

AltovaXML 2010

Benutzer- und Referenzhandbuch

AltovaXML 2010 Benutzer- und Referenzhandbuch

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

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

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

Published: 2010

© 2010 Altova GmbH

Inhaltsverzeichnis

1	Einführung	3
1.1	Produkt-Features	4
1.2	Funktionsumfang	5
1.3	Systemanforderungen und Installation	7
1.4	Zu dieser Dokumentation	8
2	Verwendung	10
2.1	Befehlszeile	11
2.1.1	XML-Validierung und Wohlgeformtheit	13
2.1.2	XSLT 1.0 Transformationen	16
2.1.3	XSLT 2.0 Transformationen	18
2.1.4	XQuery 1.0 Ausführungen	20
2.2	COM-Schnittstelle	22
2.2.1	Registrieren von AltovaXML als COM Serverobjekt	23
2.2.2	AltovaXML Objektmodell	24
2.2.3	Applikation	25
2.2.4	XMLValidator	26
2.2.5	XSLT1	28
2.2.6	XSLT2	30
2.2.7	XQuery	33
2.2.8	Beispiele	36
	<i>Visual Basic</i>	36
	<i>JScript</i>	37
	<i>C++</i>	38
2.3	Java-Schnittstelle	40
2.3.1	Schnittstellen	42
	<i>IAltovaXMLEngine</i>	42
	<i>IAltovaXMLFactory</i>	43
	<i>IExecutable</i>	44
	<i>IReleasable</i>	44
	<i>IXMLValidator</i>	45
	<i>IXQuery</i>	46
	<i>IXSLT</i>	48

2.3.2	Klassen	50
	<i>AltovaXMLFactory</i>	50
	<i>XMLValidator</i>	51
	<i>XQuery</i>	54
	<i>XSLT1</i>	58
	<i>XSLT2</i>	60
2.4	.NET-Schnittstelle	64
2.4.1	Allgemeine Verwendung und Beispiel	66
2.4.2	Altova.AltovaXML.XMLValidator	68
2.4.3	Altova.AltovaXML.XSLT1	70
2.4.4	Altova.AltovaXML.XSLT2	72
2.4.5	Altova.AltovaXML.XQuery	74
3	Explizite Freigabe von AltovaXML COM-Server von C# und VB.NETaus	78
4	OOXML- und ZIP-Dateien	80
5	Prozessorinformationen	82
5.1	Altova XML Validator	83
5.2	XSLT 1.0-Prozessor: Implementierungsinformationen	84
5.3	XSLT 2.0-Prozessor: Implementierungsinformationen	86
	5.3.1 Allgemeine Informationen	87
	5.3.2 XSLT 2.0-Elemente und -Funktionen	89
5.4	XQuery 1.0-Prozessor: Implementierungsinformationen	90
5.5	XPath 2.0- und XQuery 1.0-Funktionen	93
	5.5.1 Allgemeine Informationen	94
	5.5.2 Unterstützung von Funktionen	96
5.6	Erweiterungen	100
	5.6.1 Java-Erweiterungsfunktionen	101
	<i>Benutzerdefinierte Klassendateien</i>	102
	<i>Benutzerdefinierte Jar-Dateien</i>	104
	<i>Java: Konstruktoren</i>	105
	<i>Java: Statische Methoden und statische Felder</i>	106
	<i>Java: Instanzmethoden und Instanzfelder</i>	106
	<i>Datentypen: XPath/XQuery in Java</i>	107
	<i>Datentypen: Java in XPath/XQuery</i>	108
	5.6.2 .NET-Erweiterungsfunktionen	110

	<i>.NET: Konstruktoren</i>	112
	<i>.NET: Statische Methoden und statische Felder</i>	112
	<i>.NET: Instanzmethoden und Instanzfelder</i>	113
	<i>Datentypen: XPath/XQuery in .NET</i>	114
	<i>Datentypen: .NET in XPath/XQuery</i>	115
5.6.3	MSXSL Scripts für XSLT	116
5.6.4	Altova Erweiterungsfunktionen	119

6 Lizenzvereinbarung 124

Index

Kapitel 1

Einführung

1 Einführung

AltovaXML 2010 ist ein XML-Applikationspaket, das den Altova XML Validator, den Altova XSLT 1.0-Prozessor, den Altova XSLT 2.0-Prozessor und den Altova XQuery 1.0-Prozessor enthält. Das Paket ist kostenlos erhältlich und kann in Form einer einzigen Installationsdatei von der [Altova Website](#) heruntergeladen werden. AltovaXML dient zum Validieren von XML-Dokumenten, zum Transformieren von XML-Dokumenten mit Hilfe von XSLT Stylesheets und zum Ausführen von XQuery-Dokumenten.

AltovaXML kann über die Befehlszeile, über eine COM-Schnittstelle, in Java-Programmen und in .NET-Applikationen verwendet werden. In dieser Dokumentation wird beschrieben, wie AltovaXML in all diesen Umgebungen eingesetzt werden kann. Des Weiteren enthält diese Dokumentation Informationen über die implementierungsspezifischen Aspekte der Prozessoren in diesem Paket.

1.1 Produkt-Features

Die wichtigsten Features von AltovaXML sind:

Paket

- XML Validator, XSLT-Prozessoren und XQuery-Prozessor in einer einzigen Installationsdatei.
- Installationsdatei zum Gratis-Download von der [Altova Website](#).
- Einfache Installation ausführbarer Dateien auf Windows-Systemen.

Befehlszeile

- Verwendung der Befehlszeile für Validierung, XSLT-Transformation und XQuery-Ausführung.
- Validierung von XML-Dokumenten gemäß den DTD- und XML-Schema-Regeln des W3C.
- Transformation von XML-Dokumenten mit Hilfe von XSLT 1.0 und XSLT 2.0 Stylesheets entsprechend den jeweiligen W3C-Spezifikationen.
- Ausführung von XQuery 1.0-Dokumenten entsprechend der W3C-Spezifikation.

COM-Schnittstelle

- Kann über eine COM-Schnittstelle verwendet werden, somit auch mit Applikationen und Script-Sprachen, die COM unterstützen.
- Die COM-Schnittstelle ist für Raw- und Dispatch-Schnittstellen implementiert.
- Über die Schnittstellen-Eigenschaften steht eine breite Palette an Funktionen für XML-Validierung, XSLT-Transformation und XQuery-Ausführung zur Verfügung.
- XML, DTD, XML Schema, XSLT und XQuery Input kann in Form von Dateien oder als Textstrings in Scripts und Applikationsdaten erfolgen.

Java-Schnittstelle

- Die AltovaXML-Funktionalität steht in Java-Klassen zur Verfügung, die in Java-Programmen verwendet werden können.
- Java-Klassen bieten Funktionen zur XML-Validierung, XSLT-Transformation und XQuery-Ausführung.

.NET-Schnittstelle

- AltovaXML ist in eine DLL eingebettet, sodass .NET-User die Funktionen von AltovaXML aufrufen können.
- Bietet von Altova signierte Primary Interop Assembly.
- Bietet eine breite Palette von Funktionen zur XML-Validierung, XSLT-Transformation und XQuery-Ausführung.
- XML, DTD, XML Schema, XSLT und XQuery Input kann in Form von Dateien oder als Textstrings in Scripts und Applikationsdaten erfolgen.

1.2 Funktionsumfang

AltovaXML bietet die unten aufgelisteten Funktionen. Die meisten dieser Funktionen sind gebräuchlich bei der Verwendung der Befehlszeile und der COM-Schnittstelle. Ein wichtiger Unterschied ist, dass Dokumente bei Verwendung der COM-Schnittstelle anhand von Textstrings über die Applikation oder den Skripting-Code konstruiert werden können (anstatt XML-, DTD-, XML Schema, XSLT oder XQuery-Dateien zu referenzieren).

XML- und XBRL-Validierung

- Validiert das vorliegende XML-Dokument und gibt den Wert "gültig" oder "ungültig" zurück.
- Die Validierung kann gegen die DTD oder das in der XML-Datei referenzierte XML-Schema oder gegen eine externe DTD oder ein XML-Schema durchgeführt werden, das über einen Befehlszeilenparameter oder die Eigenschaft einer COM-Schnittstelle bereitgestellt wird.
- Überprüft die Wohlgeformtheit des vorliegenden XML-Dokuments. Dies erfolgt separat zur Validierung.
- Validiert XBRL-Dokumente. Das XBRL-Dokument wird anhand einer XBRL-Taxonomie (die eine `.xsd`-Datei ist) entsprechend den XBRL-Regeln validiert.

XSLT-Transformationen

- Transformierung des vorliegenden XML-Dokuments anhand des mitgelieferten XSLT 1.0 - oder XSLT 2.0-Dokuments.
- Das XML-Dokument kann über eine URL als Input bereitgestellt werden. Bei Verwendung über die COM-Schnittstelle kann das XML-Dokument auch als Textstring geliefert werden.
- Das XSLT-Dokument kann über eine URL als Input bereitgestellt werden. Bei Verwendung über die COM-Schnittstelle kann das XSLT-Dokument auch als Textstring geliefert werden.
- Gibt Ausgabedokumente unter dem definierten Pfad zurück. Bei Aufruf über die COM-Schnittstelle können die Ausgabedokumente auch als String zurückgegeben werden.
- Die XSLT-Parameter können über die Befehlszeile und über die COM-Schnittstelle bereitgestellt werden.

XQuery-Ausführung

- Führt das bereitgestellte XQuery 1.0-Dokument aus, optional gegen ein in einem Befehlszeilenparameter definiertes XML-Dokument oder eine COM-Schnittstelleneigenschaft.
- Ein XQuery-Dokument kann über eine URL als Input bereitgestellt werden. Bei Verwendung über die COM-Schnittstelle kann das XQuery-Dokument als Alternative auch als Textstring bereitgestellt werden.
- Ein XML-Dokument kann über eine URL als Input bereitgestellt werden. Bei Verwendung über die COM-Schnittstelle kann das XML-Dokument als Alternative auch als Textstring bereitgestellt werden.
- Gibt Ausgabedokumente unter dem definierten Pfad zurück. Bei Aufruf über die COM-Schnittstelle können die Ausgabedokumente auch als String zurückgegeben werden.
- Externe XQuery-Variablen können über die Befehlszeile und die COM-Schnittstelle bereitgestellt werden.
- Zu den Serialisierungsoptionen gehören: Ausgabecodierung, Ausgabemethode (d.h. ob

die Ausgabe als XML, XHTML, HTML oder Text erfolgen soll), Auslassen der XML-Deklaration und Einrückung.

1.3 Systemanforderungen und Installation

Systemanforderungen

AltovaXML wird auf Windows NT, Windows XP, Windows Server 2003, Windows Server 2008, Windows Vista und Windows 7 unterstützt. Um AltovaXML über eine COM-Schnittstelle verwenden zu können, benötigt der Benutzer Benutzerrechte für die COM-Schnittstelle, d.h. die Berechtigung, die Applikation zu registrieren und die jeweiligen Applikationen und/oder Scripts ausführen zu können.

Installation

AltovaXML ist eine selbstextrahierende Datei, die auf der [Altova Website](#) zum Download zur Verfügung steht und die AltovaXML mit den nötigen Registrierungen installiert. Nachdem Sie die Installationsdatei (`AltovaXML2010.exe`) auf Ihren Rechner heruntergeladen haben, doppelklicken Sie darauf, um die Installation zu starten. AltovaXML wird unter `Programme` im Ordner `Altova/AltovaXML2010` installiert. Alle erforderlichen Registrierungen, damit AltovaXML über eine COM-Schnittstelle, als Java-Schnittstelle und in der .NET-Umgebung verwendet werden kann, werden vom Installationsprogramm vorgenommen. Dazu gehört auch die Registrierung der AltovaXML exe-Datei als COM-Serverobjekt, die Installation der Datei `AltovaXMLLib.dll` (für die Verwendung der Java-Schnittstelle) im Verzeichnis `WINDIR\system32\` das Hinzufügen der Datei `Altova.AltovaXML.dll` zur .NET Referenzbibliothek.

Beachten Sie bitte Folgendes:

- Zur Verwendung der Befehlszeile, rufen Sie die installierte exe-Datei auf (`AltovaXML.exe`). Diese Datei kann in einen anderen Ordner auf Ihrem Rechner oder im Netzwerk kopiert werden, auf den Sie Zugriff haben und über den Sie diese Datei aufrufen können.
- Sie können AltovaXML sofort über die COM-Schnittstelle verwenden, da die installierte exe-Datei `AltovaXML_COM.exe` bereits als COM-Serverobjekt registriert wurde. Wenn Sie die exe-Datei `AltovaXML_COM.exe` in einem anderen Ordner auf Ihrem Rechner oder im Netzwerk speichern, müssen Sie sie dort manuell als COM-Serverobjekt registrieren. Eine Anleitung dazu finden Sie unter, [Registrieren von AltovaXML als COM-Serverobjekt](#).
- Um AltovaXML über eine Java-Schnittstelle verwenden zu können, muss `AltovaXML_COM.exe` als COM-Serverobjekt registriert sein und der Pfad zur Datei `AltovaXML.jar` (installiert im Ordner `Altova/AltovaXML2010`) muss als COM-Serverobjekt zur `CLASSPATH`-Registrierung hinzugefügt werden (dies erfolgt automatisch während der Installation). Bei der Installation wird auch die Datei `AltovaXMLLib.dll` im Verzeichnis `WINDIR\system32\` gespeichert. Beachten Sie jedoch: Wenn Sie nach der Installation den Pfad zur Datei `AltovaXML_COM.exe` ändern, müssen Sie sie anschließend manuell als COM-Serverobjekt registrieren. Nähere Informationen dazu siehe [Registrieren von AltovaXML als COM Serverobjekt](#) und [Java-Schnittstelle](#).

1.4 Zu dieser Dokumentation

Diese Dokumentation ist die offizielle Produktdokumentation zu AltovaXML und enthält ausführliche Informationen darüber. Sie ist in die folgenden Abschnitte gegliedert:

- In der [Einführung](#) werden die Features des Produkts AltovaXML, seine Funktionen und die wichtigsten Systemanforderungen für die Verwendung von AltovaXML sowie dessen Installation beschrieben.
- Im Abschnitt [Verwendung](#) wird beschrieben, wie man AltovaXML über die Befehlszeile und eine COM-Schnittstelle verwendet. Im Abschnitt [Befehlszeile](#) finden Sie nähere Informationen über die Syntax, mit der man die verschiedenen Funktionen von AltovaXML aufrufen kann. Im Abschnitt [COM-Schnittstelle](#) finden Sie Informationen darüber, wie AltovaXML als COM-Schnittstelle verwendet werden kann; Sie finden darin eine ausführliche Beschreibung des Objektmodells, seiner Schnittstellen und der Schnittstelleneigenschaften. Der Abschnitt [Java-Schnittstelle](#) enthält eine Beschreibung zur Verwendung von AltovaXML mit Java sowie eine Auflistung der definierten Java-Schnittstellen und -Klassen. Der Abschnitt [.NET-Schnittstelle](#) enthält eine Beschreibung der Verwendung und eine Auflistung der verschiedenen Methoden und Eigenschaften, die verwendet werden können.
- Der Abschnitt [Prozessor-Informationen](#) enthält eine Beschreibung der implementierungsspezifischen Aspekte der verschiedenen Prozessoren, die die Komponenten von AltovaXML bilden. Jeder Prozessor wird separat beschrieben.

Kapitel 2

Verwendung

2 Verwendung

Nachdem Sie AltovaXML heruntergeladen und im gewünschten Ordner installiert haben, können Sie es auf folgende Arten verwenden:

- Durch Aufruf der Anwendung über die [Befehlszeile](#),
- Durch Verwendung der Anwendung über eine [COM-Schnittstelle](#),
- Durch Verwendung der Anwendung über eine [Java-Schnittstelle](#) und
- Durch Verwendung der Anwendung in der [.NET-Umgebung](#).

2.1 Befehlszeile

Um AltovaXML über die Befehlszeile verwenden zu können, muss die exe-Datei (`AltovaXML.exe`) in einem Ordner auf Ihrem Rechner oder dem Netzwerk installiert sein, bzw. dorthin kopiert worden sein und Sie müssen Zugriff auf diesen Ordner haben. Die allgemeine Syntax zum Aufrufen der Applikation ist:

```
AltovaXML functionality arg1 ... argN [options]
```

wobei

<code>AltovaXML</code>	Ruft die Applikation auf.
<code>functionality</code>	Gibt an, ob die Funktion XML-Validierung, Wohlgeformtheitsprüfung, XSLT 1.0-Transformation, XSLT 2.0-Transformation oder XQuery 1.0-Ausführung aufgerufen wird. Die jeweiligen Werte sind <code>-validate</code> (oder <code>-v</code>), <code>-wellformed</code> (oder <code>-w</code>), <code>-xslt1</code> , <code>-xslt2</code> , <code>-xquery</code> (oder <code>-xq</code>).
<code>arg1 ... argN</code>	Die Argumente der aufgerufenen Funktionalität.
<code>options</code>	Jede Funktionalität hat ihre eigenen Optionen. Diese Optionen sind in den entsprechenden Unterabschnitten dieses Abschnitts beschrieben.

Allgemeine Optionen

<code>-help, -h, oder -?</code>	Zeigt Informationen zur Verwendung an, z.B. eine Liste aller Argumente und Optionen.
<code>-version, -ver</code>	Zeigt die Programmversion an.

Es stehen die folgenden Funktionalitäten zur Verfügung. Die zulässigen Argumente und Optionen für die einzelnen Funktionalitäten sind in den folgenden Abschnitten näher beschrieben:

- [XML-Validierung und Wohlgeformtheit](#)
- [XSLT 1.0-Transformationen](#)
- [XSLT 2.0-Transformationen](#)
- [XQuery 1.0-Ausführung](#)

Zusammenfassung zur Verwendung

Im Folgenden finden Sie einen Überblick über die Verwendung der Befehlszeile. Nähere Informationen finden Sie im jeweiligen Abschnitt.

[Verwendung des Altova XML Validators](#)

- `-validate <Dateiname> [-schema <Dateiname> | -dtd <Dateiname>]`
- `-wellformed <Dateiname>`

[Verwendung des Altova XSLT 1.0-Prozessors](#)

- `-xslt1 <Dateiname> -in <Dateiname> [-param name=value] [-out <Dateiname>]`

Verwendung des Altova XSLT 2.0-Prozessors

- `-xslt2 <Dateiname> -in <Dateiname> [-param name=value] [-out <Dateiname>]`

Verwendung des Altova XQuery 1.0-Prozessors

- `-xquery <Dateiname> [-in <Dateiname>] [-param name=value] [-out <Dateiname>] [serialization options]`

Hinweis: Wenn der Dateiname oder der Pfad ein Leerzeichen enthält, sollte der gesamte Pfadname in Anführungszeichen gesetzt werden, z.B.: "c:\My Files\MyXML.xml" oder "c:\MyFiles\My XML.xml".

2.1.1 XML-Validierung und Wohlgeformtheit

XML-Validierungssyntax

Die Syntax zum Aufrufen der **XML-Validierung** lautet:

```
AltovaXML -validate xmlfile [-schema schemafilename | -dtd dtddfilename]
          [options]
```

wobei

AltovaXML die Applikation aufruft
 -validate (or -v) angibt, dass der Altova XML Validator zum Validieren der Datei "XML-Datei" zu verwenden ist.

Es stehen die folgenden Optionen zur Verfügung:

-schema (oder -s) gibt an, dass die XML-Schema-Datei Schemadatei für die Validierung zu verwenden ist.
 -dtd (or -d) definiert, dass die DTD-Datei DTD-Datei für die Validierung zu verwenden ist.
 -xbrlConsistency (oder -xc) die Semantik von XBRL-Dokumenten überprüft.

Mapping-Optionen

Über den XML-Katalog und die globalen Altova-Ressourcen stehen die folgenden Mapping-Optionen zur Verfügung. (Globale Altova-Ressourcen können nur verwendet werden, wenn ein Produkt, das globale Altova Ressourcen unterstützt, wie z.B. Altova XMLSpy, installiert ist).

-catalog (oder -c) [<filename> Aktiviert die Katalogzuordnung unter Verwendung des angegebenen Katalogs. Wenn keine Datei definiert ist, wird ein Katalog namens RootCatalog.xml aus dem AltovaXML-Applikationsordner als Standardordner verwendet.
 -globalresources (oder -gr) [<filename> Aktiviert die Zuordnung von globalen Altova-Ressourcen unter Verwendung der angegebenen globalen Ressourcen-XML-Datei oder - falls keine Datei definiert wird - unter Verwendung von GlobalResources.xml in My Documents/Altova.
 -globalresourceconfig (oder -gc) [<name> Definiert die aktive Konfigurationsdatei für globale Ressourcen.

Anmerkung zu globalen Ressourcen

Zum Auswählen einer Ressource unter Verwendung der Methode "globale Altova-Ressourcen" sind zwei Einstellungen erforderlich:

- Die XML-Datei für globale Ressourcen enthält Definitionen der globalen Ressourcen. Diese Datei kann mit der Option -globalresources (oder -gr) definiert werden. Falls keine Datei angegeben wird, so wird die Datei GlobalResources.xml im Ordner My Documents/Altova verwendet.
- Jede globale Ressource in der XML-Datei für globale Ressourcen kann mehrere Konfigurationen haben, wobei jede Konfiguration einer Ressource zugeordnet ist. Mit Hilfe der Option -globalresourceconfig (oder -gc) können Sie angeben, welche

Konfiguration verwendet werden soll. Mit der Erweiterung wird definiert, welche Ressource verwendet werden soll.

Hinweis:

- Wenn kein XML-Schema bzw. keine DTD-Datei als Befehlszeilenoption definiert ist, muss im XML-Dokument selbst ein XML-Schema oder eine DTD-Datei definiert sein.
- Wenn in der Befehlszeile ein XML-Schema oder eine DTD-Datei angegeben sind **und** in der XML-Datei ein XML-Schema oder eine DTD-Datei referenziert ist, dann wird die in der Befehlszeilenoption angegebene Datei für die Validierung verwendet.
- Wenn ein XBRL-Instanzdokument validiert wird, wird dies anhand der XBRL-Taxonomie, bei der es sich um eine .xsd-Datei handelt, durchgeführt. Wenn zusätzlich zur Syntaxvalidierung eine semantische Validierung erforderlich ist, verwenden Sie die Option `-xbrlConsistency`.

Syntax für die Wohlgeformtheitsprüfung

Die Syntax zum Aufrufen der **Wohlgeformtheitsprüfung** lautet:

```
AltovaXML -wellformed XML-Datei
```

wobei

```
AltovaXML           die Applikation aufruft
-wellformed (oder   angibt, dass zur Überprüfung der Wohlgeformtheit der Datei
-w)                XML-Datei der Altova XML Validator zu verwenden ist.
```

Mapping-Optionen

Über den XML-Katalog und die globalen Altova-Ressourcen stehen die folgenden Mapping-Optionen zur Verfügung. (Globale Altova-Ressourcen können nur verwendet werden, wenn ein Produkt, das globale Altova Ressourcen unterstützt, wie z.B. Altova XMLSpy, installiert ist).

```
-catalog (oder -c)   Aktiviert die Katalogzuordnung unter Verwendung des angegebenen
[ <filename>]       Katalogs. Wenn keine Datei definiert ist, wird ein Katalog namens
                    RootCatalog.xml aus dem AltovaXML-Applikationsordner als
                    Standardordner verwendet.

-globalresources     Aktiviert die Zuordnung von globalen Altova-Ressourcen unter
(oder -gr)          Verwendung der angegebenen globalen Ressourcen-XML-Datei
[ <filename>]       oder - falls keine Datei definiert wird - unter Verwendung von
                    GlobalResources.xml in My Documents/Altova.

-globalresourcecon   Definiert die aktive Konfigurationsdatei für globale Ressourcen.
fig (oder -gc)      [ <name>]
```

Anmerkung zu globalen Ressourcen

Zum Auswählen einer Ressource unter Verwendung der Methode "globale Altova-Ressourcen" sind zwei Einstellungen erforderlich:

- Die XML-Datei für globale Ressourcen enthält Definitionen der globalen Ressourcen. Diese Datei kann mit der Option `-globalresources` (oder `-gr`) definiert werden. Falls keine Datei angegeben wird, so wird die Datei `GlobalResources.xml` im Ordner `My Documents/Altova` verwendet.

- Jede globale Ressource in der XML-Datei für globale Ressourcen kann mehrere Konfigurationen haben, wobei jede Konfiguration einer Ressource zugeordnet ist. Mit Hilfe der Option `-globalresourceconfig` (oder `-gc`) können Sie angeben, welche Konfiguration verwendet werden soll. Mit der Erweiterung wird definiert, welche Ressource verwendet werden soll.

Beispiele

- `AltovaXML -validate test.xml -schema testschema.xsd`
- `AltovaXML -v test.xml -dtd testdtd.dtd`
- `AltovaXML -wellformed test.xml`
- `AltovaXML -w test.xml`
- `AltovaXML -v test.xml -dtd testdtd.dtd -c MyCatalog.xml`
- `AltovaXML -validate test.xml -schema testschema.xsd -xc`

Hinweis: Bei Verwendung von AltovaXML in Batch-Befehlen, beachten Sie bitte Folgendes:

- Der Rückgabecode des letzten ausgeführten Befehls wird in der Variable `errorlevel` gespeichert, deren Wert mit einem Batch-Befehl wie z.B. `ECHO %errorlevel%` abgerufen werden kann.
- Die Rückgabecodes sind 0 = wohlgeformt/gültig; 1 = nicht wohlgeformt/ungültig.

2.1.2 XSLT 1.0 Transformationen

Syntax

Die Syntax zum Aufrufen der **XSLT 1.0 Transformation** lautet:

```
AltovaXML -xslt1 xslt-Datei -in xml-Datei [-out Ausgabedatei]
      [options]
```

wobei

AltovaXML	die Applikation aufruft.
-xslt1	angibt, dass für eine XSLT-Transformation der Altova XSLT 1.0 Prozessor zu verwenden ist; der Prozessor verwendet die XSLT 1.0-Datei <i>xslt-Datei</i> für die Transformation.
-in	angibt, dass die XML-Datei <i>xml-Datei</i> transformiert werden soll und deren Pfad angibt.
-out	die <i>Ausgabedatei</i> und deren Pfad angibt. Wenn diese Option weggelassen wird, wird die Ausgabe in die Standardausgabe geschrieben.

Es stehen die folgenden Optionen zur Verfügung:

-param	Ruft die Anweisung <code>paramname=XPath expression</code> auf. Der <code>-param</code> Switch wird vor den einzelnen globalen Parametern verwendet. Wenn in einem XPath-Ausdruck - egal ob in einem XPath-Ausdruck selbst oder in einem String Literal im Ausdruck - Leerzeichen enthalten sind, müssen doppelte Anführungszeichen verwendet werden. Siehe Beispiele.
-xslstack	Die Stack-Größe ist die maximale Tiefe, bis zu der Anweisungen ausgeführt werden. Sie kann über den Wert <code>-xslstack</code> geändert werden. Der zulässige Mindestwert beträgt 100. Die Standard-Stack-Größe ist 1000. Bei Überschreitung der Stack-Größe während einer Transformation wird ein Fehler ausgegeben.
-namedTemplate (oder -n)	Definiert die named Template am Anfang. Das Argument wird durch ein Leerzeichen von seinem Wert getrennt. Beispiel: <code>-namedTemplate MyTemplate</code>
-mode (oder -m)	Definiert den Vorlagenmodus am Anfang. Sets the initial template mode. Das Argument wird durch ein Leerzeichen von seinem Wert getrennt. Beispiel: <code>-mode MyMode</code>

Mapping-Optionen

Über den XML-Katalog und die globalen Altova-Ressourcen stehen die folgenden Mapping-Optionen zur Verfügung. (Globale Altova-Ressourcen können nur verwendet werden, wenn ein Produkt, das globale Altova Ressourcen unterstützt, wie z.B. Altova XMLSpy, installiert ist).

-catalog (oder -c) [<filename>]	Aktiviert die Katalogzuordnung unter Verwendung des angegebenen Katalogs. Wenn keine Datei definiert ist, wird ein Katalog namens <code>RootCatalog.xml</code> aus dem AltovaXML-Applikationsordner als Standardordner verwendet.
-------------------------------------	---

`-globalresources` (oder `-gr`)
[<filename>] Aktiviert die Zuordnung von globalen Altova-Ressourcen unter Verwendung der angegebenen globalen Ressourcen-XML-Datei oder - falls keine Datei definiert wird - unter Verwendung von `GlobalResources.xml` in `My Documents/Altova`.

`-globalresourceconfig` (oder `-gc`)
[<name>] Definiert die aktive Konfigurationsdatei für globale Ressourcen.

Anmerkung zu globalen Ressourcen

Zum Auswählen einer Ressource unter Verwendung der Methode "globale Altova-Ressourcen" sind zwei Einstellungen erforderlich:

- Die XML-Datei für globale Ressourcen enthält Definitionen der globalen Ressourcen. Diese Datei kann mit der Option `-globalresources` (oder `-gr`) definiert werden. Falls keine Datei angegeben wird, so wird die Datei `GlobalResources.xml` im Ordner `My Documents/Altova` verwendet.
- Jede globale Ressource in der XML-Datei für globale Ressourcen kann mehrere Konfigurationen haben, wobei jede Konfiguration einer Ressource zugeordnet ist. Mit Hilfe der Option `-globalresourceconfig` (oder `-gc`) können Sie angeben, welche Konfiguration verwendet werden soll. Mit der Erweiterung wird definiert, welche Ressource verwendet werden soll.

Hinweis:

- Die XSLT-Datei muss in der Befehlszeilenanweisung definiert werden; eine XSLT-Datei, die in einer `<?xml-stylesheet?>` Verarbeitungsanweisung im XML-Dokument referenziert wird, wird nicht automatisch verwendet.
- Wenn der `-out` Parameter nicht angegeben wird, wird die Ausgabe in die Standardausgabe geschrieben.

Beispiele

- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -c MyCatalog.xml`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=//node/@att1`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date="//node/@att1 | //node/@att2"`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title='stringwithoutspace'`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title="'string with spaces' "`

2.1.3 XSLT 2.0 Transformationen

Syntax

Die Syntax zum Aufrufen der **XSLT 2.0 Transformation** lautet:

```
AltovaXML -xslt2 xslt-Datei -in xml-Datei [-out Ausgabedatei]
      [options]
```

wobei

AltovaXML	die Applikation aufruft.
-xslt2	angibt, dass für eine XSLT-Transformation der Altova XSLT 2.0 Prozessor zu verwenden ist; der Prozessor verwendet die XSLT 2.0-Datei <i>xslt-Datei</i> für die Transformation.
-in	angibt, dass die XML-Datei <i>xml-Datei</i> transformiert werden soll und deren Pfad angibt.
-out	die <i>Ausgabedatei</i> und deren Pfad angibt. Wenn diese Option weggelassen wird, wird die Ausgabe in die Standardausgabe geschrieben.

Es stehen die folgenden Optionen zur Verfügung:

-param	Ruft die Anweisung <code>paramname=XPath expression</code> auf. Der <code>-param</code> Switch wird vor den einzelnen globalen Parametern verwendet. Wenn in einem XPath-Ausdruck - egal ob in einem XPath-Ausdruck selbst oder in einem String Literal im Ausdruck - Leerzeichen enthalten sind, müssen doppelte Anführungszeichen verwendet werden. Siehe Beispiele
-xslstack	Die Stack-Größe ist die maximale Tiefe, bis zu der Anweisungen ausgeführt werden. Sie kann über den Wert <code>-xslstack</code> geändert werden. Der zulässige Mindestwert beträgt 100. Die Standard-Stack-Größe ist 1000. Bei Überschreitung der Stack-Größe während einer Transformation wird ein Fehler ausgegeben.
-namedTemplate (oder -n)	Definiert die named Template am Anfang. Das Argument wird durch ein Leerzeichen von seinem Wert getrennt. Beispiel: <code>-namedTemplate MyTemplate</code>
-mode (oder -m)	Definiert den Vorlagenmodus am Anfang. Sets the initial template mode. Das Argument wird durch ein Leerzeichen von seinem Wert getrennt. Beispiel: <code>-mode MyMode</code>

Mapping-Optionen

Über den XML-Katalog und die globalen Altova-Ressourcen stehen die folgenden Mapping-Optionen zur Verfügung. (Globale Altova-Ressourcen können nur verwendet werden, wenn ein Produkt, das globale Altova Ressourcen unterstützt, wie z.B. Altova XMLSpy, installiert ist).

-catalog (oder -c) [<filename>]	Aktiviert die Katalogzuordnung unter Verwendung des angegebenen Katalogs. Wenn keine Datei definiert ist, wird ein Katalog namens <code>RootCatalog.xml</code> aus dem AltovaXML-Applikationsordner als Standardordner verwendet.
-------------------------------------	---

`-globalresources` Aktiviert die Zuordnung von globalen Altova-Ressourcen unter
(oder `-gr`)
[<filename>] Verwendung der angegebenen globalen Ressourcen-XML-Datei
oder - falls keine Datei definiert wird - unter Verwendung von
GlobalResources.xml in My Documents/Altova.

`-globalresourcecon` Definiert die aktive Konfigurationsdatei für globale Ressourcen.
fig (oder `-gc`)
[<name>]

Anmerkung zu globalen Ressourcen

Zum Auswählen einer Ressource unter Verwendung der Methode "globale Altova-Ressourcen" sind zwei Einstellungen erforderlich:

- Die XML-Datei für globale Ressourcen enthält Definitionen der globalen Ressourcen. Diese Datei kann mit der Option `-globalresources` (oder `-gr`) definiert werden. Falls keine Datei angegeben wird, so wird die Datei `GlobalResources.xml` im Ordner `My Documents/Altova` verwendet.
- Jede globale Ressource in der XML-Datei für globale Ressourcen kann mehrere Konfigurationen haben, wobei jede Konfiguration einer Ressource zugeordnet ist. Mit Hilfe der Option `-globalresourceconfig` (oder `-gc`) können Sie angeben, welche Konfiguration verwendet werden soll. Mit der Erweiterung wird definiert, welche Ressource verwendet werden soll.

Hinweis:

- Die XSLT-Datei muss in der Befehlszeilenanweisung definiert werden; eine XSLT-Datei, die in einer `<?xml-stylesheet?>` Verarbeitungsanweisung im XML-Dokument referenziert wird, wird nicht automatisch verwendet.
- Wenn der `-out` Parameter nicht angegeben wird, wird die Ausgabe in die Standardausgabe geschrieben.
- Der XSLT 2.0-Prozessor kann zur Verarbeitung eines XSLT 1.0 Stylesheet im Rückwärtskompatibilitätsmodus verwendet werden. Die Ausgabe kann sich allerdings von der eines XSLT 1.0-Prozessors, der dasselbe XSLT 1.0 Stylesheet verarbeitet, unterscheiden.

Beispiele

- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -c MyCatalog.xml`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date="//node/@att1`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date="//node/@att1 | //node/@att2"`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title='stringwithoutspace'`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title="'string with spaces' "`

2.1.4 XQuery 1.0 Ausführungen

Syntax

Die Syntax zum Aufrufen einer XQuery 1.0 Ausführung lautet:

```
AltovaXML -xquery xquery-Datei [-in XML-Input-Datei -out
  Ausgabedatei] [options]
```

wobei

AltovaXML	die Applikation aufruft.
-xquery (oder -xq)	angibt, dass für eine XQuery-Ausführung der Datei <i>xquery-Datei</i> der Altova XQuery 1.0-Prozessor zu verwenden ist.
-in	die XML-Input-Datei definiert.
-out	die Ausgabedatei und deren Pfad definiert. Wenn diese Option weggelassen wird, wird die Ausgabe in die Standardausgabe geschrieben.

Es stehen die folgenden Optionen zur Verfügung:

-var	Gibt eine externe Variable und deren Wert an. Hat die Form <i>name=value</i> . Es können beliebig viele externe Variablen verwendet werden, jeder muss jedoch das Schlüsselwort <i>-var</i> vorangestellt werden. Variablenwerte müssen Strings sein, die der lexikalischen Form des Datentyps entsprechen, als der die Variable deklariert wurde.
-xparam	Gibt einen XQuery-Parameternamen und den Wert des Parameters an. Hat die Form <i>name=XPathExpression</i> . Setzen Sie den XPath-Ausdruck in doppelte Anführungszeichen, wenn der Ausdruck Leerzeichen enthält. Setzen Sie String-Literalzeichen im XPath-Ausdruck in einfache Anführungszeichen. Es können beliebig viele Parameter verwendet werden, jedoch muss jedem das Schlüsselwort <i>-xparam</i> vorangestellt werden.
-outputMethod (oder -om)	Serialisierungsoption zum Definieren des Ausgabeart. Gültige Werte sind <i>xml</i> , <i>html</i> , <i>xhtml</i> und <i>text</i> . Standardwert ist <i>xml</i> .
-omitXMLDeclaration (oder -od)	Serialisierungsoption, um festzulegen, ob die XML-Deklaration in der Ausgabe enthalten sein soll oder nicht. Gültige Werte sind <i>yes</i> und <i>no</i> . Standardwert ist <i>yes</i> .
-outputIndent (oder -oi)	Serialisierungsoption, um festzulegen, ob die Ausgabe Einrückungen enthalten soll oder nicht. Gültige Werte sind <i>yes</i> und <i>no</i> . Standardwert ist <i>no</i> .
-outputEncoding (oder -oe)	Serialisierungsoption zur Definition des Zeichensatzes der Ausgabe. Gültige Werte sind Namen in der IANA Zeichensatz-Registrierdatei. Standardwert ist <i>UTF-8</i> .

Mapping-Optionen

Über den XML-Katalog und die globalen Altova-Ressourcen stehen die folgenden Mapping-Optionen zur Verfügung. (Globale Altova-Ressourcen können nur verwendet werden, wenn ein Produkt, das globale Altova Ressourcen unterstützt, wie z.B. Altova XMLSpy, installiert ist).

<code>-catalog (oder -c)</code> [<filename>]	Aktiviert die Katalogzuordnung unter Verwendung des angegebenen Katalogs. Wenn keine Datei definiert ist, wird ein Katalog namens <code>RootCatalog.xml</code> aus dem AltovaXML-Applikationsordner als Standardordner verwendet.
<code>-globalresources (oder -gr)</code> [<filename>]	Aktiviert die Zuordnung von globalen Altova-Ressourcen unter Verwendung der angegebenen globalen Ressourcen-XML-Datei oder - falls keine Datei definiert wird - unter Verwendung von <code>GlobalResources.xml</code> in <code>My Documents/Altova</code> .
<code>-globalresourceconfig (oder -gc)</code> [<name>]	Definiert die aktive Konfigurationsdatei für globale Ressourcen.

Anmerkung zu globalen Ressourcen

Zum Auswählen einer Ressource unter Verwendung der Methode "globale Altova-Ressourcen" sind zwei Einstellungen erforderlich:

- Die XML-Datei für globale Ressourcen enthält Definitionen der globalen Ressourcen. Diese Datei kann mit der Option `-globalresources (oder -gr)` definiert werden. Falls keine Datei angegeben wird, so wird die Datei `GlobalResources.xml` im Ordner `My Documents/Altova` verwendet.
- Jede globale Ressource in der XML-Datei für globale Ressourcen kann mehrere Konfigurationen haben, wobei jede Konfiguration einer Ressource zugeordnet ist. Mit Hilfe der Option `-globalresourceconfig (oder -gc)` können Sie angeben, welche Konfiguration verwendet werden soll. Mit der Erweiterung wird definiert, welche Ressource verwendet werden soll.

Hinweis: Wenn der `-out` Parameter nicht angegeben wird, wird die Ausgabe in die Standardausgabe geschrieben.

Beispiele

- `AltovaXML -xquery testquery.xq -out testout.xml`
- `AltovaXML -xquery testquery.xq -in products.xml -out testout.xml -var company=Altova -var date=2006-01-01`
 - `AltovaXML -xquery testquery.xq -out testout.xml -xparam source=" doc('c:\test\books.xml')//book "`
- `AltovaXML -xquery testquery.xq -in products.xml -out testout.xml -var company=Altova -omitXMLDeclaration no -oe ASCII`

2.2 COM-Schnittstelle

Wenn AltovaXML als COM Serverobjekt registriert ist, kann das Programm von Applikationen und Script-Sprachen aus aufgerufen werden, die Programmierunterstützung für COM-Aufrufe bieten. Dies ist nützlich, da auf diese Art die XML-Dokumentvalidierung, XSLT-Transformationen (XSLT 1.0 und XSLT 2.0) und XQuery 1.0 Dokumentausführungen von AltovaXML über eine breite Palette von Benutzerapplikationen ausgeführt werden können.

Um AltovaXML mit Applikationen und Script-Sprachen verwenden zu können, die eine COM-Schnittstelle haben, müssen Sie AltovaXML zuerst als COM Serverobjekt registrieren. Nähere Informationen dazu finden Sie unter [Registrieren von AltovaXML als COM Serverobjekt](#).

Das AltovaXML Objektmodell und seine Eigenschaften werden in den folgenden Unterabschnitten dieses Abschnitts beschrieben. (Beachten Sie: Sie können sowohl die Raw-Schnittstelle als auch die Dispatch-Schnittstelle von COM verwenden. Die Raw-Schnittstelle wird für Programmiersprachen wie z.B. C++ verwendet. Die Dispatch-Schnittstelle wird für Script-Sprachen (wie z.B. JavaScript) verwendet, die das Übergeben von Parametern über eine Referenz nicht gestatten.) Sie können AltovaXML daher mit folgenden Sprachen verwenden:

- Skriptsprachen wie z.B. JavaScript oder anderen Skriptsprachen, die die COM-Schnittstelle unterstützen.
- Programmiersprachen wie C++ oder andere, die die COM-Schnittstelle unterstützen.
- Java und .NET, für die Schnittstellen als Wrapper erstellt werden, wobei Klassen rund um die COM-Schnittstelle erstellt werden.

Dieser Abschnitt über die Verwendung der COM-Schnittstelle endet mit einer Gruppe von Beispielen, wie verschiedene Funktionen von AltovaXML über verschiedene Benutzerapplikationen aufgerufen werden können.

Beispiele

Zusätzliche Beispieldateien finden Sie im Ordner `Examples` des Applikationsordners.

2.2.1 Registrieren von AltovaXML als COM Serverobjekt

Wenn Sie AltovaXML 2010 registrieren, wird `AltovaXML_COM.exe` automatisch als COM-Serverobjekt registriert. Wenn Sie den Pfad von `AltovaXML_COM.exe` ändern müssen, sollten Sie AltovaXML am besten deinstallieren und dann im gewünschten Ordner neu installieren. Auf diese Art wird die notwendige Deregistrierung und Registrierung vom Installer durchgeführt. Wenn Sie `AltovaXML_COM.exe` auf einen anderen Rechner kopieren, müssen Sie AltovaXML unter dem neuen Pfad manuell als COM-Serverobjekt registrieren, wie im Folgenden erklärt. Es wird in dieser Beschreibung davon ausgegangen, dass AltovaXML erfolgreich installiert wurde.

Manuelle Registrierung

So registrieren Sie AltovaXML als COM-Serverobjekt:

1. Kopieren Sie `AltovaXML_COM.exe` in den gewünschten Ordner. Wenn sich der Ordner nicht auf Ihrem lokalen Rechner befindet, mappen Sie ihn auf einen Netzwerkordner.
2. Öffnen Sie die Windows Eingabeaufforderung oder klicken Sie im Menü "Start" auf **Ausführen...**
3. Registrieren Sie die Applikation mit Hilfe des Parameters `/regserver` als COM-Serverobjekt. Wenn `AltovaXML_COM.exe` sich z.B. im Ordner `c:\AltovaXML` befindet, geben Sie ein:

```
c:\AltovaXML\AltovaXML_COM.exe /regserver
```

und drücken Sie die Eingabetaste.

Überprüfen, ob die Registrierung erfolgreich war

Wenn die Registrierung erfolgreich war, sollte die Registrierdatei die Klassen `AltovaXML.Application` und `AltovaXML.Application.1` enthalten. Diese beiden Klassen finden Sie normalerweise unter `HKEY_LOCAL_MACHINE\SOFTWARE\Classes`.

Registrierung manuell löschen

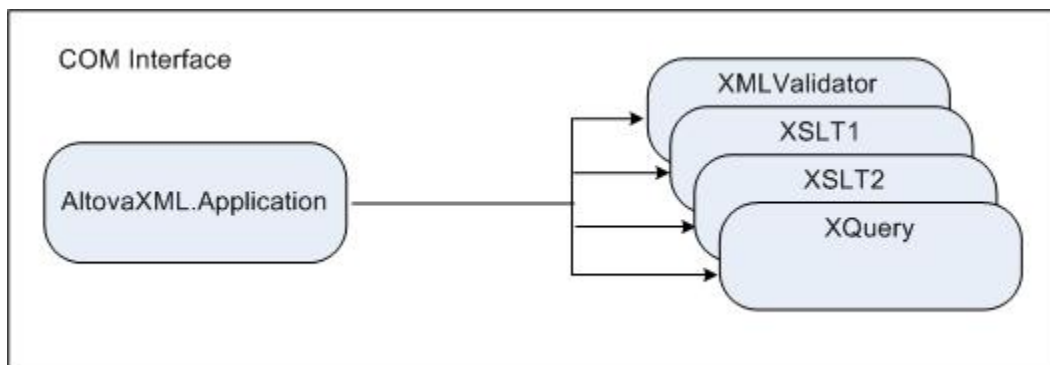
Wenn Sie `AltovaXML_COM.exe` manuell registriert haben und diese Registrierung nun löschen möchten, müssen Sie dies manuell tun. Um die Registrierung von AltovaXML manuell zu löschen, rufen Sie die Applikation über den Parameter `/unregserver` auf. Wenn sich die `AltovaXML.exe`-Datei z.B. im Ordner `c:\AltovaXML` befindet, öffnen Sie die Windows-Eingabeaufforderung und geben Sie ein `c:\AltovaXML\AltovaXML_COM.exe /unregserver` und drücken Sie die Eingabetaste. Sie können im Registrierungs-Editor überprüfen, ob die entsprechenden Klassen gelöscht wurden.

Anmerkung: Wenn AltovaXML vom Installer registriert wurde, sollte die Registrierung vom Installer gelöscht werden, d.h. durch Deinstallation von AltovaXML vom Rechner.

2.2.2 AltovaXML Objektmodell

Um die Funktionalitäten von AltovaXML verwenden zu können, benötigen Sie die Applikationsschnittstelle. Dieses Objekt enthält die vier Objekte, die die AltovaXML-Funktionalitäten bereitstellen: XML-Validierung, XSLT 1.0-Transformationen, XSLT 2.0-Transformationen und XQuery 1.0-Dokumentverarbeitung. Diese Objekte haben zwei Schnittstellen: die Dispatch-Schnittstelle und die Raw-Schnittstelle, sodass sie sowohl in Skriptsprachen als auch in Applikationen verwendet werden können.

Das Objektmodell der AltovaXML API wird im folgenden Diagramm dargestellt.



Im Folgenden sehen Sie die Hierarchie des Objektmodells. Die fünf Schnittstellen werden in den jeweiligen Abschnitten näher beschrieben. Die Eigenschaften und deren Verwendung werden im Abschnitt zu der entsprechenden Schnittstelle behandelt.

- [Applikation](#)
 - [XMLValidator](#)
 - [XSLT1](#)
 - [XSLT2](#)
 - [XQuery](#)

Hinweis:

Beachten Sie die folgenden allgemeinen Hinweise zur Verwendung der COM-Schnittstelle:

- Der Begriff XML-Dokument bezieht sich nicht nur auf ein in einer XML-Datei enthaltenes XML-Dokument, sondern auch auf ein mit der Eigenschaft `InputXMLFromText` erstelltes XML-Dokument.
- Eigenschaften, die als Input einen Ressourcenpfad nehmen, akzeptieren absolute Pfade sowie das HTTP- und FTP-Protokoll.
- Bei Verwendung relativer Pfade durch eine Methode, um eine Ressource zu finden, sollte die Auflösung des relativen Pfads im aufrufenden Modul definiert sein.

2.2.3 Applikation

Beschreibung

AltovaXML.Application ist die Root für alle anderen Objekte. Sie ist das einzige Objekt, das Sie mit der CreateObject Funktion (von VisualBasic) oder anderen ähnlichen COM-Funktionen erstellen können.

Eigenschaften

AltovaXML.Application hat die vier unten aufgelisteten Eigenschaften. Jede dieser Funktionen gibt die Schnittstelle für die jeweilige Komponente zurück. Die Details zu den einzelnen Schnittstellen finden Sie in den entsprechenden Abschnitten weiter unten.

- [XMLValidator](#)
- [XSLT1](#)
- [XSLT2](#)
- [XQuery](#)

Beispiele

Im Folgenden sehen Sie ein Visual Basic-Script, das zuerst das AltovaXML-Objekt erstellt und anschließend die Eigenschaften der Applikationsschnittstelle aufruft.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
    objAltovaXML.XMLValidator.InputXMLFileName =  
"c:\AltovaXML\test.xml"  
    Sheet1.Cells(5, 2) = objAltovaXML.XMLValidator.IsValid  
  
    objAltovaXML.XSLT1.InputXMLFromText = "<?xml version='1.0'  
encoding=' UTF-8' ?><a><b/></a>"  
    objAltovaXML.XSLT1.XSLFileName = "c:\workarea\altova_xml\1.xslt"  
    Sheet1.Cells(6, 2) =  
objAltovaXML.XSLT1.ExecuteAndGetResultAsString  
  
End Sub
```

2.2.4 XMLValidator

Beschreibung

Die `XMLValidator` Schnittstelle stellt Methoden bereit, um Folgendes zu testen:

- die Wohlgeformtheit eines XML-Dokuments.
- die Gültigkeit eines XML-Dokuments entsprechend einer DTD oder eines XML-Schemas, die/das von innerhalb des XML-Dokuments referenziert wird.
- die Gültigkeit eines XML-Dokuments entsprechend einer DTD oder eines XML-Schema, die/das extern über den Code bereitgestellt wird.
- die Gültigkeit eines XBRL-Dokuments entsprechend einer XBRL-Taxonomie (einer `.xsd`-Datei)

All diese Methoden geben den Booleschen Wert `TRUE` oder `FALSE` zurück. *Siehe Beispiele unten.*

Hinweis: Wo String-Inputs als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus definiert werden, der den relativen Pfad auflöst.

Methoden

Es stehen die folgenden Methoden zur Verfügung:

`IsWellFormed`

`IsWellFormed` überprüft die Wohlgeformtheit des XML-Dokuments. Gibt `TRUE` zurück, wenn das XML-Dokument wohlgeformt ist und `FALSE`, wenn es nicht wohlgeformt ist.

`IsValid`

`IsValid` validiert das XML-Dokument gegen die DTD oder das XML-Schema, die/das im XML-Dokument referenziert wird. Gibt `TRUE` zurück, wenn das XML-Dokument gültig ist, `FALSE`, wenn es ungültig ist. Um das Dokument gegen eine DTD oder ein XML-Schema zu validieren, das/die im XML-Dokument nicht referenziert wird, verwenden Sie die Methode `IsValidWithExternalSchemaOrDTD`.

`IsValidWithExternalSchemaOrDTD`

`IsValidWithExternalSchemaOrDTD` validiert das XML-Dokument gegen die DTD oder das XML-Schema, die/das durch eine der folgenden Eigenschaften bereitgestellt wurde: `SchemaFileName`, `DTDFileName`, `SchemaFromText` oder `DTDFromText`. Wurden für mehrere dieser Eigenschaften Werte definiert, so verwendet die Methode `IsValidWithExternalSchemaOrDTD` die als letzte definierte Eigenschaft. Wenn das XML-Dokument gültig ist, wird `TRUE` zurückgegeben, wenn es ungültig ist, `FALSE`. Um das Dokument gegen eine DTD oder ein XML-Schema zu validieren, die/das im XML-Dokument referenziert wird, verwenden Sie die Methode `IsValid`.

Hinweis: Die Validierung und Wohlgeformtheitsprüfung muss immer erfolgen, nachdem das XML-Dokument und/oder die DTD oder das XML-Schema-Dokument den jeweiligen Eigenschaften zugewiesen wurde.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

`InputXMLFileName`

Ein String-Input, der als URL zum Auffinden der XML-Datei gelesen wird, die validiert werden soll.

SchemaFileName

Ein String-Input, der als URL zum Auffinden der XML-Schemadatei gelesen wird, gegen die das XML-Dokument validiert werden soll.

DTDFileName

Ein String-Input, der als URL zum Auffinden der DTD-Datei gelesen wird, gegen die das XML-Dokument validiert werden soll.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt.

SchemaFromText

Ein String-Input, der ein XML-Schema-Dokument erstellt.

DTDFromText

Ein String-Input, der ein DTD-Dokument erstellt.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

TreatXBRLInconsistenciesAsErrors

Gibt semantische XBRL-Inkonsistenzen als Fehler zurück, wenn diese Eigenschaft auf "True" gesetzt wird. Die Standardeinstellung ist `False`.

Beispiele

Im Folgenden sehen Sie eine Visual Basic-Prozedur, die zeigt, wie Methoden und Eigenschaften der `XMLValidator` Schnittstelle verwendet werden können. Dieser Code ist zur Verwendung in einem Makro in einem MS Excel-Arbeitsblatt gedacht und referenziert Zellen im Arbeitsblatt, die den Pfad zu Eingabe- oder Ausgabedaten angeben. Die Datei `c:\AltovaXML\test.xml` enthält in diesem Fall eine Referenz auf eine DTD.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
    objAltovaXML.XMLValidator.InputXMLFromText = "<?xml version='1.0'  
encoding=' UTF-8' ?><a><b/></a>"  
    Sheet1.Cells(4, 2) = objAltovaXML.XMLValidator.IsWellFormed  
  
    objAltovaXML.XMLValidator.InputXMLFileName =  
"c:\AltovaXML\test.xml"  
    Sheet1.Cells(5, 2) = objAltovaXML.XMLValidator.IsValid  
  
    objAltovaXML.XMLValidator.InputXMLFileName =  
"c:\AltovaXML\test.xml"  
    objAltovaXML.XMLValidator.DTDFileName = "c:\AltovaXML\test.dtd"  
    Sheet1.Cells(6, 2) =  
objAltovaXML.XMLValidator.IsValidWithExternalSchemaOrDTD  
  
    objAltovaXML.XMLValidator.InputXMLFromText = "<?xml version='1.0'  
encoding=' UTF-8' ?><a><b/></a>"  
    objAltovaXML.XMLValidator.DTDFileName = "c:\AltovaXML\test.dtd"  
    Sheet1.Cells(7, 2) =  
objAltovaXML.XMLValidator.IsValidWithExternalSchemaOrDTD  
End Sub
```

2.2.5 XSLT1

Beschreibung

Die `XSLT1`-Schnittstelle stellt Methoden und Eigenschaften zum Ausführen einer XSLT 1.0-Transformation mittels des Altova XSLT 1.0-Prozessors bereit. Das Ergebnis kann in einer Datei oder als String zurückgegeben werden. Über die Schnittstelle können außerdem XSLT-Parameter an das XSLT-Stylesheet übergeben werden. Die URLs von XML- und XSLT-Dateien können als Strings über Schnittstelleneigenschaften bereitgestellt werden. Alternativ dazu können XML- und XSLT-Dokumente innerhalb des Script- oder Programmiercodes als Textstrings konstruiert werden. *Siehe Beispiele unten.*

Hinweis: Wo String-Inputs als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus definiert werden, der den relativen Pfad auflöst.

Methoden

Es stehen die folgenden Methoden zur Verfügung:

Execute

`Execute` führt die XSLT 1.0-Transformation aus und speichert das Ergebnis als Ausgabedatei, deren Name und Pfad als Input-String für die `Execute` Methode bereitgestellt wird.

ExecuteAndGetResultAsString

`ExecuteAndGetResultAsString` führt eine XSLT 1.0 Transformation aus und gibt das Ergebnis als UTF-16 Textstring zurück.

AddExternalParameter

Nimmt einen Parameternamen und den Wert dieses Parameters als Input-Argumente. Die einzelnen externen Parameter und deren Werte müssen in einem separaten Aufruf der Methode definiert werden. Bei Angabe eines externen Parameters mit dem Namen eines existierenden (nicht gelöschten) Parameters wird ein Fehler ausgegeben. Da Parameterwerte XPath-Ausdrücke sind, müssen Parameterwerte, die Strings sind, in einfache Anführungszeichen eingeschlossen werden. *Siehe Beispiel unten.*

ClearExternalParameterList

Es sollte kein Argument bereitgestellt werden. Die Methode `ClearExternalParameterList` löscht die Liste der externen Parameter, die mit `AddExternalParameter` Methoden erstellt wurden.

Hinweis: Die Transformation muss immer erfolgen, nachdem das XML-Dokument und das XSLT-Dokument zugewiesen wurden.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

InputXMLFileName

Ein String-Input, der als URL gelesen wird, über die die zu transformierende XML-Datei gefunden wird.

XSLFileName

Ein String-Input, der als URL zum Auffinden der XSLT-Datei gelesen wird, die für die Transformation verwendet werden soll.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt.

XSLFromText

Ein String-Input, der ein XSLT-Dokument erstellt.

XSLStackSize

Die Stack-Größe ist die maximale Tiefe, bis zu der Anweisungen ausgeführt werden sollen. Die Stack-Größe kann über die Eigenschaft `XSLStackSize` geändert werden. Die zulässige Mindest-Stack-Größe ist 100. Die Standard-Stack-Größe ist 1000. Bei Überschreitung der Stack-Größe während einer Transformation wird ein Fehler ausgegeben.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

JavaExtensionsEnabled

Aktiviert die Java-Erweiterungen. Sie können festlegen, ob Java-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

DotNetExtensionsEnabled

Aktiviert die .NET-Erweiterungen. Sie können festlegen, ob .NET-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

Beispiele

Im Folgenden sehen Sie eine Visual Basic-Prozedur, die zeigt, wie die verschiedenen Methoden und Eigenschaften oder XSLT1-Schnittstellen verwendet werden können. Dieser Code dient zur Verwendung als Makro in einem MS Excel-Arbeitsblatt und referenziert Zellen, im Arbeitsblatt, die Pfade zu Input- und Output-Daten definieren.

```
Sub CommandButton1_Click()
Set objAltovaXML = CreateObject("AltovaXML.Application")

    objAltovaXML.XSLT1.InputXMLFromText = "<?xml version='1.0'
encoding=' UTF-8' ?>
    <a><b/></a>"
    objAltovaXML.XSLT1.XSLFileName = "c:\AltovaXML\test.xslt"
    objAltovaXML.XSLT1.Execute "c:\AltovaXML\test_result.xml

    objAltovaXML.XSLT1.XSLStackSize = "500"
    objAltovaXML.XSLT1.InputXMLFromText = "<?xml version='1.0'
encoding=' UTF-8' ?>
    <company><name/><year>2005</year></company>"
    objAltovaXML.XSLT1.XSLFileName = "c:\AltovaXML\test.xslt"
    objAltovaXML.XSLT1.AddExternalParameter "web", "' www.altova.com' "
    objAltovaXML.XSLT1.AddExternalParameter "year", "/company/year"
    Sheet1.Cells(6, 2) =
objAltovaXML.XSLT1.ExecuteAndGetResultAsString
    objAltovaXML.XSLT1.ClearExternalParameterList
    objAltovaXML.XSLT1.AddExternalParameter "web",
"' www.nanonull.com' "
    objAltovaXML.XSLT1.AddExternalParameter "year", "/company/year"
    Sheet1.Cells(7, 2) =
objAltovaXML.XSLT1.ExecuteAndGetResultAsString
End Sub
```

2.2.6 XSLT2

Beschreibung

Die `XSLT2`-Schnittstelle stellt Methoden und Eigenschaften zum Ausführen einer XSLT 2.0-Transformation mittels des Altova XSLT 2.0-Prozessors bereit. Das Ergebnis kann in einer Datei oder als String zurückgegeben werden. Über die Schnittstelle können außerdem XSLT-Parameter an das XSLT-Stylesheet übergeben werden. Die URLs von XML- und XSLT-Dateien können als Strings über Schnitstelleigenschaften bereitgestellt werden. Alternativ dazu können XML- und XSLT-Dokumente innerhalb des Script- oder Programmiercodes als Textstrings konstruiert werden. *Siehe Beispiele unten.*

Hinweis:

- Wo String-Inputs als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus definiert werden, der den relativen Pfad auflöst.
- Der XSLT 2.0-Prozessor kann zur Verarbeitung eines XSLT 1.0 Stylesheet im Rückwärtskompatibilitätsmodus verwendet werden. Die Ausgabe kann sich allerdings von der eines XSLT 1.0-Prozessors, der dasselbe XSLT 1.0 Stylesheet verarbeitet, unterscheiden.

Methoden

Es stehen die folgenden Methoden zur Verfügung:

`Execute`

`Execute` führt die XSLT 2.0-Transformation aus und speichert das Ergebnis als Ausgabedatei, deren Name und Pfad als Input-String für die `Execute` Methode bereitgestellt werden.

`ExecuteAndGetResultAsString`

`ExecuteAndGetResultAsString` führt eine XSLT 2.0 Transformation aus und gibt das Ergebnis als UTF-16 Textstring zurück.

`AddExternalParameter`

Nimmt einen Parameternamen und den Wert dieses Parameters als Input-Argumente. Die einzelnen externen Parameter und deren Werte müssen in einem separaten Aufruf der Methode definiert werden. Bei Angabe eines externen Parameters mit dem Namen eines existierenden (nicht gelöschten) Parameters wird ein Fehler ausgegeben. Da Parameterwerte XPath-Ausdrücke sind, müssen Parameterwerte, die Strings sind, in einfache Anführungszeichen eingeschlossen werden. *Siehe Beispiel unten.* Beachten Sie in den Beispielen, dass der Parameter `date` einen Wert erhält, der eine XPath 2.0-Funktion ist (`current-date()`).

`ClearExternalParameterList`

Es sollte kein Argument bereitgestellt werden. Die Methode `ClearExternalParameterList` löscht die Liste der externen Parameter, die mit `AddExternalParameter` Methoden erstellt wurden.

`InitialTemplateName`

Definiert die anfangs verwendete named Template. Das Argument ist der Name der Vorlage, an der die Verarbeitung beginnen soll.

`InitialTemplateMode`

Definiert den anfangs verwendeten Modus für die Verarbeitung. Das Argument ist der Name des erforderlichen Anfangsmodus. Vorlagen mit diesem Moduswert werden verarbeitet.

Hinweis: Die Transformation muss immer erfolgen, nachdem das XML-Dokument und das XSLT-Dokument zugewiesen wurden.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

InputXMLFileName

Ein String-Input, der als URL gelesen wird, über die die zu transformierende XML-Datei gefunden wird.

XSLFileName

Ein String-Input, der als URL zum Auffinden der XSLT-Datei gelesen wird, die für die Transformation verwendet werden soll.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt.

XSLFromText

Ein String-Input, der ein XSLT-Dokument erstellt.

XSLStackSize

Die Stack-Größe ist die maximale Tiefe, bis zu der Anweisungen ausgeführt werden sollen. Die Stack-Größe kann über die Eigenschaft `XSLStackSize` geändert werden. Die zulässige Mindest-Stack-Größe ist 100. Die Standard-Stack-Größe ist 1000. Bei Überschreitung der Stack-Größe während einer Transformation wird ein Fehler ausgegeben.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

JavaExtensionsEnabled

Aktiviert die Java-Erweiterungen. Sie können festlegen, ob Java-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

DotNetExtensionsEnabled

Aktiviert die .NET-Erweiterungen. Sie können festlegen, ob .NET-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

Beispiele

Im Folgenden sehen Sie eine Visual Basic-Prozedur, die zeigt, wie die verschiedenen Methoden und Eigenschaften oder `XSLT2`-Schnittstellen verwendet werden können. Dieser Code dient zur Verwendung als Makro in einem MS Excel-Arbeitsblatt und referenziert Zellen, im Arbeitsblatt, die Pfade zu Input- und Output-Daten definieren.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
    objAltovaXML.XSLT2.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?>  
    <a><b/></a>"  
    objAltovaXML.XSLT2.XSLFileName = "c:\AltovaXML\test.xslt"  
    Sheet1.Cells(7, 2) =  
objAltovaXML.XSLT2.ExecuteAndGetResultAsString  
  
    objAltovaXML.XSLT2.XSLStackSize = "500"  
    objAltovaXML.XSLT2.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?>  
    <company><name/><year>2005</year></company>"  
    objAltovaXML.XSLT2.XSLFileName = "c:\workarea\AltovaXML\2.xslt"
```

```
objAltovaXML.XSLT2.AddExternalParameter "date", "current-date()"
objAltovaXML.XSLT2.AddExternalParameter "hq", "'Vienna, Austria'"
Sheet1.Cells(8, 2) =
objAltovaXML.XSLT2.ExecuteAndGetResultAsString
objAltovaXML.XSLT2.AddExternalParameter "web",
"'www.nanonull.com'"
objAltovaXML.XSLT2.AddExternalParameter "year", "/company/year"
objAltovaXML.XSLT2.Execute
"c:\workarea\AltovaXML\test_result_xslt2.xml"
Sheet1.Cells(9, 2) =
objAltovaXML.XSLT2.ExecuteAndGetResultAsString
End Sub
```

2.2.7 XQuery

Beschreibung

Die XQuery-Schnittstelle stellt Methoden und Eigenschaften zum Ausführen einer XQuery 1.0-Transformation mittels des Altova XQuery-Prozessors bereit. Das Ergebnis kann in einer Datei oder als String zurückgegeben werden. Über die Schnittstelle können außerdem XQuery-Variablen an das XQuery-Dokument übergeben werden. Die URLs von XQuery- und XML-Dateien können als Strings über Schnittstelleneigenschaften bereitgestellt werden. Alternativ dazu können XML- und XQuery-Dokumente innerhalb des Script- oder Programmiercodes als Textstrings konstruiert werden. *Siehe Beispiele unten.*

Hinweis: Wo String-Inputs als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus definiert werden, der den relativen Pfad auflöst.

Methoden

Es stehen die folgenden Methoden zur Verfügung:

Execute

`Execute` führt die XQuery 1.0-Transformation aus und speichert das Ergebnis als Ausgabedatei, deren Name und Pfad als Input-String für die `Execute` Methode bereitgestellt werden.

ExecuteAndGetResultAsString

`ExecuteAndGetResultAsString` führt eine XQuery 1.0-Transformation aus und gibt das Ergebnis als UTF-16 Textstring zurück.

AddExternalVariable

Nimmt einen Variablennamen und den Wert dieser Variable als Input-Argumente. Die einzelnen externen Variablen und deren Werte müssen in einem separaten Aufruf der Methode definiert werden. Variablen müssen im XQuery-Dokument deklariert werden, optional mit einer Typdeklaration. Unabhängig von der Typdeklaration für die externe Variable im XQuery-Dokument sind für den Variablenwert, der für die Methode `AddExternalVariable` bereitgestellt wird, keine speziellen Trennzeichen wie z.B. Anführungszeichen erforderlich (*siehe Beispiel unten*). Die lexikalische Form muss jedoch der des erwarteten Typs entsprechen (so muss z.B. eine Variable vom Typ `xs:date` einen Wert in der lexikalischen Form `2004-01-31` haben; ein Wert in der lexikalischen Form `2004/Jan/01` verursacht dagegen einen Fehler). Beachten Sie, dass dies auch bedeutet, dass Sie eine XQuery 1.0-Funktion (z.B. `current-date()`) nicht als den Wert einer externen Variable verwenden können (da die lexikalische Form der Funktion in der eigenen Schreibweise entweder nicht dem erforderlichen Datentyp entspricht - wenn der Datentyp in der Deklaration der externen Variable definiert ist - oder als String gelesen wird, wenn der Datentyp nicht definiert ist. Wenn eine externe Variable bereitgestellt wird, die den Namen einer existierenden (nicht gelöschten) Variable hat, so wird ein Fehler ausgegeben.

AddExternalVariableAsXPath

Nimmt einen Variablennamen und den Wert dieser Variable als Input-Argumente. Ist ähnlich wie die Methode `AddExternalVariable`, mit der Ausnahme, dass `AddExternalVariableAsXPath` zuerst als XPath 2.0 Ausdruck ausgewertet wird. Auf diese Art können auch Nodes und Sequenzen mit mehr als einem Element verarbeitet werden.

ClearExternalVariableList

Es sollte kein Argument bereitgestellt werden. Die Methode `ClearExternalVariableList` löscht die Liste der externen Variablen, die mit `AddExternalVariable` Methoden erstellt wurden.

Hinweis: Die Definition des optionalen XML-Dokuments muss immer vor der

XQuery-Ausführung erfolgen.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

XQueryFileName

Ein String-Input, der als URL zum Auffinden der auszuführenden XQuery-Datei gelesen wird. Wenn sowohl die `XQueryFileName` Eigenschaft als auch die `QueryFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

InputXMLFileName

Ein String-Input, der als URL zum Auffinden der in die Query zu ladenden XML-Datei gelesen wird. XQuery Navigationsausdrücke werden in Beziehung auf den Dokumentnode dieses XML-Dokuments ausgewertet. Wenn sowohl die `InputFileName` Eigenschaft als auch die `InputXMLFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

XQueryFromText

Ein String-Input, der ein XQuery-Dokument erstellt. Wenn sowohl die `XQueryFileName` Eigenschaft als auch die `XQueryFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt. XQuery Navigationsausdrücke werden in Bezug auf den Dokument-Node dieses XML-Dokuments ausgewertet. Wenn sowohl die `InputFileName` Eigenschaft als auch die `InputXMLFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

JavaExtensionsEnabled

Aktiviert die Java-Erweiterungen. Sie können festlegen, ob Java-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

DotNetExtensionsEnabled

Aktiviert die .NET-Erweiterungen. Sie können festlegen, ob .NET-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern

Hinweis: Wenn ein XML-Dokument definiert ist und nicht für eine neue XQuery-Ausführung erforderlich ist, sollte dies durch Zuweisung eines leeren Strings gelöscht werden.

Es sind die folgenden Serialisierungsoptionen definiert:

OutputMethod

Die benötigte Ausgabemethode kann durch Bereitstellung des erforderlichen Werts als String-Argument definiert werden. Gültige Werte sind: `xml`, `xhtml`, `html` und `text`. Z.B.: `objAltovaXML.XQuery.OutputMethod = "xml"`. Wenn der Wert ungültig ist, wird er ignoriert. Die Standardausgabemethode ist `xml`.

OutputOmitXMLDeclaration

Sie können angeben, ob die XML-Deklaration in der Ausgabe inkludiert werden soll oder nicht.

Geben Sie dazu `true` oder `false` (Groß- und Kleinschreibung ist irrelevant) als Boolesches Argument an. Z.B.: `objAltovaXML.XQuery.OutputOmitXMLDeclaration = "FALSE"`. Wenn der Wert ungültig ist, wird ein Fehler ausgegeben. Die Standardoption ist `TRUE`.

OutputIndent

Sie können festlegen, ob die Ausgabe Einrückungen enthalten soll oder nicht. Geben Sie dazu `true` oder `false` (Groß- und Kleinschreibung ist irrelevant) als Boolesches Argument an. z.B.: `objAltovaXML.XQuery.OutputIndent = "TRUE"`. Wenn der Wert ungültig ist, wird ein Fehler ausgegeben. Die Standardoption ist `False`.

OutputEncoding

Die erforderliche Ausgabecodierung kann durch Angabe des Codierungswerts als String-Argument definiert werden. Z.B.: `objAltovaXML.XQuery.OutputEncoding = "UTF-8"`. Wenn der Wert ungültig ist, wird er ignoriert. Die Standardausgabecodierung ist `UTF-8`.

Hinweis: Bei den Serialisierungsoptionen gibt es Unterschiede bei der Verwendung der Raw-Schnittstelle und der Dispatch-Schnittstelle. Wenn bei Verwendung der Raw-Schnittstelle kein Argument mit diesen Eigenschaften bereitgestellt wird, so wird der aktuelle Wert der Eigenschaft zurückgegeben. Die Eingabe müsste in etwa so aussehen: `put_OutputOption(VARIANT_BOOL bVal)` bzw. `so VARIANT_BOOL bVal = get_OutputOption()`, um Werte zu setzen und abzurufen. Bei der Dispatch-Schnittstelle können Sie `b = myXQuery.OutputOption` verwenden, um Werte abzurufen und `myXQuery.OutputOption = b` um Werte zu definieren. Bei der Dispatch-Schnittstelle würde z.B. `Sheet1.Cells(10, 2) = objAltovaXML.XQuery.OutputEncoding` die aktuelle Ausgabecodierung abrufen.

Beispiele

Im Folgenden sehen Sie eine einzelne Visual Basic-Prozedur, die zeigt, wie die verschiedenen Methoden und Eigenschaften der `XQuery`-Schnittstelle verwendet werden können. Dieser Code war als Verwendung als Makro in einem MS Excel-Arbeitsblatt gedacht und die Referenzen auf Zellen im Arbeitsblatt geben den Pfad zu Input- oder Output-Daten an.

```
Sub CommandButton1_Click()
Set objAltovaXML = CreateObject("AltovaXML.Application")

objAltovaXML.XQuery.InputXMLFileName = "c:\AltovaXML\test.xml"
objAltovaXML.XQuery.XQueryFromText = " xquery version '1.0';
  declare variable $string as xs:string external;
  declare variable $num as xs:decimal external;
  declare variable $date as xs:date external;
  $string, ' ', 2*$num, ' ', $date "
objAltovaXML.XQuery.AddExternalVariable "string", "A string"
objAltovaXML.XQuery.AddExternalVariable "num", "2.1"
objAltovaXML.XQuery.AddExternalVariable "date", "2005-04-21"
Sheet1.Cells(10, 2) = objAltovaXML.XQuery.OutputEncoding
objAltovaXML.XQuery.OutputMethod = "text"
Sheet1.Cells(11, 2) = objAltovaXML.XQuery.OutputMethod
objAltovaXML.XQuery.OutputIndent = "TRUE"
Sheet1.Cells(12, 2) = objAltovaXML.XQuery.OutputIndent
objAltovaXML.XQuery.OutputOmitXMLDeclaration = "FALSE"
Sheet1.Cells(13, 2) = objAltovaXML.XQuery.OutputOmitXMLDeclaration
Sheet1.Cells(14, 2) =
objAltovaXML.XQuery.ExecuteAndGetResultAsString
End Sub
```

2.2.8 Beispiele

Dieser Abschnitt enthält Beispielcode in (i) Visual Basic für ein Excel Makro; (ii) JScript; und (iii) C++. In diesen Beispielen sehen Sie, wie AltovaXML mit einer COM-Schnittstelle verwendet werden kann.

Nähere Beispiele finden Sie in den Beispieldateien im Applikationsordner im Ordner `Examples`.

Visual Basic

Das folgende Visual Basic-Beispiel ist der Code für ein Makro in einem Excel-Arbeitsblatt (*Screenshot unten*). Das Makro wurde der Schaltfläche `Run Expressions` zugewiesen. Wenn Sie auf die Schaltfläche klicken, wird der Visual Basic-Code ausgeführt.

	A	B
1	XQuery or XML in Application	Result
2	element a {for \$i in (-3 to 3) return -\$i}	<a>3 2 1 0 -1 -2 -3
3	<node>6;154;738-34</node>	6.154.738 34
4		A code-generated string
5	Run Expressions	

Codebeispiel

Der folgende Visual Basic-Code verwendet die `XQuery`-Schnittstelle.

```
Sub CommandButton1_Click()
Set objAltovaXML = CreateObject("AltovaXML.Application")

objAltovaXML.XQuery.XQueryFromText = Sheet1.Cells(2, 1)
Sheet1.Cells(2, 2) =
objAltovaXML.XQuery.ExecuteAndGetResultAsString

objAltovaXML.XQuery.InputXMLFromText = Sheet1.Cells(3, 1)
objAltovaXML.XQuery.XQueryFromText = "translate(node, ';'-'', '.' ')"
Sheet1.Cells(3, 2) =
objAltovaXML.XQuery.ExecuteAndGetResultAsString

objAltovaXML.XQuery.InputXMLFromText = "<a myAttr=' A
code-generated string' />"
objAltovaXML.XQuery.XQueryFromText = "string(/a/@*)"
Sheet1.Cells(4, 2) =
objAltovaXML.XQuery.ExecuteAndGetResultAsString
End Sub
```

Wenn Sie im Excel-Arbeitsblatt auf die Schaltfläche **Run Expressions** klicken, werden die folgenden drei XQuery-Anweisungen ausgeführt:

1. Der Input für die `XQueryFromText` Eigenschaft ist ein XQuery-Ausdruck, der als Text aus dem Excel-Arbeitsblatt Zelle 2A geholt wird. Die `ExecuteAndGetResultAsString` Eigenschaft führt den XQuery-Ausdruck aus und platziert das Ergebnis in das Excel-Arbeitsblatt, Zelle 2B.
2. Der Input für die `InputXMLFromText` Eigenschaft ist ein XML-Fragment aus dem Excel-Arbeitsblatt, Zelle 3A. Der XQuery-Ausdruck wird an die `XQueryFromText` Eigenschaft direkt im Code übergeben. Das Ergebnis wird in das Excel-Arbeitsblatt, Zelle 3 B platziert.

- Die `InputXMLFromText` Eigenschaft erstellt eine XML-Baumstruktur anhand des bereitgestellten XML-Fragments. Der XQuery-Ausdruck wird an die `XQueryFromText` Eigenschaft direkt im Code übergeben und das Ergebnis wird in das Excel-Arbeitsblatt, Zelle 4B platziert.

JScript

Im Folgenden sehen Sie eine JScript-Codedatei, die zeigt, wie AltovaXML über die COM-Schnittstelle verwendet werden kann.

Codebeispiel

```
// ////////////////////////////////// global variables //////////////////////////////////
var objAltovaXML = null;

// ////////////////////////////////// Helpers //////////////////////////////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);

    if (objAltovaXML != null)
        objAltovaXML.Quit();

    WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
    if (objErr != null)
        Exit ("ERROR: (" + (objErr.number & 0xffff) + ") " + objErr.description +
" - " + strText);
    else
        Exit ("ERROR: " + strText);
}

function CreateGlobalObjects ()
{
    // create the AltovaXML connection
    // if there is a running instance of AltovaXML (that never had a
connection) - use it
    // otherwise, we automatically create a new instance
    try
    {
        objAltovaXML = WScript.GetObject("", "AltovaXML.Application");
        //WScript.Echo("Successfully accessing AltovaXML.Application");
    }
    catch(err)
    {
        WScript.Echo(err)
        { Exit("Can't access or create AltovaXML.Application"); }
    }
}

// ////////////////////////////////// MAIN //////////////////////////////////

CreateGlobalObjects();

objAltovaXML.XQuery.InputXMLFromText = " \
<bib> \
<book year=\"1994\"> \
    <title>TCP/IP Illustrated</title> \
    <author><last>Stevens</last><first>W.</first></author> \
```

```

        <publisher>AW</publisher> \
        <price>65.95</price> \
    </book> \
    <book year="1992"> \
        <title>Advanced Programming in the Unix Environment</title> \
        <author><last>Stevens</last><first>W.</first></author> \
        <publisher>AW</publisher> \
        <price>65.95</price> \
    </book> \
    <book year="2000"> \
        <title>Data on the Web</title> \
        <author><last>Abiteboul</last><first>Serge</first></author> \
        <author><last>Abiteboul</last><first>Serge</first></author> \
        <author><last>Abiteboul</last><first>Serge</first></author> \
        <publisher>John Jameson Publishers</publisher> \
        <price>39.95</price> \
    </book> \
    <book year="1999"> \
        <title>Digital TV</title> \

<editor><last>Gassy</last><first>Viktor</first><affiliation>CITI</affiliation>
</editor> \
    <publisher>Kingston Academic Press</publisher> \
    <price>129.95</price> \
</book> \
</bib> ";

```

```

objAltovaXML.XQuery.XQueryFromText = "\
(: Filename: xmpQ1.xq :) \
(: Source: http://www.w3.org/TR/xquery-use-cases/#xmp-data :) \
(: Section: 1.1.1.9 Q1 :) \
(: List books published by AW after 1991, including their year and title.:)
\
<bib> \
{ \
    for $b in /bib/book where $b/publisher = "AW" and $b/@year > 1991 \
        return <book year="{ $b/@year }"> { $b/title } </book>
\
} \
</bib> ";

```

```

var sResult = objAltovaXML.XQuery.ExecuteAndGetResultAsString();
WScript.Echo(sResult);

```

C++

Im Folgenden sehen Sie eine C++-Codedatei, die zeigt, wie AltovaXML über die COM-Schnittstelle verwendet werden kann.

Codebeispiel

```

// TestAltovaXML.cpp : Defines the entry point for the console application.
//
#include "objbase.h"
#include <iostream>
#include "atlbase.h"

#import "AltovaXML_COM.exe" no_namespace raw_interfaces_only
// - or -
// #import "AltovaXML_COM.exe" raw_interfaces_only
// using namespace AltovaXMLLib;

int main(int argc, char* argv[])

```

```

{
    HRESULT hr = S_OK;
    hr = CoInitialize(NULL);
    if ( hr == S_OK )
    {
        IApplicationPtr ipApplication;

        hr = CoCreateInstance(
            __uuidof( Application
),
            NULL,
            CLSCTX_ALL,
            __uuidof( IApplication),

reinterpret_cast<void**>( &ipApplication)
        );

        if ( hr == S_OK )
        {
            IXQueryPtr ipXQuery;
            hr = ipApplication->get_XQuery( &ipXQuery );

            if ( hr == S_OK )
            {
                CComBSTR sXQExpr( "(1 to 10)[. mod 2 != 0]" );
                BSTR bstrResult;

                hr = ipXQuery->put_XQueryFromText( sXQExpr );
                hr = ipXQuery->ExecuteAndGetResultAsString(
&bstrResult );

                std::cout << (char*)_bstr_t(bstrResult) <<
std::endl;

                ipXQuery.Release();
            }

            ipApplication.Release();
        }

        CoUninitialize();
    }
    return 0;
}

```

2.3 Java-Schnittstelle

Über die AltovaXML Java-Schnittstelle (`AltovaXML.jar`) wird mittels Funktionen in der `AltovaXMLLib.dll` eine Verbindung zur AltovaXML COM-Schnittstelle hergestellt. Diese DLL wird bei der Installation von AltovaXML mit dem AltovaXML Installer im Verzeichnis `WINDIR\system32\` installiert. `AltovaXML.jar` enthält das Paket `com.altova.engines`, welches die Altova-Prozessoren enthält.

Installation

Um die Java-Schnittstelle verwenden zu können, muss die Datei `AltovaXML.jar` zur `CLASSPATH`. COM Registrierung hinzugefügt werden. Dies geschieht automatisch bei der Installation. Wenn Sie den Pfad zur Datei `AltovaXML_COM.exe` nach der Installation ändern, müssen Sie AltovaXML durch Ausführen des Befehls `AltovaXML_COM.exe /regserver` als COM-Serverobjekt registrieren. Nähere Informationen dazu finden Sie unter [Registrieren von AltovaXML als COM Serverobjekt](#).

Dokumentation

Dieser Abschnitt enthält eine ausführliche Beschreibung der AltovaXML Java-Schnittstelle und ist auch im HTML-Format im ZIP-Archiv `AltovaXMLJavaDocs.zip`, welches Sie im `AltovaXML2010` Anwendungsordner finden, verfügbar.

Beispiele

Nähere Beispiele finden Sie in den Beispieldateien im Applikationsordner im Ordner `Examples`.

Das `com.altova.engines` Paket

Zur Verwendung der Java-Schnittstelle benötigen Sie das Paket `com.altova.engines`. Dies ist die Java-Schnittstelle für das AltovaXML COM Serverobjekt; sie bietet Zugriff auf den XML Validator und den XSLT 1.0-, den XSLT 2.0- und den XQuery 1.0-Prozessor.

Das Paket `com.altova.engines` stellt über die nativen Funktionen in der im Verzeichnis `WINDIR\system32\` installierten Datei `AltovaXMLLib.dll` die Verbindung mit der AltovaXML COM Schnittstelle her.

Um eine Verbindung mit einer neuen Instanz des AltovaXML COM Serverobjekts herzustellen, verwenden Sie die statische Methode `getInstance()` der Klasse `AltovaXMLFactory`. Aus der Schnittstelle, die zurückgegeben wird, können Sie mittels der Funktion `getENGINEINSTANCEInstance()` den gewünschten Prozessor auswählen.

Im Folgenden finden Sie ein Codebeispiel, bei dem die Java-Schnittstelle verwendet wird:

```
import com.altova.engines.*;

/**
 * Test application for AltovaXML COM components java interface
 */
public class AltovaXMLTest {
    /**
     * public constructor for AltovaXMLTest
     */
    public AltovaXMLTest(){
    }

    /**
     * application main
     */
    public static void main(String[] args) {
```

```
System.out.println("AltovaXML Java Interface Test Application");

//request a COM server object - fails if AltovaXML is not registered
IAltovaXMLFactory objXmlApp = AltovaXMLFactory.getInstance();

if ( objXmlApp != null ) {
    //get interface for the XQuery engine
    IXQuery xquery = objXmlApp.getXQueryInstance();
    //set XQuery statement
    xquery.setXQueryStatement("<doc><a>{1 to 3}</a>This data is
well-formed.</doc>");
    //execute the statement previously set.
    //There was no input XML specified so the initial context is
empty.
    String sres = xquery.executeAndGetResultAsString();
    //release XQuery engine's connection to the COM server object
    xquery.releaseInstance();
    System.out.println(sres);

    IXMLValidator validator = objXmlApp.getXMLValidatorInstance();
    validator.setInputXMLFromText(sres);
    boolean b = validator.isWellFormed();
    if ( b )
        System.out.println("XML data is well-formed.");
    else
        System.out.println("Data is not well-formed.");
    validator.releaseInstance();

    //release Application object connection to the COM server object.
    //After this the COM server object will shut down automatically.
    objXmlApp.releaseInstance();
} else{
    System.out.println("Creating instance of IAltovaXMLFactory
failed.");
    System.out.println("Please make sure AltovaXML.exe is correctly
registered!");
}
}
```

2.3.1 Schnittstellen

Im Folgenden sehen Sie eine Übersicht über die Schnittstellen von `com.altova.engines`. Ausführliche Beschreibungen finden Sie in den jeweiligen Abschnitten.

- [IAltovaXMLEngine](#)
Ausgangsschnittstelle für XML Validator sowie den XSLT 1.0-, XSLT 2.0- und XQuery 1.0-Prozessor.
- [IAltovaXMLFactory](#)
Schnittstelle für AltovaXML COM Object Wrapper.
- [IExecutable](#)
Executable-Schnittstelle für Prozessoren.
- [IReleasable](#)
Schnittstelle für Release-Funktionalität.
- [IXMLValidator](#)
Schnittstelle für XML Validator.
- [IXQuery](#)
Schnittstelle für den XQuery 1.0-Prozessor.
- [IXSLT](#)
Schnittstelle für die XSLT-Prozessoren.

IAltovaXMLEngine

Ausgangsschnittstelle für den XMLValidator und den XSLT 1.0-, XSLT 2.0- und XQuery-Prozessor. Öffentliche Schnittstelle, die [IReleasable](#) erweitert.

Superschnittstelle: [IReleasable](#)

Subschnittstelle: [XMLValidator](#), [IXQuery](#), [IXSLT](#)

Implementierende Klassen: [XMLValidator](#), [XQuery](#), [XSLT1](#), [XSLT2](#)

Methoden

Es sind die folgenden Methoden definiert.

setInputXMLFileName

```
public void setInputXMLFileName(java.lang.String filename)
```

Definiert den Dateinamen für die XML-Input-Daten. Bitte beachten Sie, dass Sie absolute URLs verwenden müssen.

Parameter:

`filename`: eine absolute URL, die den Basispfad zu den XML-Daten angibt.

setInputXMLFromText

```
public void setInputXMLFromText(java.lang.String text)
```

Definiert den Textwert für die XML-Inputdaten. Z.B.: `setInputXMLFromText("<doc><a>text </doc>")`

Parameter:

`text`: ein String, der XML-Daten enthält.

getLastErrorMessage

```
public java.lang.String getLastErrorMessage()
```

Ruft die letzte Fehlermeldung vom Prozessor ab.

Rückgabewert:
ein String, der die letzte Fehlermeldung enthält.

IAltovaXMLFactory

Schnittstelle für AltovaXML COM Object Wrapper. Bietet Zugriff auf die Schnittstellen des XMLValidators, des XSLT 1.0-, XSLT 2.0- und XQuery 1.0-Prozessors. Öffentliche Schnittstelle, die [IReleasable](#) erweitert.

Superschnittstelle: [IReleasable](#)
Implementierende Klassen: [AltovaXMLFactory](#)

Methoden

Es sind die folgenden Methoden definiert.

getXQueryInstance

public [IXQuery](#) **getXQueryInstance()**

Erstellt eine neue Instanz der XQuery-Klasse für die aktuelle XQuery-Prozessor-Instanz. Die Verbindung des Objekts mit dem Prozessor muss nach der Verwendung wieder freigegeben werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion

[releaseInstance\(\)](#).

Rückgabewert:

die [IXQuery](#) Schnittstelle der neu erstellten Klasse.

getXSLT1Instance

public [IXSLT](#) **getXSLT1Instance()**

Erstellt eine neue Instanz der XSLT1-Klasse für die aktuelle XSLT 1.0-Prozessor-Instanz. Die Verbindung des Objekts mit dem Prozessor muss nach der Verwendung wieder freigegeben werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion

[releaseInstance\(\)](#).

Rückgabewert:

die [IXSLT](#) Schnittstelle der neu erstellten Klasse.

getXSLT2Instance

public [IXSLT](#) **getXSLT2Instance()**

Erstellt eine neue Instanz der XSLT2-Klasse für die aktuelle XSLT 2.0-Prozessor-Instanz. Die Verbindung des Objekts mit dem Prozessor muss nach der Verwendung wieder freigegeben werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion

[releaseInstance\(\)](#).

Rückgabewert:

die [IXSLT](#) Schnittstelle der neu erstellten Klasse.

getXMLValidatorInstance

public [IXMLValidator](#) **getXMLValidatorInstance()**

Erstellt eine neue Instanz der XMLValidator-Klasse für die aktuelle XML Validator-Instanz. Die Verbindung des Objekts mit dem Prozessor muss nach der Verwendung wieder freigegeben werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion

[releaseInstance\(\)](#).

Rückgabewert:

die [IXMLValidator](#) Schnittstelle der neu erstellten Klasse.

IExecutable

Executable-Schnittstelle für Prozessoren. Öffentliche Schnittstelle.

Subschnittstelle: [IXQuery](#), [IXSLT](#)

Implementierende Klassen: [XQuery](#), [XSLT1](#), [XSLT2](#)

Methoden

Es sind die folgenden Methoden definiert.

execute

```
public boolean execute(java.lang.String outfilename)
```

Führt das Ergebnis aus und speichert es in einer Datei. Bei einem Fehler können Sie die in [IAltovaXMLEngine](#) deklarierte Funktion [getLastErrorMessage\(\)](#) verwenden, um zusätzliche Informationen abzurufen.

Parameter:

outfilename: eine absolute URL, die den Pfad der Ausgabedatei angibt.

Rückgabewert:

true bei erfolgreicher Ausführung, false bei Fehler.

executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()
```

Führt das Ergebnis aus und speichert es als String. Bei einem Fehler können Sie die in [IAltovaXMLEngine](#) deklarierte Funktion [getLastErrorMessage\(\)](#) verwenden, um zusätzliche Informationen abzurufen.

Rückgabewert:

String, der das serialisierte Ergebnis enthält. Bei einem Fehler wird der leere String zurückgegeben.

enableJavaExtensions

```
public void enableJavaExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die Java-Erweiterungsfunktionen.

enableDotNetExtensions

```
public void enableDotNetExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die .NET-Erweiterungsfunktionen.

IReleasable

Öffentliche Schnittstelle für Release-Funktionalität. Wenn ein Objekt, das diese Schnittstelle implementiert, nicht mehr verwendet wird, muss die Funktion `releaseInstance()` aufgerufen werden, um die Verbindung mit dem COM Server zu trennen. Sobald alle Verbindungen zum COM-Server getrennt sind, wird der COM-Server automatisch heruntergefahren.

Subschnittstelle: [IXQuery](#), [IXSLT](#)

Implementierende Klassen: [XQuery](#), [XSLT1](#), [XSLT2](#)

Methoden

Es sind die folgenden Methoden definiert.

releaseInstance

```
public void releaseInstance()
```

Trennt die Verbindung des Objekts zum COM-Server.

IXMLValidator

Schnittstelle für den XML Validator. Öffentliche Schnittstelle, die [IAltovaXMLEngine](#) erweitert.

Superschnittstelle: [IAltovaXMLEngine](#), [IReleasable](#)

Implementierende Klassen: [XMLValidator](#)

Methoden

Es sind die folgenden Methoden definiert.

isValid

```
public boolean isValid()
```

Validiert die XML-Input-Daten gegen die/das darin definierte DTD/Schema.

Rückgabewert:

`true` bei erfolgreicher Validierung, `false` bei fehlgeschlagener Validierung. Wenn die Validierung fehlschlägt, können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

isWellFormed

```
public boolean isWellFormed()
```

Überprüft die XML-Input-Daten auf Wohlgeformtheit.

Rückgabewert:

`true` bei Erfolg, `false` bei Fehler. Wenn bei der Überprüfung ein Fehler gefunden wird, können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

isValidWithExternalSchemaOrDTD

```
public boolean isValidWithExternalSchemaOrDTD()
```

Validiert die XML-Input-Daten gegen die externe DTD/das externe Schema, die/das mit den Funktionen `setDTDFileName()`, `setDTDFromText()`, `setSchemaFileName()`, `setSchemaFromText()` definiert werden kann.

Rückgabewert:

`true` bei Erfolg, `false` bei Fehler. Wenn bei der Überprüfung ein Fehler gefunden wird, können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

setSchemaFileName

```
public void setSchemaFileName(java.lang.String filename)
```

Definiert den Dateinamen für das externe Schema.

Parameter:

`filename`: eine absolute URL, die den Basispfad für das Schema angibt.

setDTDFileName

```
public void setDTDFilename(java.lang.String filename)
```

Definiert den Dateinamen für die externe DTD.

Parameter:

filename: eine absolute URL, die den Basispfad für die DTD angibt.

setSchemaFromText

```
public void setSchemaFromText(java.lang.String text)
```

Definiert den Textwert für das externe Schema.

Parameter:

text: ein String, der das Schema als Text enthält

setDTDFromText

```
public void setDTDFromText(java.lang.String text)
```

Definiert den Textwert für die externe DTD.

Parameter:

text: String, der die DTD als Text enthält.

IXQuery

Schnittstelle für den XQuery-Prozessor. Öffentliche Schnittstelle, die [IAltovaXMLEngine](#) und [IExecutable](#) erweitert.

Superschnittstelle: [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#)

Implementierende Klassen: [XQuery](#)

Methoden

Es sind die folgenden Methoden definiert.

setXQueryFileName

```
public void setXQueryFileName(java.lang.String filename)
```

Definiert den Dateinamen des XQuery-Dokuments.

Parameter:

filename: eine absolute URL, die den Basispfad zur XQuery-Datei angibt.

setXQueryStatement

```
public void setXQueryStatement(java.lang.String text)
```

Definiert den Textwert der XQuery-Anweisung.

Parameter:

text: ein String, der die XQuery-Anweisung enthält.

setOutputEncoding

```
public void setOutputEncoding(java.lang.String encoding)
```

Definiert die Kodierung des Ergebnisdokuments.

Parameter:

encoding: ein String, der den Namen der Kodierung enthält (z.B.: UTF-8, UTF-16, ASCII, 8859-1, 1252).

getOutputEncoding

```
public java.lang.String getOutputEncoding()
```

Ruft die für das Ergebnisdokument angegebene Kodierung ab.

Rückgabewert:

ein String, der einen Kodierungsnamen enthält.

setOutputIndent

```
public void setOutputIndent(boolean indent)
```

Aktiviert/deaktiviert die Option zum Einrücken des Texts im Ergebnisdokument.

Parameter:

`indent`: Boolescher Wert zum Aktivieren/Deaktivieren der Einrückung in der Ausgabe.

getOutputIndent

```
public boolean getOutputIndent()
```

Ruft ab, ob die Ausgabe für das Ergebnisdokument Einrückungen enthalten soll oder nicht.

Rückgabewert:

Boolescher Wert, der angibt, ob die Ausgabe eingerückt werden soll (`true`) oder nicht (`false`).

setOutputMethod

```
public void setOutputMethod(java.lang.String method)
```

Definiert die Serialisierungsmethode für das Ergebnisdokument.

Parameter:

`method`: ein String, der die Serialisierungsmethode enthält. (Gültige Werte sind: `xml`, `xhtml`, `html`, `text`).

getOutputMethod

```
public java.lang.String getOutputMethod()
```

Ruft die Serialisierungsmethode für das Ergebnisdokument ab.

Rückgabewert:

ein String, der die Serialisierungsmethode für das Ausgabedokument enthält.

setOutputOmitXMLDeclaration

```
public void setOutputOmitXMLDeclaration(boolean decl)
```

Aktiviert/deaktiviert die Serialisierungsoption `omitXMLDeclaration` für das Ergebnisdokument.

Parameter:

`decl`: neuer Boolescher Wert für den Parameter `omit-xml-declaration`.

getOutputOmitXMLDeclaration

```
public boolean getOutputOmitXMLDeclaration()
```

Ruft den Wert der Option `omitXMLDeclaration` ab, der für das Ergebnisdokument definiert wurde.

Rückgabewert:

Boolescher Wert, der angibt, ob das Ergebnisdokument eine XML-Deklaration enthält (`true`) oder nicht (`false`).

addExternalVariable

```
public void addExternalVariable(java.lang.String name,  
                               java.lang.String val)
```

Fügt einen Namen und Wert für eine externe Variable hinzu.

Parameter:

`name`: ein String, der einen gültigen QName als Variablennamen enthält.

`val`: ein String, der den Wert der Variable enthält; der Wert wird als String verwendet.

addExternalVariableAsXPath

```
public void addExternalVariableAsXPath(java.lang.String name,  
                                         java.lang.String val)
```

Fügt einen Namen und Wert für eine externe Variable hinzu, wobei der Wert als XPath 2.0-Ausdruck ausgewertet wird.

Parameter:

`name`: ein String, der einen gültigen QName als Variablennamen enthält..

`val`: ein String, der den Wert der Variable enthält; der Wert wird als XPath 2.0-Ausdruck ausgewertet.

clearExternalVariableList

```
public void clearExternalVariableList()
```

Löscht die Liste der externen Variablen.

IXSLT

Schnittstelle für XSLT-Prozessoren. Öffentliche Schnittstelle, die [IAltovaXMLEngine](#) und [IExecutable](#) erweitert.

Superschnittstelle: [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#)

Implementierende Klassen: [XSLT1](#) und [XSLT2](#)

Hinweis:

Der XSLT 2.0-Prozessor kann zur Verarbeitung eines XSLT 1.0 Stylesheet im Rückwärtskompatibilitätsmodus verwendet werden. Die Ausgabe kann sich allerdings von der eines XSLT 1.0-Prozessors, der dasselbe XSLT 1.0 Stylesheet verarbeitet, unterscheiden.

Methoden

Es sind die folgenden Methoden definiert.

setXSLTFileName

```
public void setXSLTFileName(java.lang.String name)
```

Definiert den Dateinamen für die XSLT-Daten.

Parameter:

`name`: eine absolute URL, die den Basispfad zur XSLT-Datendatei angibt.

setXSLTFromText

```
public void setXSLTFromText(java.lang.String text)
```

Definiert den Textwert für die XSLT-Daten.

Parameter:

`text`: ein String, der serialisierte XSLT-Daten enthält.

addExternalParameter

```
public void addExternalParameter(java.lang.String name,  
                                   java.lang.String val)
```

Fügt den Namen und Wert eines externen Parameters hinzu.

Parameter:

`name`: ein String, der einen gültigen QName als Parameternamen hat.

`val`: ein String, der den Wert des Parameters enthält. Der Wert wird als ein XPath-Ausdruck

ausgewertet.

clearExternalParameterList

```
public void clearExternalParameterList()
```

Löscht die Liste der externen Parameter.

setXSLTStackSize

```
public void addExternalParameter(long nVal)
```

Die Stack-Größe definiert, bis zu welcher Tiefe Anweisungen ausgeführt werden sollen. Wird die Stack-Größe während einer Transformation überschritten, wird ein Fehler zurückgegeben.

Parameter:

nVal: numerischer Wert für die neue Stack-Größe. Muss größer als 100 sein. Der anfangs eingestellte Wert ist 1000.

2.3.2 Klassen

Im Folgenden sehen Sie eine Übersicht über die Klassen von `com.altova.engines`. Ausführliche Beschreibungen dazu finden Sie in den entsprechenden Abschnitten.

- [AltovaXMLFactory](#)
Erstellt über einen nativen Aufruf eine neue AltovaXML COM Serverobjekt-Instanz und bietet Zugriff auf die AltovaXML Prozessoren.
- [XMLValidator](#)
Klasse, die den XMLValidator enthält.
- [XQuery](#)
Klasse, die den XQuery 1.0-Prozessor enthält.
- [XSLT1](#)
Klasse, die den XSLT 1.0-Prozessor enthält.
- [XSLT2](#)
Klasse, die den XSLT 2.0-Prozessor enthält.

AltovaXMLFactory

Öffentliche Klasse `AltovaXMLFactory`
erweitert `java.lang.Object`
implementiert [IAltovaXMLFactory](#)

Implementierte Schnittstellen: [IAltovaXMLFactory](#), [IReleasable](#)

Beschreibung

Erstellt mittels native Call ein neues AltovaXML COM Serverobjekt und bietet Zugriff auf die AltovaXML-Prozessoren. Die Beziehung zwischen `AltovaXMLFactory` und dem AltovaXML COM-Objekt ist eine 1:1-Beziehung. Das bedeutet, dass anschließende Aufrufe der `getENGINENameInstance()` Funktion Schnittstellen für dieselbe Prozessorinstanz zurückgeben.

Methoden

Es sind die folgenden Methoden definiert.

getInstance

`public static IAltovaXMLFactory getInstance()`
Erstellt ein neues `AltovaXMLFactory` Objekt und verbindet es mit einem neuen AltovaXML COM Serverobjekt.

Rückgabewert:

die Schnittstelle [IAltovaXMLFactory](#) für das neu erstellte `AltovaXMLFactory` Objekt oder Null, wenn die Erstellung des COM-Objekts fehlgeschlagen ist. Im zweiten Fall sollten Sie sicher stellen, dass `AltovaXML.exe` ordnungsgemäß als COM-Serverobjekt [registriert ist](#).

releaseInstance

`public void releaseInstance()`
Gibt die Verbindung des Objekts mit dem COM-Server frei.

Wird definiert durch:

[releaseInstance](#) in der Schnittstelle [IReleasable](#).

getXQueryInstance


```
public IXQuery getXQueryInstance()
```

Erstellt eine neue Instanz einer XQuery-Klasse für die aktuelle XQuery-Prozessorinstanz. Nach Verwendung muss die Verbindung des Objekts mit dem Prozessor wieder gelöst werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion [releaseInstance\(\)](#).

Wird definiert durch:

[getXQueryInstance](#) in der Schnittstelle [IAltovaXMLFactory](#).

Rückgabewert:

die [IXQuery](#) Schnittstelle der neu erstellten Klasse.

getXSLT1Instance

```
public IXSLT getXSLT1Instance()
```

Erstellt eine neue Instanz der [XSLT1](#) Klasse für die aktuelle XSLT 1.0 Prozessorinstanz. Nach Verwendung muss die Verbindung des Objekts mit dem Prozessor wieder gelöst werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion [releaseInstance\(\)](#).

Wird definiert durch:

[getXSLT1Instance](#) in den Schnittstelle [IAltovaXMLFactory](#).

Rückgabewert:

die [IXSLT](#) Schnittstelle der neu erstellten Klasse.

getXSLT2Instance

```
public IXSLT getXSLT2Instance()
```

Erstellt eine neue Instanz der [XSLT2](#) Klasse für die aktuelle XSLT 2.0 Prozessorinstanz. Nach Verwendung muss die Verbindung des Objekts mit dem Prozessor wieder gelöst werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion [releaseInstance\(\)](#).

Wird definiert durch:

[getXSLT2Instance](#) in der Schnittstelle [IAltovaXMLFactory](#).

Rückgabewert:

die [IXSLT](#) Schnittstelle der neu erstellten Klasse.

getXMLValidatorInstance

```
public IXMLValidator getXMLValidatorInstance()
```

Erstellt eine neue Instanz der [XMLValidator](#) Klasse für die aktuelle XML Validator-Instanz. Nach Verwendung muss die Verbindung des Objekts mit dem Prozessor wieder gelöst werden. Verwenden Sie dazu die in der [IReleasable](#) Schnittstelle deklarierte Funktion [releaseInstance\(\)](#).

Wird definiert durch:

[getXMLValidatorInstance](#) in der Schnittstelle [IAltovaXMLFactory](#).

Rückgabewert:

die [IXMLValidator](#) Schnittstelle der neu erstellten Klasse.

XMLValidator

Öffentliche Klasse XMLValidator
erweitert java.lang.Object
implementiert [IXMLValidator](#)

Implementierte Schnittstellen: [IAltovaXMLEngine](#), [IReleasable](#), [IXMLValidator](#)

Beschreibung

Klasse, die den XMLValidator enthält. Keine direkte Konstruktion/kein direkter Zugriff möglich. Erlaubt durch Aufruf der Funktion [getXMLValidatorInstance\(\)](#) auf eine Instanz von [IAltovaXMLFactory](#) Zugriff auf die [IXMLValidator](#) Schnittstelle.

Konstruktoren

Es ist der folgende Konstruktor definiert.

XMLValidator

geschützter **XMLValidator**(lang nValidatorPtr)

Methoden

Die folgenden Methoden sind definiert.

releaseInstance

public void **releaseInstance**()

Gibt die Verbindung des Objekts zum COM Server frei.

Definiert durch:

[releaseInstance](#) in Schnittstelle [IReleasable](#).

setInputXMLFileName

public void **setInputXMLFileName**(java.lang.String str)

Definiert den Dateinamen für die XML-Eingabedaten. Beachten Sie, dass absolute URLs verwendet werden müssen.

Definiert durch:

[setInputXMLFileName](#) in Schnittstelle [IAltovaXMLEngine](#).

Parameter:

str: eine absolute URL, die den Basisordner der XML-Daten angibt.

setInputXMLFromText

public void **setInputXMLFromText**(java.lang.String str)

Definiert den Textwert für die XML-Eingabedaten. Beispiel: `setInputXMLFromText("<doc> <a>text </doc>")`

Definiert durch:

[setInputXMLFromText](#) in der Schnittstelle [IAltovaXMLEngine](#).

Parameter:

str: ein String, der XML-Daten enthält.

getLastErrorMessage

public java.lang.String **getLastErrorMessage**()

Ruft die letzte Fehlermeldung vom Prozessor ab.

Definiert durch:

[getLastErrorMessage](#) in Schnittstelle [IAltovaXMLEngine](#).

Rückgabewert:

ein String, der die letzte Fehlermeldung enthält.

isValid

public boolean **isValid**()

Validiert die XML-Eingabedaten gegen die darin definierte DTD/das Schema.

Definiert durch:

[isValid](#) in Schnittstelle [IXMLValidator](#).

Rückgabewert:

`true` bei erfolgreicher Validierung, `false` bei fehlgeschlagener Validierung. Bei einer fehlgeschlagenen Validierung können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage](#) zusätzliche Informationen abrufen.

isWellFormed

```
public boolean isWellFormed()
```

Überprüft die XML-Eingabedaten auf Wohlgeformtheit.

Definiert durch:

[isWellFormed](#) in Schnittstelle [IXMLValidator](#).

Rückgabewert:

`true` bei Wohlgeformtheit, `false` bei Fehler. Falls die Wohlgeformtheitsprüfung fehlschlägt, können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage](#) zusätzliche Informationen abrufen.

isValidWithExternalSchemaOrDTD

```
public boolean isValidWithExternalSchemaOrDTD()
```

Validiert die XML-Eingabedaten gegen die externe DTD / das externe Schema, welche mit Hilfe der Funktionen `setDTDFileName()`, `setDTDFromText()`, `setSchemaFileName()` und `setSchemaFromText()` definiert werden können. *Eine Beschreibung dieser Funktionen finden Sie weiter unten.*

Definiert durch:

[isValidWithExternalSchemaOrDTD](#) in Schnittstelle [IXMLValidator](#).

Rückgabewert:

`true` bei erfolgreicher Validierung, `false` bei fehlgeschlagener Validierung. Wenn die Validierung fehlschlägt, können Sie mit Hilfe der in [IAltovaXMLEngine](#) definierten Funktion [getLastErrorMessage](#) zusätzliche Informationen abrufen.

setSchemaFileName

```
public void setSchemaFileName(java.lang.String str)
```

Definiert den Dateinamen des externen Schemas.

Definiert durch:

[setSchemaFileName](#) in Schnittstelle [IXMLValidator](#).

Parameter:

`str`: eine absolute URL, die den Basisordner des Schemas angibt.

setDTDFileName

```
public void setDTDFileName(java.lang.String str)
```

Definiert den Dateinamen der externen DTD.

Definiert durch:

[setDTDFileName](#) in Schnittstelle [IXMLValidator](#).

Parameter:

`str`: eine absolute URL, die den Basisordner der DTD angibt.

setSchemaFromText

```
public void setSchemaFromText(java.lang.String str)
```

Definiert den Textwert für das externe Schema.

Definiert durch:

[setSchemaFromText](#) in Schnittstelle [IXMLValidator](#).

Parameter:

str: ein String, der Schema als Text enthält.

setDTDFromText

```
public void setDTDFromText(java.lang.String str)
```

Definiert den Textwert für die externe DTD.

Definiert durch:

[setDTDFromText](#) in Schnittstelle [IXMLValidator](#).

Parameter:

str: ein String, der DTD als Text enthält.

XQuery

Öffentliche Klasse XQuery
erweitert java.lang.Object
implementiert [IXQuery](#)

Implementierte Schnittstellen: [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#), [IXQuery](#)

Beschreibung

Klasse, die den XQuery-Prozessor enthält. Keine direkte Konstruktion/kein direkter Zugriff möglich. Erlaubt durch Aufruf der Funktion [getXQueryInstance\(\)](#) auf eine Instanz von [IAltovaXMLFactory](#) Zugriff auf die [IXQuery](#) Schnittstelle.

Konstruktoren

Es ist der folgende Konstruktor definiert.

XQuery

```
geschützt XQuery(lang nXQueryPtr)
```

Methoden

Die folgenden Methoden sind definiert.

releaseInstance

```
public void releaseInstance()
```

Gibt die Verbindung des Objekts zum COM Server frei.

Definiert durch:

[releaseInstance](#) in Schnittstelle [IReleasable](#).

execute

```
public boolean execute(java.lang.String sOutFile)
```

Führt die Abfrage aus und speichert das Ergebnis in einer Datei. Bei einem Fehler können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

Definiert durch:

[execute](#) in Schnittstelle [IExecutable](#).

Parameter:

sOutFile: eine absolute URL, die den Ordner der Ausgabedatei angibt.

Rückgabewert:

`true` bei erfolgreicher Ausführung, `false` bei Fehler

executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()
```

Führt die Abfrage durch und gibt das Ergebnis in Form eines UTF 16-Textstring zurück. Bei einem Fehler können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

Definiert durch:

[executeAndGetResultAsString](#) in Schnittstelle [IExecutable](#).

Rückgabewert:

String, der das serialisierte Ergebnis enthält. Bei einem Fehler wird der leere String zurückgegeben.

setInputXMLFileName

```
public void setInputXMLFileName(java.lang.String str)
```

Definiert den Dateinamen für die XML-Eingabedaten. Beachten Sie, dass absolute URLs verwendet werden müssen.

Definiert durch:

[setInputXMLFileName](#) in Schnittstelle [IAltovaXMLEngine](#).

Parameter:

`str`: eine absolute URL, die den Basisordner der XML-Daten angibt.

setInputXMLFromText

```
public void setInputXMLFromText(java.lang.String str)
```

Definiert den Textwert für die XML-Eingabedaten. Beispiel: `setInputXMLFromText("<doc> <a>text </doc>")`

Definiert durch:

[setInputXMLFromText](#) in der Schnittstelle [IAltovaXMLEngine](#).

Parameter:

`str`: ein String, der XML-Daten enthält.

getLastErrorMessage

```
public java.lang.String getLastErrorMessage()
```

Ruft die letzte Fehlermeldung vom Prozessor ab.

Definiert durch:

[getLastErrorMessage](#) in Schnittstelle [IAltovaXMLEngine](#).

Rückgabewert:

ein String, der die letzte Fehlermeldung enthält.

setXQueryFileName

```
public void setXQueryFileName(java.lang.String str)
```

Definiert den Dateinamen des XQuery-Dokuments.

Definiert durch:

[setXQueryFileName](#) in Schnittstelle [IXQuery](#).

Parameter:

`str`: eine absolute URL, die den Basisordner der XQuery-Datei angibt.

setXQueryStatement

```
public void setXQueryStatement(java.lang.String str)
```

Definiert den Textwert für die XQuery-Anweisung

Definiert durch:

[setXQueryStatement](#) in Schnittstelle [IXMLValidator](#).

Parameter:

`str`: ein String, der die XQuery-Anweisung als Text enthält.

setOutputEncoding

```
public void setOutputEncoding(java.lang.String str)
```

Definiert die Kodierung für das Ergebnisdokument.

Definiert durch:

[setOutputEncoding](#) in Schnittstelle [IXQuery](#).

Parameter:

`str`: ein String der den Namen der Kodierung enthält (z.B.: UTF-8, UTF-16, ASCII, 8859-1, 1252)

getOutputEncoding

```
public java.lang.String getOutputEncoding()
```

Ruft die für das Ergebnisdokument definierte Kodierung auf.

Definiert durch:

[getOutputEncoding](#) in Schnittstelle [IXQuery](#).

Rückgabewert:

ein String, der den Namen der Kodierung enthält.

setOutputIndent

```
public void setOutputIndent(boolean bVal)
```

Aktiviert/deaktiviert die Einrückoption für das Ergebnisdokument.

Definiert durch:

[setOutputIndent](#) in Schnittstelle [IXQuery](#).

Parameter:

`bVal`: Boolescher Wert zum Aktivieren/Deaktivieren der Einrückung.

getOutputIndent

```
public boolean getOutputIndent()
```

Ruft die für das Ergebnisdokument definierte Einrückoption für das Ausgabedokument ab.

Definiert durch:

[getOutputIndent](#) in Schnittstelle [IXQuery](#).

Rückgabewert:

der aktuelle Wert des Serialisierungsparameters für die Einrückung.

setOutputMethod

```
public void setOutputMethod(java.lang.String str)
```

Definiert die Serialisierungsmethode für das Ergebnisdokument.

Definiert durch:

[setOutputMethod](#) in Schnittstelle [IXQuery](#).

Parameter:

`str`: ein String, der die Serialisierungsmethode enthält. Gültige Werte: xml, xhtml, html, text.

getOutputMethod

```
public java.lang.String getOutputMethod()
```

Ruft die Serialisierungsmethode für das Ergebnisdokument ab.
Definiert durch:
[getOutputMethod](#) in Schnittstelle [IXQuery](#).
Rückgabewert:
die aktuelle Serialisierungsmethode.

```
setOutputOmitXMLDeclaration
```

```
public void setOutputOmitXMLDeclaration(boolean bVal)
```

Aktiviert/deaktiviert die Serialisierungsoption `omitXMLDeclaration` für das Ergebnisdokument.
Definiert durch:
[setOutputOmitXMLDeclaration](#) in Schnittstelle [IXQuery](#).
Parameter:
`bVal`: ein neuer Boolescher Wert für den `omit-xml-declaration` Parameter.

```
getOutputOmitXMLDeclaration
```

```
public boolean getOutputOmitXMLDeclaration()
```

Ruft den Wert der Option `omitXMLDeclaration`, die für das Ergebnisdokument definiert wurde, ab.
Definiert durch:
[getOutputOmitXMLDeclaration](#) in Schnittstelle [IXQuery](#).
Rückgabewert:
Boolescher Wert des `omit-xml-declaration` Parameters.

```
addExternalVariable
```

```
public void addExternalVariable(java.lang.String strName,  
                                java.lang.String strVal)
```

Fügt den Namen und Wert einer externen Variable hinzu.
Definiert durch:
[addExternalVariable](#) in Schnittstelle [IXQuery](#).
Parameter:
`strName`: ein String, der einen gültigen QName als Variablennamen, enthält.
`strVal`: ein String, der den Wert der Variablen enthält; dieser Wert wird als String verwendet.

```
addExternalVariableAsXPath
```

```
public void addExternalVariableAsXPath(java.lang.String strName,  
                                        java.lang.String strVal)
```

Fügt den Namen und Wert für eine externe Variable hinzu, wobei der Wert als XPath 2.0-Ausdruck ausgewertet wird.
Definiert durch:
[addExternalVariableAsXPath](#) in Schnittstelle [IXQuery](#).
Parameter:
`strName`: ein String, der einen gültigen QName als Variablennamen enthält.
`strVal`: ein String, der den Wert der Variablen enthält; dieser Wert wird als XPath 2.0-Ausdruck ausgewertet.

```
clearExternalVariableList
```

```
public void clearExternalVariableList()
```

Löscht die Liste der externen Variablen.

Definiert durch:

[clearExternalVariableList](#) in Schnittstelle [IXQuery](#).

enableJavaExtensions

```
public void enableJavaExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die Java-Erweiterungsfunktionen.

Wird definiert durch:

[enableJavaExtensions](#) in der Schnittstelle [IExecutable](#).

enableDotNetExtensions

```
public void enableDotNetExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die .NET-Erweiterungsfunktionen.

Wird definiert durch:

[enableJavaExtensions](#) in der Schnittstelle [IExecutable](#).

XSLT1

Öffentliche Klasse XSLT1
erweitert java.lang.Object
implementiert [IXSLT](#)

Implementierte Schnittstellen: [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#), [IXSLT](#)

Beschreibung

Klasse, die den XSLT 1.0-Prozessor enthält. Keine direkte Konstruktion/kein direkter Zugriff möglich. Erlaubt durch Aufruf der Funktion [getXSLT1Instance\(\)](#) auf eine Instanz von [IAltovaXMLFactory](#) Zugriff auf die [IXSLT](#) Schnittstelle.

Konstruktoren

Es ist der folgende Konstruktor definiert.

XSLT1

```
geschützt XSLT1(lang nXSLT1Ptr)
```

Methoden

Die folgenden Methoden sind definiert.

releaseInstance

```
public void releaseInstance()
```

Gibt die Verbindung des Objekts zum COM Server frei.

Definiert durch:

[releaseInstance](#) in Schnittstelle [IReleasable](#).

execute

```
public boolean execute(java.lang.String sOutFile)
```

Führt die Abfrage aus und speichert das Ergebnis in einer Datei. Bei einem Fehler können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

Definiert durch:

[execute](#) in Schnittstelle [IExecutable](#).

Parameter:

sOutFile: eine absolute URL, die den Ordner der Ausgabedatei angibt.

Rückgabewert:

`true` bei erfolgreicher Ausführung, `false` bei Fehler

executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()
```

Führt die Anweisung durch und gibt das Ergebnis in Form eines UTF 16-Textstring zurück. Bei einem Fehler können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

Definiert durch:

[executeAndGetResultAsString](#) in Schnittstelle [IExecutable](#).

Rückgabewert:

String, der das serialisierte Ergebnis enthält. Bei einem Fehler wird der leere String zurückgegeben.

setInputXMLFileName

```
public void setInputXMLFileName(java.lang.String str)
```

Definiert den Dateinamen für die XML-Eingabedaten. Beachten Sie, dass absolute URLs verwendet werden müssen.

Definiert durch:

[setInputXMLFileName](#) in Schnittstelle [IAltovaXMLEngine](#).

Parameter:

`str`: eine absolute URL, die den Basisordner der XML-Daten angibt.

setInputXMLFromText

```
public void setInputXMLFromText(java.lang.String str)
```

Definiert den Textwert für die XML-Eingabedaten. Beispiel: `setInputXMLFromText("<doc> <a>text </doc>")`

Definiert durch:

[setInputXMLFromText](#) in der Schnittstelle [IAltovaXMLEngine](#).

Parameter:

`str`: ein String, der XML-Daten enthält.

getLastErrorMessage

```
public java.lang.String getLastErrorMessage()
```

Ruft die letzte Fehlermeldung vom Prozessor ab.

Definiert durch:

[getLastErrorMessage](#) in Schnittstelle [IAltovaXMLEngine](#).

Rückgabewert:

ein String, der die letzte Fehlermeldung enthält.

setXSLTFileName

```
public void setXSLTFileName(java.lang.String str)
```

Definiert den Dateinamen für die XSLT-Daten.

Definiert durch:

[setXSLTFileName](#) in Schnittstelle [IXSLT](#).

Parameter:

`str`: eine absolute URL, die den Basisordner der XSLT-Daten angibt.

setXSLTFromText

```
public void setXSLTFromText(java.lang.String str)
```

Definiert den Textwert für die XSLT-Daten.

Definiert durch:

[setXSLTFromText](#) in Schnittstelle [IXSLT](#).

Parameter:

str: ein String, der die serialisierten XSLT-Daten enthält.

addExternalParameter

```
public void addExternalParameter(java.lang.String strName,  
                                 java.lang.String strVal)
```

Fügt den Namen und Wert eines externen Parameters hinzu.

Definiert durch:

[addExternalParameter](#) in Schnittstelle [IXSLT](#).

Parameter:

strName: ein String, der einen gültigen QName als Parameternamen, enthält.

strVal: ein String, der den Wert des Parameters enthält; dieser Wert wird als XPath-Ausdruck ausgewertet.

clearExternalParameterList

```
public void clearExternalParameterList()
```

Löscht die Liste der externen Parameter.

Definiert durch:

[clearExternalParameterList](#) in Schnittstelle [IXSLT](#).

setXSLTStackSize

```
public void addExternalParameter(long nVal)
```

Die Stack Size (Stapelgröße) gibt die maximale Tiefe für die ausgeführten Anweisungen an. Wenn die Stapelgröße bei der Transformation überschritten wird, wird ein Fehler ausgegeben.

Definiert durch:

[setXSLTStackSize](#) in Schnittstelle [IXSLT](#).

Parameter:

nVal: numerischer Wert für neue Stapelgröße. Muss größer als 100 sein. Der Anfangswert ist 1000.

enableJavaExtensions

```
public void enableJavaExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die Java-Erweiterungsfunktionen.

Wird definiert durch:

[enableJavaExtensions](#) in der Schnittstelle [IExecutable](#).

enableDotNetExtensions

```
public void enableDotNetExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die .NET-Erweiterungsfunktionen.

Wird definiert durch:

[enableJavaExtensions](#) in der Schnittstelle [IExecutable](#).

XSLT2

Öffentliche Klasse XSLT2
erweitert java.lang.Object
implementiert [IXSLT](#)

Implementierte Schnittstellen: [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#), [IXSLT](#)

Beschreibung

Klasse, die den XSLT 2.0-Prozessor enthält. Keine direkte Konstruktion/kein direkter Zugriff möglich. Erlaubt durch Aufruf der Funktion [getXSLT2Instance\(\)](#) auf eine Instanz von [IAltovaXMLFactory](#) Zugriff auf die [IXSLT](#) Schnittstelle. Beachten Sie: Der XSLT 2.0-Prozessor kann zur Verarbeitung eines XSLT 1.0 Stylesheet im Rückwärtskompatibilitätsmodus verwendet werden. Die Ausgabe kann sich allerdings von der eines XSLT 1.0-Prozessors, der dasselbe XSLT 1.0 Stylesheet verarbeitet, unterscheiden.

Konstruktoren

Es ist der folgende Konstruktor definiert.

XSLT2

geschützt **XSLT2**(lang nXSLT2Ptr)

Methoden

Die folgenden Methoden sind definiert.

releaseInstance

```
public void releaseInstance()
```

Gibt die Verbindung des Objekts zum COM Server frei.

Definiert durch:

[releaseInstance](#) in Schnittstelle [IReleasable](#).

execute

```
public boolean execute(java.lang.String sOutFile)
```

Führt die Abfrage aus und speichert das Ergebnis in einer Datei. Bei einem Fehler können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

Definiert durch:

[execute](#) in Schnittstelle [IExecutable](#).

Parameter:

sOutFile: eine absolute URL, die den Ordner der Ausgabedatei angibt.

Rückgabewert:

true bei erfolgreicher Ausführung, false bei Fehler

executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()
```

Führt die Anweisung durch und gibt das Ergebnis in Form eines UTF-16-Textstring zurück. Bei einem Fehler können Sie mit Hilfe der in [IAltovaXMLEngine](#) deklarierten Funktion [getLastErrorMessage\(\)](#) zusätzliche Informationen abrufen.

Definiert durch:

[executeAndGetResultAsString](#) in Schnittstelle [IExecutable](#).

Rückgabewert:

String, der das serialisierte Ergebnis enthält. Bei einem Fehler wird der leere String zurückgegeben.

setInputXMLFileName

```
public void setInputXMLFileName(java.lang.String str)
```

Definiert den Dateinamen für die XML-Eingabedaten. Beachten Sie, dass absolute URLs verwendet werden müssen.

Definiert durch:

[setInputXMLFileName](#) in Schnittstelle [IAltovaXMLEngine](#).

Parameter:

str: eine absolute URL, die den Basisordner der XML-Daten angibt.

setInputXMLFromText

```
public void setInputXMLFromText(java.lang.String str)
```

Definiert den Textwert für die XML-Eingabedaten. Beispiel: setInputXMLFromText(

```
"<doc> <a>text</a> </doc>" )
```

Definiert durch:

[setInputXMLFromText](#) in der Schnittstelle [IAltovaXMLEngine](#).

Parameter:

str: ein String, der XML-Daten enthält.

getLastErrorMessage

```
public java.lang.String getLastErrorMessage()
```

Ruft die letzte Fehlermeldung vom Prozessor ab.

Definiert durch:

[getLastErrorMessage](#) in Schnittstelle [IAltovaXMLEngine](#).

Rückgabewert:

ein String, der die letzte Fehlermeldung enthält.

setXSLTFileName

```
public void setXSLTFileName(java.lang.String str)
```

Definiert den Dateinamen für die XSLT-Daten.

Definiert durch:

[setXSLTFileName](#) in Schnittstelle [IXSLT](#).

Parameter:

str: eine absolute URL, die den Basisordner der XSLT-Daten angibt.

setXSLTFromText

```
public void setXSLTFromText(java.lang.String str)
```

Definiert den Textwert für die XSLT-Daten.

Definiert durch:

[setXSLTFromText](#) in Schnittstelle [IXSLT](#).

Parameter:

str: ein String, der die serialisierten XSLT-Daten enthält.

addExternalParameter

```
public void addExternalParameter(java.lang.String strName,  
                                java.lang.String strVal)
```

Fügt den Namen und Wert eines externen Parameters hinzu.

Definiert durch:

[addExternalParameter](#) in Schnittstelle [IXSLT](#).

Parameter:

strName: ein String, der einen gültigen QName als Parameternamen, enthält.

strVal: ein String, der den Wert des Parameters enthält; dieser Wert wird als

XPath-Ausdruck ausgewertet.

clearExternalParameterList

```
public void clearExternalParameterList()
```

Löscht die Liste der externen Parameter.

Definiert durch:

[clearExternalParameterList](#) in Schnittstelle [IXSLT](#).

```
setInitialTemplateName
```

```
public void setInitialTemplateName(java.lang.String str)
```

Definiert den anfangs verwendeten Vorlagennamen für die Transformation.

```
setInitialTemplateMode
```

```
public void setInitialTemplateMode(java.lang.String str)
```

Definiert den anfangs verwendeten Vorlagenmodus für die Transformation.

```
setXSLTStackSize
```

```
public void addExternalParameter(long nVal)
```

Die Stack Size (Stapelgröße) gibt die maximale Tiefe für die ausgeführten Anweisungen an.

Wenn die Stapelgröße bei der Transformation überschritten wird, wird ein Fehler ausgegeben.

Definiert durch:

[setXSLTStackSize](#) in Schnittstelle [IXSLT](#).

Parameter:

nVal: numerischer Wert für neue Stapelgröße. Muss größer als 100 sein. Der Anfangswert ist 1000.

```
enableJavaExtensions
```

```
public void enableJavaExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die Java-Erweiterungsfunktionen.

Wird definiert durch:

[enableJavaExtensions](#) in der Schnittstelle [IExecutable](#).

```
enableDotNetExtensions
```

```
public void enableDotNetExtensions(boolean bEnable)
```

Aktiviert/Deaktiviert die .NET-Erweiterungsfunktionen.

Wird definiert durch:

[enableJavaExtensions](#) in der Schnittstelle [IExecutable](#).

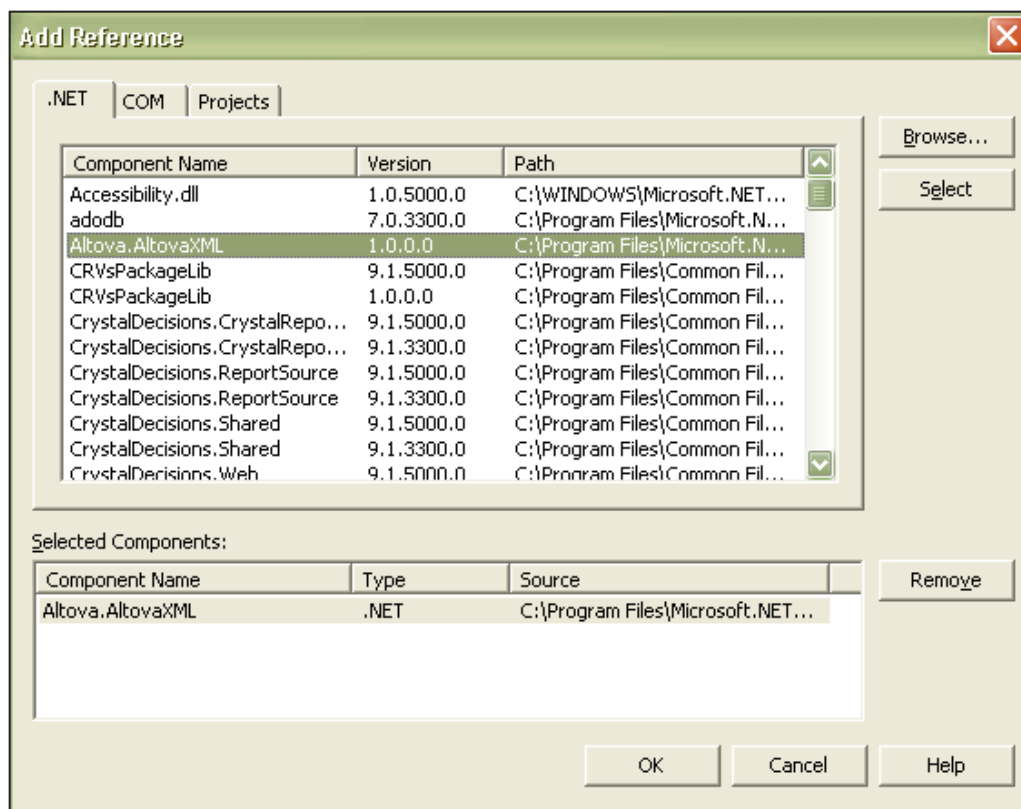
2.4 .NET-Schnittstelle

Die .NET-Schnittstelle ist als Hülle rund um die AltovaXML COM-Schnittstelle gebaut. Sie dient als primäre von Altova signierte Interop-Assembly und verwendet den Namespace `Altova.AltovaXML`. Um AltovaXML in Ihrem .NET-Projekt verwenden zu können, müssen Sie: (i) eine Referenz zur AltovaXML DLL (welche den Namen `Altova.AltovaXML.dll` trägt) in Ihrem Projekt hinzufügen und (ii) AltovaXML als COM Serverobjekt registriert haben. Sobald diese (unten beschriebenen) Anforderungen erfüllt sind, können Sie die AltovaXML Funktionalitäten in Ihrem Projekt nutzen.

Hinzufügen der AltovaXML DLL als Referenz zum Projekt

Das AltovaXML Paket enthält eine signierte DLL-Datei namens `Altova.AltovaXML.dll`, die bei der Installation von AltovaXML mit Hilfe des AltovaXML Installationsprogramms automatisch zum globalen Assembly Cache (und zur .NET Referenzbibliothek) hinzugefügt wird. (Normalerweise im Ordner `C:\WINDOWS\assembly`). Um diese DLL als Referenz in einem .NET-Projekt hinzuzufügen, gehen Sie folgendermaßen vor:

1. Klicken Sie bei geöffnetem .NET-Projekt auf **Project | Add Reference**. Daraufhin wird das Dialogfeld "Add Reference" (*Abbildung unten*) angezeigt und Sie sehen darin eine Liste der installierten .NET-Komponenten.



2. Wählen Sie in der Komponentenliste `Altova.AltovaXML` aus, doppelklicken Sie darauf oder klicken Sie auf die Schaltfläche "Select" und anschließend auf OK.

Registrieren von AltovaXML als COM-Serverobjekt

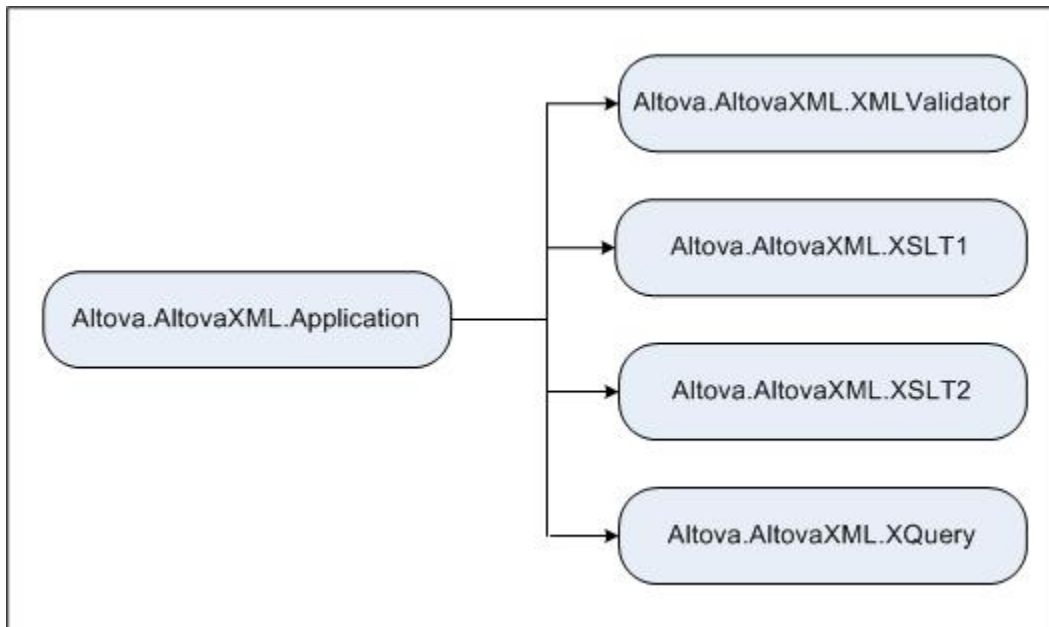
Die COM-Registrierung erfolgt automatisch durch das AltovaXML Installationsprogramm. Wenn Sie den Ordner der Datei `AltovaXML.COM.exe` nach der Installation ändern, sollten Sie AltovaXML als COM-Serverobjekt registrieren. Führen Sie dazu den Befehl `AltovaXML`

`_COM.exe /regserver` aus. (Beachten Sie, dass der richtige Pfad zur Datei `AltovaXML_COM.exe` eingegeben werden muss. Nähere Informationen siehe [Registrieren von AltovaXML als COM-Serverobjekt](#).)

Sobald die `Altova.AltovaXML.dll` der .NET-Schnittstelle zur Verfügung steht und `AltovaXML` als COM-Serverobjekt registriert wurde, stehen die `AltovaXML` Funktionalitäten in Ihrem .NET-Projekt zur Verfügung.

2.4.1 Allgemeine Verwendung und Beispiel

Die Klassen und Methoden, die Ihnen zur Verwendung zur Verfügung stehen, sind die im Abschnitt [COM-Schnittstelle](#) beschriebenen, befinden sich aber im Namespace `Altova`. `AltovaXML`. Sie sind in den folgenden Abschnitten aufgelistet. Ausgangspunkt ist das `Altova.AltovaXML.Application` Objekt. Wenn Sie dieses Objekt erstellen, wird eine Verbindung zu einem neuen AltovaXML COM Serverobjekt erstellt. Das Objektmodell wird im Diagramm unten gezeigt.



Beispiel

Die Verwendung der AltovaXML Klassen und Methoden im .NET Framework wird unten anhand eines Beispiels im C#-Code für ein Button Event gezeigt:

```

private void button1_Click(object sender, System.EventArgs e)
{
    Altova.AltovaXML.ApplicationClass appXML = new
Altova.AltovaXML.ApplicationClass();
    Altova.AltovaXML.XMLValidator XMLValidator =
appXML.XMLValidator;
    XMLValidator.InputXMLFromText = "<test>Is this data well-formed?
<a></test>";
    if ( XMLValidator.IsWellFormed() )
    {
        MessageBox.Show( this, "The input data is well-formed" ) ;
    }
    else
    {
        MessageBox.Show( this, "The input data is not well-formed" ) ;
    }
}
  
```

Der oben angeführte Code führt Folgendes aus:

1. Das `Altova.AltovaXML.ApplicationClass` Objekt wird erstellt. Dieses Objekt stellt eine Verbindung zu einem neuen AltovaXML COM Serverobjekt her.

2. Die XML-Validator-Funktionalität wird über `Altova.AltovaXML.XMLValidator` aufgerufen.
3. Die `InputXMLFromText` Eigenschaft von `Altova.AltovaXML.XMLValidator` liefert die XML-Eingabedaten.
4. Die `IsWellFormed` Methode von `Altova.AltovaXML.XMLValidator` überprüft, ob die vorliegenden XML-Daten wohlgeformt sind und gibt den Wert `TRUE` oder `FALSE` zurück.

Siehe dazu auch die Beispieldateien im Ordner `Examples` des Applikationsordners.

2.4.2 Altova.AltovaXML.XMLValidator

Beschreibung

Das `Altova.AltovaXML.XMLValidator` Objekt stellt Methoden bereit um Folgendes zu überprüfen:

- die Wohlgeformtheit eines XML-Dokuments.
- die Gültigkeit eines XML-Dokuments gemäß einer DTD oder eines im XML-Dokument referenzierten XML-Schemas.
- die Gültigkeit eines XML-Dokuments gemäß einer DTD oder eines XML-Schemas, die/das extern über den Code bereitgestellt wird.
- die Gültigkeit eines XBRL-Dokuments gemäß einer XBRL-Taxonomie (`.xsd`-Datei)

All diese Methoden geben die Booleschen Werte `TRUE` oder `FALSE` zurück.

Anmerkung: Wenn String-Eingabewerte als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus zur Auflösung des relativen Pfads definiert sein.

Methoden

Die folgenden Methoden stehen zur Verfügung:

`IsValidFormed`

`IsValidFormed` überprüft die Wohlgeformtheit des XML-Dokuments. Gibt den Wert `TRUE` zurück, wenn das XML-Dokument wohlgeformt ist und den Wert `FALSE`, wenn es nicht wohlgeformt ist.

`IsValid`

`IsValid` validiert das XML-Dokument gegen die DTD oder das XML-Schema, die/das im XML-Dokument referenziert ist. Gibt den Wert `TRUE` zurück, wenn das XML-Dokument gültig ist und bei Ungültigkeit den Wert `FALSE`. Um ein Dokument gegen eine DTD oder ein XML-Schema zu validieren, die/das nicht im XML-Dokument referenziert ist, verwenden Sie die Methode `IsValidWithExternalSchemaOrDTD`.

`IsValidWithExternalSchemaOrDTD`

`IsValidWithExternalSchemaOrDTD` validiert das XML-Dokument gegen eine DTD oder ein XML-Schema, die/das durch eine der folgenden Eigenschaften bereitgestellt wird:

`SchemaFileName`, `DTDFileName`, `SchemaFromText` oder `DTDFromText`. Wenn für mehr als eine dieser Eigenschaften Werte definiert sind, verwendet die

`IsValidWithExternalSchemaOrDTD` Methode die zuletzt definierte Eigenschaft. Gibt den Wert `TRUE` zurück, wenn das XML-Dokument gültig ist und bei Ungültigkeit den Wert `FALSE`.

Um das Dokument gegen eine DTD oder ein XML-Schema zu validieren, die/das im XML-Dokument referenziert ist, verwenden Sie die Methode `IsValid`.

Anmerkung: Die Validierung und Wohlgeformtheitsprüfung muss immer erfolgen, nachdem das XML-Dokument und/oder die DTD bzw. das XML-Schema-Dokument den entsprechenden Eigenschaften zugewiesen wurde.

Eigenschaften

Die folgenden Eigenschaften sind definiert:

`InputXMLFileName`

Ein Eingabestring, der als URL gelesen wird. Definiert den Pfad zu der zu validierenden XML-Datei.

SchemaFileName

Ein Eingabestring, der als URL gelesen wird. Definiert den Pfad zur XML-Schemadatei, gegen die das XML-Dokument validiert werden soll.

DTDFilename

Ein Eingabestring, der als URL gelesen wird. Definiert den Pfad zur DTD-Datei, gegen die das XML-Dokument validiert werden soll.

InputXMLFromText

Ein Eingabestring, der ein XML-Dokument erstellt.

SchemaFromText

Ein Eingabestring, der ein XML-Schema-Dokument erstellt.

DTDFromText

Ein Eingabestring, der ein DTD-Dokument erstellt.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

2.4.3 Altova.AltovaXML.XSLT1

Beschreibung

Das `Altova.AltovaXML.XSLT1` Objekt stellt Methoden und Eigenschaften zur Ausführung einer XSLT 1.0-Transformation mit Hilfe des Altova XSLT 1.0-Prozessors bereit. Die Ergebnisse können in einer Datei gespeichert werden oder als String zurückgegeben werden. Mit Hilfe dieses Objekts können außerdem XSLT-Parameter an das XSLT Stylesheet übergeben werden. Die URLs von XML und XSLT-Dateien können über die Eigenschaften des Objekts bereitgestellt werden. Alternativ dazu können das XML- und das XSLT-Dokument auch innerhalb des Codes als Textstrings erstellt werden.

Anmerkung: Wenn Eingabestrings als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus zur Auflösung des relativen Pfads definiert sein.

Methoden

Die folgenden Methoden stehen zur Verfügung:

Execute

`Execute` führt die XSLT 1.0-Transformation aus und speichert das Ergebnis als Ausgabedatei, deren Name und Pfad als Input-String für die `Execute` Methode bereitgestellt wird.

ExecuteAndGetResultAsString

`ExecuteAndGetResultAsString` führt eine XSLT 1.0 Transformation aus und gibt das Ergebnis als UTF 16-Textstring zurück.

AddExternalParameter

Nimmt einen Parameternamen und den Wert dieses Parameters als Input-Argumente. Die einzelnen externen Parameter und deren Werte müssen in einem separaten Aufruf der Methode definiert werden. Bei Angabe eines externen Parameters mit dem Namen eines existierenden (nicht gelöschten) Parameters wird ein Fehler ausgegeben. Da Parameterwerte XPath-Ausdrücke sind, müssen Parameterwerte, die Strings sind, in einfache Anführungszeichen eingeschlossen werden.

ClearExternalParameterList

Es sollte kein Argument bereitgestellt werden. Die Methode `ClearExternalParameterList` löscht die Liste der externen Parameter, die mit `AddExternalParameter` Methoden erstellt wurden.

Hinweis: Die Transformation muss immer erfolgen, nachdem das XML-Dokument und das XSLT-Dokument zugewiesen wurden.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

InputXMLFileName

Ein String-Input, der als URL gelesen wird, über die die zu transformierende XML-Datei gefunden wird.

XSLFileName

Ein String-Input, der als URL zum Auffinden der XSLT-Datei gelesen wird, die für die Transformation verwendet werden soll.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt.

XSLFromText

Ein String-Input, der ein XSLT-Dokument erstellt.

XSLStackSize

Die Stack-Größe ist die maximale Tiefe, bis zu der Anweisungen ausgeführt werden sollen. Die Stack-Größe kann über die Eigenschaft `XSLStackSize` geändert werden. Die zulässige Mindest-Stack-Größe ist 100. Die Standard-Stack-Größe ist 1000. Bei Überschreitung der Stack-Größe während einer Transformation wird ein Fehler ausgegeben.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

JavaExtensionsEnabled

Aktiviert die Java-Erweiterungen. Sie können festlegen, ob Java-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

DotNetExtensionsEnabled

Aktiviert die .NET-Erweiterungen. Sie können festlegen, ob .NET-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

2.4.4 Altova.AltovaXML.XSLT2

Beschreibung

Das `Altova.AltovaXML.XSLT2` Objekt stellt Methoden und Eigenschaften zum Ausführen einer XSLT 2.0-Transformation mittels des Altova XSLT 2.0-Prozessors bereit. Das Ergebnis kann in einer Datei oder als String zurückgegeben werden. Über das Objekt können außerdem XSLT-Parameter an das XSLT-Stylesheet übergeben werden. Die URLs von XML- und XSLT-Dateien können als Strings über Eigenschaften des Objekts bereitgestellt werden. Alternativ dazu können XML- und XSLT-Dokumente innerhalb Codes als Textstrings konstruiert werden.

Hinweis:

- Wo String-Inputs als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus definiert werden, der den relativen Pfad auflöst.
- Der XSLT 2.0-Prozessor kann zur Verarbeitung eines XSLT 1.0 Stylesheet im Rückwärtskompatibilitätsmodus verwendet werden. Die Ausgabe kann sich allerdings von der eines XSLT 1.0-Prozessors, der dasselbe XSLT 1.0 Stylesheet verarbeitet, unterscheiden.

Methoden

Es stehen die folgenden Methoden zur Verfügung:

Execute

`Execute` führt die XSLT 2.0-Transformation aus und speichert das Ergebnis als Ausgabedatei, deren Name und Pfad als Input-String für die `Execute` Methode bereitgestellt werden.

ExecuteAndGetResultAsString

`ExecuteAndGetResultAsString` führt eine XSLT 2.0 Transformation aus und gibt das Ergebnis als UTF 16-Textstring zurück.

AddExternalParameter

Nimmt einen Parameternamen und den Wert dieses Parameters als Input-Argumente. Die einzelnen externen Parameter und deren Werte müssen in einem separaten Aufruf der Methode definiert werden. Bei Angabe eines externen Parameters mit dem Namen eines existierenden (nicht gelöschten) Parameters wird ein Fehler ausgegeben. Da Parameterwerte XPath-Ausdrücke sind, müssen Parameterwerte, die Strings sind, in einfache Anführungszeichen eingeschlossen werden.

ClearExternalParameterList

Es sollte kein Argument bereitgestellt werden. Die Methode `ClearExternalParameterList` löscht die Liste der externen Parameter, die mit `AddExternalParameter` Methoden erstellt wurden.

InitialTemplateName

Definiert die anfangs verwendete named Template. Das Argument ist der Name der Vorlage, an der die Verarbeitung beginnen soll.

InitialTemplateMode

Definiert den anfangs verwendeten Modus für die Verarbeitung. Das Argument ist der Name des Anfangsmodus. Vorlagen mit diesem Moduswert werden verarbeitet.

Hinweis: Die Transformation muss immer erfolgen, nachdem das XML-Dokument und das XSLT-Dokument zugewiesen wurden.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

InputXMLFileName

Ein String-Input, der als URL gelesen wird, über die die zu transformierende XML-Datei gefunden wird.

XSLFileName

Ein String-Input, der als URL zum Auffinden der XSLT-Datei gelesen wird, die für die Transformation verwendet werden soll.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt.

XSLFromText

Ein String-Input, der ein XSLT-Dokument erstellt.

XSLStackSize

Die Stack-Größe ist die maximale Tiefe, bis zu der Anweisungen ausgeführt werden sollen. Die Stack-Größe kann über die Eigenschaft `XSLStackSize` geändert werden. Die zulässige Mindest-Stack-Größe ist 100. Die Standard-Stack-Größe ist 1000. Bei Überschreitung der Stack-Größe während einer Transformation wird ein Fehler ausgegeben.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

JavaExtensionsEnabled

Aktiviert die Java-Erweiterungen. Sie können festlegen, ob Java-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

DotNetExtensionsEnabled

Aktiviert die .NET-Erweiterungen. Sie können festlegen, ob .NET-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

2.4.5 Altova.AltovaXML.XQuery

Beschreibung

Das `Altova.AltovaXML.XQuery` Objekt stellt Methoden und Eigenschaften zum Ausführen einer XQuery 1.0-Transformation mittels des Altova XQuery 1.0-Prozessors bereit. Das Ergebnis kann in einer Datei gespeichert oder als String zurückgegeben werden. Über das Objekt können außerdem XQuery-Variablen an das XQuery-Dokument übergeben werden. Die URLs von XQuery- und XML-Dateien können als Strings über Eigenschaften des Objekts bereitgestellt werden. Alternativ dazu können das XML- und das XQuery-Dokument innerhalb des Codes als Textstrings konstruiert werden.

Hinweis: Wo String-Inputs als URLs interpretiert werden sollen, sollten absolute Pfade verwendet werden. Bei Verwendung eines relativen Pfads sollte im aufrufenden Modul ein Mechanismus definiert werden, der den relativen Pfad auflöst.

Methoden

Es stehen die folgenden Methoden zur Verfügung:

Execute

`Execute` führt die XQuery 1.0-Transformation aus und speichert das Ergebnis als Ausgabedatei, deren Name und Pfad als Input-String für die `Execute` Methode bereitgestellt werden.

ExecuteAndGetResultAsString

`ExecuteAndGetResultAsString` führt eine XQuery 1.0-Transformation aus und gibt das Ergebnis als UTF 16- Textstring zurück.

AddExternalVariable

Nimmt einen Variablennamen und den Wert dieser Variable als Input-Argumente. Die einzelnen externen Variablen und deren Werte müssen in einem separaten Aufruf der Methode definiert werden. Variablen müssen im XQuery-Dokument deklariert werden, optional mit einer Typdeklaration. Unabhängig von der Typdeklaration für die externe Variable im XQuery-Dokument sind für den Variablenwert, der für die Methode `AddExternalVariable` bereitgestellt wird, keine speziellen Trennzeichen wie z.B. Anführungszeichen erforderlich. Die lexikalische Form muss jedoch der des erwarteten Typs entsprechen (so muss z.B. eine Variable vom Typ `xs:date` einen Wert in der lexikalischen Form `2004-01-31` haben; ein Wert in der lexikalischen Form `2004/Jan/01` verursacht dagegen einen Fehler). Beachten Sie, dass dies auch bedeutet, dass Sie eine XQuery 1.0-Funktion (z.B. `current-date()`) nicht als den Wert einer externen Variable verwenden können (da die lexikalische Form der Funktion in der eigenen Schreibweise entweder nicht dem erforderlichen Datentyp entspricht - wenn der Datentyp in der Deklaration der externen Variable definiert ist - oder als String gelesen wird, wenn der Datentyp nicht definiert ist. Wenn eine externe Variable bereitgestellt wird, die den Namen einer existierenden (nicht gelöschten) Variable hat, so wird ein Fehler ausgegeben.

ClearExternalVariableList

Es sollte kein Argument bereitgestellt werden. Die Methode `ClearExternalVariableList` löscht die Liste der externen Variablen, die mit `AddExternalVariable` Methoden erstellt wurden.

Hinweis: Die Definition des optionalen XML-Dokuments muss immer vor der XQuery-Ausführung erfolgen.

Eigenschaften

Es sind die folgenden Eigenschaften definiert:

XQueryFileName

Ein String-Input, der als URL zum Auffinden der auszuführenden XQuery-Datei gelesen wird. Wenn sowohl die `XQueryFileName` Eigenschaft als auch die `QueryFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

InputXMLFileName

Ein String-Input, der als URL zum Auffinden der in die Query zu ladende XML-Datei gelesen wird. XQuery Navigationsausdrücke werden in Beziehung auf den Dokumentnode dieses XML-Dokuments ausgewertet. Wenn sowohl die `InputFileName` Eigenschaft als auch die `InputXMLFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

XQueryFromText

Ein String-Input, der ein XQuery-Dokument erstellt. Wenn sowohl die `XQueryFileName` Eigenschaft als auch die `XQueryFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

InputXMLFromText

Ein String-Input, der ein XML-Dokument erstellt. XQuery Navigationsausdrücke werden in Bezug auf den Dokument-Node dieses XML-Dokuments ausgewertet. Wenn sowohl die `InputFileName` Eigenschaft als auch die `InputXMLFromText` Eigenschaft definiert ist, so wird diejenige Eigenschaft verwendet, die in der Codesequenz als letzte definiert ist.

LastErrorMessage

Gibt die letzte Fehlermeldung zurück.

JavaExtensionsEnabled

Aktiviert die Java-Erweiterungen. Sie können festlegen, ob Java-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

DotNetExtensionsEnabled

Aktiviert die .NET-Erweiterungen. Sie können festlegen, ob .NET-Erweiterungen aktiviert werden sollen oder nicht, indem Sie den Wert `true` oder `false` (keine Berücksichtigung der Groß- und Kleinschreibung) als Boolesches Argument liefern.

Hinweis: Wenn ein XML-Dokument definiert ist und nicht für eine neue XQuery-Ausführung erforderlich ist, sollte dies durch Zuweisung eines leeren Strings gelöscht werden.

Es sind die folgenden Serialisierungsoptionen definiert:

OutputMethod

Die benötigte Ausgabemethode kann durch Bereitstellung des erforderlichen Werts als String-Argument definiert werden. Gültige Werte sind: `xml`, `xhtml`, `html` und `text`. Z.B.: `objAltovaXML.XQuery.OutputMethod = "xml"`. Wenn der Wert ungültig ist, wird er ignoriert. Die Standardausgabemethode ist `xml`.

OutputOmitXMLDeclaration

Sie können angeben, ob die XML-Deklaration in der Ausgabe inkludiert werden soll oder nicht. Geben Sie dazu `true` oder `false` (Groß- und Kleinschreibung ist irrelevant) als Boolesches Argument an. Z.B.: `objAltovaXML.XQuery.OutputOmitXMLDeclaration = "FALSE"`. Wenn der Wert ungültig ist, wird ein Fehler ausgegeben. Die Standardoption ist `TRUE`.

OutputIndent

Sie können festlegen, ob die Ausgabe Einrückungen enthalten soll oder nicht. Geben Sie dazu

`true` oder `false` (Groß- und Kleinschreibung ist irrelevant) als Boolesches Argument an. Z.B.: `objAltovaXML.XQuery.OutputIndent = "TRUE"`. Wenn der Wert ungültig ist, wird ein Fehler ausgegeben. Die Standardoption ist `False`.

OutputEncoding

Die erforderliche Ausgabecodierung kann durch Angabe des Codierungswerts als String-Argument definiert werden. Z.B.: `objAltovaXML.XQuery.OutputEncoding = "UTF-8"`. Wenn der Wert ungültig ist, wird er ignoriert. Die Standardausgabecodierung ist UTF-8.

Hinweis: Bei den Serialisierungsoptionen gibt es Unterschiede bei der Verwendung der Raw-Schnittstelle und der Dispatch-Schnittstelle. Wenn bei Verwendung der Raw-Schnittstelle kein Argument mit diesen Eigenschaften bereitgestellt wird, so wird der aktuelle Wert der Eigenschaft zurückgegeben. Die Eingabe müsste in etwa so aussehen: `put_OutputOption(VARIANT_BOOL bVal)` bzw. `VARIANT_BOOL bVal = get_OutputOption()`, um Werte zu setzen und abzurufen. Bei der Dispatch-Schnittstelle können Sie `b = myXQuery.OutputOption` verwenden, um Werte abzurufen und `myXQuery.OutputOption = b` um Werte zu definieren. Bei der Dispatch-Schnittstelle würde z.B. `Sheet1.Cells(10, 2) = objAltovaXML.XQuery.OutputEncoding` die aktuelle Ausgabecodierung abrufen.

Kapitel 3

Explizite Freigabe von AltovaXML COM-Server von C# un

3 Explizite Freigabe von AltovaXML COM-Server von C# und VB.NETaus

Wenn Referenzen auf das COM Server-Objekt nicht freigegeben werden, wenn das Objekt außerhalb des Geltungsbereichs zu liegen kommt, können die AltovaXML COM-Referenzen explizit vom C#-Code aus mittels der ReleaseComObject-Methoden freigegeben werden, wie unten gezeigt.

Beispiel:

```
private void button1_Click(object sender, EventArgs e)
{
    Altova.AltovaXML.ApplicationClass AltovaXML = new Altova.
AltovaXML.ApplicationClass();
    Altova.AltovaXML.IXSLT2 XSLT2 = AltovaXML.XSLT2;

    XSLT2.InputXMLFileName =
"C:\\Projects\\files\\XMLSpyExeFolder\\Examples\\OrgChart.xml";
    XSLT2.XSLFileName =
"C:\\Projects\\files\\XMLSpyExeFolder\\Examples\\OrgChart.xsl";
    XSLT2.Execute(
"C:\\Projects\\files\\XMLSpyExeFolder\\Examples\\OrgChart_out.html");

    // you must release ALL references to all components that
you received.
    System.Runtime.InteropServices.Marshal.ReleaseComObject(
XSLT2);
    XSLT2 = null;
    System.Runtime.InteropServices.Marshal.ReleaseComObject(
AltovaXML);
    AltovaXML = null;
}
```

- Am Ende der Methode wird der AltovaXML.exe **Server** beendet.
- Wenn Sie nicht **alle** ReleaseComObject Methoden aufrufen, werden die Exe-Server erst beim Beenden der C#-Applikation beendet.

Kapitel 4

OOXML- und ZIP-Dateien

4 OOXML- und ZIP-Dateien

Damit die erzeugten Dateien in Form einer ZIP-Datei oder Open Office XML (OOXML)-Dateien wie `.docx` ausgegeben werden, muss das ZIP-Protokoll im Dateipfad definiert werden.

Beispiel:

```
filename.zip| zip/filename.xxx  
filename.docx| zip/filename.xxx
```

In AltovaXML kann die Ausgabe in eine ZIP-Datei mit den folgenden Operationen definiert werden:

COM-Schnittstelle und .NET-Schnittstelle

Die Ausgabe wird mit Hilfe der `Execute` Methode generiert. Das Argument der Methode definiert den Namen und Pfad der Ausgabedatei. Für ZIP-Dateien muss das ZIP-Protokoll (siehe folgendes Beispiel) verwendet werden:

```
xslt2.Execute(c:\Mydocs\orgchart.zip| zip\main.xml)  
xslt2.Execute(c:\Mydocs\orgchart.docx| zip\main.out)  
xslt2.Execute(c:\Mydocs\orgchart.docx| zip\)
```

Befehlszeile

Stellen Sie bei Verwendung der Befehlszeile sicher, dass die Ausgabe-URI innerhalb von Anführungszeichen steht. Andernfalls würde das Pipe-Zeichen (`|`) vom Befehlszeilensystem interpretiert werden. Ein Beispiel:

```
AltovaXML -in input.xml -xslt2 transform.xslt -out "c:\results.zipart.zip|  
zip\result.xml"
```

Das `xsl:result-document` Element

Im Falle des `xsl:result-document` Elements von XSLT 2.0 muss für die Ausgabe-URI das ZIP-Protokoll verwendet werden. Bei OOXML-Dokumenten muss das ZIP-Protokoll für die Ausgabe-URI jedes `xsl:result-document` Elements verwendet werden, das beim Erstellen von Dateien für das OOXML-Dokument verwendet wird. Wenn die `xsl:result-document` Elemente relative Ausgabe-URIs definieren, definieren Sie das ZIP-Protokoll für das Hauptergebnis, dessen URI anschließend als Basis-URI zum Auflösen der relativen Ausgabe-URIs dient.

Kapitel 5

Prozessorinformationen

5 Prozessorinformationen

Dieser Abschnitt enthält Informationen über implementierungsspezifische Features des Altova XML Validators, des Altova XSLT 1.0-Prozessors, des Altova XSLT 2.0-Prozessors und des Altova XQuery-Prozessors.

5.1 Altova XML Validator

Der Altova XML Validator implementiert die folgenden Regeln und entspricht diesen:

- [XML 1.0 \(Fourth Edition\)](#)
- [XML Namespaces \(1.0\)](#)
- [XML Schemas \(Structures\)](#)
- [XML Schema \(Datatypes\)](#)

5.2 XSLT 1.0-Prozessor: Implementierungsinformationen

Der Altova XSLT 1.0-Prozessor ist in die Altova XML-Produkte XMLSpy, StyleVision, Authentic und MapForce integriert. Außerdem steht er im kostenlosen AltovaXML-Paket zur Verfügung. Im Altova XSLT 1.0-Prozessor sind die W3C-Spezifikationen implementiert und entsprechen der [XSLT 1.0 Recommendation vom 16. November 1999](#) und der [XPath 1.0 Recommendation vom 16. November 1999](#). Im Folgenden sind Einschränkungen und implementierungsspezifisches Verhalten aufgelistet.

Einschränkungen

- Die Elemente `xsl: preserve-space` und `xsl: strip-space` werden nicht unterstützt.
- Wenn das Attribut `method` von `xsl: output` auf HTML gesetzt ist, oder wenn standardmäßig HTML output ausgewählt ist, werden Sonderzeichen in der XML- oder XSLT-Datei direkt als Sonderzeichen in die HTML-Dokument eingefügt und nicht als HTML-Zeichenreferenzen in der Ausgabe. So wird z.B. das Zeichen ` ` (die Dezimalzeichenreferenz für ein geschütztes Leerzeichen) nicht als ` ` in den HTML-Code eingefügt sondern direkt als geschütztes Leerzeichen.

Behandlung der Implementierung von Nur-Whitespace Nodes in XML-Quelldokumenten

Aus den XML-Daten (und somit dem XML-Infoset), die an den Altova XSLT 1.0-Prozessor übergeben werden, werden boundary-whitespace-only Text Nodes entfernt. (Ein boundary-whitespace-only Text Node ist ein Textnode, der nur Whitespaces enthält und der zwischen zwei Elementen innerhalb eines mixed-content-Elements vorkommt). Dies kann sich auf den Wert auswirken, der von den Funktionen `fn: position()`, `fn: last()` und `fn: count()` zurückgegeben wird.

In jeder Node-Auswahl, bei der auch Text Nodes ausgewählt werden, sind normalerweise auch Boundary-whitespace-only Text Nodes enthalten. Da jedoch beim XML Infoset, das von den Altova-Prozessoren verwendet wird, boundary-whitespace-only Text Nodes entfernt werden, sind diese Nodes im XML Infoset nicht vorhanden. Folglich unterscheidet sich die Größe der Auswahl und die Nummerierung der Nodes von einer, in der diese Text Nodes enthalten sind. Aus diesem Grund können sich die Ergebnisse der Funktionen `fn: position()`, `fn: last()` und `fn: count()` von denen anderer Prozessoren unterscheiden.

Am häufigsten tritt die Situation, dass boundary-whitespace-only Text Nodes als gleichrangige Elemente anderer Elemente überprüft werden, dann auf, wenn `xsl: apply-templates` verwendet wird, um Templates anzuwenden. Wenn die Funktionen `fn: position()`, `fn: last()` und `fn: count()` in Patterns mit einer Namensüberprüfung verwendet werden (z.B. `para[3]`, kurz für `para[position()=3]`), sind boundary-whitespace-only Nodes irrelevant, da nur die benannten Elemente (`para` im obigen Beispiel) ausgewählt werden. (Beachten Sie jedoch, dass boundary-whitespace-only Nodes in Patterns, in denen ein Platzhalterzeichen wie z.B. `*[10]` verwendet wird, sehr wohl relevant sind).

Hinweis: Wenn der boundary-whitespace-only Text Node in der Ausgabe benötigt wird, fügen Sie das erforderliche Whitespace-Zeichen in eines der benachbarten Child-Elemente ein. So erzeugt z.B. das XML-Fragment:

```
<para>This is <b>bold</b> <i>italic</i>. </para>
```

bei Verarbeitung mit dem XSLT Template

```
<xsl:template match="para">
  <xsl:apply-templates/>
</xsl:template>
```

folgendes Resultat:

```
This is bolditalic.
```

Um in der Ausgabe ein Leerzeichen zwischen `bold` und `italic` zu erhalten, fügen Sie entweder in das Element `` oder `<i>` in der XML-Quelle ein Leerzeichen ein. z.B.:

```
<para>This is <b>bold</b> <i> italic</i>. </para> or  
<para>This is <b>bold<#x20;</b> <i>italic</i>. </para> or  
<para>This is <b>bold</b><i>&#x20; italic</i>. </para>
```

Wenn eines der oben genannten `para`-Elemente mit demselben XSLT Template verarbeitet wird, erhalten Sie folgendes Ergebnis:

```
This is bold italic.
```

5.3 XSLT 2.0-Prozessor: Implementierungsinformationen

Der Altova XSLT 2.0-Prozessor ist in die Altova XML-Produkte XMLSpy, StyleVision, Authentic und MapForce integriert. Außerdem steht er im kostenlosen AltovaXML-Paket zur Verfügung. In diesem Abschnitt werden die implementierungsspezifischen Aspekte des Prozessors beschrieben. Im ersten Teil des Abschnitts finden Sie allgemeine Informationen über den Prozessor. Im Anschluss daran finden Sie eine Liste der implementierungsspezifischen Aspekte der XSLT 2.0-Funktionen.

Informationen über das implementierungsspezifische Verhalten von XPath 2.0-Funktionen finden Sie im Abschnitt [XPath 2.0- und XQuery 1.0-Funktionen](#).

5.3.1 Allgemeine Informationen

Der Altova XSLT 2.0-Prozessor entspricht der [XSLT 2.0 Recommendation](#) vom 23 January 2007. Beachten Sie bitte die folgenden allgemeinen Informationen über den Prozessor.

Rückwärtskompatibilität

Der Altova XSLT 2.0-Prozessor ist rückwärtskompatibel. Die Rückwärtskompatibilität des XSLT 2.0-Prozessors kommt nur dann zum Einsatz, wenn Sie den XSLT 2.0-Prozessor von Altova XML zur Verarbeitung eines XSLT 1.0 Stylesheets verwenden. Beachten Sie, dass sich das Ergebnis des XSLT 1.0-Prozessors und des rückwärtskompatiblen XSLT 2.0-Prozessors unter Umständen unterscheiden kann.

Bei allen anderen Altova-Produkten wird Rückwärtskompatibilität nie benötigt, da automatisch der entsprechende Prozessor für die Transformation ausgewählt wird. Angenommen, Sie legen fest, dass ein bestimmtes XML-Dokument in XMLSpy mit einem XSLT 1.0 Stylesheet verarbeitet werden soll. Bei Aufruf des Transformationsbefehls wählt XMLSpy zur Transformation automatisch den XSLT 1.0-Prozessor aus.

Anmerkung:

Diese Auswahl basiert auf dem Wert im Attribut `version` des Stylesheet-Elements `stylesheet` oder `transform`.

Namespaces

In Ihrem XSLT 2.0 Stylesheet sollten die folgenden Namespaces deklariert sein, damit Sie die in XSLT 2.0 verfügbaren Typ-Konstrukturen und Funktionen verwenden können. Normalerweise werden die unten aufgelisteten Präfixe verwendet; bei Bedarf können Sie auch andere Präfixe verwenden.

Namespace Name	Präfix	Namespace URI
XML Schema-Typen	xs:	http://www.w3.org/2001/XMLSchema
XPath 2.0-Funktionen	fn:	http://www.w3.org/2005/xpath-functions

Normalerweise werden diese Namespaces im Element `xsl:stylesheet` oder `xsl:transform` deklariert, wie unten gezeigt:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  ...
/>
```

Beachten Sie bitte die folgenden Punkte:

- Der Altova XSLT 2.0-Prozessor verwendet als Default Functions Namespace den Namespace für XPath 2.0- und XQuery 1.0-Funktionen (siehe Tabelle oben). Sie können daher XPath 2.0- und XSLT 2.0-Funktionen in Ihrem Stylesheet ohne Präfix verwenden. Wenn Sie den Namespace für XPath 2.0-Funktionen in Ihrem Stylesheet mit einem Präfix deklarieren, können Sie zusätzlich dazu das in der Deklaration zugewiesene Präfix verwenden.
- Bei Verwendung von Typ-Konstrukturen und Typen aus dem XML Schema-Namespaces, muss bei Aufruf des Typ-Konstruktors (z.B. `xs:date`) das in der

jeweiligen Namespace-Deklaration verwendeten Präfix verwendet werden.

- In den Candidate Recommendations vom 23 January 2007 wurden die Datentypen `untypedAtomic` und `duration` (`dayTimeDuration` und `yearMonthDuration`), die sich zuvor im XPath Datatypes Namespace befanden (normalerweise mit dem Präfix `xdtd`) in den XML-Schema-Namespaces verschoben.
- Einige XPath 2.0-Funktionen haben denselben Namen wie XML Schema-Datentypen. So gibt es z.B. für die XPath-Funktionen `fn:string` und `fn:boolean` XML-Schema-Datentypen mit demselben lokalen Namen: `xs:string` und `xs:boolean`. Wenn Sie daher den XPath-Ausdruck `string('Hello')` verwenden, wird der Ausdruck als `fn:string('Hello')` ausgewertet und nicht als `xs:string('Hello')`.

Schemafähigkeit

Der Altova XSLT 2.0-Prozessor ist schemafähig.

Whitespaces im XML-Dokument

Standardmäßig entfernt der Altova XSLT 2.0-Prozessor alle boundary whitespaces aus boundary-whitespace-only Nodes im XML-Quelldokument. Dies wirkt sich auf die Werte aus, die die Funktionen `fn:position()`, `fn:last()`, `fn:count()` und `fn:deep-equal()` zurückgeben. Nähere Informationen dazu finden Sie in den Abschnitten über XPath 2.0- und XQuery 1.0-Funktionen unter [Nur-Whitespace Nodes im XML-Quelldokument](#).

Hinweis: Wenn in der Ausgabe ein boundary-whitespace-only-Text Node erforderlich ist, fügen Sie das gewünschte Whitespace-Zeichen in eines der beiden benachbarten Child-Elemente ein. So erzeugt z.B. das XML-Fragment:

```
<para>This is <b>bold</b> <i>italic</i>.</para>
```

bei Verarbeitung mit dem XSLT Template

```
<xsl:template match="para">
  <xsl:apply-templates/>
</xsl:template>
```

folgendes Resultat:

```
This is bolditalic.
```

Um in der Ausgabe ein Leerzeichen zwischen `bold` und `italic` zu erhalten, fügen Sie entweder in das Element `` oder `<i>` in der XML-Quelle ein Leerzeichen ein. z.B.:

```
<para>This is <b>bold</b> <i> italic</i>.</para> oder
<para>This is <b>bold<#x20;</b> <i>italic</i>.</para> oder
<para>This is <b>bold</b><i>&#x20;italic</i>.</para>
```

Wenn nun ein solches XML-Fragment mit demselben XSLT Template verarbeitet wird, erhalten Sie folgendes Ergebnis:

```
This is bold italic.
```

XSLT 2.0-Elemente und -Funktionen

Einschränkungen und implementierungsspezifisches Verhalten von XSLT 2.0-Elementen und -Funktionen werden im Abschnitt [XSLT 2.0-Elemente und -Funktionen](#) aufgelistet.

XPath 2.0-Funktionen

Implementierungsspezifisches Verhalten von XPath 2.0-Funktionen wird im Abschnitt [XPath 2.0- und XQuery 1.0-Funktionen](#) aufgelistet.

5.3.2 XSLT 2.0-Elemente und -Funktionen

Einschränkungen

Die Elemente `xsl:preserve-space` und `xsl:strip-space` werden nicht unterstützt.

Implementierungsspezifisches Verhalten

Im Folgenden finden Sie eine Beschreibung, wie der Altova XSLT 2.0-Prozessor implementierungsspezifische Aspekte des Verhaltens bestimmter XSLT 2.0-Funktionen behandelt.

`xsl:result-document`

Zusätzlich werden die folgenden Kodierungen unterstützt: `base16tobinary` und `base64tobinary`.

`function-available`

Die Funktion überprüft, ob in-scope-Funktionen (XSLT 2.0, XPath 2.0 und Erweiterungsfunktionen) verfügbar sind.

`unparsed-text`

Das Attribut `href` akzeptiert (i) relative Pfade für Dateien im Basis-URI-Ordner und (ii) absolute Pfade mit oder ohne das `file://`-Protokoll. Zusätzlich werden die folgenden Kodierungen unterstützt: `binarytobase16` und `binarytobase64`.

5.4 XQuery 1.0-Prozessor: Implementierungsinformationen

Der Altova XQuery 1.0-Prozessor ist in die Altova XML-Produkte XMLSpy und MapForce integriert. Er steht auch in Form des kostenlosen Pakets AltovaXML zur Verfügung. Dieser Abschnitt enthält Informationen über implementierungsspezifische Aspekte seines Verhaltens.

Standardkonformität

Der Altova XQuery 1.0-Prozessor entspricht den [XQuery 1.0 CR](#) vom 23 January 2007. Der Query-Standard stellt frei, wie viele Funktionen zu implementieren sind. Im Folgenden finden Sie eine Liste, wie der Altova XQuery 1.0-Prozessor diese Funktionen implementiert.

Schemafähigkeit

Der Altova XQuery 1.0-Prozessor ist **schemafähig**.

Codierung

Die UTF-8 und die UTF-16 Zeichen-Kodierungen werden unterstützt.

Namespaces

Die folgenden Namespace-URIs und die damit verknüpften Bindings sind vordefiniert.

Namespace-Name	Präfix	Namespace URI
XML Schema-Typen	xs:	http://www.w3.org/2001/XMLSchema
Schema-Instanz	xsi:	http://www.w3.org/2001/XMLSchema-instance
Built-in Funktionen	fn:	http://www.w3.org/2005/xpath-functions
Lokale Funktionen	local:	http://www.w3.org/2005/xquery-local-functions

Beachten Sie bitte die folgenden Punkte:

- Der Altova XQuery 1.0-Prozessor ist so konfiguriert, dass die oben aufgelisteten Präfixe an die entsprechenden Namespaces gebunden sind.
- Da der oben angeführte Namespace für Built-in Funktionen der Default Functions Namespace in XQuery ist, muss beim Aufruf von built-in-Funktionen das Präfix `fn:` nicht verwendet werden (`string("Hello")` ruft z.B. die Funktion `fn:string` auf). Das Präfix `fn:` kann jedoch verwendet werden, um eine built-in-Funktion aufzurufen, ohne die Namespace im Abfrage-Prolog deklarieren zu müssen (z.B.: `fn:string("Hello")`).
- Sie können den Default Functions Namespace durch Deklaration des `default function namespace`-Ausdrucks im Abfrageprolog ändern.
- Bei Verwendung von Typen aus dem XML Schema- und dem XPath-Datentypen-Namespace kann das Präfix `xs:` verwendet werden, ohne dass Sie den Namespace explizit deklarieren müssen und dieses Präfix im Abfrageprolog daran binden müssen. (Beispiele: `xs:date` und `xs:yearMonthDuration`.) Wenn Sie ein anderes Präfix verwenden möchten, muss dieses im Abfrageprolog für die Namespaces explizit deklariert werden. (Beispiel `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Beachten Sie, dass die Datentypen `untypedAtomic`, `dayTimeDuration` und `yearMonthDuration` mit den Candidate Recommendations vom 23 January 2007 aus dem XPath Datentypen-Namespace in den XML-Schema Namespace verschoben

wurden, d.h. `xs: yearMonthDuration`.

Wenn Namespaces für Funktionen, Typ-Konstrukturen, Node Tests usw. falsch zugewiesen wurden, wird ein Fehler ausgegeben. Beachten Sie jedoch, dass einige Funktionen denselben Namen wie Schema-Datentypen haben, z.B. `fn: string` und `fn: boolean`. (Sowohl `xs: string` als auch `xs: boolean` ist definiert.) Das Namespace-Präfix legt fest, ob die Funktion oder der Typ-Konstruktor verwendet wird.

XML-Quelldokument und Validierung

XML-Dokumente, die bei der Ausführung eines XQuery-Dokuments mit dem Altova XQuery 1.0-Prozessor verwendet werden, müssen wohlgeformt sein. Sie müssen jedoch nicht gemäß einem XML-Schema gültig sein. Wenn die Datei nicht gültig ist, wird die ungültige Datei ohne Schemainformationen geladen. Wenn die XML-Datei mit einem externen Schema verknüpft ist und gemäß diesem Schema gültig ist, werden für die XML-Daten nachträglich Validierungsinformationen generiert und für die Überprüfung der Abfrage verwendet.

Statische und dynamische Typ-Überprüfung

In der statischen Analysephase werden Aspekte der Abfrage überprüft wie z.B. die Syntax, ob externe Referenzen (z.B. für Module) vorhanden sind, ob aufgerufene Funktionen und Variablen definiert sind, usw. In der statischen Analysephase wird keine Typ-Überprüfung durchgeführt. Wenn in dieser Phase ein Fehler gefunden wird, wird eine Meldung ausgegeben und die Ausführung wird gestoppt.

Die dynamische Typ-Überprüfung wird in Laufzeit durchgeführt, während die Abfrage ausgeführt wird. Wenn ein Typ mit den Anforderungen einer Operation nicht kompatibel ist, wird ein Fehler ausgegeben. So gibt z.B. der Ausdruck `xs: string("1") + 1` einen Fehler zurück, weil die Operation "Addition" nicht an einem Operanden vom Typ `xs: string` ausgeführt werden kann.

Bibliotheksmodule

Bibliotheksmodule dienen zum Speichern von Funktionen und Variablen, damit diese wiederverwendet werden können. Der Altova XQuery 1.0-Prozessor unterstützt Module, die in einer **einzigen externen XQuery-Datei** gespeichert sind. Eine solche Moduldatei muss im Prolog eine `module`-Deklaration enthalten, in der ein Target Namespace zugewiesen wird. Hier ein Beispielmodul:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

Alle im Modul deklarierten Funktionen und Variablen gehören zu dem mit dem Modul verknüpften Namespace. Das Modul wird durch `Import` in eine XQuery-Datei mittels der `import module`-Anweisung im Abfrageprolog verwendet. Die `import module`-Anweisung importiert nur Funktionen und Variablen, die direkt in der Bibliotheksmodul-Datei deklariert sind:

```
import module namespace modlib = "urn:module-library" at
    "modulefilename.xq";
if ($modlib:company = "Altova")
then modlib:webaddress()
else error("No match found.")
```

Externe Funktionen

Externe Funktionen, d.h. diejenigen Funktionen, die das Schlüsselwort `external` verwenden, werden nicht unterstützt:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations

Die Standard Collation ist die Unicode Codepoint Collation. Derzeit werden keine anderen Collations unterstützt. Vergleiche, wie u.a. die Funktion `fn:max` basieren auf dieser Collation.

Zeichennormalisierung

Es wird keine Zeichennormalisierungsform unterstützt.

Präzision von numerischen Typen

- Der Datentyp `xs:integer` hat eine beliebige Präzision, dh. er kann beliebig viele Stellen haben.
- Der Datentyp `xs:decimal` kann nach dem Dezimalpunkt maximal 20 Stellen haben.
- Die Datentypen `xs:float` und `xs:double` sind auf 15 Stellen beschränkt.

Unterstützung für XQuery-Anweisungen

Die `pragma` Anweisung wird nicht unterstützt. Gegebenenfalls wird sie ignoriert und der Fallback-Ausdruck wird evaluiert.

Unterstützung für XQuery-Funktionen

Informationen zu implementierungsspezifischem Verhalten von XQuery 1.0-Funktionen finden Sie im Abschnitt [XPath 2.0- und XQuery 1.0-Funktionen](#).

5.5 XPath 2.0- und XQuery 1.0-Funktionen

Die XPath 2.0- und XQuery 1.0-Funktionen werden von folgenden Prozessoren überprüft:

- dem **Altova XPath 2.0-Prozessor**, der (i) eine Komponente des Altova XSLT 2.0-Prozessors ist und (ii) im XPath Evaluator von Altova XMLSpy verwendet wird, um XPath-Ausdrücke hinsichtlich des in XMLSpy aktiven XML-Dokuments auszuwerten.
- dem **Altova XQuery 1.0-Prozessor**.

In diesem Abschnitt wird beschrieben, wie XPath 1.0- und XQuery 1.0-Funktionen vom Altova XPath 2.0-Prozessor und dem Altova XQuery 1.0-Prozessor behandelt werden. Es sind nur die Funktionen aufgelistet, die implementierungsspezifisch behandelt werden oder bei denen sich eine einzelne Funktion in einer der drei Umgebungen, in denen sie verwendet wird (also in XSLT 2.0, in XQuery 1.0 und im XPath Evaluator von XMLSpy), anders verhält. Beachten Sie, dass in diesem Abschnitt nicht beschrieben wird, wie diese Funktionen verwendet werden. Nähere Informationen über die Verwendung dieser Funktionen finden Sie im [XQuery 1.0 and XPath 2.0 Functions and Operators PR des W3C](#) vom 23 January 2007.

5.5.1 Allgemeine Informationen

Standardkonformität

- Der Altova XPath 2.0-Prozessor implementiert die [XPath 2.0 Recommendation](#) vom 23 January 2007. Der Altova XQuery 1.0-Prozessor implementiert die [XQuery 1.0 Recommendation](#) vom 23 January 2007. Die Unterstützung der XPath 2.0- und XQuery 1.0-Funktionen in diesen beiden Prozessoren entspricht den Vorgaben in den [XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation](#) vom 23 January 2007.
- Der Altova XPath 2.0-Prozessor entspricht den Vorgaben für [XML 1.0 \(Fourth Edition\)](#) und [XML Namespaces \(1.0\)](#).

Default Functions Namespace

Der Default Functions Namespace wurde dem Standard entsprechend eingestellt. Funktionen können daher ohne Präfix aufgerufen werden.

Boundary-whitespace-only-Nodes in XML-Quelldokumenten

Aus den XML-Daten (und somit dem XML-Infoset), die an den Altova XPath 2.0- und den Altova XQuery 1.0-Prozessor übergeben werden, werden boundary-whitespace-only Text Nodes entfernt. (Ein boundary-whitespace-only Text Node ist ein Child-Textnode, der nur Whitespaces enthält und der zwischen zwei Elementen innerhalb eines mixed-content-Elements vorkommt). Dies kann sich auf den Wert auswirken, der von den Funktionen `fn:position()`, `fn:last()`, `fn:count()` und `fn:deep-equal()` zurückgegeben wird.

In jeder Node-Auswahl, bei der auch Text Nodes ausgewählt werden, sind normalerweise auch Boundary-whitespace-only Text Nodes enthalten. Da jedoch beim XML Infoset, das von den Altova-Prozessoren verwendet wird, boundary-whitespace-only Text Nodes entfernt werden, sind diese Nodes im XML Infoset nicht vorhanden. Folglich unterscheidet sich die Größe der Auswahl und die Nummerierung der Nodes von einer, in der diese Text Nodes enthalten sind. Aus diesem Grund können sich die Ergebnisse der Funktionen `fn:position()`, `fn:last()`, `fn:count()` und `fn:deep-equal()` von denen anderer Prozessoren unterscheiden.

Am häufigsten tritt die Situation, dass boundary-whitespace-only Text Nodes als gleichrangige Elemente anderer Elemente überprüft werden, dann auf, wenn `xsl:apply-templates` verwendet wird, um Templates anzuwenden. Wenn die Funktionen `fn:position()`, `fn:last()` und `fn:count()` in Patterns mit einer Namensüberprüfung verwendet werden (z.B. `para[3]`, kurz für `para[position()=3]`), sind boundary-whitespace-only Nodes irrelevant, da nur die benannten Elemente (`para` im obigen Beispiel) ausgewählt werden. (Beachten Sie jedoch, dass boundary-whitespace-only Nodes in Patterns, in denen ein Platzhalterzeichen wie z.B. `*[10]` verwendet wird, sehr wohl relevant sind).

Numerische Notation

Wenn bei der Ausgabe ein `xs:double` in einen String konvertiert wird, wird die logarithmische Darstellung (z.B. `1.0E12`) verwendet, wenn der absolute Wert kleiner als 0,000001 oder größer als 1.000.000 ist. Andernfalls wird der Wert als Dezimalwert oder Integer angegeben.

Präzision von `xs:decimal`

Die Präzision bezieht sich auf die Anzahl der Stellen in einer Zahl. Laut Spezifikation sind mindestens 18 Stellen erforderlich. Bei Divisionen, bei denen ein Ergebnis vom Typ `xs:decimal` erzeugt wird, beträgt die Präzision 19 Kommastellen ohne Runden.

Implizite Zeitzone

Beim Vergleich zweier `date`, `time`, oder `dateTime` Werte muss die Zeitzone der verglichenen Werte bekannt sein. Wenn die Zeitzone in einem solchen Wert nicht explizit angegeben ist, wird die implizite Zeitzone verwendet. Als implizite Zeitzone wird die der Systemuhr verwendet. Der Wert kann mit Hilfe der Funktion `fn:implicit-timezone()` überprüft werden.

Collations

Es wird nur die Unicode Codepoint Collation unterstützt. Es können keine anderen Collations verwendet werden. Der Vergleich von Strings, u.a. auch für die Funktionen `fn:max` und `fn:min` basiert auf dieser Collation.

Namespace-Achse

Die Namespace-Achse wird in XPath 2.0 nicht mehr verwendet, wird aber weiterhin unterstützt. Um Namespace-Informationen mit XPath 2.0-Mechanismen aufzurufen, verwenden Sie die Funktionen `fn:in-scope-prefixes()`, `fn:namespace-uri()` und `fn:namespace-uri-for-prefix()`.

Static Type Extensions

Die optionale Funktion zum Überprüfen von statischen Typen wird nicht unterstützt.

5.5.2 Unterstützung von Funktionen

In der nachfolgenden Tabelle wird das implementierungsspezifische Verhalten bestimmter Funktionen (in alphabetischer Reihenfolge) aufgelistet. Beachten Sie bitte die folgenden allgemeinen Punkte:

- Wenn bei einer Funktion eine Sequenz von einem Datenelement als Argument erwartet wird und eine Sequenz von mehr als einem Datenelement gesendet wird, wird im allgemeinen ein Fehler zurückgegeben.
- Alle String-Vergleiche werden unter Verwendung der Unicode Codepoint Collation ausgeführt.
- Ergebnisse, bei denen es sich um QNames handelt, werden in der Form `[prefix:] localname` serialisiert.

Funktionsname	Anmerkungen
base-uri	<ul style="list-style-type: none">• Wenn im XML-Quelldokument externe Entities verwendet werden und ein Node in der externen Entity als Input-Node-Argument der Funktion <code>base-uri()</code> definiert ist, wird trotzdem die Basis-URI des inkludierenden XML-Dokuments verwendet und nicht die Basis-URI der externen Entity.• Die Basis-URI eines Node im XML-Dokument kann mit Hilfe des Attributs <code>xml:base</code> geändert werden.

<p>collection</p>	<ul style="list-style-type: none"> • Das Argument ist eine relative URI, die gegen die aktuelle Basis-URI aufgelöst wird. • Wenn mit der aufgelösten URI eine XML-Datei identifiziert wird, so wird diese XML-Datei als Katalog behandelt, der eine Sammlung von Dateien referenziert. Diese Datei muss die folgende Form haben: <pre><collection> <doc href="uri-1" /> <doc href="uri-2" /> <doc href="uri-3" /> </collection></pre> Die von den href Attributen referenzierten Dateien werden geladen und ihre Dokument-Nodes werden als Sequenz zurückgegeben. • Wenn mit der aufgelösten URI keine XML-Datei mit der oben beschriebenen Katalogstruktur identifiziert wird, dann wird der Argument-String (in dem die Platzhalterzeichen ? und * zulässig sind) als Suchstring verwendet. XML-Dateien, deren Namen dem Suchausdruck entsprechen, werden geladen und ihre Dokument-Nodes werden als Sequenz zurückgegeben. Siehe Beispiel unten. • XSLT-Beispiel: Der Ausdruck <code>collection("c:\MyDocs*.xml")//Title</code> gibt eine Sequenz aller DocTitle Elemente in den .xml Dateien im Ordner MyDocs zurück. • XQuery-Beispiel: Der Ausdruck <code>{for \$i in collection(c:\MyDocs*.xml) return element doc{base-uri(\$i)}}</code> gibt die Basis-URIs aller .xml Dateien im Ordner MyDocs zurück, wobei jede URI sich innerhalb eines doc Elemente befindet. • Die Standard-Collection ist leer.
<p>count</p>	<ul style="list-style-type: none"> • Siehe Anmerkungen zu Whitespaces im Abschnitt Allgemeine Informationen.
<p>current-date, current-dateTime, current-time</p>	<ul style="list-style-type: none"> • The argument is a relative URI that is resolved against the current base URI. • If the resolved URI identifies an XML file, then this XML file is treated as a catalog which references a collection of files. This file must have the form: <pre><collection> <doc href="uri-1" /> <doc href="uri-2" /> <doc href="uri-3" /> </collection></pre> The files referenced by the href attributes are loaded, and their document nodes are returned as a sequence. • If the resolved URI does not identify an XML file with the catalog structure described above, then the argument string is used as a file-search expression (in which wildcards such as ? and *) are allowed. The specified directory, as identified by the base URI and search string, is searched. XML files with names that matches the search expression are loaded, and their document nodes are returned as a sequence. • The default collection is empty.

deep-equal	<ul style="list-style-type: none"> • Siehe Anmerkung zu Whitespace im Abschnitt Allgemeine Informationen.
doc	<ul style="list-style-type: none"> • Es wird nur dann ein Fehler ausgegeben, wenn unter dem angegebenen Pfad keine XML-Datei vorhanden ist oder wenn die Datei nicht wohlgeformt ist. Die Datei wird validiert, wenn ein Schema vorhanden ist. Wenn die Datei nicht gültig ist, so wird die ungültige Datei ohne Schema-Informationen geladen.
id	<ul style="list-style-type: none"> • In einem wohlgeformten aber ungültigen Dokument, das zwei oder mehr Elemente mit demselben ID-Wert enthält, wird das erste Element in der Dokumentenreihenfolge zurückgegeben.
in-scope-prefixes	<ul style="list-style-type: none"> • Im XML-Dokument dürfen nur Default Namespaces entdeclariert werden. Doch selbst wenn ein Default Namespace an einem Element-Node entdeclariert wird, wird das Präfix für den Default Namespace, welches der Nulllängenstring ist, für diesen Node zurückgegeben.
last	<ul style="list-style-type: none"> • Siehe Anmerkung zu Whitespace im Abschnitt Allgemeine Informationen.
lower-case	<ul style="list-style-type: none"> • Es wird der Unicode-Zeichensatz unterstützt.
normalize-unicode	<ul style="list-style-type: none"> • Es werden die Normalisierungsformulare NFC, NFD, NFKC und NFKD unterstützt.
position	<ul style="list-style-type: none"> • Siehe Anmerkung zu Whitespace im Abschnitt Allgemeine Informationen.
resolve-uri	<ul style="list-style-type: none"> • Wenn das zweite, optionale Argument weggelassen wird, wird die aufzulösende URI (das erste Argument) gegen die Basis-URI aus dem statischen Kontext aufgelöst. Dies ist die URI des XSLT Stylesheets oder die im Prolog des XQuery-Dokuments angegebene Basis-URI. • Die relative URI (das erste Argument) wird nach dem letzten "/" in der Pfadnotation der Basis-URI-Notation angehängt. • Wenn der Wert des ersten Arguments der Nulllängenstring ist, wird die Basis-URI aus dem statischen Kontext zurückgegeben und diese URI inkludiert den Dateinamen des Dokuments, von dem die Basis-URI des statischen Kontexts abgeleitet wird (z.B. die XSLT- oder XML-Datei).
static-base-uri	<ul style="list-style-type: none"> • Die Basis-URI aus dem statischen Kontext ist die Basis-URI des XSLT Stylesheets oder die im Prolog des XQuery-Dokuments angegebene Basis-URI. • Bei Verwendung des XPath Evaluators in der XMLSpy IDE ist die Basis-URI aus dem statischen Kontext die URI des aktiven XML-Dokuments.

upper-case	<ul style="list-style-type: none">• Der Unicode-Zeichensatz wird unterstützt.
------------	---

5.6 Erweiterungen

Es gibt in Programmiersprachen wie Java und C# eine Reihe von fertigen Funktionen, die nicht als XPath 2.0 / XQuery 1.0-Funktionen oder XSLT 2.0-Funktionen zur Verfügung stehen. Ein gutes Beispiel für solche Funktionen sind die mathematischen in Java verfügbaren Funktionen wie z.B. `sin()` und `cos()`. Stünden diese Funktionen für die Erstellung von XSLT Stylesheets und XQuery-Abfragen zur Verfügung, würde sich der Einsatzbereich von Stylesheets und Abfragen erweitern und die Erstellung von Stylesheets wäre viel einfacher.

Die Altova-Prozessoren (XSLT 1.0, XSLT 2.0 und XQuery 1.0), die in einer Reihe von Altova-Produkten verwendet werden, unterstützen die Verwendung von Erweiterungsfunktionen in Java und .NET. Zusätzlich unterstützen die Altova XSLT-Prozessoren MSXSL Scripts für XSLT 1.0 und 2.0.

Beachten Sie, dass Erweiterungsfunktionen immer von XPath-Ausdrücken aus aufgerufen werden. In diesem Abschnitt wird beschrieben, wie Sie Erweiterungsfunktionen und MSXSL Scripts in Ihren XSLT Stylesheets und XQuery-Abfragen verwenden können. Diese Beschreibungen finden Sie in den folgenden Abschnitten:

- [Java-Erweiterungsfunktionen](#)
- [.NET-Erweiterungsfunktionen](#)
- [MSXSL Scripts für XSLT](#)
- [Altova Erweiterungsfunktionen](#)

Hauptsächlich werden dabei die folgenden beiden Punkte behandelt: (i) Wie Funktionen in den entsprechenden Bibliotheken aufgerufen werden; und (ii) welche Regeln beim Konvertieren von Argumenten in einem Funktionsaufruf in das erforderliche Format der Funktion befolgt werden und welche Regeln bei der Rückwärtskonvertierung (Funktionsresultat in XSLT/XQuery Datenobjekt) befolgt werden.

Voraussetzungen

Damit die Erweiterungsfunktionen unterstützt werden, muss auf dem Rechner, auf dem die XSLT-Transformation oder die XQuery-Ausführung stattfindet, eine Java Runtime-Umgebung (zum Aufrufen der Java-Funktionen) installiert sein oder es muss Zugriff auf eine solche bestehen.

5.6.1 Java-Erweiterungsfunktionen

Eine Java-Erweiterungsfunktion kann in einem XPath- oder XQuery-Ausdruck verwendet werden, um einen Java-Konstruktor oder eine Java-Methode (statisch oder Instanz) aufzurufen.

Ein Feld in einer Java-Klasse wird als Methode ohne Argument betrachtet. Bei einem Feld kann es sich um ein statisches Feld oder eine Instanz handeln. Wie man Felder aufruft, wird in den entsprechenden Unterabschnitten zu statischen Feldern und Instanzen beschrieben.

Dieser Abschnitt enthält die folgenden Unterabschnitte:

- [Java: Konstruktoren](#)
- [Java: Statische Methoden und statische Felder](#)
- [Java: Instanzmethoden und Instanzfelder](#)
- [Datentypen: XSLT/XQuery in Java](#)
- [Datentypen: Java in XSLT/XQuery](#)

Form der Erweiterungsfunktion

Die Erweiterungsfunktion im XPath/XQuery-Ausdruck muss die folgenden Form haben

`präfix:fname()`.

- Der Teil `präfix:` kennzeichnet die Erweiterungsfunktion als Java-Funktion, indem er die Erweiterungsfunktion mit einer in-scope Namespace-Deklaration verknüpft, deren URI mit `java:` beginnen muss (*Beispiele siehe unten*). Die Namespace-Deklaration sollte eine Java-Klasse bezeichnen, z.B.:
`xmlns:myns="java:java.lang.Math"`. Sie könnte aber auch einfach lauten:
`xmlns:myns="java"` (ohne Doppelpunkt), wobei die Identifizierung der Java-Klasse dem `fname()` Teil der Erweiterungsfunktion überlassen bleibt.
- Der Teil `fname()` identifiziert die aufgerufene Java-Methode und liefert die Argumente für die Methode (*Beispiele siehe unten*). Wenn die durch das `präfix:` Teil identifizierte Namespace URI jedoch keine Java-Klasse bezeichnet (*siehe vorheriger Punkt*), dann sollte die Java-Klasse im `fname()` Teil vor der Klasse identifiziert werden und von der Klasse durch einen Punkt getrennt sein (*siehe zweites XSLT-Beispiel unten*).

Anmerkung: Die aufgerufene Klasse muss sich unter dem Klassenpfad des Rechners befinden.

XSLT-Beispiel

Hier sehen Sie zwei Beispiele dafür, wie eine statische Methode aufgerufen werden kann. Im ersten Beispiel ist der Klassenname (`java.lang.Math`) in der Namespace URI enthalten und darf daher nicht im `fname()` Teil enthalten sein. Im zweiten Beispiel liefert der `präfix:` Teil das Präfix `java:`, während der `fname()` Teil die Klasse sowie die Methode identifiziert.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
  select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
  select="jmath:java.lang.Math.cos(3.14)" />
```

Die in der Erweiterungsfunktion (im Beispiel oben `cos()`) angegebene Methode muss mit dem Namen einer öffentlichen statischen Methode in der angegebenen Java-Klasse (im Beispiel oben `java.lang.Math`) übereinstimmen.

XQuery-Beispiel

Hier sehen Sie ein XQuery-Beispiel, das dem XSLT-Beispiel oben ähnlich ist:

```
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```

Benutzerdefinierte Java-Klassen

Wenn Sie Ihre eigenen Java-Klassen erstellt haben, werden die Methoden in diesen Klassen unterschiedlich aufgerufen, je nachdem: (i) ob die Klassen über eine JAR-Datei oder eine Klassendatei aufgerufen werden, und (ii) ob sich diese Dateien (JAR oder Klasse) im aktuellen Verzeichnis befinden (im selben Verzeichnis wie das XSLT- oder XQuery-Dokument) oder nicht. Wie Sie diese Dateien finden, wird in den Abschnitten [Benutzerdefinierte Klassendateien](#) und [Benutzerdefinierte Jar-Dateien](#) beschrieben. Pfade zu Klassendateien, die sich nicht im aktuellen Verzeichnis befinden, und Pfade zu allen JAR-Dateien müssen jedoch angegeben werden.

Anmerkung: Wenn Sie einen Namespace zu einem XSLT-Stylesheet hinzufügen möchten, das anhand eines in SPS in StyleVision generiert wurde, muss der Namespace zum `schema`-Element der obersten Ebene des XML-Schemas, auf dem das SPS basiert, hinzugefügt werden. Beachten Sie, dass die folgende Namespace Deklaration `xmlns:java="java"` standardmäßig automatisch in jedem SPS erstellt wird, das in StyleVision entworfen wurde.

Benutzerdefinierte Klassendateien

Wenn der Zugriff über eine Klassendatei erfolgt, gibt es zwei Möglichkeiten:

- Die Klassendatei befindet sich in einem Paket. Die XSLT- oder XQuery-Datei befindet sich im selben Ordner wie das Java-Paket.
- Die Klassendatei befindet sich nicht in einem Paket. Die XSLT- oder XQuery-Datei befindet sich im selben Ordner wie die Klassendatei.
- Die Klassendatei befindet sich in einem Paket. Die XSLT- oder XQuery-Datei befindet sich in irgendeinem beliebig gewählten Ordner.
- Die Klassendatei befindet sich nicht in einem Paket. Die XSLT- oder XQuery-Datei befindet sich in irgendeinem beliebig gewählten Ordner.

Gesetzt der Fall, die Klassendatei befindet sich nicht in einem Paket, sondern im selben Ordner wie das XSLT- oder XQuery-Dokument, so muss der Dateipfad nicht angegeben werden, da alle Klassen im Ordner gefunden werden. Die Syntax zum Identifizieren einer Klasse lautet:

```
java:classname
```

wobei

```
java: angibt, dass eine benutzerdefinierte Java-Funktion aufgerufen wird;
      (Java-Klassen im aktuellen Verzeichnis werden standardmäßig geladen)
classname der Name der Klasse der erforderlichen Methode ist
```

die Klasse in einer Namespace URI identifiziert wird und der Namespace einem Methodenaufruf als Präfix vorangestellt wird.

Klassendatei in einem Paket, XSLT/XQuery-Datei befindet sich im selben Ordner wie das Java-Paket

Im Beispiel unten wird die Methode `getVehicleType()` der Klasse `Car` des Pakets `com.altova.extfunc` aufgerufen. Das Paket `com.altova.extfunc` befindet sich im Ordner `JavaProject`. Die XSLT-Datei befindet sich ebenfalls im Ordner `JavaProject`.

```
<xsl:stylesheet version="2.0"
```

```

    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
  <a>
    <xsl:value-of select="car:getVehicleType()" />
  </a>
</xsl:template>

</xsl:stylesheet>

```

Die Klassendatei befindet sich nicht in einem Paket, die XSLT/XQuery-Datei befindet sich im selben Ordner wie die Klassendatei

Im Beispiel unten wird die Methode `getVehicleType()` der Klasse `Car` class des Pakets `com.altova.extfunc` aufgerufen. Die Klassendatei `Car` class befindet sich im folgenden Ordner: `JavaProject/com/altova/extfunc`. Die XSLT-Datei befindet sich ebenfalls im Ordner `JavaProject/com/altova/extfunc`.

```

<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
  <a>
    <xsl:value-of select="car:getVehicleType()" />
  </a>
</xsl:template>

</xsl:stylesheet>

```

Die Klassendatei befindet sich in einem Paket, die XSLT/XQuery-Datei befindet sich in einem beliebigen Ordner

Im Beispiel unten wird die Methode `getCarColor()` der Klasse `Car` class des Pakets `com.altova.extfunc` aufgerufen. Das Paket `com.altova.extfunc` befindet sich im Ordner `JavaProject`. Die XSLT-Datei befindet sich in einem beliebigen Ordner. In diesem Fall muss der Pfad des Pakets mit der URI als Abfragestring definiert werden. Die Syntax lautet:

```
java:classname[?path=uri-of-classfile]
```

wobei

`java:` angibt, dass eine benutzerdefinierte Java-Funktion aufgerufen wird
`uri-of-classfile` die URI der Klassendatei ist
`classname` der Name der Klasse der benötigten Methode ist

die Klasse in einer Namespace URI identifiziert wird und der Namespace einem Methodenaufruf als Präfix vorangestellt wird. Im Beispiel unten sehen Sie, wie eine Klassendatei aufgerufen wird, die sich in einem anderen als dem aktuellen Verzeichnis befindet.

```

<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"

```

```

        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:car="
java: com.altova.extfunc.Car?path=file:///C:/JavaProject/" >

    <xsl:output exclude-result-prefixes="fn car xsl xs"/>

    <xsl:template match="/">
        <xsl:variable name="myCar" select="car:new(' red' )" />
        <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
    </xsl:template>

</xsl:stylesheet>

```

Die Klassendatei befindet sich nicht in einem Paket, die XSLT/XQuery-Datei befindet sich in einem beliebigen Ordner

Im Beispiel unten wird die Methode `getCarColor()` der Klasse `Car` class des Pakets `com.altova.extfunc` aufgerufen. Das Paket `com.altova.extfunc` befindet sich im Ordner `JavaProject`. Die XSLT-Datei befindet sich in einem beliebigen Ordner. Der Pfad der Klassendatei wird in der Namespace-URI als Abfragestring definiert. Die Syntax lautet:

```
java: classname[?path=uri-of-classfile]
```

wobei

`java:` angibt, dass eine benutzerdefinierte Java-Funktion aufgerufen wird
`uri-of-classfile` die URI der Klassendatei ist
`classname` der Name der Klasse der benötigten Methode ist

die Klasse in einer Namespace URI identifiziert wird und der Namespace einem Methodenaufruf als Präfix vorangestellt wird. Im Beispiel unten sehen Sie, wie eine Klassendatei aufgerufen wird, die sich in einem anderen als dem aktuellen Verzeichnis befindet.

```

<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:car="
java: Car?path=file:///C:/JavaProject/com/altova/extfunc/" >

    <xsl:output exclude-result-prefixes="fn car xsl xs"/>

    <xsl:template match="/">
        <xsl:variable name="myCar" select="car:new(' red' )" />
        <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
    </xsl:template>

</xsl:stylesheet>

```

Anmerkung: Wenn ein Pfad über eine Erweiterungsfunktion angegeben wird, wird er zum `ClassLoader` hinzugefügt.

Benutzerdefinierte Jar-Dateien

JAR-Dateien

Wenn der Zugriff über eine JAR-Datei erfolgt, muss die URI der JAR-Datei mit Hilfe der folgenden Syntax definiert werden:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

Die Methode wird anschließend durch Verwendung des Präfix der Namespace URI

aufgerufen, der die Klasse bezeichnet: `classNS: method()`

wobei im obigen Beispiel:

`java:` angibt, dass eine Java-Funktion aufgerufen wird
`classname` der Name der Klasse der benutzerdefinierten Klasse ist
`?` das Trennzeichen zwischen dem Klassennamen und dem Pfad ist
`path=jar:` angibt, dass es sich um einen Pfad zu einer JAR-Datei handelt
`uri-of-jarfile` die URI der jar-Datei angibt
`! /` das Trennzeichen am Ende des Pfades ist
`classNS: method()` der Aufruf der Methode ist

Alternativ dazu kann der Klassenname mit dem Methodenaufruf angegeben werden. Hier sehen Sie zwei Beispiele für die Syntax:

```
xmlns: ns1="java: docx. layout. pages?path=jar: file: ///c: /projects/docs/docx. jar! /"
"
    ns1: main()

xmlns: ns2="java?path=jar: file: ///c: /projects/docs/docx. jar! /"
ns2: docx. layout. pages. main()
```

Hier sehen Sie ein komplettes XSLT-Beispiel, in dem eine JAR-Datei verwendet wird, um eine Java-Erweiterungsfunktion aufzurufen.:

```
<xsl:stylesheet version="2.0"
    xmlns: xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns: xs="http://www.w3.org/2001/XMLSchema"
    xmlns: fn="http://www.w3.org/2005/xpath-functions"
    xmlns: car="java?path=jar: file: ///C: /test/Carl. jar! /" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
    <xsl:variable name="myCar" select="car: Carl. new(' red' )" />
    <a><xsl: value-of select="car: Carl. getCarColor( $myCar) "/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

Anmerkung: Wenn ein Pfad über eine Erweiterungsfunktion angegeben wird, wird er zum `ClassLoader` hinzugefügt.

Java: Konstruktoren

Eine Erweiterungsfunktion kann dazu verwendet werden, um einen Java-Konstruktor aufzurufen. Alle Konstruktoren werden mit der Pseudofunktion `new()` aufgerufen.

Wenn das Ergebnis eines Java-Konstruktors [implizit in XPath/XQuery-Datentypen konvertiert werden kann](#), dann gibt die Java-Erweiterungsfunktion eine Sequenz zurück, bei der es sich um einen XPath/XQuery-Datentyp handelt. Wenn das Ergebnis eines Konstruktoraufrufs nicht in einen passenden XPath/XQuery-Datentyp konvertiert werden kann, dann erstellt der Konstruktor ein wrapped Java-Objekt mit einem Typ, der den Namen der Klasse hat, die dieses Java-Objekt zurückgibt. Wenn z.B. ein Konstruktor für die Klasse `java.util.Date` aufgerufen wird (`java.util.Date.new()`), so wird ein Objekt vom Typ `java.util.Date` zurückgegeben. Das lexikalische Format des zurückgegebenen Objekts stimmt unter Umständen nicht mit dem lexikalischen Format des XPath-Datentyps überein und der Wert müsste daher in das lexikalische Format des erforderlichen XPath-Datentyps und anschließend in den erforderlichen

XPath-Datentyp konvertiert werden.

Ein von einem Konstruktor erstelltes Java-Objekt kann für zwei Zwecke verwendet werden:

- Es kann einer Variable zugewiesen werden:

```
<xsl:variable name="currentdate" select="date:new()" xmlns:date="
java:java.util.Date" />
```
- Es kann an eine Erweiterungsfunktion übergeben werden (siehe [Instanzmethode und Instanzfelder](#)):

```
<xsl:value-of select="date:toString( date:new() )" xmlns:date="
java:java.util.Date" />
```

Java: Statische Methoden und statische Felder

Eine statische Methode wird direkt über ihren Java-Namen und durch Angabe der Argumente für die Methode aufgerufen. Statische Felder (Methoden, die keine Argumente haben), wie z.B. die Konstantenwertfelder `E` und `PI` werden ohne Angabe eines Arguments aufgerufen.

XSLT-Beispiele

Hier sehen Sie einige Beispiele dafür, wie statische Methoden und Felder aufgerufen werden können:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
select="jMath:E() * jMath:cos(3.14)" />
```

Beachten Sie, dass die Erweiterungsfunktionen die Form `prefix:fname()` haben. Das Präfix ist in allen drei Fällen `jMath:`. Es ist mit der Namespace URI `java:java.lang.Math` verknüpft. (Die Namespace URI muss mit `java:` beginnen. In den obigen Beispielen wurde es um den Klassennamen erweitert (`java.lang.Math`.) Der Teil `fname()` der Erweiterungsfunktionen muss mit dem Namen der öffentlichen Klasse (z.B. `java.lang.Math`) gefolgt vom Namen einer öffentlichen statischen Methode mit ihrem/ihren Argument(en) (wie z.B. `(3.14)`) oder einem öffentlichen statischen Feld (z.B. `PI()`) übereinstimmen.

In den obigen Beispielen wurde der Klassenname in die Namespace URI inkludiert. Wäre sie nicht in der Namespace URI enthalten, müsste sie in den `fname()` Teil der Erweiterungsfunktion inkludiert werden. Z.B:

```
<xsl:value-of xmlns:java="java:"
select="java:java.lang.Math.cos(3.14)" />
```

XQuery-Beispiel

Ein ähnliches Beispiel in XQuery wäre:

```
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```

Java: Instanzmethoden und Instanzfelder

Bei einer Instanzmethode wird als erstes Argument eines Methodenaufrufs ein Java-Objekt an die Methode übergeben. Ein solches Java-Objekt würde normalerweise mit Hilfe einer Erweiterungsfunktion (z.B. eines Konstruktoraufrufs) oder eines Stylesheet-Parameters/einer

Stylesheet-Variablen erstellt. Ein XSLT-Beispiel dafür wäre:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()" />
  <xsl:template match="/">
    <enrollment institution-id="Altova School"
      date="{date:toString($CurrentDate)}"
      type="{jlang:Object.toString(jlang:Object.getClass( date:new()
    ))}">
  </enrollment>
</xsl:template>
</xsl:stylesheet>
```

Im Beispiel oben wird der Wert des Node `enrollment/@type` folgendermaßen erstellt:

1. Es wird ein Objekt mit einem Konstruktor für die Klasse `java.util.Date` (mit dem Konstruktor `date:new()`) erstellt.
2. Dieses Java-Objekt wird als das Argument der Methode `jlang.Object.getClass` übergeben.
3. Das mit der Methode `getClass` abgerufene Objekt wird als das Argument an die Methode `jlang.Object.toString` übergeben.

Das Ergebnis (der Wert von `@type`) ist ein String, der den Wert `java.util.Date` hat.

Ein Instanzfeld unterscheidet sich theoretisch insofern von einer Instanzmethode, als es sich nicht um ein Java-Objekt per se handelt, das als Argument an das Instanzfeld übergeben wird. Stattdessen wird ein Parameter oder eine Variable als Argument übergeben. Der Parameter/die Variable kann allerdings selbst den Wert enthalten, der von einem Java-Objekt zurückgegeben wird. So erhält z.B. der Parameter `CurrentDate` den Wert, der von einem Konstruktor für die Klasse `java.util.Date` zurückgegeben wird. Dieser Wert wird anschließend als Argument an die Instanzmethode `date:toString` übergeben, um den Wert von `/enrollment/@date` bereitzustellen.

Datentypen: XPath/XQuery in Java

Wenn von einem XPath/XQuery-Ausdruck aus eine Java-Funktion aufgerufen wird, spielt der Datentyp der Argumente der Funktion eine wichtige Rolle, welche von mehreren Java-Klassen desselben Namens aufgerufen wird.

In Java gelten die folgenden Regeln:

- Wenn es mehr als eine Java-Methode mit demselben Namen gibt, jede aber eine andere Anzahl von Argumenten als die andere(n) hat, so wird die Java-Methode ausgewählt, die der Anzahl der Argumente im Funktionsaufruf am ehesten entspricht.
- Die XPath/XQuery-Datentypen "string", "number" und "boolean" (*siehe Liste unten*) werden implizit in einen entsprechenden Java-Datentyp konvertiert. Wenn der bereitgestellte XPath/XQuery-Datentyp in mehr als einen Java-Typ konvertiert werden kann (z.B: `xs:integer`), so wird jener Java-Typ ausgewählt, der für die ausgewählte Methode deklariert wurde. Wenn die aufgerufene Java-Methode z.B. `fx(decimal)` und der bereitgestellte XPath/XQuery-Datentyp `xs:integer` ist, so wird `xs:integer` in den Java-Datentyp `decimal` konvertiert.

In der Tabelle unten sehen Sie eine Liste der impliziten Konvertierungen der XPath/XQuery-Datentypen "string", "number" und "boolean" in Java-Datentypen.

<code>xs:string</code>	<code>java.lang.String</code>
<code>xs:boolean</code>	<code>boolean (primitive)</code> , <code>java.lang.Boolean</code>
<code>xs:integer</code>	<code>int</code> , <code>long</code> , <code>short</code> , <code>byte</code> , <code>float</code> , <code>double</code> und die Wrapper-Klassen davon wie z.B. <code>java.lang.Integer</code>
<code>xs:float</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code>
<code>xs:double</code>	<code>double (primitive)</code> , <code>java.lang.Double</code>
<code>xs:decimal</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code> , <code>java.lang.Double</code>

Die oben aufgelisteten Subtypen von XML-Schema-Datentypen (die in XPath und XQuery verwendet werden) werden ebenfalls in den/die Java-Typ(en), der/die dem übergeordneten Subtyp entsprechen, konvertiert.

In einige Fällen ist es nicht möglich, auf Basis der verfügbaren Informationen die richtige Java-Methode auszuwählen. Nehmen Sie als Beispiel den folgenden Fall.

- Das bereitgestellte Argument ist ein `xs:untypedAtomic` Wert 10 und ist für die Methode `mymethod(float)` bestimmt.
- Es gibt jedoch eine weitere Methode in der Klasse, die ein Argument eines anderen Datentypes erhält: `mymethod(double)`.
- Da die Methodennamen dieselben sind und der bereitgestellte Typ (`xs:untypedAtomic`) sowohl in `float` als auch `double` korrekt konvertiert werden könnte, kann es geschehen, dass `xs:untypedAtomic` in `double` anstelle von `float` konvertiert wird.
- Infolgedessen handelt es sich dann bei der ausgewählten Methode nicht um die benötigte Methode, sodass nicht das erwartete Ergebnis erzielt wird. Als Umgehungslösung können Sie eine benutzerdefinierte Methode mit einem anderen Namen erstellen und diese Methode verwenden.

Typen, die in der Liste oben nicht enthalten sind (z.B. `xs:date`), werden nicht konvertiert und generieren einen Fehler. Beachten Sie jedoch, dass es in einigen Fällen unter Umständen möglich ist, den benötigten Java-Typ mittels eines Java-Konstruktors zu erstellen.

Datentypen: Java in XPath/XQuery

Wenn eine Java-Methode einen Wert zurückgibt und der Datentyp des Werts "string", "numeric" oder "boolean" ist, wird anschließend in den entsprechenden XPath/XQuery-Typ konvertiert. So werden z.B. die Java-Datentypen `java.lang.Boolean` und `boolean` in `xsd:boolean` konvertiert.

Von Funktionen zurückgegebene eindimensionale Arrays werden zu einer Sequenz erweitert. Mehrdimensionale Arrays werden nicht konvertiert und sollten daher in einen Wrapper gesetzt werden.

Wenn ein wrapped Java-Objekt oder ein Datentyp zurückgegeben wird, bei dem es sich nicht um den Typ "string", "numeric" oder "boolean" handelt, können Sie sicherstellen, dass die Konvertierung in den benötigten XPath/XQuery-Typ erfolgt, indem Sie zuerst eine Java-Methode (e.g. `toString`) verwenden, um das Java-Objekt in einen String zu konvertieren. In XPath/XQuery kann der String geändert werden, damit er der lexikalischen Darstellung des benötigten Typs entspricht, und anschließend z.B. mit Hilfe des Ausdrucks `cast as` in den

benötigten Typ konvertiert werden.

5.6.2 .NET-Erweiterungsfunktionen

Wenn Sie mit der .NET-Plattform arbeiten, können Sie Erweiterungsfunktionen verwenden, die in jeder beliebigen der .NET-Sprachen geschrieben wurden (z.B. C#). Eine .NET Erweiterungsfunktion kann in einem XPath- oder XQuery-Ausdruck verwendet werden, um einen Konstruktor, eine Eigenschaft oder Methode (statische oder Instanz) in einer .NET-Klasse aufzurufen.

Eine Eigenschaft einer .NET-Klasse wird mit der Syntax `get_PropertyName()` aufgerufen.

Dieser Abschnitt ist in die folgenden Unterabschnitte gegliedert:

- [.NET: Konstruktoren](#)
- [.NET: Statische Methoden und statische Felder](#)
- [.NET: Instanzmethoden und Instanzfelder](#)
- [Datentypen: XSLT/XQuery in .NET](#)
- [Datentypen: .NET in XSLT/XQuery](#)

Form der Erweiterungsfunktion

Die Erweiterungsfunktion im XPath/XQuery-Ausdruck muss die folgende Form haben

`präfix:fname()`.

- Der Teil `präfix:` ist mit einer URI verknüpft, die die benötigte .NET-Klasse definiert.
- Der Teil `fname()` identifiziert den Konstruktor, die Eigenschaft oder die Methode (statisch oder Instanz) innerhalb der .NET-Klasse und liefert alle gegebenenfalls benötigten Argumente.
- Die URI muss mit `clitype:` beginnen (welches die Funktion als .NET-Erweiterungsfunktion kennzeichnet).
- Die Form `präfix:fname()` der Erweiterungsfunktion kann mit Systemklassen und mit Klassen in einer geladenen Assembly verwendet werden. Wenn eine Klasse allerdings geladen werden muss, müssen zusätzliche Parameter mit den benötigten Informationen bereitgestellt werden.

Parameter

Zum Laden einer Assembly werden die folgenden Parameter verwendet:

<code>asm</code>	Der Name der zu ladenden Assembly
<code>ver</code>	Die Versionsnummer: eine Maximalzahl von vier Ganzzahlen, die durch Punkte getrennt sind
<code>sn</code>	Das Key Token des Strong Name der Assembly (16 Hex-Stellen).
<code>from</code>	Eine URI gibt den Pfad der zu ladenden Assembly (DLL) an. Wenn die URI relativ ist, ist sie relativ zum XSLT- oder XQuery-Dokument. Wenn dieser Parameter vorhanden ist, werden alle anderen Parameter ignoriert.
<code>partialname</code>	Der partielle Name der Assembly. Er wird für <code>Assembly.LoadWith.PartialName()</code> bereitgestellt, welches versucht wird, die Assembly zu laden. Wenn <code>partialname</code> vorhanden ist, werden alle anderen Parameter ignoriert.
<code>loc</code>	Die Locale, z.B. <code>en-US</code> . Die Standardeinstellung ist <code>neutral</code>

Wenn die Assembly aus einer DLL geladen werden soll, verwenden Sie den `from` Parameter und lassen Sie den `sn` Parameter weg. Wenn die Assembly aus dem Global Assembly Cache (GAC) geladen werden soll, verwenden Sie den `sn` Parameter und lassen Sie den `from`

Parameter weg.

Vor dem ersten Parameter muss ein Fragezeichen eingefügt werden. Parameter müssen durch ein Semikolon getrennt werden. Der Wert des Parameternamens wird durch ein Ist-Gleich-Zeichen angegeben (*siehe Beispiele unten*).

Beispiele für Namespace-Deklarationen

Ein Beispiel für eine Namespace Deklaration in XSLT, die die Systemklasse `System.Environment`: identifiziert.

```
xmlns:myns="clitype: System.Environment"
```

Ein Beispiel für eine Namespace Deklaration in XSLT, die die zu ladende Klasse als `Trade.Forward.Scrip`: identifiziert.

```
xmlns:myns="clitype: Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

Ein Beispiel für eine Namespace-Deklaration in XQuery, die die Systemklasse `MyManagedDLL.testClass` identifiziert. Es werden zwei Klassen unterschieden:

1. Wenn die Assembly aus dem GAC geladen wird:

```
declare namespace cs="clitype: MyManagedDLL.testClass?asm=MyManagedDLL;
ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. Wenn die Assembly aus der DLL geladen wird (vollständige und partielle Referenzen unten):

```
declare namespace
cs="clitype: MyManagedDLL.testClass?from=file:///C:/Altova
Projects/extFunctions/MyManagedDLL.dll;

declare namespace
cs="clitype: MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

XSLT-Beispiel

Hier sehen Sie ein vollständiges XSLT-Beispiel, in dem Funktionen in der Systemklasse `System.Math`: aufgerufen werden:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" omit-xml-declaration="yes" />
  <xsl:template match="/">
    <math xmlns:math="clitype: System.Math">
      <sqrt><xsl:value-of select="math: Sqrt(9)"/></sqrt>
      <pi><xsl:value-of select="math: PI()"/></pi>
      <e><xsl:value-of select="math: E()"/></e>
      <pow><xsl:value-of select="math: Pow(math: PI(), math: E())"/></pow>
    </math>
  </xsl:template>
</xsl:stylesheet>
```

Die Namespace-Deklaration für das Element `math` verknüpft das Präfix `math:` mit der URI `clitype: System.Math`. Der Beginn der URI `clitype:` gibt an, dass danach entweder eine Systemklasse oder eine geladene Klasse definiert wird. Das Präfix `math:` im XPath-Ausdruck verknüpft die Erweiterungsfunktionen mit der URI (und durch Erweiterung der Klasse) `System.Math`. Die Erweiterungsfunktionen identifizieren Methoden in der Klasse `System.Math` und stellen Argumente bereit, wo dies erforderlich ist.

XQuery-Beispiel

Hier sehen Sie ein XQuery-Beispielfragment ähnlich dem XSLT-Beispiel oben:

```
<math xmlns:math="clitype: System. Math" >
  { math: Sqrt( 9) }
</math>
```

Wie beim XSLT-Beispiel weiter oben identifiziert die Namespace-Deklaration die .NET-Klasse, in diesem Fall eine Systemklasse. Der XQuery-Ausdruck identifiziert die aufzurufenden Methode und liefert das Argument.

.NET: Konstruktoren

Eine Erweiterungsfunktion kann verwendet werden, um einen .NET-Konstruktor aufzurufen. Alle Konstruktoren werden mit der Pseudofunktion `new()` aufgerufen. Wenn es mehrere Konstruktoren für eine Klasse gibt, wird der Konstruktor ausgewählt, der der Anzahl der bereitgestellten Argumente am ehesten entspricht. Wenn kein passender Konstruktor gefunden wird, der den bereitgestellten Argumenten entspricht, wird die Fehlermeldung 'No constructor found' zurückgegeben.

Konstruktoren, die XPath/XQuery-Datentypen zurückgeben

Wenn das Ergebnis eines .NET-Konstruktors [implizit in XPath/XQuery-Datentypen konvertiert werden kann](#), gibt die .NET-Erweiterungsfunktion eine Sequenz zurück, bei der es sich um einen XPath/XQuery-Datentyp handelt.

Konstruktoren, die .NET-Objekte zurückgeben

Wenn das Ergebnis eines .NET-Konstruktoraufrufs nicht in einen passenden XPath/XQuery-Datentyp konvertiert werden kann, erstellt der Konstruktor ein wrapped .NET-Objekt mit einem Typ, der der Name der Klasse ist, die dieses Objekt zurückgibt. Wenn z.B. ein Konstruktor für die Klasse `System.DateTime` aufgerufen wird (mit `System.DateTime.new()`), so wird ein Objekt mit dem Typ `System.DateTime` zurückgegeben.

Das lexikalische Format des zurückgegebenen Objekts stimmt unter Umständen nicht mit dem lexikalischen Format eines erforderlichen XPath-Datentyps überein. In solchen Fällen müsste der zurückgegebene Wert: (i) in das lexikalische Format des benötigten XPath-Datentyps konvertiert werden; und (ii) auf den erforderlichen XPath-Datentyp gecastet werden.

Ein von einem Konstruktor erstelltes .NET-Objekt kann für drei Zwecke verwendet werden:

- Es kann innerhalb einer Variable verwendet werden:

```
<xsl:variable name="currentdate" select="date: new( 2008, 4, 29) "
xmlns:date="clitype: System. DateTime" />
```
- Es kann an eine Erweiterungsfunktion übergeben werden (siehe [Instanzmethode und Instanzfelder](#)):

```
<xsl:value-of select="date: ToString( date: new( 2008, 4, 29) )" xmlns:date
="clitype: System. DateTime" />
```
- Es kann in einen String, eine Zahl oder einen Booleschen Ausdruck konvertiert werden:
- ```
<xsl:value-of select="xs: integer(data: get_ Month(date: new(2008, 4, 29)))
" xmlns:date="clitype: System. DateTime" />
```

### .NET: Statische Methoden und statische Felder

Eine statische Methode wird direkt über ihren Namen und durch Angabe der Argumente für die Methode aufgerufen. Der im Aufruf verwendete Name muss exakt mit einer öffentlichen statischen Methode in der angegebenen Klasse übereinstimmen. Wenn der Methodenname

und die Anzahl der in der Funktion angegebenen Argumente mit mehr als einer Methode in einer Klasse übereinstimmen, werden die Typen der bereitgestellten Argumente nach der besten Übereinstimmung überprüft. Wenn keine eindeutig passende Methode gefunden werden kann, wird ein Fehler ausgegeben.

**Anmerkung:** Ein Feld in einer .NET-Klasse wird als Methode ohne Argument betrachtet. Eine Eigenschaft wird mit der Syntax `get_PropertyName()` aufgerufen.

### Beispiele

Ein XSLT-Beispiel, in dem Sie einen Methodenaufruf mit einem Argument ( `System.Math.Sin(arg)` ) sehen:

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

Ein XSLT-Beispiel, in dem Sie einen Aufruf eines Felds (wird als Methode ohne Argument betrachtet) sehen (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

Ein XSLT-Beispiel, in dem Sie einen Aufruf einer Eigenschaft (Syntax ist `get_PropertyName()`) (`System.String()`) sehen:

```
<xsl:value-of select="string:get_Length(' my string ')" xmlns:string="clitype:System.String"/>
```

Ein XQuery-Beispiel, in dem Sie einen Aufruf einer Methode mit einem Argument ( `System.Math.Sin(arg)` ) sehen:

```
<sin xmlns:math="clitype:System.Math">
 { math:Sin(30) }
</sin>
```

### .NET: Instanzmethoden und Instanzfelder

Bei einer Instanzmethode wird als erstes Argument des Methodenaufrufs ein .NET-Objekt an die Methode übergeben. Dieses .NET-Objekt wird normalerweise mit Hilfe einer Erweiterungsfunktion (z.B. durch einen Konstruktoraufruf) oder einen Stylesheet-Parameter/eine Stylesheet-Variable erstellt. Ein XSLT-Beispiel dieser Art wäre:

```
<xsl:stylesheet version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:fn="http://www.w3.org/2005/xpath-functions">
 <xsl:output method="xml" omit-xml-declaration="yes"/>
 <xsl:template match="/">
 <xsl:variable name="releasedate"
 select="date:new(2008, 4, 29)"
 xmlns:date="clitype:System.DateTime"/>
 <doc>
 <date>
 <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
 xmlns:date="clitype:System.DateTime"/>
 </date>
 </date>
 </template>
```

```

 <xsl:value-of select="date: ToString($releasedate) "
 xmlns: date="clitype: System. DateTime" />
 </date>
</doc>
</xsl:template>
</xsl:stylesheet>

```

Im Beispiel oben wird ein `System.DateTime` Konstruktor (`new(2008, 4, 29)`) verwendet, um ein .NET-Objekt vom Typ `System.DateTime` zu erstellen. Diese Objekt wird zweimal erstellt, einmal als Wert der Variablen `releasedate`, ein zweites Mal als das erste und einzige Argument der Methode `System.DateTime.ToString()`. Die Instanzmethode `System.DateTime.ToString()` wird zwei Mal aufgerufen, beide Male mit dem `System.DateTime` Konstruktor (`new(2008, 4, 29)`) als erstem und einzigem Argument. In einer dieser Instanzen wird die Variable `releasedate` verwendet, um das .NET-Objekt abzurufen.

### Instanzmethoden und Instanzfelder

Der Unterschied zwischen einer Instanzmethode und einem Instanzfeld ist ein theoretischer. In einer Instanzmethode wird ein .NET-Objekt direkt als Argument übergeben; in einem Instanzfeld wird stattdessen ein Parameter oder eine Variable übergeben - auch wenn der Parameter bzw. die Variable selbst ein .NET-Objekt enthalten kann. So enthält z.B. die Variable `releasedate` im Beispiel oben ein .NET-Objekt und es ist diese Variable, die als das Argument von `ToString()` an den zweiten `date` Elementkonstruktor übergeben wird. Die `ToString()` Instanz im ersten `date` Element ist daher eine Instanzmethode, während die zweite als Instanzfeld betrachtet wird. Das in beiden Instanzen erzeugte Ergebnis ist jedoch dasselbe.

### Datentypen: XPath/XQuery in .NET

Wenn in einem XPath/XQuery-Ausdruck eine .NET-Erweiterungsfunktion verwendet wird, spielen die Datentypen der Argumente der Funktion eine wichtige Rolle bei der Entscheidung, welche der vielen .NET-Methoden mit demselben Namen aufgerufen werden soll.

In .NET gelten die folgenden Regeln:

- Wenn es mehr als eine Methode mit demselben Namen in einer Klasse gibt, so stehen nur die Methoden zur Auswahl, die dieselbe Anzahl von Argumenten wie der Funktionsaufruf haben.
- Die XPath/XQuery-Datentypen "string", "number" und "boolean" (*siehe Liste unten*) werden implizit in einen entsprechenden .NET-Datentyp konvertiert. Wenn der bereitgestellte XPath/XQuery-Datentyp in mehr als einen .NET-Typ konvertiert werden kann (z.B. `xs:integer`), so wird jener .NET-Typ ausgewählt, der für die ausgewählte Methode deklariert wurde. Wenn die aufgerufene .NET-Methode z.B. `fx(double)` und der bereitgestellte XPath/XQuery-Datentyp `xs:integer` ist, so wird `xs:integer` in den .NET-Datentyp `double`

In der Tabelle unten sehen Sie eine Liste der impliziten Konvertierungen der XPath/XQuery-Datentypen "string", "number" und "boolean" in .NET-Datentypen.

<code>xs:string</code>	<code>StringValue</code> , <code>string</code>
<code>xs:boolean</code>	<code>BooleanValue</code> , <code>bool</code>
<code>xs:integer</code>	<code>IntegerValue</code> , <code>decimal</code> , <code>long</code> , <code>integer</code> , <code>short</code> , <code>byte</code> , <code>double</code> , <code>float</code>
<code>xs:float</code>	<code>FloatValue</code> , <code>float</code> , <code>double</code>
<code>xs:double</code>	<code>DoubleValue</code> , <code>double</code>



<code>xs:decimal</code>	<code>DecimalValue, decimal, double, float</code>
-------------------------	---------------------------------------------------

Die oben aufgelisteten Subtypen von XML-Schema-Datentypen (die in XPath und XQuery verwendet werden) werden ebenfalls in den/die .NET-Typ(en), der/die dem übergeordneten Subtyp entsprechen, konvertiert.

In einige Fällen ist es nicht möglich, auf Basis der verfügbaren Informationen die richtige .NET-Methode auszuwählen. Nehmen Sie als Beispiel den folgenden Fall.

- Das bereitgestellte Argument ist ein `xs:untypedAtomic` Wert 10 und ist für die Methode `mymethod(float)` bestimmt.
- Es gibt jedoch eine weitere Methode in der Klasse, die ein Argument eines anderen Datentypes erhält: `mymethod(double)`.
- Da die Methodennamen dieselben sind und der bereitgestellte Typ (`xs:untypedAtomic`) sowohl in `float` als auch `double` korrekt konvertiert werden könnte, kann es geschehen, dass `xs:untypedAtomic` in `double` anstelle von `float` konvertiert wird.
- Infolgedessen handelt es sich dann bei der ausgewählten Methode nicht um die benötigte Methode, sodass nicht das erwartete Ergebnis erzielt wird. Als Umgehungslösung können Sie eine benutzerdefinierte Methode mit einem anderen Namen erstellen und diese Methode verwenden.

Typen, die in der Liste oben nicht enthalten sind (z.B. `xs:date`), werden nicht konvertiert und generieren einen Fehler.

#### **Datentypen: .NET in XPath/XQuery**

Wenn eine .NET-Methode einen Wert zurückgibt und der Datentyp des Werts "string", "numeric" oder "boolean" ist, wird er anschließend in den entsprechenden XPath/XQuery-Typ konvertiert. So wird z.B. der .NET-Datentyp `decimal` in `xsd:decimal` konvertiert.

Wenn ein .NET-Objekt oder ein Datentyp zurückgegeben wird, bei dem es sich nicht um den Typ "string", "numeric" oder "boolean" handelt, können Sie sicherstellen, dass die Konvertierung in den benötigten XPath/XQuery-Typ erfolgt, indem Sie zuerst eine .NET-Methode (z.B. `System.DateTime.ToString()`) verwenden, um das .NET-Objekt in einen String zu konvertieren. In XPath/XQuery kann der String geändert werden, damit er der lexikalischen Darstellung des benötigten Typs entspricht, und anschließend z.B. mit Hilfe des Ausdrucks `cast as` in den benötigten Typ konvertiert werden.

### 5.6.3 MSXSL Scripts für XSLT

Das Element `<msxsl:script>` enthält benutzerdefinierte Funktionen und Variablen, die von XPath-Ausdrücken im XSLT-Stylesheet aufgerufen werden können. Das Element `<msxsl:script>` ist ein Element der obersten Ebene, d.h. es muss ein Child-Element von `<xsl:stylesheet>` oder `<xsl:transform>` sein.

Das Element `<msxsl:script>` muss sich im Namespace `urn:schemas-microsoft-com:xslt` (siehe Beispiel unten) befinden.

#### Scripting-Sprache und Namespace

Die im Block verwendete Scripting-Sprache wird im Attribut `language` des Elements `<msxsl:script>` definiert und der für Funktionsaufrufe von XPath-Ausdrücken aus zu verwendende Namespace wird durch das Attribut `implements-prefix` (siehe unten) identifiziert.

```
<msxsl:script language="scripting-language implements-prefix="user-namespace-prefix">
 function-1 or variable-1
 ...
 function-n or variable-n
</msxsl:script>
```

Das Element `<msxsl:script>` interagiert mit der Windows Scripting Runtime. Daher können nur Sprachen, die auf Ihrem Rechner installiert sind, im Element `<msxsl:script>` verwendet werden. **Um MSXSL Scripts verwenden zu können muss die Plattform .NET Framework 2.0 oder höher installiert sein.** Folglich können die .NET Scripting Sprachen innerhalb des Elements `<msxsl:script>` verwendet werden.

Das Attribut `language` akzeptiert dieselben Werte wie das Attribut `language` des HTML `<script>` Elements. Wenn das Attribut `language` nicht definiert ist, wird als Standardsprache Microsoft JScript verwendet.

Das Attribut `implements-prefix` erhält einen Wert, der ein Präfix eines deklarierten in-scope Namespace ist. Bei diesem Namespace handelt es sich normalerweise um einen Benutzer-Namespace, der für eine Funktionsbibliothek reserviert ist. Alle Funktionen und Variablen, die im Element `<msxsl:script>` definiert sind, werden sich im Namespace befinden, der durch das im Attribut `implements-prefix` definierte Präfix identifiziert wird. Wenn eine Funktion von einem XPath-Ausdruck aus aufgerufen wird, muss sich der vollständig qualifizierte Funktionsname im selben Namespace wie die Funktionsdefinition befinden.

#### Beispiel

Hier sehen Sie ein Beispiel für ein vollständiges XSLT Stylesheet, in dem eine Funktion verwendet wird, die in einem `<msxsl:script>` Element definiert ist.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:fn="http://www.w3.org/2005/xpath-functions"
 xmlns:msxsl="urn:schemas-microsoft-com:xslt"
 xmlns:user="http://mycompany.com/mynamespace">

 <msxsl:script language="VBScript" implements-prefix="user">
 <![CDATA[
```

```

' Input: A currency value: the wholesale price
' Returns: The retail price: the input value plus 20% margin,
' rounded to the nearest cent
dim a as integer = 13
Function AddMargin(WholesalePrice) as integer
 AddMargin = WholesalePrice * 1.2 + a
End Function
]]>
</msxsl:script>

<xsl:template match="/">
 <html>
 <body>
 <p>
 Total Retail Price =
 $<xsl:value-of select="user: AddMargin(50)"/>

 Total Wholesale Price =
 $<xsl:value-of select="50"/>

 </p>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>

```

### Datentypen

Die Werte von Parametern, die an und aus dem Script-Block heraus übergeben werden, sind auf XPath-Datentypen beschränkt. Diese Einschränkung gilt nicht für Daten, die zwischen Funktionen und Variablen innerhalb des Script-Blocks übergeben werden.

### Assemblies

Eine Assembly kann über das Element `msxsl: assembly` in das Script importiert werden. Die Assembly wird über einen Namen oder eine URL identifiziert. Die Assembly wird beim Kompilieren des Stylesheet importiert. Hier sehen Sie ein einfaches Beispiel, wie das Element `msxsl: assembly` zu verwenden ist.

```

<msxsl:script>
 <msxsl:assembly name="myAssembly.assemblyName" />
 <msxsl:assembly href="pathToAssembly" />
 ...
</msxsl:script>

```

Der Assembly-Name kann ein vollständiger Name sein, wie z.B.:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

oder ein Kurzname wie z.B. "myAssembly.Draw".

### Namespaces

Namespaces können mit dem Element `msxsl: using` deklariert werden. Auf diese Art können Assembly-Klassen ohne ihre Namespaces in das Script geschrieben werden, wodurch Sie sich das mühsame Eintippen ersparen. Hier sehen Sie, wie das Element `msxsl: using` verwendet wird, um Namespaces zu deklarieren.

```

<msxsl:script>
 <msxsl:using namespace="myAssemblyNS.NamespaceName" />

```

...

```
</msxsl:script>
```

Der Wert des `namespace` Attributs ist der Name des Namespace.

## 5.6.4 Altova Erweiterungsfunktionen

Altova Erweiterungsfunktionen befinden sich im Namespace `http://www.altova.com/xslt-extension` und werden in diesem Abschnitt mit dem Präfix `altova:` gekennzeichnet, das an den oben angegebenen Namespace gebunden ist.

In der aktuellen Version Ihres Altova-Produkts werden die folgenden Erweiterungsfunktionen auf die unten beschriebene Art unterstützt. Beachten Sie jedoch, dass eine oder mehrere dieser Funktionen in zukünftigen Produktversionen eventuell nicht mehr unterstützt werden, bzw. dass sich das Verhalten einzelner Funktionen ändern kann. Um Informationen über die Unterstützung für Altova Erweiterungsfunktionen in der jeweiligen Release zu erhalten, schlagen Sie bitte in der Dokumentation der jeweils aktuellen Release nach.

- [altova: evaluate\(\)](#)
- [altova: distinct-nodes\(\)](#)
- [altova: encode-for-rtf\(\)](#)
- [altova: xbrl-labels\(\)](#)
- [altova: xbrl-footnotes\(\)](#)

### `altova: evaluate()`

Die `altova: evaluate()` Funktion erhält einen XPath-Ausdruck als obligatorisches Argument, der als String übergeben wird, und gibt das Resultat des ausgewerteten Ausdrucks zurück.

```
altova: evaluate(XPathExp as xs: string)
```

Beispiel:

```
altova: evaluate(' //Name[1] ')
```

Beachten Sie im obigen Beispiel, dass der Ausdruck `//Name[ 1]` durch Einschließen in einfache Anführungszeichen als String übergeben wird. Die Funktion `altova: evaluate` gibt den Inhalt des ersten `Name`-Elements im Dokument zurück.

Die Funktion `altova: evaluate` kann zusätzliche (optionale) Argumente erhalten. Diese Argumente sind die Werte der einzelnen Variablen und haben die Namen `p1`, `p2`, `p3... pN`, die im XPath-Ausdruck verwendet werden können.

```
altova: evaluate(XPathExp as xs: string [, p1value ... pNvalue])
```

wobei

- die Variablennamen die Form `pX` haben müssen, wobei `X` eine Ganzzahl ist
- die Reihenfolge der Argumente der Funktion vom zweiten Argument an der Reihenfolge der Variablen mit den Namen `p1` bis `pN` entspricht. Daher wird das zweite Argument der Wert der Variablen `p1`, das dritte Argument der der Variablen `p2`, usw.
- die Werte der Variablen den Typ `item*` haben müssen

Beispiel:

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
Outputs "hi 20 10"
```

Beachten Sie im obigen Beispiel folgende Punkte:

- Das zweite Argument des Ausdrucks `altova: evaluate` ist der der Variablen `$p1`

- zugewiesene Wert, das dritte Argument ist das der Variablen `$p2` zugewiesene usw.
- Beachten Sie, dass das vierte Argument der Funktion ein String-Wert ist. Als String-Wert wird dieser innerhalb von Anführungszeichen gesetzt.
- Das `select` Attribut des Elements `xs:variable` liefert den XPath-Ausdruck. Da dieser Ausdruck den Typ `xs:string`, haben muss, wird er in einfache Anführungszeichen gesetzt..

Im Folgenden ein weiteres Beispiel für die Verwendung:

```
<xs:variable name="xpath" select="'$p1'" />
<xs:value-of select="altova:evaluate($xpath, //Name[1])" />
Outputs value of the first Name element.

<xs:variable name="xpath" select="'$p1'" />
<xs:value-of select="altova:evaluate($xpath, '//Name[1]')" />
Outputs "//Name[1]"
```

Die `altova:evaluate()` Erweiterungsfunktion ist in Situationen nützlich, in denen ein XPath-Ausdruck im XSLT-Stylesheet einen oder mehrere Teile enthält, die dynamisch ausgewertet werden müssen. Angenommen ein Benutzer gibt seinen Request für das Sortierkriterium ein und das Sortierkriterium ist im Attribut `UserReq/@sortkey` gespeichert. Im Stylesheet könnten Sie den folgenden Ausdruck haben:

```
<xs:sort select="altova:evaluate(.. /UserReq/@sortkey)" order="ascending"/>
```

Die `altova:evaluate()` Funktion liest das `sortkey` Attribut des `UserReq` Child-Elements des Parent des Kontext-Node. Angenommen der Wert des `sortkey` Attributs ist `Price`, dann wird von der `altova:evaluate()` Funktion `Price` zurückgegeben und wird zum Wert des `select` Attributs:

```
<xs:sort select="Price" order="ascending"/>
```

Wenn diese `sort` Anweisung im Kontext eines Elements namens `Order` vorkommt, dann werden die `Order` Elemente nach den Werten Ihrer `Price` Children sortiert. Alternativ dazu, wenn der Wert von `@sortkey` z.B. `Date` ist, werden die `Order` Elemente nach den Werten ihrer `Date` Children sortiert. Das Sortierkriterium für `Order` wird also zur Laufzeit aus dem `sortkey` Attribut ausgewählt. Diese hätte man mit einem Ausdruck wie dem folgenden nicht bewerkstelligen können:

```
<xs:sort select=".. /UserReq/@sortkey" order="ascending"/>
```

Im oben gezeigten Beispiel wäre das Sortierkriterium das `sortkey` Attribut selbst, nicht `Price` oder `Date` (oder jeder beliebige andere Inhalt von `sortkey`)

Variablen können in der Erweiterungsfunktion `altova:evaluate()` wie im Beispiel unten gezeigt verwendet werden:

- Statische Variablen: `<xs:value-of select="$i3, $i2, $i1" />`  
*Gibt die Werte von drei Variablen aus.*
- Dynamischer XPath-Ausdruck mit dynamischen Variablen:  
`<xs:variable name="xpath" select="'$p3, $p2, $p1'" />`  
`<xs:value-of select="altova:evaluate( $xpath, 10, 20, 30 )" />`  
*Gibt "30 20 10" aus*
- Dynamischer XPath-Ausdruck ohne dynamische Variable:  
`<xs:variable name="xpath" select="'$p3, $p2, $p1'" />`  
`<xs:value-of select="altova:evaluate( $xpath )" />`  
*Gibt einen Fehler aus.: Es wurde keine Variable für \$p3 definiert*

**Hinweis:**

Der statische Kontext enthält Namespaces, Typen und Funktionen - aber keine Variablen - aus der aufrufenden Umgebung. Die Basis-URI und der Standard-Namespace werden vererbt.

**altova:distinct-nodes()**

Die `altova:distinct-nodes()` Funktion erhält eine Gruppe von einem oder mehreren Nodes als Input und gibt dieselbe Gruppe ohne Nodes mit doppelt vorhandenen Werten zurück. Der Vergleich wird mittels der XPath/XQuery-Funktion `fn:deep-equal` durchgeführt.

```
altova:distinct-nodes($arg as node()*) as node()*
```

**altova:encode-for-rtf()**

Die `altova:encode-for-rtf()` Funktion konvertiert den Input-String in Code für RTF.

```
altova:encode-for-rtf($inputstr as xs:string?,
 $preserveallwhitespace as xs:boolean,
 $preservenewlines as xs:boolean) as xs:string
```

Whitespaces und neue Zeilen werden gemäß dem für die entsprechenden Parameter definierten Booleschen Wert beibehalten.

**altova:xbrl-labels()**

Die `altova:xbrl-labels()` Funktion erhält zwei Input-Argumente: einen Node-Namen und den Pfad der Taxonomiedatei, die den Node enthält. Die Funktion gibt die XBRL Labels zurück, die mit dem Input-Node verknüpft sind.

```
altova:xbrl-labels($name as xs:QName, $file as xs:string) as node()*
```

**altova:xbrl-footnotes()**

Die `altova:footnotes()` Funktion erhält einen Node als Input-Argument und gibt die durch den Input-Node referenzierte Gruppe der XBRL-Fußnoten-Nodes zurück.

```
altova:footnotes($arg as node()) as node()*
```





## **Kapitel 6**

---

### **Lizenzvereinbarung**

## 6 Lizenzvereinbarung

**THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS**  
ALTOVA DEVELOPER LICENSE AGREEMENT  
FOR ALTOVAXML SOFTWARE

Licensor:

Altova GmbH  
Rudolfsplatz 13a/9  
A-1010 Wien  
Austria

**Important – Read Carefully. Notice to User**

**This Altova Developer License Agreement ("DLA") governs your right to use, bundle, integrate and distribute AltovaXML software (the "Software"). Additional information about the Software can be found on the Altova Web Site. This DLA is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software and any accompanying documentation, including, without limitation, printed materials, 'online' files, or electronic documentation ("Documentation"). By installing the Software, or including the Software in your application, or distributing the Software, or otherwise using the Software, you agree to be bound by the terms of this DLA as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below. 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 or distribute the Software, and you must destroy any downloaded copies of the Software in your possession or control. Please go to our Web site at <http://www.altova.com/ALTOVAXMLdla> to download and print a copy of this DLA for your files and <http://www.altova.com/privacy> to review the privacy policy.**

**1. SOFTWARE LICENSE**

**(a) License Grant.** Upon your acceptance of this DLA, Altova grants you a non-exclusive, non-transferable limited worldwide license to: (i) develop software applications that include the Software and/or Documentation, (ii) reproduce the Software and/or Documentation, and (iii) distribute the Software in executable form and Documentation in the manner hereinafter provided to end users for the purpose of being used in conjunction with a software application developed by you.

**(b) Internal Use.** You may install the Software on a server within your network for the purpose of downloading and installing the Software (to an unlimited number of client computers on your internal network).

**(c) External Use.** You may distribute the Software and/or Documentation to any third party electronically or via download from the website or on physical media such as CD-ROMS or diskettes as part of or in conjunction with products that you have developed.

**(d) Distribution Restrictions.** In addition to the restrictions and obligations provided in other sections of this DLA, your license to distribute the Software and/or Documentation is further subject to all of the following restrictions: (i) the Software and/or Documentation shall only be licensed and not sold; (ii) you may not make the Software and/or Documentation available as a stand alone product and if distributed as part of a product bundle you may charge for the product bundle provided that you license such product bundle at the same or lower fee at which you license any reasonably equivalent product bundle which does not include the Software; (iii) you must use the Software and/or Documentation provided by Altova AS IS and may not impair, alter or remove Altova's copyright or license statements or any other files; and (iv) other Altova products cannot be distributed or used under this DLA.

**(e) Title.** This DLA gives you a limited license to reproduce and distribute the Software and/or

Documentation. Altova and its suppliers retain all right, title and interest, including all copyright and intellectual property rights, in and to, the Software and/or Documentation and all copies thereof. All rights not specifically granted in this DLA are reserved by Altova.

**(f) Reverse Engineering.** 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 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.

**(g) Additional Restrictions.** You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software and/or Documentation to third parties except to the limited extent expressly provided herein. You may not copy, distribute or make derivative works of the Software and/or Documentation except as expressly set forth above, and any copies that you are permitted to make pursuant to this DLA must contain the same copyright, patent and other intellectual property markings that appear on or in the Software and/or Documentation. You may not alter, modify, adapt or translate the Software and/or Documentation or any part thereof. 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 and/or Documentation to be placed in the public domain; or use the Software and/or Documentation in any computer environment not specified in this DLA. You will comply with applicable law and Altova's instructions regarding the use of the Software and/or Documentation. You agree to notify your employees and agents who may have access to the Software and/or Documentation of the restrictions contained in this DLA and to ensure their compliance with these restrictions. You agree to indemnify, hold harmless, and defend Altova from and against any claims or lawsuits, including attorney's fees that arise or result from your use or distribution of the Software and/or Documentation.

## 2. INTELLECTUAL PROPERTY RIGHTS

**Acknowledgement of Altova's Rights.** You acknowledge that the Software and/or Documentation 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 and/or Documentation are the valuable trade secrets and confidential information of Altova and its suppliers. The Software and/or Documentation 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/or Documentation, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and/or Documentation and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software and/or Documentation. 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., XMLSPY, AUTHENTIC, STYLEVISION, MAPFORCE, SCHEMAAGENT, DIFFDOG, UMODEL MARKUP YOUR MIND, AXAD, NANONULL, and ALTOVA are trademarks and/or registered trademark of Altova GmbH. Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows 95, Windows 98, Windows NT, Windows 2000 and Windows XP 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 DLA does not grant you any intellectual property rights in the Software and/or Documentation. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web site.

**3. WARRANTY DISCLAIMER AND LIMITATION OF LIABILITY**

(a) THE SOFTWARE AND/OR DOCUMENTATION ARE PROVIDED TO YOU FREE OF CHARGE, AND ON AN "AS-IS" BASIS. ALTOVA PROVIDES NO TECHNICAL SUPPORT OR WARRANTIES FOR THE SOFTWARE AND/OR DOCUMENTATION. TO THE MAXIMUM EXTENT PERMITTED BY LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES AND REPRESENTATIONS, WHETHER EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE; MERCHANTABILITY; SATISFACTORY QUALITY, INFORMATIONAL CONTENT, OR ACCURACY, QUIET ENJOYMENT, TITLE, AND NON- INFRINGEMENT. ALTOVA DOES NOT WARRANT THAT THE SOFTWARE IS ERROR-FREE OR WILL OPERATE WITHOUT INTERRUPTION. IF APPLICABLE LAW REQUIRES ANY WARRANTIES WITH RESPECT TO THE SOFTWARE, ALL SUCH WARRANTIES ARE LIMITED IN DURATION TO 30 DAYS FROM THE DATE OF INSTALLATION OR USE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER LEGAL RIGHTS THAT VARY FROM STATE TO STATE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL INDEMNIFY AND HOLD HARMLESS ALTOVA FROM ANY 3RD PARTY SUIT TO THE EXTENT BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND/OR DOCUMENTATION IN YOUR USE. WITHOUT LIMITATION, THE SOFTWARE IS NOT INTENDED FOR USE IN HAZARDOUS ENVIRONMENTS REQUIRING FAIL-SAFE CONTROLS INCLUDING WITHOUT LIMITATION THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL, LIFE SUPPORT, OR WEAPONS SYSTEMS, WHERE THE FAILURE OF THE SOFTWARE COULD LEAD TO DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE.

(b) TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, 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 AND/OR DOCUMENTATION, OR ANY PROVISION OF THIS DLA, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. WHERE LEGALLY, LIABILITY CANNOT BE EXCLUDED, BUT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIVE DOLLARS (USD. \$5.00) IN TOTAL. 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 AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE GREATEST EXTENT PERMITTED BY LAW. THE FOREGOING LIMITATIONS ON LIABILITY ARE INTENDED TO APPLY TO THE WARRANTIES AND DISCLAIMERS ABOVE AND ALL OTHER ASPECTS OF THIS DLA.

**4. DATA USE**

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 DLA. By your acceptance of the terms of this DLA or use of the Software, you authorize the collection, use and

disclosure of information collected by Altova for the purposes provided for in this -DLA and/or the Privacy Policy as revised from time to time. European users understand and consent to the processing of personal information in the United States for the purposes described herein. Altova has the right in its sole discretion to amend this provision of the DLA 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.

**5. EXPORT RULES AND GOVERNMENT RESTRICTED RIGHTS**

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 is 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 **DLA**. 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 and/or Documentation was obtained. In particular, but without limitation, the Software and/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 and/or Documentation, 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.

**6. TERM AND TERMINATION**

Without prejudice to any other rights or remedies of Altova, this DLA may be terminated (a) by you giving Altova written notice of termination; or (b) by Altova, for any or no reason, giving you written notice of termination or (c) Altova giving you written notice of termination if you fail to comply with the terms and conditions of the DLA. Upon any termination of this DLA, you must cease all use of the Software and/or Documentation, licensed hereunder, 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 and/or Documentation remain in your possession or control. The terms and conditions set forth in Sections 1 (e), (f), (g), 2,3, 5, 6 , and 7 survive termination of this agreement as applicable.

**7. GENERAL PROVISIONS**

If you are located in the European Union and are using the Software and/or Documentation in the European Union and not in the United States, then this DLA 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 and/or Documentation 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 and/or Documentation in the United States then this DLA will be governed by and construed in accordance with the law 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 and/or Documentation resides in the federal or state courts of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts 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 and/or Documentation in the United States, then this DLA 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 and/or Documentation 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 DLA 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. This DLA contains the entire agreement and understanding of the parties with respect to 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 DLA 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 DLA 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 DLA. This DLA 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 DLA 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 DLA. If, for any reason, any provision of this DLA is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this DLA, and this DLA 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: 2005-06-28

# Index

## **.NET Erweiterungsfunktionen,**

- Datentypkonvertierungen, .NET in XPath/XQuery, 115
- Datentypkonvertierungen, XPath/XQuery in .NET, 114
- für XSLT und XQuery, 110
- Instanzmethoden, Instanzfelder, 113
- Konstrukturen, 112
- statische Methoden, statische Felder, 112
- Übersicht, 110

## **.NET-Schnittstelle,**

- Beispielcode, 66
- Features, 4
- Objekt Altova.AltovaXML.XMLValidator, 68
- Objekt Altova.AltovaXML.XQuery, 74
- Objekt Altova.AltovaXML.XSLT1, 70
- Objekt Altova.AltovaXML.XSLT2, 72
- Objektmodell, 66
- Verwendung, 64, 66

## A

### **Altova XSLT 1.0-Prozessor,**

- Einschränkungen und implementierungsspezifisches Verhalten, 84

### **Altova XSLT 2.0-Prozessor,**

- allgemeine Informationen, 87
- Informationen, 86

### **Altova.AltovaXML.Applikationsobjekt, 66**

### **Altova.AltovaXML.dll, 7, 64**

### **AltovaXML,**

- Befehlszeilenfeatures, 4
- Benutzerhandbuch, 3
- Dokumentation, 3
- Einführung, 3
- Features der COM-Schnittstelle, 4
- Funktionsumfang, 5
- Hauptfunktionen, 4
- Installation, 7
- Paket, 4
- Systemanforderungen, 7

- Verwendung, 10

### **AltovaXML registrieren,**

- als COM-Serverobjekt, 7

### **AltovaXML.jar, 40**

- und CLASSPATH, 7

### **AltovaXMLLib.dll, 7, 40**

### **Atomisierung von Nodes,**

- in XPath 2.0- und XQuery 1.0-Auswertung, 94

## B

### **Befehlszeile,**

- Features, 4
- für XQuery 1.0 Ausführungen, 20
- für XSLT 1.0 Transformationen, 16
- für XSLT 2.0 Transformationen, 18
- Hilfe, 11
- Validierung und Wohlgeformtheitsprüfung, 13
- Versionsinformationen, 11
- Zusammenfassung Verwendung, 11

### **Beispiel, 66**

### **Beispiele, 22, 36, 40**

### **Bibliotheksmodule,**

- in XQuery-Dokument, 90

## C

### **C# Beispielcode,**

- für .NET-Schnittstelle, 66

### **C++-Beispielcode,**

- für COM-Schnittstelle, 38

### **CLASSPATH,**

- und AltovaXML.jar, 7

### **Collations,**

- in XPath 2.0, 94
- in XQuery-Dokument, 90

### **COM interface,**

- XQuery interface, 33

### **COM Server,**

- freigeben, 78

### **COM Serverobjekt,**

- AltovaXML registrieren, 11

### **com.altova-Prozessoren, 40**

### **COM-Schnittstelle,**

**COM-Schnittstelle,**

- Applikationsobjekt, 25
- Beispielcode, 36
- C++-Beispielcode, 38
- Features, 4
- JScript-Beispielcode, 37
- Objektmodell, 24
- Validator-Schnittstelle, 26
- Verwendung, 22
- Visual Basic-Beispielcode, 36
- XSLT1-Schnittstelle, 28
- XSLT2-Schnittstelle, 30

**COM-Serverobjekt,**

- AltovaXML registrieren, 7
- Registrieren von AltovaXML, 23

**count() function in XPath 2.0,**

- siehe fn:count(), 94

**count()-Funktion,**

- in XPath 1.0, 84

**D****Datentypen,**

- in XPath 2.0 und XQuery 1.0, 94

**deep-equal() function in XPath 2.0,**

- siehe fn:deep-equal(), 94

**Default Functions Namespace,**

- für XPath 2.0- und XQuery 1.0-Ausdrücke, 94
- in XSLT 2.0 Stylesheets, 87

**Dispatch-Schnittstelle,**

- Beschreibung, 22

**Dokumentation,**

- Übersicht, 8

**Dot NET,**

- siehe .NET, 64

**E****Encoding,**

- in XQuery-Dokument, 90

**Erweiterungsfunktionen für XSLT und XQuery, 100****Erweiterungsfunktionen in .NET für XSLT und XQuery,**

- siehe .NET Erweiterungsfunktionen, 110

**Erweiterungsfunktionen in Java für XSLT und XQuery,**

- siehe Java-Erweiterungsfunktionen, 101

**Erweiterungsfunktionen in MSXSL Scripts, 116****Externe Funktionen,**

- in XQuery-Dokument, 90

**F****fn:base-uri in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:collection in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:count() in XPath 2.0,**

- und Whitespace, 94

**fn:current-date in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:current-dateTime in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:current-time in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:data in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:deep-equal() in XPath 2.0,**

- und Whitespace, 94

**fn:id in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:idref in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:index-of in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:in-scope-prefixes in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:last() in XPath 2.0,**

- und Whitespace, 94

**fn:lower-case in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:normalize-unicode in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:position() in XPath 2.0,**

- und Whitespace, 94

**fn:resolve-uri in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:static-base-uri in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**fn:upper-case in XPath 2.0,**

- Unterstützung in Altova-Prozessoren, 96

**Freigegeben,**



**Freigeben,**

COM server, 78

**functions,**

XPath 2.0 and XQuery 1.0, 93

**Funktionen,**

siehe unter XSLT 2.0-Funktionen, 89

von AltovaXML, 5

## H

**Hilfe,**

über die Befehlszeile, 11

## I

**Implementierungsspezifisches Verhalten,**

von XSLT 2.0-Funktionen, 89

**Implizite Zeitzone,**

und XPath 2.0-Funktionen, 94

**Installation,**

von AltovaXML, 7

## J

**Java Erweiterungsfunktionen,**

Datentypkonvertierungen, Java in Xpath/XQuery, 108

Instanzmethoden, Instanzfelder, 106

Konstruktoren, 105

statische Methoden, statische Felder, 106

**Java extension functions,**

datatype conversions, XPath/XQuery to Java, 107

**Java-Erweiterungsfunktionen,**

benutzerdefinierte JAR-Dateien, 104

benutzerdefinierte Klassendateien, 102

für XSLT und XQuery, 101

Übersicht, 101

**Java-Klasse AltovaXMLFactory,**

Beschreibung, 50

**Java-Klasse XMLValidator,**

Beschreibung, 51

**Java-Klasse XQuery,**

Beschreibung, 54

**Java-Klasse XSLT1,**

Beschreibung, 58

**Java-Klasse XSLT2,**

Beschreibung, 60

**Java-Schnittstelle,**

Beispielcode, 40

Features, 4

Installation, 40

Übersicht über die Klassen, 42

Übersicht über Klassen, 50

Verwendung, 40

zusätzliche Dokumentation, 40

**Java-Schnittstelle IAltovaXMLEngine,**

Beschreibung, 42

**Java-Schnittstelle IAltovaXMLFactory,**

Beschreibung, 43

**Java-Schnittstelle IExecutable,**

Beschreibung, 44

**Java-Schnittstelle IReleasable,**

Beschreibung, 44

**Java-Schnittstelle IXMLValidator,**

Beschreibung, 45

**Java-Schnittstelle IXQuery,**

Beschreibung, 46

**Java-Schnittstelle IXSLT,**

Beschreibung, 48

**JScript-Beispielcode,**

für COM-Schnittstelle, 37

## L

**last() function in XPath 2.0,**

siehe fn:last(), 94

**last()-Funktion,**

in XPath 1.0, 84

**Löschen der Registrierung von AltovaXML als COM-Serverobjekt, 23**

## M

**msxsl:Script, 116**

## N

### Namespaces,

- im XSLT 2.0 Stylesheet, 87
- in XQuery-Dokument, 90

## O

### OOXML-Dateien, 80

## P

### position() function,

- in XPath 1.0, 84

### position() function in XPath 2.0,

- siehe fn:position(), 94

### Prozessorinformationen, 82

## Q

### QName Serialisierung,

- Rückgabe durch XPath 2.0 -Funktionen, 96

## R

### Raw-Schnittstelle,

- Beschreibung, 22

### Registrieren von AltovaXML,

- als COM-Serverobjekt, 23

### Registrierung von AltovaXML,

- als COM Serverobjekt, 11

### Rückwärtskompatibilität,

- des XSLT 2.0-Prozessors, 87

## S

### Schemafähigkeit,

- des XPath 2.0- und XQuery-Prozessors, 94

### Schema-Validierung eines XML-Dokuments,

- für XQuery, 90

### Scripts in XSLT/XQuery,

- siehe Erweiterungsfunktionen, 100

### Standardkonformität,

- der Altova-Prozessoren, 82
- des Altova XML Validators, 83

## V

### Validierung,

- über die .NET-Schnittstelle, 68
- über die Befehlszeile, 13
- verfügbare Funktionen, 5

### Versionsinformationen,

- Befehlszeile, 11

### Verwendung,

- von AltovaXML, 10

### Visual Basic-Beispielcode,

- für COM-Schnittstelle, 36

## W

### Whitespace im XML-Dokument,

- Behandlung durch den Altova XSLT 2.0-Prozessor, 87

### Whitespace Nodes im XML-Dokument,

- und Behandlung durch den XSLT 1.0-Prozessor, 84

### Whitespace-Behandlung,

- und XPath 2.0-Funktionen, 94

### Wohlgeformtheitsprüfung,

- über die .NET-Schnittstelle, 68
- über die Befehlszeile, 13

## X

### XML-Validierung,

- siehe Validierung, 68

### XPath 2.0 Functions,

- implementation information, 93

### XPath 2.0-Funktionen,

- allgemeine Informationen, 94
- spezifische Funktionen siehe unter fn:, 94

**XPath-Funktionen - Unterstützung,**

für einzelne Funktionen siehe unter fn:, 96

**XQuery,**

Erweiterungsfunktionen, 100

**XQuery 1.0 Functions,**

implementation information, 93

**XQuery 1.0 Transformationen,**

über die .NET-Schnittstelle, 74

**XQuery 1.0-Funktionen,**

allgemeine Informationen, 94

spezifische Funktionen siehe unter fn:, 94

**XQuery 1.0-Prozessor,**

Informationen, 90

**XQuery-Ausführung,**

verfügbare Funktionen, 5

**XQuery-Ausführungen,**

über die BefehlszeileCommand line, 20

**xs:QName,**

siehe auch QName, 96

**xsl:preserve-space, 84****xsl:strip-space, 84****XSLT,**

Erweiterungsfunktionen, 100

**XSLT 1.0 Transformationen,**

über die .NET-Schnittstelle, 70

über die Befehlszeile, 16

**XSLT 1.0-Prozessor,**

Einschränkungen und implementierungsspezifisches Verhalten, 84

**XSLT 2.0 Stylesheet,**

Namespace-Deklarationen, 87

**XSLT 2.0 Transformationen,**

über die .NET-Schnittstelle, 72

über die Befehlszeile, 18

**XSLT 2.0-Funktionen,**

implementierungsspezifisches Verhalten, 89

spezifische Funktionen siehe unter fn:, 89

**XSLT 2.0-Prozessor,**

allgemeine Informationen, 87

Informationen, 86

**XSLT-Transformationen,**

verfügbare Funktionen, 5

in der HTML-Ausgabe von XSLT-Transformationen, 84

**Zeichennormalisierung,**

in XQuery-Dokument, 90

**ZIP-Dateien, 80**

## Z

**Zeichen-Entities,**