interface IUMLInvocationAction	Operation Arguments
	Interface IUMLMessage	Operation OwnedArguments
	Interface IUMLMultiplicityElement	Operation LowerValues UpperValues
	Interface IUMLNamedElement	Operation ClientDependencies OwnedHyperlinks SupplierDependencies
	Interface IUMLNamespace	Operation ElementImports ImportedMembers Members OwnedMembers OwnedRules PackageImports PackageMerges
	Interface IUMLNode	Operation NestedNodes
	Interface IUMLObjectNode	Operation ExceptionHandlers InStates
	Interface IUMLOpaqueAction	Operation InputValues OutputValues
	Interface IUMLPackage	Operation NestedPackages OwnedStereotypes OwnedTypes PackagedElements ProfileApplications
	Interface IUMLProperty	Operation Qualifiers
	Interface IUMLProtocolTransition	Operation Referred
	Interface IUMLRegion	Operation SubVertices Transitions
	Interface IUMLRelationship	Operation RelatedElements
	Interface IUMLSignal	Operation OwnedAttributes
	Interface IUMLSlot	Operation Values
	Interface IUMLState	Operation ConnectionPoints Connections Regions
	Interface IUMLStateMachine	Operation ConnectionPoints Regions SubmachineStates
	Interface IUMLStructuredActivityNode	Operation Edges Nodes
	Interface IUMLStructuredClassifier	Operation OwnedAttributes OwnedConnectors
	Interface IUMLTemplateableElement	Operation OwnedTemplateBindings
	Interface IUMLTemplateBinding	Operation ParameterSubstitutions
	Interface IUMLTemplateSignature	Operation OwnedTemplateParameters
	Interface IUMLTimeExpression	Operation Observations
	Interface IUMLTransition	Operation Triggers
	Interface IUMLType	Operation TypedElements
	Interface IUMLUseCase	Operation Extends ExtensionPoints Includes Subjects
	Interface IUMLVertex	Operation Incomings Outgoings

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**			
	return	**return**	**bool**			

© 2014 Altova GmbH

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**			

Operation **IUMLDataList::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

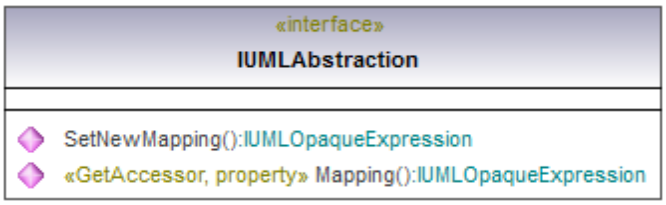
IUMLElement

This is a list of elements as defined by OMG in the UML 2.4 Superstructure Specification (also see http://www.uml.org).

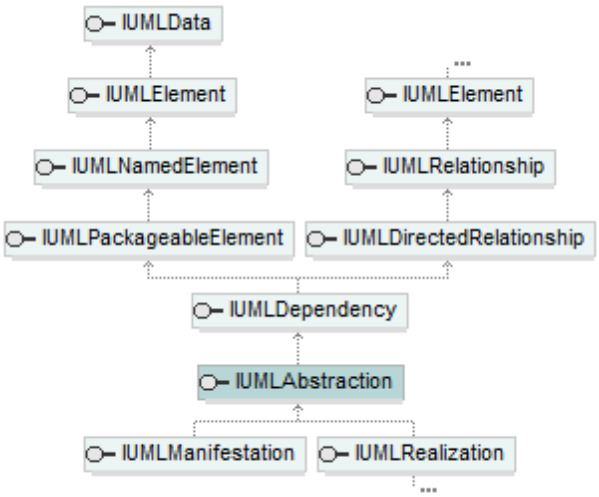
UModelAPI - IUMLAbstraction

Interface **IUMLAbstraction**

diagram



hierarchy



Operation **IUMLAbstraction::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression**			

Operation **IUMLAbstraction::SetNewMapping**

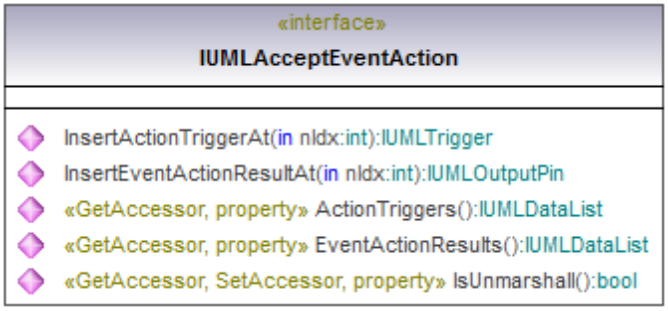
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

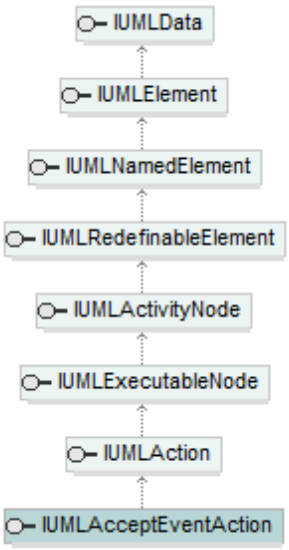
UModelAPI - IUMLAcceptEventAction

Interface **IUMLAcceptEventAction**

diagram



 © 2014 Altova GmbH

hierarchy	

Operation **IUMLAcceptEventAction::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTrigger.					

Operation **IUMLAcceptEventAction::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOutputPin.					

Operation **IUMLAcceptEventAction::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLAcceptEventAction::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLAcceptEventAction::IsUnmarshall**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLAction

Interface **IUMLAction**

diagram	«interface» **IUMLAction** InsertLocalPostConditionAt(in nIdx:int):IUMLConstraint InsertLocalPreConditionAt(in nIdx:int):IUMLConstraint «GetAccessor, property» LocalPostConditions():IUMLDataList «GetAccessor, property» LocalPreConditions():IUMLDataList «GetAccessor, property» ActionInputPin():IUMLActionInputPin «GetAccessor, property» InputPins():IUMLDataList «GetAccessor, property» OutputPins():IUMLDataList «GetAccessor, property» ActionContext():IUMLClassifier «GetAccessor, SetAccessor, property» IsLocallyReentrant():bool
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLRedefinableElement IUMLActivityNode IUMLExecutableNode IUMLAction IUMLAcceptEventAction, IUMLInvocationAction ..., IUMLStructuredActivityNode ..., IUMLOpaqueAction, IUMLValueSpecificationAction

Operation **IUMLAction::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLAction::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActionInputPin**			

Operation **IUMLAction::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInputPin.					

Operation **IUMLAction::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLAction::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

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**			
documentation	A list of elements of type IUMLConstraint.					

Operation **IUMLAction::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

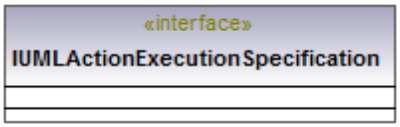
Operation **IUMLAction::OutputPins**

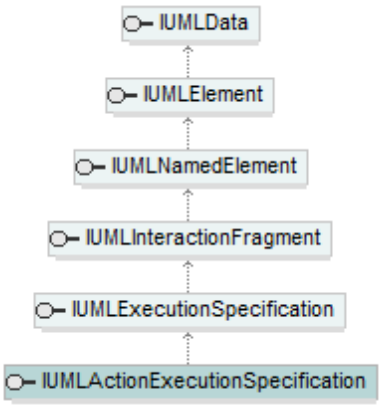
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOutputPin.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActionExecutionSpecification

Interface **IUMLActionExecutionSpecification**

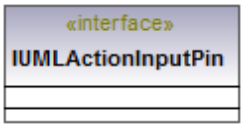
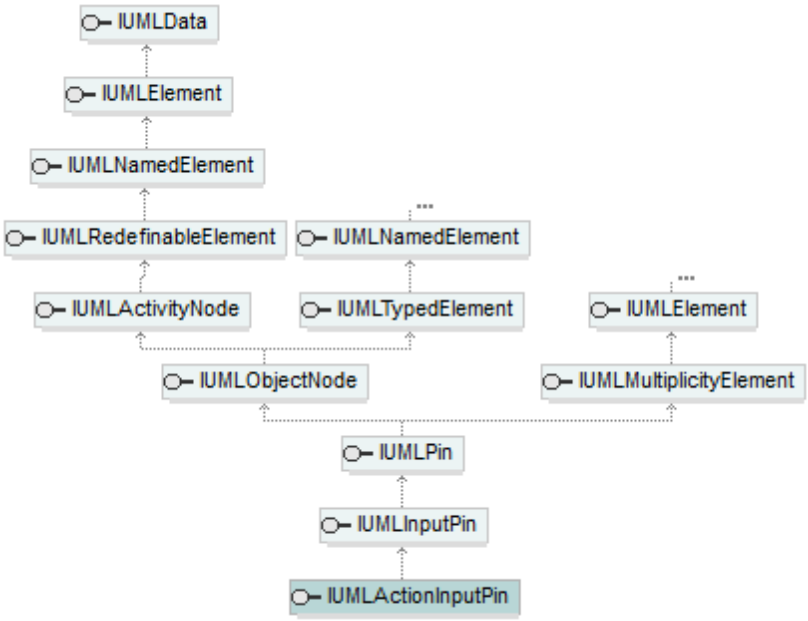
diagram	

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActionInputPin

Interface **IUMLActionInputPin**

diagram	
hierarchy	
typedElements	Interface **IUMLAction** Operation **ActionInputPin** Interface **IUMLDataAll** Operation **ActionInputPin**

 © 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActivity

Interface **IUMLActivity**

diagram	«interface» **IUMLActivity** InsertActivityGroupAt(in nIdx:int, in strKind:string):IUMLActivityGroup InsertActivityNodeAt(in nIdx:int, in strKind:string):IUMLActivityNode InsertActivityEdgeAt(in nIdx:int, in strKind:string, in ipFrom:IUMLActivityNode, in ipTo:IUMLActivityNode):IUMLActivityEdge «GetAccessor, property» ActivityGroups():IUMLDataList «GetAccessor, property» ActivityNodes():IUMLDataList «GetAccessor, property» ActivityEdges():IUMLDataList «GetAccessor, SetAccessor, property» IsActivityReadOnly():bool «GetAccessor, SetAccessor, property» SingleExecution():bool
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLPackageableElement IUMLNamedElement IUMLNamedElement IUMLElement IUMLElement IUMLType IUMLNamespace IUMLRedefinableElement IUMLParameterableElement IUMLTemplateableElement IUMLClassifier IUMLStructuredClassifier IUMLClassifier IUMLEncapsulatedClassifier IUMLBehavioredClassifier IUMLClass IUMLBehavior IUMLActivity
typedElements	Interface **IUMLActivityEdge** Interface **IUMLActivityGroup** Interface **IUMLDataAll** Operation **Activity** Operation **InActivity** Operation **Activity** **InActivity**

Operation **IUMLActivity::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLActivity::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityGroup.					

Operation **IUMLActivity::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLDataList**
documentation	A list of elements of type IUMLActivityNode.		

Operation **IUMLActivity::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLActivityNode**			
	ipTo	**in**	**IUMLActivityNode**			
	return	**return**	**IUMLActivityEdge**			

Operation **IUMLActivity::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityGroup**			

Operation **IUMLActivity::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityNode**			

Operation **IUMLActivity::IsActivityReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLActivity::SingleExecution**

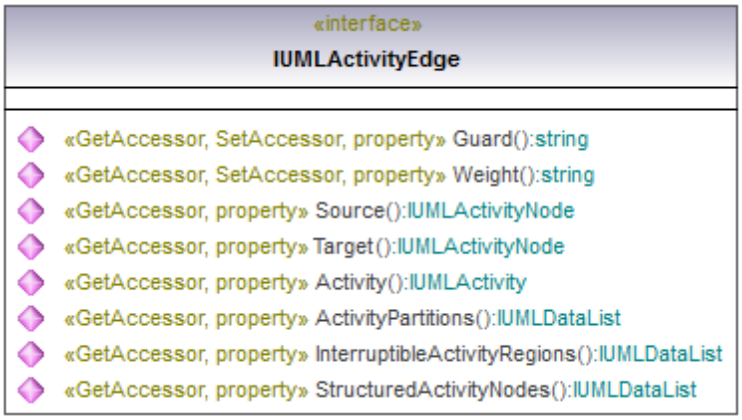
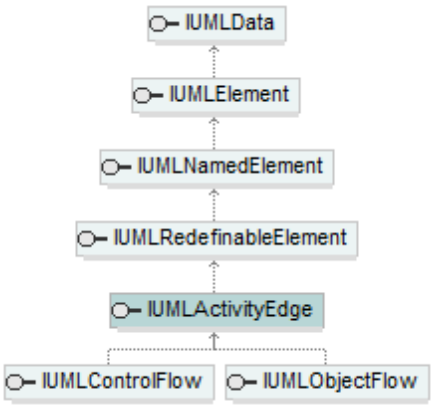
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - IUMLActivityEdge

Interface **IUMLActivityEdge**

diagram	
hierarchy	
typedElements	Interface **IUMLActivity** Interface **IUMLActivityPartition** Interface **IUMLDataAll** Interface **IUMLInterruptibleActivityRegion** Interface **IUMLStructuredActivityNode** Operation **InsertActivityEdgeAt** Operation **InsertEdgeAt** Operation **InsertActivityEdgeAt** **InsertEdgeAt** **InsertInterruptingEdgeAt** Operation **InsertInterruptingEdgeAt** Operation **InsertEdgeAt**

Operation **IUMLActivityEdge::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLActivityEdge::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityPartition.					

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**			
documentation	A list of elements of type IUMLInterruptibleActivityRegion.					

Operation **IUMLActivityEdge::Source**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLActivityEdge::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStructuredActivityNode.					

Operation **IUMLActivityEdge::Target**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLActivityEdge::Weight**

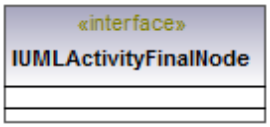
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

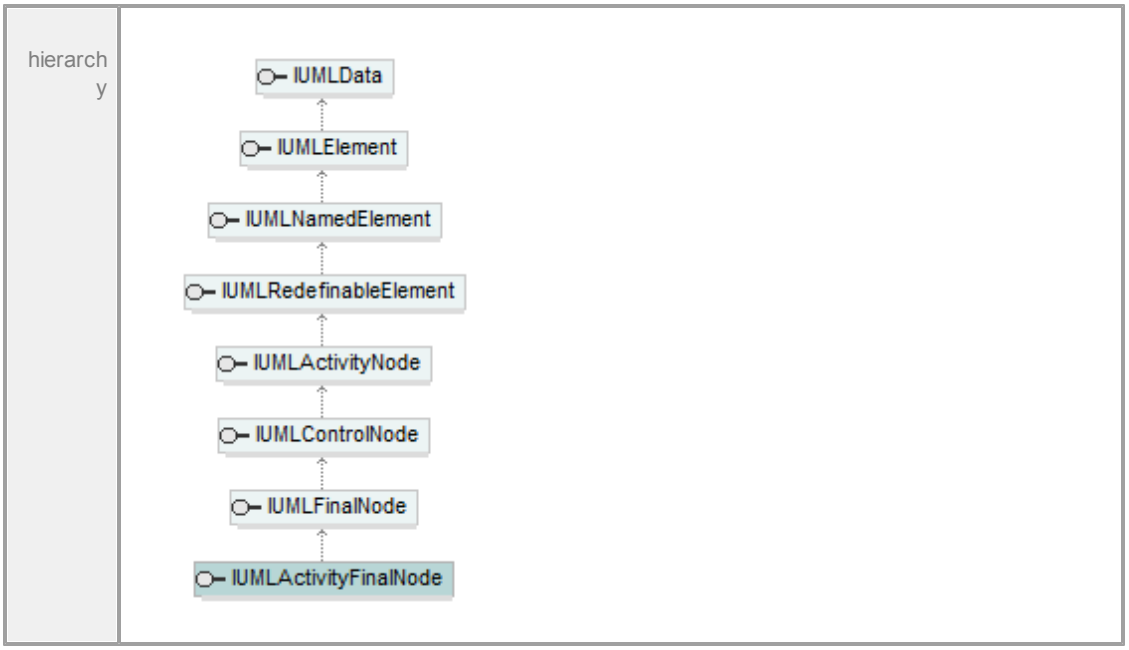
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActivityFinalNode

Interface **IUMLActivityFinalNode**

diagram

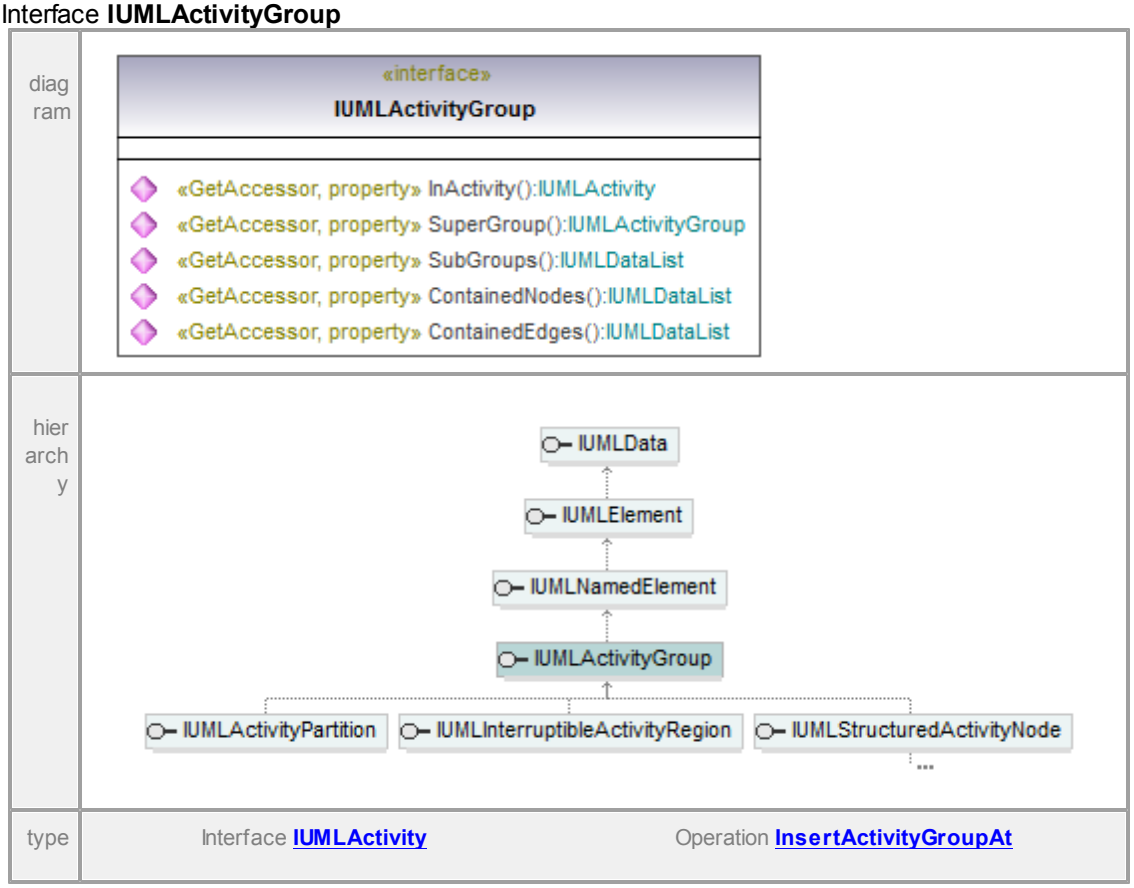




UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActivityGroup

Interface **IUMLActivityGroup**



dElements	Interface **IUMLActivityGroup** Interface **IUMLDataAll**	Operation **SuperGroup** Operation **InsertActivityGroupAt SuperGroup**

Operation **IUMLActivityGroup::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLActivityGroup::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityNode.					

Operation **IUMLActivityGroup::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLActivityGroup::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityGroup.					

Operation **IUMLActivityGroup::SuperGroup**

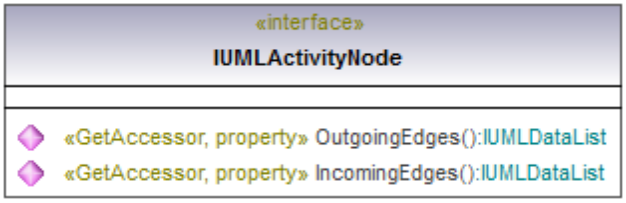
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityGroup**			

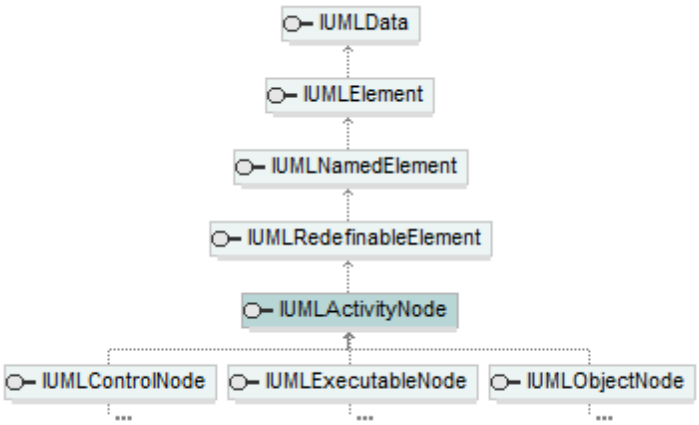
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActivityNode

Interface **IUMLActivityNode**

diagram



hierarchy	
typedElements	Interface **IUMLActivity** — Operation **InsertActivityEdgeAt** **InsertActivityNodeAt** Interface **IUMLActivityEdge** — Operation **Source** **Target** Interface **IUMLActivityPartition** — Operation **InsertNodeAt** Interface **IUMLDataAll** — Operation **InsertActivityEdgeAt** **InsertActivityNodeAt** **InsertNodeAt** **Source** **Target** Interface **IUMLInterruptibleActivityRegion** — Operation **InsertNodeAt** Interface **IUMLStructuredActivityNode** — Operation **InsertNodeAt**

Operation **IUMLActivityNode::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

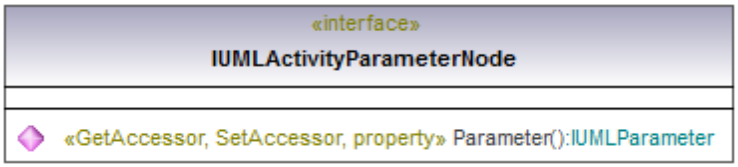
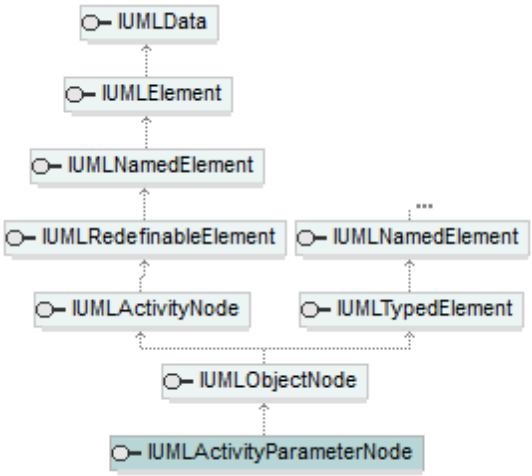
Operation **IUMLActivityNode::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLActivityParameterNode

Interface **IUMLActivityParameterNode**

diagram	
hierarchy	

Operation **IUMLActivityParameterNode::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLActivityPartition

Interface **IUMLActivityPartition**

diagram	«interface» **IUMLActivityPartition** InsertSubPartitionAt(in nIdx:int, in strKind:string):IUMLActivityPartition InsertNodeAt(in nIdx:int, in ipVal:IUMLActivityNode):void EraseNodeAt(in nIdx:int):void InsertEdgeAt(in nIdx:int, in ipEdge:IUMLActivityEdge):void EraseEdgeAt(in nIdx:int):void «GetAccessor, SetAccessor, property» IsDimension():bool «GetAccessor, SetAccessor, property» IsExternal():bool «GetAccessor, SetAccessor, property» IsHorizontal():bool «GetAccessor, property» SuperPartition():IUMLActivityPartition «GetAccessor, property» SubPartitions():IUMLDataList «GetAccessor, property» Nodes():IUMLDataList «GetAccessor, property» Edges():IUMLDataList
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLActivityGroup IUMLActivityPartition
typedElements	Interface **IUMLActivityPartition** — Operation **InsertSubPartitionAt** **SuperPartition** Interface **IUMLDataAll** — Operation **InsertSubPartitionAt** **SuperPartition**

Operation **IUMLActivityPartition::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

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**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLActivityPartition::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityPartition**			

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**			
documentation	A list of elements of type IUMLActivityNode.					

Operation **IUMLActivityPartition::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityPartition.					

© 2014 Altova GmbH

Operation **IUMLActivityPartition::SuperPartition**

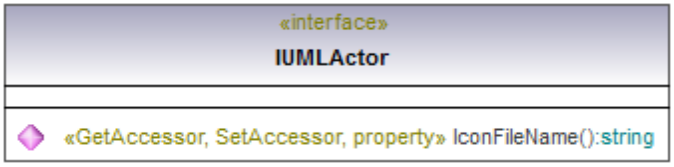
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityPartition**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

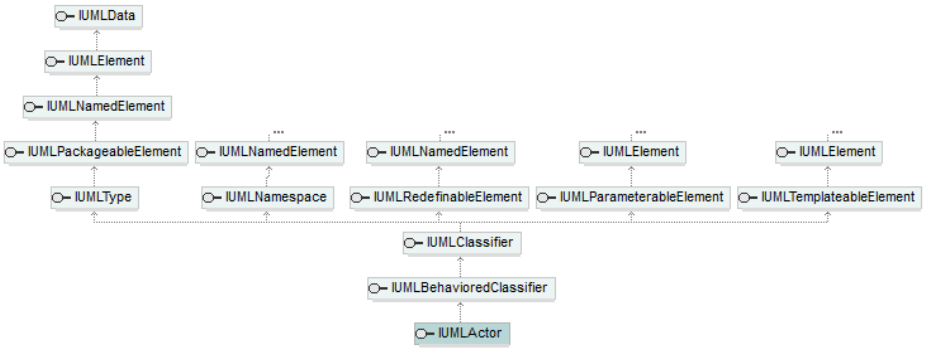
UModelAPI - IUMLActor

Interface **IUMLActor**

diagram



hierarchy



Operation **IUMLActor::IconFileName**

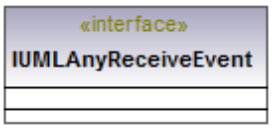
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

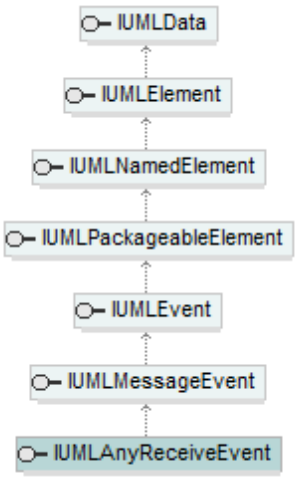
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLAnyReceiveEvent

Interface **IUMLAnyReceiveEvent**

diagram

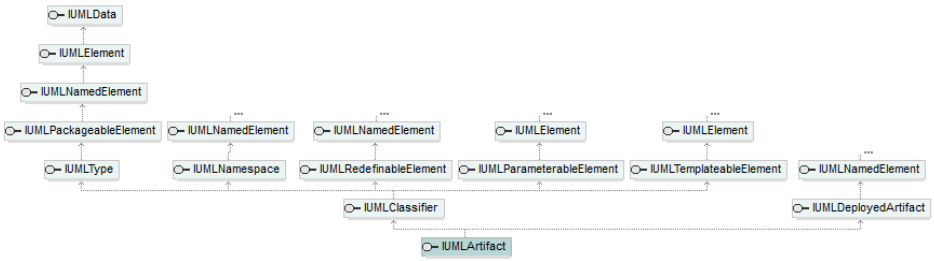


hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLArtifact

Interface **IUMLArtifact**

diagram	«interface» **IUMLArtifact** InsertManifestationAt(in nIdx:int, in ipUtilizedElement:IUMLPackageableElement):IUMLManifestation InsertNestedArtifactAt(in nIdx:int):IUMLArtifact InsertOwnedOperationAt(in nIdx:int):IUMLOperation InsertOwnedAttributeAt(in nIdx:int):IUMLProperty «GetAccessor, SetAccessor, property» FileName():string «GetAccessor, property» Manifestations():IUMLDataList «GetAccessor, property» NestedArtifacts():IUMLDataList «GetAccessor, property» OwnedAttributes():IUMLDataList «GetAccessor, property» OwnedOperations():IUMLDataList
hierarchy	
typedElements	Interface **IUMLArtifact** Interface **IUMLDataAll** Operation **InsertNestedArtifactAt** Operation **InsertNestedArtifactAt**

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**			
	return	**return**	**IUMLManifestation**			

Operation **IUMLArtifact::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLArtifact**			

Operation **IUMLArtifact::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLArtifact::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLArtifact::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLManifestation.					

Operation **IUMLArtifact::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLArtifact.					

Operation **IUMLArtifact::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLArtifact::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLDataList**
documentation	A list of elements of type IUMLOperation.		

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLAssociation

Interface **IUMLAssociation**

diagram	«interface» IUMLAssociation «GetAccessor, property» MemberEnds():IUMLDataList «GetAccessor, property» OwnedEnds():IUMLDataList «GetAccessor, property» NavigableOwnedEnds():IUMLDataList «GetAccessor, property» EndTypes():IUMLDataList
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLPackageableElement, IUMLNamedElement, IUMLNamedElement, IUMLElement, IUMLElement IUMLType, IUMLNamespace, IUMLRedefinableElement, IUMLParameterableElement, IUMLTemplateableElement, IUMLElement IUMLClassifier, IUMLRelationship IUMLAssociation IUMLAssociationClass
typedElements	Interface **IUMLConnector** Operation **ConnectorType** Interface **IUMLDataAll** Operation **Association** **ConnectorType** **OwningAssociation** Interface **IUMLProperty** Operation **Association** **OwningAssociation**

Operation **IUMLAssociation::EndTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLType.					

Operation **IUMLAssociation::MemberEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLAssociation::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

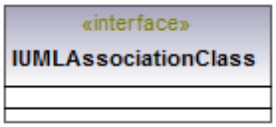
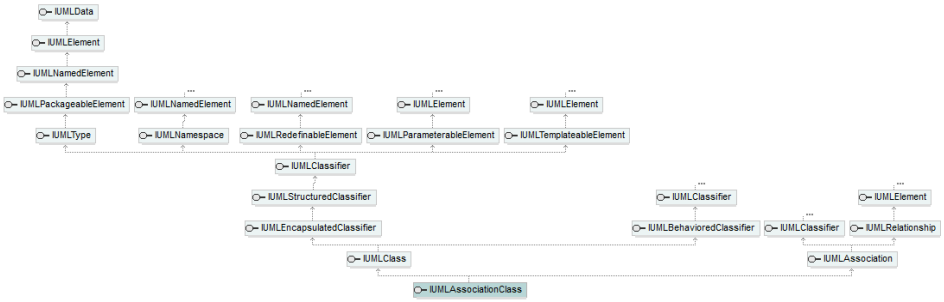
Operation **IUMLAssociation::OwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLAssociationClass

Interface **IUMLAssociationClass**

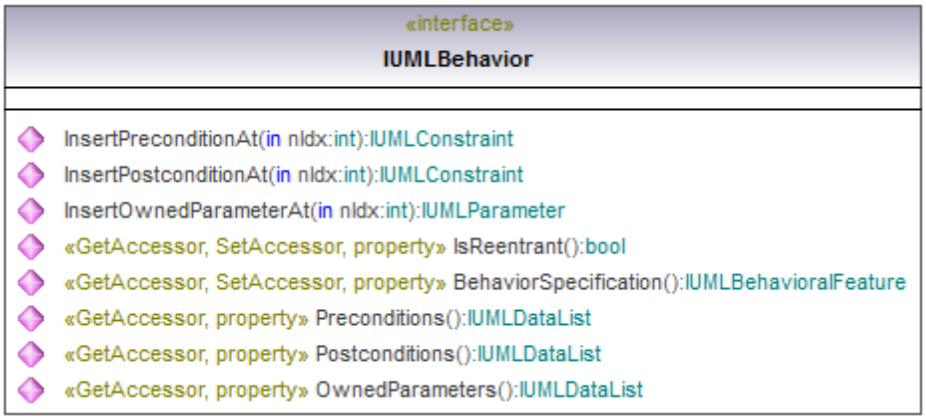
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

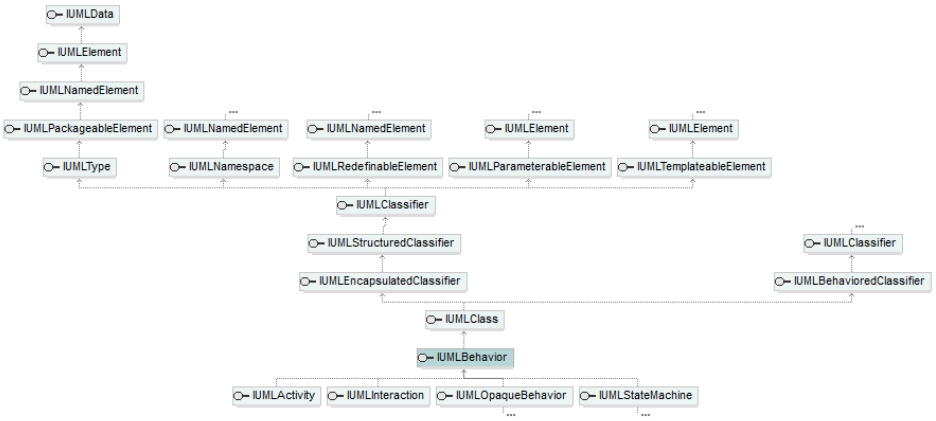
UModelAPI - IUMLBehavior

Interface **IUMLBehavior**

diagram



hierarchy



typedElements	Interface **IUMLBehavioredClassifier**	Operation **InsertOwnedBehaviorAt**
	Interface **IUMLBehaviorExecutionSpecification**	Operation **BehaviorExecution**
	Interface **IUMLCallBehaviorAction**	Operation **Behavior**
	Interface **IUMLDataAll**	Operation **Behavior** **BehaviorExecution** **DecisionInput** **DoActivity** **Effect** **Entry** **Exit** **InsertOwnedBehaviorAt** **Selection** **SetNewDoActivity** **SetNewEffect** **SetNewEntry** **SetNewExit** **Transformation**
	Interface **IUMLDecisionNode**	Operation **DecisionInput**
	Interface **IUMLObjectFlow**	Operation **Transformation**
	Interface **IUMLObjectNode**	Operation **Selection**
	Interface **IUMLState**	Operation **DoActivity** **Entry** **Exit** **SetNewDoActivity** **SetNewEntry** **SetNewExit**
	Interface **IUMLTransition**	Operation **Effect** **SetNewEffect**

Operation **IUMLBehavior::BehaviorSpecification**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavioralFeature**			

Operation **IUMLBehavior::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLParameter**			

Operation **IUMLBehavior::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLBehavior::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

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**			
documentation	A list of elements of type IUMLParameter.					

Operation **IUMLBehavior::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

Operation **IUMLBehavior::Preconditions**

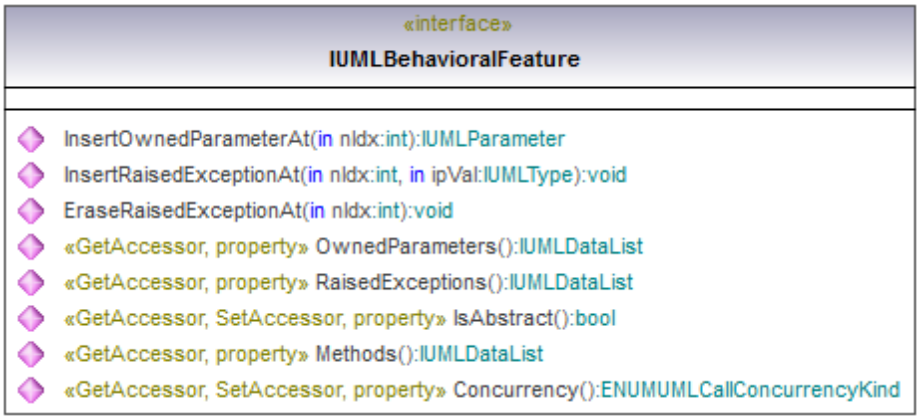
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

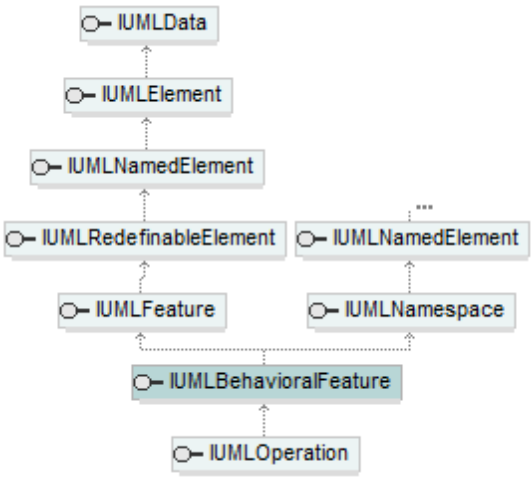
UModelAPI - IUMLBehavioralFeature

Interface **IUMLBehavioralFeature**

diagram



hierarchy



typedElements		
	Interface **IUMLBehavior**	Operation **BehaviorSpecification**
	Interface **IUMLDataAll**	Operation **BehaviorSpecification**

Operation **IUMLBehavioralFeature::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLCallConcurrencyKind**			

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**			

 © 2014 Altova GmbH

	return	**return**	**IUMLParameter**

Operation **IUMLBehavioralFeature::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLType**			
	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**			
documentation	A list of elements of type IUMLBehavior.					

Operation **IUMLBehavioralFeature::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLParameter.					

Operation **IUMLBehavioralFeature::RaisedExceptions**

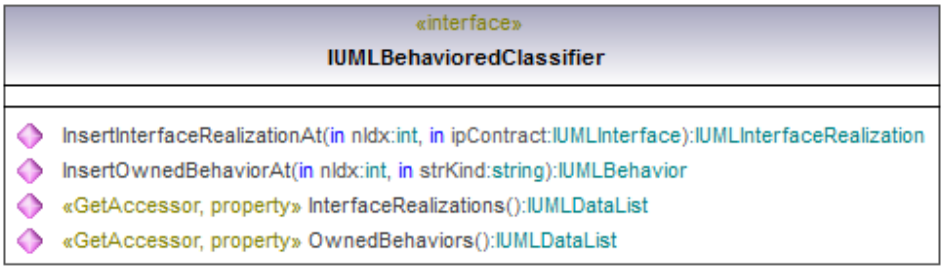
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLType.					

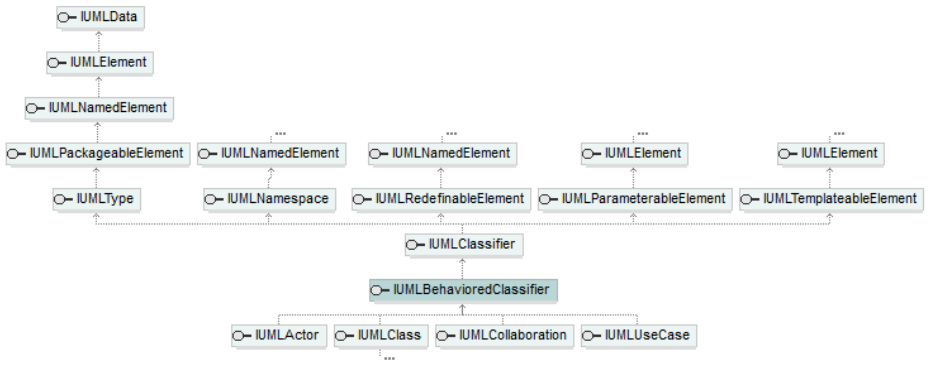
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLBehavioredClassifier

Interface **IUMLBehavioredClassifier**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInterfaceRealization** Operation **ImplementingClassifier** Operation **ImplementingClassifier**

Operation **IUMLBehavioredClassifier::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipContract	**in**	**IUMLInterface**			
	return	**return**	**IUMLInterface Realization**			

Operation **IUMLBehavioredClassifier::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLBehavioredClassifier::InterfaceRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInterfaceRealizations.					

Operation **IUMLBehavioredClassifier::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLBehavior.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

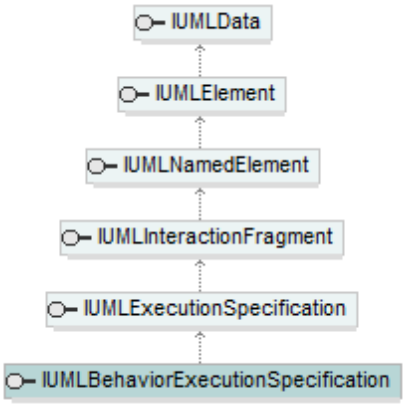
 © 2014 Altova GmbH

UModelAPI - IUMLBehaviorExecutionSpecification

Interface **IUMLBehaviorExecutionSpecification**

diagram

hierarchy



Operation **IUMLBehaviorExecutionSpecification::BehaviorExecution**

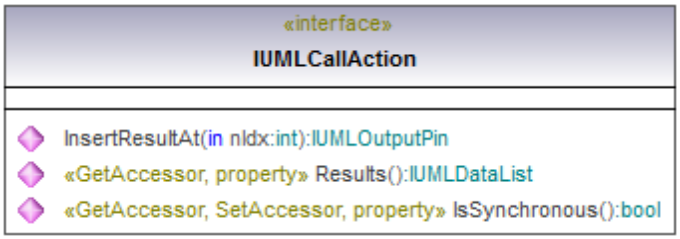
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

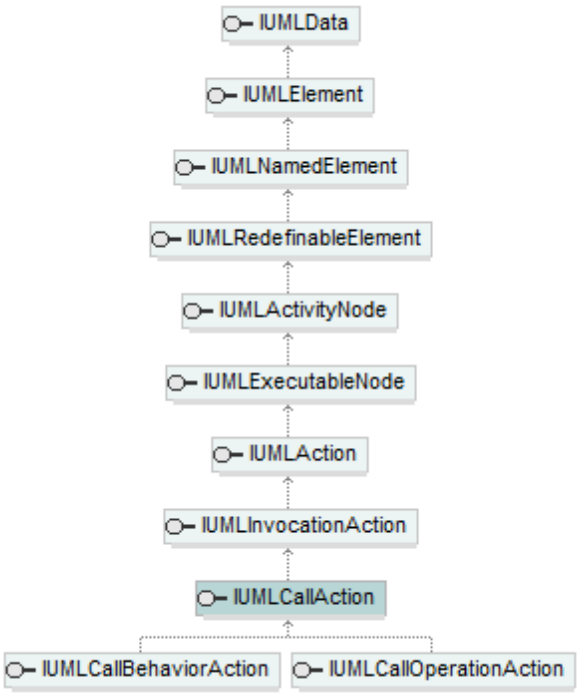
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLCallAction

Interface **IUMLCallAction**

diagram



hierarchy	

Operation **IUMLCallAction::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLCallAction::IsSynchronous**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

Operation **IUMLCallAction::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOutputPin.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

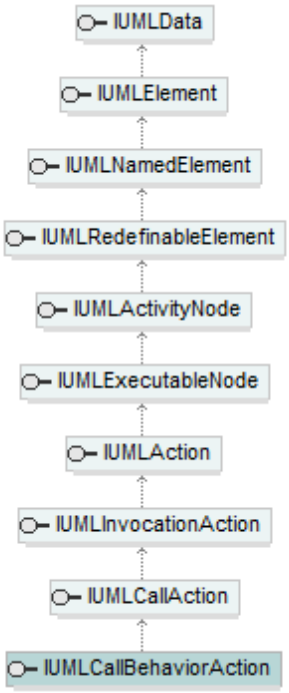
 © 2014 Altova GmbH

UModelAPI - IUMLCallBehaviorAction

Interface **IUMLCallBehaviorAction**

diagram

hierarchy



Operation **IUMLCallBehaviorAction::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

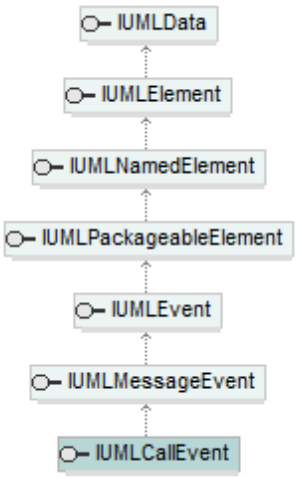
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLCallEvent

Interface **IUMLCallEvent**

diagram

hierarchy



Operation **IUMLCallEvent::Operation**

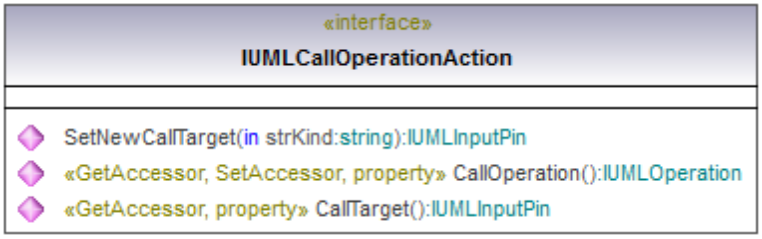
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

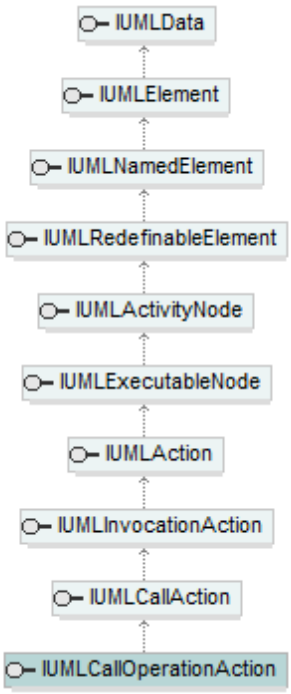
UModelAPI - IUMLCallOperationAction

Interface **IUMLCallOperationAction**

diagram



 © 2014 Altova GmbH

hierarchy	

Operation **IUMLCallOperationAction::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLCallOperationAction::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

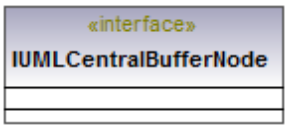
Operation **IUMLCallOperationAction::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

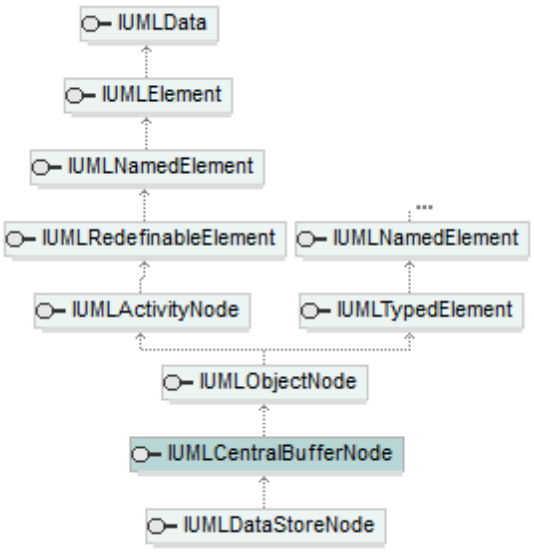
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLCentralBufferNode

Interface **IUMLCentralBufferNode**

diagram	

hierarchy

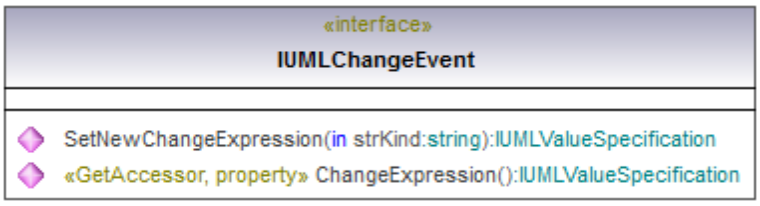


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

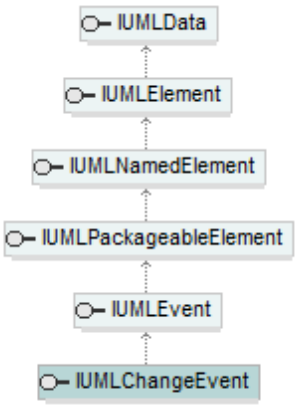
UModelAPI - IUMLChangeEvent

Interface **IUMLChangeEvent**

diagram



hierarchy



Operation **IUMLChangeEvent::ChangeExpression**

paramete	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLValueSpecification**			

Operation **IUMLChangeEvent::SetNewChangeExpression**

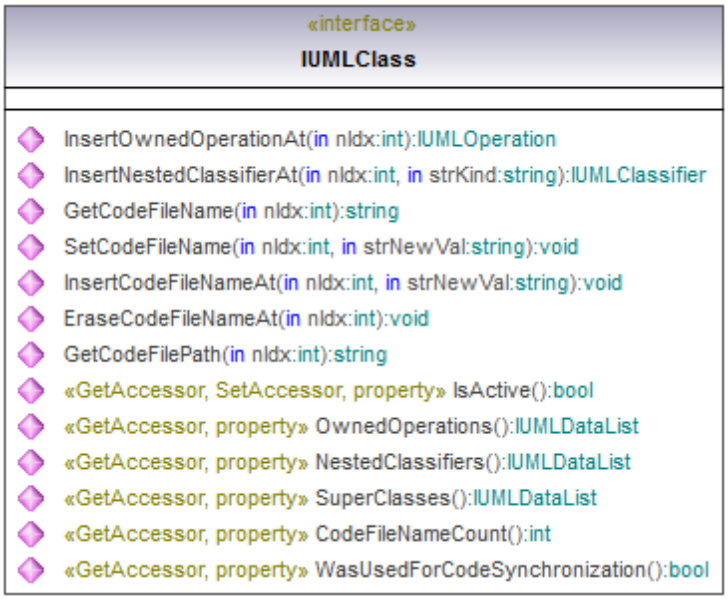
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLClass

Interface **IUMLClass**

diagram



hierarchy

typedElements

Interface **IUMLClassifier**	Operation **Class**
Interface **IUMLDataAll**	Operation **Class**
Interface **IUMLOperation**	Operation **Class**
Interface **IUMLProperty**	Operation **Class**

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**			
	return	**return**	**IUMLClassifier**			

Operation **IUMLClass::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

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**			

 © 2014 Altova GmbH

documentation	A list of elements of type IUMLClassifier.

Operation **IUMLClass::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOperation.					

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**			
documentation	A list of elements of type IUMLClass.					

Operation **IUMLClass::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLClassifier

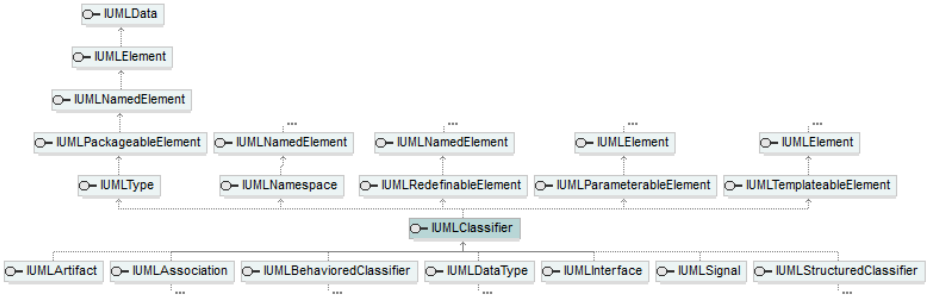
Interface **IUMLClassifier**

diagram

«interface»
IUMLClassifier

- InsertGeneralizationAt(in nIdx:int, in ipGeneral:IUMLClassifier):IUMLGeneralization
- InsertOwnedUseCaseAt(in nIdx:int):IUMLUseCase
- InsertCollaborationUseAt(in nIdx:int):IUMLCollaborationUse
- «GetAccessor, property» Generalizations():IUMLDataList
- «GetAccessor, property» Generals():IUMLDataList
- «GetAccessor, property» Specifics():IUMLDataList
- «GetAccessor, property» Features():IUMLDataList
- «GetAccessor, property» NestingInterface():IUMLInterface
- «GetAccessor, property» Attributes():IUMLDataList
- «GetAccessor, property» InheritedMembers():IUMLDataList
- «GetAccessor, property» OwnedUseCases():IUMLDataList
- «GetAccessor, property» UseCases():IUMLDataList
- «GetAccessor, property» CollaborationUses():IUMLDataList
- «GetAccessor, SetAccessor, property» IsFinalSpecialization():bool
- «GetAccessor, property» Class():IUMLClass
- «GetAccessor, SetAccessor, property» IsAbstract():bool

hierarchy



typedElements

Interface **IUMLAction**
Interface **IUMLClass**
Interface **IUMLClassifier**
Interface **IUMLClassifierTemplateParameter**
Interface **IUMLComponent**
Interface **IUMLComponentRealization**
Interface **IUMLDataAll**

Operation **ActionContext**
Operation **InsertNestedClassifierAt**
Operation **InsertGeneralizationAt**
Operation **InsertConstrainingClassifierAt**

Operation **InsertRealizationAt**
Operation **RealizingClassifier**

Operation **ActionContext** **Classifier** **General** **InsertConstrainingClassifierAt** **InsertConveyedAt** **InsertExceptionTypeAt** **InsertGeneralizationAt** **InsertNestedClassifierAt** **InsertRealizationAt** **InsertSubjectAt**

 © 2014 Altova GmbH

	RealizingClassifier Specific Interface IUMLExceptionHandler Operation InsertExceptionTypeAt Interface IUMLGeneralization Operation General Specific Interface IUMLInformationFlow Operation InsertConveyedAt Interface IUMLInstanceSpecification Operation Classifier Interface IUMLInterface Operation InsertNestedClassifierAt Interface IUMLProperty Operation Classifier Interface IUMLUseCase Operation InsertSubjectAt

Operation **IUMLClassifier::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLClassifier::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLClassifier::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLCollaborationUse.					

Operation **IUMLClassifier::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLFeature.					

Operation **IUMLClassifier::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGeneralization.					

Operation **IUMLClassifier::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLClassifier::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLClassifier::InsertCollaborationUseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLCollaborationUse**			

Operation **IUMLClassifier::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipGeneral	**in**	**IUMLClassifier**			
	return	**return**	**IUMLGeneralization**			

Operation **IUMLClassifier::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLUseCase**			

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**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLClassifier::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLUseCase.					

Operation **IUMLClassifier::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

© 2014 Altova GmbH

Operation **IUMLClassifier::UseCases**

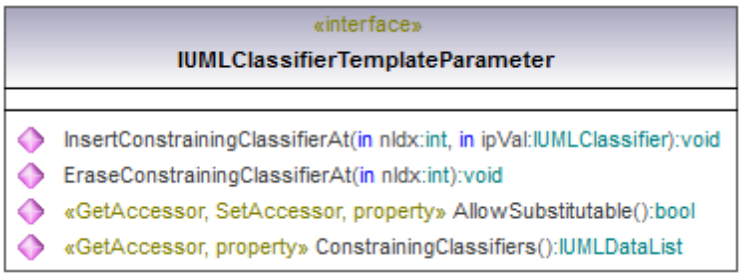
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLUseCase.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

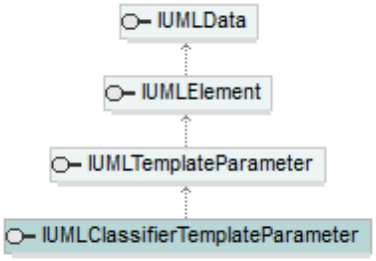
UModelAPI - IUMLClassifierTemplateParameter

Interface **IUMLClassifierTemplateParameter**

diagram



hierarchy



typedElements		
	Interface **IUMLDataAll**	Operation **InsertOwnedTemplateParameterAt**
	Interface **IUMLTemplateSignature**	Operation **InsertOwnedTemplateParameterAt**

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**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLClassifierTemplateParameter::EraseConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLClassifierTemplateParameter::InsertConstrainingClassifierAt**

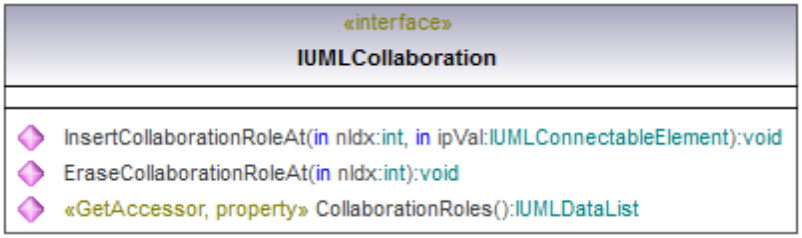
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

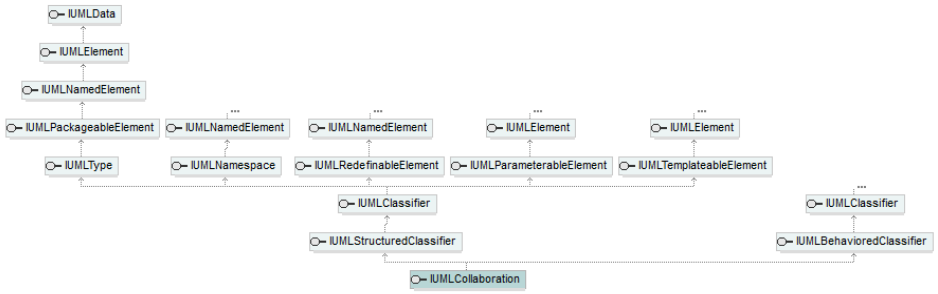
UModelAPI - IUMLCollaboration

Interface **IUMLCollaboration**

diagram



hierarchy



typedElements		
	Interface **IUMLCollaborationUse**	Operation **CollaborationType**
	Interface **IUMLDataAll**	Operation **CollaborationType**

Operation **IUMLCollaboration::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

documentation	
	A list of elements of type IUMLCollaboration.

Operation **IUMLCollaboration::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

 © 2014 Altova GmbH

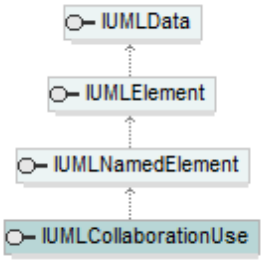
Operation **IUMLCollaboration::InsertCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnectableElement**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLCollaborationUse

Interface **IUMLCollaborationUse**

diagram	
hierarchy	
typedElements	Interface **IUMLClassifier** Interface **IUMLDataAll** Operation **InsertCollaborationUseAt** Operation **InsertCollaborationUseAt**

Operation **IUMLCollaborationUse::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLCollaboration**			

Operation **IUMLCollaborationUse::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLDependency.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLCombinedFragment

Interface **IUMLCombinedFragment**

diagram	«interface» **IUMLCombinedFragment** InsertOperandAt(in nIdx:int):IUMLInteractionOperand «GetAccessor, SetAccessor, property» InteractionOperator():ENUMUMLInteractionOperatorKind «GetAccessor, property» Operands():IUMLDataList
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLInteractionFragment IUMLCombinedFragment

Operation **IUMLCombinedFragment::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInteractionOperand**			

Operation **IUMLCombinedFragment::InteractionOperator**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLInteractionOperatorKind**			

Operation **IUMLCombinedFragment::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInteractionOperand.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLComment

Interface **IUMLComment**

diagram	«interface» **IUMLComment** InsertAnnotatedElementAt(in nIdx:int, in ipVal:IUMLElement):void EraseAnnotatedElementAt(in nIdx:int):void InsertOwnedCommentTextHyperlinkAt(in nFromTextPos:int, in nToTextPos:int, in strAddress:string):IUMLCommentTextHyperlink «GetAccessor, SetAccessor, property» Body():string «GetAccessor, property» OwningElement():IUMLElement «GetAccessor, property» AnnotatedElements():IUMLDataList «GetAccessor, property» OwnedHyperlinks():IUMLDataList
hierarchy	IUMLData IUMLElement IUMLComment
typedElements	Interface **IUMLDataAll** — Operation **InsertOwnedCommentAt**, **OwnedDocComment** Interface **IUMLElement** — Operation **InsertOwnedCommentAt**, **OwnedDocComment**

Operation **IUMLComment::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

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**			
	ipVal	**in**	**IUMLElement**			
	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**	**IUMLComment TextHyperlink**

Operation **IUMLComment::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLComment::OwningElement**

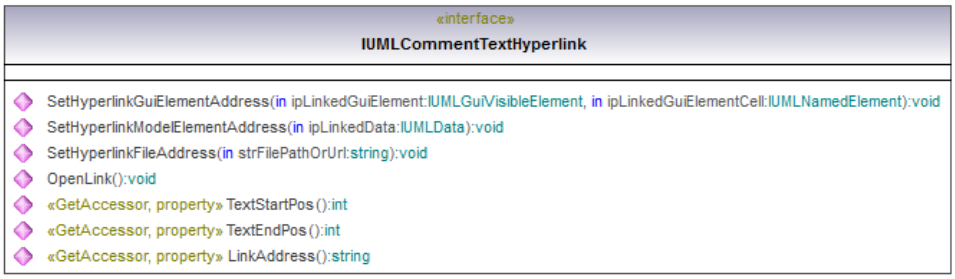
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

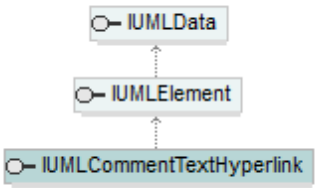
UModelAPI - IUMLCommentTextHyperlink

Interface **IUMLCommentTextHyperlink**

diagram



hierarchy



typedElements

Interface **IUMLComment**	Operation **InsertOwnedCommentTextHyperlinkAt**
Interface **IUMLDataAll**	Operation **InsertOwnedCommentTextHyperlinkAt**

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**			

 © 2014 Altova GmbH

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		**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell		**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **IUMLCommentTextHyperlink::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLCommentTextHyperlink::TextStartPos**

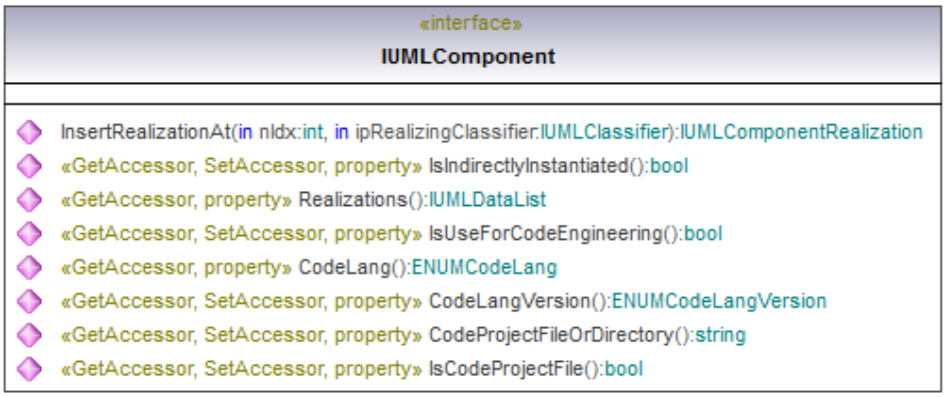
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

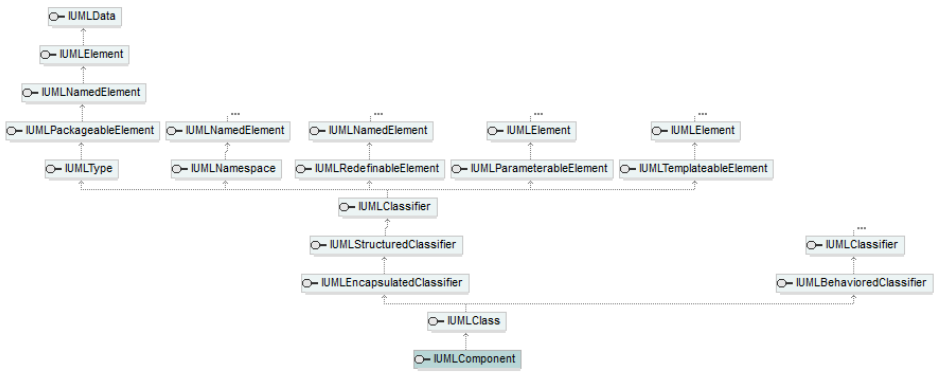
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLComponent

Interface **IUMLComponent**

diagram



hierarchy	
typedElements	Interface **IUMLComponentRealization** Interface **IUMLDataAll** Operation **Abstraction** Operation **Abstraction**

Operation **IUMLComponent::CodeLang**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLang**			

Operation **IUMLComponent::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

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**			
	return	**return**	**IUMLComponentRealization**			

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

 © 2014 Altova GmbH

r	**return**	**return**	**bool**

Operation **IUMLComponent::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLComponentRealization.					

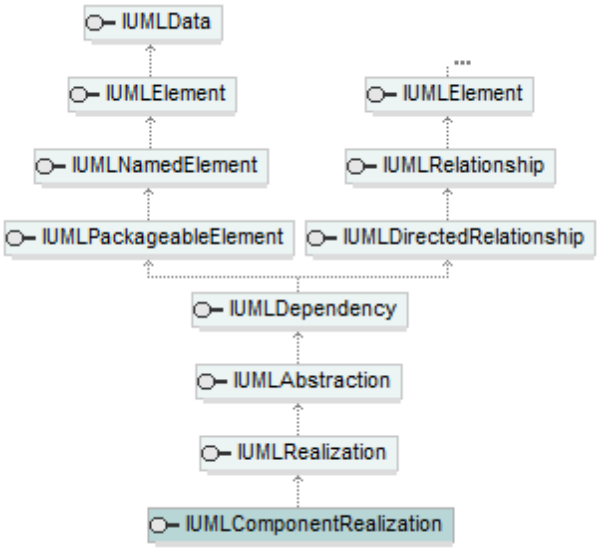
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLComponentRealization

Interface **IUMLComponentRealization**

diagram

hierarchy



typedElements	Interface **IUMLComponent**	Operation **InsertRealizationAt**
	Interface **IUMLDataAll**	Operation **InsertRealizationAt**

Operation **IUMLComponentRealization::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComponent**			

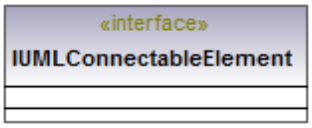
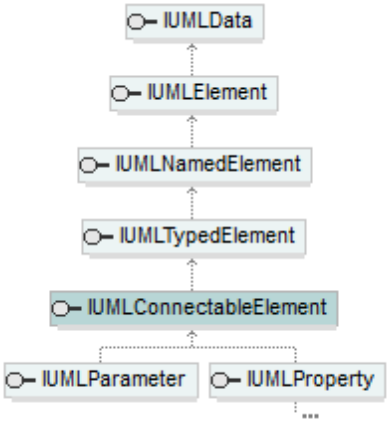
Operation **IUMLComponentRealization::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLConnectableElement

Interface **IUMLConnectableElement**

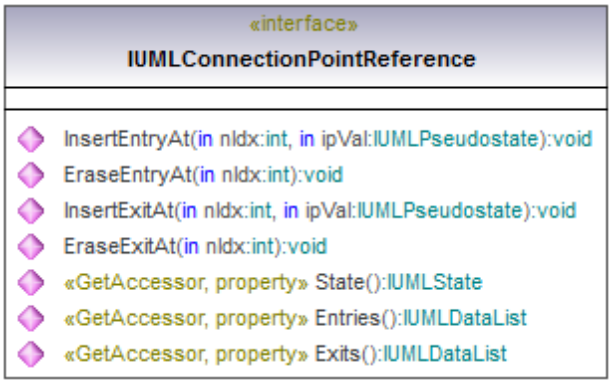
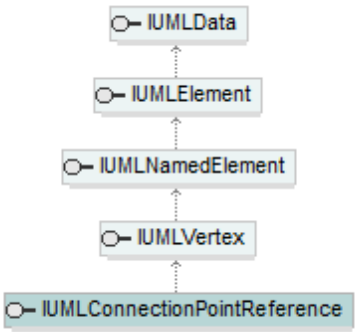
diagram	
hierarchy	
typedElements	Interface **IUMLCollaboration** Interface **IUMLConnectorEnd** Interface **IUMLDataAll** Interface **IUMLLifeline** Interface **IUMLStructuredClassifier** Operation **InsertCollaborationRoleAt** Operation **Role** Operation **InsertCollaborationRoleAt** **InsertOwnedConnectorAt** **Represents** **Role** Operation **Represents** Operation **InsertOwnedConnectorAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLConnectionPointReference

Interface **IUMLConnectionPointReference**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLState** Operation **InsertConnectionAt** Operation **InsertConnectionAt**

Operation **IUMLConnectionPointReference::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

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

r	**return**	**return**	**IUMLDataList**
documentation	A list of elements of type IUMLPseudostate.		

Operation **IUMLConnectionPointReference::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

Operation **IUMLConnectionPointReference::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

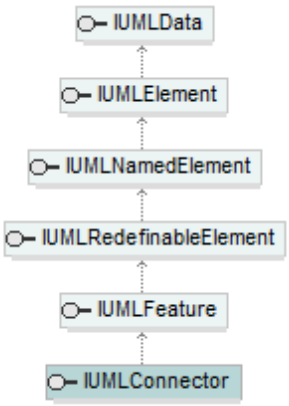
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLConnector

Interface **IUMLConnector**

diagram

hierarchy



 © 2014 Altova GmbH

typedElements	Interface **IUMLDataAll** Interface **IUMLInformationFlow** Interface **IUMLStructuredClassifier**	Operation **InsertOwnedConnectorAt** **InsertRealizingConnectorAt** Operation **InsertRealizingConnectorAt** Operation **InsertOwnedConnectorAt**

Operation **IUMLConnector::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLConnectorKind**			
documentation	**Deprecated:** Since UML2.4 (UModel2010r2) 'ConnectorKind' is derived and cannot be set anymore.					

Operation **IUMLConnector::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

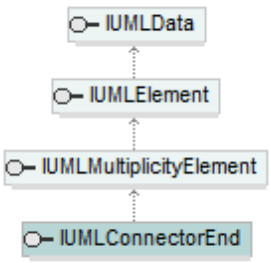
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLConnectorEnd

Interface **IUMLConnectorEnd**

diagram

hierarchy



Operation **IUMLConnectorEnd::Role**

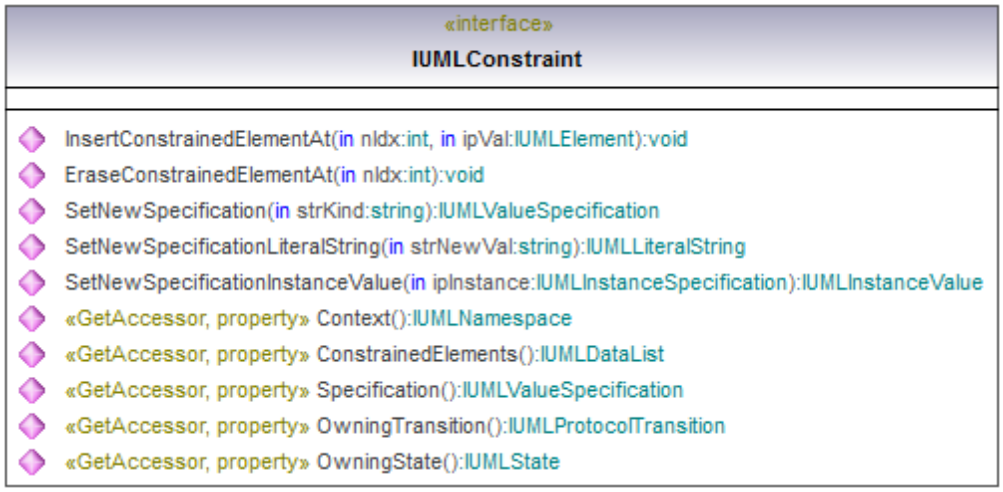
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

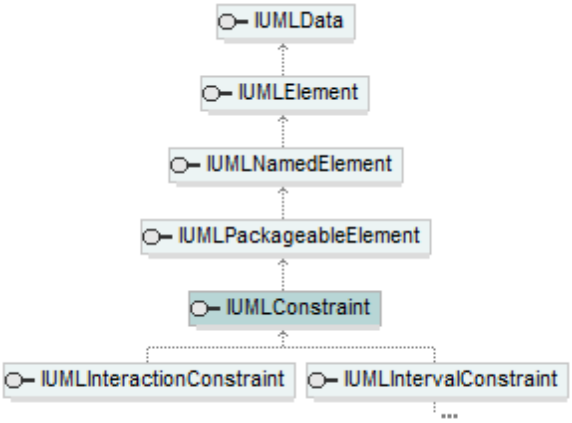
UModelAPI - IUMLConstraint

Interface **IUMLConstraint**

diagram



hierarchy



typed Elements

Interface **IUMLAction**	Operation **InsertLocalPostConditionAt** **InsertLocalPreConditionAt**
Interface **IUMLBehavior**	Operation **InsertPostconditionAt** **InsertPreconditionAt**
Interface **IUMLDataAll**	Operation **InsertLocalPostConditionAt** **InsertLocalPreConditionAt** **InsertOwnedRuleAt** **InsertPostconditionAt** **InsertPreconditionAt** **Invariant** **OwningConstraint** **PostCondition** **PreCondition** **SetNewInvariant** **SetNewPostCondition** **SetNewPreCondition** **SetNewStateInvariant** **SetNewTransitionGuard** **StateInvariant** **TransitionGuard**
Interface **IUMLNamespace**	Operation **InsertOwnedRuleAt**
Interface **IUMLProtocolTransition**	Operation **PostCondition** **PreCondition** **SetNewPostCondition** **SetNewPreCondition**

 © 2014 Altova GmbH

	Interface **IUMLState** Interface **IUMLStateInvariant** Interface **IUMLTransition** Interface **IUMLValueSpecification**	Operation **SetNewStateInvariant StateInvariant** Operation **Invariant SetNewInvariant** Operation **SetNewTransitionGuard TransitionGuard** Operation **OwningConstraint**

Operation **IUMLConstraint::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

Operation **IUMLConstraint::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

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**			
	return	**return**	**void**			

Operation **IUMLConstraint::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLConstraint::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolTransition**			

Operation **IUMLConstraint::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLConstraint::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			

	return	**return**	**IUMLInstanceValue**

Operation **IUMLConstraint::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLConstraint::Specification**

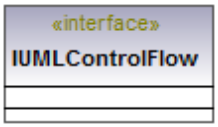
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

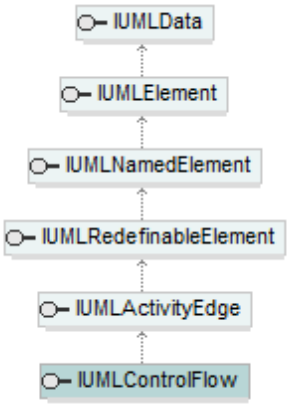
UModelAPI - IUMLControlFlow

Interface **IUMLControlFlow**

diagram



hierarchy

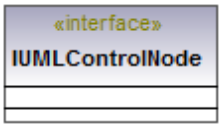


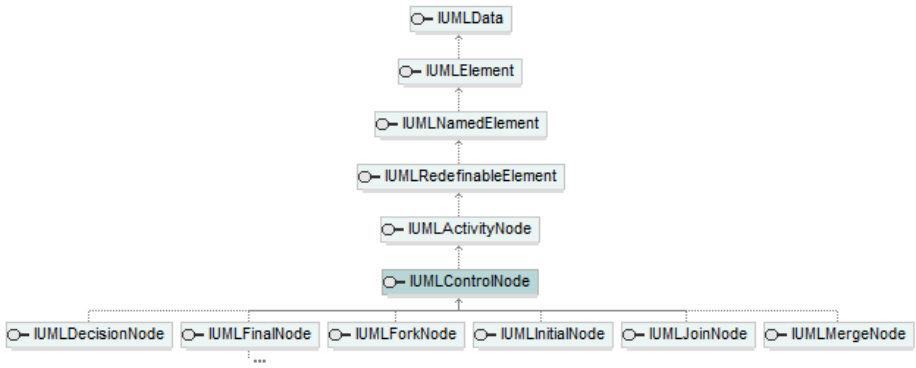
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLControlNode

Interface **IUMLControlNode**

diagram

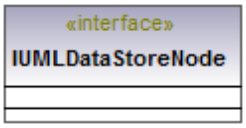
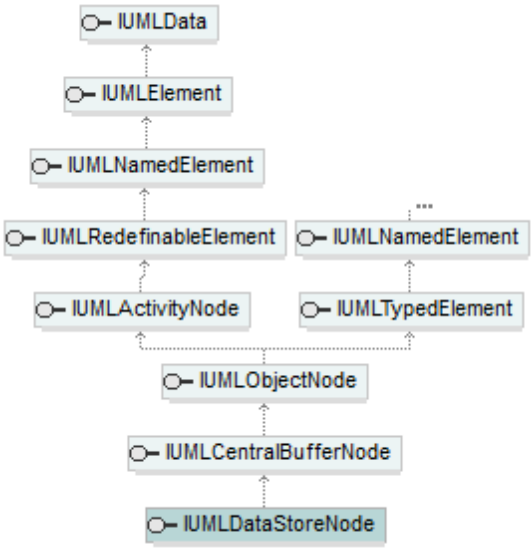


hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDataStoreNode

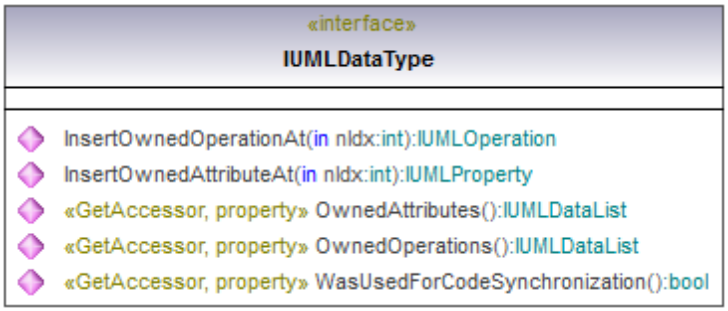
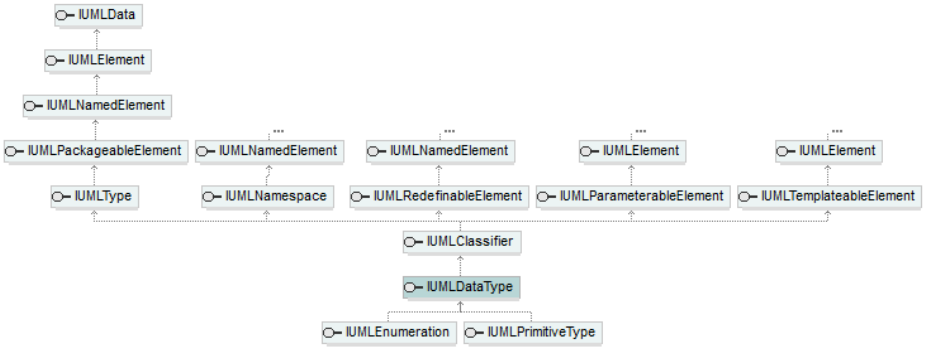
Interface **IUMLDataStoreNode**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDataType

Interface **IUMLDataType**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLOperation** Interface **IUMLProperty** Operation **Datatype** Operation **Datatype** Operation **Datatype**

Operation **IUMLDataType::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLDataType::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLDataType::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLDataType::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

 © 2014 Altova GmbH

documentation	A list of elements of type IUMLOperation.

Operation **IUMLDataType::WasUsedForCodeSynchronization**

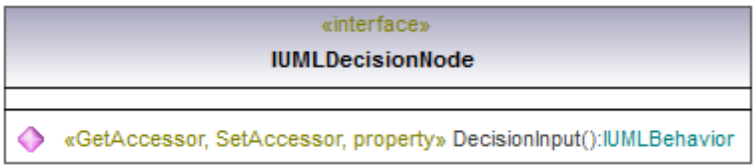
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

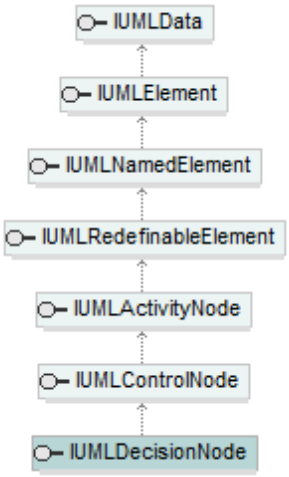
UModelAPI - IUMLDecisionNode

Interface **IUMLDecisionNode**

diagram



hierarchy



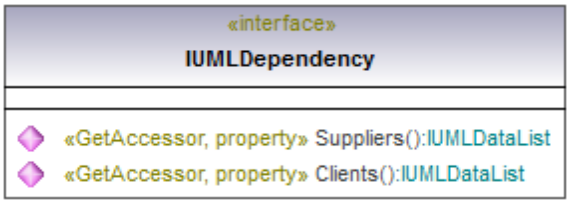
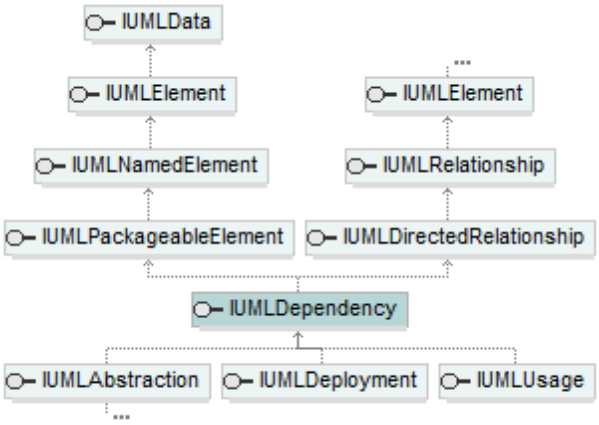
Operation **IUMLDecisionNode::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDependency

Interface **IUMLDependency**

diagram	
hierarchy	

Operation **IUMLDependency::Clients**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLDependency::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

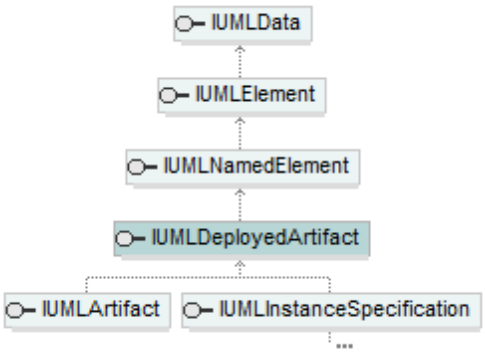
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDeployedArtifact

Interface **IUMLDeployedArtifact**

diagram	

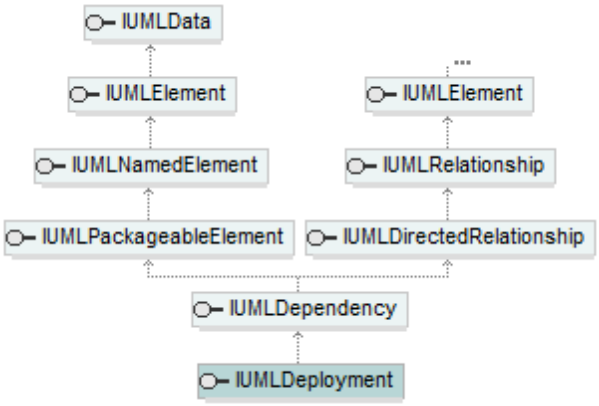
 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLDeploymentTarget** Operation **InsertDeploymentAt** Operation **InsertDeploymentAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDeployment

Interface **IUMLDeployment**

diagram	«interface» **IUMLDeployment** «GetAccessor, property» Location():IUMLDeploymentTarget
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLDeploymentTarget** Operation **InsertDeploymentAt** Operation **InsertDeploymentAt**

Operation **IUMLDeployment::Location**

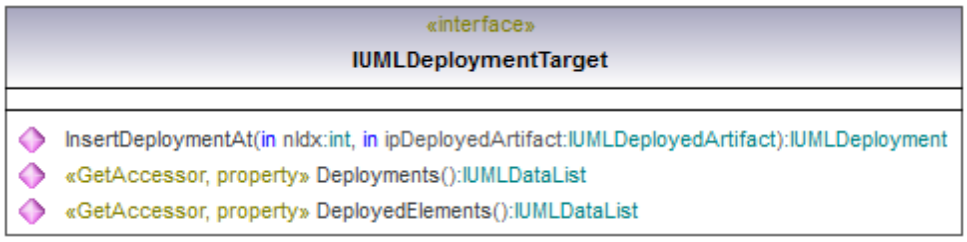
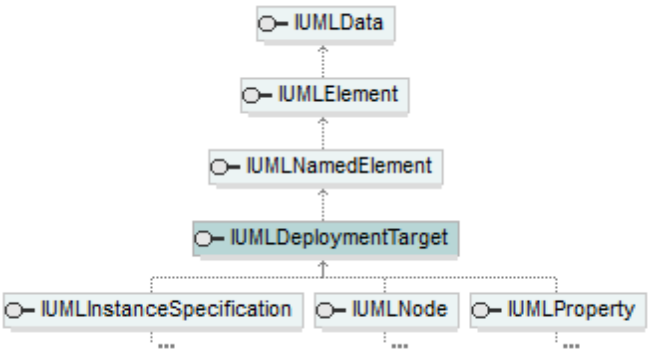
parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLDeploymentTarget**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDeploymentTarget

Interface **IUMLDeploymentTarget**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Operation **Location** Interface **IUMLDeployment** Operation **Location**

Operation **IUMLDeploymentTarget::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageableElement.					

Operation **IUMLDeploymentTarget::Deployments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLDeployment.					

Operation **IUMLDeploymentTarget::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

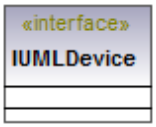
r	**nIdx**	**in**	**int**
	ipDeployedArtifact	**in**	**IUMLDeployed Artifact**
	return	**return**	**IUMLDeployment**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

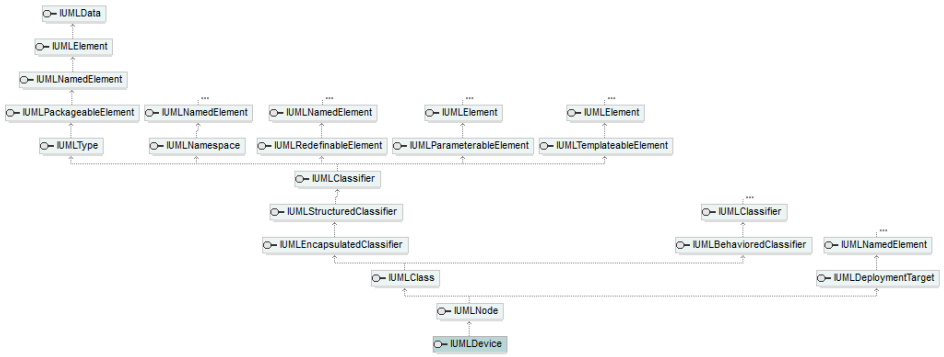
UModelAPI - IUMLDevice

Interface **IUMLDevice**

diagram



hierarchy



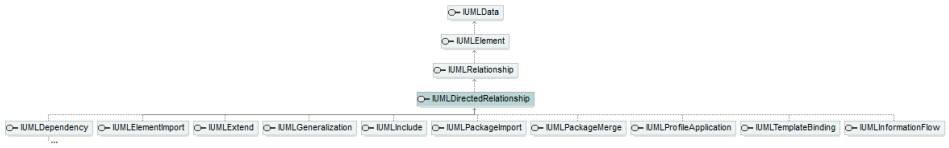
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLDirectedRelationship

Interface **IUMLDirectedRelationship**

diagram

hierarchy



Operation **IUMLDirectedRelationship::Sources**

paramete	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLDataList**
documentation	A list of elements of type IUMLElement.		

Operation **IUMLDirectedRelationship::Targets**

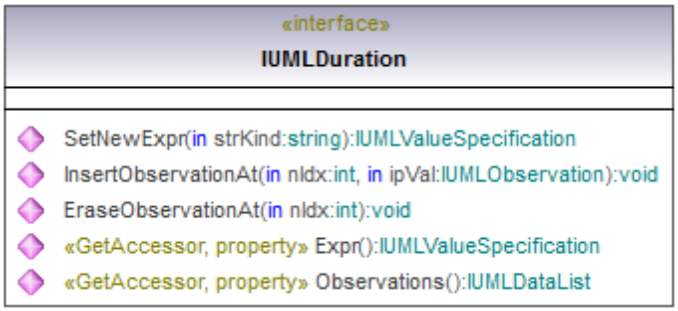
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

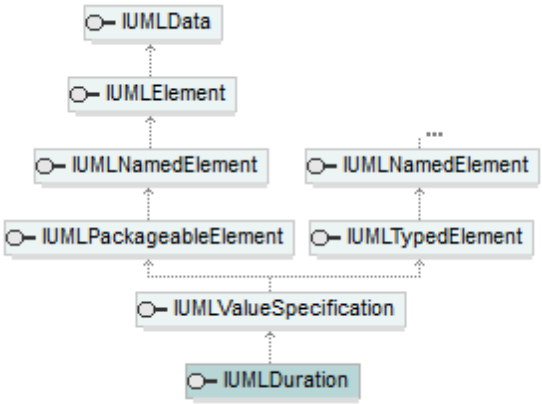
UModelAPI - IUMLDuration

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

 © 2014 Altova GmbH

r	**return**	**return**	**IUMLValueSpecification**

Operation **IUMLDuration::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation**			
	return	**return**	**void**			

Operation **IUMLDuration::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLObservation.					

Operation **IUMLDuration::SetNewExpr**

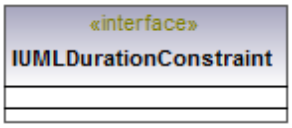
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

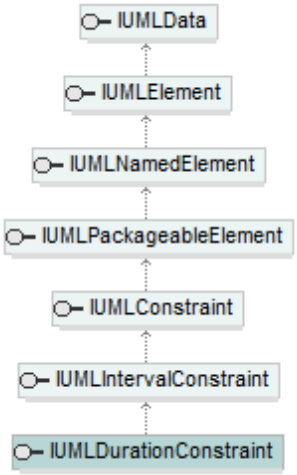
UModelAPI - IUMLDurationConstraint

Interface **IUMLDurationConstraint**

diagram



hierarchy

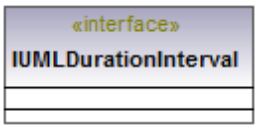


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

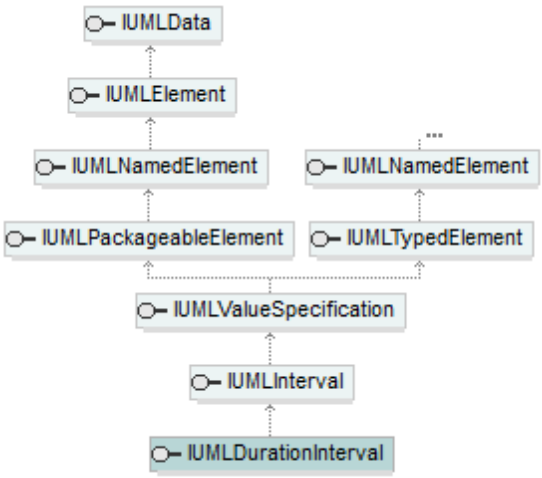
UModelAPI - IUMLDurationInterval

Interface **IUMLDurationInterval**

diagram



hierarchy

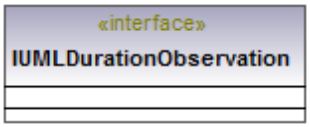


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

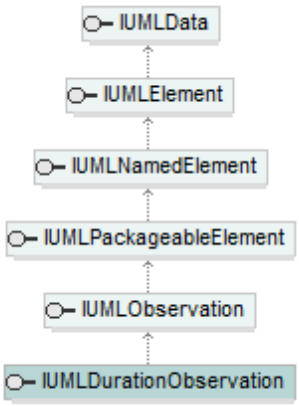
UModelAPI - IUMLDurationObservation

Interface **IUMLDurationObservation**

diagram



 © 2014 Altova GmbH

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLElement

Interface **IUMLElement**

diagram	«interface» **IUMLElement** EraseFromModel():void InsertOwnedCommentAt(in nIdx:int):IUMLComment ApplyStereotype(in ipStereotype:IUMLStereotype):IUMLStereotypeApplication UnapplyStereotype(in ipStereotype:IUMLStereotype):void IsStereotypeApplied(in ipStereotype:IUMLStereotype):bool IsPredefinedStereotypeApplied(in nStereotype:ENUMUMLPredefinedElement):bool GetStereotypeApplicationForStereotype(in ipStereotype:IUMLStereotype):IUMLStereotypeApplication GetStereotypeApplicationForPredefinedStereotype(in nElement:ENUMUMLPredefinedElement):IUMLStereotypeApplication FindPredefinedOwnedElement(in nElement:ENUMUMLPredefinedElement, in bRecursive:bool):IUMLData ApplyPredefinedStereotype(in nStereotype:ENUMUMLPredefinedElement):IUMLStereotypeApplication UnapplyPredefinedStereotype(in nStereotype:ENUMUMLPredefinedElement):void GetOwnedElementsOfKind(in strKind:string, in bRecursive:bool):IUMLDataList «GetAccessor, property» OwnedElements():IUMLDataList «GetAccessor, property» Owner():IUMLElement «GetAccessor, property» OwnedComments():IUMLDataList «GetAccessor, property» AllAppliableStereotypes():IUMLDataList «GetAccessor, property» AppliedStereotypes():IUMLDataList «GetAccessor, property» StereotypeApplications():IUMLDataList «GetAccessor, SetAccessor, property» OwnedDocCommentBody():string «GetAccessor, property» OwnedDocComment():IUMLComment
hierarchy	
typedElements	Interface **IGenerateSequenceDiagramDlg** Interface **IUMLComment** Interface **IUMLConstraint** Operation **DiagramOwner** Operation **InsertAnnotatedElementAt OwningElement** Operation **InsertConstrainedElementAt**

	Interface **IUMLDataAll**	Operation **AppliedElement** **Element** **InsertAnnotatedElementAt** **InsertConstrainedElementAt** **InsertPackagedElementRelationshipAt** **IsElementVisible** **LinkedOwner** **Owner** **OwningElement** **SetElementVisible** **TextLabelElement**
	Interface **IUMLElement**	Operation **Owner**
	Interface **IUMLGuiDiagram**	Operation **LinkedOwner**
	Interface **IUMLGuiLink**	Operation **Element**
	Interface **IUMLGuiNodeLink**	Operation **IsElementVisible** **SetElementVisible**
	Interface **IUMLGuiTextLabel**	Operation **TextLabelElement**
	Interface **IUMLPackage**	Operation **InsertPackagedElementRelationshipAt**
	Interface **IUMLStereotypeApplication**	Operation **AppliedElement**

Operation **IUMLElement::AllAppliableStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStereotype.					

Operation **IUMLElement::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStereotype.					

Operation **IUMLElement::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLElement::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLElement::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			
documen	Use this function to erase the element from the model and all diagrams.					

 © 2014 Altova GmbH

tation	Use IUMLGuiDiagram::EraseFromDiagram to erase from diagram only.

Operation **IUMLElement::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLData**			

Operation **IUMLElement::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLDataList**			
documentation	get all owned elements of the specified kind (*strKind*)					

Operation **IUMLElement::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLElement::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLElement::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLComment**			

Operation **IUMLElement::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**bool**			

Operation **IUMLElement::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			

	return	**return**	**bool**

Operation **IUMLElement::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLComment.					

Operation **IUMLElement::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComment**			

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**			
documentation	A list of elements of type IUMLElement.					

Operation **IUMLElement::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLElement::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLStereotypeApplication.					

Operation **IUMLElement::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**void**			

Operation **IUMLElement::UnapplyStereotype**

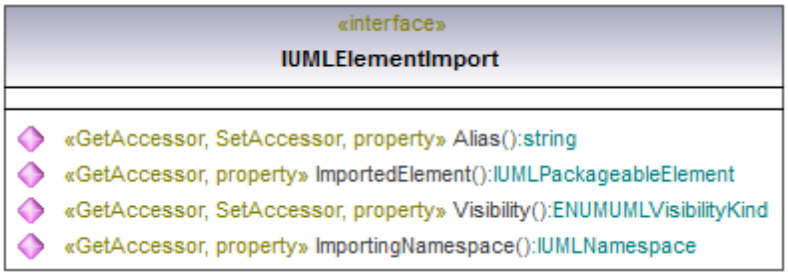
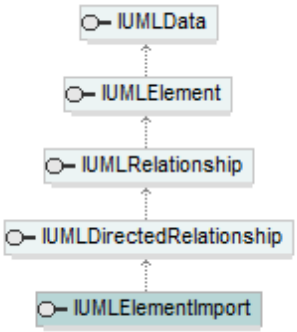
parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**void**			

© 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLElementImport

Interface **IUMLElementImport**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamespace** Operation **InsertElementImportAt** Operation **InsertElementImportAt**

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**			

Operation **IUMLElementImport::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLElementImport::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

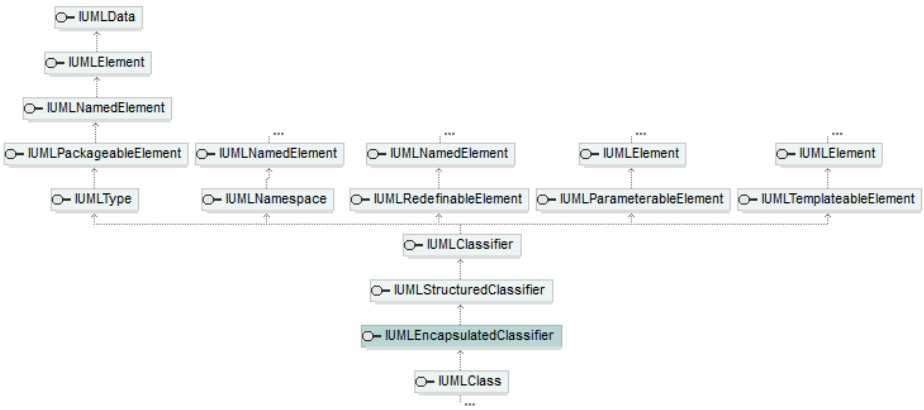
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLEncapsulatedClassifier

Interface **IUMLEncapsulatedClassifier**

diagram

hierarchy



Operation **IUMLEncapsulatedClassifier::OwnedPorts**

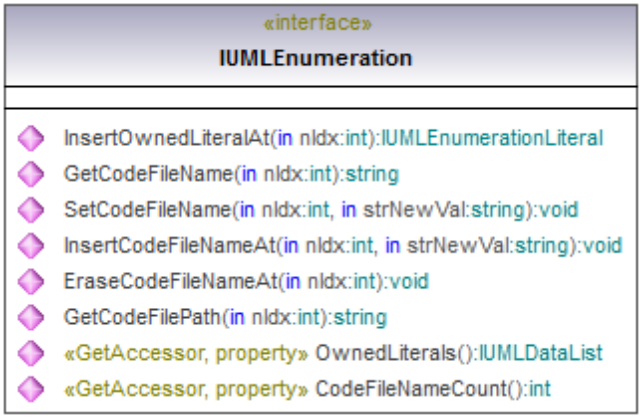
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

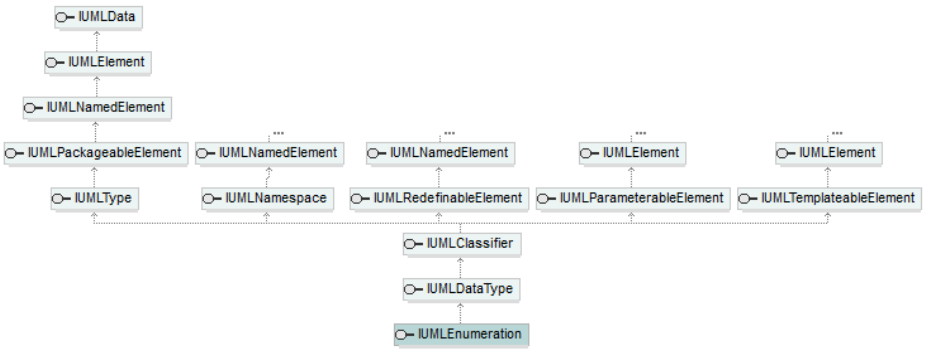
UModelAPI - IUMLEnumeration

Interface **IUMLEnumeration**

diagram



 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLEnumerationLiteral** Operation **Enumeration** Operation **Enumeration**

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**			

Operation **IUMLEnumeration::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLEnumerationLiteral.					

Operation **IUMLEnumeration::SetCodeFileName**

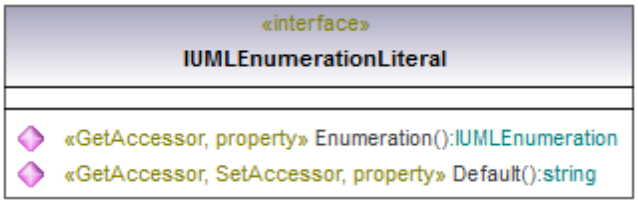
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

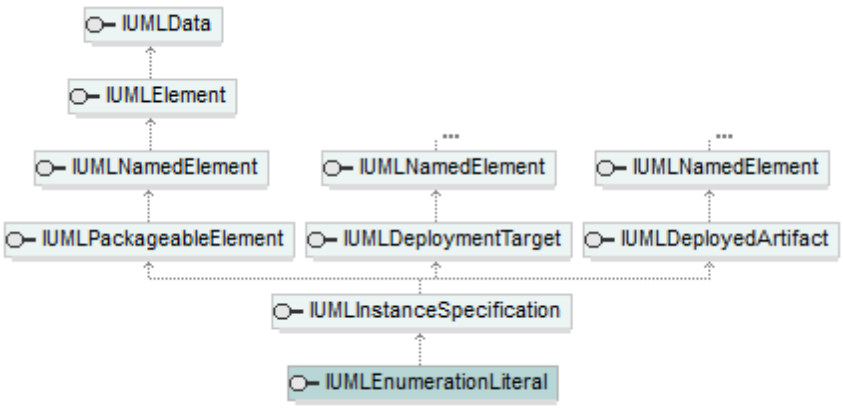
UModelAPI - IUMLEnumerationLiteral

Interface **IUMLEnumerationLiteral**

diagram



hierarchy



typedElements		
	Interface **IUMLDataAll**	Operation **InsertOwnedLiteralAt**
	Interface **IUMLEnumeration**	Operation **InsertOwnedLiteralAt**

Operation **IUMLEnumerationLiteral::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLEnumerationLiteral::Enumeration**

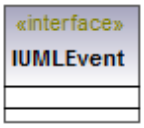
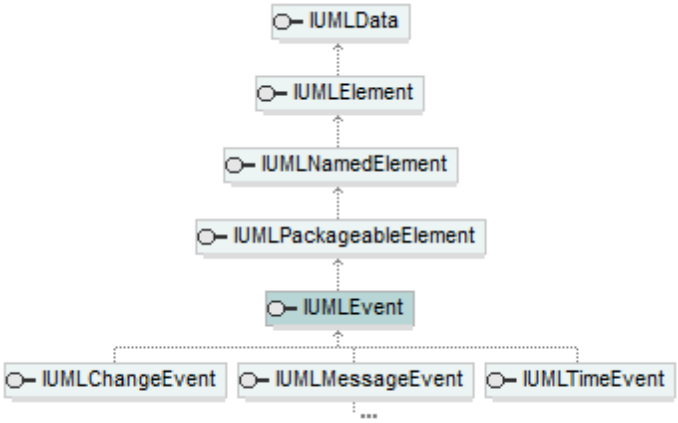
 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEnumeration**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLEvent

Interface **IUMLEvent**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Operation **Event** **OccurringEvent** Interface **IUMLOccurrenceSpecification** Operation **OccurringEvent** Interface **IUMLTrigger** Operation **Event**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLExceptionHandler

Interface **IUMLExceptionHandler**

diagram	«interface» **IUMLExceptionHandler** InsertExceptionTypeAt(in nIdx:int, in ipVal:IUMLClassifier):void EraseExceptionTypeAt(in nIdx:int):void «GetAccessor, property» ProtectedNode():IUMLExecutableNode «GetAccessor, SetAccessor, property» HandlerBody():IUMLExecutableNode «GetAccessor, property» ExceptionTypes():IUMLDataList «GetAccessor, SetAccessor, property» ExceptionInput():IUMLObjectNode
hierarchy	IUMLData IUMLElement IUMLExceptionHandler
typedElements	Interface **IUMLDataAll** Interface **IUMLExecutableNode** Operation **InsertHandlerAt** Operation **InsertHandlerAt**

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**			

Operation **IUMLExceptionHandler::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLExceptionHandler::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

Operation **IUMLExceptionHandler::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default

© 2014 Altova GmbH

r	**nIdx**	**in**	**int**
	ipVal	**in**	**IUMLClassifier**
	return	**return**	**void**

Operation **IUMLExceptionHandler::ProtectedNode**

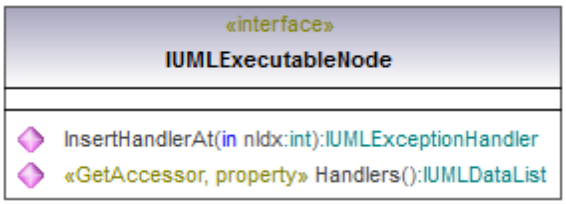
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

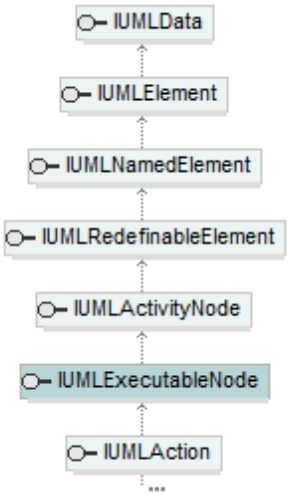
UModelAPI - IUMLExecutableNode

Interface **IUMLExecutableNode**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**
Interface **IUMLExceptionHandler**
Operation **HandlerBody** **ProtectedNode**
Operation **HandlerBody** **ProtectedNode**

Operation **IUMLExecutableNode::Handlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

documentation

A list of elements of type IUMLExceptionHandler.

Operation **IUMLExecutableNode::InsertHandlerAt**

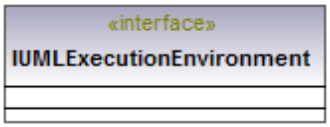
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLException Handler**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

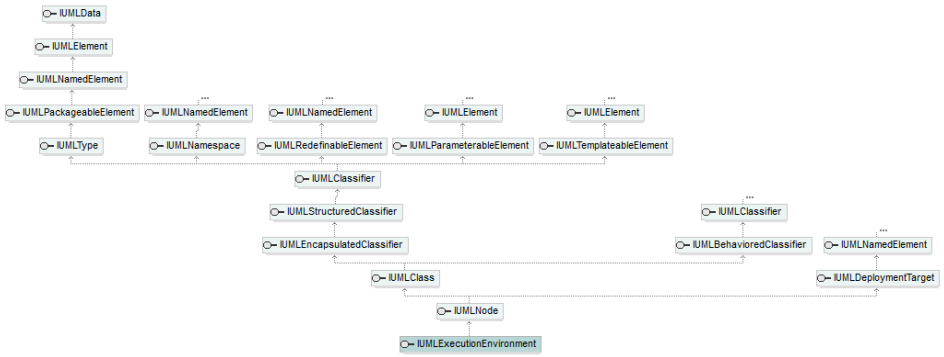
UModelAPI - IUMLExecutionEnvironment

Interface **IUMLExecutionEnvironment**

diagram



hierarchy

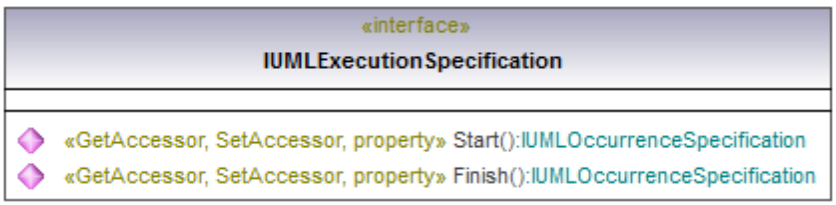


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

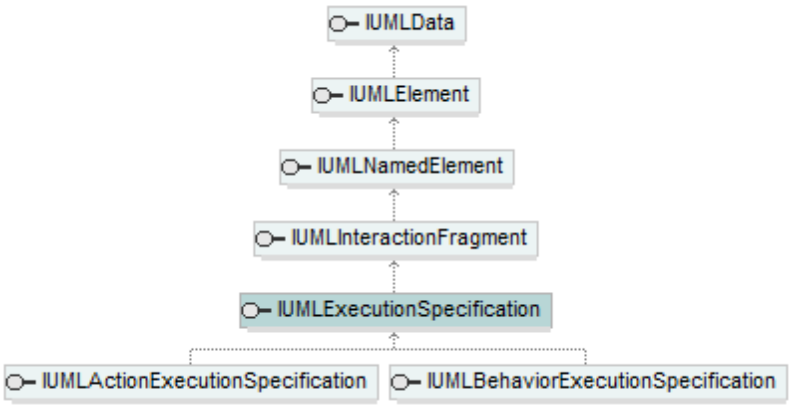
UModelAPI - IUMLExecutionSpecification

Interface **IUMLExecutionSpecification**

diagram



 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLOccurrenceSpecification** Operation **ExecutionSpecificationFinish** **ExecutionSpecificationStart** Operation **ExecutionSpecificationFinish** **ExecutionSpecificationStart**

Operation **IUMLExecutionSpecification::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

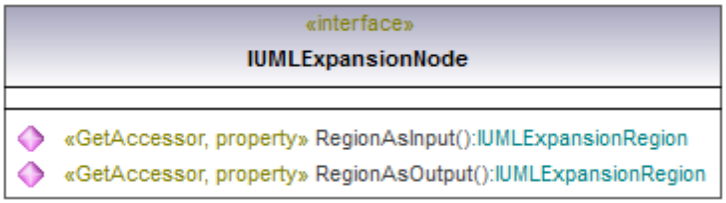
Operation **IUMLExecutionSpecification::Start**

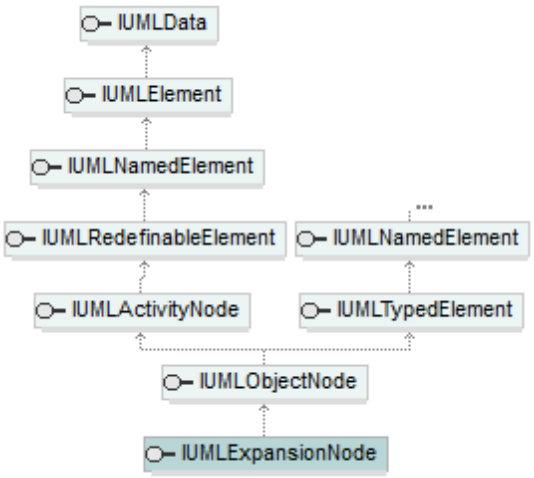
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLExpansionNode

Interface **IUMLExpansionNode**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Operation **InsertInputElementAt** **InsertOutputElementAt** Interface **IUMLExpansionRegion** Operation **InsertInputElementAt** **InsertOutputElementAt**

Operation **IUMLExpansionNode::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

Operation **IUMLExpansionNode::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

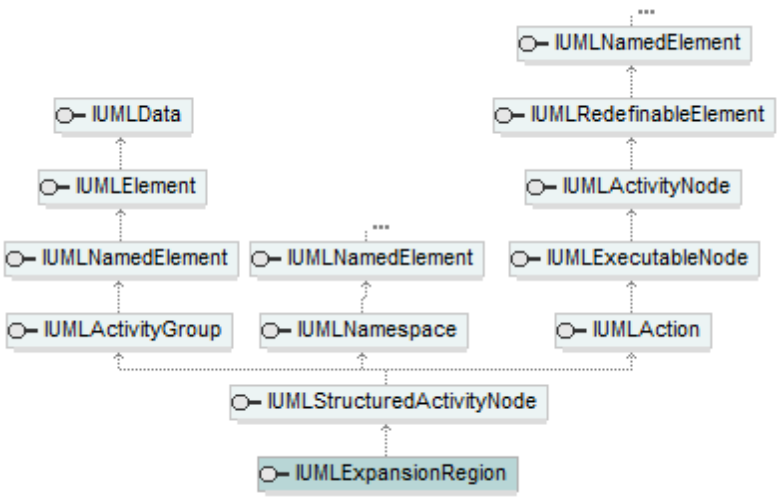
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLExpansionRegion

Interface **IUMLExpansionRegion**

diagram	«interface» **IUMLExpansionRegion** InsertInputElementAt(in nIdx:int, in ipNode:IUMLExpansionNode):void EraseInputElementAt(in nIdx:int):void InsertOutputElementAt(in nIdx:int, in ipNode:IUMLExpansionNode):void EraseOutputElementAt(in nIdx:int):void «GetAccessor, SetAccessor, property» Mode():ENUMUMLExpansionKind «GetAccessor, property» InputElements():IUMLDataList «GetAccessor, property» OutputElements():IUMLDataList

 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLExpansionNode** Operation **RegionAsInput** **RegionAsOutput** Operation **RegionAsInput** **RegionAsOutput**

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**			
documentation	A list of elements of type IUMLExpansionNode.					

Operation **IUMLExpansionRegion::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLExpansionRegion::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansio**			

	return	**return**	**nNode** **void**

Operation **IUMLExpansionRegion::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLExpansionKind**			

Operation **IUMLExpansionRegion::OutputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExpansionNode.					

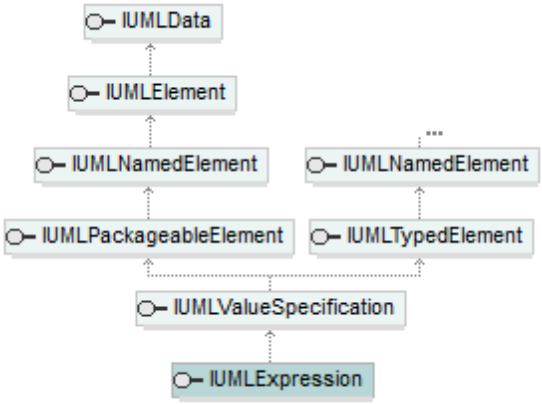
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLExpression

Interface **IUMLExpression**

diagram

hierarchy



typedElements		
	Interface **IUMLDataAll**	Operation **Expression**
	Interface **IUMLValueSpecification**	Operation **Expression**

Operation **IUMLExpression::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

 © 2014 Altova GmbH

documentation	A list of elements of type IUMLValueSpecification.

Operation **IUMLExpression::Symbol**

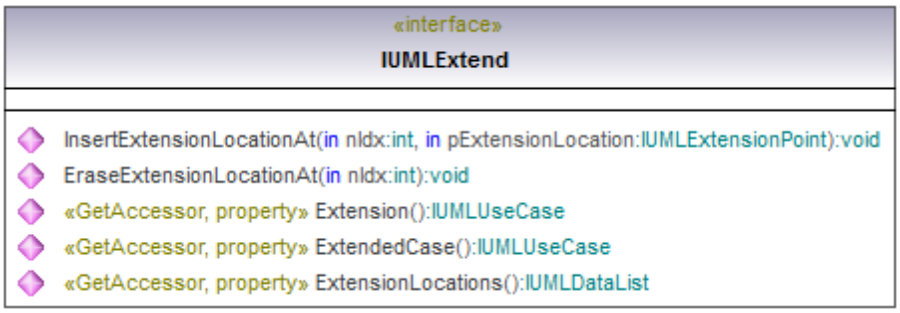
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

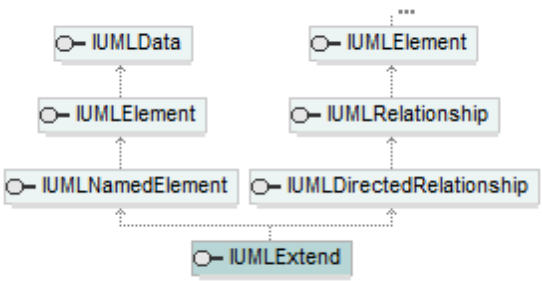
UModelAPI - IUMLExtend

Interface **IUMLExtend**

diagram



hierarchy



typedElements	Interface **IUMLDataAll**	Operation **InsertExtendAt**
	Interface **IUMLUseCase**	Operation **InsertExtendAt**

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**			

Operation **IUMLExtend::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLExtend::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExtensionPoint.					

Operation **IUMLExtend::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pExtensionLocation	**in**	**IUMLExtensionPoint**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLExtensionPoint

Interface **IUMLExtensionPoint**

diagram	«interface» IUMLExtensionPoint «GetAccessor, property» UseCase():IUMLUseCase
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLRedefinableElement IUMLExtensionPoint
typedElements	Interface **IUMLDataAll** Interface **IUMLExtend** Interface **IUMLUseCase** Operation **InsertExtensionLocationAt** Operation **InsertExtensionPointAt** Operation **InsertExtensionLocationAt** Operation **InsertExtensionPointAt**

Operation **IUMLExtensionPoint::UseCase**

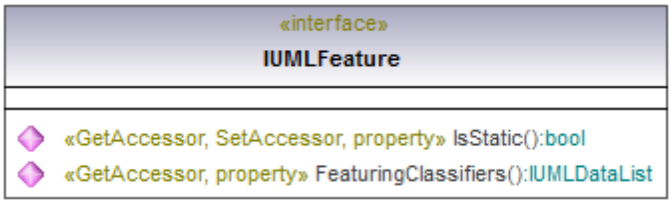
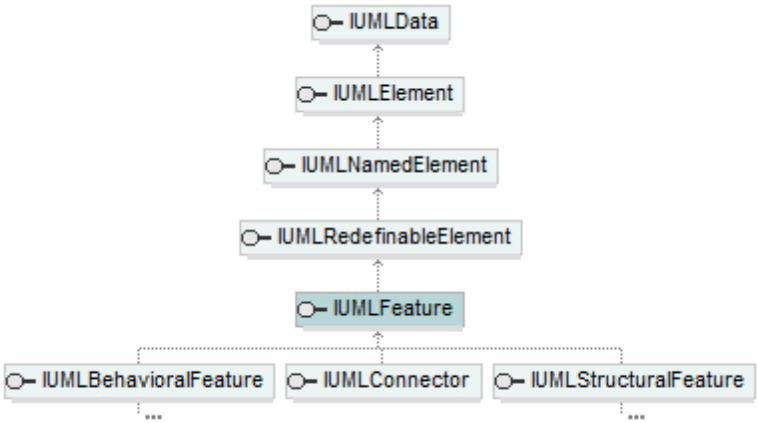
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

© 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLFeature

Interface **IUMLFeature**

diagram	
hierarchy	

Operation **IUMLFeature::FeaturingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

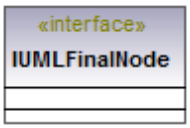
Operation **IUMLFeature::IsStatic**

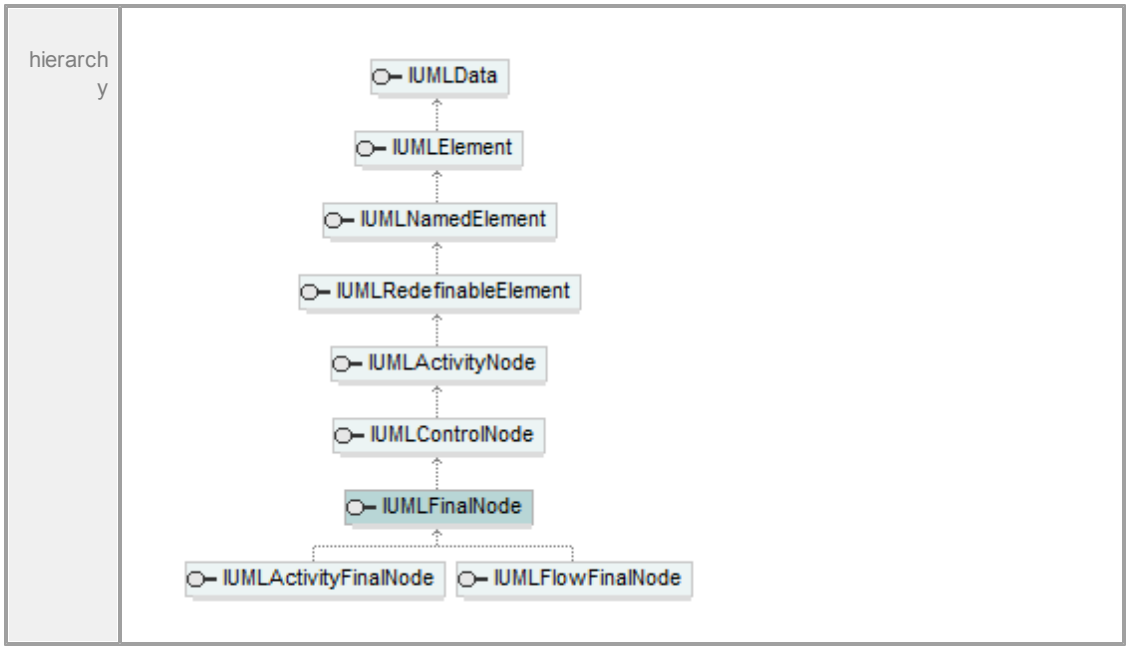
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLFinalNode

Interface **IUMLFinalNode**

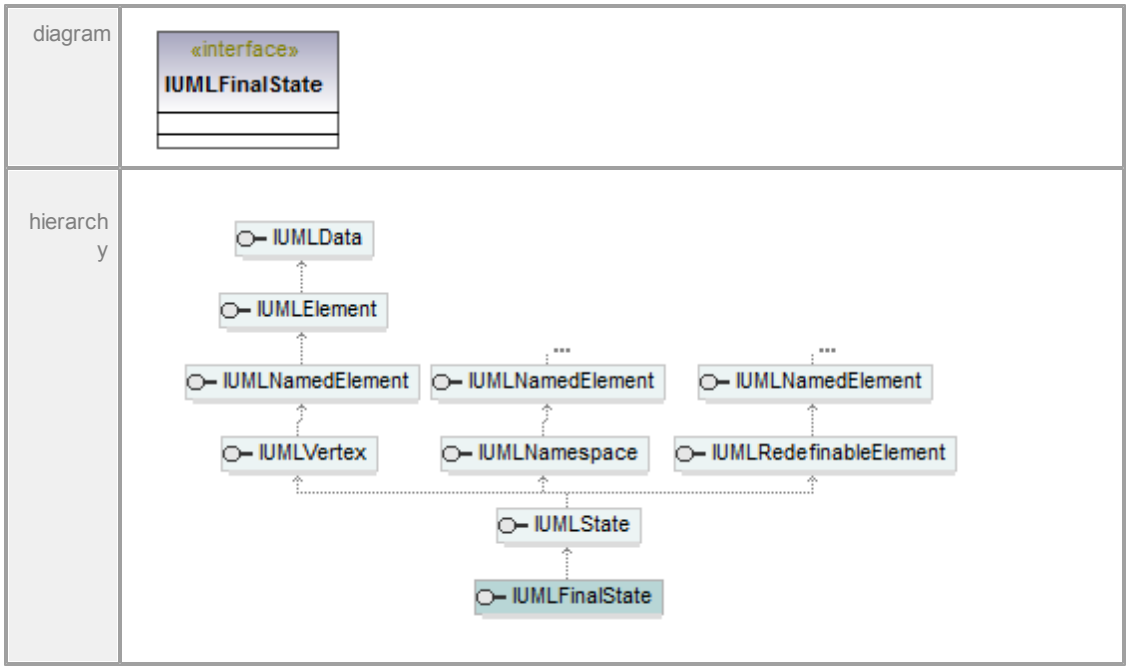
diagram	



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLFinalState

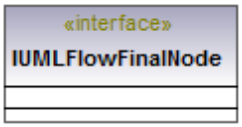
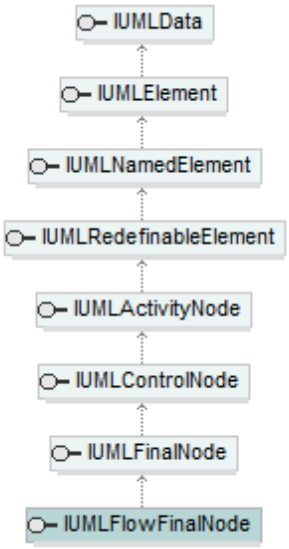
Interface **IUMLFinalState**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLFlowFinalNode

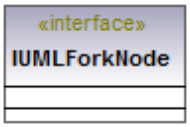
Interface **IUMLFlowFinalNode**

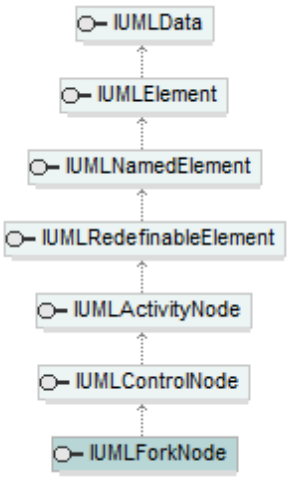
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLForkNode

Interface **IUMLForkNode**

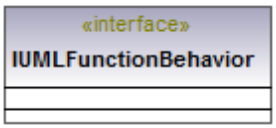
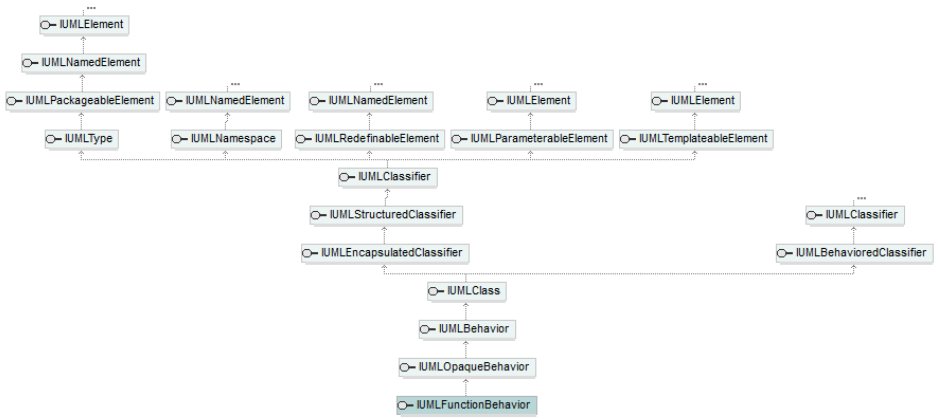
diagram	

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLFunctionBehavior

Interface **IUMLFunctionBehavior**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGate

Interface **IUMLGate**

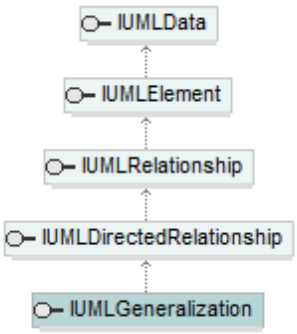
diagram	«interface» IUMLGate
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLMessageEnd IUMLGate
typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLInteractionUse** Operation **InsertActualGateAt** **InsertFormalGateAt** Operation **InsertFormalGateAt** Operation **InsertActualGateAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGeneralization

Interface **IUMLGeneralization**

diagram	«interface» IUMLGeneralization «GetAccessor, property» General():IUMLClassifier «GetAccessor, property» Specific():IUMLClassifier «GetAccessor, SetAccessor, property» IsSubstitutable():bool

hierarchy	
typedElements	Interface **IUMLClassifier** Operation **InsertGeneralizationAt** Interface **IUMLDataAll** Operation **InsertGeneralizationAt**

Operation **IUMLGeneralization::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLGeneralization::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

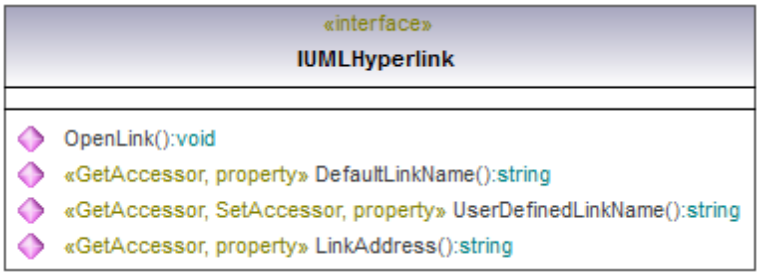
Operation **IUMLGeneralization::Specific**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

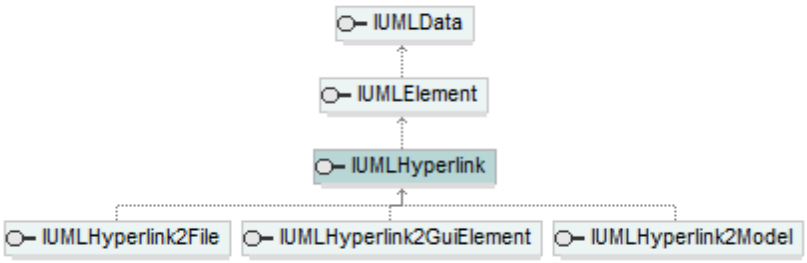
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLHyperlink

Interface **IUMLHyperlink**

diagram	

 © 2014 Altova GmbH

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**

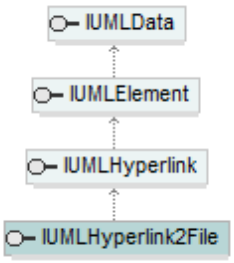
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLHyperlink2File

Interface **IUMLHyperlink2File**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamedElement** Operation **InsertOwnedHyperlink2FileAt** Operation **InsertOwnedHyperlink2FileAt**

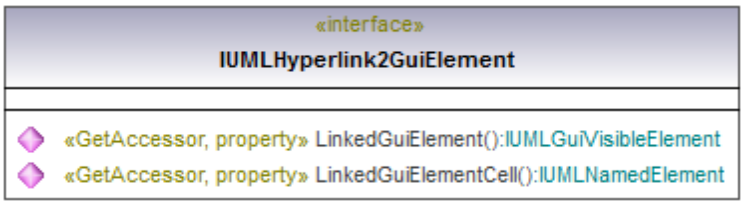
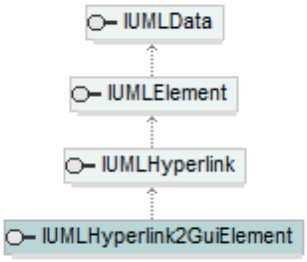
Operation **IUMLHyperlink2File::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLHyperlink2GuiElement

Interface **IUMLHyperlink2GuiElement**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamedElement** Operation **InsertOwnedHyperlink2GuiElementAt** Operation **InsertOwnedHyperlink2GuiElementAt**

Operation **IUMLHyperlink2GuiElement::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiVisibl**			

	eElement

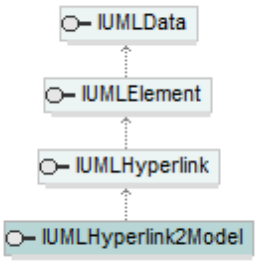
Operation **IUMLHyperlink2GuiElement::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamedElement**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLHyperlink2Model

Interface **IUMLHyperlink2Model**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamedElement** Operation **InsertOwnedHyperlink2ModelAt** Operation **InsertOwnedHyperlink2ModelAt**

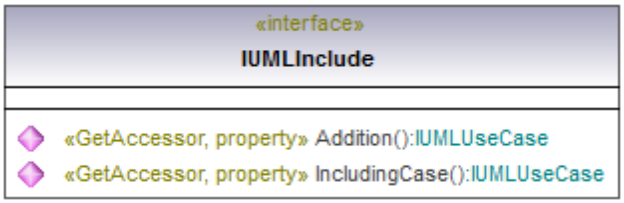
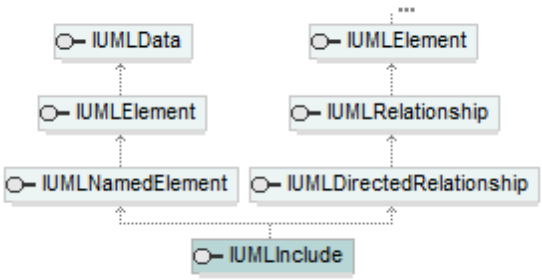
Operation **IUMLHyperlink2Model::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLInclude

Interface **IUMLInclude**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLUseCase** Operation **InsertIncludeAt** Operation **InsertIncludeAt**

Operation **IUMLInclude::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLInclude::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

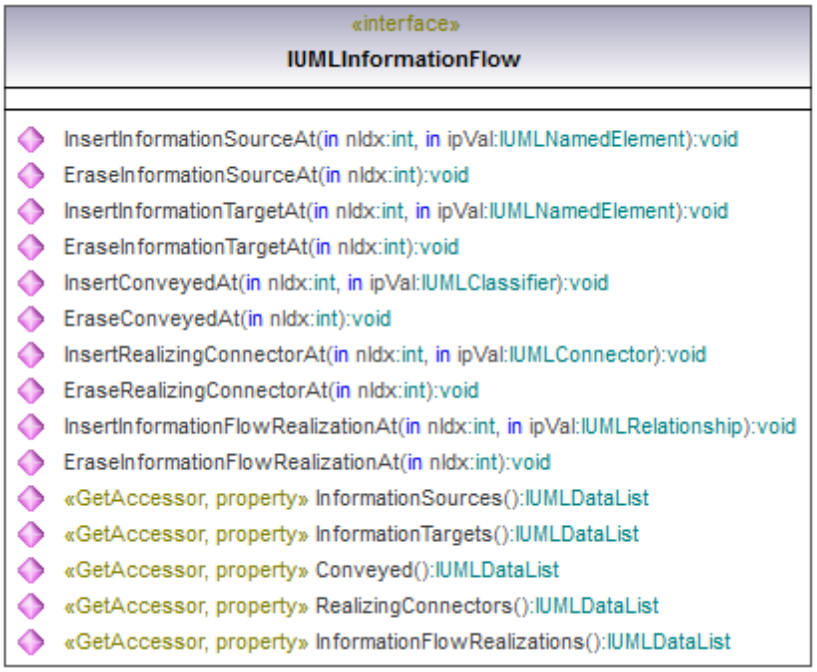
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

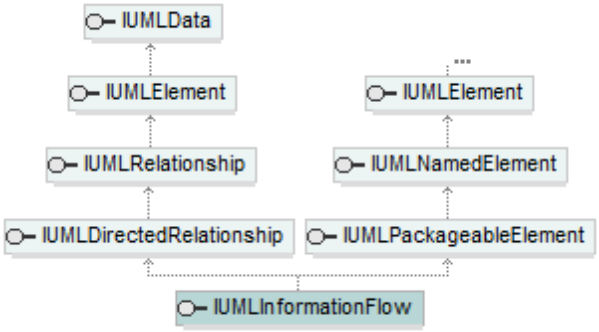
UModelAPI - IUMLInformationFlow

Interface **IUMLInformationFlow**

diagram



hierarchy



Operation **IUMLInformationFlow::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::EraseInformationFlowRealizationAt**

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**			

Operation **IUMLInformationFlow::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLInformationFlow::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLRelationship**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

© 2014 Altova GmbH

Operation **IUMLInformationFlow::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnector**			
	return	**return**	**void**			

Operation **IUMLInformationFlow::RealizingConnectors**

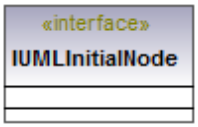
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

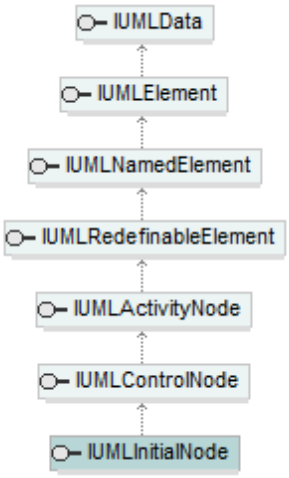
UModelAPI - IUMLInitialNode

Interface **IUMLInitialNode**

diagram



hierarchy

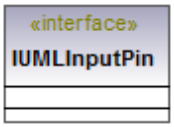


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

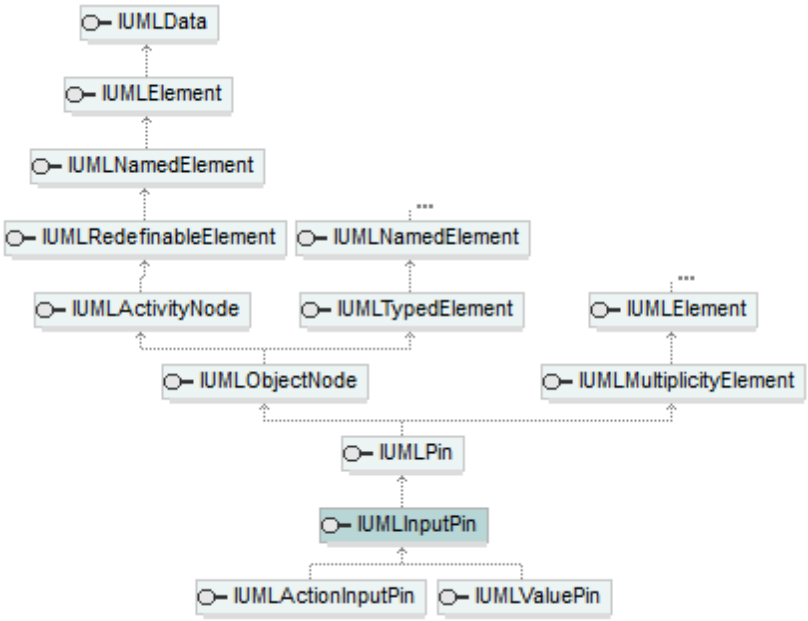
UModelAPI - IUMLInputPin

Interface **IUMLInputPin**

diagram



hierarchy



typedElements

Interface **IUMLCallOperationAction**	Operation **CallTarget** **SetNewCallTarget**
Interface **IUMLDataAll**	Operation **CallTarget** **InsertArgumentAt** **InsertInputValueAt** **SetNewCallTarget** **SetNewSignalTarget** **SignalTarget**
Interface **IUMLInvocationAction**	Operation **InsertArgumentAt**
Interface **IUMLOpaqueAction**	Operation **InsertInputValueAt**
Interface **IUMLSendSignalAction**	Operation **SetNewSignalTarget** **SignalTarget**

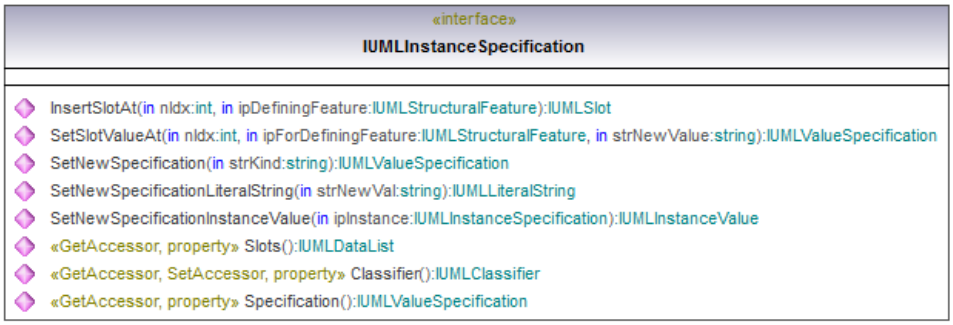
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

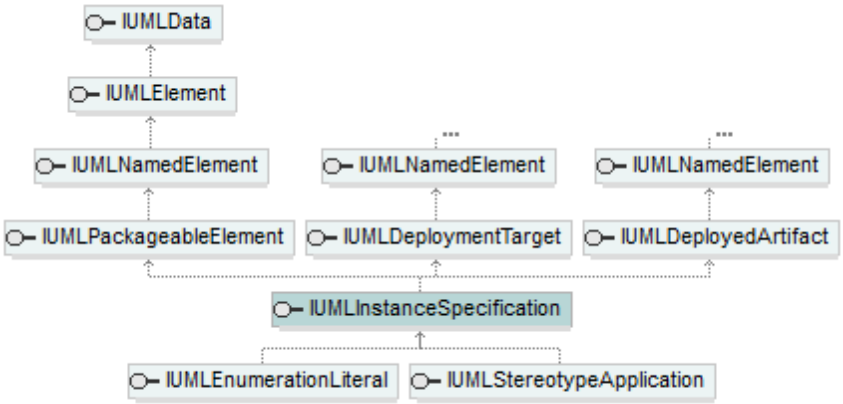
UModelAPI - IUMLInstanceSpecification

Interface **IUMLInstanceSpecification**

diagram



hierarchy



typedElements

Interface **IUMLConstraint**	Operation **SetNewSpecificationInstanceValue**
Interface **IUMLDataAll**	Operation **Instance** **OwningInstance** **OwningInstanceSpec** **SetNewDefaultValueInstanceValue** **SetNewSpecificationInstanceValue**
Interface **IUMLInstanceSpecification**	Operation **SetNewSpecificationInstanceValue**
Interface **IUMLInstanceValue**	Operation **Instance**
Interface **IUMLParameter**	Operation **SetNewDefaultValueInstanceValue**
Interface **IUMLProperty**	Operation **SetNewDefaultValueInstanceValue**
Interface **IUMLSlot**	Operation **OwningInstance**
Interface **IUMLValueSpecification**	Operation **OwningInstanceSpec**

Operation **IUMLInstanceSpecification::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLInstanceSpecification::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature**			
	return	**return**	**IUMLSlot**			

Operation **IUMLInstanceSpecification::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInstanceSpecification::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLInstanceSpecification::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLInstanceSpecification::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInstanceSpecification::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLSlot.					

Operation **IUMLInstanceSpecification::Specification**

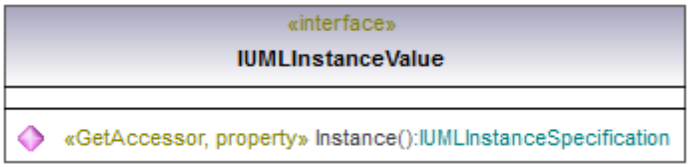
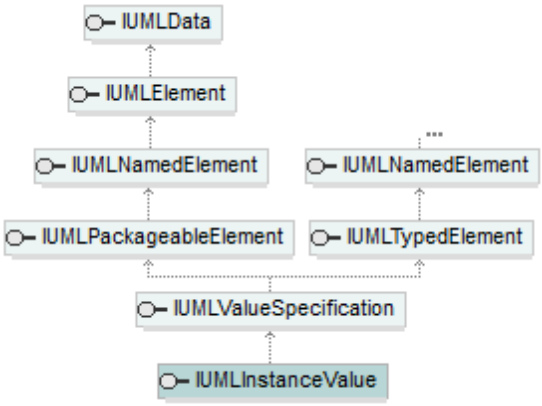
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - IUMLInstanceValue

Interface **IUMLInstanceValue**

diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** Interface **IUMLDataAll** Interface **IUMLInstanceSpecification** Interface **IUMLParameter** Interface **IUMLProperty** Operation **SetNewSpecificationInstanceValue** Operation **SetNewDefaultValueInstanceValue** **SetNewSpecificationInstanceValue** Operation **SetNewSpecificationInstanceValue** Operation **SetNewDefaultValueInstanceValue** Operation **SetNewDefaultValueInstanceValue**

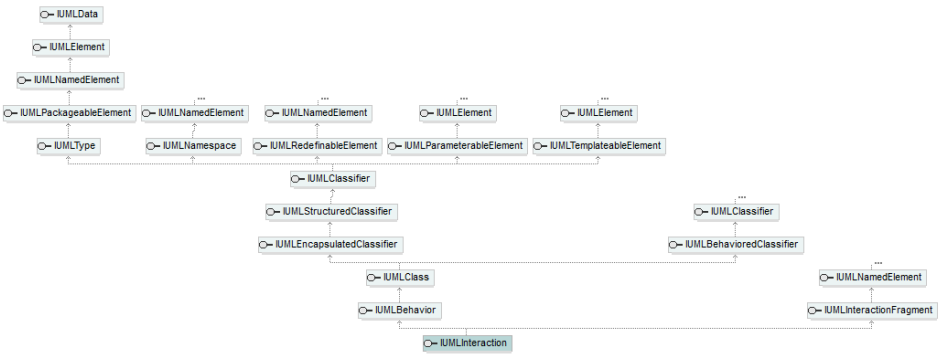
Operation **IUMLInstanceValue::Instance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLInteraction

Interface **IUMLInteraction**

diagram	«interface» **IUMLInteraction** InsertLifelineAt(in nIdx:int):IUMLLifeline InsertFragmentAt(in nIdx:int, in strKind:string):IUMLInteractionFragment InsertMessageAt(in nIdx:int):IUMLMessage InsertFormalGateAt(in nIdx:int):IUMLGate «GetAccessor, property» Lifelines():IUMLDataList «GetAccessor, property» Fragments():IUMLDataList «GetAccessor, property» Messages():IUMLDataList «GetAccessor, property» FormalGates():IUMLDataList
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteractionUse** Operation **RefersTo** Operation **RefersTo**

Operation **IUMLInteraction::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGate.					

Operation **IUMLInteraction::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInteractionFragment.					

Operation **IUMLInteraction::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLInteraction::InsertFragmentAt**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInteractionFragment**			

Operation **IUMLInteraction::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLLifeline**			

Operation **IUMLInteraction::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLMessage**			

Operation **IUMLInteraction::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLLifeline.					

Operation **IUMLInteraction::Messages**

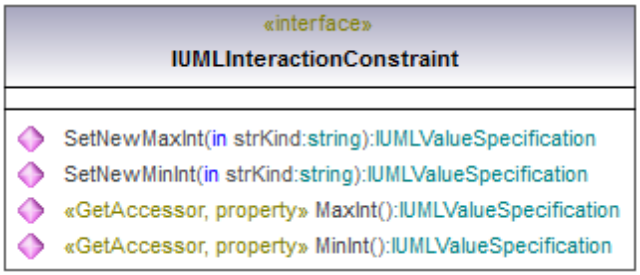
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLMessage.					

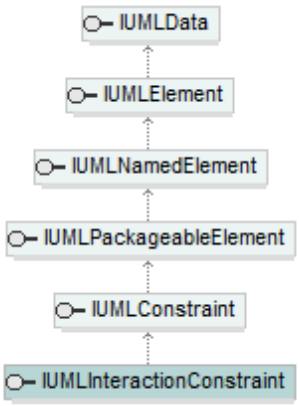
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLInteractionConstraint

Interface **IUMLInteractionConstraint**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteractionOperand** Operation **OperandGuard** **SetNewOperandGuard** Operation **OperandGuard** **SetNewOperandGuard**

Operation **IUMLInteractionConstraint::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInteractionConstraint::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInteractionConstraint::SetNewMaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInteractionConstraint::SetNewMinInt**

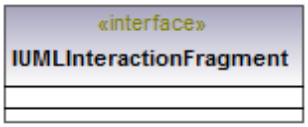
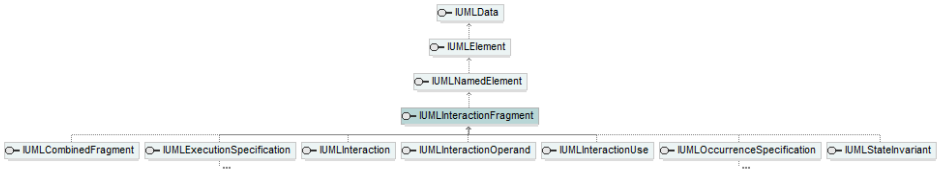
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLInteractionFragment

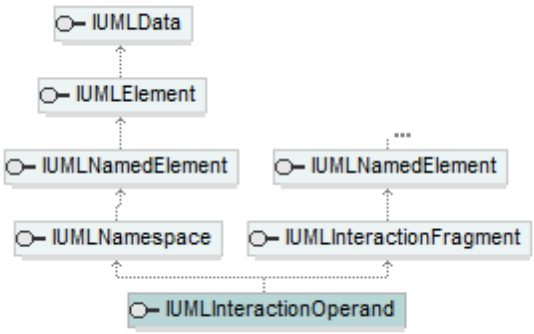
Interface **IUMLInteractionFragment**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLLifeline** Operation **InsertCoveredByAt** **InsertFragmentAt** Operation **InsertFragmentAt** Operation **InsertCoveredByAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLInteractionOperand

Interface **IUMLInteractionOperand**

diagram	
hierarchy	
typedElements	Interface **IUMLCombinedFragment** Interface **IUMLDataAll** Operation **InsertOperandAt** Operation **InsertOperandAt**

Operation **IUMLInteractionOperand::OperandGuard**

parameters	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLInteractionConstraint**

Operation **IUMLInteractionOperand::SetNewOperandGuard**

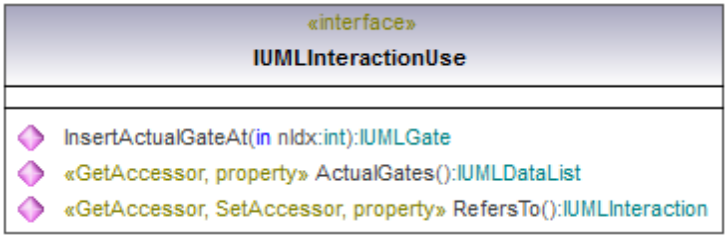
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

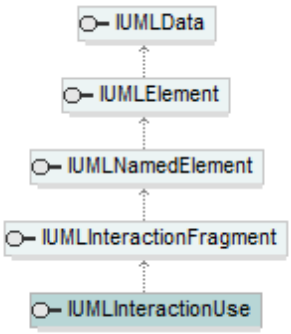
UModelAPI - IUMLInteractionUse

Interface **IUMLInteractionUse**

diagram



hierarchy



Operation **IUMLInteractionUse::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLGate.					

Operation **IUMLInteractionUse::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLInteractionUse::RefersTo**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

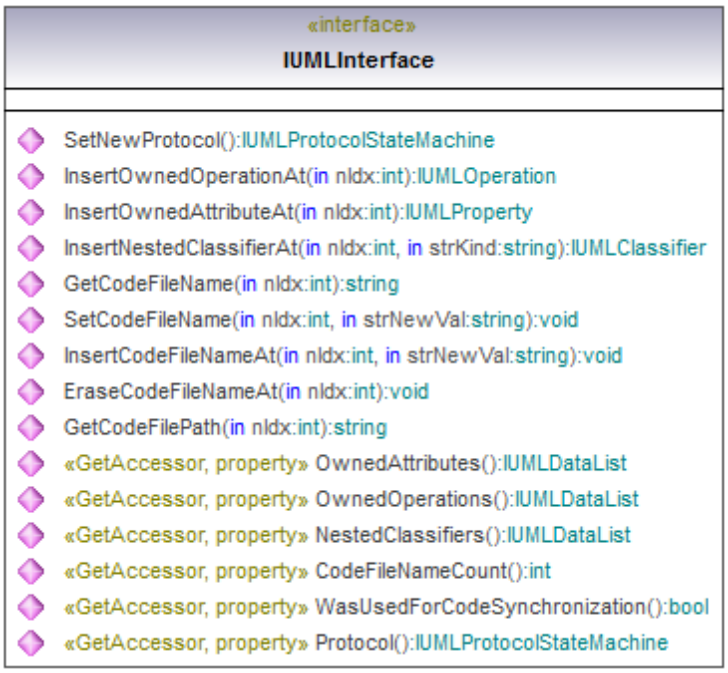
r	**return**	**return**	**IUMLInteraction**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

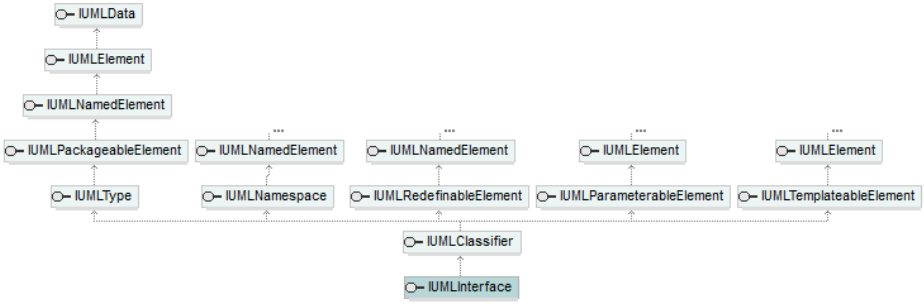
UModelAPI - IUMLInterface

Interface **IUMLInterface**

diagram



hierarchy



typedElements

Interface **IUMLBehavioredClassifier**	Operation **InsertInterfaceRealizationAt**
Interface **IUMLClassifier**	Operation **NestingInterface**
Interface **IUMLDataAll**	Operation **Contract**
	InsertInterfaceRealizationAt
	Interface **NestingInterface**
Interface **IUMLInterfaceRealization**	Operation **Contract**
Interface **IUMLOperation**	Operation **Interface**
Interface **IUMLProperty**	Operation **Interface**
Interface	Operation **Interface**

	IUMLProtocolStateMachine

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**			
	return	**return**	**void**			

Operation **IUMLInterface::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	IUMLClassifier			

Operation **IUMLInterface::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLProperty			

Operation **IUMLInterface::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	IUMLOperation			

Operation **IUMLInterface::NestedClassifiers**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

Operation **IUMLInterface::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLInterface::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLOperation.					

Operation **IUMLInterface::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

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**			

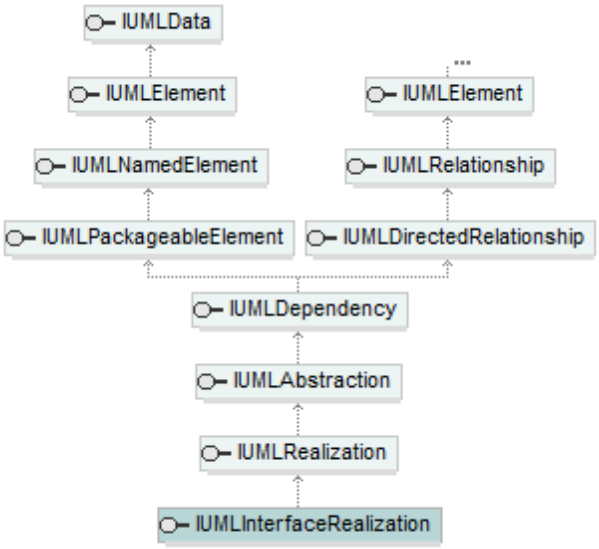
Operation **IUMLInterface::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLInterfaceRealization

Interface **IUMLInterfaceRealization**

diagram	
hierarchy	
typedElements	Interface **IUMLBehavioredClassifier** Operation **InsertInterfaceRealizationAt** Interface **IUMLDataAll** Operation **InsertInterfaceRealizationAt**

Operation **IUMLInterfaceRealization::Contract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLInterfaceRealization::ImplementingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavioredClassifier**			

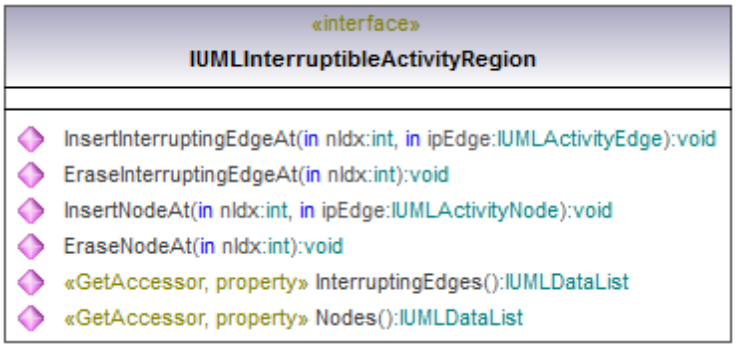
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

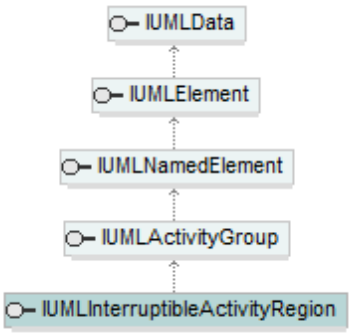
UModelAPI - IUMLInterruptibleActivityRegion

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**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLInterruptibleActivityRegion::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityEdge.					

Operation **IUMLInterruptibleActivityRegion::Nodes**

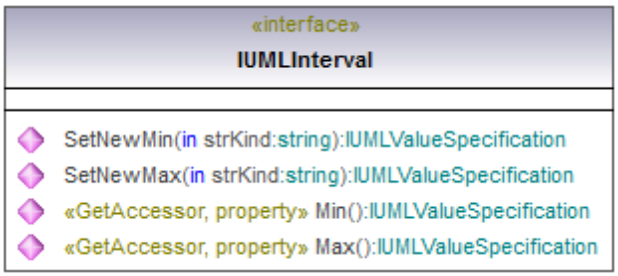
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLActivityNode.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

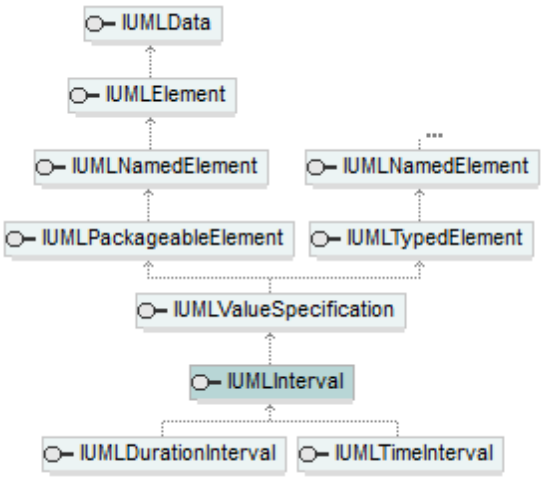
UModelAPI - IUMLInterval

Interface **IUMLInterval**

diagram



hierarchy



Operation **IUMLInterval::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpe**			

	cification

Operation **IUMLInterval::Min**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInterval::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLInterval::SetNewMin**

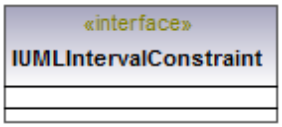
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

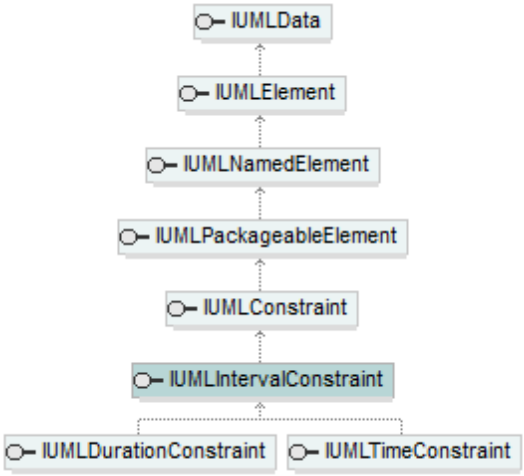
UModelAPI - IUMLIntervalConstraint

Interface **IUMLIntervalConstraint**

diagram



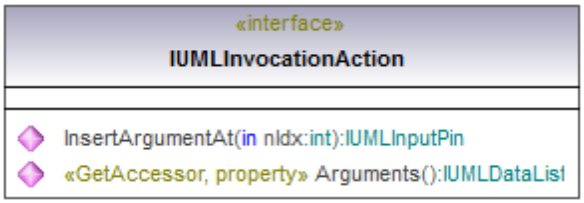
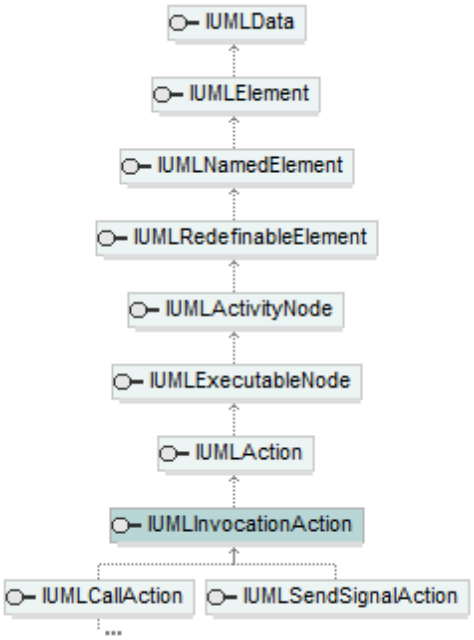
hierarchy



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLInvocationAction

Interface **IUMLInvocationAction**

diagram	
hierarchy	

Operation **IUMLInvocationAction::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInputPin.					

Operation **IUMLInvocationAction::InsertArgumentAt**

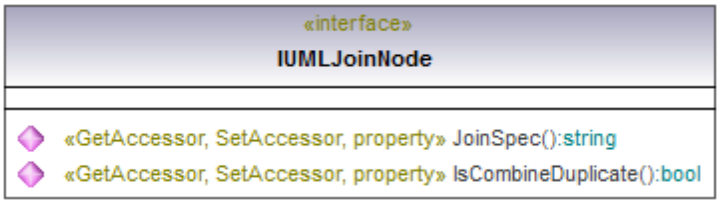
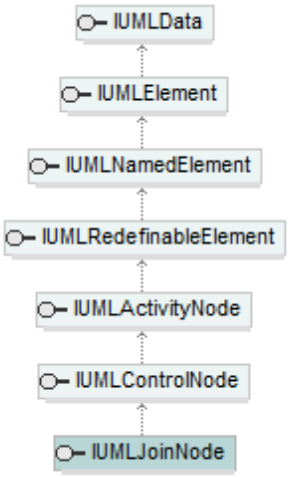
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

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** 09/04/12 14:02:42

UModelAPI - IUMLLifeline

Interface **IUMLLifeline**

diagram	«interface» **IUMLLifeline** SetNewSelector(in strKind:string):IUMLValueSpecification InsertCoveredByAt(in nIdx:int, in ipVal:IUMLInteractionFragment):void EraseCoveredByAt(in nIdx:int):void «GetAccessor, property» Selector():IUMLValueSpecification «GetAccessor, SetAccessor, property» Represents():IUMLConnectableElement
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLLifeline
typedElements	Interface **IUMLDataAll** — Operation **Covered** **GetSourceLifeline** **GetTargetLifeline** **InsertLifelineAt** Interface **IUMLInteraction** — Operation **InsertLifelineAt** Interface **IUMLMessage** — Operation **GetSourceLifeline** **GetTargetLifeline** Interface **IUMLOccurrenceSpecification** — Operation **Covered** Interface **IUMLStateInvariant** — Operation **Covered**

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**			
	return	**return**	**void**			

Operation **IUMLLifeline::Represents**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement**			

© 2014 Altova GmbH

Operation **IUMLLifeline::Selector**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLLifeline::SetNewSelector**

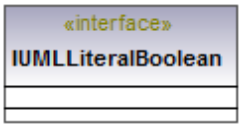
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

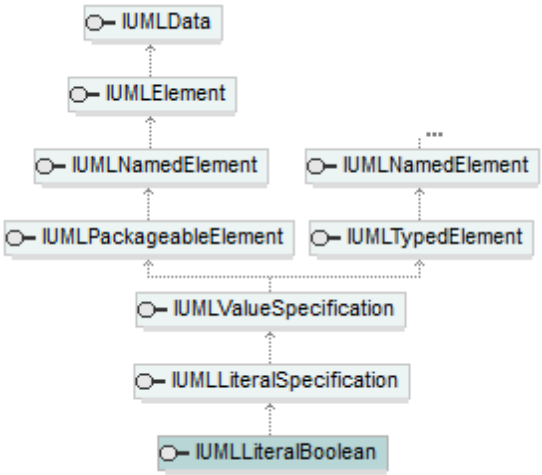
UModelAPI - IUMLLiteralBoolean

Interface **IUMLLiteralBoolean**

diagram



hierarchy

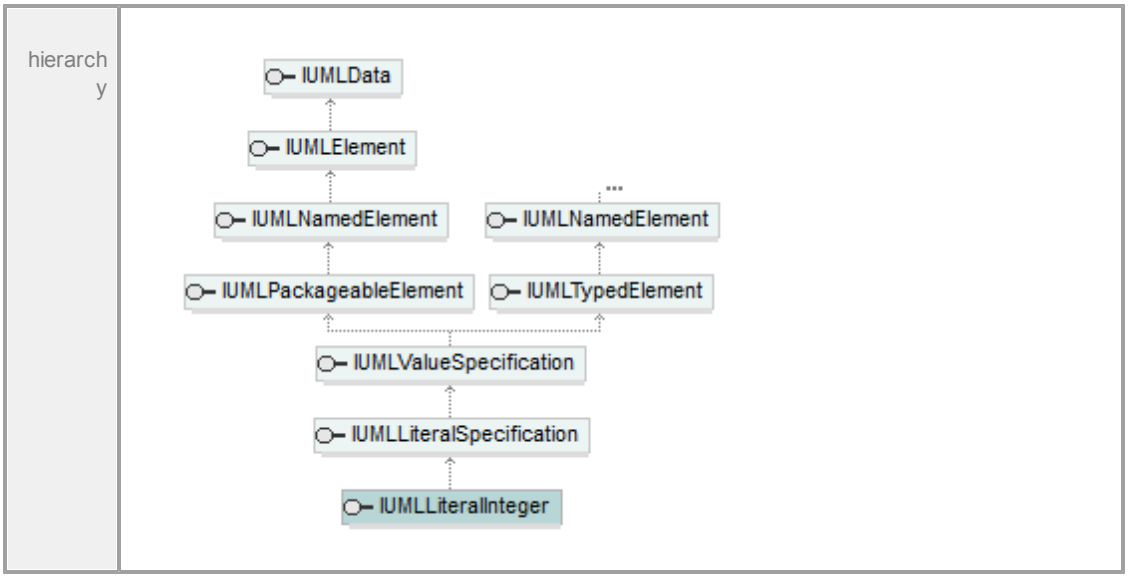


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLLiteralInteger

Interface **IUMLLiteralInteger**

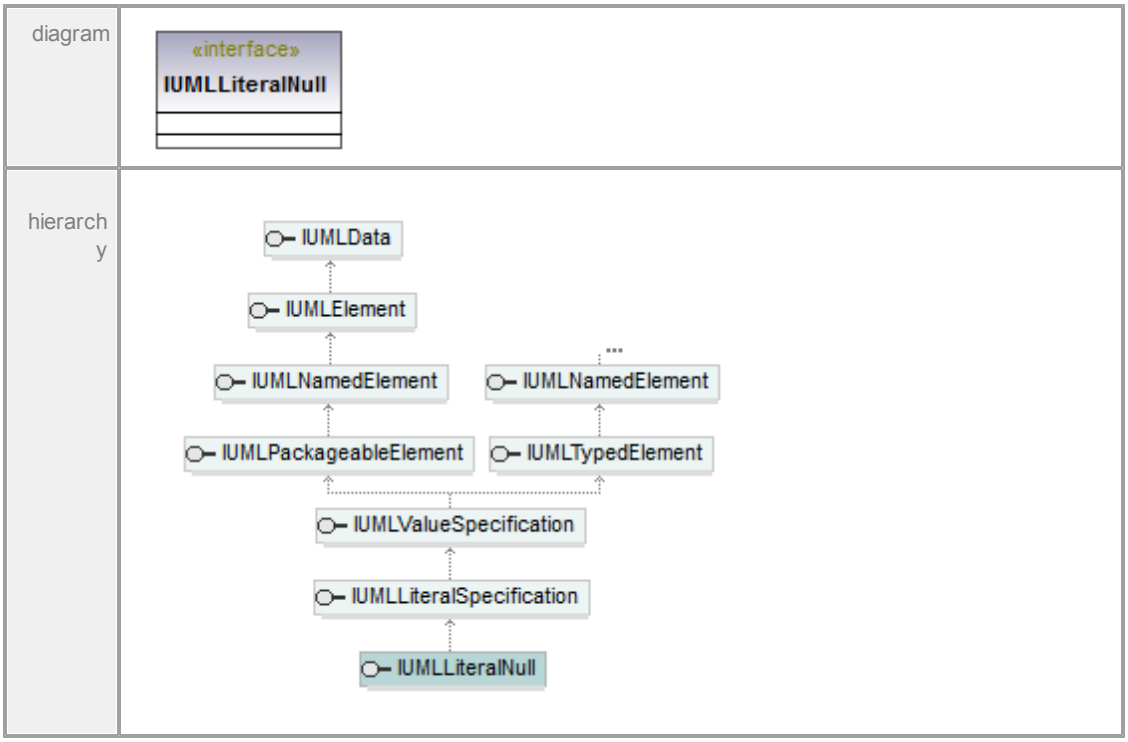
diagram



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLLiteralNull

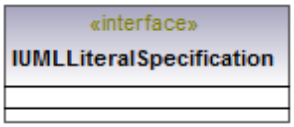
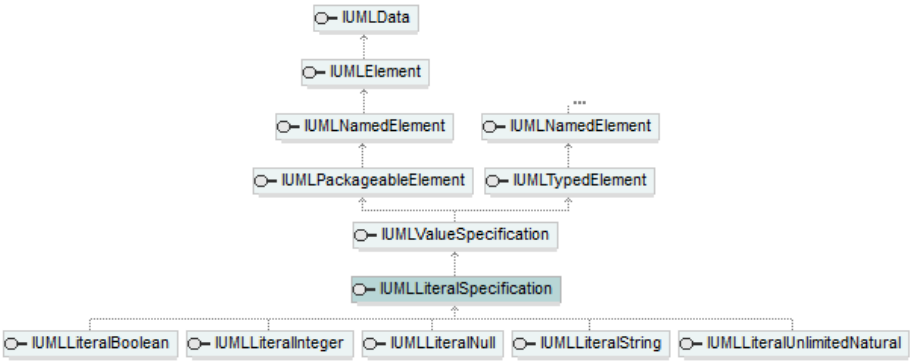
Interface **IUMLLiteralNull**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLLiteralSpecification

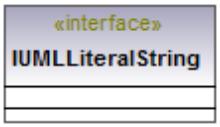
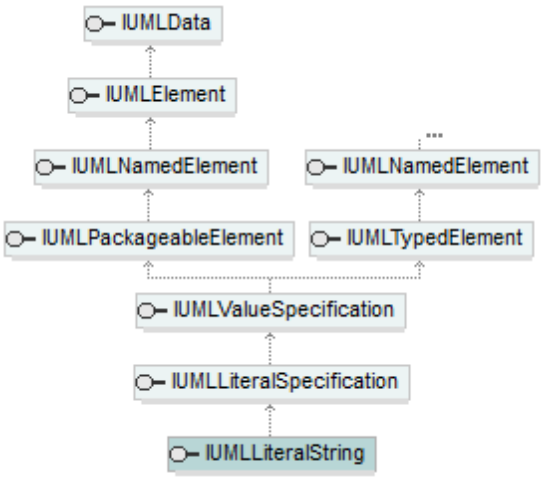
Interface **IUMLLiteralSpecification**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLLiteralString

Interface **IUMLLiteralString**

diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** Operation **SetNewSpecificationLiteralString**

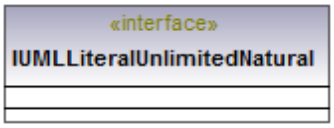
	Interface **IUMLDataAll**	Operation **SetNewDefaultValueLiteralString** **SetNewSpecificationLiteralString**
	Interface **IUMLInstanceSpecification**	Operation **SetNewSpecificationLiteralString**
	Interface **IUMLParameter**	Operation **SetNewDefaultValueLiteralString**
	Interface **IUMLProperty**	Operation **SetNewDefaultValueLiteralString**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

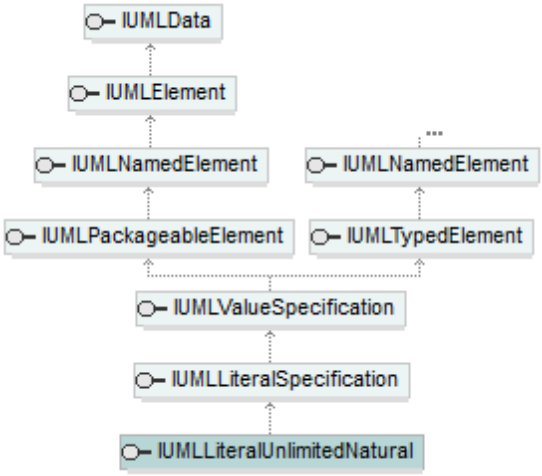
UModelAPI - IUMLLiteralUnlimitedNatural

Interface **IUMLLiteralUnlimitedNatural**

diagram



hierarchy

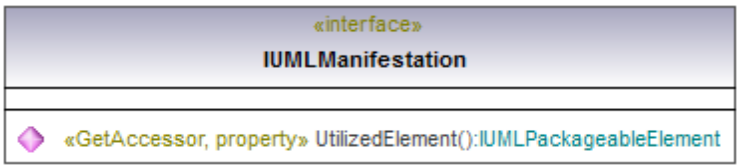


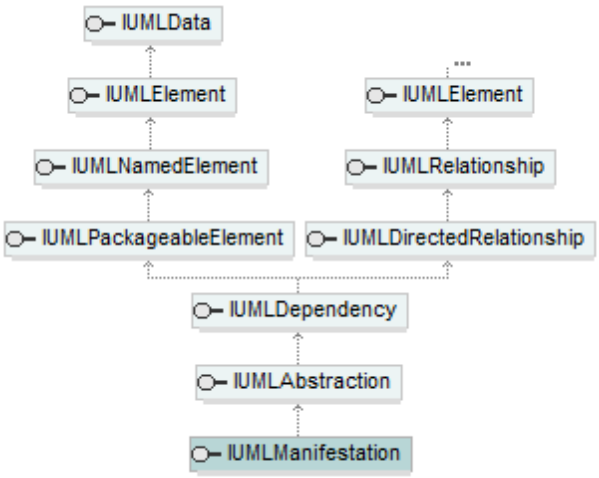
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLManifestation

Interface **IUMLManifestation**

diagram



hierarchy	
typedElements	Interface **IUMLArtifact** Operation **InsertManifestationAt** Interface **IUMLDataAll** Operation **InsertManifestationAt**

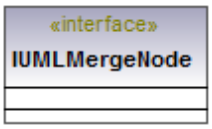
Operation **IUMLManifestation::UtilizedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement**			

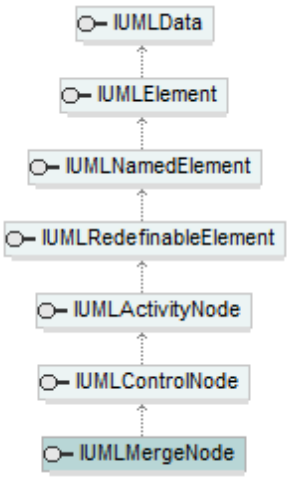
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLMergeNode

Interface **IUMLMergeNode**

diagram	

hierarchy

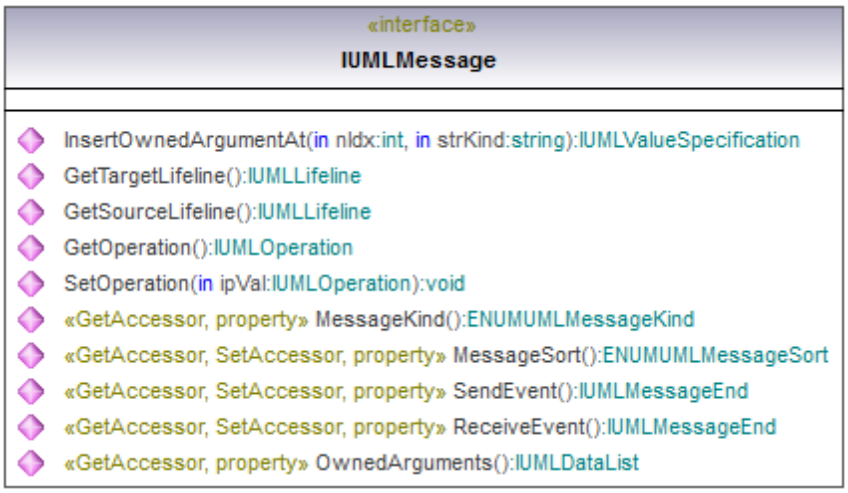


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

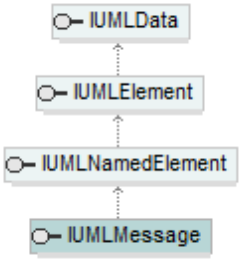
UModelAPI - IUMLMessage

Interface **IUMLMessage**

diagram



hierarchy



 © 2014 Altova GmbH

typedElements	Interface **IUMLDataAll** Interface **IUMLInteraction** Interface **IUMLMessageEnd**	Operation **InsertMessageAt Message** Operation **InsertMessageAt** Operation **Message**

Operation **IUMLMessage::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLMessage::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLMessage::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLMessage::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLMessage::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageKind**			

Operation **IUMLMessage::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLMessageSort**			

Operation **IUMLMessage::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

Operation **IUMLMessage::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd**			

Operation **IUMLMessage::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessage**			

	End

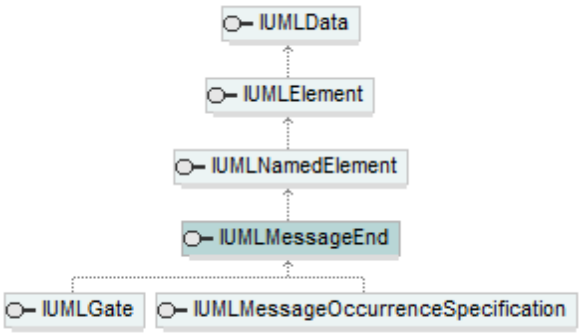
Operation **IUMLMessage::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLOperation**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLMessageEnd

Interface **IUMLMessageEnd**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Operation **ReceiveEvent** **SendEvent** Interface **IUMLMessage** Operation **ReceiveEvent** **SendEvent**

Operation **IUMLMessageEnd::Message**

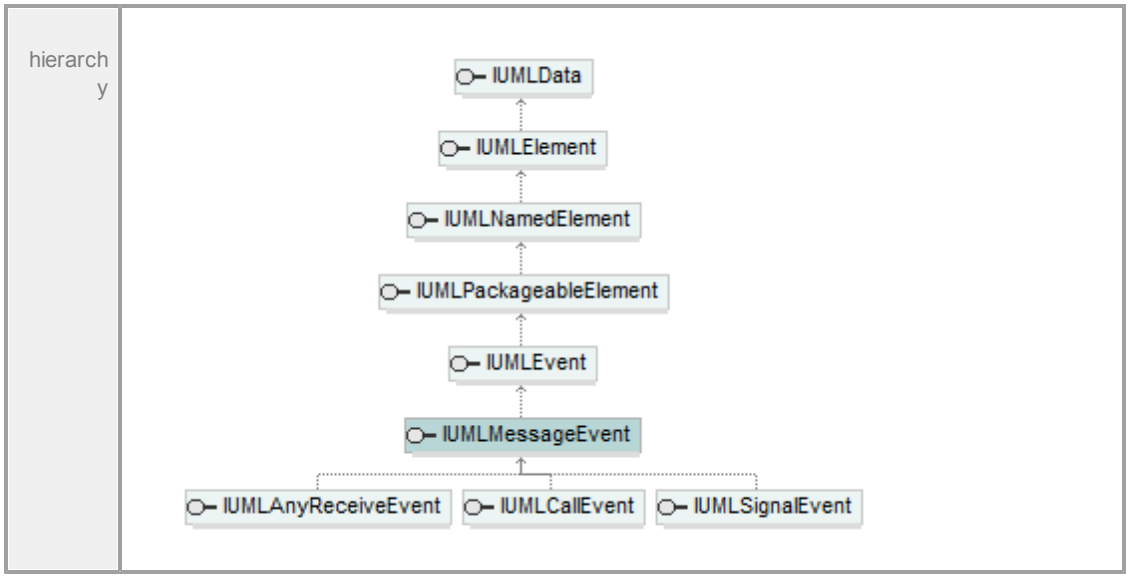
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessage**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLMessageEvent

Interface **IUMLMessageEvent**

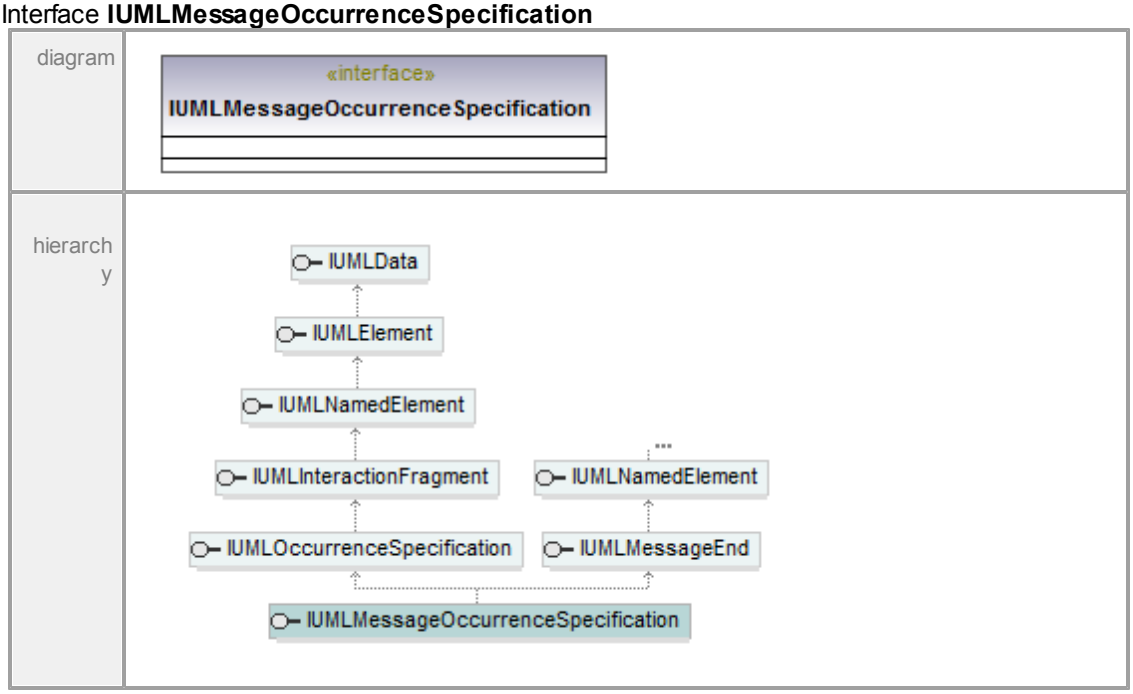
diagram	



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLMessageOccurrenceSpecification

Interface **IUMLMessageOccurrenceSpecification**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLModel

Interface **IUMLModel**

diagram	«interface» **IUMLModel** «GetAccessor, SetAccessor, property» Viewpoint():string
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLNamedElement IUMLPackageableElement IUMLNamespace IUMLPackage IUMLModel

Operation **IUMLModel::Viewpoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

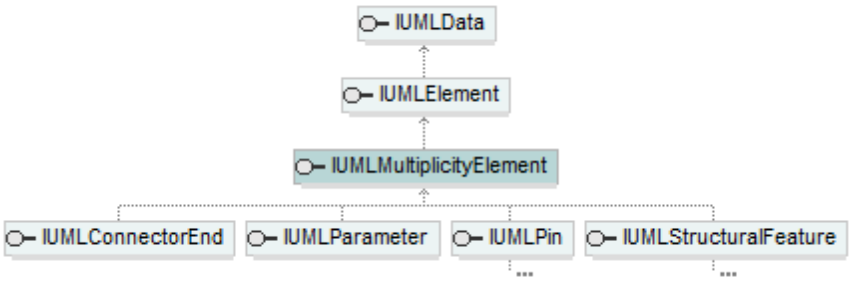
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLMultiplicityElement

Interface **IUMLMultiplicityElement**

diagram	«interface» **IUMLMultiplicityElement** InsertLowerUpperValueAt(in nIdx:int, in strLower:string, in strUpper:string):void GetMultiplicity(in bWithBrackets:bool):string SetMultiplicity(in strNewVal:string):void «GetAccessor, SetAccessor, property» IsOrdered():bool «GetAccessor, SetAccessor, property» IsUnique():bool «GetAccessor, property» UpperValues():IUMLDataList «GetAccessor, property» LowerValues():IUMLDataList

 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Operation **OwningLower** **OwningUpper** Interface **IUMLValueSpecification** Operation **OwningLower** **OwningUpper**

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**			
documentation	A list of elements of type IUMLValueSpecification.					

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**			
documentation	A list of elements of type IUMLValueSpecification.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLNamedElement

Interface **IUMLNamedElement**

diagram	«interface» **IUMLNamedElement** SetName(in strStartWith:string):string InsertOwnedHyperlink2GuiElementAt(in nIdx:int, in ipLinkedGuiElement:IUMLGuiVisibleElement, in ipLinkedGuiElementCell:IUMLNamedElement):IUMLHyperlink2GuiElement InsertOwnedHyperlink2FileAt(in nIdx:int, in strFilePathOrUrl:string):IUMLHyperlink2File InsertOwnedHyperlink2ModelAt(in nIdx:int, in ipLinkedData:IUMLData):IUMLHyperlink2Model «GetAccessor, SetAccessor, property» Name():string «GetAccessor, property» QualifiedName():string «GetAccessor, property» Namespace():IUMLNamespace «GetAccessor, SetAccessor, property» Visibility():ENUMUMLVisibilityKind «GetAccessor, property» SupplierDependencies():IUMLDataList «GetAccessor, property» ClientDependencies():IUMLDataList «GetAccessor, property» OwnedHyperlinks():IUMLDataList
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLActivityGroup, IUMLCollaborationUse, IUMLDeployedArtifact, IUMLDeploymentTarget, IUMLExtend, IUMLInclude, IUMLInteractionFragment, IUMLLifeline, IUMLMessage, IUMLMessageEnd, IUMLNamespace, IUMLPackageableElement, IUMLRedefinableElement, IUMLTrigger, IUMLTypedElement, IUMLVertex
typedElements	Interface **IUMLCommentTextHyperlink** Interface **IUMLDataAll** Interface **IUMLGuiTextHyperlink** Interface **IUMLHyperlink2GuiElement** Interface **IUMLInformationFlow** Interface **IUMLNamedElement** Interface **IUMLNamespace** Interface **IUMLTimeObservation**	Operation **SetHyperlinkGuiElementAddress** Operation **FindOwnedMemberWithQualifiedName** **InsertInformationSourceAt** **InsertInformationTargetAt** **InsertOwnedHyperlink2GuiElementAt** **LinkedGuiElementCell** **SetHyperlinkGuiElementAddress** **TimeObservationEvent** Operation **SetHyperlinkGuiElementAddress** Operation **LinkedGuiElementCell** Operation **InsertInformationSourceAt** **InsertInformationTargetAt** Operation **InsertOwnedHyperlink2GuiElementAt** Operation **FindOwnedMemberWithQualifiedName** Operation **TimeObservationEvent**

Operation **IUMLNamedElement::ClientDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documen	A list of elements of type IUMLDependency.					

 © 2014 Altova GmbH

tation	

Operation **IUMLNamedElement::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**IUMLHyperlink2File**			

Operation **IUMLNamedElement::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**IUMLHyperlink2GuiElement**			

Operation **IUMLNamedElement::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**IUMLHyperlink2Model**			

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**			

Operation **IUMLNamedElement::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLHyperlink.					

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 will find and set a unique name (starting with 'strStartWith') that the element is distinguishable in its parent namespace.		

Operation **IUMLNamedElement::SupplierDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLDependency.					

Operation **IUMLNamedElement::Visibility**

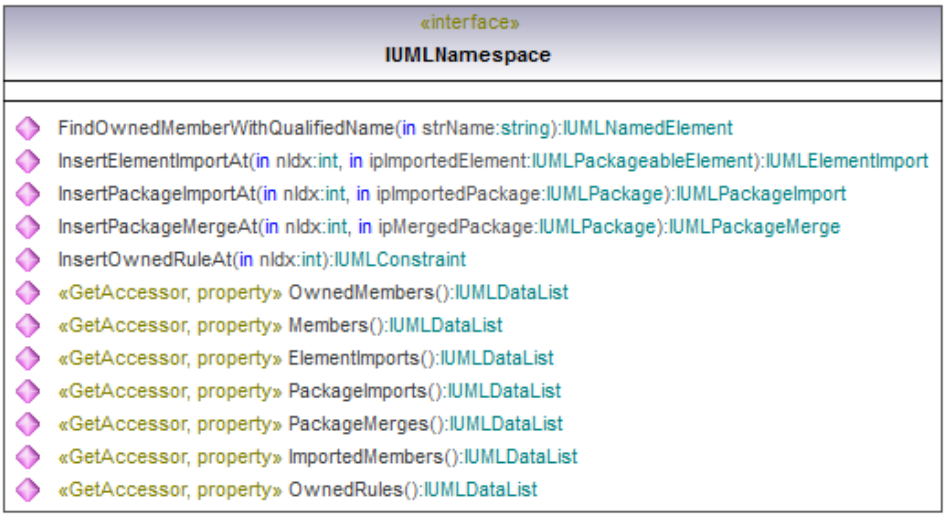
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

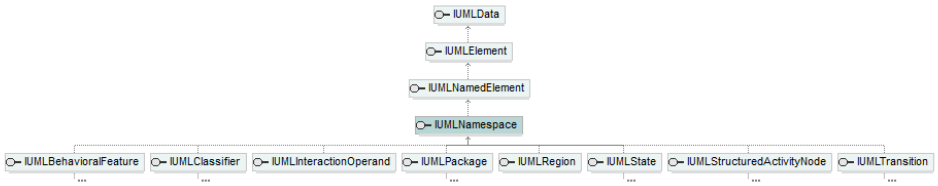
UModelAPI - IUMLNamespace

Interface **IUMLNamespace**

diagram



hierarchy



typedElements

Interface **IUMLConstraint**
Interface **IUMLDataAll**
Interface **IUMLElementImport**
Interface **IUMLNamedElement**
Operation **Context**
Operation **ContextImportingNamespaceNamespace**
Operation **ImportingNamespace**
Operation **Namespace**

	Interface **IUMLPackageImport**	Operation **ImportingNamespace**

Operation **IUMLNamespace::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElementImport.					

Operation **IUMLNamespace::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	strName	**in**	**string**			
	return	**return**	**IUMLNamedElement**			

Operation **IUMLNamespace::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageableElement.					

Operation **IUMLNamespace::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedElement	**in**	**IUMLPackageableElement**			
	return	**return**	**IUMLElementImport**			

Operation **IUMLNamespace::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLNamespace::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageImport**			

Operation **IUMLNamespace::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipMergedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageMerge**			

Operation **IUMLNamespace::Members**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLNamespace::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNamedElement.					

Operation **IUMLNamespace::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConstraint.					

Operation **IUMLNamespace::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageImport.					

Operation **IUMLNamespace::PackageMerges**

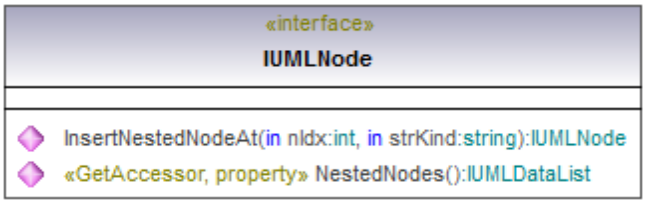
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageMerge.					

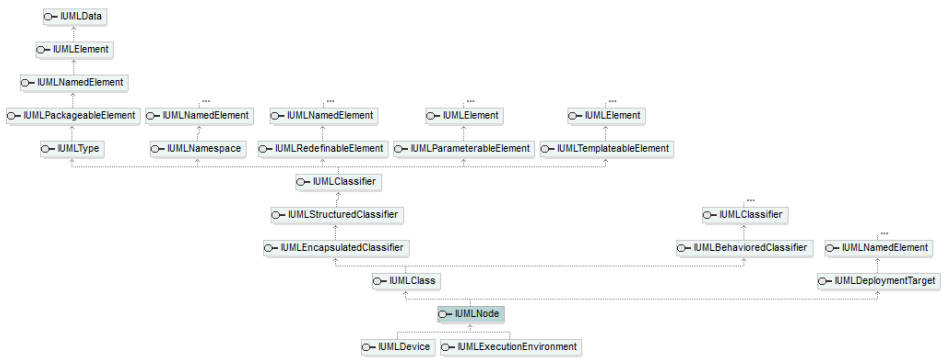
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLNode

Interface **IUMLNode**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNode** Operation **InsertNestedNodeAt** Operation **InsertNestedNodeAt**

Operation **IUMLNode::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLNode**			

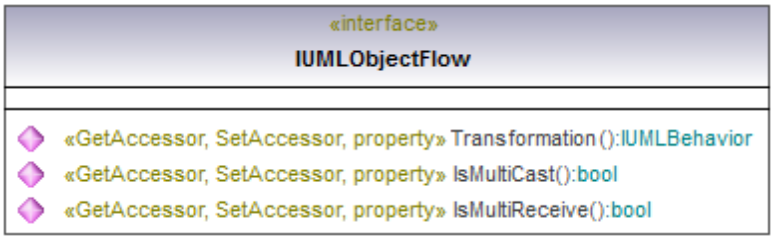
Operation **IUMLNode::NestedNodes**

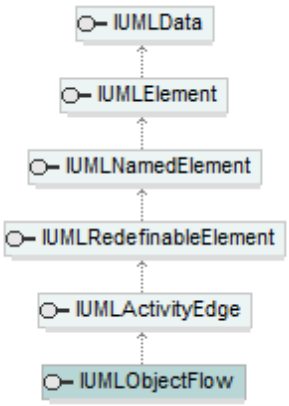
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLNode.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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**			

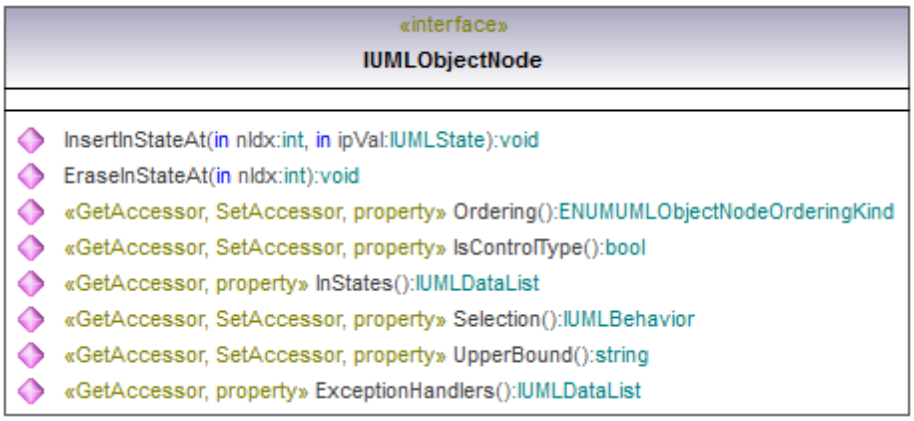
Operation **IUMLObjectFlow::Transformation**

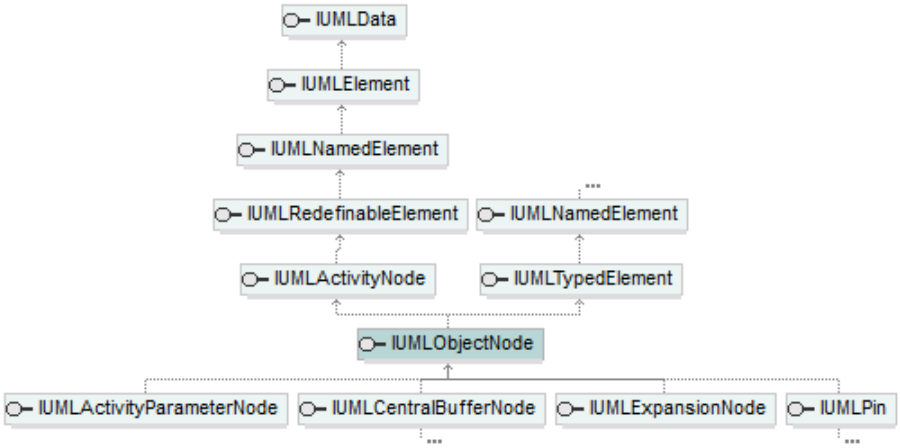
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLObjectNode

Interface **IUMLObjectNode**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLExceptionHandler** Operation **ExceptionInput** Operation **ExceptionInput**

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**			
documentation	A list of elements of type IUMLExceptionHandler.					

Operation **IUMLObjectNode::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLState**			
	return	**return**	**void**			

Operation **IUMLObjectNode::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLState.					

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**			

Operation **IUMLObjectNode::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLObjectNode::UpperBound**

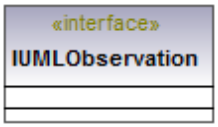
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

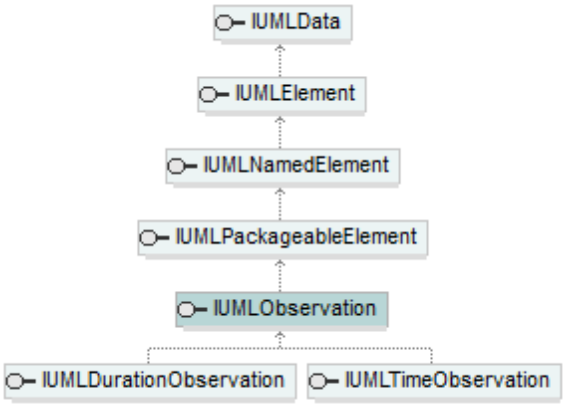
UModelAPI - IUMLObservation

Interface **IUMLObservation**

diagram



hierarchy



typedElements		
	Interface **IUMLDataAll**	Operation **InsertObservationAt**
	Interface **IUMLDuration**	Operation **InsertObservationAt**
	Interface **IUMLTimeExpression**	Operation **InsertObservationAt**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLOccurrenceSpecification

Interface **IUMLOccurrenceSpecification**

diagram	«interface» **IUMLOccurrenceSpecification** «GetAccessor, property» OccurringEvent():IUMLEvent «GetAccessor, property» ExecutionSpecificationStart():IUMLExecutionSpecification «GetAccessor, property» ExecutionSpecificationFinish():IUMLExecutionSpecification «GetAccessor, property» Covered():IUMLLifeline
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLInteractionFragment IUMLOccurrenceSpecification IUMLMessageOccurrenceSpecification
typedElements	Interface **IUMLDataAll** Interface **IUMLExecutionSpecification** Operation **Finish** **Start** Operation **Finish** **Start**

Operation **IUMLOccurrenceSpecification::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLOccurrenceSpecification::OccurringEvent**

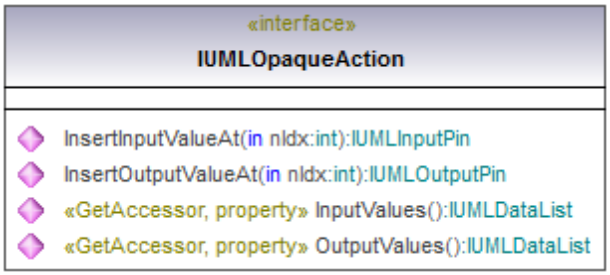
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

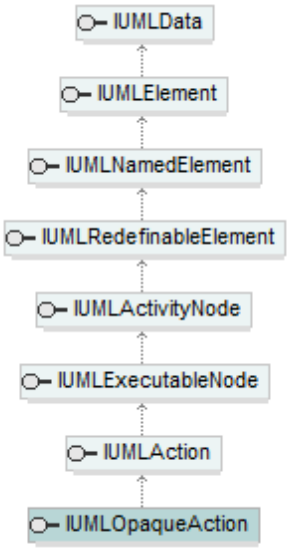
UModelAPI - IUMLOpaqueAction

Interface **IUMLOpaqueAction**

diagram



hierarchy



Operation **IUMLOpaqueAction::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLOpaqueAction::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLOpaqueAction::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLOpaqueAction::OutputValues**

 © 2014 Altova GmbH

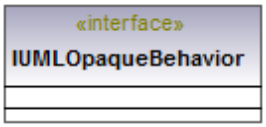
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

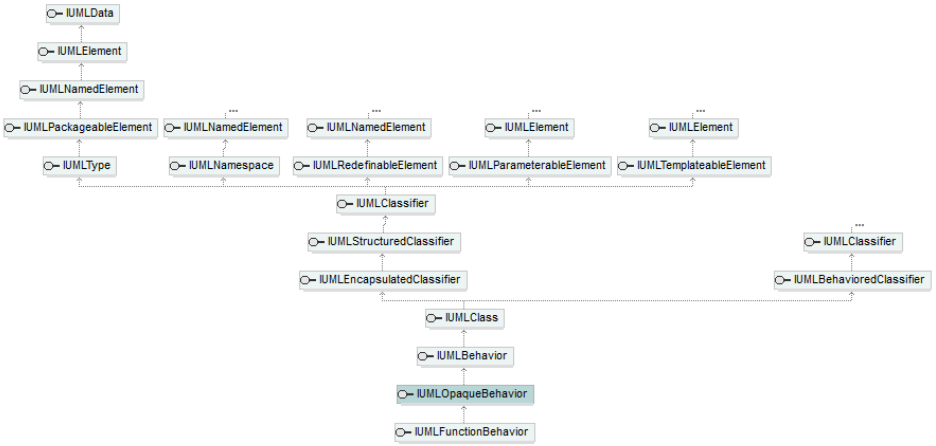
UModelAPI - IUMLOpaqueBehavior

Interface **IUMLOpaqueBehavior**

diagram



hierarchy

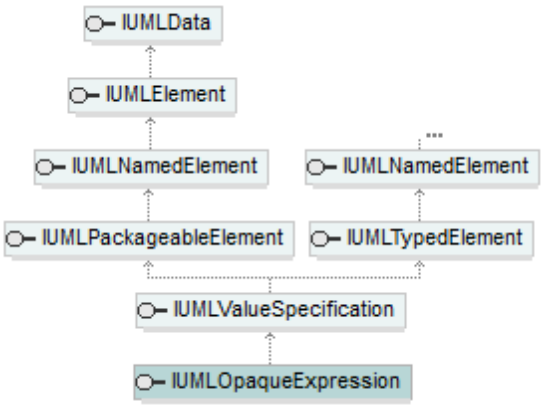


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLOpaqueExpression

Interface **IUMLOpaqueExpression**

diagram

hierarchy	
typedElements	Interface **IUMLAbstraction** Operation **Mapping** **SetNewMapping** Interface **IUMLDataAll** Operation **Mapping** **SetNewMapping**

Operation **IUMLOpaqueExpression::Body**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

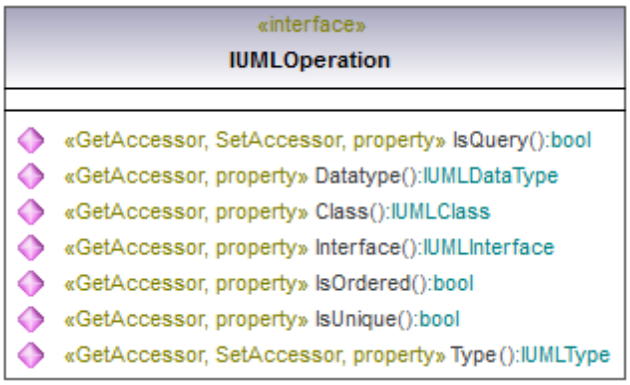
Operation **IUMLOpaqueExpression::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

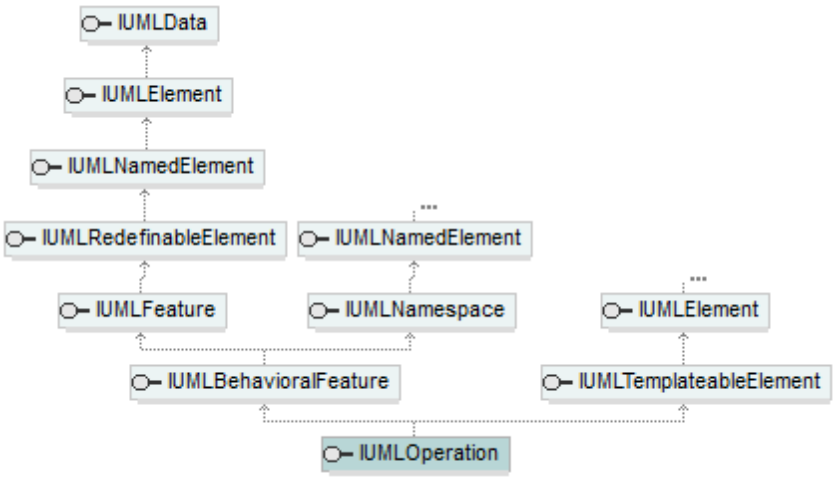
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLOperation

Interface **IUMLOperation**

diagram	

 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IDocument** Operation **GenerateSequenceDiagram** Interface **IUMLArtifact** Operation **InsertOwnedOperationAt** Interface **IUMLCallEvent** Operation **Operation** Interface **IUMLCallOperationAction** Operation **CallOperation** Interface **IUMLClass** Operation **InsertOwnedOperationAt** Interface **IUMLDataAll** Operation **CallOperation** **CodeOperation** **GetOperation** **InsertOwnedOperationAt** **Operation** **SetOperation** Interface **IUMLDataType** Operation **InsertOwnedOperationAt** Interface **IUMLGuiSequenceDiagram** Operation **CodeOperation** Interface **IUMLInterface** Operation **InsertOwnedOperationAt** Interface **IUMLMessage** Operation **GetOperation** **SetOperation** Interface **IUMLParameter** Operation **Operation**

Operation **IUMLOperation::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLOperation::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType**			

Operation **IUMLOperation::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

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**

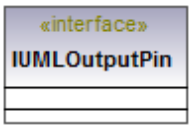
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

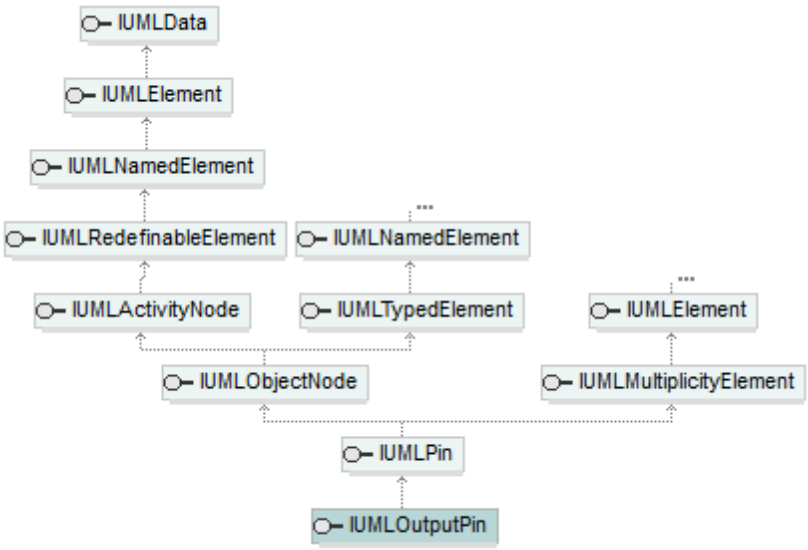
UModelAPI - IUMLOutputPin

Interface **IUMLOutputPin**

diagram



hierarchy



typedElements

Interface **IUMLAcceptEventAction**
Interface **IUMLCallAction**
Interface **IUMLDataAll**
Interface **IUMLOpaqueAction**
Interface **IUMLValueSpecificationAction**
Operation **InsertEventActionResultAt**
Operation **InsertResultAt**
Operation **InsertEventActionResultAt**
InsertOutputValueAt
InsertResultAt **Result**
Operation **InsertOutputValueAt**
Operation **Result**

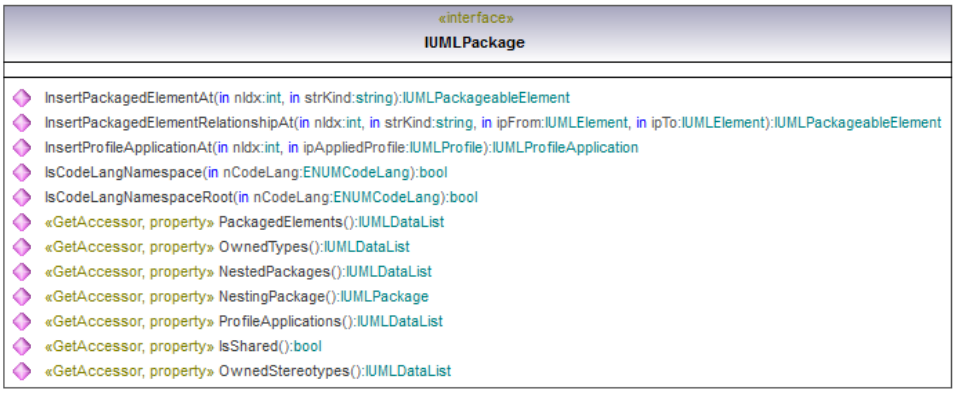
 © 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

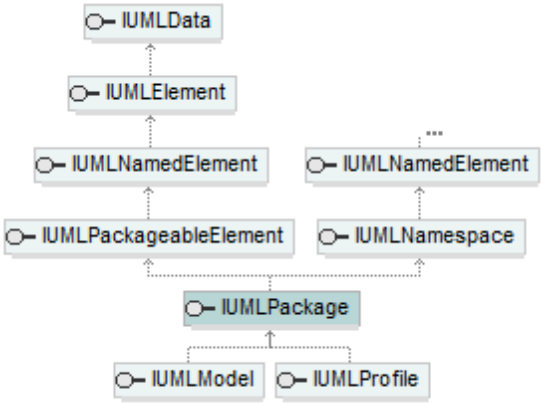
UModelAPI - IUMLPackage

Interface **IUMLPackage**

diagram



hierarchy



typedElements

Interface **IDocument**	Operation **RootPackage**
Interface **IImportSourceDlg**	Operation **ImportTarget**
Interface **IModelTransformationDlg**	Operation **SourcePackage** **TargetPackage**
Interface **IUMLDataAll**	Operation **ApplyingPackage** **ImportedPackage** **InsertPackageImportAt** **InsertPackageMergeAt** **MergedPackage** **NestingPackage** **OwningPackage** **ReceivingPackage**
Interface **IUMLNamespace**	Operation **InsertPackageImportAt** **InsertPackageMergeAt**
Interface **IUMLPackage**	Operation **NestingPackage**
Interface **IUMLPackageableElement**	Operation **OwningPackage**
Interface **IUMLPackageImport**	Operation **ImportedPackage**
Interface **IUMLPackageMerge**	Operation **MergedPackage** **ReceivingPackage**

	Interface IUMLProfileApplication — Operation ApplyingPackage Interface IUMLType — Operation Package

Operation **IUMLPackage::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLPackage::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLElement**			
	ipTo	**in**	**IUMLElement**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLPackage::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipAppliedProfile	**in**	**IUMLProfile**			
	return	**return**	**IUMLProfileApplication**			

Operation **IUMLPackage::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	return	**return**	**bool**			

Operation **IUMLPackage::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	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**			
documentation	A list of elements of type IUMLPackage.					

© 2014 Altova GmbH

Operation **IUMLPackage::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLPackage::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLPackage::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLType.					

Operation **IUMLPackage::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPackageableElement.					

Operation **IUMLPackage::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProfileApplication.					

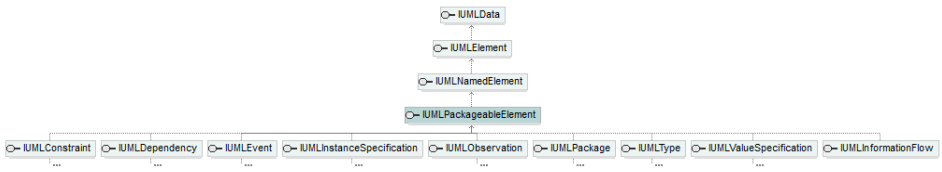
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLPackageableElement

Interface **IUMLPackageableElement**

diagram

hierarchy



typedElements

Interface **IUMLArtifact**
Interface **IUMLDataAll**
Operation **InsertManifestationAt**
Operation **ImportedElement**
InsertElementImportAt

	Interface **IUMLElementImport** Interface **IUMLManifestation** Interface **IUMLNamespace** Interface **IUMLPackage**	Operation **InsertManifestationAt** Operation **InsertPackagedElementAt** Operation **InsertPackagedElementRelationshipAt** Operation **UtilizedElement** Operation **ImportedElement** Operation **UtilizedElement** Operation **InsertElementImportAt** Operation **InsertPackagedElementAt** Operation **InsertPackagedElementRelationshipAt**

Operation **IUMLPackageableElement::OwningPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

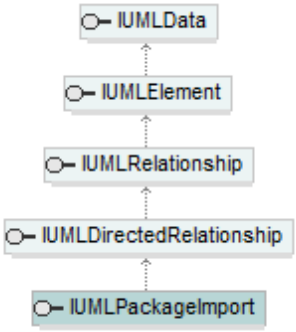
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLPackageImport

Interface **IUMLPackageImport**

diagram

hierarchy



typedElements	Interface **IUMLDataAll** Interface **IUMLNamespace**	Operation **InsertPackageImportAt** Operation **InsertPackageImportAt**

Operation **IUMLPackageImport::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLPackageImport::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**return**	**return**	**IUMLNamespace**

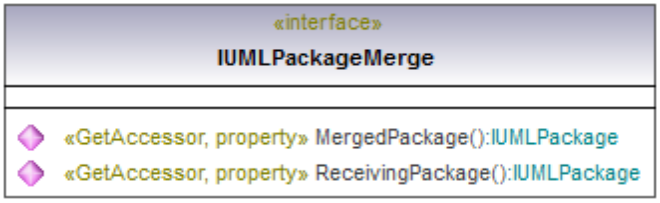
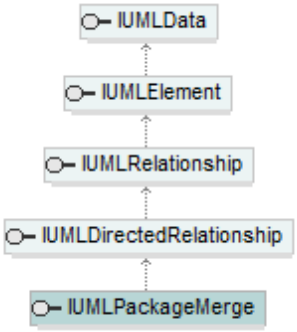
Operation **IUMLPackageImport::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLVisibilityKind**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLPackageMerge

Interface **IUMLPackageMerge**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLNamespace** Operation **InsertPackageMergeAt** Operation **InsertPackageMergeAt**

Operation **IUMLPackageMerge::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLPackageMerge::ReceivingPackage**

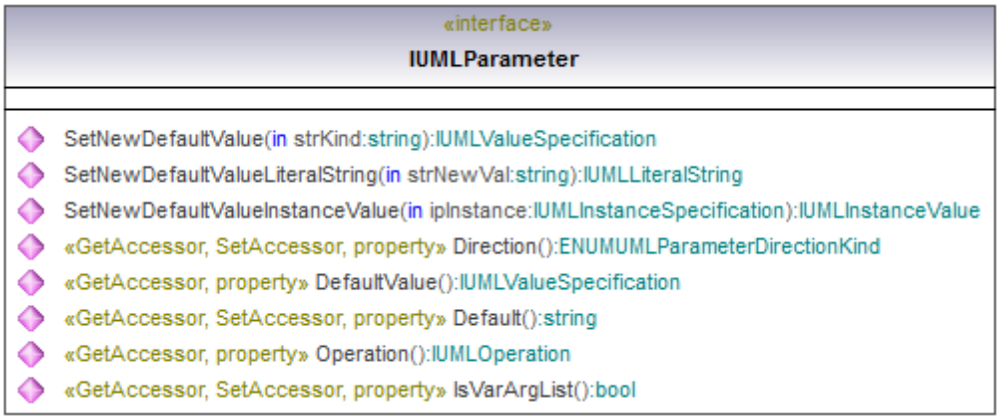
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

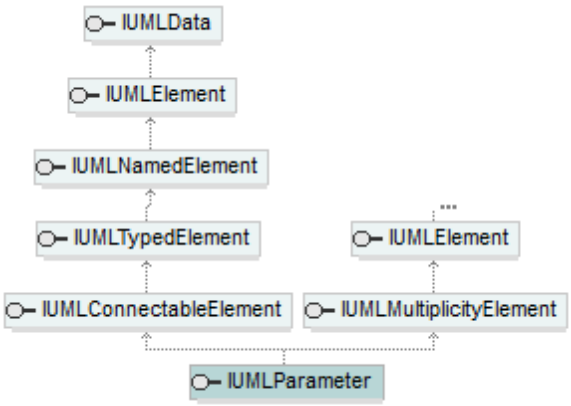
UModelAPI - IUMLParameter

Interface **IUMLParameter**

diagram



hierarchy



typed Elements

Interface **IUMLActivityParameterNode**	Operation **Parameter**
Interface **IUMLBehavior**	Operation **InsertOwnedParameterAt**
Interface **IUMLBehavioralFeature**	Operation **InsertOwnedParameterAt**
Interface **IUMLDataAll**	Operation **InsertOwnedParameterAt** **OwningParameter** **Parameter**
Interface **IUMLValueSpecification**	Operation **OwningParameter**

Operation **IUMLParameter::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLParameter::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLParameter::Direction**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLParameterDirectionKind**			

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**			

Operation **IUMLParameter::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLParameter::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLParameter::SetNewDefaultValueLiteralString**

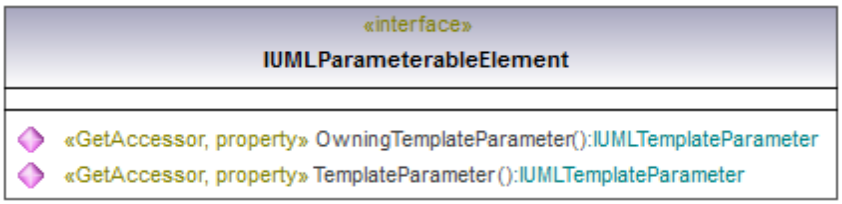
parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

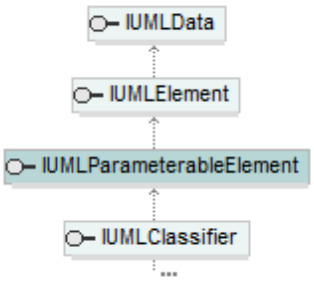
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLParameterableElement

Interface **IUMLParameterableElement**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **Actual** **InsertParameterSubstitutionAt** **OwnedActual** **OwnedParameteredElement** **ParameteredElement** **SetNewOwnedParameteredElement** Interface **IUMLTemplateBinding** — Operation **InsertParameterSubstitutionAt** Interface **IUMLTemplateParameter** — Operation **OwnedParameteredElement** **ParameteredElement** **SetNewOwnedParameteredElement** Interface **IUMLTemplateParameterSubstitution** — Operation **Actual** **OwnedActual**

Operation **IUMLParameterableElement::OwningTemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

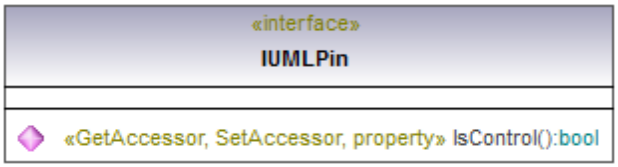
Operation **IUMLParameterableElement::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

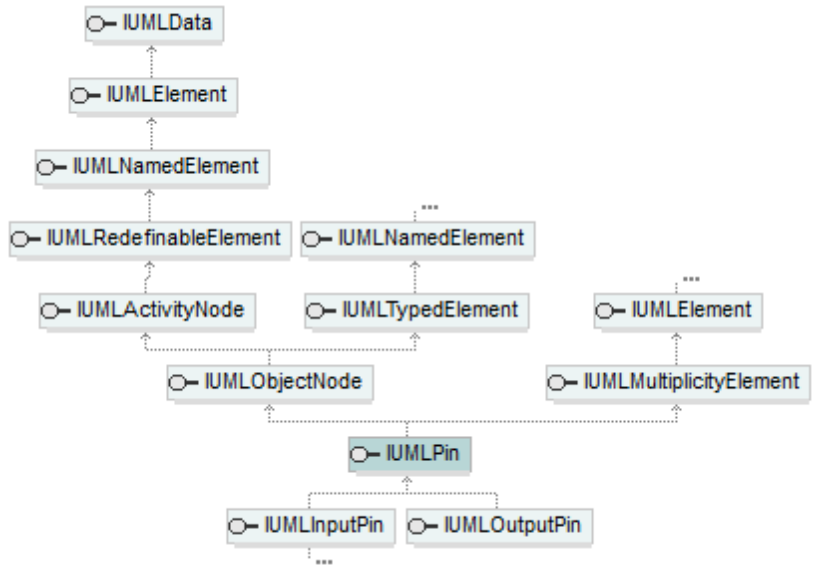
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLPin

Interface **IUMLPin**

diagram	

hierarchy



Operation **IUMLPin::IsControl**

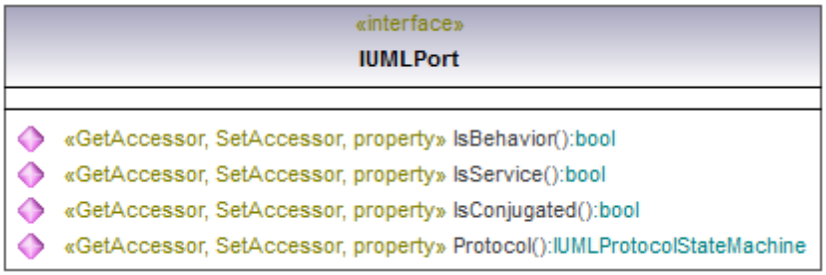
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

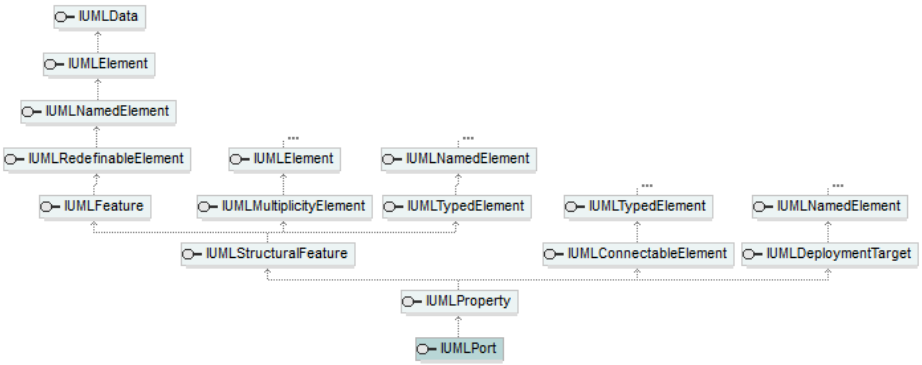
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLPort

Interface **IUMLPort**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLStructuredClassifier** Operation **InsertOwnedPortAt** Operation **InsertOwnedPortAt**

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**			

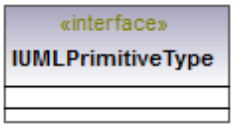
Operation **IUMLPort::Protocol**

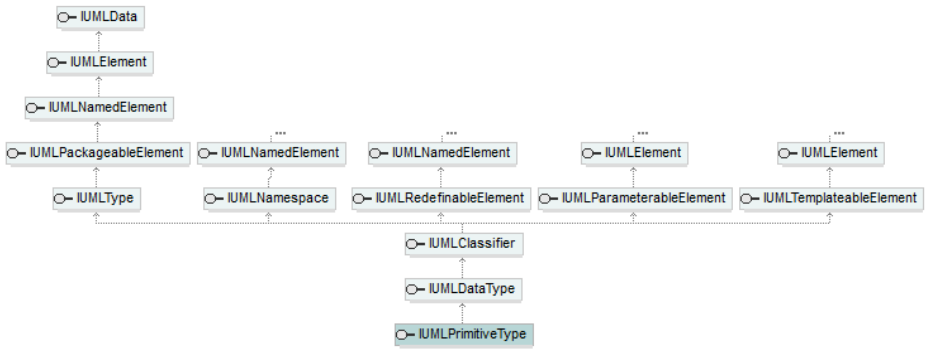
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLPrimitiveType

Interface **IUMLPrimitiveType**

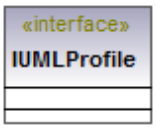
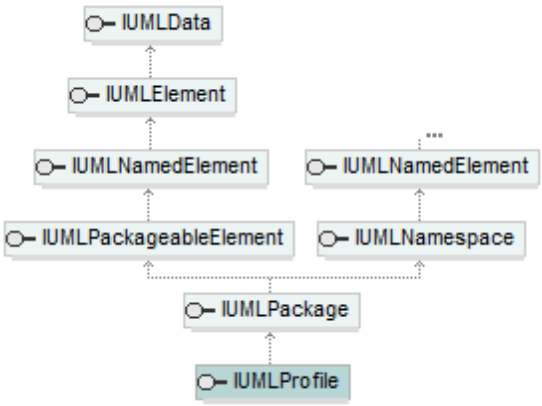
diagram	

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLProfile

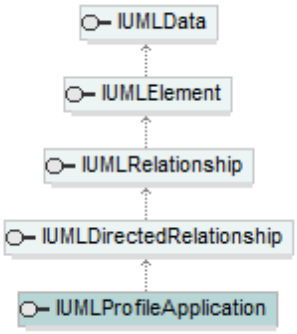
Interface **IUMLProfile**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLPackage** Interface **IUMLProfileApplication** Operation **AppliedProfile** **InsertProfileApplicationAt** Operation **InsertProfileApplicationAt** Operation **AppliedProfile**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLProfileApplication

Interface **IUMLProfileApplication**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLPackage** Operation **InsertProfileApplicationAt** Operation **InsertProfileApplicationAt**

Operation **IUMLProfileApplication::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProfile**			

Operation **IUMLProfileApplication::ApplyingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLProperty

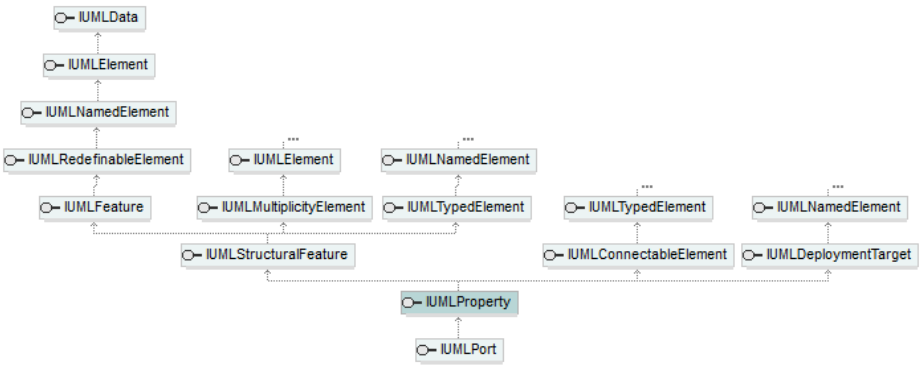
Interface **IUMLProperty**

diagram

«interface»
IUMLProperty

- InsertQualifierAt(in nIdx:int):IUMLProperty
- SetNewDefaultValue(in strKind:string):IUMLValueSpecification
- SetNewDefaultValueLiteralString(in strNewVal:string):IUMLLiteralString
- SetNewDefaultValueInstanceValue(in ipInstance:IUMLInstanceSpecification):IUMLInstanceValue
- «GetAccessor, SetAccessor, property» IsDerived():bool
- «GetAccessor, SetAccessor, property» IsDerivedUnion():bool
- «GetAccessor, property» IsComposite():bool
- «GetAccessor, property» Opposite():IUMLProperty
- «GetAccessor, property» Association():IUMLAssociation
- «GetAccessor, property» OwningAssociation():IUMLAssociation
- «GetAccessor, SetAccessor, property» IsNavigable():bool
- «GetAccessor, SetAccessor, property» IsOwnedEnd():bool
- «GetAccessor, property» AssociationEnd():IUMLProperty
- «GetAccessor, property» Qualifiers():IUMLDataList
- «GetAccessor, SetAccessor, property» Aggregation():ENUMUMLAggregationKind
- «GetAccessor, property» OwningSignal():IUMLSignal
- «GetAccessor, property» Datatype():IUMLDataType
- «GetAccessor, property» Class():IUMLClass
- «GetAccessor, property» Interface():IUMLInterface
- «GetAccessor, property» DefaultValue():IUMLValueSpecification
- «GetAccessor, SetAccessor, property» Default():string
- «GetAccessor, property» Classifier():IUMLClassifier

hierarchy



typedElements

Interface **IUMLArtifact**	Operation **InsertOwnedAttributeAt**
Interface **IUMLDataAll**	Operation **AssociationEnd** **InsertOwnedAttributeAt** **InsertQualifierAt** **Opposite** **OwningProperty**
Interface **IUMLDataType**	Operation **InsertOwnedAttributeAt**
Interface **IUMLInterface**	Operation **InsertOwnedAttributeAt**
Interface **IUMLProperty**	Operation **AssociationEnd** **InsertQualifierAt** **Opposite**
Interface **IUMLSignal**	Operation **InsertOwnedAttributeAt**

	Interface **IUMLStructuredClassifier** Interface **IUMLValueSpecification**	Operation **InsertOwnedAttributeAt** Operation **OwningProperty**

Operation **IUMLProperty::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLAggregationKind**			

Operation **IUMLProperty::Association**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLProperty::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLProperty::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLProperty::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLProperty::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType**			

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**			

Operation **IUMLProperty::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLProperty::Interface**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

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**			

Operation **IUMLProperty::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLProperty::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLProperty::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLProperty::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpe**			

	cification

Operation **IUMLProperty::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	IUMLInstanceSpecification			
	return	**return**	IUMLInstanceValue			

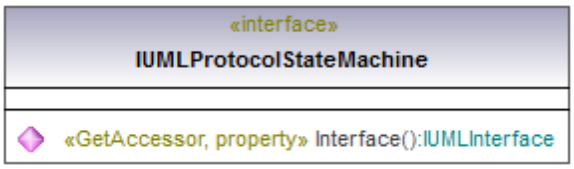
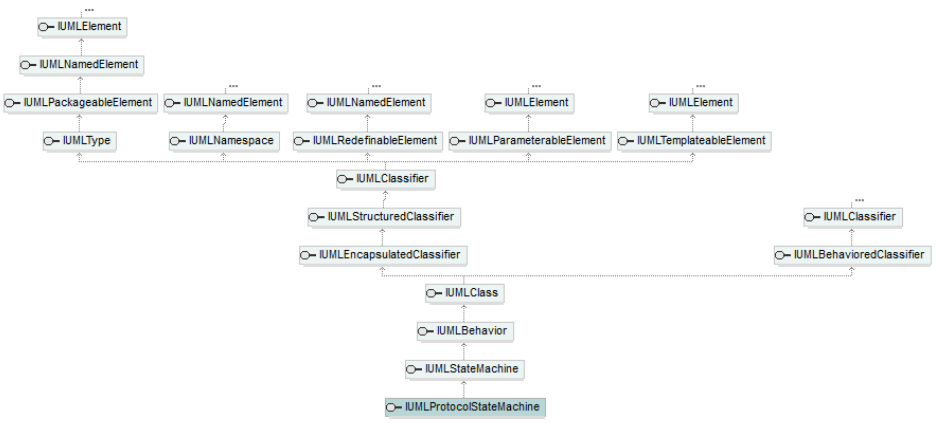
Operation **IUMLProperty::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	IUMLLiteralString			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLProtocolStateMachine

Interface **IUMLProtocolStateMachine**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInterface** Interface **IUMLPort** Operation **Protocol SetNewProtocol** Operation **Protocol SetNewProtocol** Operation **Protocol**

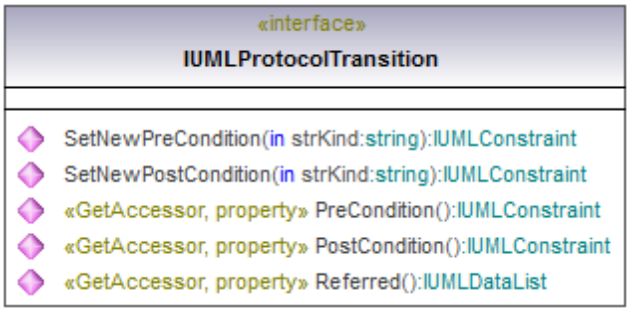
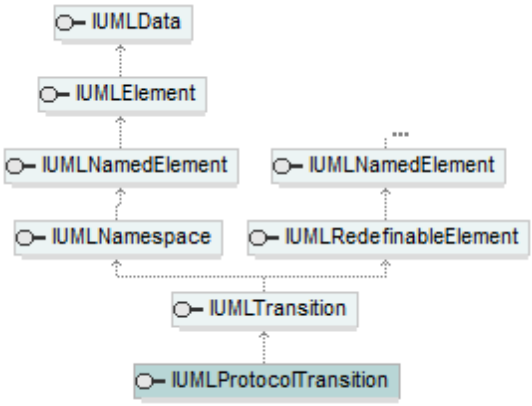
Operation **IUMLProtocolStateMachine::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLInterface			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLProtocolTransition

Interface **IUMLProtocolTransition**

diagram	
hierarchy	
typedElements	Interface **IUMLConstraint** Operation **OwningTransition** Interface **IUMLDataAll** Operation **OwningTransition**

Operation **IUMLProtocolTransition::PostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLProtocolTransition::PreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLProtocolTransition::Referred**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLProtocolTransition::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default

r	**strKind**	**in**	**string**
	return	**return**	**IUMLConstraint**

Operation **IUMLProtocolTransition::SetNewPreCondition**

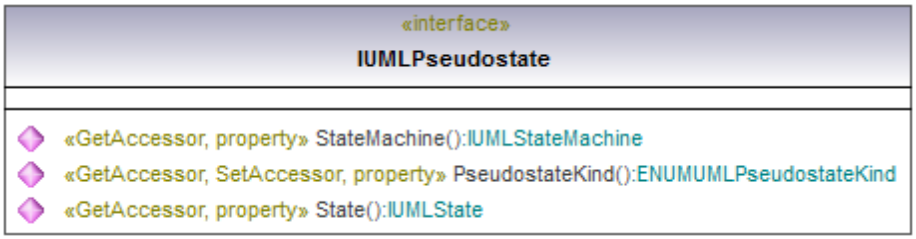
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

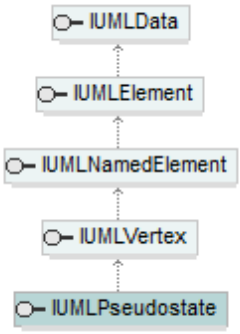
UModelAPI - IUMLPseudostate

Interface **IUMLPseudostate**

diagram



hierarchy



typedElements		
	Interface **IUMLConnectionPointReference**	Operation **InsertEntryAt** **InsertExitAt**
	Interface **IUMLDataAll**	Operation **InsertConnectionPointAt** **InsertEntryAt** **InsertExitAt**
	Interface **IUMLState**	Operation **InsertConnectionPointAt**
	Interface **IUMLStateMachine**	Operation **InsertConnectionPointAt**

Operation **IUMLPseudostate::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLPseudostateKind**			

Operation **IUMLPseudostate::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLPseudostate::StateMachine**

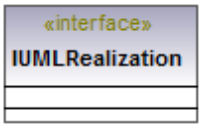
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

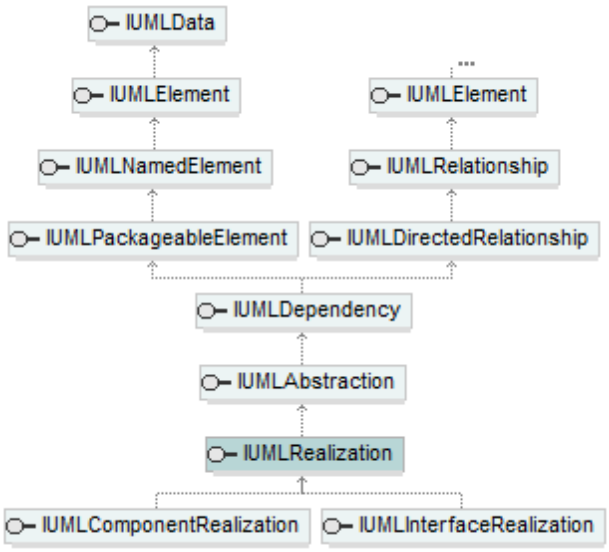
UModelAPI - IUMLRealization

Interface **IUMLRealization**

diagram



hierarchy

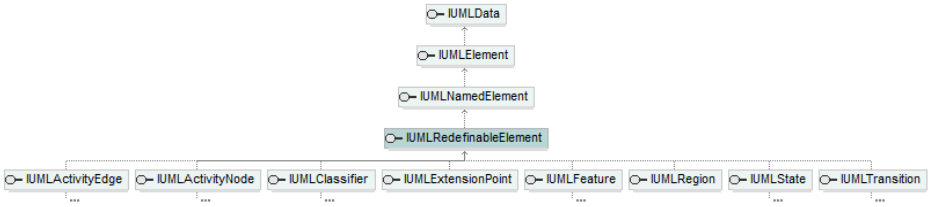


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLRedefinableElement

Interface **IUMLRedefinableElement**

diagram

hierarchy	

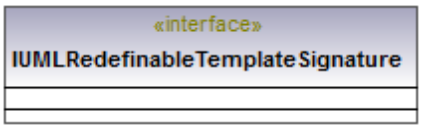
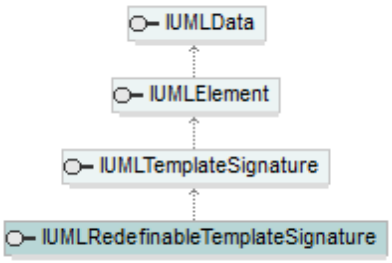
Operation **IUMLRedefinableElement::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLRedefinableTemplateSignature

Interface **IUMLRedefinableTemplateSignature**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLRegion

Interface **IUMLRegion**

diagram	«interface» **IUMLRegion** InsertSubVertexAt(in nIdx:int, in strKind:string):IUMLVertex InsertTransitionAt(in nIdx:int, in ipSource:IUMLVertex, in ipTarget:IUMLVertex):IUMLTransition «GetAccessor, property» SubVertices():IUMLDataList «GetAccessor, property» Transitions():IUMLDataList «GetAccessor, property» StateMachine():IUMLStateMachine «GetAccessor, property» State():IUMLState
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLNamedElement IUMLRedefinableElement IUMLNamespace IUMLRegion
typedElements	Interface **IUMLDataAll** Interface **IUMLState** Interface **IUMLStateMachine** Interface **IUMLVertex** Operation **Container InsertRegionAt** Operation **InsertRegionAt** Operation **InsertRegionAt** Operation **Container**

Operation **IUMLRegion::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLVertex**			

Operation **IUMLRegion::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSource	**in**	**IUMLVertex**			
	ipTarget	**in**	**IUMLVertex**			
	return	**return**	**IUMLTransition**			

Operation **IUMLRegion::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

Operation **IUMLRegion::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

Operation **IUMLRegion::SubVertices**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLVertex.					

Operation **IUMLRegion::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTransition.					

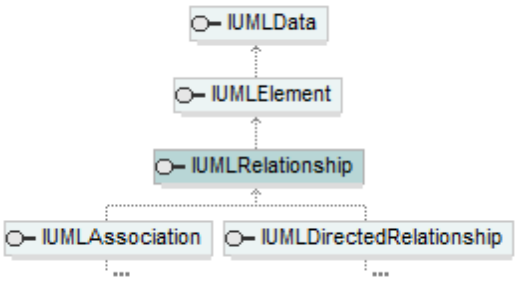
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLRelationship

Interface **IUMLRelationship**

diagram

hierarchy



typedElements		
	Interface **IUMLDataAll**	Operation **InsertInformationFlowRealizationAt**
	Interface **IUMLInformationFlow**	Operation **InsertInformationFlowRealizationAt**

Operation **IUMLRelationship::RelatedElements**

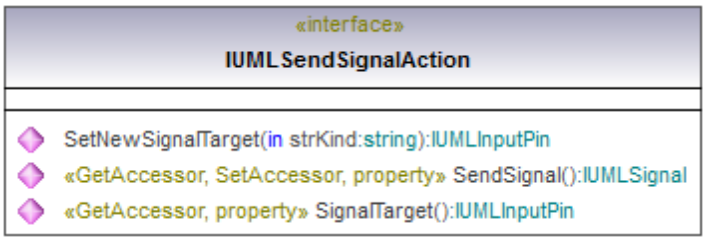
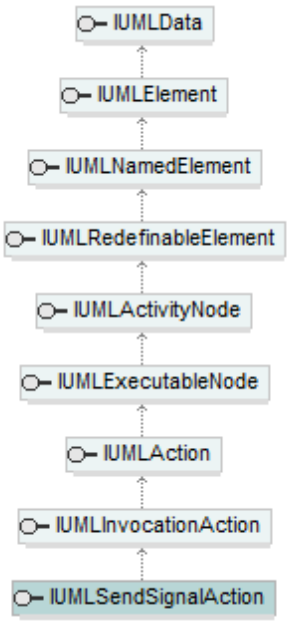
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLElement.					

 © 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLSendSignalAction

Interface **IUMLSendSignalAction**

diagram	
hierarchy	

Operation **IUMLSendSignalAction::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLSendSignalAction::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

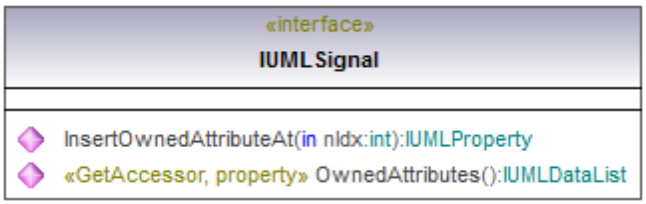
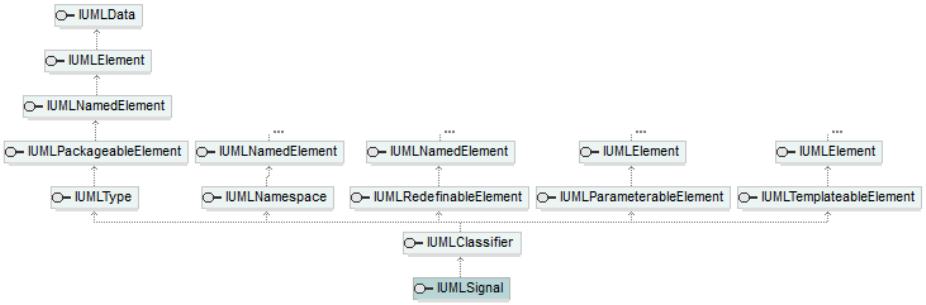
Operation **IUMLSendSignalAction::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLSignal

Interface **IUMLSignal**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLProperty** Interface **IUMLSendSignalAction** Interface **IUMLSignalEvent** Operation **OwningSignal** **SendSignal** **Signal** Operation **OwningSignal** Operation **SendSignal** Operation **Signal**

Operation **IUMLSignal::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLSignal::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLSignalEvent

Interface **IUMLSignalEvent**

diagram	«interface» IUMLSignalEvent «GetAccessor, SetAccessor, property» Signal():IUMLSignal
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLPackageableElement IUMLEvent IUMLMessageEvent IUMLSignalEvent

Operation **IUMLSignalEvent::Signal**

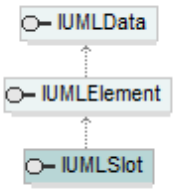
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLSlot

Interface **IUMLSlot**

diagram	«interface» IUMLSlot InsertValueAt(in nIdx:int, in strKind:string):IUMLValueSpecification «GetAccessor, property» OwningInstance():IUMLInstanceSpecification «GetAccessor, property» Values():IUMLDataList «GetAccessor, property» DefiningFeature():IUMLStructuralFeature

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLInstanceSpecification** Interface **IUMLValueSpecification** Operation **InsertSlotAt OwningSlot** Operation **InsertSlotAt** Operation **OwningSlot**

Operation **IUMLSlot::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStructuralFeature**			

Operation **IUMLSlot::InsertValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLSlot::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLSlot::Values**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLValueSpecification.					

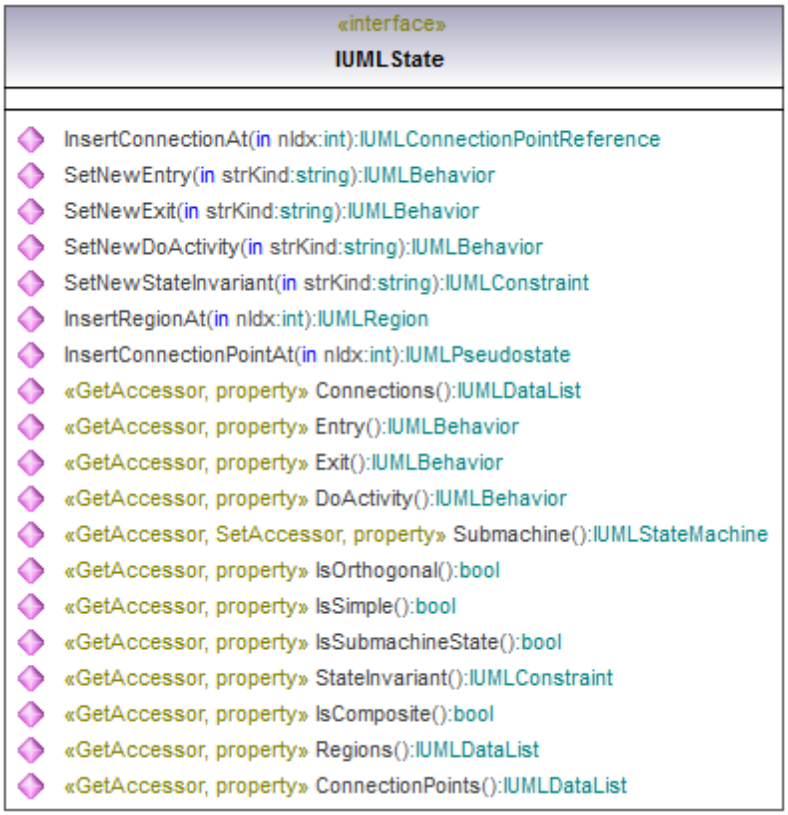
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

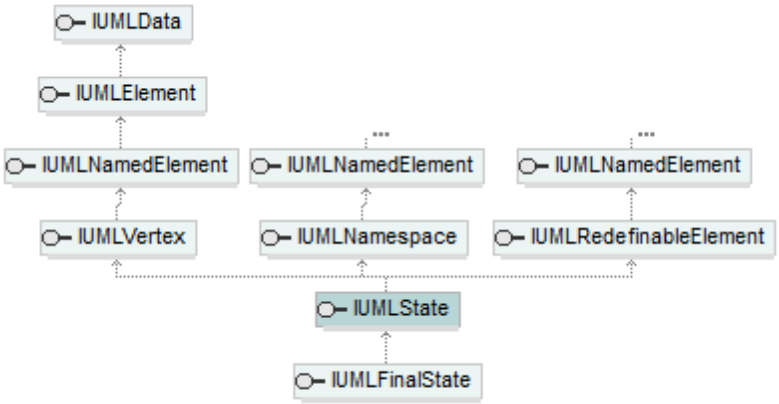
UModelAPI - IUMLState

Interface **IUMLState**

diagram



hierarchy



typedElements

Interface **IUMLConnectionPointReference**	Operation **State**
Interface **IUMLConstraint**	Operation **OwningState**
Interface **IUMLDataAll**	Operation **InsertInStateAt** **OwningState** **State**
Interface **IUMLObjectNode**	Operation **InsertInStateAt**
Interface **IUMLPseudostate**	Operation **State**
Interface **IUMLRegion**	Operation **State**

Operation **IUMLState::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

Operation **IUMLState::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConnectionPointReference.					

Operation **IUMLState::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConnectionPointReference**			

Operation **IUMLState::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate**			

Operation **IUMLState::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion**			

Operation **IUMLState::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

 © 2014 Altova GmbH

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**			
documentation	A list of elements of type IUMLRegion.					

Operation **IUMLState::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLState::SetNewStateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLState::StateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLState::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMac**			

	hine

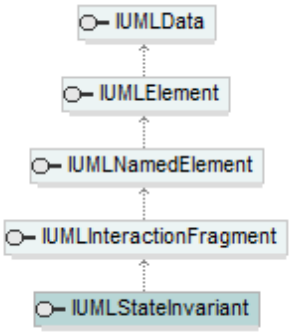
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLStateInvariant

Interface **IUMLStateInvariant**

diagram

hierarchy



Operation **IUMLStateInvariant::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLStateInvariant::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLStateInvariant::SetNewInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLStateMachine

Interface **IUMLStateMachine**

diagram	«interface» **IUMLStateMachine** InsertRegionAt(in nIdx:int):IUMLRegion InsertConnectionPointAt(in nIdx:int):IUMLPseudostate «GetAccessor, property» Regions():IUMLDataList «GetAccessor, property» SubmachineStates():IUMLDataList «GetAccessor, property» ConnectionPoints():IUMLDataList
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLPackageableElement ... IUMLNamedElement ... IUMLNamedElement ... IUMLElement ... IUMLElement IUMLType IUMLNamespace IUMLRedefinableElement IUMLParameterableElement IUMLTemplateableElement IUMLClassifier IUMLStructuredClassifier ... IUMLClassifier IUMLEncapsulatedClassifier IUMLBehavioredClassifier IUMLClass IUMLBehavior IUMLStateMachine IUMLProtocolStateMachine
typedElements	Interface **IDocument** Interface **IUMLDataAll** Interface **IUMLPseudostate** Interface **IUMLRegion** Interface **IUMLState** Operation **GenerateStateMachineCode** Operation **StateMachine** **Submachine** Operation **StateMachine** Operation **StateMachine** Operation **Submachine**

Operation **IUMLStateMachine::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLPseudostate.					

Operation **IUMLStateMachine::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPseudostate**			

Operation **IUMLStateMachine::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion**			

Operation **IUMLStateMachine::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLRegion.					

Operation **IUMLStateMachine::SubmachineStates**

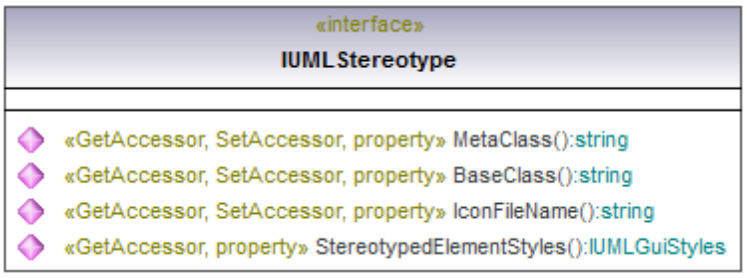
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLState.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

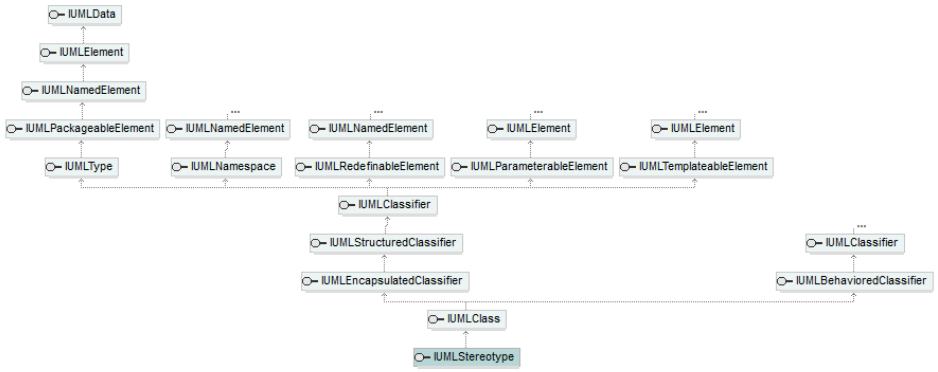
UModelAPI - IUMLStereotype

Interface **IUMLStereotype**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**

Operation **ApplyStereotype**
GetStereotypeApplicationForStereotype
IsStereotypeApplied
Stereotype
UnapplyStereotype

Interface **IUMLElement**

Operation **ApplyStereotype**
GetStereotypeApplicationForStereotype
IsStereotypeApplied

 © 2014 Altova GmbH

	Interface **IUMLStereotypeApplication**	Operation	**UnapplyStereotype Stereotype**

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**

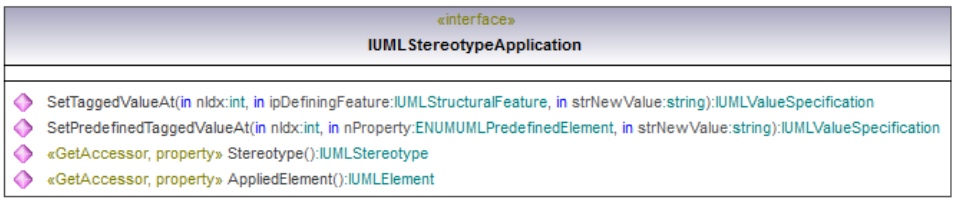
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

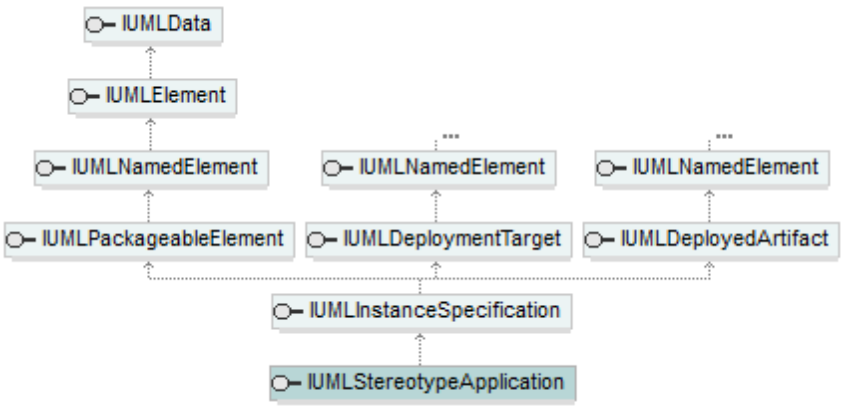
UModelAPI - IUMLStereotypeApplication

Interface **IUMLStereotypeApplication**

diagram



hierarchy



typedEle	Interface **IUMLDataAll**	Operation **ApplyPredefinedStereotype**

ments	Interface IUMLElement	Operation	ApplyStereotype GetStereotypeApplicationFor PredefinedStereotype GetStereotypeApplicationFor Stereotype ApplyPredefinedStereotype ApplyStereotype GetStereotypeApplicationFor PredefinedStereotype GetStereotypeApplicationFor Stereotype

Operation **IUMLStereotypeApplication::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLElement			

Operation **IUMLStereotypeApplication::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nProperty	**in**	ENUMUMLPredefinedElement			
	strNewValue	**in**	**string**			
	return	**return**	IUMLValueSpecification			

Operation **IUMLStereotypeApplication::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	IUMLStructuralFeature			
	strNewValue	**in**	**string**			
	return	**return**	IUMLValueSpecification			

Operation **IUMLStereotypeApplication::Stereotype**

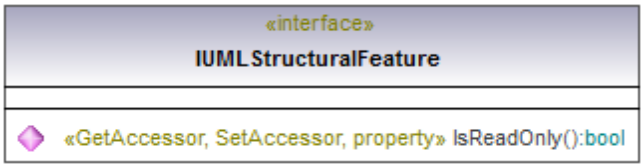
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLStereotype			

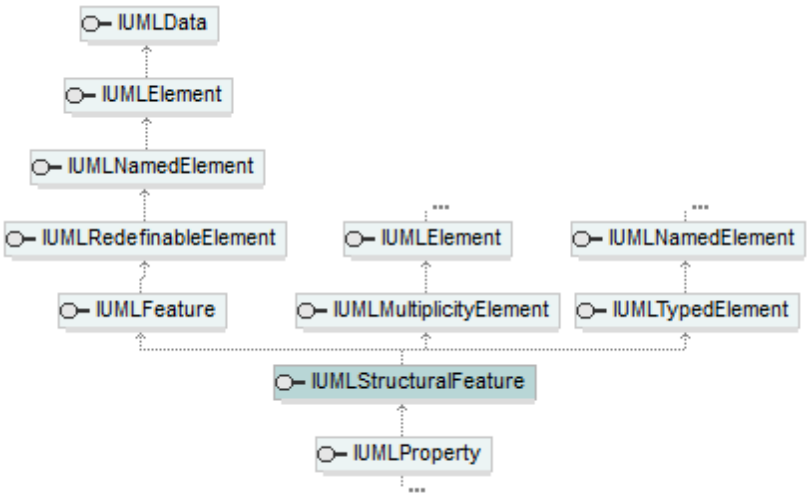
UML documentation generated by UModel UML Editor http://www.altova.com/umodel 09/04/12 14:02:42

UModelAPI - IUMLStructuralFeature

Interface **IUMLStructuralFeature**

diagram



hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **DefiningFeature** **InsertSlotAt** **SetSlotValueAt** **SetTaggedValueAt** Interface **IUMLInstanceSpecification** — Operation **InsertSlotAt** **SetSlotValueAt** Interface **IUMLSlot** — Operation **DefiningFeature** Interface **IUMLStereotypeApplication** — Operation **SetTaggedValueAt**

Operation **IUMLStructuralFeature::IsReadOnly**

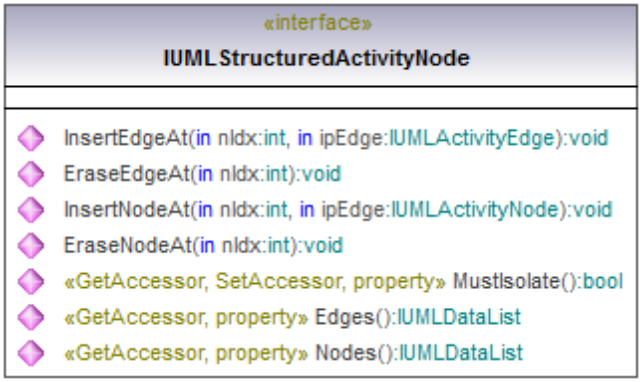
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

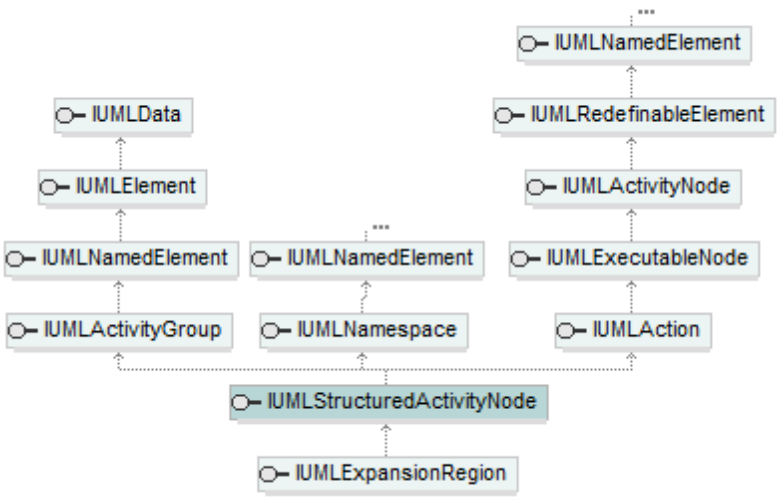
UModelAPI - IUMLStructuredActivityNode

Interface **IUMLStructuredActivityNode**

diagram



hierarchy



Operation **IUMLStructuredActivityNode::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLStructuredActivityNode::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

 © 2014 Altova GmbH

Operation **IUMLStructuredActivityNode::Nodes**

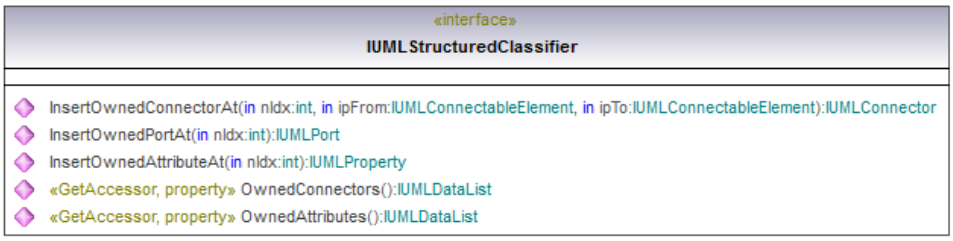
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

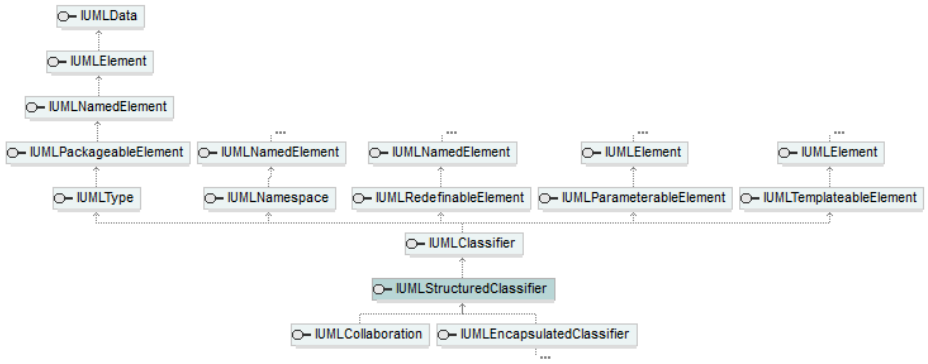
UModelAPI - IUMLStructuredClassifier

Interface **IUMLStructuredClassifier**

diagram



hierarchy



Operation **IUMLStructuredClassifier::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLStructuredClassifier::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFrom	**in**	**IUMLConnectableElement**			
	ipTo	**in**	**IUMLConnectableElement**			
	return	**return**	**IUMLConnector**			

Operation **IUMLStructuredClassifier::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

	return	**return**	**IUMLPort**

Operation **IUMLStructuredClassifier::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLProperty.					

Operation **IUMLStructuredClassifier::OwnedConnectors**

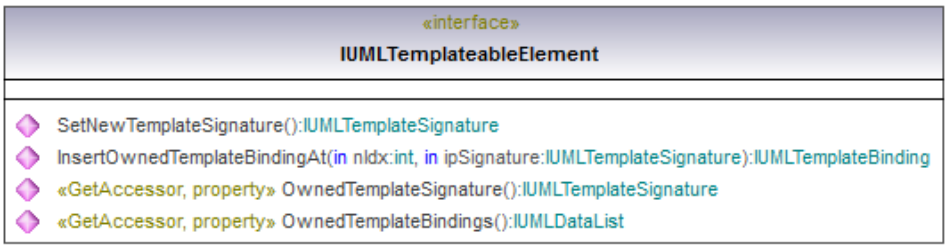
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLConnector.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

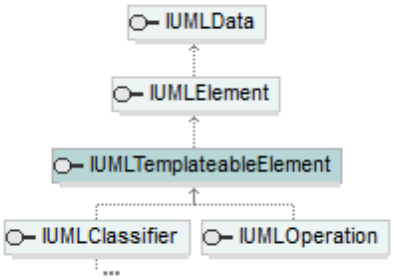
UModelAPI - IUMLTemplateableElement

Interface **IUMLTemplateableElement**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**	Operation **BoundElement** **Template**
Interface **IUMLTemplateBinding**	Operation **BoundElement**
Interface **IUMLTemplateSignature**	Operation **Template**

Operation **IUMLTemplateableElement::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSignature	**in**	**IUMLTemplate**			

 © 2014 Altova GmbH

	return	**return**	**Signature IUMLTemplate Binding**

Operation **IUMLTemplateableElement::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTemplateBinding.					

Operation **IUMLTemplateableElement::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplate Signature**			

Operation **IUMLTemplateableElement::SetNewTemplateSignature**

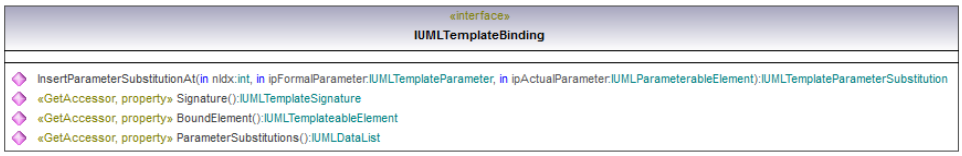
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplate Signature**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

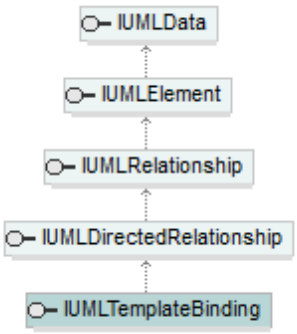
UModelAPI - IUMLTemplateBinding

Interface **IUMLTemplateBinding**

diagram



hierarchy



typedElements

Interface **IUMLDataAll**
Interface **IUMLTemplateableElement**
Interface **IUMLTemplateParameterSub**
Operation **InsertOwnedTemplateBindingAt TemplateBinding**
Operation **InsertOwnedTemplateBindingAt**
Operation **TemplateBinding**

	stitution

Operation **IUMLTemplateBinding::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateableElement**			

Operation **IUMLTemplateBinding::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFormalParameter	**in**	**IUMLTemplateParameter**			
	ipActualParameter	**in**	**IUMLParameterableElement**			
	return	**return**	**IUMLTemplateParameterSubstitution**			

Operation **IUMLTemplateBinding::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTemplateParameterSubstitution.					

Operation **IUMLTemplateBinding::Signature**

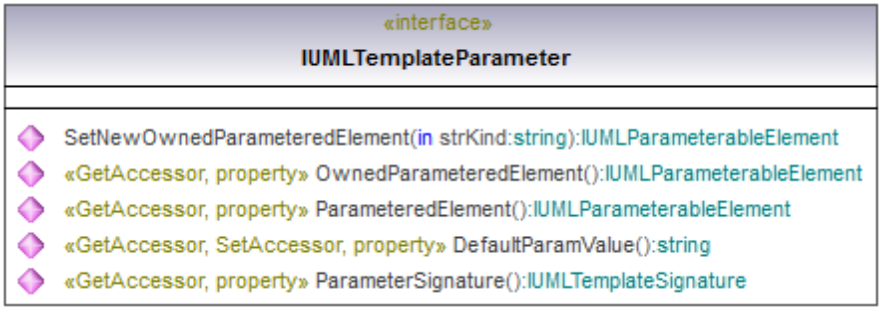
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

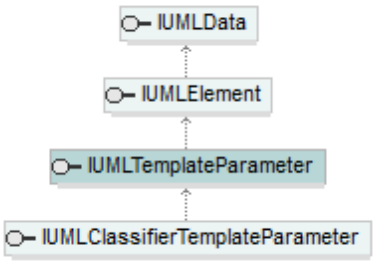
UModelAPI - IUMLTemplateParameter

Interface **IUMLTemplateParameter**

diagram



 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLParameterableElement** Interface **IUMLTemplateBinding** Interface **IUMLTemplateParameterSubstitution** Operation **Formal** Operation **InsertParameterSubstitutionAt** Operation **OwningTemplateParameter** Operation **TemplateParameter** Operation **OwningTemplateParameter** Operation **TemplateParameter** Operation **InsertParameterSubstitutionAt** Operation **Formal**

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**			

Operation **IUMLTemplateParameter::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameter::ParameterSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

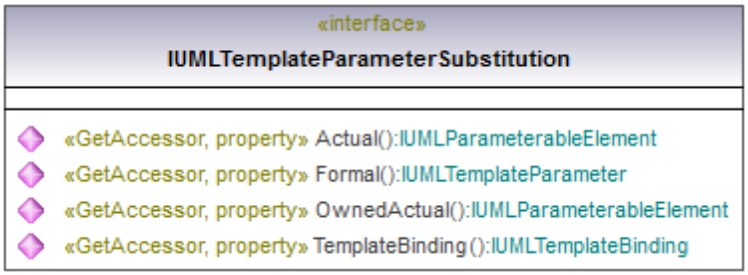
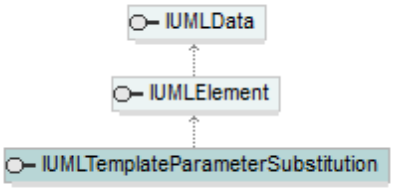
Operation **IUMLTemplateParameter::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLParameterableElement**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTemplateParameterSubstitution

Interface **IUMLTemplateParameterSubstitution**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLTemplateBinding** Operation **InsertParameterSubstitutionAt** Operation **InsertParameterSubstitutionAt**

Operation **IUMLTemplateParameterSubstitution::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameterSubstitution::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

Operation **IUMLTemplateParameterSubstitution::OwnedActual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLTemplateParameterSubstitution::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateBinding**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLTemplateSignature

Interface **IUMLTemplateSignature**

diagram	«interface» **IUMLTemplateSignature** InsertOwnedTemplateParameterAt(in nIdx:int):IUMLClassifierTemplateParameter «GetAccessor, property» OwnedTemplateParameters():IUMLDataList «GetAccessor, property» Template():IUMLTemplateableElement
hierarchy	IUMLData IUMLElement IUMLTemplateSignature IUMLRedefinableTemplateSignature
typedElements	Interface **IUMLDataAll** — Operation **InsertOwnedTemplateBindingAt** **OwnedTemplateSignature** **ParameterSignature** **SetNewTemplateSignature** **Signature** Interface **IUMLTemplateableElement** — Operation **InsertOwnedTemplateBindingAt** **OwnedTemplateSignature** **SetNewTemplateSignature** Interface **IUMLTemplateBinding** — Operation **Signature** Interface **IUMLTemplateParameter** — Operation **ParameterSignature**

Operation **IUMLTemplateSignature::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLClassifierTemplateParameter**			

Operation **IUMLTemplateSignature::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTemplateParameter.					

Operation **IUMLTemplateSignature::Template**

parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLTemplateableElement**

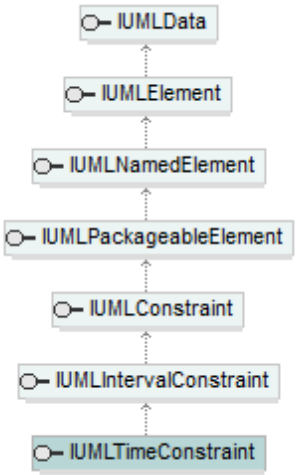
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTimeConstraint

Interface **IUMLTimeConstraint**

diagram

hierarchy



Operation **IUMLTimeConstraint::IsFirstEvent**

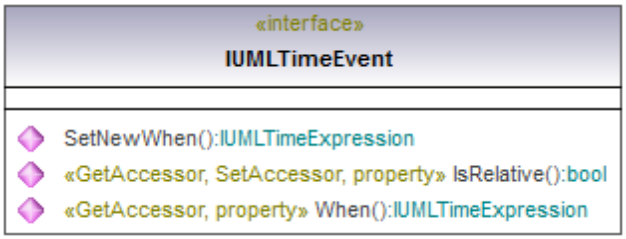
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

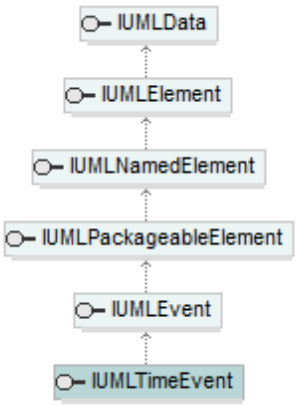
UModelAPI - IUMLTimeEvent

Interface **IUMLTimeEvent**

diagram



 © 2014 Altova GmbH

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**			

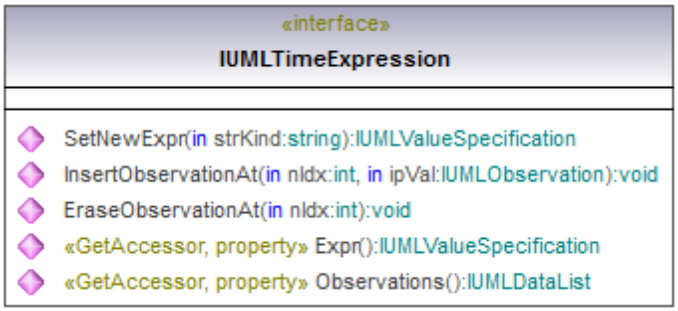
Operation **IUMLTimeEvent::When**

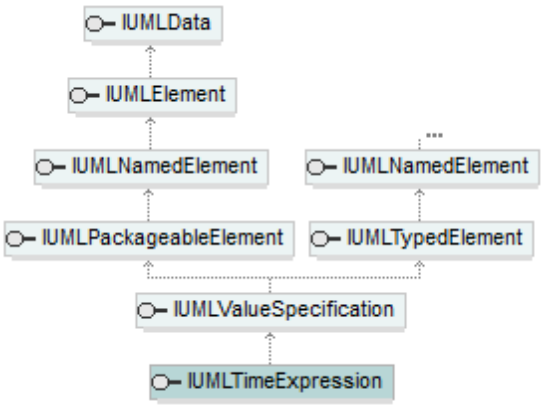
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTimeExpression

Interface **IUMLTimeExpression**

diagram	

hierarchy	
typedElements	Interface **IUMLDataAll** Operation **SetNewWhen** **When** Interface **IUMLTimeEvent** Operation **SetNewWhen** **When**

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**			

Operation **IUMLTimeExpression::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation**			
	return	**return**	**void**			

Operation **IUMLTimeExpression::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLObservation.					

Operation **IUMLTimeExpression::SetNewExpr**

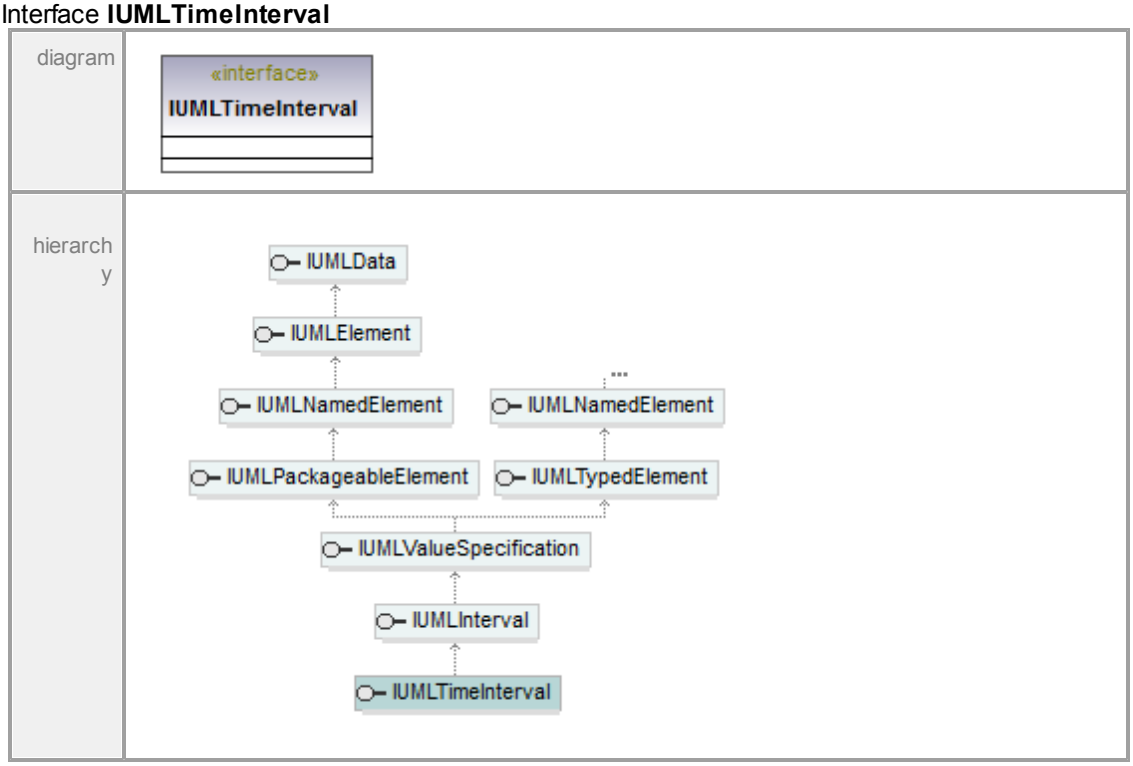
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

 © 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTimeInterval

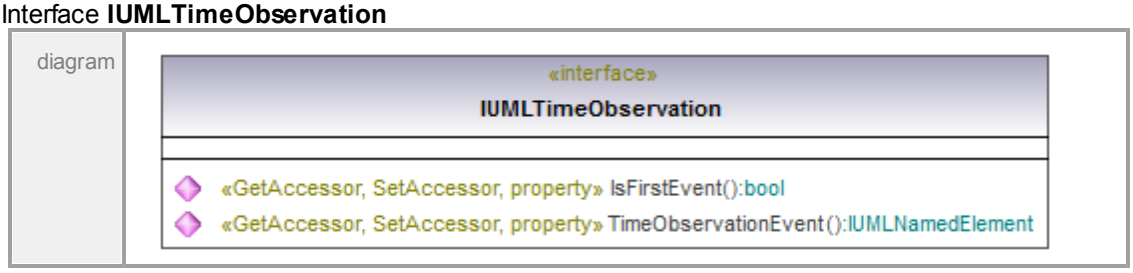
Interface **IUMLTimeInterval**

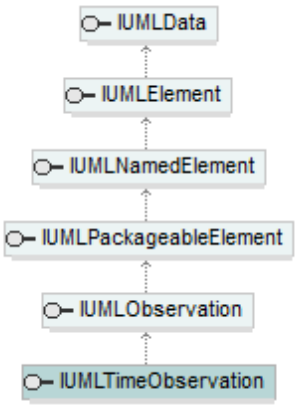


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTimeObservation

Interface **IUMLTimeObservation**



hierarchy	

Operation **IUMLTimeObservation::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

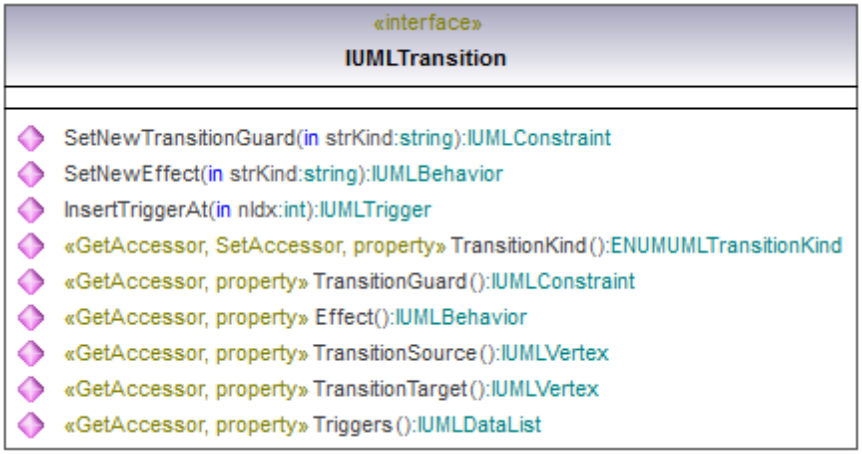
Operation **IUMLTimeObservation::TimeObservationEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamedElement**			

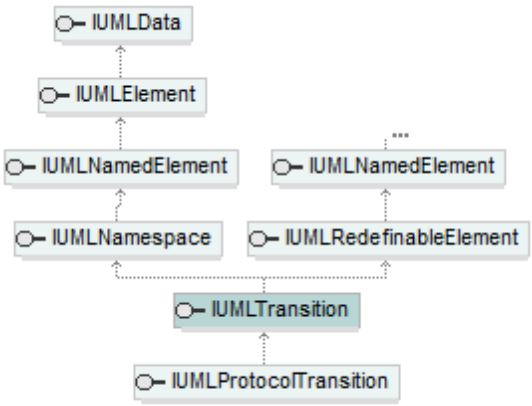
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTransition

Interface **IUMLTransition**

diagram	

 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLRegion** Operation **InsertTransitionAt** Operation **InsertTransitionAt**

Operation **IUMLTransition::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLTransition::InsertTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLTransition::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLTransition::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLTransition::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLTransition::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLTransitionKind**			

Operation **IUMLTransition::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLTransition::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLTransition::Triggers**

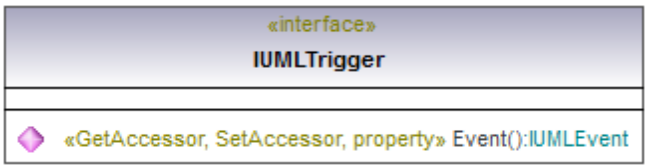
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTrigger.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

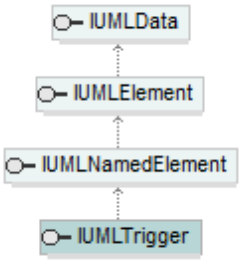
UModelAPI - IUMLTrigger

Interface **IUMLTrigger**

diagram



hierarchy



typedElements		
	Interface **IUMLAcceptEventAction**	Operation **InsertActionTriggerAt**
	Interface **IUMLDataAll**	Operation **InsertActionTriggerAt** **InsertTriggerAt**
	Interface **IUMLTransition**	Operation **InsertTriggerAt**

Operation **IUMLTrigger::Event**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEvent**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLType

Interface **IUMLType**

diagram	«interface» **IUMLType** «GetAccessor, property» TypedElements():IUMLDataList «GetAccessor, property» Package():IUMLPackage
hierarchy	IUMLData IUMLElement IUMLNamedElement IUMLPackageableElement IUMLType IUMLClassifier ...
typedElements	Interface **IUMLBehavioralFeature** Interface **IUMLDataAll** Interface **IUMLOperation** Interface **IUMLTypedElement** Operation **InsertRaisedExceptionAt** Operation **InsertRaisedExceptionAtType** Operation **Type** Operation **Type**

Operation **IUMLType::Package**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

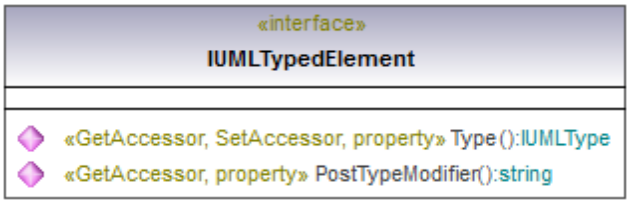
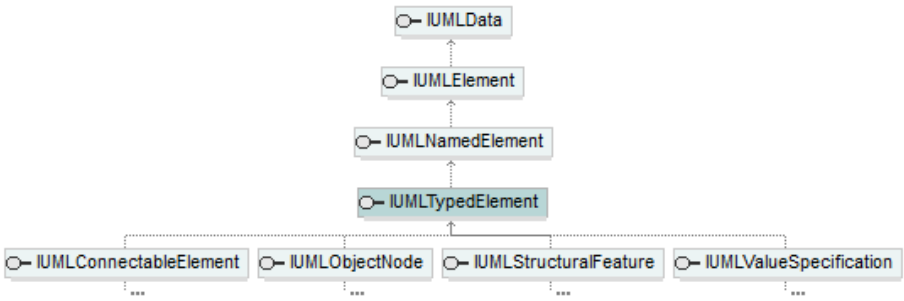
Operation **IUMLType::TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTypedElement.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLTypedElement

Interface **IUMLTypedElement**

diagram	
hierarchy	

Operation **IUMLTypedElement::PostTypeModifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

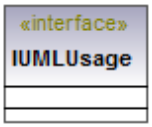
Operation **IUMLTypedElement::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

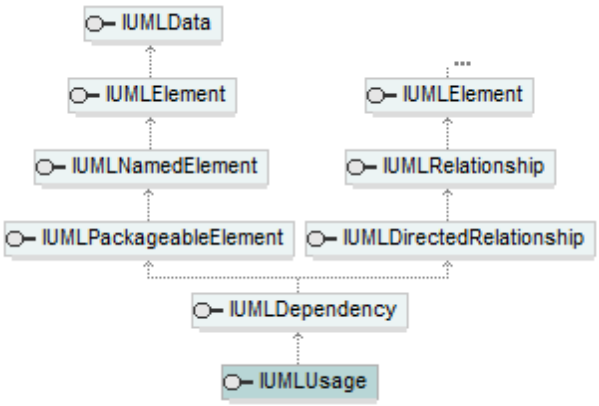
UModelAPI - IUMLUsage

Interface **IUMLUsage**

diagram	

 © 2014 Altova GmbH

hierarchy

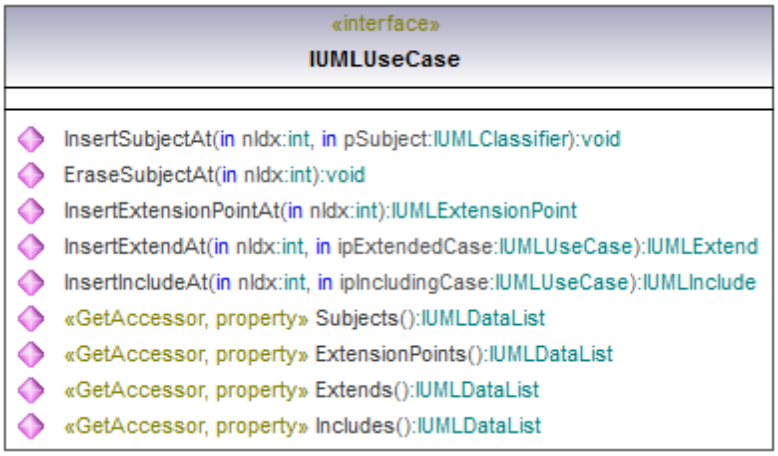


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

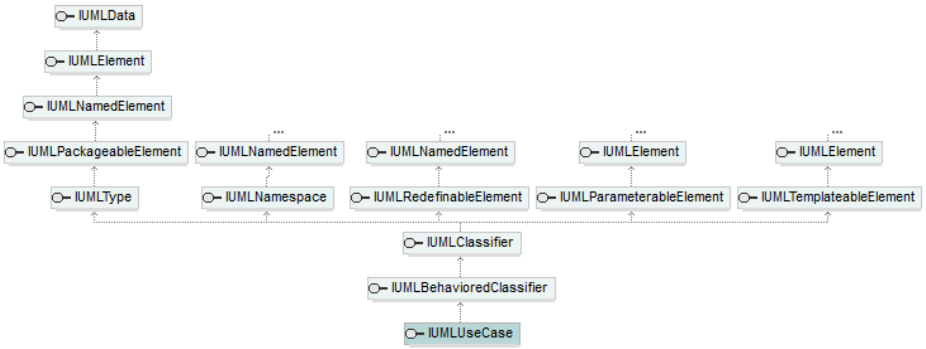
UModelAPI - IUMLUseCase

Interface **IUMLUseCase**

diagram



hierarchy



typedElements

Interface **IUMLClassifier**
Interface **IUMLDataAll**
Operation **InsertOwnedUseCaseAt**
Operation **Addition ExtendedCase**

	Interface **IUMLExtend** Interface **IUMLExtensionPoint** Interface **IUMLInclude** Interface **IUMLUseCase**	Operation **Extension IncludingCase InsertExtendAt InsertIncludeAt InsertOwnedUseCaseAt UseCase** Operation **ExtendedCase Extension** Operation **UseCase** Operation **Addition IncludingCase** Operation **InsertExtendAt InsertIncludeAt**

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**			
documentation	A list of elements of type IUMLExtend.					

Operation **IUMLUseCase::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLExtensionPoint.					

Operation **IUMLUseCase::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLInclude.					

Operation **IUMLUseCase::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipExtendedCase	**in**	**IUMLUseCase**			
	return	**return**	**IUMLExtend**			

Operation **IUMLUseCase::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExtensionPoint**			

Operation **IUMLUseCase::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default

© 2014 Altova GmbH

r	**nIdx**	**in**	**int**
	ipIncludingCase	**in**	**IUMLUseCase**
	return	**return**	**IUMLInclude**

Operation **IUMLUseCase::InsertSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pSubject	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLUseCase::Subjects**

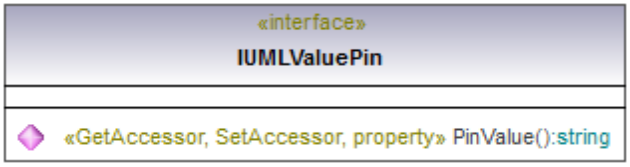
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLClassifier.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

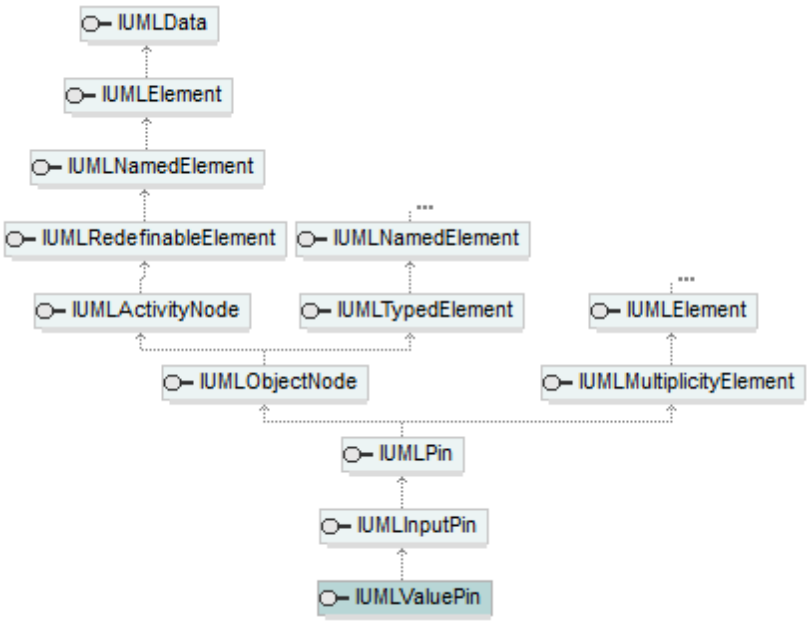
UModelAPI - IUMLValuePin

Interface **IUMLValuePin**

diagram



hierarchy



Operation **IUMLValuePin::PinValue**

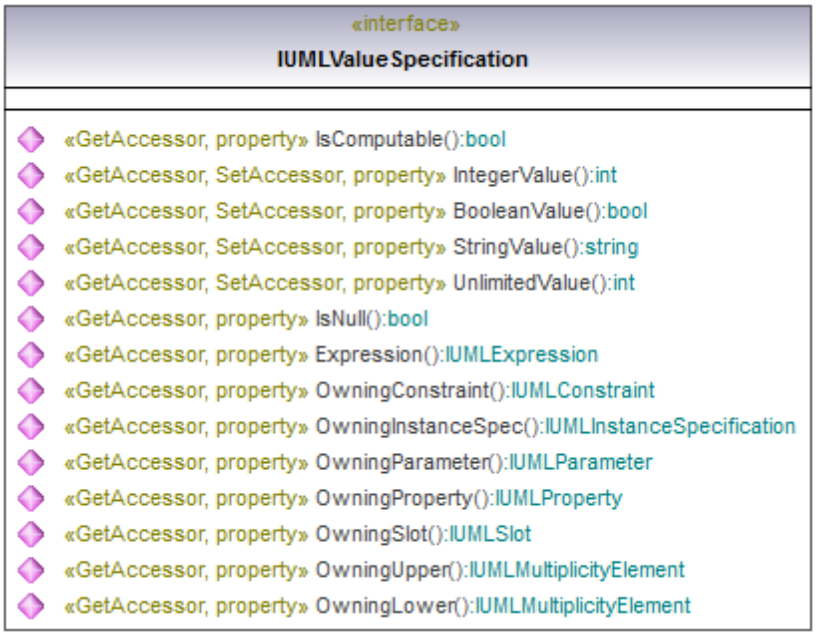
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

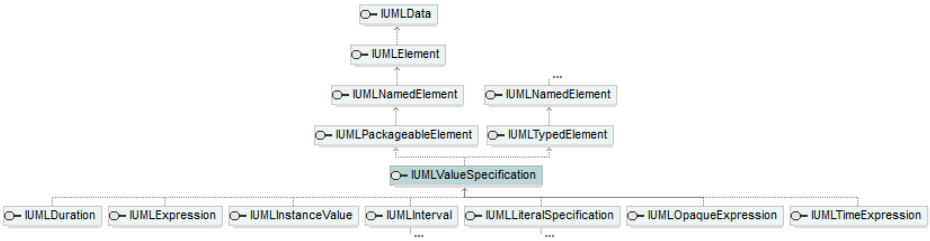
UModelAPI - IUMLValueSpecification

Interface **IUMLValueSpecification**

diagram



hierarchy



typedElements

Interface **IUMLChangeEvent** — Operation **ChangeExpression** **SetNewChangeExpression**

Interface **IUMLConstraint** — Operation **SetNewSpecification** **Specification**

Interface **IUMLDataAll** — Operation **ActionValue** **ChangeExpression** **DefaultValue** **Expr** **InsertOwnedArgumentAt** **InsertValueAt** **Max** **MaxInt** **Min** **MinInt** **Selector** **SetNewActionValue** **SetNewChangeExpression** **SetNewDefaultValue** **SetNewExpr** **SetNewMax** **SetNewMaxInt** **SetNewMin** **SetNewMinInt** **SetNewSelector** **SetNewSpecification** **SetPredefinedTaggedValueAt** **SetSlotValueAt** **SetTaggedValueAt** **Specification**

Interface **IUMLDuration** — Operation **Expr** **SetNewExpr**

	Interface IUMLInstanceSpecification	Operation SetNewSpecification SetSlotValueAt Specification
	Interface IUMLInteractionConstraint	Operation MaxInt MinInt SetNewMaxInt SetNewMinInt
	Interface IUMLInterval	Operation Max Min SetNewMax SetNewMin
	Interface IUMLLifeline	Operation Selector SetNewSelector
	Interface IUMLMessage	Operation InsertOwnedArgumentAt
	Interface IUMLParameter	Operation DefaultValue SetNewDefaultValue
	Interface IUMLProperty	Operation DefaultValue SetNewDefaultValue
	Interface IUMLSlot	Operation InsertValueAt
	Interface IUMLStereotypeApplication	Operation SetPredefinedTaggedValueAt SetTaggedValueAt
	Interface IUMLTimeExpression	Operation Expr SetNewExpr
	Interface IUMLValueSpecificationAction	Operation ActionValue SetNewActionValue

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**			

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**			

Operation **IUMLValueSpecification::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

© 2014 Altova GmbH

Operation **IUMLValueSpecification::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLValueSpecification::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

Operation **IUMLValueSpecification::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLValueSpecification::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSlot**			

Operation **IUMLValueSpecification::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLValueSpecification::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

Operation **IUMLValueSpecification::UnlimitedValue**

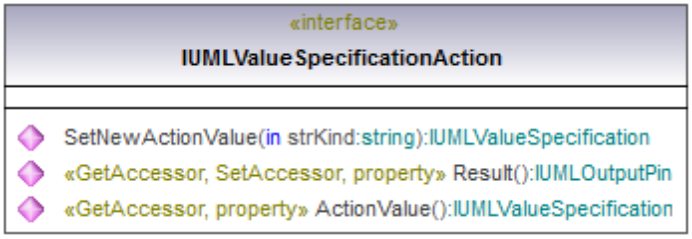
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

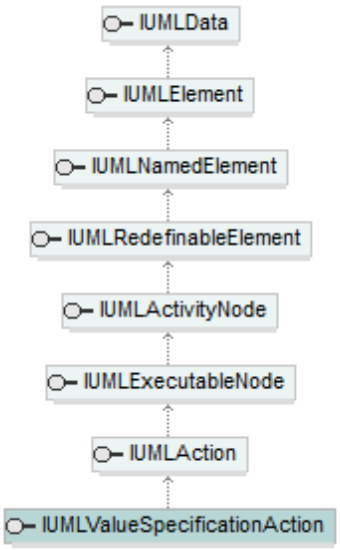
UModelAPI - IUMLValueSpecificationAction

Interface **IUMLValueSpecificationAction**

diagram



hierarchy



Operation **IUMLValueSpecificationAction::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLValueSpecificationAction::Result**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOutputPin**			

Operation **IUMLValueSpecificationAction::SetNewActionValue**

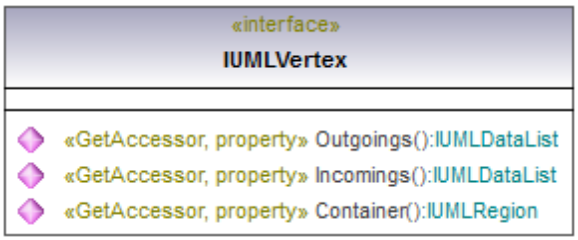
parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

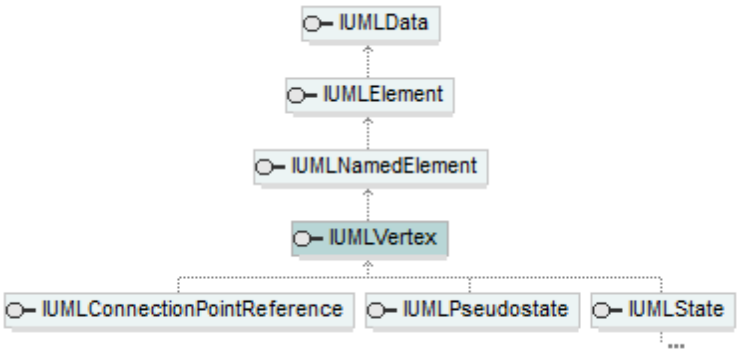
UModelAPI - IUMLVertex

Interface **IUMLVertex**

diagram



 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **InsertSubVertexAt**, **InsertTransitionAt**, **TransitionSource**, **TransitionTarget** Interface **IUMLRegion** — Operation **InsertSubVertexAt**, **InsertTransitionAt** Interface **IUMLTransition** — Operation **TransitionSource**, **TransitionTarget**

Operation **IUMLVertex::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLRegion**			

Operation **IUMLVertex::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTransition.					

Operation **IUMLVertex::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of elements of type IUMLTransition.					

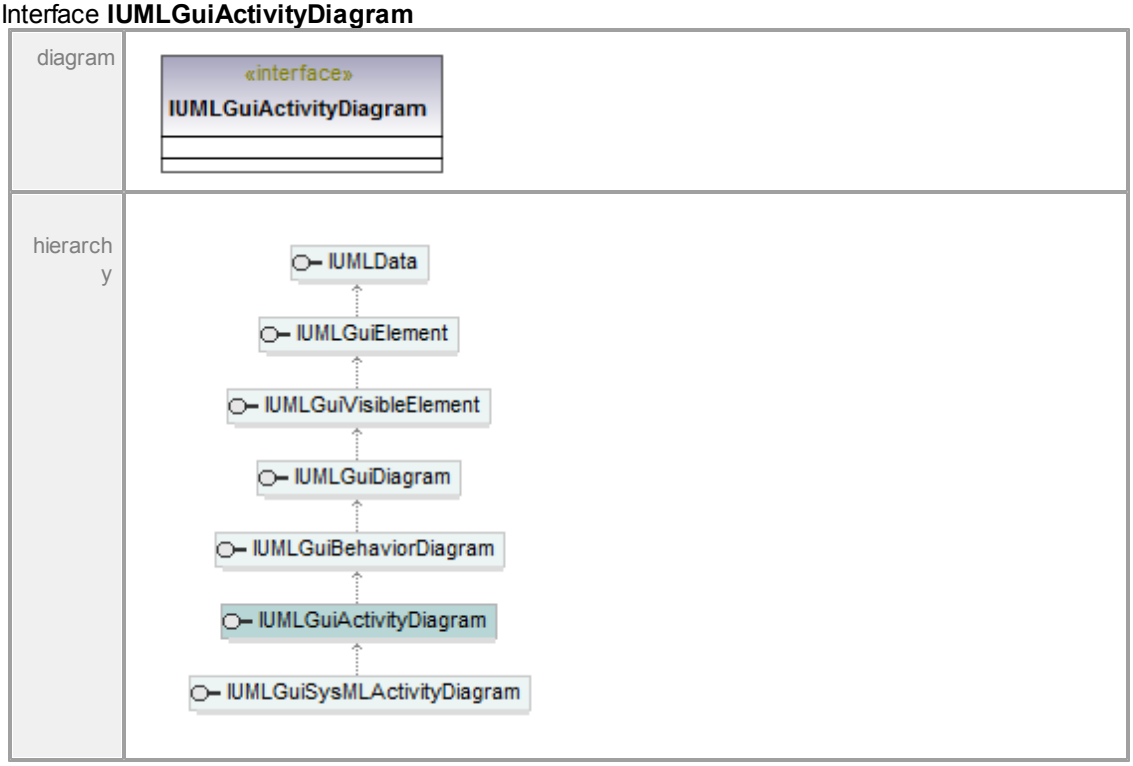
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

IUMLGuiElement

This is a list of elements from Altova for diagrams and to show IUMLElements on diagrams.

UModelAPI - IUMLGuiActivityDiagram

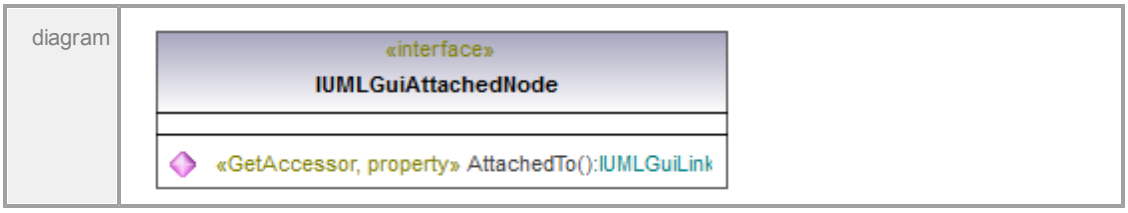
Interface **IUMLGuiActivityDiagram**



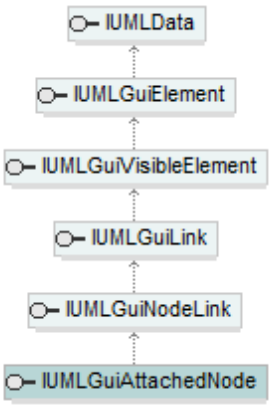
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiAttachedNode

Interface **IUMLGuiAttachedNode**



 © 2014 Altova GmbH

hierarchy	
documentation	This GUI element is a node (possibly without a linked IUMLElement) which is directly attached to a IUMLGuiNodeLink. It disappears and pops up based on data set in the element of the IUMLGuiNodeLink 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.

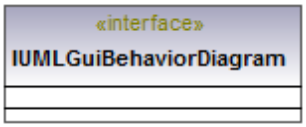
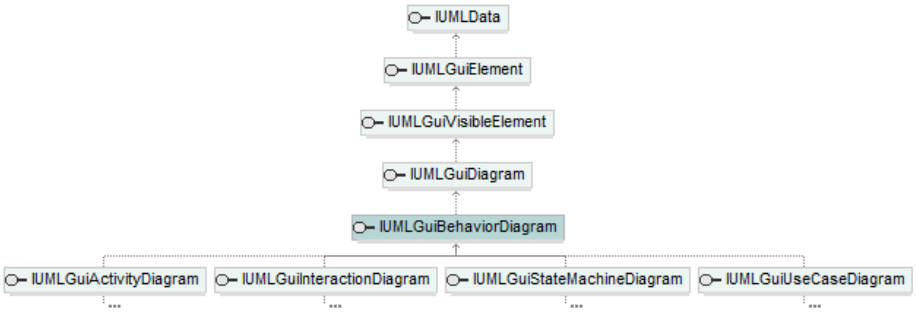
Operation **IUMLGuiAttachedNode::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiBehaviorDiagram

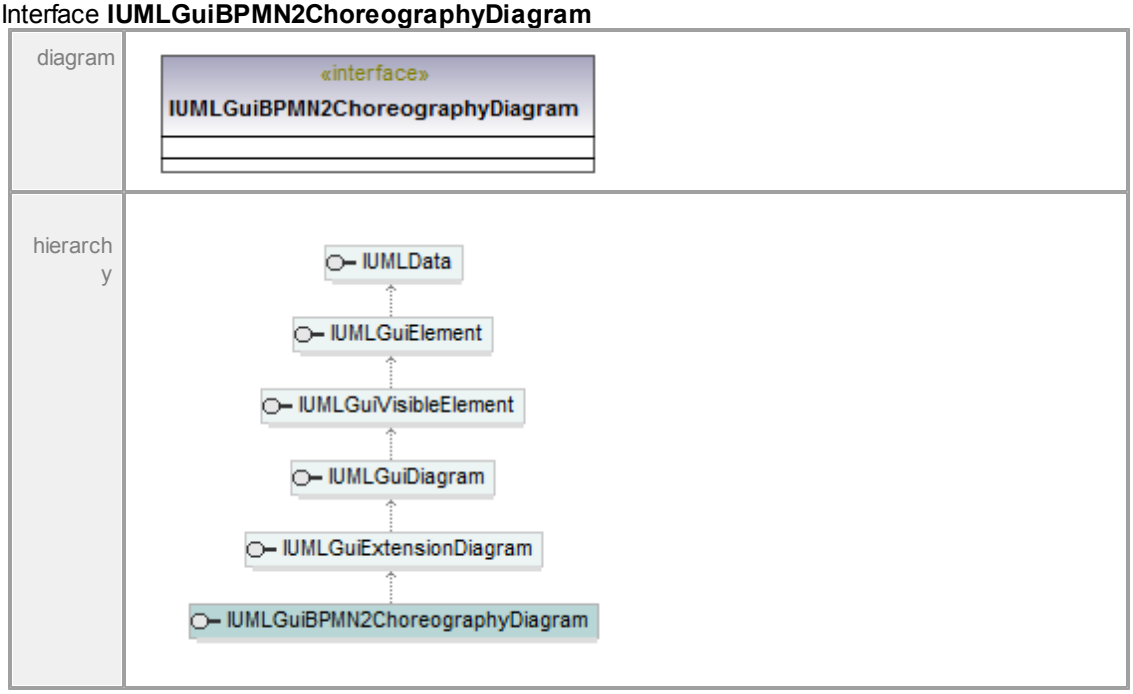
Interface **IUMLGuiBehaviorDiagram**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiBPMN2ChoreographyDiagram

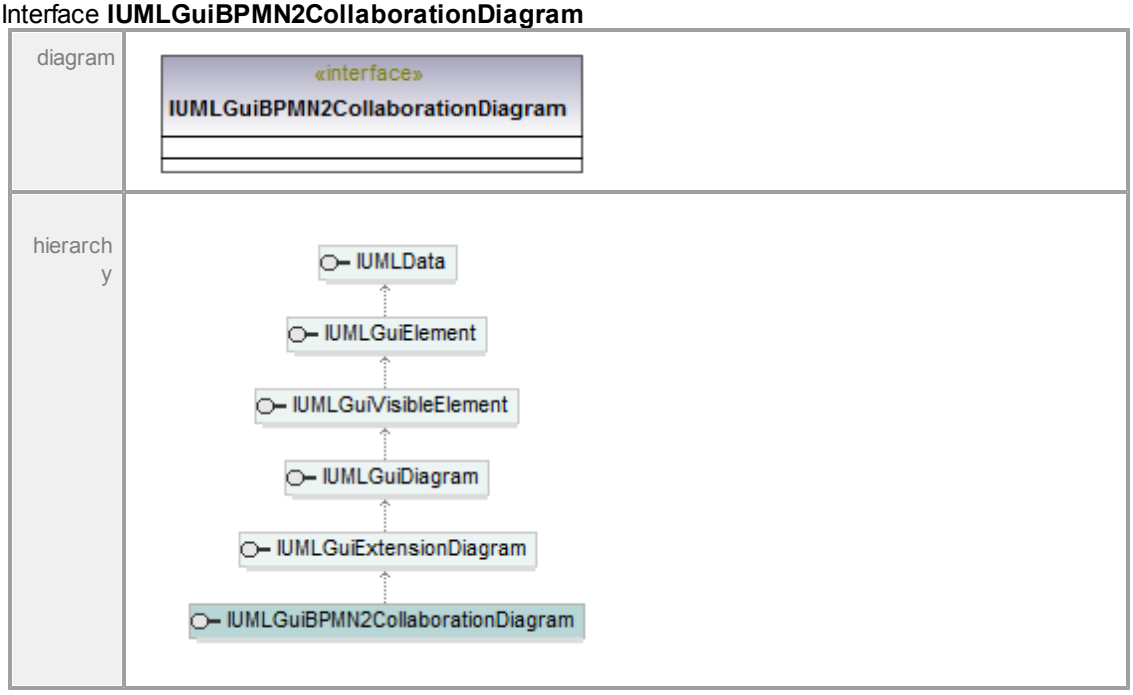
Interface **IUMLGuiBPMN2ChoreographyDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiBPMN2CollaborationDiagram

Interface **IUMLGuiBPMN2CollaborationDiagram**

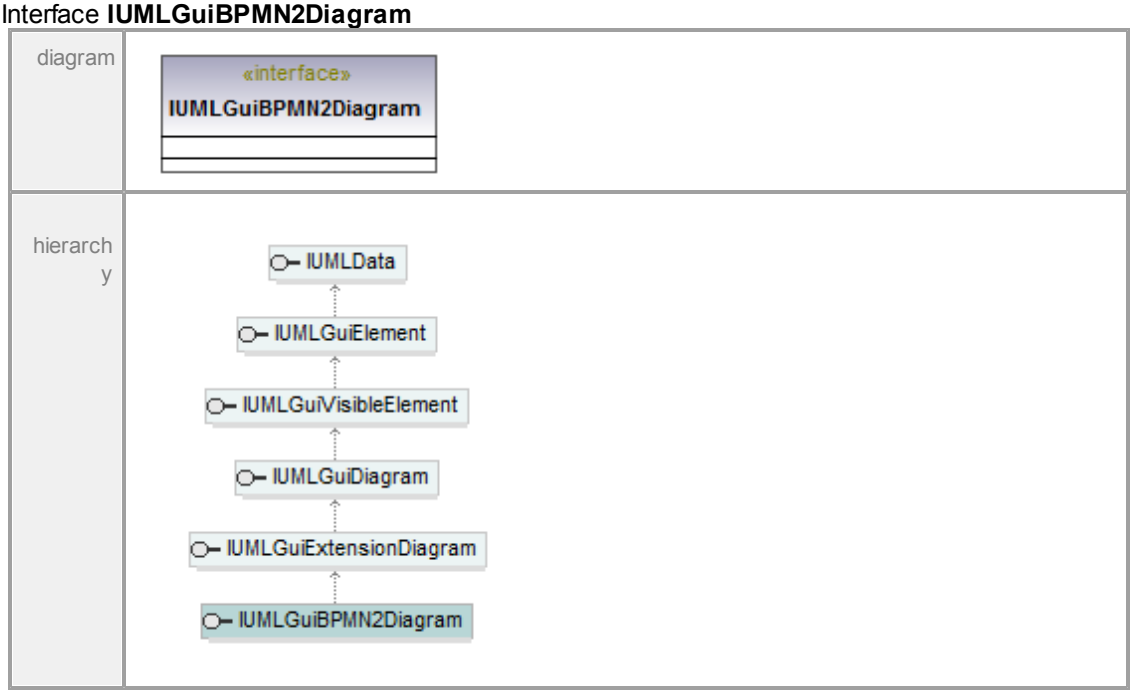


 © 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiBPMN2Diagram

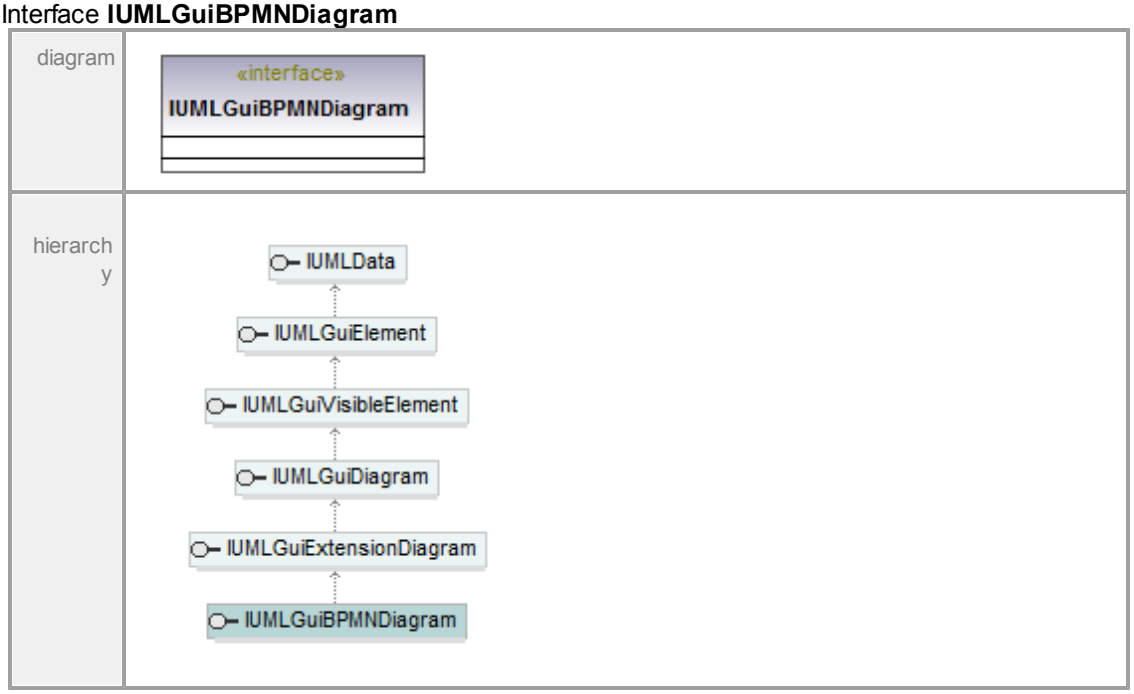
Interface **IUMLGuiBPMN2Diagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiBPMNDiagram

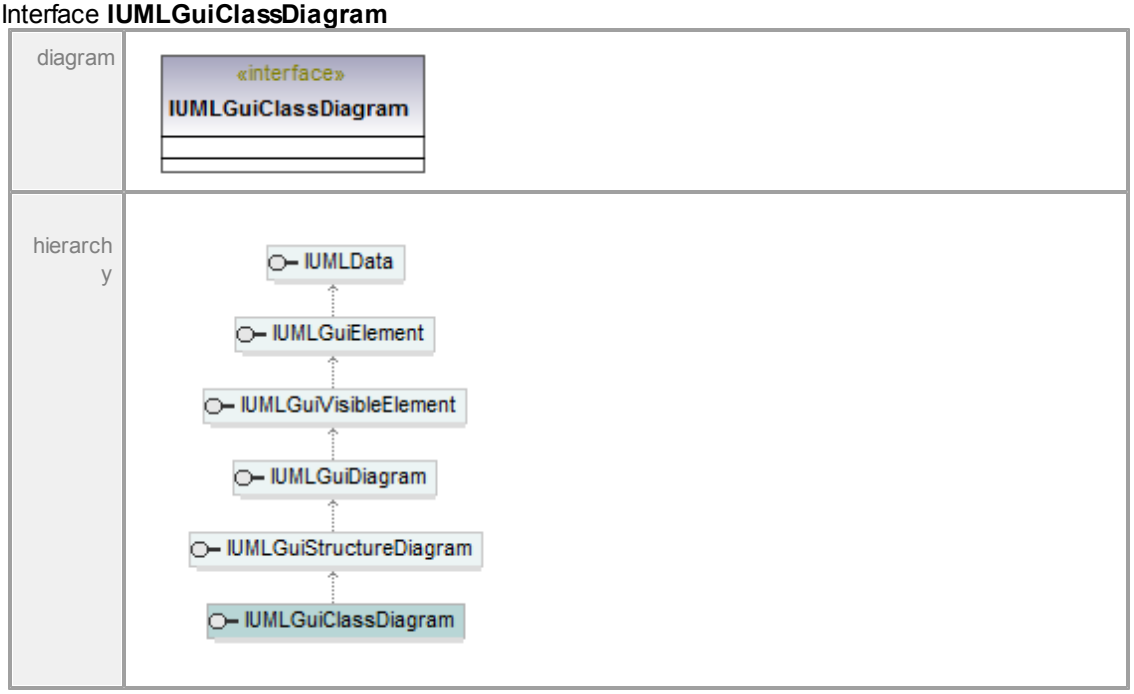
Interface **IUMLGuiBPMNDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiClassDiagram

Interface **IUMLGuiClassDiagram**

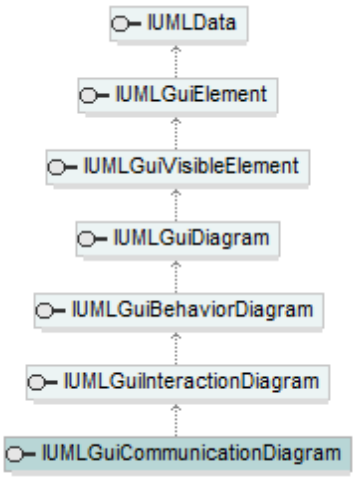


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiCommunicationDiagram

Interface **IUMLGuiCommunicationDiagram**

 © 2014 Altova GmbH

hierarchy	

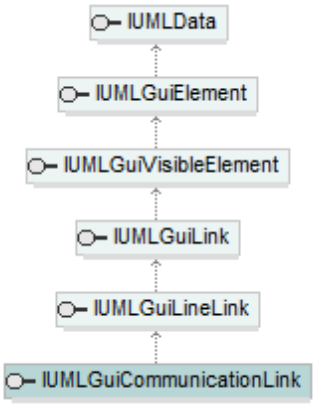
Operation **IUMLGuiCommunicationDiagram::AddUMLGuiCommunicationLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink**			
	ipToLink	**in**	**IUMLGuiLink**			
	return	**return**	**IUMLGuiCommunicationLink**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiCommunicationLink

Interface **IUMLGuiCommunicationLink**

diagram	
hierarchy	

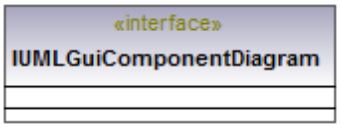
typedElements	Interface **IUMLGuiCommunicationDiagram**	Operation **AddUMLGuiCommunicationLink**
documentation	This line link is used on communication diagrams to provide a connection link for messages between lifelines.	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

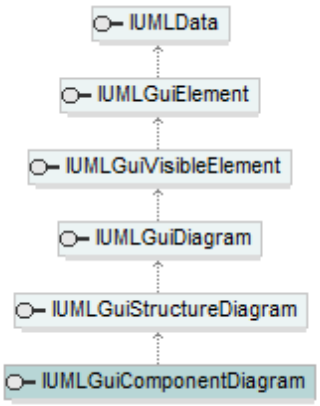
UModelAPI - IUMLGuiComponentDiagram

Interface **IUMLGuiComponentDiagram**

diagram



hierarchy

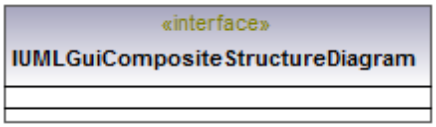


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

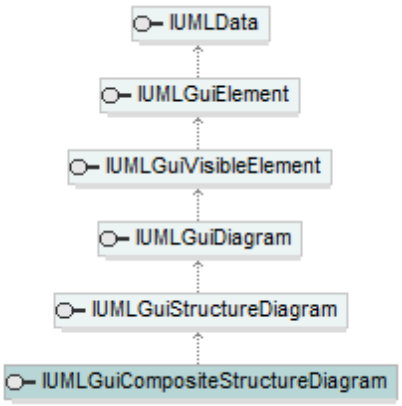
UModelAPI - IUMLGuiCompositeStructureDiagram

Interface **IUMLGuiCompositeStructureDiagram**

diagram



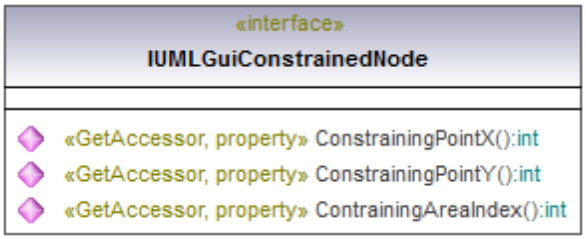
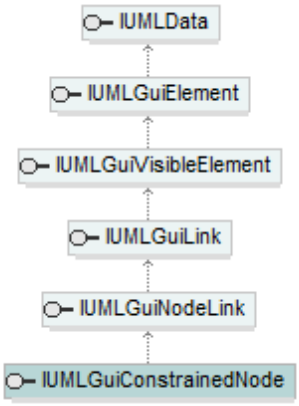
 © 2014 Altova GmbH

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiConstrainedNode

Interface **IUMLGuiConstrainedNode**

diagram	
hierarchy	
documentation	This node link is used to represent objects on diagrams which 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

r	**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.					

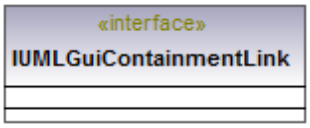
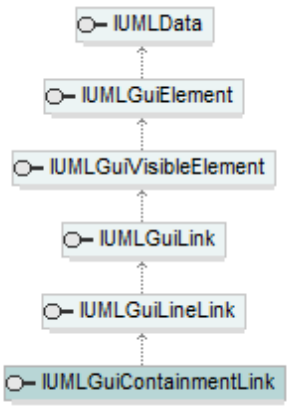
Operation **IUMLGuiConstrainedNode::ContrainingAreaIndex**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			
documentation	Defines the index of the area where this node is currently in and to which its relative position has its origin.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiContainmentLink

Interface **IUMLGuiContainmentLink**

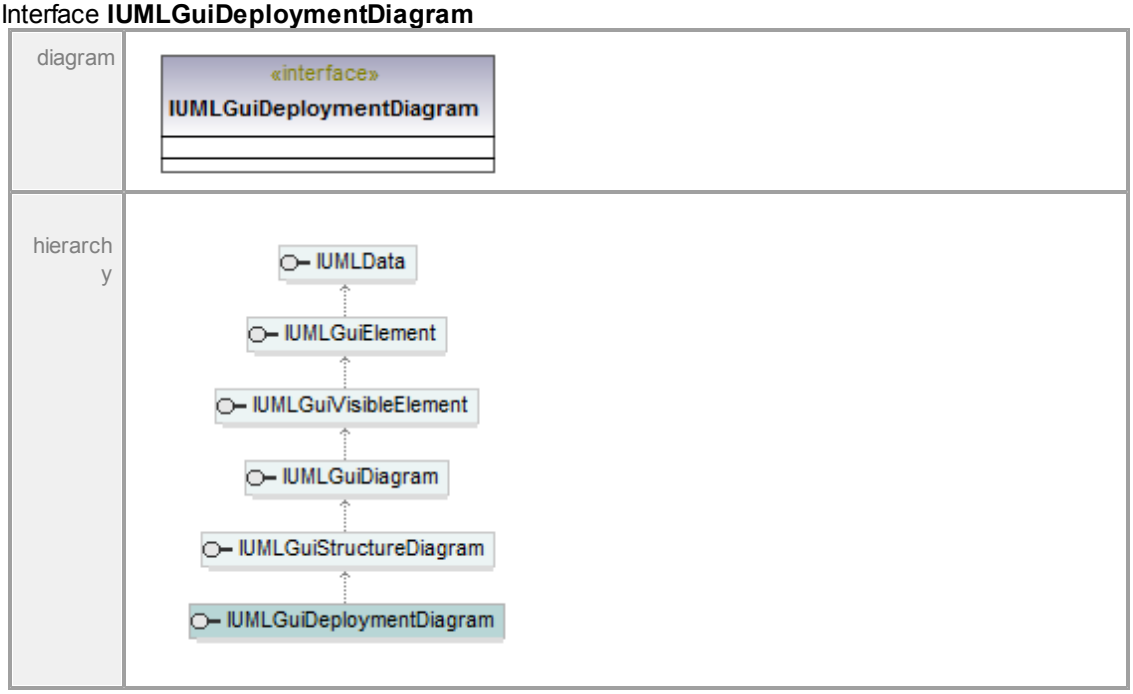
diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Operation **AddUMLGuiContainmentLink** Operation **AddUMLGuiContainmentLink**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLGuiDeploymentDiagram

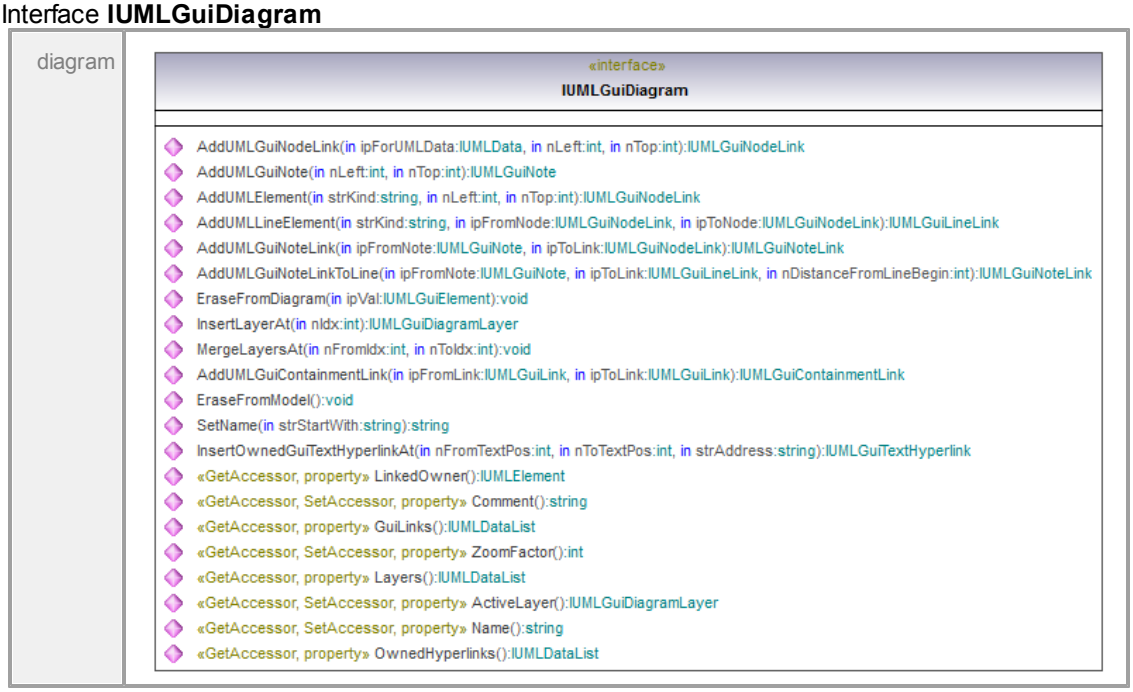
Interface **IUMLGuiDeploymentDiagram**

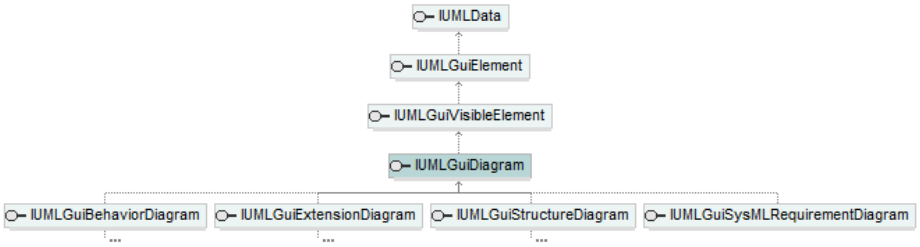


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiDiagram

Interface **IUMLGuiDiagram**



hierarchy	
typedElements	Interface **IDiagramWindow** Interface **IDocument** Interface **IUMLDataAll** Interface **IUMLGuiRootElement** Interface **IUMLGuiSubDiagramNode** Operation **Diagram** Operation **OpenDiagram** Operation **InsertOwnedDiagramAtReferencedDiagram** Operation **InsertOwnedDiagramAt** Operation **ReferencedDiagram**
documentation	Represents an UML diagram and contains all layers, nodes (represented as IUMLGuiNodeLink) and lines (represented as IUMLGuiLineLink).Use the property GuiLinks to access these.

Operation **IUMLGuiDiagram::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLGuiDiagram::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			
documentation	Adds a new UML element (e.g. IUMLClass, IUMLPackage,...) to the model and shows it with a new IUMLGuiNodeLink on the diagram.					

Operation **IUMLGuiDiagram::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink**			
	ipToLink	**in**	**IUMLGuiLink**			
	return	**return**	**IUMLGuiContainmentLink**			

Operation **IUMLGuiDiagram::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLData**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			
documentation	Adds a new IUMLGuiNodeLink for an existing UML element (e.g. IUMLClass, IUMLPackage,...) on the diagram.					

 © 2014 Altova GmbH

Operation **IUMLGuiDiagram::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNote**			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiLineLink**			
	nDistanceFromLineBegin	**in**	**int**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLGuiDiagram::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	ipFromNode	**in**	**IUMLGuiNodeLink**			
	ipToNode	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiLineLink**			
documentation	Adds a new UML line element (e.g. IUMLGeneralization, IUMLAssociation,...) to the model and shows it with a new IUMLGuiLineLink 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**			
	return	**return**	**void**			
documentation	Use this function to erase the element from the diagram only. Use IUMLElement::EraseFromModel 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**			
documentation	A list of elements of type IUMLGuiLink which are displayed directly on this diagram. Usually, these are IUMLGuiNodeLinks and IUMLGuiLineLinks.					

Operation **IUMLGuiDiagram::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLGuiDiagram::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLGuiTextHyperlink**			

Operation **IUMLGuiDiagram::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	A list of all layers in the diagam. The list contains elements of type IUMLGuiDiagramLayer.					

Operation **IUMLGuiDiagram::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

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**			

© 2014 Altova GmbH

Operation **IUMLGuiDiagram::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

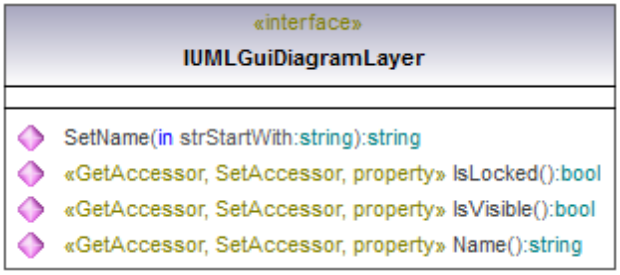
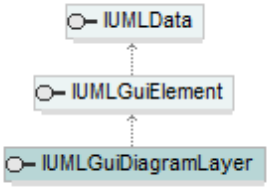
Operation **IUMLGuiDiagram::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiDiagramLayer

Interface **IUMLGuiDiagramLayer**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Interface **IUMLGuiLink** Operation **ActiveLayer** **InsertLayerAtLayer** Operation **ActiveLayer** **InsertLayerAtLayer**
documentation	Represents a layer on an IUMLGuiDiagram. 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**			

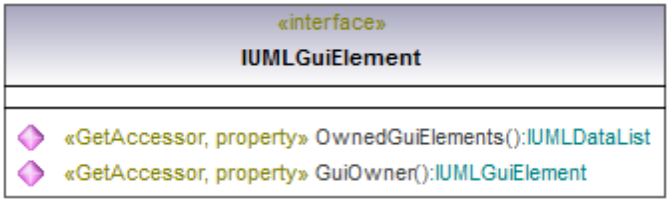
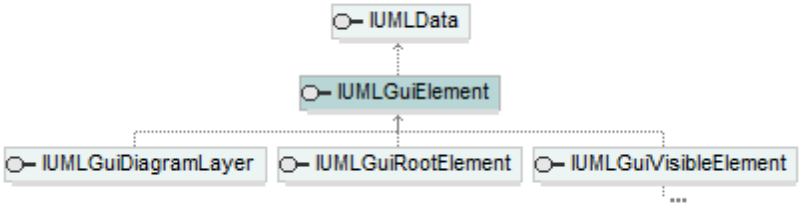
Operation **IUMLGuiDiagramLayer::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith	**in**	**string**			
	return	**return**	**string**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiElement

Interface **IUMLGuiElement**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Interface **IUMLGuiDiagram** Interface **IUMLGuiElement** Interface **IUMLGuiLineLink** Operation **EraseFromDiagram** **GuiOwner** **LineBegin** **LineEnd** Operation **EraseFromDiagram** Operation **GuiOwner** Operation **LineBegin** **LineEnd**
documentation	The base class for all graphical objects.

Operation **IUMLGuiElement::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLGuiElement::OwnedGuiElements**

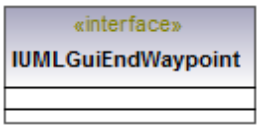
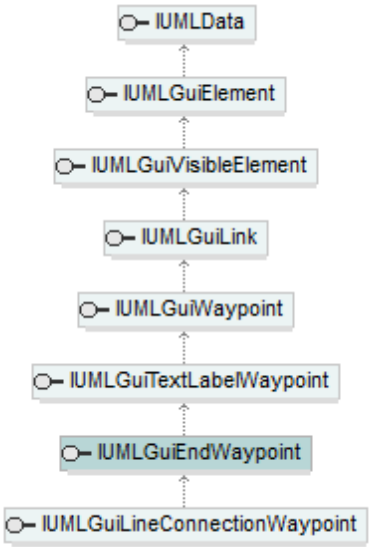
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a derived list of all owned Gui elements. All elements in this list are a subtype if IUMLGuiElement.					

 © 2014 Altova GmbH

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiEndWaypoint

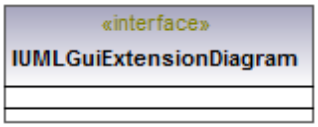
Interface **IUMLGuiEndWaypoint**

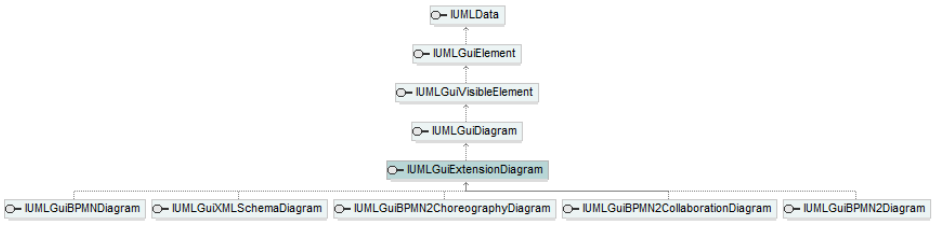
diagram	
hierarchy	
documentation	A special waypoint which only occurrs at the end or the begin of a line represented by a IUMLGuiLineLink.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiExtensionDiagram

Interface **IUMLGuiExtensionDiagram**

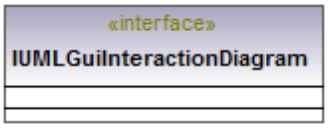
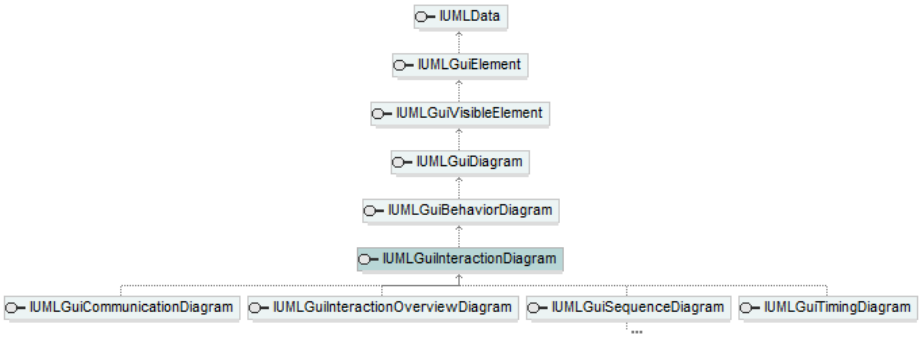
diagram	

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.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiInteractionDiagram

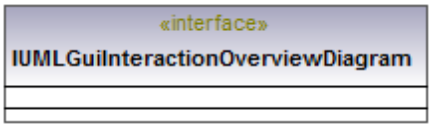
Interface **IUMLGuiInteractionDiagram**

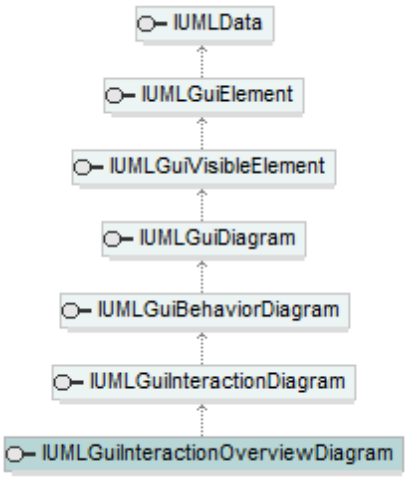
diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiInteractionOverviewDiagram

Interface **IUMLGuiInteractionOverviewDiagram**

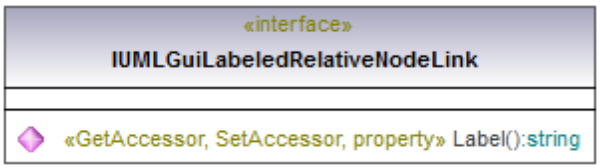
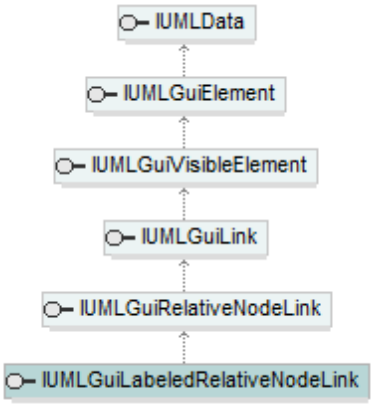
diagram	

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiLabeledRelativeNodeLink

Interface **IUMLGuiLabeledRelativeNodeLink**

diagram	
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**

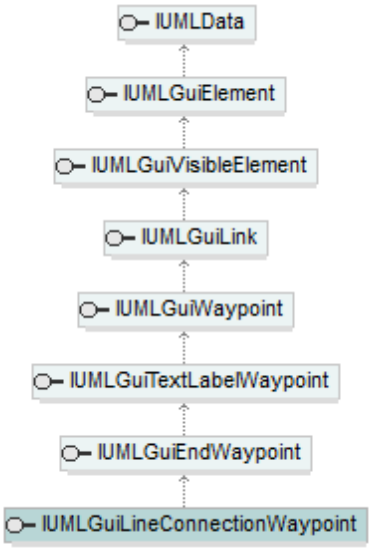
parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**string**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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 property, the waypoint sets a floating fixed position for itself on the line.

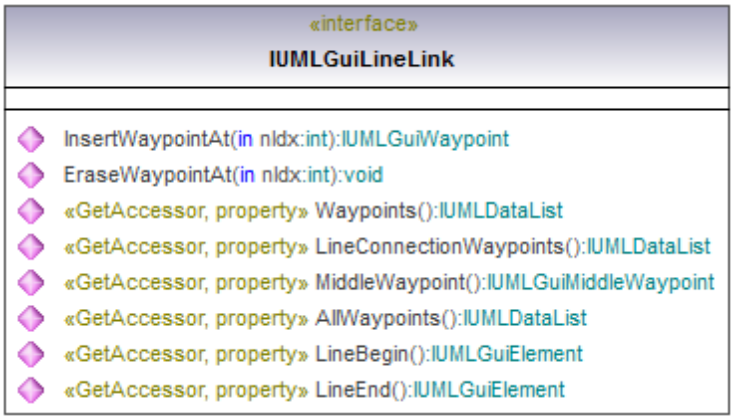
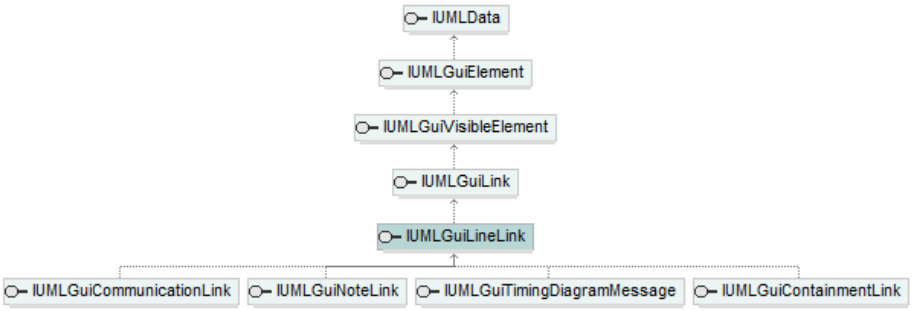
Operation **IUMLGuiLineConnectionWaypoint::DistanceFromLineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiLineLink

Interface **IUMLGuiLineLink**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **AddUMLGuiNoteLinkToLine** **AddUMLLineElement** Interface **IUMLGuiDiagram** — Operation **AddUMLGuiNoteLinkToLine** **AddUMLLineElement**
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 property. Two of these waypoints are usually of type IUMLGuiEndWaypoint. There may be also a Middlewaypoint accessible with the property MiddleWaypoint which can be used for example to access textlabels and a LineConnectionWaypoint when the line is connected to another line, like to a IUMLGuiNoteLink. The LineBegin property refers the first object and LineEnd property refers the second graphical object which the line connects.

Operation **IUMLGuiLineLink::AllWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
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.					

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**			

Operation **IUMLGuiLineLink::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			
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**			
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.					

Operation **IUMLGuiLineLink::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			
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**			

Operation **IUMLGuiLineLink::Waypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
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.					

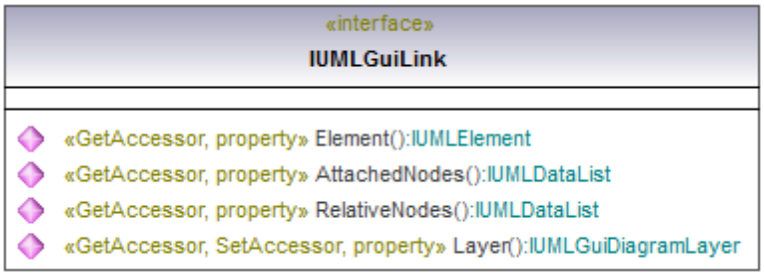
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

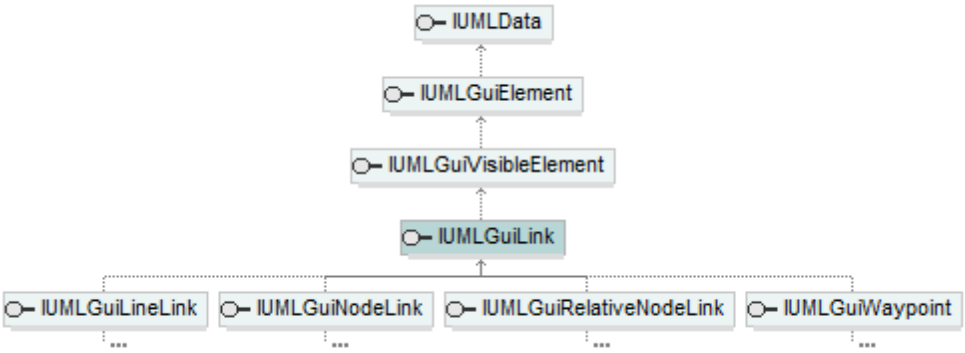
UModelAPI - IUMLGuiLink

Interface **IUMLGuiLink**

diagram



hierarchy



typedElements

Interface **IDiagramWindow**	Operation **FocusedGuiElement** **ScrollToGuiElement** **SelectGuiElement**
Interface **IUMLDataAll**	Operation **AddUMLGuiContainmentLink** **AttachedTo**
Interface **IUMLGuiAttachedNode**	Operation **AttachedTo**
Interface **IUMLGuiCommunicationDiagram**	Operation **AddUMLGuiCommunicationLink**
Interface **IUMLGuiDiagram**	Operation **AddUMLGuiContainmentLink**

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 property. IUMLGuiLinks further can have attached nodes (IUMLGuiAttachedNode) which appear when necessary, like Tagged Values and are associated with a Layer on the diagram.

Operation **IUMLGuiLink::AttachedNodes**

paramete	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**IUMLDataList**
documentation	Returns a derived list of all attached nodes of this element. All elements in this list are of type (or subtype) of IUMLGuiAttachedNode.		

Operation **IUMLGuiLink::Element**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLGuiLink::Layer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer**			

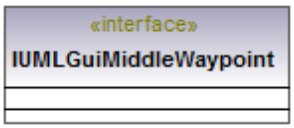
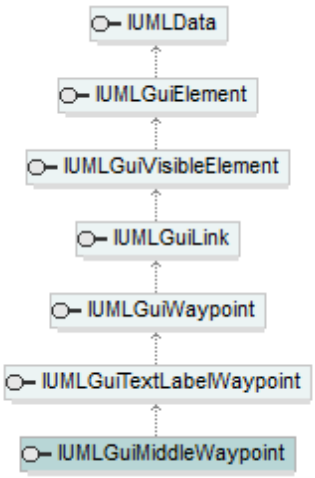
Operation **IUMLGuiLink::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a list of relative nodes to this gui link. The list contains only elements of type (or subtype of) IUMLGuiRelativeNodeLink.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiMiddleWaypoint

Interface **IUMLGuiMiddleWaypoint**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** Operation **MiddleWaypoint** Interface **IUMLGuiLineLink** Operation **MiddleWaypoint**

 © 2014 Altova GmbH

documentation	A middle waypoint is a special waypoint on a line (IUMLGuiLineLink) 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** 09/04/12 14:02:42

UModelAPI - IUMLGuiNodeLink

Interface **IUMLGuiNodeLink**

diagram	«interface» **IUMLGuiNodeLink** SetRect(in nLeft:int, in nTop:int, in nRight:int, in nBottom:int):void MoveTo(in nLeft:int, in nTop:int):void IsElementVisible(in ipElement:IUMLElement):bool SetElementVisible(in ipElement:IUMLElement, in bVisible:bool):void «GetAccessor, SetAccessor, property» Top():int «GetAccessor, SetAccessor, property» Left():int «GetAccessor, SetAccessor, property» Bottom():int «GetAccessor, SetAccessor, property» Right():int «GetAccessor, property» OwnedGuiNodeLinks():IUMLDataList «GetAccessor, property» OwningGuiNodeLink():IUMLGuiNodeLink
hierarchy	IUMLData IUMLGuiElement IUMLGuiVisibleElement IUMLGuiLink IUMLGuiNodeLink IUMLGuiAttachedNode, IUMLGuiConstrainedNode, IUMLGuiNote, IUMLGuiSeparatedNodeLink, IUMLGuiSubDiagramNode, IUMLGuiTickMark, IUMLGuiTimingDiagramLifeline ...
typedElements	Interface **IUMLDataAll** — Operation **AddUMLElement**, **AddUMLGuiNodeLink**, **AddUMLGuiNoteLink**, **AddUMLLineElement**, **OwningGuiNodeLink** Interface **IUMLGuiDiagram** — Operation **AddUMLElement**, **AddUMLGuiNodeLink**, **AddUMLGuiNoteLink**, **AddUMLLineElement** Interface **IUMLGuiNodeLink** — Operation **OwningGuiNodeLink**
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 and IsElementVisible functions for this.

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**			
	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**			
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.					

Operation **IUMLGuiNodeLink::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiNodeLink**			

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**			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

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**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiNote

Interface **IUMLGuiNote**

diagram	«interface» **IUMLGuiNote** InsertOwnedGuiTextHyperlinkAt(in nFromTextPos:int, in nToTextPos:int, in strAddress:string):IUMLGuiTextHyperlink «GetAccessor, SetAccessor, property» NoteText():string «GetAccessor, property» OwnedHyperlinks():IUMLDataList
hierarchy	IUMLData IUMLGuiElement IUMLGuiVisibleElement IUMLGuiLink IUMLGuiNodeLink IUMLGuiNote
typedElements	Interface **IUMLDataAll** — Operation **AddUMLGuiNote**, **AddUMLGuiNoteLink**, **AddUMLGuiNoteLinkToLine** Interface **IUMLGuiDiagram** — Operation **AddUMLGuiNote**, **AddUMLGuiNoteLink**, **AddUMLGuiNoteLinkToLine**
documentation	A IUMLGuiNote 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**			

Operation **IUMLGuiNote::NoteText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

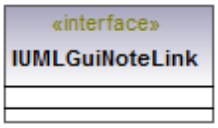
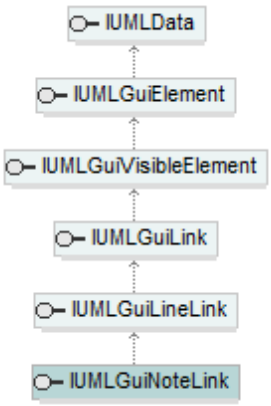
Operation **IUMLGuiNote::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiNoteLink

Interface **IUMLGuiNoteLink**

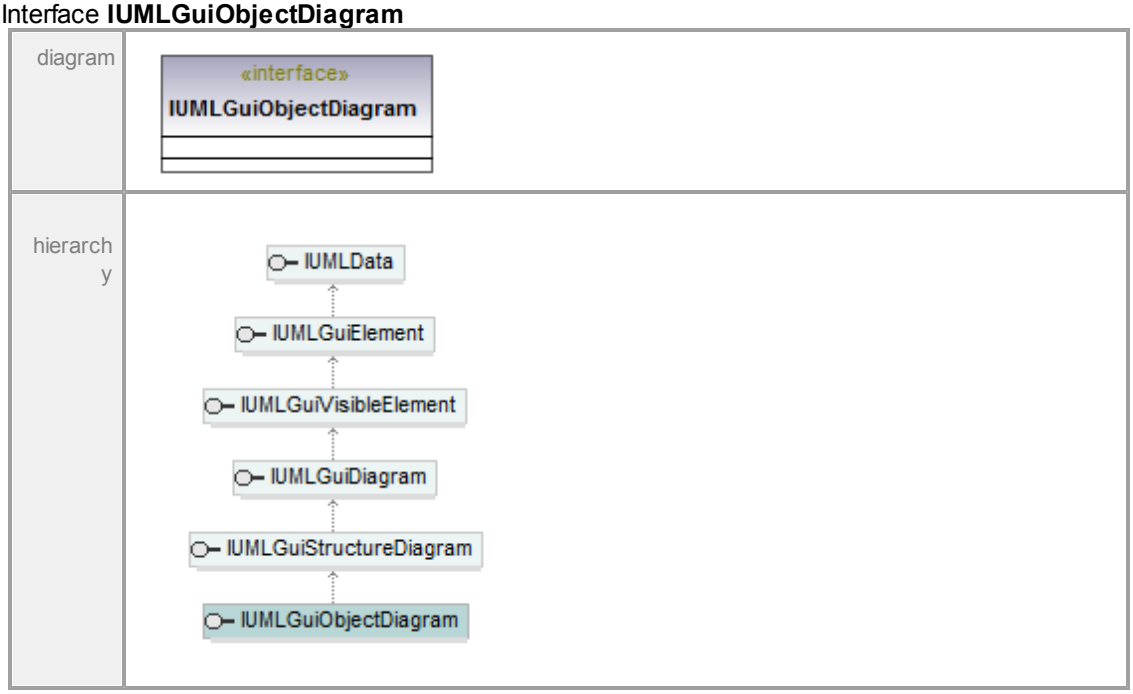
diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **AddUMLGuiNoteLink**, **AddUMLGuiNoteLinkToLine** Interface **IUMLGuiDiagram** — Operation **AddUMLGuiNoteLink**, **AddUMLGuiNoteLinkToLine**
documentation	A notelink is a special IUMLGuiLineLink which connects a IUMLGuiNote with another IUMLGuiLink element. It is displayed as a dotted line.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLGuiObjectDiagram

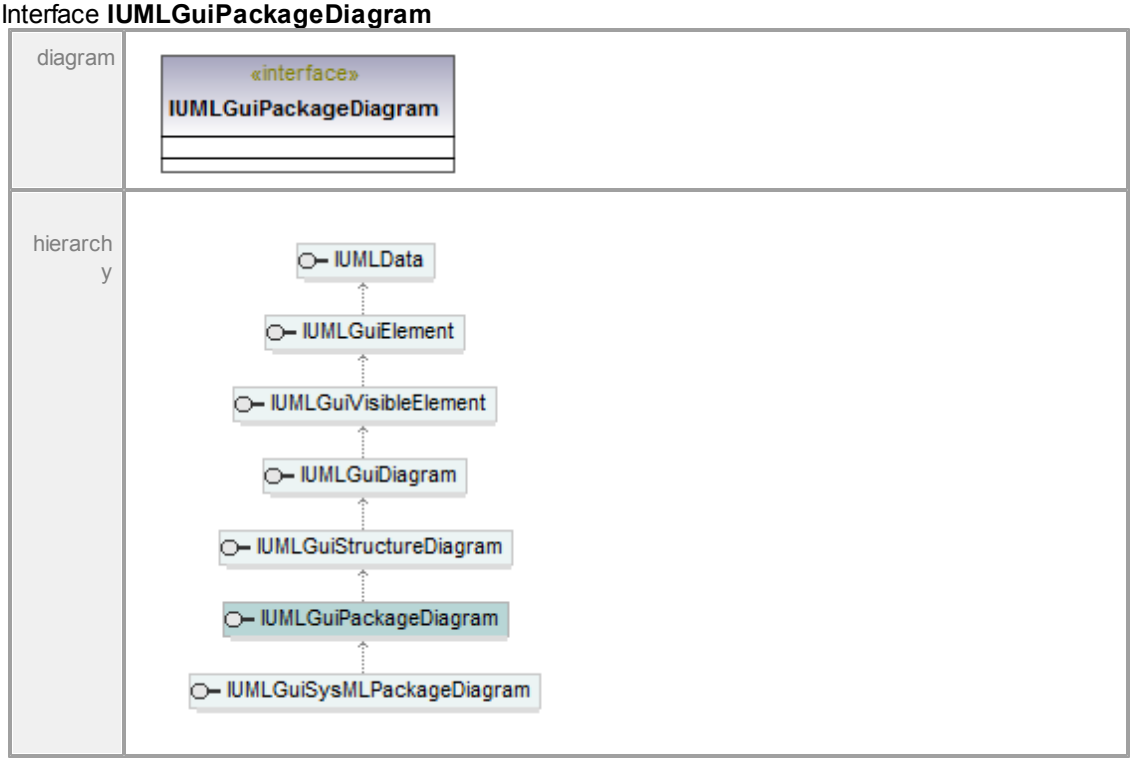
Interface **IUMLGuiObjectDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiPackageDiagram

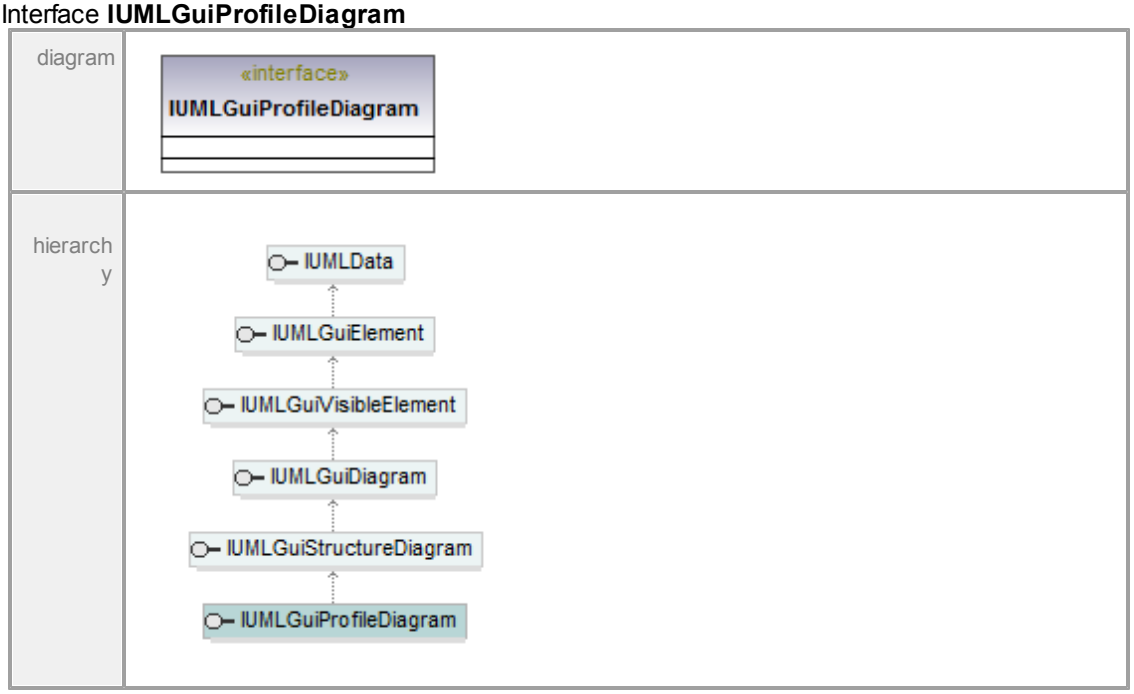
Interface **IUMLGuiPackageDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiProfileDiagram

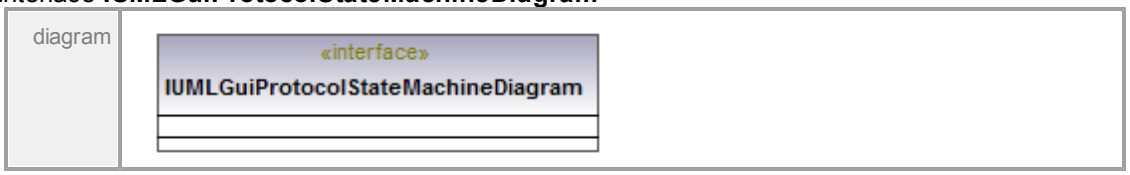
Interface **IUMLGuiProfileDiagram**



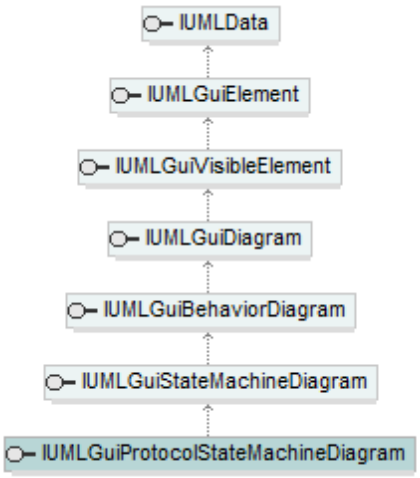
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiProtocolStateMachineDiagram

Interface **IUMLGuiProtocolStateMachineDiagram**



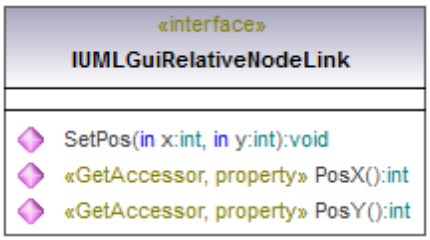
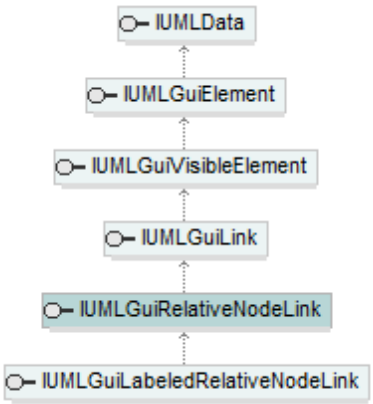
 © 2014 Altova GmbH

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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**			

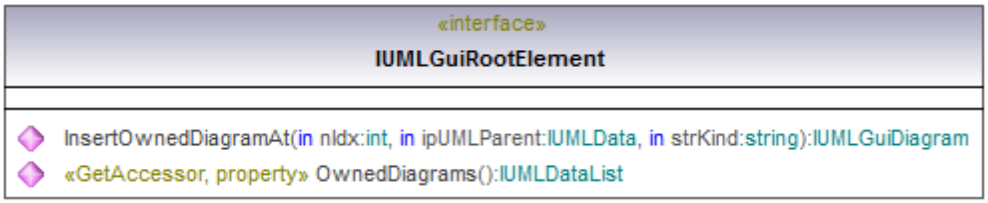
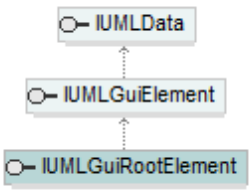
Operation **IUMLGuiRelativeNodeLink::SetPos**

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** 09/04/12 14:02:42

UModelAPI - IUMLGuiRootElement

Interface **IUMLGuiRootElement**

diagram	
hierarchy	
typedElements	Interface **IDocument** Operation **GuiRoot**
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**			
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiDiagram**			

 © 2014 Altova GmbH

Operation **IUMLGuiRootElement::OwnedDiagrams**

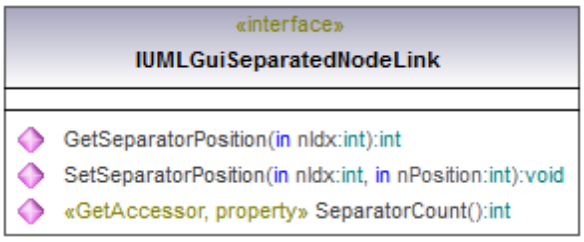
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			
documentation	Returns a list of all diagrams in this UModel project. All elements in this list are of type (or subtype of) IUMLGuiDiagram.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

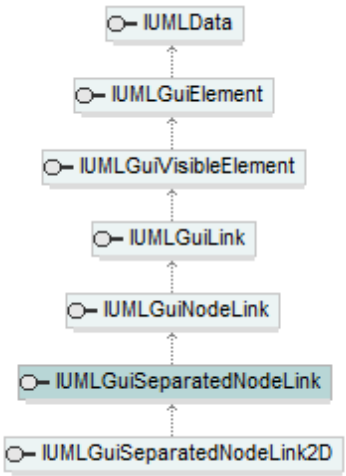
UModelAPI - IUMLGuiSeparatedNodeLink

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**

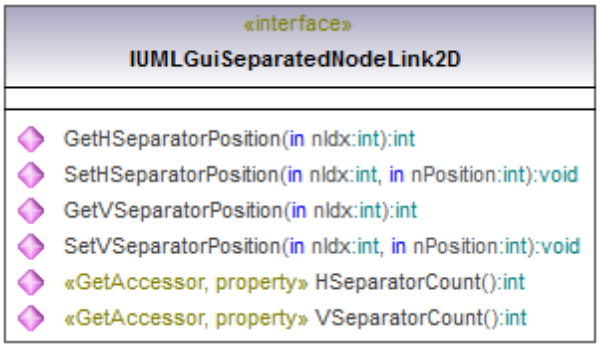
parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	nPosition	**in**	**int**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

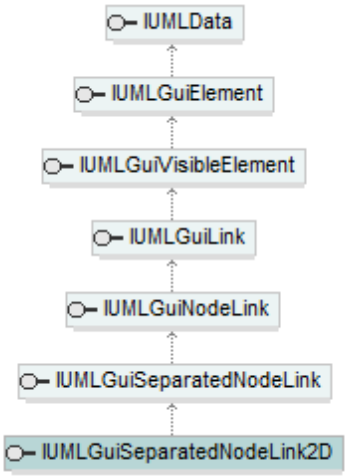
UModelAPI - IUMLGuiSeparatedNodeLink2D

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, 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**

 © 2014 Altova GmbH

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**

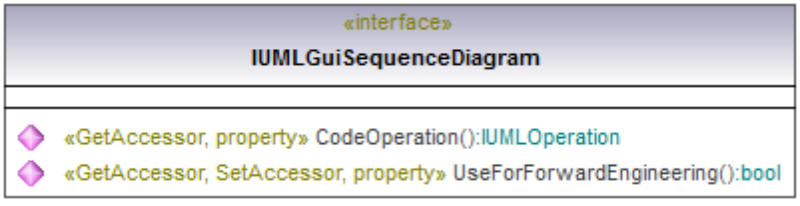
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

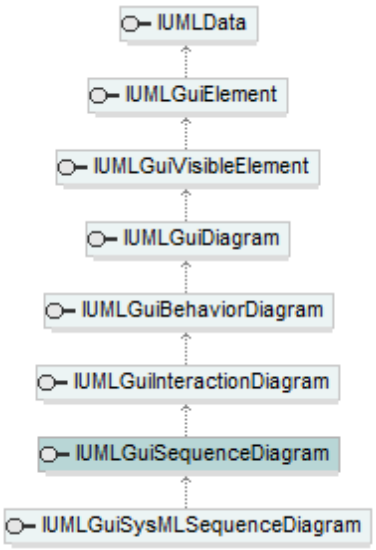
UModelAPI - IUMLGuiSequenceDiagram

Interface **IUMLGuiSequenceDiagram**

diagram



hierarchy



Operation **IUMLGuiSequenceDiagram::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLGuiSequenceDiagram::UseForForwardEngineering**

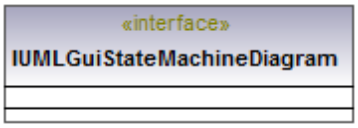
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			

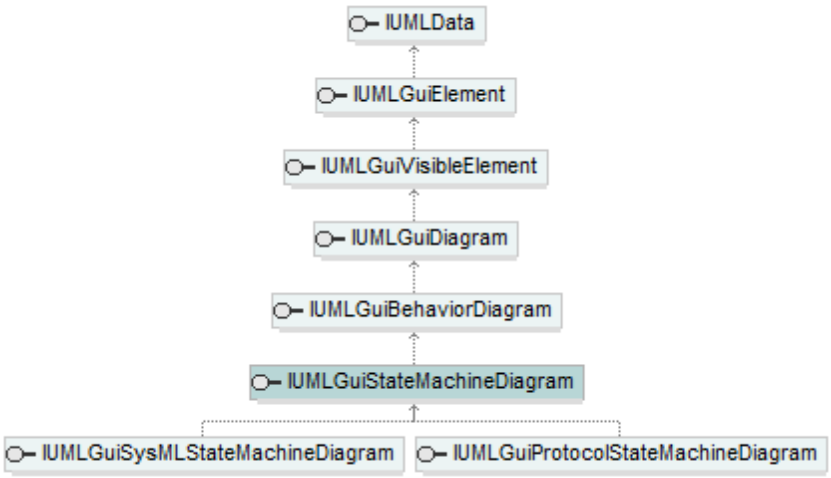
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiStateMachineDiagram

Interface **IUMLGuiStateMachineDiagram**

diagram

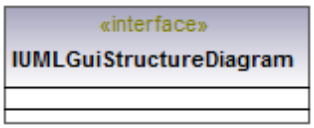
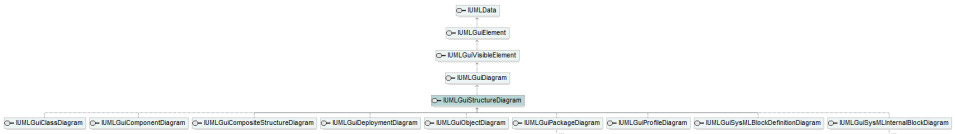


hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiStructureDiagram

Interface **IUMLGuiStructureDiagram**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiStyle

Interface **IUMLGuiStyle**

diagram	«interface» IUMLGuiStyle «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, property» Kind():ENUMUMLGuiStyleKind «GetAccessor, property» Name():string «GetAccessor, SetAccessor, property» Value():string «GetAccessor, property» UsedValue():string
typedElements	Interface **IUMLGuiStyles** Operation **GetStyle** **Item**

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**			

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**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - IUMLGuiStyles

Interface **IUMLGuiStyles**

diagram	«interface» **IUMLGuiStyles** GetStyle(in nKind:ENUMUMLGuiStyleKind):IUMLGuiStyle GetName(in nKind:ENUMUMLGuiStyleKind):string GetValue(in nKind:ENUMUMLGuiStyleKind):string SetValue(in nKind:ENUMUMLGuiStyleKind, in strNewVal:string):void GetUsedValue(in nKind:ENUMUMLGuiStyleKind):string «GetAccessor, property» Count():int Item(in nIdx:int):IUMLGuiStyle «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch
typedElements	Interface **IDocument** Interface **IUMLDataAll** Interface **IUMLGuiVisibleElement** Interface **IUMLStereotype** Operation **ElementFamilyStyles** **LineStyles** **NodeStyles** **ProjectStyles** Operation **StereotypedElementStyles** **Styles** Operation **Styles** Operation **StereotypedElementStyles**

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**			
	return	**return**	**string**			

Operation **IUMLGuiStyles::GetStyle**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	return	**return**	**IUMLGuiStyle**			

Operation **IUMLGuiStyles::GetUsedValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			

	return	**return**	**string**

Operation **IUMLGuiStyles::GetValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	return	**return**	**string**			

Operation **IUMLGuiStyles::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiStyle**			

Operation **IUMLGuiStyles::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IDispatch**			

Operation **IUMLGuiStyles::SetValue**

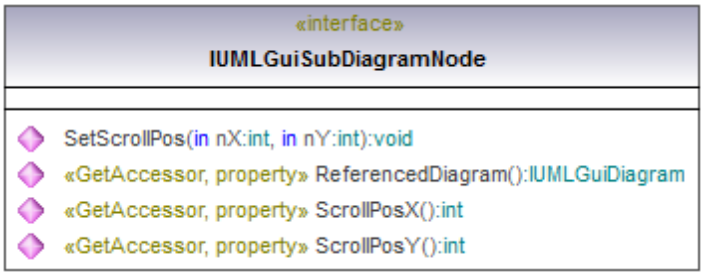
parameter	name	direction	type	type modifier	multiplicity	default
	nKind	**in**	**ENUMUMLGuiStyleKind**			
	strNewVal	**in**	**string**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

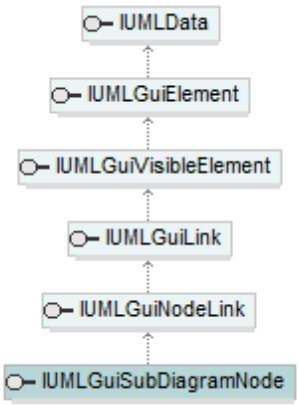
UModelAPI - IUMLGuiSubDiagramNode

Interface **IUMLGuiSubDiagramNode**

diagram



 © 2014 Altova GmbH

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 controls the diagrams which is shown inside the node.

Operation **IUMLGuiSubDiagramNode::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagram**			

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**			

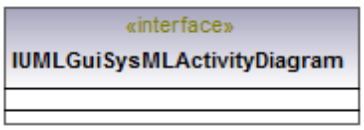
Operation **IUMLGuiSubDiagramNode::SetScrollPos**

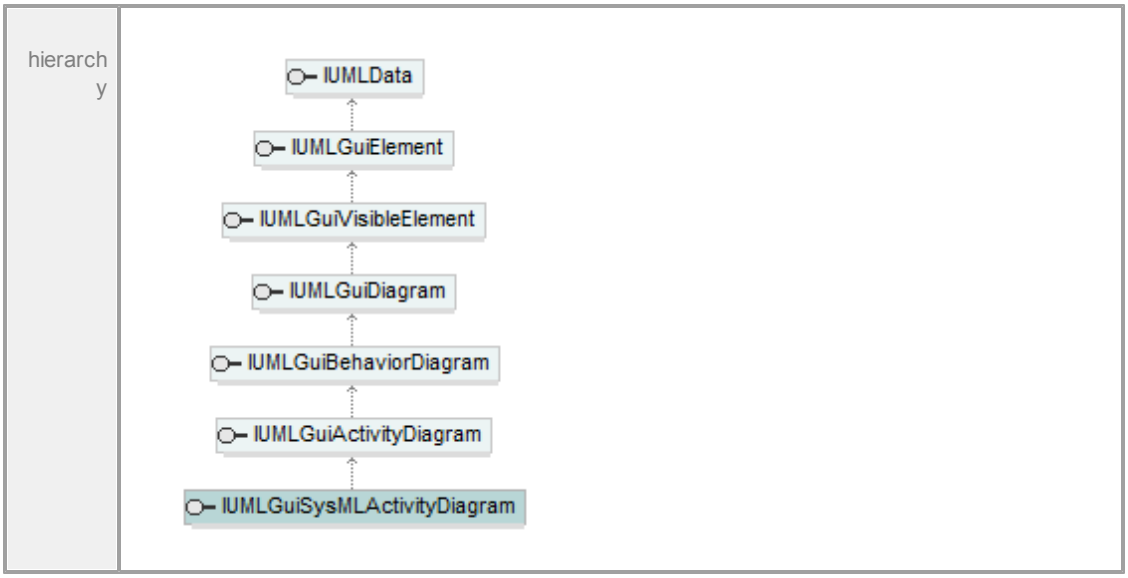
parameter	name	direction	type	type modifier	multiplicity	default
	nX	**in**	**int**			
	nY	**in**	**int**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiSysMLActivityDiagram

Interface **IUMLGuiSysMLActivityDiagram**

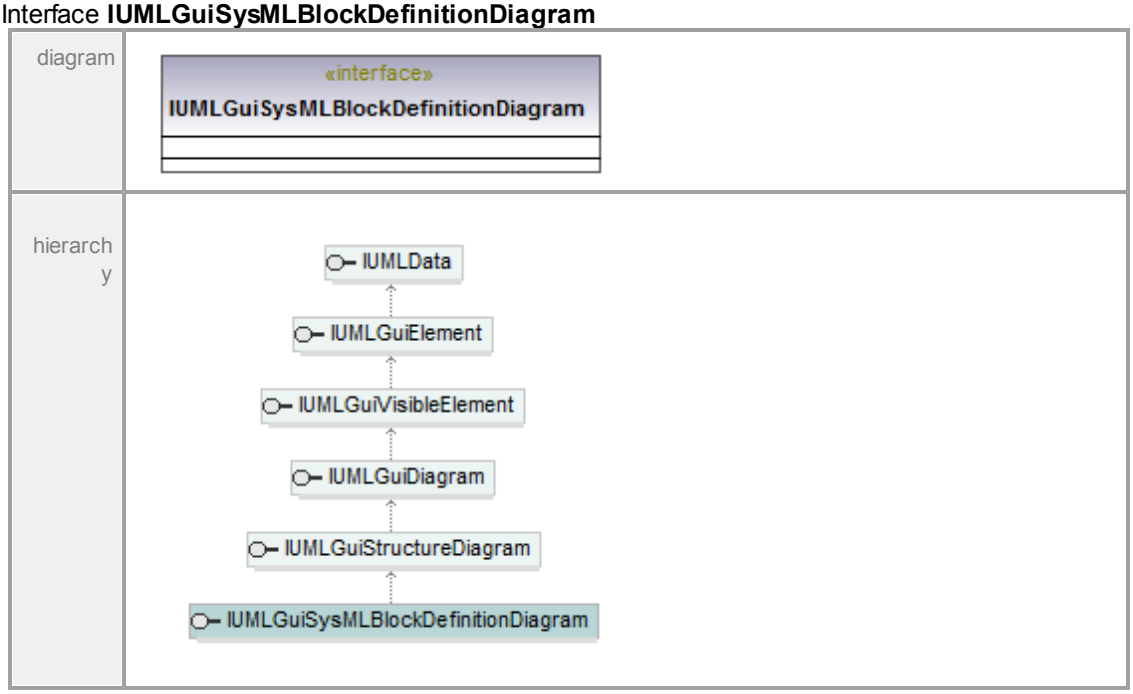
diagram	



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiSysMLBlockDefinitionDiagram

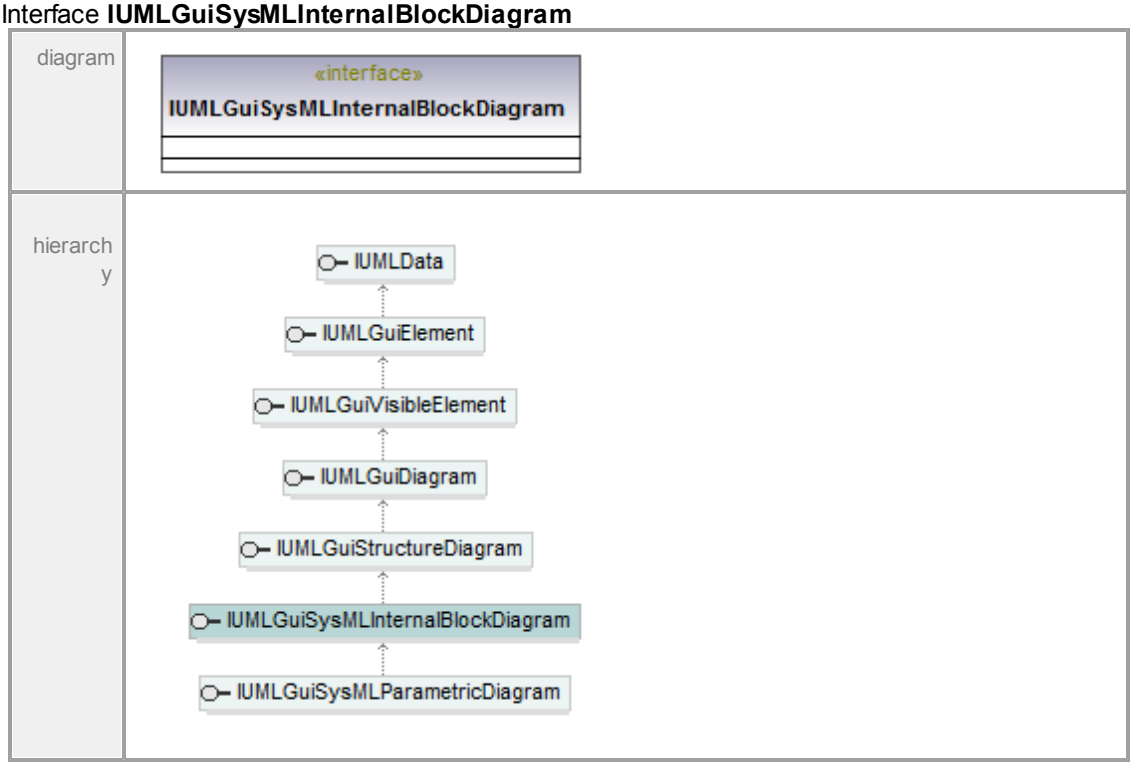
Interface **IUMLGuiSysMLBlockDefinitionDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiSysMLInternalBlockDiagram

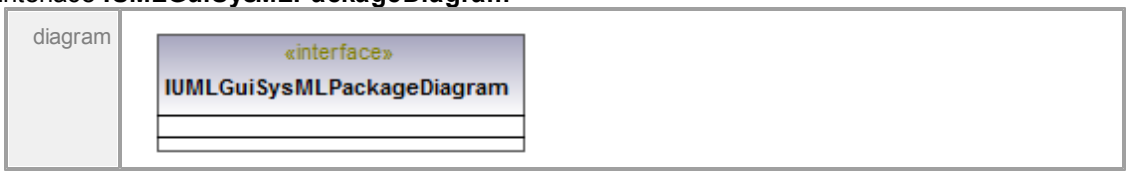
Interface **IUMLGuiSysMLInternalBlockDiagram**

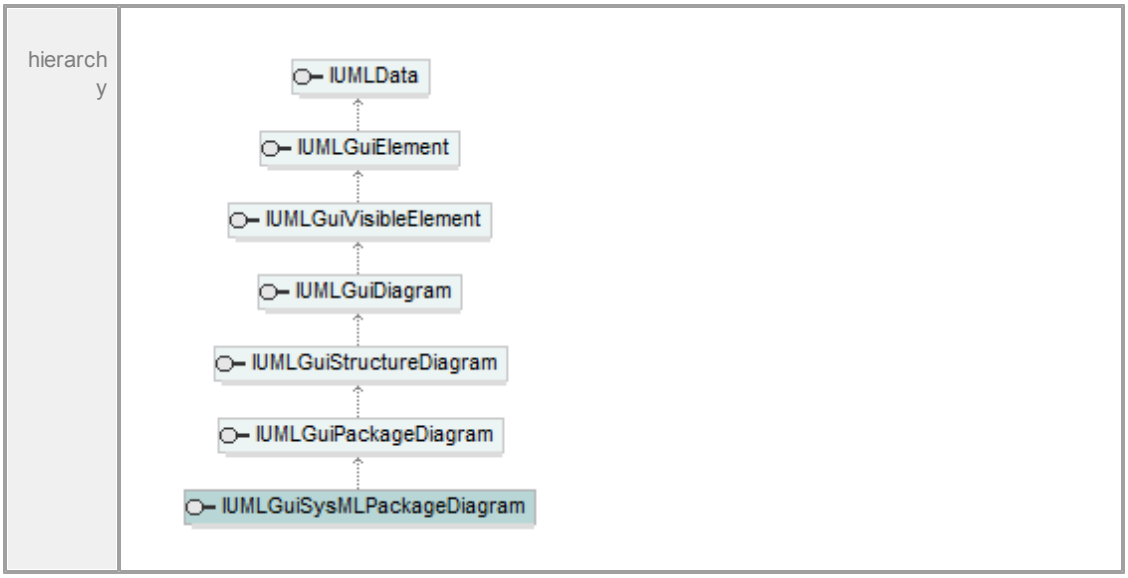


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiSysMLPackageDiagram

Interface **IUMLGuiSysMLPackageDiagram**

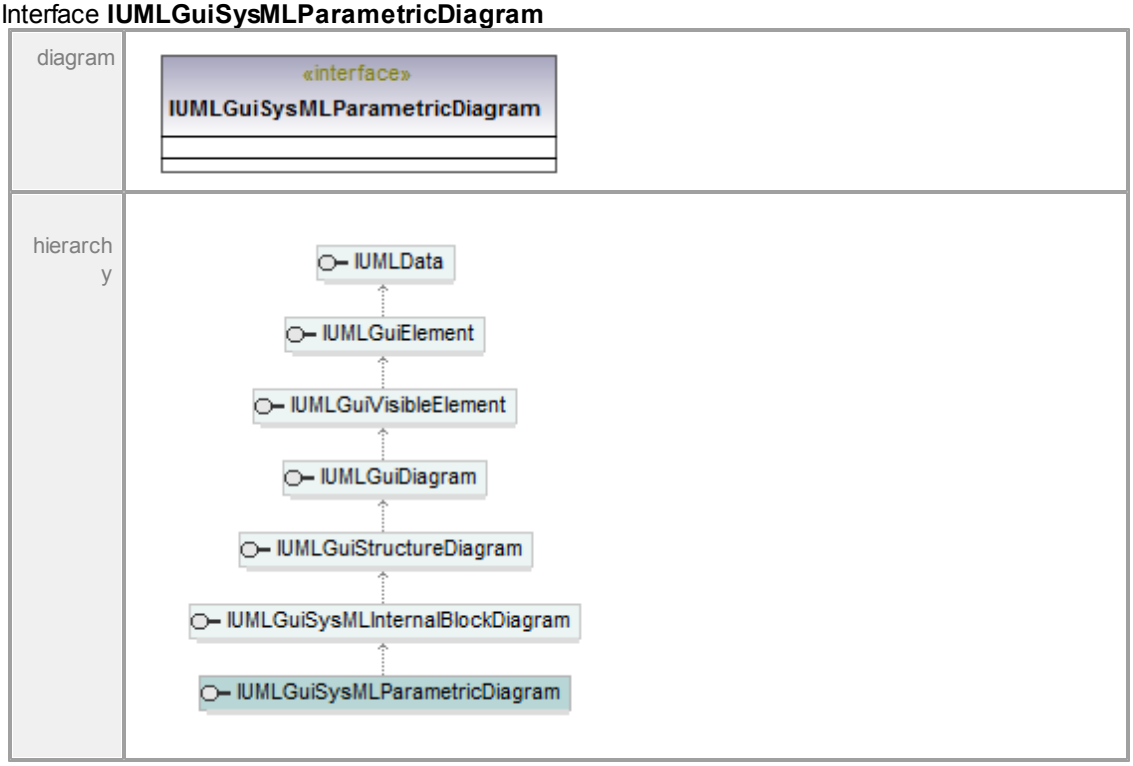




UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiSysMLParametricDiagram

Interface **IUMLGuiSysMLParametricDiagram**

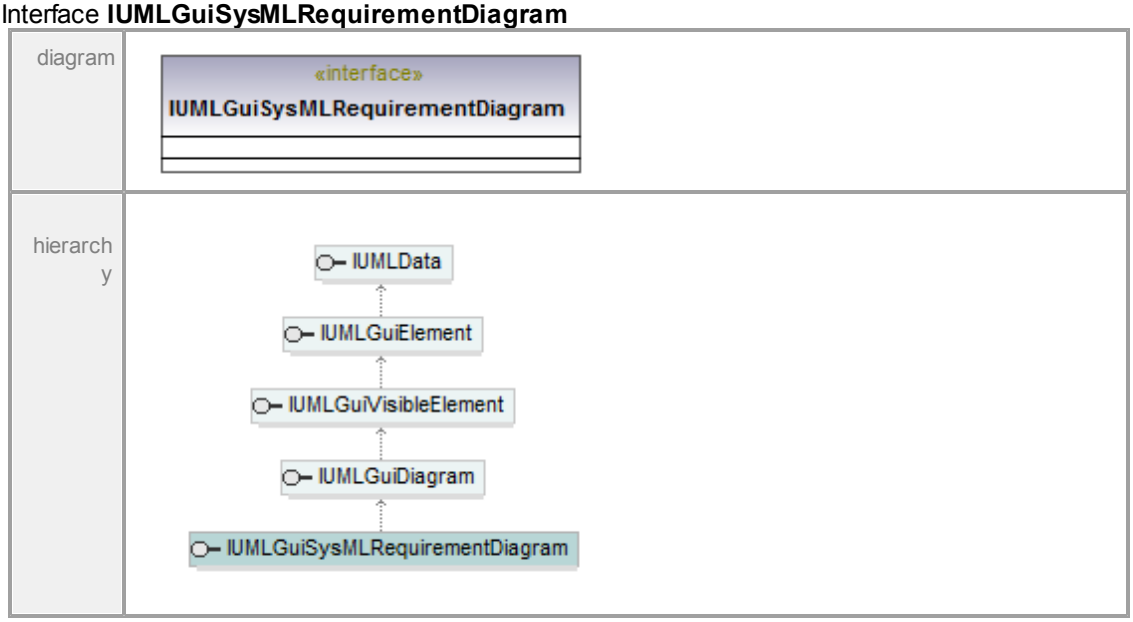


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - IUMLGuiSysMLRequirementDiagram

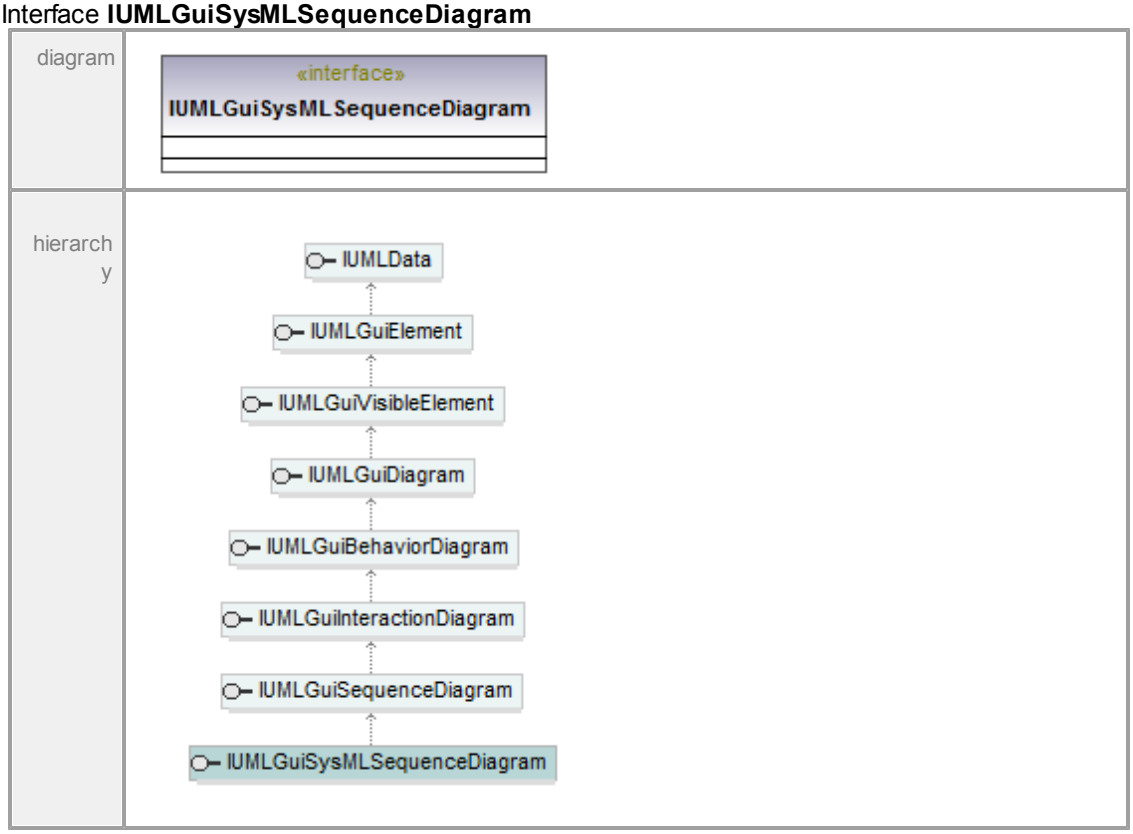
Interface **IUMLGuiSysMLRequirementDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiSysMLSequenceDiagram

Interface **IUMLGuiSysMLSequenceDiagram**

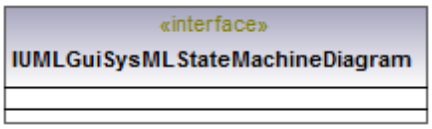


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

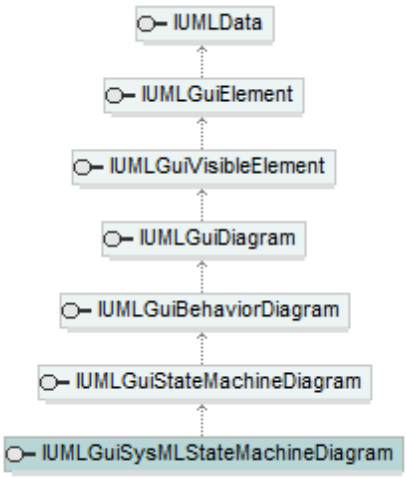
UModelAPI - IUMLGuiSysMLStateMachineDiagram

Interface **IUMLGuiSysMLStateMachineDiagram**

diagram



hierarchy

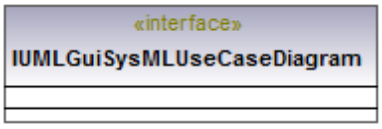


UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

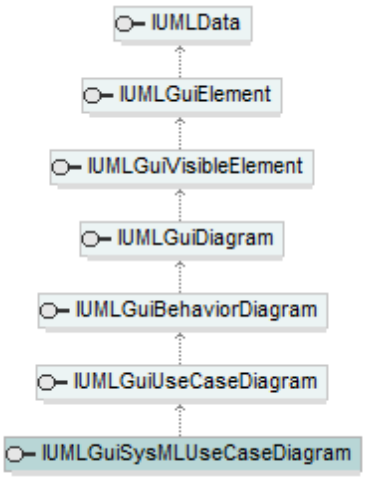
UModelAPI - IUMLGuiSysMLUseCaseDiagram

Interface **IUMLGuiSysMLUseCaseDiagram**

diagram



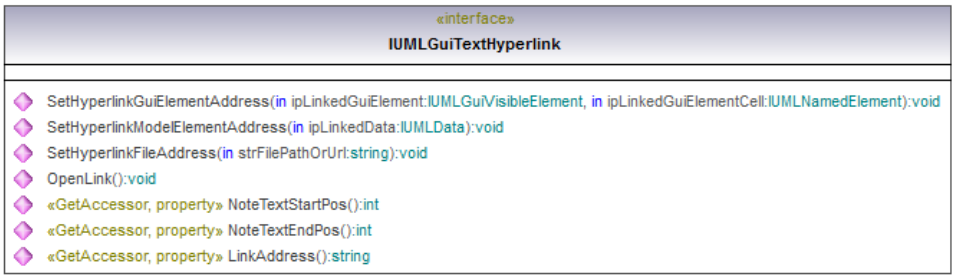
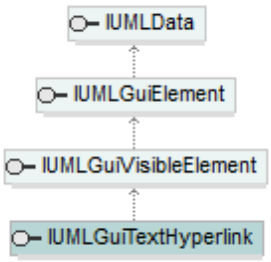
 © 2014 Altova GmbH

hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiTextHyperlink

Interface **IUMLGuiTextHyperlink**

diagram	
hierarchy	
typedElements	Interface **IUMLDataAll** — Operation **InsertOwnedGuiTextHyperlinkAt** Interface **IUMLGuiDiagram** — Operation **InsertOwnedGuiTextHyperlinkAt** Interface **IUMLGuiNote** — Operation **InsertOwnedGuiTextHyperlinkAt**
documentation	Text Hyperlinks store an URL together with a begin and an end number referencing positions in some text. Any text between a such a begin and end position is displayed as hyperlink and triggers UModel

	to open the URL when clicked. This is used in IUMLGuiNote 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**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLGuiTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - IUMLGuiTextLabel

Interface **IUMLGuiTextLabel**

diagram	«interface» **IUMLGuiTextLabel** SetPos(in nX:int, in nY:int):void «GetAccessor, property» TextLabelKind():ENUMUMLGuiTextLabelKind «GetAccessor, property» TextLabelElement():IUMLElement «GetAccessor, property» IsPositioned():bool «GetAccessor, property» PosX():int «GetAccessor, property» PosY():int
hierarchy	IUMLData IUMLGuiElement IUMLGuiVisibleElement IUMLGuiTextLabel
typedElements	Interface **IUMLDataAll** — Operation **GetTextLabelText**, **IsTextLabelVisible**, **SetTextLabelVisible** Interface **IUMLGuiTextLabelWaypoint** — Operation **GetTextLabelText**, **IsTextLabelVisible**, **SetTextLabelVisible**
documentation	A text label is an graphical object displaying additional data at the begin, end or at the center of a line. The IUMLGuiTextLabel interface provides access to this element. The text label can reference an UML Element using the TextLabelElement property and has a TextLabelKind 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

r	**nX**	**in**	**int**
	nY	**in**	**int**
	return	**return**	**void**

Operation **IUMLGuiTextLabel::TextLabelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLGuiTextLabel::TextLabelKind**

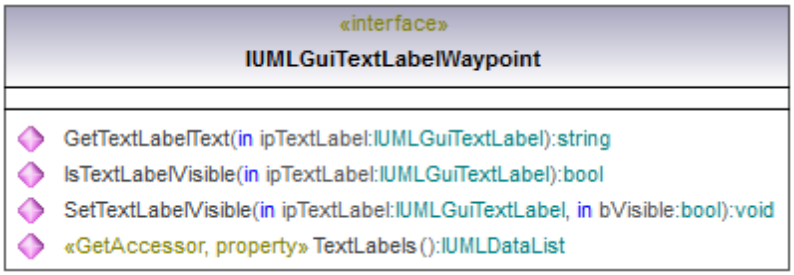
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLGuiTextLabelKind**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

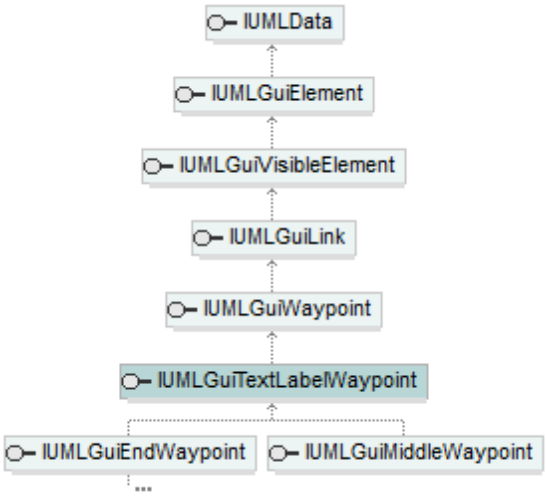
UModelAPI - IUMLGuiTextLabelWaypoint

Interface **IUMLGuiTextLabelWaypoint**

diagram



hierarchy



documentation

A text label waypoint is a special waypoint as part of a IUMLGuiLineLink 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.

 © 2014 Altova GmbH

Operation **IUMLGuiTextLabelWaypoint::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	return	**return**	**string**			

Operation **IUMLGuiTextLabelWaypoint::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	return	**return**	**bool**			

Operation **IUMLGuiTextLabelWaypoint::SetTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	bVisible	**in**	**bool**			
	return	**return**	**void**			

Operation **IUMLGuiTextLabelWaypoint::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

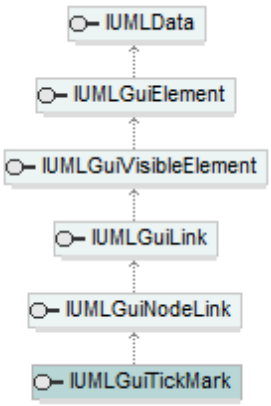
documentation	Returns a list of all text labels of this waypoint. Contains only elements of type (or subtype of) IUMLGuiTextLabel.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiTickMark

Interface **IUMLGuiTickMark**

diagram

hierarchy	
typedElements	Interface **IUMLGuiTimingDiagram** Operation **AddUMLGuiTickMark**
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 property which is displayed as text below the vertical line.

Operation **IUMLGuiTickMark::Value**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

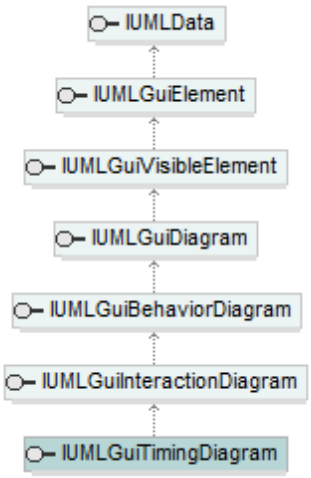
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiTimingDiagram

Interface **IUMLGuiTimingDiagram**

diagram	«interface» **IUMLGuiTimingDiagram** AddUMLGuiTickMark(in ipOnNode:IUMLGuiTimingDiagramLifeline, in nPosX:int, in nPosY:int):IUMLGuiTickMark

 © 2014 Altova GmbH

hierarchy	

Operation **IUMLGuiTimingDiagram::AddUMLGuiTickMark**

parameter	name	direction	type	type modifier	multiplicity	default
	ipOnNode	**in**	**IUMLGuiTimingDiagramLifeline**			
	nPosX	**in**	**int**			
	nPosY	**in**	**int**			
	return	**return**	**IUMLGuiTickMark**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiTimingDiagramLifeline

Interface **IUMLGuiTimingDiagramLifeline**

diagram	«interface» **IUMLGuiTimingDiagramLifeline** GetTimeTickLength(in nIdx:int):int SetTimeTickLength(in nIdx:int, in nNewVal:int):void GetStateIndex(in nTimeTickIndex:int):int SetStateIndex(in nTimeTickIndex:int, in nNewVal:int):void SetStateIndexErased(in nTimeTickIndex:int):void GetVisualStatePosition(in nStateIndex:int):int SetVisualStatePosition(in nStateIndex:int, in nNewVal:int):void «GetAccessor, SetAccessor, property» NameCompartmentEndPos():int «GetAccessor, SetAccessor, property» GeneralValueLifelineNameCompartmentEndPos():int «GetAccessor, SetAccessor, property» StateCompartmentEndPos():int «GetAccessor, SetAccessor, property» IsShowAsGeneralValueLifeline():bool «GetAccessor, SetAccessor, property» TimeTickLengthCount():int «GetAccessor, SetAccessor, property» VisualStatePositionCount():int
hierarchy	IUMLData IUMLGuiElement IUMLGuiVisibleElement IUMLGuiLink IUMLGuiNodeLink IUMLGuiTimingDiagramLifeline
typedElements	Interface **IUMLGuiTimingDiagram** Operation **AddUMLGuiTickMark**
documentation	A IUMLGuiTimingDiagramLifeline 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**

© 2014 Altova GmbH

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**			

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**

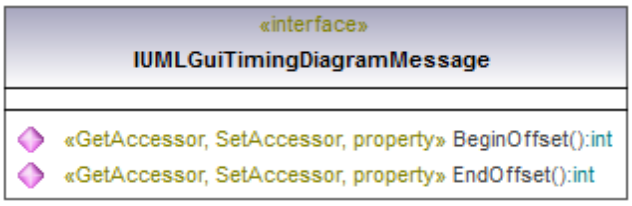
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

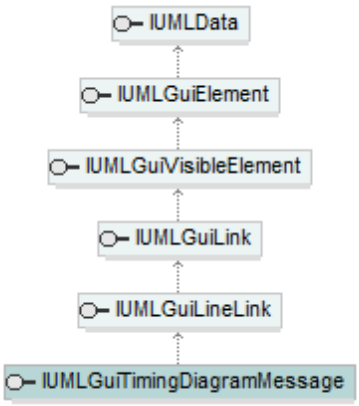
UModelAPI - IUMLGuiTimingDiagramMessage

Interface **IUMLGuiTimingDiagramMessage**

diagram



hierarchy



documentation

A IUMLGuiTimingDiagramMessage is a line usually connecting two IUMLGuiTimingDiagramLifelines. 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

 © 2014 Altova GmbH

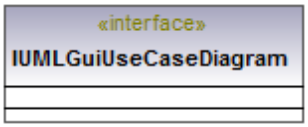
r	**return**	**return**	**int**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

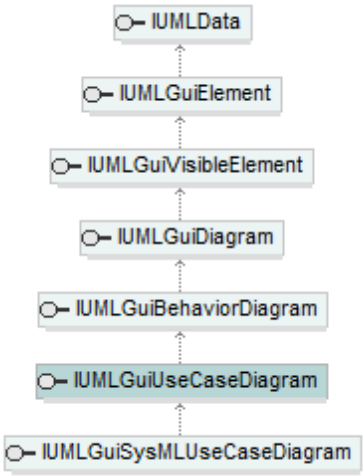
UModelAPI - IUMLGuiUseCaseDiagram

Interface **IUMLGuiUseCaseDiagram**

diagram



hierarchy

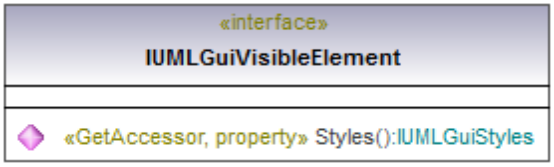


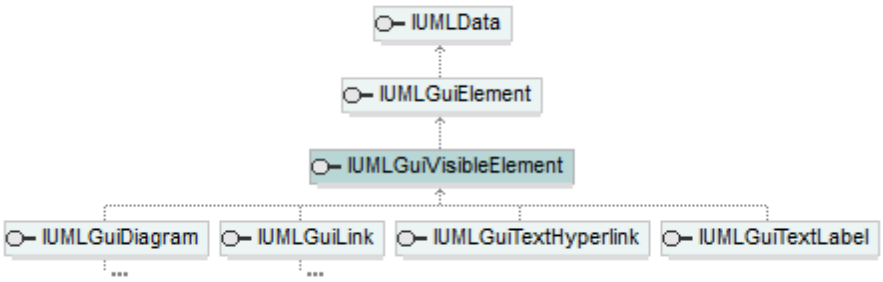
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiVisibleElement

Interface **IUMLGuiVisibleElement**

diagram



hierarchy	
typedElements	Interface **IUMLCommentTextHyperlink** Interface **IUMLDataAll** Interface **IUMLGuiTextHyperlink** Interface **IUMLHyperlink2GuiElement** Interface **IUMLNamedElement** Operation **SetHyperlinkGuiElementAddress** Operation **InsertOwnedHyperlink2GuiElementAt** **LinkedGuiElement** **SetHyperlinkGuiElementAddress** Operation **SetHyperlinkGuiElementAddress** Operation **LinkedGuiElement** Operation **InsertOwnedHyperlink2GuiElementAt**
documentation	The IUMLGuiVisibleElement 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.

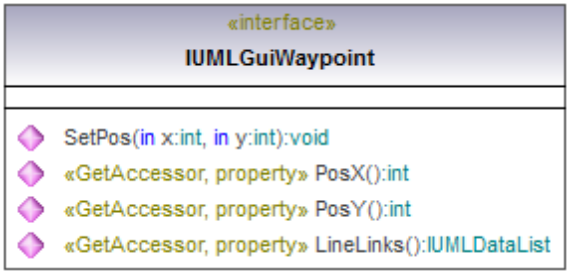
Operation **IUMLGuiVisibleElement::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

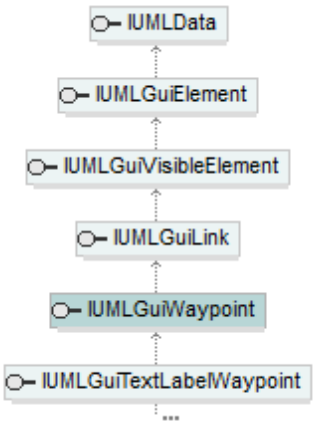
UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - IUMLGuiWaypoint

Interface **IUMLGuiWaypoint**

diagram	

 © 2014 Altova GmbH

hierarchy	
typedElements	Interface **IUMLDataAll** Operation **InsertWaypointAt** Interface **IUMLGuiLineLink** Operation **InsertWaypointAt**
documentation	A IUMLGuiWaypoint is a part of a IUMLGuiLineLink 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**			

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**			

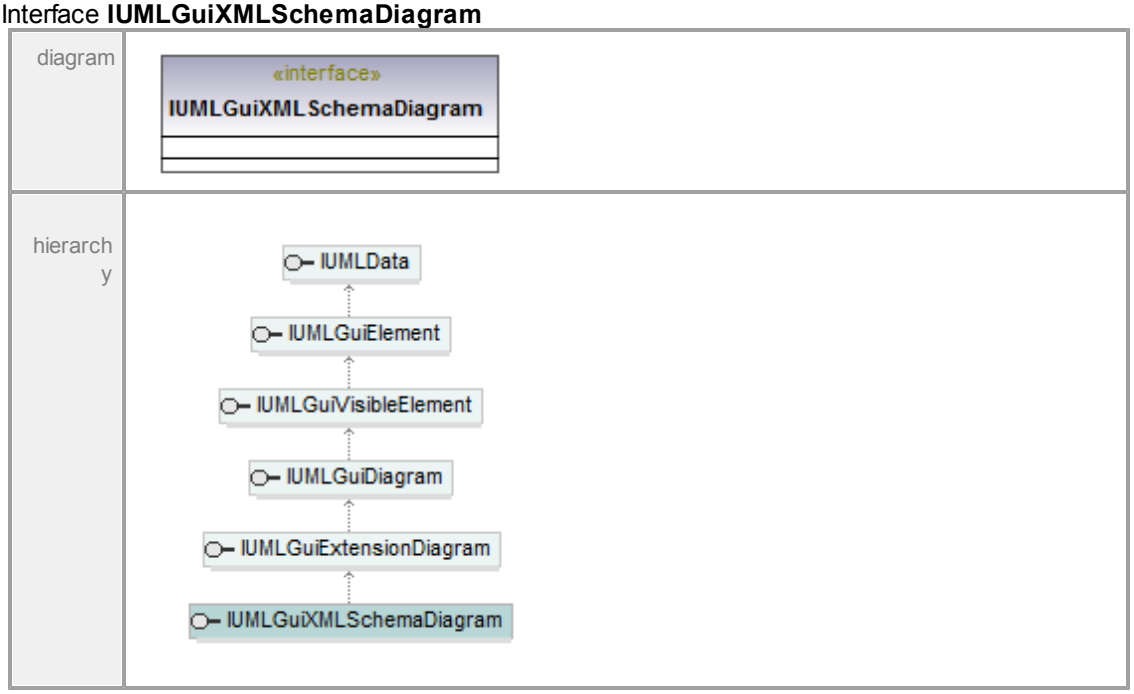
Operation **IUMLGuiWaypoint::SetPos**

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** 09/04/12 14:02:42

UModelAPI - IUMLGuiXMLSchemaDiagram

Interface **IUMLGuiXMLSchemaDiagram**



UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

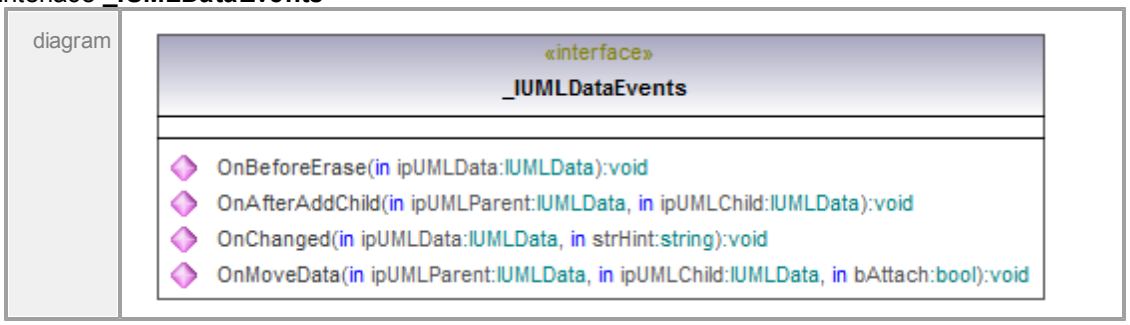
Events

This is a list of all events sent by the UModel API on UMLData level.

Also see UMLData events and eventfilters.

UModelAPI - _IUMLDataEvents

Interface **_IUMLDataEvents**



Operation **_IUMLDataEvents::OnAfterAddChild**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLParent	**in**	**IUMLData**			
	ipUMLChild	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **_IUMLDataEvents::OnBeforeErase**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	return	**return**	**void**			

Operation **_IUMLDataEvents::OnChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	**in**	**IUMLData**			
	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**			
	ipUMLChild	**in**	**IUMLData**			
	bAttach	**in**	**bool**			
	return	**return**	**void**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

Enumerations

This is a list of all enumerations used by the UModel API on UMLData level. If your scripting environment does not support enumerations use the number-values instead.

.

UModelAPI - ENUMUMLAggregationKind

Enumeration **ENUMUMLAggregationKind**

diagram	«enumeration» **ENUMUMLAggregationKind** eAggregation_None = 0 eAggregation_Shared = 1 eAggregation_Composite = 2
typedElements	Interface **IUMLDataAll** Operation **Aggregation** Interface **IUMLProperty** Operation **Aggregation**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLCallConcurrencyKind

Enumeration **ENUMUMLCallConcurrencyKind**

diagram	«enumeration» **ENUMUMLCallConcurrencyKind** eCallConcurrency_Sequential = 0 eCallConcurrency_Guarded = 1 eCallConcurrency_Concurrent = 2
typedElements	Interface **IUMLBehavioralFeature** Operation **Concurrency** Interface **IUMLDataAll** Operation **Concurrency**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLConnectorKind

Enumeration **ENUMUMLConnectorKind**

diagram	«enumeration» **ENUMUMLConnectorKind** eConnector_Assembly = 0 eConnector_Delegation = 1
typedElements	Interface **IUMLConnector** Operation **ConnectorKind** Interface **IUMLDataAll** Operation **ConnectorKind**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - ENUMUMLDataEventFilter

Enumeration **ENUMUMLDataEventFilter**

diagram	«enumeration» **ENUMUMLDataEventFilter** eUMLDataEvent_None = 0 eUMLDataEvent_EraseData = 1 eUMLDataEvent_EraseDataOrChild = 2 eUMLDataEvent_Erase = 3 eUMLDataEvent_AddChild = 4 eUMLDataEvent_AddChildOrGrandChild = 8 eUMLDataEvent_Add = 12 eUMLDataEvent_ChangeData = 16 eUMLDataEvent_ChangeDataOrChild = 32 eUMLDataEvent_Change = 48 eUMLDataEvent_MoveChild = 64 eUMLDataEvent_MoveChildOrGrandChild = 128 eUMLDataEvent_Move = 192 eUMLDataEvent_AllNoMove = 63 eUMLDataEvent_All = 255

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLDBDataSourceMethod

Enumeration **ENUMUMLDBDataSourceMethod**

diagram	«enumeration» **ENUMUMLDBDataSourceMethod** eUMLDB_DataSourceMethod_ODBC = 0 eUMLDB_DataSourceMethod_ADO = 1 eUMLDB_DataSourceMethod_OracleNative = 2 eUMLDB_DataSourceMethod_Access = 3	
typedElements	Interface **IImportDatabaseDlg**	Operation **SelectNewDataSourceByConnectionString**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLExpansionKind

Enumeration **ENUMUMLExpansionKind**

diagram	«enumeration» **ENUMUMLExpansionKind** eExpansion_parallel = 0 eExpansion_iterative = 1 eExpansion_stream = 2
typedElements	Interface **IUMLDataAll** Interface **IUMLExpansionRegion** Operation **Mode** Operation **Mode**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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 (http://manual.altova.com/UModel/umodelprofessional/).]***
typedElements	Interface **IUMLGuiStyle** Interface **IUMLGuiStyles** Operation **Kind** Operation **GetName** **GetStyle** **GetUsedValue** **GetValue** **SetValue**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - ENUMUMLGuiTextLabelKind

Enumeration **ENUMUMLGuiTextLabelKind**

diagram

«enumeration»
ENUMUMLGuiTextLabelKind

eTextLabel_Element_Stereotype = 0
eTextLabel_Association_Name = 1
eTextLabel_Property_Name = 2
eTextLabel_Property_Multiplicity = 3
eTextLabel_Property_Constraint = 4
eTextLabel_Link_Name = 5
eTextLabel_LinkBegin_PropertyName = 6
eTextLabel_LinkEnd_PropertyName = 7
eTextLabel_Dependency = 8
eTextLabel_Usage = 9
eTextLabel_Manifestation = 10
eTextLabel_Deployment = 11
eTextLabel_Include = 12
eTextLabel_Extend = 13
eTextLabel_ProfileApplication = 14
eTextLabel_MessageString = 15
eTextLabel_Element_Constraint = 16
eTextLabel_ActivityEdge_Name = 17
eTextLabel_ActivityEdge_Guard = 18
eTextLabel_ActivityEdge_Weight = 19
eTextLabel_ObjectFlow_MultiCast = 20
eTextLabel_ObjectFlow_MultiReceive = 21
eTextLabel_ExceptionHandler_ExceptionType = 22
eTextLabel_Transition_Expression = 23
eTextLabel_DependencyRoleBinding_RoleName = 24
eTextLabel_Connector_Name = 25
eTextLabel_PackageMerge = 26
eTextLabel_PackageImport = 27
eTextLabel_ElementImport = 28
eTextLabel_BPMNConditionExpression = 29
eTextLabel_Abstraction = 30
eTextLabel_MemberEnd_Stereotype = 31
eTextLabel_ObjectFlow_DecisionInput = 32
eTextLabel_InformationFlow = 33
eTextLabel_DotNetPropertyName = 34

typedElements		
	Interface **IUMLDataAll**	Operation **TextLabelKind**
	Interface **IUMLGuiTextLabel**	Operation **TextLabelKind**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLInteractionOperatorKind

Enumeration **ENUMUMLInteractionOperatorKind**

diagram	«enumeration» **ENUMUMLInteractionOperatorKind** eInteractionOperator_Seq = 0 eInteractionOperator_Alt = 1 eInteractionOperator_Opt = 2 eInteractionOperator_Break = 3 eInteractionOperator_Par = 4 eInteractionOperator_Strict = 5 eInteractionOperator_Loop = 6 eInteractionOperator_Critical = 7 eInteractionOperator_Neg = 8 eInteractionOperator_Assert = 9 eInteractionOperator_Ignore = 10 eInteractionOperator_Consider = 11
typedElements	Interface **IUMLCombinedFragment** Operation **InteractionOperator** Interface **IUMLDataAll** Operation **InteractionOperator**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLMessageKind

Enumeration **ENUMUMLMessageKind**

diagram	«enumeration» **ENUMUMLMessageKind** eMessage_Complete = 0 eMessage_Lost = 1 eMessage_Found = 2 eMessage_Unknown = 3
typedElements	Interface **IUMLDataAll** Operation **MessageKind** Interface **IUMLMessage** Operation **MessageKind**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

© 2014 Altova GmbH

UModelAPI - ENUMUMLMessageSort

Enumeration **ENUMUMLMessageSort**

diagram	«enumeration» **ENUMUMLMessageSort** eMessageSort_SynchCall = 0 eMessageSort_AsynchCall = 1 eMessageSort_AsynchSignal = 2 eMessageSort_CreateMessage = 3 eMessageSort_DeleteMessage = 4 eMessageSort_Reply = 5
typedElements	Interface **IUMLDataAll** Operation **MessageSort** Interface **IUMLMessage** Operation **MessageSort**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLObjectNodeOrderingKind

Enumeration **ENUMUMLObjectNodeOrderingKind**

diagram	«enumeration» **ENUMUMLObjectNodeOrderingKind** eObjectNodeOrdering_unordered = 0 eObjectNodeOrdering_ordered = 1 eObjectNodeOrdering_LIFO = 2 eObjectNodeOrdering_FIFO = 3
typedElements	Interface **IUMLDataAll** Operation **Ordering** Interface **IUMLObjectNode** Operation **Ordering**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLParameterDirectionKind

Enumeration **ENUMUMLParameterDirectionKind**

diagram	«enumeration» **ENUMUMLParameterDirectionKind** eParameterDirection_In = 0 eParameterDirection_InOut = 1 eParameterDirection_Out = 2 eParameterDirection_Return = 3
typedElements	Interface **IUMLDataAll** Operation **Direction** Interface **IUMLParameter** Operation **Direction**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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 (http://manual.altova.com/UModel/umodelprofessional/).]***
typedElements	Interface **IUMLDataAll** — Operation **ApplyPredefinedStereotype**, **FindPredefinedOwnedElement**, **GetStereotypeApplicationForPredefinedStereotype**, **IsPredefinedStereotypeApplied**, **SetPredefinedTaggedValueAt**, **UnapplyPredefinedStereotype** Interface **IUMLElement** — Operation **ApplyPredefinedStereotype**, **FindPredefinedOwnedElement**, **GetStereotypeApplicationForPredefinedStereotype**, **IsPredefinedStereotypeApplied**, **UnapplyPredefinedStereotype** Interface **IUMLStereotypeApplication** — Operation **SetPredefinedTaggedValueAt**
documentation	**Deprecated:** ePredefined_Java_finalStereotypeOfProperty ePredefined_Java_finalStereotypeOfOperation ePredefined_Java_finalStereotypeOfClass

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

 © 2014 Altova GmbH

UModelAPI - ENUMUMLPseudostateKind

Enumeration **ENUMUMLPseudostateKind**

diagram	«enumeration» **ENUMUMLPseudostateKind** ePseudostate_Initial = 0 ePseudostate_DeepHistory = 1 ePseudostate_ShallowHistory = 2 ePseudostate_Join = 3 ePseudostate_Fork = 4 ePseudostate_Junction = 5 ePseudostate_Choice = 6 ePseudostate_EntryPoint = 7 ePseudostate_ExitPoint = 8 ePseudostate_Terminate = 9
typedElements	Interface **IUMLDataAll** Operation **PseudostateKind** Interface **IUMLPseudostate** Operation **PseudostateKind**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLTransitionKind

Enumeration **ENUMUMLTransitionKind**

diagram	«enumeration» **ENUMUMLTransitionKind** eTransition_Internal = 0 eTransition_Local = 1 eTransition_External = 2
typedElements	Interface **IUMLDataAll** Operation **TransitionKind** Interface **IUMLTransition** Operation **TransitionKind**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - ENUMUMLVisibilityKind

Enumeration **ENUMUMLVisibilityKind**

diagram	«enumeration» **ENUMUMLVisibilityKind** eVisibility_Unspecified = -1 eVisibility_Public = 0 eVisibility_Protected = 1 eVisibility_Private = 2 eVisibility_Package = 3
typedElements	Interface **ILocalOptionsEditing** — Operation **OperationsDefaultVisibility** Operation **PropertiesDefaultVisibility** Interface **IUMLDataAll** — Operation **Visibility** Interface **IUMLElementImport** — Operation **Visibility** Interface **IUMLNamedElement** — Operation **Visibility** Interface **IUMLPackageImport** — Operation **Visibility**

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

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 (http://manual.altova.com/UModel/umodelprofessional/).]***
hierarchy	IUMLDataAll ↑ UMLData

Operation **IUMLDataAll::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComponent**			

Operation **IUMLDataAll::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActionInputPin**			

 © 2014 Altova GmbH

Operation **IUMLDataAll::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLDataAll::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLDataAll::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			

Operation **IUMLDataAll::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	**in**	**IUMLGuiLink**			
	ipToLink	**in**	**IUMLGuiLink**			
	return	**return**	**IUMLGuiContainmentLink**			

Operation **IUMLDataAll::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	**in**	**IUMLData**			
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNodeLink**			

Operation **IUMLDataAll::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	**in**	**int**			
	nTop	**in**	**int**			
	return	**return**	**IUMLGuiNote**			

Operation **IUMLDataAll::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLDataAll::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	**in**	**IUMLGuiNote**			
	ipToLink	**in**	**IUMLGuiLineLink**			
	nDistanceFromLineBegin	**in**	**int**			
	return	**return**	**IUMLGuiNoteLink**			

Operation **IUMLDataAll::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	ipFromNode	**in**	**IUMLGuiNodeL**			

© 2014 Altova GmbH

	ipToNode	**in**	**ink IUMLGuiNodeLink**			
	return	**return**	**IUMLGuiLineLink**			

Operation **IUMLDataAll::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLAggregationKind**			

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**			

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**			

Operation **IUMLDataAll::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProfile**			

Operation **IUMLDataAll::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ApplyingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Association**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLDataAll::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiLink**			

Operation **IUMLDataAll::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

 © 2014 Altova GmbH

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**			

Operation **IUMLDataAll::BehaviorExecution**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehaviorafeature**			

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**			

Operation **IUMLDataAll::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::ChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClass**			

Operation **IUMLDataAll::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::ClientDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Clients**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMCodeLangVersion**			

Operation **IUMLDataAll::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOperation**			

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**			

Operation **IUMLDataAll::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**return**	**return**	**IUMLCollaboration**			

Operation **IUMLDataAll::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**string**			

Operation **IUMLDataAll::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**ENUMUMLCallConcurrencyKind**			

Operation **IUMLDataAll::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**ENUMUMLConnectorKind**			

Operation **IUMLDataAll::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLAssociation**			

Operation **IUMLDataAll::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ConstrainingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ConstrainingPointX**

parameter	name	direction	type	type modifier	multiplicity	default
r	**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**			

Operation **IUMLDataAll::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLRegion**			

Operation **IUMLDataAll::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLDataAll::Contract**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

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**			

Operation **IUMLDataAll::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataType**			

Operation **IUMLDataAll::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Default**

 © 2014 Altova GmbH

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**			

Operation **IUMLDataAll::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStructuralFeature**			

Operation **IUMLDataAll::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Deployments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLParameterDirectionKind**			

Operation **IUMLDataAll::DistanceFromLineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

Operation **IUMLDataAll::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Element**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Enumeration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLEnumeration**			

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**			

© 2014 Altova GmbH

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**			

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

r	**ipVal**	**in**	**IUMLGuiEleme nt**			
	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**			

 © 2014 Altova GmbH

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**			

Operation **IUMLDataAll::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::EventFilter**

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**			

Operation **IUMLDataAll::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLObjectNode**			

Operation **IUMLDataAll::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLDataAll::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutionSpecification**			

Operation **IUMLDataAll::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::Exits**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpression**			

Operation **IUMLDataAll::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

© 2014 Altova GmbH

Operation **IUMLDataAll::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::FeaturingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	**in**	**ENUMUMLPredefinedElement**			
	bRecursive	**in**	**bool**			
	return	**return**	**IUMLData**			

Operation **IUMLDataAll::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

Operation **IUMLDataAll::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

Operation **IUMLDataAll::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	bRecursive	**in**	**bool**			

© 2014 Altova GmbH

	return	**return**	**IUMLDataList**			

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**			

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**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	return	**return**	**IUMLStereotypeApplication**			

Operation **IUMLDataAll::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	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

r	**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**			

Operation **IUMLDataAll::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLDataAll::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

Operation **IUMLDataAll::Handlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**	**IUMLBehaviredClassifier**			

Operation **IUMLDataAll::ImportedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackageableElement**			

© 2014 Altova GmbH

Operation **IUMLDataAll::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLDataAll::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivity**			

Operation **IUMLDataAll::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLDataAll::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLActivityNode**			
	ipTo	**in**	**IUMLActivityNode**			
	return	**return**	**IUMLActivityEdge**			

Operation **IUMLDataAll::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityGroup**			

Operation **IUMLDataAll::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityNode**			

Operation **IUMLDataAll::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			

 © 2014 Altova GmbH

	return	return	IUMLGate

Operation **IUMLDataAll::InsertAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLElement			
	return	return	void			

Operation **IUMLDataAll::InsertArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLInputPin			

Operation **IUMLDataAll::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strNewVal	in	string			
	return	return	void			

Operation **IUMLDataAll::InsertCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLConnectableElement			
	return	return	void			

Operation **IUMLDataAll::InsertCollaborationUseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLCollaborationUse			

Operation **IUMLDataAll::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLConnectionPointReference			

Operation **IUMLDataAll::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLPseudostate			

Operation **IUMLDataAll::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLElement			
	return	return	void			

Operation **IUMLDataAll::InsertConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLInteractionFragment**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDeployedArtifact	**in**	**IUMLDeployedArtifact**			
	return	**return**	**IUMLDeployment**			

Operation **IUMLDataAll::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedElement	**in**	**IUMLPackageableElement**			
	return	**return**	**IUMLElementImport**			

Operation **IUMLDataAll::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

© 2014 Altova GmbH

Operation **IUMLDataAll::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLPseudostate**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipExtendedCase	**in**	**IUMLUseCase**			
	return	**return**	**IUMLExtend**			

Operation **IUMLDataAll::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pExtensionLocation	**in**	**IUMLExtensionPoint**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExtensionPoint**			

Operation **IUMLDataAll::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGate**			

Operation **IUMLDataAll::InsertFragmentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLInteractionFragment**			

Operation **IUMLDataAll::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipGeneral	**in**	**IUMLClassifier**			
	return	**return**	**IUMLGeneralization**			

Operation **IUMLDataAll::InsertHandlerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLExceptionHandler**			

Operation **IUMLDataAll::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipIncludingCase	**in**	**IUMLUseCase**			
	return	**return**	**IUMLInclude**			

Operation **IUMLDataAll::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLRelationship**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInputValueAt**

© 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLState**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipContract	**in**	**IUMLInterface**			
	return	**return**	**IUMLInterface Realization**			

Operation **IUMLDataAll::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipEdge	**in**	**IUMLActivityEdge**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiDiagramLayer**			

Operation **IUMLDataAll::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLLifeline**			

Operation **IUMLDataAll::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

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**			
	return	**return**	**IUMLManifestation**			

Operation **IUMLDataAll::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLMessage**			

Operation **IUMLDataAll::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLArtifact**			

Operation **IUMLDataAll::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLNode**			

Operation **IUMLDataAll::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLActivityNode**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLObservation**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertOperandAt**

© 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLInteractionOperand**			

Operation **IUMLDataAll::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipNode	**in**	**IUMLExpansionNode**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLComment**			

Operation **IUMLDataAll::InsertOwnedCommentTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLCommentTextHyperlink**			

Operation **IUMLDataAll::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFrom	**in**	**IUMLConnectableElement**			
	ipTo	**in**	**IUMLConnectableElement**			
	return	**return**	**IUMLConnector**			

Operation **IUMLDataAll::InsertOwnedDiagramAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipUMLParent	**in**	**IUMLData**			
	strKind	**in**	**string**			
	return	**return**	**IUMLGuiDiagram**			

Operation **IUMLDataAll::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	**in**	**int**			
	nToTextPos	**in**	**int**			
	strAddress	**in**	**string**			
	return	**return**	**IUMLGuiTextHyperlink**			

Operation **IUMLDataAll::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strFilePathOrUrl	**in**	**string**			
	return	**return**	**IUMLHyperlink2File**			

Operation **IUMLDataAll::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedGuiElement	**in**	**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell	**in**	**IUMLNamedElement**			
	return	**return**	**IUMLHyperlink2GuiElement**			

Operation **IUMLDataAll::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipLinkedData	**in**	**IUMLData**			
	return	**return**	**IUMLHyperlink2Model**			

Operation **IUMLDataAll::InsertOwnedLiteralAt**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLEnumerationLiteral**			

Operation **IUMLDataAll::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOperation**			

Operation **IUMLDataAll::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLParameter**			

Operation **IUMLDataAll::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLPort**			

Operation **IUMLDataAll::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSignature	**in**	**IUMLTemplateSignature**			
	return	**return**	**IUMLTemplateBinding**			

Operation **IUMLDataAll::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLClassifierTemplateParameter**			

Operation **IUMLDataAll::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	**nIdx**	**in**	**int**
	strKind	**in**	**string**
	return	**return**	**IUMLPackageableElement**

Operation **IUMLDataAll::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	ipFrom	**in**	**IUMLElement**			
	ipTo	**in**	**IUMLElement**			
	return	**return**	**IUMLPackageableElement**			

Operation **IUMLDataAll::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipImportedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageImport**			

Operation **IUMLDataAll::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipMergedPackage	**in**	**IUMLPackage**			
	return	**return**	**IUMLPackageMerge**			

Operation **IUMLDataAll::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipFormalParameter	**in**	**IUMLTemplateParameter**			
	ipActualParameter	**in**	**IUMLParameterableElement**			
	return	**return**	**IUMLTemplateParameterSubstitution**			

Operation **IUMLDataAll::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLConstraint**			

© 2014 Altova GmbH

Operation **IUMLDataAll::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipAppliedProfile	**in**	**IUMLProfile**			
	return	**return**	**IUMLProfileApplication**			

Operation **IUMLDataAll::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLType**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipRealizingClassifier	**in**	**IUMLClassifier**			
	return	**return**	**IUMLComponentRealization**			

Operation **IUMLDataAll::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipVal	**in**	**IUMLConnector**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLRegion**			

Operation **IUMLDataAll::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipDefiningFeature	**in**	**IUMLStructuralFeature**			

	return	**return**	**IUMLSlot**			

Operation **IUMLDataAll::InsertSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	pSubject	**in**	**IUMLClassifier**			
	return	**return**	**void**			

Operation **IUMLDataAll::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLActivityPartition**			

Operation **IUMLDataAll::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLVertex**			

Operation **IUMLDataAll::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipSource	**in**	**IUMLVertex**			
	ipTarget	**in**	**IUMLVertex**			
	return	**return**	**IUMLTransition**			

Operation **IUMLDataAll::InsertTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLTrigger**			

Operation **IUMLDataAll::InsertValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::InsertWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	return	**return**	**IUMLGuiWaypoint**			

Operation **IUMLDataAll::Instance**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**return**	**return**	**IUMLInstanceSpecification**			

Operation **IUMLDataAll::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::IntegerValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**int**			

Operation **IUMLDataAll::InteractionOperator**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**ENUMUMLInteractionOperatorKind**			

Operation **IUMLDataAll::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLInterface**			

Operation **IUMLDataAll::InterfaceRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InterruptibleActivityRegions**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**bool**			

Operation **IUMLDataAll::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
r	**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**			
	return	**return**	**bool**			

Operation **IUMLDataAll::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	**in**	**ENUMCodeLang**			
	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**			

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**			

© 2014 Altova GmbH

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**			
	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**			

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

© 2014 Altova GmbH

r	**nStereotype**	**in**	**ENUMUMLPredefinedElement**			
	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**			
	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**			
	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**			
	return	**return**	**bool**			

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**			

 © 2014 Altova GmbH

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**			

Operation **IUMLDataAll::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLDataAll::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiElement**			

Operation **IUMLDataAll::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamedElement**			

Operation **IUMLDataAll::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLData**			

Operation **IUMLDataAll::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::LocalPostConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Location**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDeploymentTarget**			

Operation **IUMLDataAll::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueEx**			

© 2014 Altova GmbH

	pression

Operation **IUMLDataAll::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification			

Operation **IUMLDataAll::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLValueSpecification			

Operation **IUMLDataAll::MemberEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::Members**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLPackage			

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			

Operation **IUMLDataAll::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLMessageKind			

Operation **IUMLDataAll::Messages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	IUMLDataList			

Operation **IUMLDataAll::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	ENUMUMLMessageSort			

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**			

Operation **IUMLDataAll::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiMiddleWaypoint**			

Operation **IUMLDataAll::Min**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLExpansionKind**			

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**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLNamespace**			

Operation **IUMLDataAll::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestedPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::NestingInterface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInterface**			

Operation **IUMLDataAll::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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

r	**return**	**return**	**int**			

Operation **IUMLDataAll::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OccurringEvent**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLEvent**			

Operation **IUMLDataAll::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**void**			

Operation **IUMLDataAll::OperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLInteractionConstraint**			

Operation **IUMLDataAll::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLOperation**			

Operation **IUMLDataAll::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLProperty**			

Operation **IUMLDataAll::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**ENUMUMLObjectNodeOrderingKind**			

Operation **IUMLDataAll::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OutputElements**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OutputPins**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OutputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedActual**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLComment**			

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**			

Operation **IUMLDataAll::OwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedPorts**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLDataAll::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLAssociation**			

Operation **IUMLDataAll::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::OwningElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLElement**			

Operation **IUMLDataAll::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiNodeLink**			

Operation **IUMLDataAll::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLDataAll::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInstanceSpecification**			

Operation **IUMLDataAll::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLDataAll::OwningPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

Operation **IUMLDataAll::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProperty**			

Operation **IUMLDataAll::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

Operation **IUMLDataAll::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSlot**			

Operation **IUMLDataAll::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

© 2014 Altova GmbH

Operation **IUMLDataAll::OwningTemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

Operation **IUMLDataAll::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolTransition**			

Operation **IUMLDataAll::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMultiplicityElement**			

Operation **IUMLDataAll::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::PackageMerges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameter**			

Operation **IUMLDataAll::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLParameterableElement**			

Operation **IUMLDataAll::ParameterSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLDataAll::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExecutableNode**			

Operation **IUMLDataAll::Protocol**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

Operation **IUMLDataAll::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLPseudostateKind**			

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**			

Operation **IUMLDataAll::RaisedExceptions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::RealizingConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd**			

Operation **IUMLDataAll::ReceivingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLPackage**			

Operation **IUMLDataAll::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiDiagram**			

Operation **IUMLDataAll::Referred**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RefersTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteraction**			

Operation **IUMLDataAll::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

Operation **IUMLDataAll::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLExpansionRegion**			

Operation **IUMLDataAll::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RelatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Represents**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConnectableElement**			

Operation **IUMLDataAll::Result**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOutputPin**			

Operation **IUMLDataAll::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Right**

 © 2014 Altova GmbH

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**			

Operation **IUMLDataAll::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::Selector**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLMessageEnd**			

Operation **IUMLDataAll::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLSignal**			

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**			
	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		**IUMLGuiVisibleElement**			
	ipLinkedGuiElementCell		**IUMLNamedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	**in**	**IUMLData**			
	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**			

Operation **IUMLDataAll::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**strKind**	**in**	**string**
	return	**return**	**IUMLInputPin**

Operation **IUMLDataAll::SetNewChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLDataAll::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLDataAll::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLBehavior**			

Operation **IUMLDataAll::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewMapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOpaqueExpression**			

Operation **IUMLDataAll::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewMaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewMin**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewOperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInteractionConstraint**			

Operation **IUMLDataAll::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLParameterableElement**			

© 2014 Altova GmbH

Operation **IUMLDataAll::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewProtocol**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLProtocolStateMachine**			

Operation **IUMLDataAll::SetNewSelector**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	**in**	**IUMLInstanceSpecification**			
	return	**return**	**IUMLInstanceValue**			

Operation **IUMLDataAll::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	**in**	**string**			
	return	**return**	**IUMLLiteralString**			

Operation **IUMLDataAll::SetNewStateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

Operation **IUMLDataAll::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	**in**	**string**			
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::SetNewWhen**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTimeExpression**			

Operation **IUMLDataAll::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	**in**	**IUMLOperation**			
	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**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

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**			

© 2014 Altova GmbH

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::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	**in**	**int**			
	ipForDefiningFeature	**in**	**IUMLStructuralFeature**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

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**			
	strNewValue	**in**	**string**			
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::SetTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	**in**	**IUMLGuiTextLabel**			
	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**			

Operation **IUMLDataAll::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLInputPin**			

Operation **IUMLDataAll::Signature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateSignature**			

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**			

Operation **IUMLDataAll::Source**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityNode**			

Operation **IUMLDataAll::Sources**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Specific**

 © 2014 Altova GmbH

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLClassifier**			

Operation **IUMLDataAll::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLValueSpecification**			

Operation **IUMLDataAll::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Start**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLOccurrenceSpecification**			

Operation **IUMLDataAll::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLState**			

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**			

Operation **IUMLDataAll::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

Operation **IUMLDataAll::Stereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStereotype**			

Operation **IUMLDataAll::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

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**			

Operation **IUMLDataAll::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLGuiStyles**			

Operation **IUMLDataAll::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLStateMachine**			

Operation **IUMLDataAll::SubmachineStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SubVertices**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::SuperGroup**

parameter	name	direction	type	type modifier	multiplicity	default

© 2014 Altova GmbH

r	**return**	**return**	**IUMLActivityGroup**			

Operation **IUMLDataAll::SuperPartition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLActivityPartition**			

Operation **IUMLDataAll::SupplierDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

Operation **IUMLDataAll::Targets**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Template**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateableElement**			

Operation **IUMLDataAll::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateBinding**			

Operation **IUMLDataAll::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLTemplateParameter**			

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**			

Operation **IUMLDataAll::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLGuiTextLabelKind**			

Operation **IUMLDataAll::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

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**			

Operation **IUMLDataAll::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLConstraint**			

Operation **IUMLDataAll::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMUMLTransitionKind**			

Operation **IUMLDataAll::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default

 © 2014 Altova GmbH

r	**return**	**return**	**IUMLDataList**			

Operation **IUMLDataAll::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLDataAll::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLVertex**			

Operation **IUMLDataAll::Triggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

Operation **IUMLDataAll::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLType**			

Operation **IUMLDataAll::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	**in**	**ENUMUMLPredefinedElement**			
	return	**return**	**void**			

Operation **IUMLDataAll::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	**in**	**IUMLStereotype**			
	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**			

Operation **IUMLDataAll::UseCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLUseCase**			

Operation **IUMLDataAll::UseCases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**IUMLDataList**			

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**			

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**			

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**			

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**			

© 2014 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**			

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**			

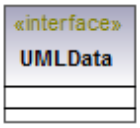
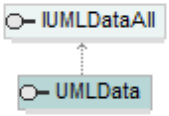
Operation **IUMLDataAll::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**int**			

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

UModelAPI - UMLData

Interface **UMLData**

diagram	
hierarchy	

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 09/04/12 14:02:42

Altova UModel® 2015

Anexos

20 Anexos

Estos anexos contienen datos técnicos sobre UModel e información importante sobre las licencias.

Información sobre la licencia

- Distribución electrónica de software
- Derechos de autor
- Contrato de licencia del usuario

© 2014 Altova GmbH

20.1 Información sobre licencias

Esta sección incluye:

- información sobre la distribución de este producto de software.
- información sobre la activación del software y modelo de licencias.
- información sobre los derechos de propiedad intelectual relacionados con este producto de software.
- el contrato de licencia para el usuario final que rige el uso de este producto de software

Los términos del contrato de licencia que aceptó al instalar el producto de software son vinculantes, por lo que rogamos lea atentamente toda esta información.

20.1.1 Distribución electrónica de software

Este producto está disponible por distribución electrónica de software, un método de distribución que ofrece ventajas únicas:

- Puede evaluar el software de forma totalmente gratuita antes de decidir si compra el producto.
- Si decide comprarlo, puede hacer un pedido en línea en el sitio web de Altova y conseguir en pocos minutos el software con licencia.
- Si realiza el pedido en línea, siempre recibirá la versión más reciente de nuestro software.
- El paquete de instalación del producto incluye un sistema de ayuda en pantalla totalmente integrado. La versión más reciente del manual del usuario está disponible en www.altova.com (i) en formato HTML y (ii) en formato PDF para descargar e imprimir si lo desea.

Período de evaluación de 30 días
Después de descargar el producto de software, puede probarlo de forma totalmente gratuita durante un plazo de 30 días. Pasados unos 20 días, el software empieza a recordarle que no tiene una licencia. El mensaje de aviso aparece una sola vez al iniciarse la aplicación. Si desea usar el programa una vez pasado el plazo de 30 días, deberá comprar una licencia permanente y aceptar el contrato de licencia de software de Altova , que se entrega en forma de código clave. La licencia puede comprarse directamente en la tienda en línea del sitio web de Altova. Después de comprar la licencia recibirá el código clave, que debe introducir en el cuadro de diálogo "Activación del software" para desbloquear el producto de forma permanente.

Distribuir la versión de evaluación a otros usuarios de su organización
Si desea distribuir la versión de evaluación en la red de su compañía o si desea usarlo en un PC que no está conectado a Internet, solamente puede distribuir los programas de instalación (siempre y cuando no se modifiquen de forma alguna). Todo usuario que acceda al instalador debe solicitar su propio código clave de evaluación (de 30 días). Una vez pasado este plazo de 30 días, todos los usuarios deben comprar también una licencia para poder seguir usando el producto.

Para más información consulte el contrato de licencia de software de Altova que aparece al final de esta sección.

 © 2014 Altova GmbH

20.1.2 Activación del software y medición de licencias

Durante el proceso de activación del software de Altova, puede que la aplicación utilice su red interna y su conexión a Internet para transmitir datos relacionados con la licencia durante la instalación, registro, uso o actualización del software a un servidor de licencias operado por Altova y para validar la autenticidad de los datos relacionados con la licencia y proteger a Altova de un uso ilegítimo del software y mejorar el servicio a los clientes. La activación es posible gracias al intercambio de datos de la licencia (como el sistema operativo, la dirección IP, la fecha y hora, la versión del software, el nombre del equipo, etc.) entre su equipo y el servidor de licencias de Altova.

Su producto incluye un módulo integrado de medición de licencias que le ayudará a evitar infracciones del contrato de licencia para el usuario final. Puede comprar una licencia de un solo usuario o de varios usuarios para el producto de software y el módulo de medición de licencias se asegura de que no se utiliza un número de licencias mayor al permitido.

Esta tecnología de medición de licencias usa su red de área local (LAN) para comunicarse con las instancias de la aplicación que se ejecutan en equipos diferentes.

Licencia de un solo usuario
Cuando se inicia la aplicación, se inicia el proceso de medición de licencias y el software envía un breve datagrama de multidifusión para averiguar si hay otras instancias del producto activas en otros equipos del mismo segmento de red al mismo tiempo. Si no recibe ninguna respuesta, la aplicación abre un puerto para escuchar a otras instancias de la aplicación.

Licencia de varios usuarios
Si se usa más de una instancia de la aplicación dentro de la misma red LAN, estas instancias se comunicarán entre ellas al iniciarse. Estas instancias intercambian códigos claves para que ayudarle a no sobrepasar por error el número máximo de licencias concurrentes. Se trata de la misma tecnología de medición de licencias que suele utilizarse en Unix y en otras herramientas de desarrollo de bases de datos. Gracias a ella puede comprar licencias de varios usuarios de uso concurrente a un precio razonable.

Las aplicaciones se diseñaron de tal modo que envían pocos paquetes pequeños de red y no cargan demasiado su red. Los puertos TCP/IP (2799) utilizados por su producto de Altova están registrados oficialmente en la IANA (para más información consulte el sitio web de la IANA www.iana.org) y nuestro módulo de medición de licencias es una tecnología probada y eficaz.

Si usa un servidor de seguridad, puede notar las comunicaciones del puerto 2799 entre los equipos que ejecutan los productos de Altova. Si quiere, puede bloquear ese tráfico, siempre y cuando esto no resulte en una infracción del contrato de licencia.

También notará que su producto de Altova ofrece varias funciones prácticas si está conectado a Internet. Estas funciones no tienen nada que ver con la tecnología de medición de licencias.

20.1.3 Derechos de propiedad intelectual

El software de Altova y sus copias (si tiene permiso de Altova para realizar copias) es propiedad intelectual de Altova y de sus proveedores. La estructura, la organización y el código del software se considera secreto comercial e información confidencial de Altova y de sus proveedores. El software está protegido por las leyes de derechos de autor, como la ley de derechos de autor de EE UU, tratados internacionales y la legislación vigente del país donde se utiliza, entre otras. Altova conserva los derechos de propiedad de todas las patentes, derechos de autor, secretos comerciales, marcas registradas y otros derechos de propiedad intelectual pertenecientes al software y los derechos de propiedad de Altova abarcan también imágenes, fotografías, animaciones, vídeos, audio, música, texto y otros applets incorporados al software y al material impreso que viene con el software. Las notificaciones de infracción de dichos derechos de autor debe enviarse al agente de derechos de autor de Altova, cuyos datos de contacto aparecen en el sitio web de Altova.

El software de Altova contiene software de terceros que también está protegido por las leyes de propiedad intelectual, incluida, entre otras, la legislación de derechos de autor mencionada en http://www.altova.com/es/legal_3rdparty.html.

Los demás nombres y marcas registradas son propiedad de sus respectivos propietarios.

© 2014 Altova GmbH

20.1.4 Contrato de licencia para el usuario final de Altova

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensor:
Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the Software in your possession or control. You may print a copy of this Agreement as part of the installation process at the time of acceptance. Alternatively, a copy of this Agreement may be found at http://www.altova.com/eula and a copy of the Privacy Policy may be found at http://www.altova.com/privacy.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license, without the right to grant sublicenses, to install and use a copy of the Software on one compatible personal computer or workstation up to the Permitted Number of computers. Subject to the limitations set forth in Section 1(c), you may install and use a copy of the Software on more than one of your compatible personal computers or workstations if you have purchased a Named-User license. Subject to the limitations set forth in Sections 1(d) and 1(e), users may use the software concurrently on a network. The Permitted Number of computers and/or users and the type of license, e.g. Installed, Named-Users, and Concurrent-User, shall be determined and specified at such time as you elect to purchase the Software. Installed user licenses are intended to be fixed and not concurrent. In other words, you cannot uninstall the Software on one machine in order to reinstall that license to a different machine and then uninstall and reinstall back to the original machine. Installations should be static. Notwithstanding the foregoing, permanent uninstallations and redeployments are acceptable in limited circumstances such as if an employee leaves the company or the machine is permanently decommissioned. During the evaluation period, hereinafter defined, only a single user may install and use the software on one (1) personal computer or workstation. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Agreement governs your use of all of the software included in the Suite.

(ii) If you have licensed SchemaAgent, then the terms and conditions of this Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable, and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the purposes described in the accompanying documentation.

(iii) If you have licensed Software that enables users to generate source code, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile the complete generated code (comprised of the combination of the Restricted Source Code and the Unrestricted Source Code) into executable object code form, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer the Restricted Source Code to a third-party in the un-compiled form unless said third-party already has a license to the Restricted Source Code through their separate agreement with Altova. Notwithstanding anything to the contrary herein, you may not distribute, incorporate or combine with other software, or otherwise use the Altova Library modules or Restricted Source Code, or any Altova intellectual property embodied in or associated with the Altova Library modules or Restricted Source Code, in any manner that would subject the Restricted Source Code to the terms of a copyleft, free software or open source license that would require the Restricted Source Code or Altova Library modules source code to be disclosed in source code form. Notwithstanding anything to the contrary herein, you may not use the Software to develop and distribute other software programs that directly compete with any Altova software or service without prior written permission. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise expressly permitted in Section 1(j) reverse engineering of the Software is strictly prohibited as further detailed therein.

(iv) In the event Restricted Source Code is incorporated into executable object code form, you will include the following statement in (1) introductory splash screens, or if none, within one or more screens readily accessible by the end-user, and (2) in the electronic and/or hard copy documentation: "Portions of this program were developed using Altova® [name of Altova Software, e.g. MapForce® 2011] and includes libraries owned by Altova GmbH, Copyright © 2007-2011 Altova GmbH (www.altova.com)."

(b) Server Use for Installation and Use of SchemaAgent. You may install one (1) copy of the Software on a computer file server within your internal network solely for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. If you have licensed SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) Named-Use. If you have licensed the "Named-User" version of the software, you may install the Software on up to five (5) compatible personal computers or workstations of which you are the primary user thereby allowing you to switch from one computer to the other as necessary provided that only one (1) instance of the Software will be used by you as the Named-User at any

given time. If you have purchased multiple Named-User licenses, each individual Named-User will receive a separate license key code.

(d) Concurrent Use in Same Physical Network or Office Location. If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers in a commercial environment only, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time and further provided that the computers on which the Software is installed are on the same physical computer network. The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses. Each separate physical network or office location requires its own set of separate Concurrent User Licenses for those wishing to use the Concurrent User versions of the Software in more than one location or on more than one network, all subject to the above Permitted Number limitations and based on the number of users using the Software. If a computer is not on the same physical network, then a locally installed user license or a license dedicated to concurrent use in a virtual environment is required. Home User restrictions and limitations with respect to the Concurrent User licenses used on home computers are set forth in Section 1(g).

(e) Concurrent Use in Virtual Environment. If you have purchased Concurrent-User Licenses, you may install a copy of the Software on a terminal server (Microsoft Terminal Server or Citrix Metaframe), application virtualization server (Microsoft App-V, Citrix XenApp, or VMWare ThinApp) or virtual machine environment within your internal network for the sole and exclusive purpose of permitting individual users within your organization to access and use the Software through a terminal server, application virtualization session, or virtual machine environment from another computer provided that the total number of users that access or use the Software concurrently at any given point in time on such network, virtual machine or terminal server does not exceed the Permitted Number; and provided that the total number of users authorized to use the Software through the terminal server, application virtualization session, or virtual machine environment does not exceed ten (10) times the Permitted Number of users. In a virtual environment, you must deploy a reliable and accurate means of preventing users from exceeding the Permitted Number of concurrent users. Altova makes no warranties or representations about the performance of Altova software in a terminal server, application virtualization session, or virtual machine environment and the foregoing are expressly excluded from the limited warranty in Section 5 hereof. Technical support is not available with respect to issues arising from use in such environments.

(f) Backup and Archival Copies. You may make one (1) backup and one (1) archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(g) Home Use (Personal and Non-Commercial). In order to further familiarize yourself with the Software and allow you to explore its features and functions, you, as the primary user of the computer on which the Software is installed for commercial purposes, may also install one copy of the Software on only one (1) home personal computer (such as your laptop or desktop) solely for your personal and non-commercial ("HPNC") use. This HPNC copy may not be used in any commercial or revenue-generating business activities, including without limitation, work-from-home, teleworking, telecommuting, or other work-related use of the Software. The HPNC copy of the Software may not be used at the same time on a home personal computer as the Software is being used on the primary computer.

(h) Key Codes, Upgrades and Updates. Prior to your purchase and as part of the registration for the thirty (30) day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or updated copy in addition to the copy of the Software that it is replacing and whose license has terminated.

(i) Title. Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Agreement, are your property unless they are created using Evaluation Software, as defined in Section 4 of this Agreement, in which case you have only a limited license to use any output that contains generated program code (including Unrestricted Source Code) such as Java, C++, C#, VB.NET or XSLT and associated project files and build scripts, as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures only for the thirty (30) day evaluation period.

(j) Reverse Engineering. Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(k) Other Restrictions. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or as otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Agreement. You may not permit any use of or access to the Software by any third party in connection with a commercial service offering, such as for a cloud-based or web-based SaaS offering.

You will comply with applicable law and Altova's instructions regarding the use of the

 © 2014 Altova GmbH

Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Agreement and to ensure their compliance with these restrictions.

(I) NO GUARANTEE. THE SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOVA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER APPLICATIONS WHERE THE FAILURE OF THE SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND ANY DATA GENERATED OR PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOVA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY THIRD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE SOFTWARE IN YOUR USE.

2. INTELLECTUAL PROPERTY RIGHTS

You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that trademark. Altova®, XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, RaptorXML™, RaptorXML Server™, RaptorXML +XBRL Server™, Powered By RaptorXML™, FlowForce Server™, StyleVision Server™, and MapForce Server™ are trademarks of Altova GmbH. (pending or registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, Windows 7, and Windows 8 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer this Agreement, the Software and all

other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-release Software"), then this Section applies. In addition, this section applies to all evaluation and/or demonstration copies of Altova software ("Evaluation Software") and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU **"AS-IS" WITH NO WARRANTIES FOR USE OR PERFORMANCE,** AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD $50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Your license to use any output created with the Evaluation Software that contains generated program code (including Unrestricted Source Code) such as Java, C++, C, VB.NET or XSLT and associated project files and build scripts as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures terminates automatically upon the expiration of the designated evaluation period but the license to use such output is revived upon your purchase of a license for the Software that you evaluated and used to create such output. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) Limited Warranty and Customer Remedies. Altova warrants to the person or entity

that first purchases a license for use of the Software pursuant to the terms of this Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by Altova shall be substantially as described in Section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. Altova's and its suppliers' entire liability and your exclusive remedy shall be, at Altova's option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet Altova's Limited Warranty and which is returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or any other malicious external code. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does not apply to Evaluation and/or Pre-release Software.

(b) No Other Warranties and Disclaimer. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA OR ITS SUPPLIER'S BREACH OF WARRANTY. ALTOVA AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ALTOVA AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTOVA OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ALTOVA'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, Altova's liability shall be limited to the greatest extent permitted by law and the limitations or exclusions of warranties and liability contained herein do not prejudice applicable statutory consumer rights of person acquiring goods otherwise than in the course of business. The disclaimer and limited liability above are fundamental to this Agreement between Altova and you.

(d) Infringement Claims. Altova will indemnify and hold you harmless and will defend or settle any claim, suit or proceeding brought against you by a third party that is based upon a

claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova's legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova's legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA'S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT. This indemnity does not apply to situations where the alleged infringement, whether patent or otherwise, is the result of a combination of the Altova software and additional elements supplied by you.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional "Support & Maintenance Package(s)" ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph 6(a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30) day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, **you are eligible to receive the version of the Software edition** that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP's Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In

 © 2014 Altova GmbH

addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova's business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or update, which is made available at certain times at Altova's sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

Updating Software may require the updating of software not covered by this Agreement before installation. Updates of the operating system and application software not specifically covered by this Agreement are your responsibility and will not be provided by Altova under this Agreement. Altova's obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the Software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova's written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova's specified release level; or (v) use of the Software with other software without Altova's prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) License Metering. The Software includes a built-in license metering module that is designed to assist you with monitoring license compliance in small local networks. The metering module attempts to communicate with other machines on your local area network. You permit Altova to use your internal network for license monitoring for this purpose. This license metering module may be used to assist with your license compliance but should not be the sole method. Should your firewall settings block said communications, you must deploy an accurate means of monitoring usage by the end user and preventing users from using the Software more than the Permitted Number.

(b) License Compliance Monitoring. You are required to utilize a process or tool to ensure that the Permitted Number is not exceeded. Without prejudice or waiver of any potential violations of the Agreement, Altova may provide you with additional compliance tools should you be unable to accurately account for license usage within your organization. If provided with such a tool by Altova, you (a) are required to use it in order to comply with the terms of this Agreement and (b) permit Altova to use your internal network for license monitoring and metering and to generate compliance reports that are communicated to Altova from time to time.

(c) Software Activation. **The Software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova Master License Server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the Software and to improve customer service. Activation is based on the exchange of license related data between your computer and the Altova Master License Server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova, or an authorized reseller as part of an effort to activate or use the Software violates Altova's intellectual property rights as well as the terms of this Agreement. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms, the license management mechanism, or the Altova Master License Server violate Altova's intellectual property rights as well as the terms of this Agreement. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.**

(d) LiveUpdate. Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(e) Use of Data. The terms and conditions of the Privacy Policy are set out in full at http://www.altova.com/privacy and are incorporated by reference into this Agreement. By your acceptance of the terms of this Agreement and/or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Agreement and/or Privacy Policy at any time. You are encouraged to review the terms of the Privacy Policy as posted on the Altova Web site from time to time.

(f) Audit Rights. You agree that Altova may audit your use of the Software for compliance with the terms of this Agreement at any time, upon reasonable notice. In the event that such audit reveals any use of the Software by you other than in full compliance with the terms of this Agreement, you shall reimburse Altova for all reasonable expenses related to such audit in addition to any other liabilities you may incur as a result of such non-compliance.

(g) Notice to European Users. Please note that the information as described in paragraph 7(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc., a company located in Beverly, Massachusetts, U.S.A., or its subsidiaries or Altova's subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive

 © 2014 Altova GmbH

allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country's level of protection. By agreeing to this Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Agreement may be terminated (a) by your giving Altova written notice of termination; (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Agreement and fail to cure such breach within ten (10) days after notice from Altova; or (c) at the request of an authorized Altova reseller in the event that you fail to make your license payment or other monies due and payable. In addition the Agreement governing your use of a previous version of the Software that you have upgraded or updated is terminated upon your acceptance of the terms and conditions of the Agreement accompanying such upgrade or update. Upon any termination of the Agreement, you must cease all use of the Software that this Agreement governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(h), 1(i), 1(j), 1(k), 1(l), 2, 5, 7, 9, 10, 11, and 11 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Agreement. Manufacturer is Altova GmbH, Rudolfsplatz 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. U.S. GOVERNMENT ENTITIES

Notwithstanding the foregoing, if you are an agency, instrumentality or department of the federal government of the United States, then this Agreement shall be governed in accordance with the laws of the United States of America, and in the absence of applicable federal law, the laws of the Commonwealth of Massachusetts will apply. Further, and notwithstanding anything to the contrary in this Agreement (including but not limited to Section 5 (Indemnification)), all claims, demands, complaints and disputes will be subject to the Contract Disputes Act (41 U.S.C. §§7101 *et seq.*), the Tucker Act (28 U.S.C. §1346(a) and §1491), or the Federal Tort Claims Act (28 U.S.C. §§1346(b), 2401-2402, 2671-2672, 2674-2680), FAR 1.601(a) and 43.102 (Contract Modifications); FAR 12.302(b), as applicable, or other applicable governing authority. For the avoidance of doubt, if you are an agency, instrumentality, or department of the federal, state or local government of the U.S. or a U.S. public and accredited educational institution, then your

indemnification obligations are only applicable to the extent they would not cause you to violate any applicable law (e.g., the Anti-Deficiency Act), and you have any legally required authorization or authorizing statute.

11. THIRD PARTY SOFTWARE

The Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at http://www.altova.com/legal_3rdparty.html and are made a part of and incorporated by reference into this Agreement. By accepting this Agreement, you are also accepting the additional terms and conditions, if any, set forth therein.

12. JURISDICTION, CHOICE OF LAW, AND VENUE

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of the Commonwealth of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of the Commonwealth of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

13. TRANSLATIONS

Where Altova has provided you with a foreign translation of the English language version, you agree that the translation is provided for your convenience only and that the English language version will control. If there is any contradiction between the English language version and a translation, then the English language version shall take precedence.

14. GENERAL PROVISIONS

This Agreement contains the entire agreement and understanding of the parties with respect to

 © 2014 Altova GmbH

the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova's Web site for Altova and the address shown in Altova's records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party's right at a later time to enforce the same or any other term of this Agreement. This Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or threatened breach of this Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Agreement. If, for any reason, any provision of this Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Agreement, and this Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2013-10-17

Índice

.

.NET Framework,
incluir archivo, 166

1

1.4,
Java, 67

5

5.0,
Java, 67

A

Abrir,
archivos de UModel en VS. NET, 425
diagrama, 82
paquetes en vista jerárquica, 77
URL, 533
Abrir proyecto,
control de código fuente, 480
Absolutos,
vínculos absolutos y relativos, 194
Abstracta,
clase, 30
Acceso directo,
asignar/eliminar, 572
mostrar en información rápida, 574
tecla, 572
teclado, 572
Acelerar,
rendimiento, 184
Activar/desactivar,
bloqueo de la capa, 101
modo compacto, 330
Actividad, 227
agregar a estado, 227
agregar operación, 227
BPMN, 345
iconos, 511
Actividades,
agregar diagrama de actividades a transición, 227
diagrama de, 208
diagramas SysML, 381
Actor,
definido por el usuario, 20
personalizar, 20
Actualizar,
actualizaciones de estado en segundo plano, 473
archivo de proyectos, 153
modelo con la base de datos, 398
Actualizar estado,
control de código fuente, 505
Advertencia,
mensajes, 103
revisión de la sintaxis, 60
Agregar, 494
a Favoritos, 84
al control de código fuente, 494
diagrama a un paquete, 20
insertar / eliminar en Estructura del modelo, 77
mover, eliminar elementos en un diagrama, 104
paquete a un proyecto, 20
proyecto al control de código fuente, 494
proyecto nuevo, 134
vínculo al modelo, 116
Ajustar,
a la cuadrícula, 104
Ajuste,
líneas de ajuste mientras se arrastran objetos, 576
Alinear,
elementos, 108
elementos mientras se arrastran, 20
Alternar visibilidad,
elementos de la capa, 101
Anexos, 1182
Anotación,
documentación, 94
esquema XML, 330
texto (BPMN), 356
Anotación de texto,
BPMN, 356

API,
documentation, 660
overview, 661
Aplicación,
externa (argumentos), 569
Application object, 661
Árbol de diagramas,
panel, 82
Archivo, 533
abrir desde URL, 533
combinar archivos de proyecto, 168
ejemplo del tutorial, 14
nuevo / cargar / guardar con archivo de procesamiento por lotes, 130
ump, 134
Archivos,
de muestra, 122
Archivos ASPRJ, 603
Archivos binarios,
importar C# y Java, 148
Archivos de proyecto,
Borland - MS Visual Studio .Net, 542
Archivos locales,
vínculos absolutos o relativos, 194
Argumentos,
herramientas externas, 569
Arrastrar y colocar,
botón secundario del mouse, 114
crear asociaciones, 190
Artefacto,
agregar al nodo, 55
BPMN, 356
manifestación, 55
Asignar,
acceso directo a un comando, 572
estereotipo, 317
Asistente para la conexión,
importar bases de datos, 386
Asociación, 30, 104, 186
agregado/compuesto, 30
BPMN, 353
calificador, 186
caso de uso, 20
clase memberEnd, 186
crear mediante arrastrar/colocar, 190
definir el tipo, 186
entre clases, 30
grosor de la línea, 104
rol, 186
ver automáticamente, 186
ver durante la ingeniería de código, 67
ver propiedad como, 104
ver propiedad con tipo, 182
ver relaciones, 189
vínculos entre objetos, 43
Asociación entre tablas,
crear relaciones, 401
Atributo,
estereotipo, 317
mostrar / ocultar, 296
mostrar / ocultar valores etiquetados, 317
seleccionar en Estructura del modelo, 86
ventana de finalización automática, 576
Attribute,
coloring, 302
Automática,
respuesta automática a mensajes, 269
Automáticamente,
agregar operación, 227
ver automáticamente las asociaciones, 186
Automático,
hipervínculo, 116
Ayuda, 584

B

Ball and socket,
notación de forma esférica, 296
Bank,
archivos de muestra, 122
Barra de herramientas, 568
activar/desactivar, 568
agregar comando, 567
crear nueva, 568
mostrar iconos de tamaño grande, 574
restaurar comandos, 568
Barras de herramientas,
restaurar, 556
Base,
clase, 37
Base de datos, 395, 398
agregar columna al modelo, 398
agregar índice al modelo, 395
agregar tabla nueva, 401

Base de datos, 395, 398
crear clases ClaveForánea, 395
escribir modelo en código / base de datos, 398
importar en UModel, 386
revisión de la sintaxis, 395
scripts de cambios durante la actualización, 398
tipos de BD compatibles, 386
Bases de datos,
compatibles, 386
conexiones JDBC, 392
BD,
correspondencia entre el modelo y la BD, 165
Binarios,
compatibilidad con binarios confusos, 148
Bloquear,
activar/desactivar bloqueo de la capa, 101
elementos de una capa, 96
Borland,
archivo de proyecto BSDJ, 542
BPMN, 344
artefactos, 356
asociación, 353
convertir 1.0 en 2.0, 344
objetos de flujo, 345
BSDJ,
proyecto Borland, 542
Buscar, 77, 539
elementos de modelado, 77, 539
elementos no utilizados, 77
pestañas de búsqueda, 76
y reemplazar, 539
Business Process Modeling Notation, 344
iconos, 528

C

C#,
código, 590
configurar importación, 142
correspondencia entre el modelo y el código, 162
error handling, 676
importar archivo binario, 148
C++,
código, 590
error handling, 676
Calificador,
asociación, 186
CallBehavior,
insertar, 209
CallOperation,
insertar, 209
Cambiar de nombre,
capa, 101
clasificador, 156
Cambiar de proveedor,
control de código fuente, 507
Cambiar nombres de clases,
efecto en el nombre del archivo de código, 158
Cambio de estado,
definir en una línea de tiempo, 288
Campo,
agregar al modelo, 398
Capas,
agregar al diagrama, 96
bloquear elemento, 96
eliminar, 96
restaurar, 96
Carpeta,
carpeta de ejemplos, 14
Carpetas,
obtener carpetas en control de código fuente, 487
Casilla de activación,
especificación de ejecución, 263
Caso de uso,
agregar, 20
asociación, 20
compartimiento, 20
multilínea, 20
Casos de uso,
diagrama de, 251
diagramas SysML, 385
iconos, 526
Catálogo,
archivo de catálogo, 576
Centrar,
elementos, 108
Cerrar,
todos los diagramas excepto los que están activos, 104
Ciclo de vida,
diagrama de, 287
iconos, 525
Clase, 137, 296
crear en espacio de nombres, 137
en diagramas de componentes, 49

© 2014 Altova GmbH

Clase, 137, 296
expandir y contraer compartimientos, 296
generar en espacio de nombres, 137
iconos, 513
interfaz con ball and socket (notación de forma esférica), 296
operación de clase (invalidar), 296
varias instancias en el diagrama, 296
Clase base,
insertar clases derivadas, 114
invalidar, 296
Clases,
abstractas y concretas, 30
agregar, 30
agregar operaciones, 30
agregar propiedades, 30
asociaciones, 30
base, 37
cambios de nombre (sincronización), 158
derivadas, 37
diagrama de, 296
diagramas, 30
habilitar ventana de finalización automática, 576
insertar clases derivadas, 114
sincronización, 153
Clasificador,
cambiar de nombre, 156
nuevo, 156
restringir, 178
Class,
syntax coloring, 302
ClaveForánea,
crear en diagrama de base de datos, 395
Code,
generate sequence diagram from, 666
generate sequence diagram manually, 667
Código, 158
agregar código a diagrama de secuencia, 283
código Java y nombres de archivos de clases, 158
condiciones para la generación, 159
directorio de destino, 60
generar código a partir de diagramas de secuencia, 279
generar diagrama de secuencia a partir de código, 274
generar varios diagramas de secuencia a partir de código, 277
ingeniería de ida y vuelta, 60
predeterminado, 576
refactorizar, 158
requisitos, 60
sincronización, 153
Código C#,
correspondencia con elementos de UModel, 162
Código de la BD,
correspondencia con elementos de UModel, 165
Código de refactorización,
nombres de clases (sincronizar), 158
Código fuente,
importar, 142
Código Java,
correspondencia con elementos de UModel, 161
Código VB.NET,
correspondencia con elementos de UModel, 164
Código XML Schema,
correspondencia con elementos de UModel, 163
Colaboración,
diagrama de estructura de un compuesto, 308
Color,
syntax coloring - enable/disable, 302
Color de fondo,
transparente, 325
Columna,
agregar al modelo, 398
Comando,
agregar a barra de herramientas / menú, 567
eliminar de menú, 573
menú contextual, 573
procesamiento de la línea de comandos, 126
restaurar menú, 573
Comandos,
línea de comandos: nuevo / cargar / guardar, 130
procesamiento de la línea de comandos, 130
COM-API,
documentation, 660
CombinaciónDePaquete, 314
Combinar,
código con el modelo, 60, 542
modelo con el código, 542
omitir directorio, 576
proyectos, 168
sincronizar en VS .NET, 425
Combinar en,
capa, 101
Comentarios,
documentación, 94
ver comentarios del código fuente en el diagrama, 104
Compacto,
modo (activar/desactivar), 330
Comparación,

Comparación,
configurar DiffDog, 473
Comparar archivos de código fuente, 502
Compartimento,
contenedor, 355
Compartimiento,
expandir uno / varios, 296
Compartir,
desde el control de código fuente, 498
paquete y diagrama, 175
Compatibilidad,
actualizar proyectos, 153
Complemento,
configuración de la interfaz del usuario, 648
editor de UModel, vista, 447
importar carpeta de ejemplos, 450
registrar, 645
registro, 646
UModel para Eclipse, 438
XMLSPY, 638
Complemento de UModel,
generación de código, 456
importar carpeta de ejemplos, 450
Complemento para Eclipse,
generación de código, 456
Componentes, 49
diagrama, 49
diagrama de, 310
iconos, 516
insertar clase, 49
realización, 49
Comportamiento,
diagramas de, 207
diagramas SysML, 381
Composición,
crear asociación, 30
Compuerta,
BPMN, 345
Comunicación,
diagrama de, 252
iconos, 514
Concreta,
clase, 30
Condiciones,
para la generación de código, 159
Conexión,
JDBC, 392
Conexiones JDBC, 392
Configuración,
control de código fuente, 576
Configurar,
interfaz gráfica XMLSPY, 648
sincronización, 153
Confusos,
compatibilidad con binarios confusos, 148
Contenedor,
compartimento, 355
Contraer,
compartimientos de clase, 296
Contraído,
subproceso, 351
Contrato de licencia para el usuario final, 1183, 1187
Control de código fuente, 458
abrir proyecto, 480
actualizar estado, 505
agregar al control de código fuente, 494
anular desprotección, 492
cambiar de proveedor, 507
comandos, 479
desproteger, 489
ejecutar interfaz nativa, 506
habilitar / deshabilitar, 483
mostrar diferencias, 502
mostrar historial, 499
obtener archivo, 485
obtener la versión más reciente, 484
opciones / configuración, 576
propiedades, 504
proteger, 491
quitar del, 497
Control de versiones,
comandos, 479
herramienta de comparación DiffDog, 473
procedimiento de instalación, 465
subproyectos y trabajo en equipo, 414
Controlador,
crear relaciones, 190
Controladores de eventos,
en el proyecto de scripting, 620
información general, 609
Convergencia,
crear en diagrama de actividades, 211
Convertir,
BPMN 1.0 en 2.0, 344
Copiar,
pegar en Estructura del modelo, Árbol de diagramas, 111

© 2014 Altova GmbH

Correo electrónico,
enviar proyecto, 533
Correspondencia,
entre elementos C# y elementos del modelo, 162
entre elementos de la BD y elementos del modelo, 165
entre elementos de VB.NET y elementos del modelo, 164
entre elementos de XML Schema y elementos del modelo, 163
entre elementos Java y elementos del modelo, 161
CPU,
carga - acelerar actualización de estado en segundo plano, 458
CR/LF,
en el archivo ump al guardarlo, 134
Crear,
esquema XML, 340
métodos getter / setter, 296
un proyecto desde cero (ingeniería de código), 137
CSPROJ - CSDPROJ,
MS Visual Studio .Net, 542
Ctrl+Barra espaciadora,
finalización automática a petición, 108
Cuadrícula,
ajustar a la cuadrícula, 104
líneas de ajuste, 576
líneas de ajuste mientras se arrastran objetos, 20
CVS, 458

D

Declaraciones globales, 613
información general, 609
Definidas por el usuario,
plantillas SPL, 153
Definido por el usuario,
actor, 20
Definidos,
símbolos definidos (importar código), 142
Definidos por el usuario,
estilos de estereotipo, 323
Dependencia,
incluir, 20
uso, 49
ver relaciones, 189
Depurar macros, 623
Derivada,
clase, 37
Derivadas,
insertar clases derivadas, 114
Desbloquear,
elementos de la capa, 101
Descargar proyecto de control de código fuente, 480
Desde cero,
crear proyecto en Visual Studio, 432
Deshabilitar el control de código fuente, 483
Deshacer desprotección, 492
Desproteger, 489
Desviar,
hipervínculos, 116
Diagram,
de perfil, 316
Diagram - sequence,
generate from code, 666
generate manually from code, 667
generate sequence diagram from code, 667
Diagrama, 296, 576
abrir, 82
agregar actividad a transición, 227
agregar código a diagrama de secuencia, 283
BPMN, 344
compartir paquete y diagrama, 175
de actividades, 208
de base de datos (importación), 386
de bloque interno SysML, 373
de casos de uso, 251
de casos de uso SysML, 385
de ciclo de vida, 287
de clases, 296
de componentes, 310
de comportamiento SysML, 381
de comunicación, 252
de esquema XML, 329, 330
de esquema XML (importación), 330
de estructura de un compuesto, 308
de estructura SysML, 369
de implementación, 311
de máquina de estados, 225
de máquina de estados SysML, 384
de objetos, 312
de paquetes, 313
de perfil y estereotipos, 317
de secuencia, 261
de secuencia SysML, 383
elementos de restricción, 77

Diagrama, 296, 576
estilos, 88
generar código a partir de diagramas de secuencia, 279
generar diagrama de dependencias entre paquetes, 313
global de interacción, 256
guardar como PNG, 533
guardar elementos como mapa de bits, 539
guardar los diagramas abiertos con el proyecto, 576
iconos, 510
insertar diagrama SysML, 369
omitir elementos de archivos incluidos, 576
usar capas, 96
varias instancias de la clase, 296
Diagrama de actividades,
crear rama / convergencia, 211
elementos, 213
insertar elementos, 209
Diagrama de bloque interno,
SysML, 373
Diagrama de ciclo de vida,
cambiar de un tipo a otro, 288
ciclo de vida, 288
evento/estímulo, 292
insertar elementos, 288
línea de vida, 288
marca de graduación, 291
mensaje, 294
restricciónDeDuración, 292
restricciónDeTiempo, 293
valor general de la línea de vida, 288
Diagrama de comunicación,
generar a partir de un diagrama de secuencia, 252
Diagrama de definición de bloques,
SysML, 370
Diagrama de paquetes,
insertar elementos, 314
SysML, 378
Diagrama de requisitos,
SysML, 380
Diagrama de secuencia, 274
agregar código a, 283
fragmento combinado, 265
generar a partir de código, 274
generar a partir de un diagrama de comunicación, 252
generar código a partir de, 279
generar diagrama de secuencia a partir de métodos getter/setter, 277
generar varios diagramas de secuencia a partir de código, 277
insertar elementos, 262
invariante de estado, 269
línea de vida, 263
mensajes, 269
nombres de operación que se deben omitir, 274
puerta, 268
uso de interacción, 268
Diagrama global de interacción,
insertar elementos, 257
Diagramas, 206
ajustar el tamaño, 104
cerrar todos excepto los que están activos, 104
de comportamiento, 207
de estructura, 295
hipervínculo, 116
panel, 104
pegar, 111
peso de las líneas, 104
propiedades, 104
ver comentarios del código fuente, 104
DiffDog,
configurar para comparación, 473
Directa,
ingeniería, 159
Directorio,
cambiar ubicación del proyecto, 134
carpeta de ejemplos, 14
importar, 67
importar código desde, 142
omitir durante la combinación, 576
para generación de código, 60
usar espacio de nombres en la ruta de acceso, 60
Directorio de trabajo,
control de código fuente, 480
Diseño, 553
Disparador,
definir disparador de transición, 227
Distribución,
de productos de software de Altova, 1183, 1184, 1186
División,
evitar división entre las páginas, 533
Documentación, 94, 194
anotación, 94
generar proyecto UML, 194
pestaña, 94, 95
vínculos relativos, 194
Documento,
vincular, 116

© 2014 Altova GmbH

Dos bandas,
fusión del proyecto, 169
Dot,
propiedad, 190

E

Eclipse, 453
complemento de UModel, 438
diferencias entre la versión independiente y la versión para Eclipse, 453
generación de código, 456
importar ejemplos de UModel, 450
iniciar complemento de UModel, 445
instalar el complemento de UModel, 447
sincronizar modelo y código, 453
Edición, 539
Editor de scripting,
descripción de la interfaz gráfica del usuario, 605
ventana principal, 605
Ejecutar,
script de cambios en la BD, 398
Ejecutar interfaz nativa, 506
Ejemplos,
archivos de ejemplo, 122
carpeta del tutorial, 14
Elemento,
agregar a Favoritos, 84
agregar restricción a, 77
asociaciones durante la importación, 67
estilos, 88
generar documentación, 194
guardar elemento seleccionado como mapa de bits, 539
propiedades, 86
restricción, 77
ver jerarquía, 91
vincular, 116
Elementos,
alinear, 108
cortar, copiar, pegar, 111
insertar, 114
insertar en diagrama de máquina de estados, 226
mover a una capa, 96
omitir elementos de archivos incluidos, 576
relaciones, 186
seleccionar todos en una capa, 96
Elementos no utilizados,
lista, 77
Elementos que faltan,
lista, 77
Elimina,
acceso directo, 572
Eliminar, 567
barra de herramientas, 568
capas, 96
comando de menú contextual, 573
comando de una barra de herramientas, 567
de Favoritos, 84
icono de una barra de herramientas, 567
relaciones entre clases, 186
En cascada,
estilos, 88
Enlace,
plantilla, 181
Entorno de scripting, 600
resumen, 602
Enumeración,
valor predeterminado, 321
y estereotipos, 321
Enviar por correo electrónico,
proyecto, 533
Error,
mensajes, 103
revisión de la sintaxis, 60
Error handling,
general description, 676
Espaciar,
en horizontal, 108
Espacio de nombres,
crear una clase en, 137
raíz de espacio de nombres Java, 159
usar para la generación de código, 60
Especializar,
generalizar, 37
Especificación de ejecución,
línea de vida, 263
Esquema,
crear esquema XML, 340
generador de código, 590
tipo de datos (definir), 337
XML, 330
XML (importación), 330
Esquema XML,
anotación, 330

© 2014 Altova GmbH

Esquema XML,
crear/generar, 340
diagrama de, 330
iconos, 527
insertar elementos, 337
modelo de contenido, 337
Estado, 227
actualizaciones en segundo plano, 473
agregar actividad, 227
definir transición entre, 227
insertar estado simple, 227
ortogonal, 233
submáquina, 233
Estado compuesto, 233
agregar región, 233
Estado de submáquina,
agregar punto de entrada/salida, 233
Estereotipo,
agregar a diagrama de perfil, 317
asignar, 317
atributos (definir), 317
estilos definidos por el usuario, 323
icono personalizado, 325
memberEnd, 317
perfiles, 317
valor etiquetado predeterminado, 321
y enumeración, 321
Estereotipos,
definición, 316
Estilos,
en cascada, prioridad, 88
estereotipo definido por el usuario, 323
pestaña, 88
varias selecciones, 88
Estructura,
diagrama de estructura SysML, 369
diagramas de, 295
Estructura de un compuesto,
diagrama de, 308
iconos, 515
insertar elementos, 308
Estructura del modelo,
abrir paquetes, 77
crear hipervínculo a un elemento de, 116
panel, 77
seleccionar atributo en, 86
Etiquetado,
valor (predeterminado), 321
Etiquetados,
valores, 317
valores (definición), 316
valores (ocultar/mostrar atributos), 317
Etiquetas,
identificadores ID y UUID, 410
Etiquetas de texto,
ver / ocultar, 189
Evento,
BPMN, 345
Evento/estímulo,
diagrama de ciclo de vida, 292
Eventos, 620
y controladores de eventos, 612
Eventos de la aplicación, 620
Eventos de la vista Authentic, 620
Eventos de la vista Cuadrícula, 620
Eventos de la vista Texto, 620
Eventos del documento, 620
Excepción,
agregar excepción generada, 296
operación Java, 142
Excepciones generadas, 142
agregar, 296
Expandido,
subproceso, 350
Expandir,
contraer paquetes, 77
todos los compartimientos de clase, 296
Exportar,
como XMI, 410
Extensión,
XMI, 410

F

Favoritos,
panel, 84
Finalización automática, 108
a petición (Ctrl+Barra espaciadora), 108
función, 30
modo único / modo múltiple, 108
ventana (tipos), 108
ventana para edición de clases, 576
Firma,
plantilla, 178, 180

Flujo,
condicional, 353
mensaje, 353
predeterminado, 353
secuencia, 353
Flujo condicional, 353
Flujo continuo,
modelar / transmitir por secuencias de actividades SysML, 381
Flujo de secuencia, 353
Flujo de trabajo,
proyecto, 134
Flujo del elemento,
crear, 373
Flujo del mensaje, 353
Flujo predeterminado, 353
Flujos,
crear flujo del elemento, 373
Formato,
estereotipo definido por el usuario, 323
ventana de finalización automática, 108
Formularios,
crear nuevos, 614
del proyecto de scripting, 614
establecer la secuencia de tabulación de los objetos, 615
información general, 609
invocar, 612
nombrar, 614
propiedades, 614
y comandos integrados, 625
y control de eventos, 618
y objetos de formulario, 615
Fragmento combinado, 265
Fuente,
comentarios del código fuente (ver en el diagrama), 104
Fusión,
a 2 bandas, 169
a 3 bandas, 170
a 3 bandas manual, 172

G

Generación de código, 60
usar un espacio de nombres como directorio, 60
Generador de código, 590
Generalizar,
especializar, 37
Generar,
código a partir de un esquema, 590
diagrama de secuencia a partir de código, 274
diagrama de secuencia a partir de diagrama de comunicación, 252
documentación del proyecto UML, 194
esquema XML, 340
RealizacionesDeComponente automáticamente, 156
respuesta a mensajes automáticamente, 269
varios diagramas de secuencia a partir de código, 277
Generate,
sequence diagram from code, 666
Generate manually,
sequence diagram from code, 667
Get,
métodos getter / setter, 296
Getter / Setter,
generar diagrama de secuencia a partir de métodos getter/setter, 277
Grupo,
BPMN, 356
Guardar,
archivos de subproyectos, 416
diagrama como imagen, 533
elementos como mapas de bits, 539

H

Habilitar,
líneas de ajuste mientras se arrastran objetos, 576
Habilitar el control de código fuente, 483
Herramientas, 556
agregar al menú Herramientas, 569
opciones, 576
Hipervínculo, 116
automático, 116
Historial,
mostrar, 499
HRESULT,
and error handling, 676

I

Icono,

Icono,
actividad, 511
agregar a barra de herramientas / menú, 567
Business Process Modeling Notation, 528
caso de uso, 526
ciclo de vida, 525
clase, 513
componentes, 516
comunicación, 514
esquema XML, 527
estructura de un compuesto, 515
icono de estereotipo personalizado, 325
implementación, 517
interacción global, 518
máquina de estados, 524
mostrar iconos de tamaño grande, 574
objeto, 519
paquete, 520
secuencia, 523
Iconos,
visibilidad, 296
Iconos de los diagramas de UModel, 510
ID,
identificadores ID y UUID, 410
Ida y vuelta,
código - modelo - código, 67
ingeniería, 60
modelo - código - modelo, 60
Igualar,
alto / ancho / tamaño, 108
Implementación,
diagrama, 55
diagrama de, 311
iconos, 517
ImportaciónDePaquete, 314
Importar, 67, 410
archivo XMI, 410
archivos binarios, 148
asociación de elementos, 67
base de datos SQL, 386
código fuente, 142
directorio, 67
en relación al archivo UMP, 142
esquema XML, 330
proyecto, 142
proyecto C#, 142
proyecto de origen, 67
XMI generado con UModel, 410
Imprimir,
vista previa, 533
Incluir, 166, 175
.NET Framework, 166
cambiar estado, 175
compartir paquete y diagrama, 175
dependencia, 20
proyecto de UModel, 166
subproyectos en el proyecto principal, 416
Índice,
agregar a la tabla, 395
Información legal, 1183
Información rápida,
mostrar accesos directos en, 574
ver, 574
Información sobre derechos de autor, 1183
Ingeniería de código, 67
crear un proyecto desde cero, 137
generar RealizacionesDeComponente, 156
importar directorio, 67
mover archivo de proyecto a una ubicación nueva, 134
ver asociaciones, 67
Iniciar,
complemento para Eclipse, 445
con el proyecto anterior, 576
UModel, 16
Insertar, 114, 209
acción (CallBehavior), 209
acción (CallOperation), 209
con..., 114
elementos, 114
elementos en diagrama de paquetes, 314
elementos en diagrama global de interacción, 257
elementos en diagramas de ciclo de vida, 288
elementos en estructura de un compuesto, 308
estado simple, 227
Instalación,
carpeta de ejemplos, 14
Instalador,
multiusuario, 14
Instalar,
sistemas de control de versiones, 465
Instancia,
diagrama, 43
objeto, 43
varias clases, 296
Integración,

Integración,
paquete de integración para el complemento para VS .NET, 424
Integrar,
proyecto de UModel en VS .NET, 432
subproyectos en el proyecto principal, 416
Inteligente,
finalización automática, 30
Interacción,
diagrama global de, 256
Interacción global,
iconos, 518
Interfaz, 296
ball and socket (notación de forma esférica), 296
implementar, 296
Interfaz del usuario, 76
configurar con ayuda del complemento, 648
Introducción, 6
Invalidar,
clase base, 296
operaciones de clase, 296
plantillas SPL predeterminadas, 153
Invariante de estado, 269
Ir a,
línea de vida, 263

J

Java,
código, 590
código y nombres de archivos de clases, 158
correspondencia entre el modelo y el código, 161
excepción, 142
importar archivo binario, 148
raíz de espacio de nombres, 159
versiones compatibles, 67
JavaDocs, 94
JavaScript,
error handling, 676
Jerarquía,
ver todas las relaciones, 91
Jerarquía del sistema,
diagrama de paquetes SysML, 378
Jerarquías (diagrama),
niveles en la documentación, 194

L

Lenguaje,
cambiar el lenguaje de scripting, 612
Lenguaje de scripting, 612
Licencia, 1187
información sobre, 1183
Licencia del producto de software, 1187
Limitar,
elementos de restricción, 77
Línea,
ortogonal, 49
peso/grosor en los diagramas, 104
Línea de tiempo,
definir cambios de estado, 288
Línea de vida,
atributos, 263
ir a, 263
propiedad de tipo, 263
valor general, 288
Línea nueva,
en línea de vida, 252
operandoDeInteracción, 265
Líneas,
de ajuste, 576
formato, 43
Líneas de ajuste, 20
Lista,
elementos no utilizados, 77
Llamada,
mensaje, 269
Llamada del mensaje,
operación ir a, 269

M

Macros,
crear con el editor de scripts, 621
definir como macros automáticas en el editor de scripts, 621
depurar, 623
editar con el editor de scripts, 621
ejecutar, 612, 622
funciones para, en declaraciones globales, 613

Macros,
información general, 609
instrucciones de uso en el proyecto de scripting, 621
Manifestación,
artefacto, 55
Manual,
sincronización manual en Eclipse, 453
sincronización manual en VS .NET, 427
Mapa de bits,
guardar elementos como, 539
Máquina de estados,
diagrama de, 225
diagrama SysML, 384
elementos, 243
estados compuestos, regiones, 233
estados, actividades, transiciones, 227
iconos, 524
insertar elementos, 226
Marca de graduación,
diagrama de ciclo de vida, 291
Marcar,
elementos de una capa, 96
Marco del diagrama,
ver título del diagrama UML, 108
Medición de licencias,
en los productos de Altova, 1185
Mejorar,
rendimiento, 184
Member end,
estereotipo, 317
MemberEnd,
asociación, 186
Mensaje, 269
crear objeto, 269
diagrama de ciclo de vida, 294
flechas, 269
insertar, 269
llamada, 269
mover, 269
numeración, 269
operación ir a, 269
Mensajes,
panel, 103
Menú, 573
agregar / eliminar comando, 567
agregar menú a, 569
archivo, 533
ayuda, 584
diseño, 553
edición, 539
eliminar comandos de, 573
herramientas, 556
personalizar, 573
predeterminado/XMLSPY, 573
proyecto, 542
ventanas, 583
vista, 555
Menú contextual,
comandos, 573
Metadatos,
salida XMI, 410
Método,
agregar excepción agregada, 296
generar diagrama de secuencia a partir de métodos, 274
generar diagrama de secuencia a partir de métodos getter/setter, 277
generar varios diagramas de secuencia a partir de métodos, 277
Métodos,
getter / setter, 296
MisDocumentos,
archivos de ejemplo, 14
Modelado,
mejorar el rendimiento, 184
Modelar,
bases de datos, 386
Modelo,
agregar vínculo, 116
cambiar el nombre de las clases (efecto en Java), 158
Modelo con el código,
ver asociaciones, 67
Modelo de contenido,
del esquema XML, 337
Modo múltiple,
finalización automática, 108
Modo único,
finalización automática, 108
Mostrar,
ocultar slot, 296
ocultar valores etiquetados / atributos, 317
valores etiquetados, 330
Mostrar / ocultar,
atributos, operaciones, 296
Mostrar diferencias, 502
Mostrar historial, 499
Mouse,

Mouse,
copiar, pegar, 111
Mover,
elemento a una capa, 96
proyecto, 134
Mover las flechas de los mensajes, 269
MS Visual Source Safe, 458
MS Visual Studio .Net,
archivo de proyecto CSPROJ - CSDPROJ, 542
MS VS .NET,
complemento de UModel, 424
Muestra,
archivos de muestra, 122
Multilínea,
caso de uso, 20
operandoDeInteracción, 265
texto del actor, 20
Multiusuario,
carpeta de ejemplos, 14

N

Navegar,
hipervínculo, 116
Nodo, 55
agregar, 55
agregar artefacto, 55
estilos, 88
propiedades de los bloques como, 370
Nombre,
de región (ver/ocultar), 233
Nota,
vincular, 116
Notas sobre programación,
del proyecto de scripting, 624
Novedades, 8
Nuevo,
clasificador, 156
Numeración,
mensajes, 269

O

Object model,
overview, 661
Objeto,
crear mensaje, 269
iconos, 519
vínculos (asociaciones), 43
Objeto de datos,
BPMN, 356
Objetos,
diagrama, 43
diagrama de, 312
Objetos de conexión, 353
Objetos de flujo, 345
Obtener archivo,
control de código fuente, 485
Obtener carpetas,
control de código fuente, 487
Obtener la versión más reciente, 484
Ocultar,
etiquetas de texto, 189
mostrar slot, 296
valores etiquetados, atributos, 317
ver capas, 96
Omitir, 576
directorios, 576
elementos en la lista, 576
nombres de operación, 274
Opción Macro automática, 621
Opciones, 576
control de código fuente, 576
herramientas, 576
proyecto, 183
Operación, 296
agregar automáticamente en actividad, 227
excepción, 142
invalidar, 296
ir a (en la llamada del mensaje), 269
mostrar / ocultar, 296
omitir en la generación de diagramas de secuencia, 274
plantilla, 182
ventana de finalización automática, 576
volver a utilizar, 37
Operaciones,
agregar, 30
Operador,
interacción, 265
Operador de interacción,
definir, 265
Operando,

© 2014 Altova GmbH

Operando,
interacción, 265
Operando de interacción,
multilínea, 265
Operation,
coloring, 302
Ordenar,
diagrama, 82
elementos en Estructura del modelo, 77
Ortogonal,
estado, 233
línea, 49
Overview,
of UModel API, 661
OwnedEnd,
asociación, 186

P

Página,
evitar división entre las páginas, 533
Paleta de objetos de formularios, 605
Panel,
Árbol de diagramas, 82
Estructura del modelo, 77
Favoritos, 84
Mensajes, 103
Vista general, 94
Panel Árbol del proyecto de scripting, 605
Panel Propiedades y eventos, 605
Paquete,
compartir, 175
crear un paquete de espacio de nombres, 137
expandir / contraer, 77
iconos, 520
perfil, 317
Paquetes,
diagrama de, 313
generar diagrama de dependencias entre, 313
Paramétrico, 376
diagrama SysML, 376
restricciones (definir), 376
Parámetro,
de procesamiento por lotes, 126
plantilla, 182
Parcial,
generar documentación parcial, 194
Pegar,
elementos en diagramas, 111
Perfil, 316
aplicación, 316
diagrama, 316
diagrama de, 316, 317
estereotipos, 317
Período de evaluación,
de los productos de software de Altova, 1183, 1184, 1186
Personalizado,
icono de estereotipo, 325
Personalizar, 567
actor, 20
comandos de las barras de herramientas / menús, 567
menú, 573
menú contextual, 573
Peso,
grosor de la línea, 104
Pestañas de búsqueda, 76
Plantilla,
enlace, 181
firma, 178, 180
operación/parámetro, 182
Plantillas,
SPL definidas por el usuario, 153
PNG,
guardar diagrama, 533
Poner en fila,
elementos, 108
Por lotes, 130
modo de procesamiento por lotes, 130
nuevo / cargar / guardar, 130
procesamiento, 126, 130
Predeterminada,
capa, 96
Predeterminadas,
plantillas SPL, 153
Predeterminado,
código de proyecto, 576
menú, 573
valor etiquetado, 317
Pretty print,
preparar proyecto para pretty print al guardarlo, 134
salida XMI, 410
Procesamiento por lotes,
modo de procesamiento por lotes, 130
Proceso,

© 2014 Altova GmbH

Proceso,
subproceso contraído, 351
subproceso expandido, 350
Property,
coloring, 302
Propiedad,
con tipo (ver), 182
crear valores iniciales, 373
de tipo línea de vida, 263
dot, 190
ver como asociación, 189
volver a utilizar, 37
Propiedades, 86
agregar, 30
control de código fuente, 504
Propiedades de los bloques,
como nodos, 370
Propiedades de los objetos de formulario, 615
Proteger, 491
Proveedor,
de control de código fuente, 458
seleccionar, 480
Proyecto, 542, 576
abrir último proyecto al iniciar UModel, 576
actualizar archivo de proyecto, 153
agregarlo al control de código fuente, 494
código predeterminado, 576
combinar, 168
crear, 134
crear paquete de espacio de nombres, 137
crear subproyectos, 416
enviar por correo electrónico, 533
estilos, 88
flujo de trabajo, 134
fusión a 2 bandas, 169
fusión a 3 bandas, 170
fusión a 3 bandas manual, 172
generar documentación, 194
guardar los diagramas abiertos, 576
guardar para pretty print, 134
importar, 142
incluir proyecto de UModel, 166
insertar paquete, 134
mover, 134
nuevo en Visual Studio, 432
opciones, 183
quitarlo del control de código fuente, 497
revisión de la sintaxis, 542
Proyecto de scripting,
controladores de eventos, 609
controladores de eventos de la aplicación, 620
declaraciones globales, 609
declaraciones globales del, 613
formularios, 609
formularios del, 614
instrucciones para crear, 612
macros, 609
macros del, 621
notas sobre programación, 624
y eventos, 620
Proyecto de scripting global,
de UModel, 603
Proyecto local, 480
Proyectos de scripting,
para proyecto de UModel, 603
para UModel, 603
Proyectos de scripting de Altova, 603
Puerta,
diagrama de secuencia, 268
Puerto, 373
conjugado (isConjugated), 373
crear atómico, 373
crear puerto de flujo, 373
crear puerto de flujo conjugado atómico, 373
insertar un puerto de flujo, 373
unir dos puertos, 373
Puerto de flujo,
crear puerto de flujo atómico conjugado, 373
definir la dirección de, 373
insertar, 373
Punto de entrada,
agregar a submáquina, 233
Punto de salida,
agregar a submáquina, 233
Puntos de vista,
diagrama de paquetes SysML, 378
PVCS Version Manager, 458

Q

Quitar,
de Favoritos, 84
del control de código fuente, 497

R

Raíz,
catálogo, 576
espacio de nombres Java, 159
sincronizar raíz / paquetes / clases, 153
Rama,
crear en diagrama de actividades, 211
Realización,
componente, 49
Realizaciones,
generar RealizacionesDeComponente, 156
RealizacionesDeComponente,
generación automática, 156
Rechazar cambios, 492
Recuperar archivo,
control de código fuente, 484
Recursos,
acelerar actualización de estado en segundo plano, 458
Referencia, 532
mostrar clase a la que se hace referencia, 104
Región,
agregar a estado compuesto, 233
nombre de región (ver/ocultar), 233
Registrar,
complemento, 646
Relación,
ver todas (pestaña Jerarquía), 91
Relaciones,
crear relaciones entre tablas, 401
crear relaciones usando los controladores, 190
elemento, 186
ver relaciones del modelo, 189
Relativos,
importar/guardar archivos relativos, 142
vínculos relativos en la documentación, 194
Rendimiento,
mejorar, 184
Repositorios, 458
Requisitos,
ingeniería directa, 159
Respuesta,
mensaje (generar automáticamente la respuesta), 269
Restaurar,
acceso directo, 572
barras de herramientas y ventanas, 556
capas, 96
comandos de la barra de herramientas, 568
comandos de menú, 573
Restricción,
agregar al diagrama, 77
asignar a varios elementos, 77
definir parámetros, 376
elemento, 77
revisar sintaxis, 542
RestricciónDeDuración,
diagrama de ciclo de vida, 292
RestricciónDeTiempo,
diagrama de ciclo de vida, 293
Restringir,
clasificadores, 178
Reutilizar,
subproyectos, 416
Revisar,
sintaxis del proyecto, 542
Rol,
asociación, 186
Ruta de acceso,
cambiar ubicación del proyecto, 134
carpeta de ejemplos, 14
importar rutas relativas al archivo UMP, 142
usar espacio de nombres en el código, 60

S

Salida,
archivo XMI de salida, 410
Salto de línea,
en el texto del actor, 20
SC,
syntax coloring, 302
Scripts de cambios,
actualizar bases de datos, 398
Secuencia,
diagrama de, 261
diagramas SysML, 383
iconos, 523
Segundo plano,
actualización de estado - aumentar el intervalo, 458
actualizaciones de estado, 473
Seleccionar atributo,

© 2014 Altova GmbH

Seleccionar atributo,
en Estructura del modelo, 86
Seleccionar todos,
elementos de una capa, 96
Sequence diagram,
generate from code, 666
generate manually from code, 667
Set,
métodos getter / setter, 296
Setter / Getter,
generar diagrama de secuencia a partir de métodos setter/getter, 277
Show,
tagged values, 334
Símbolos,
definidos (importar código), 142
iconos de visibilidad, 296
Sincronización,
cambiar nombres de las clases, 158
opciones de configuración, 153
Sincronizar,
código de combinación a partir del modelo, 60
combinar modelo con el código, 67
en Eclipse, 453
en la ubicación nueva, 134
en VS .NET, 427
modelo / código en VS .NET, 425
modelo con la BD, 398
nombre de las clases y el nombre del archivo de código, 158
raíz / paquetes / clases, 153
Sintaxis,
archivo de procesamiento por lotes, 126
errores y advertencias, 60
revisar, 60
revisar la sintaxis de los elementos de la BD, 395
revisar sintaxis del proyecto, 542
revisión de la sintaxis: mensajes, 103
Slot,
mostrar / ocultar, 296
Sobrescribir,
código con el modelo, 542
modelo con el código, 542
Socket,
Ball and socket (notación de forma esférica), 296
Solución,
archivos de UModel, 425
SPL,
plantillas SPL definidas por el usuario, 153
SQL,
importar en UModel, 386
StarTeam, 458
Subclase,
insertar en un diagrama, 114
Subproceso,
contraído, 351
expandido, 350
Subproyecto,
abrir / editar, 416
control de versiones, 414
crear, 416
guardar archivos de subproyectos, 416
opciones para trabajo en equipo, 414
Syntax coloring, 302
SysML, 369
diagrama de actividades, 381
diagrama de bloque interno, 373
diagrama de casos de uso, 385
diagrama de definición de bloques, 370
diagrama de estructura, 369
diagrama de máquina de estados, 384
diagrama de paquetes, 378
diagrama de requisitos, 380
diagrama de secuencia, 383
diagrama paramétrico, 376
diagramas de comportamiento, 381
insertar un diagrama SysML, 369
introducción, 369

T

Tabla,
agregar tabla nueva al modelo, 401
relaciones (crear), 401
Tagged values,
show, 334
Tamaño,
panel de diagramas, 104
Tecla de acceso rápido, 572
Tipo,
propiedad (ver), 182
propiedad de tipo línea de vida, 263
Tipo de datos,
definir en esquema, 337
Tipos,

Tipos,
finalización automática, 108
Tipos de datos,
agregar (finalización automática), 108
Tipos de filtros,
ventana de finalización automática, 108
Título del diagrama,
ver, 108
Todos,
expandir / contraer, 296
Trabajo en equipo,
subproyectos, 414
Transición, 227
agregar diagrama de actividades a, 227
definir disparador, 227
definir entre estados, 227
Transparente,
color de fondo, 325
Tres bandas,
fusión del proyecto, 170
fusión manual del proyecto, 172
Tutorial, 14
archivos de ejemplo, 14
carpeta de ejemplos, 14
objetivos, 14

U

Ubicación,
mover proyecto, 134
UML,
compartir diagrama UML, 175
diagramas, 206
iconos de visibilidad, 296
plantillas, 178
ver título del diagrama, 108
UModel,
API, 660
complemento para Eclipse, 438
complemento para VS .NET, 424
correspondencia con código C#, 162
correspondencia con código Java, 161
correspondencia con el código de la BD, 165
correspondencia con el código VB.NET, 164
correspondencia con el código XML Schema, 163
diferencias entre la versión independiente y la versión para Eclipse, 453
diferencias entre la versión independiente y la versión para VS .NET, 427
importar XMI generado, 410
iniciar, 16
UModel API, 660
overview, 661
UModel para Eclipse,
editor, vista perspectiva, 447
UMP,
abrir archivos UMP en VS .NET, 425
cambiar ubicación del proyecto, 134
extensión de archivo, 134
URL,
abrir archivo desde, 533
Uso,
dependencia, 49
Uso de interacción, 268
Usuario,
carpeta de ejemplos multiusuario, 14
UUID,
identificadores únicos universales, 410

V

Valor general de la línea de vida,
diagrama de ciclo de vida, 288
Valores,
etiquetados, 316, 317
etiquetados (mostrar), 330
Valores etiquetados,
mostrar, 330
Valores iniciales,
agregar a propiedad (SysML), 373
value,
tagged, show, 334
Variables,
argumentos de herramientas externas, 569
Varios elementos,
ver estilos, 88
VB.NET,
correspondencia entre el modelo y el código, 164
Velocidad,
acelerar actualización de estado en segundo plano, 458
Ventanas, 583

© 2014 Altova GmbH

Ventanas, 583
restaurar, 556
Ver, 91
capas, 96
etiquetas de texto, 189
o ajustar a la cuadrícula, 104
ocultar capas, 96
propiedad como asociación, 104, 182
relaciones del modelo, 189
todas las relaciones (pestaña Jerarquía), 91
u ocultar nombre de región, 233
varias instancias del elemento, 296
vista en forma de diagrama, 91
Vincular,
crear hipervínculo, 116
elemento de Estructura del modelo, 116
Vínculos,
relativos a la documentación, 194
Visibilidad,
iconos (seleccionar), 296
Vista, 555
Vista en forma de diagrama,
un solo conjunto de relaciones, 91
Vista general,
panel, 94
Visual Basic,
error handling, 676
Visual Studio,
crear proyecto desde cero, 432
VS .NET, 427
archivos de UModel, 425
complemento de UModel, 424
diferencias entre la versión independiente y la versión para VS .NET, 427
sincronizar modelo y código, 427

W

Web,
hipervínculo, 116

X

XMI, 410
extensiones, 410
pretty print, 410
XML Schema,
correspondencia entre el modelo y el esquema, 163
XMLSPY,
complemento, 638
registrar el complemento, 646

Z

Zoom,
ajustar el tamaño, 104

© 2014 Altova GmbH