Manual de referencia y del usuario

Manual del usuario y referencia de Altova RaptorXML Development Edition 2014

Fecha de publicación: 2013

Tabla de contenido

Altova RaptorXML Development Edition 2014

Introducción a RaptorXML

1 Introducción a RaptorXML

Altova RaptorXML Development Edition (en adelante RaptorXML) es el rapidísimo motor XML y XBRL de tercera generación de Altova, optimizado para los estándares más recientes y para entornos de informática en paralelo. RaptorXML es compatible con múltiples plataformas y aprovecha la omnipresencia actual de equipos multinúcleo para ofrecer rapidísimas funciones de procesamiento de datos XML y XBRL.

*** Nota:** las funciones de procesamiento XBRL solamente están disponibles en RaptorXML+XBRL Server (no están disponibles en RaptorXML Server ni en RaptorXML Development Edition).

Ediciones y sistemas operativos
Altova ofrece tres ediciones diferentes de RaptorXML, diseñadas para satisfacer diferentes requisitos. Las tres ediciones de RaptorXML se describen en el apartado Ediciones e interfaces . Mientras que la edición Development Edition solamente es compatible con el sistema operativo Windows, las dos ediciones servidor están disponibles para Windows, Linux y Mac OS X. Para más información consulte el apartado Requisitos del sistema.

Características y especificaciones compatibles
RaptorXML ofrece funciones de validación XML, transformación XSLT y ejecución de XQuery dotadas de numerosas y potentes opciones. Para ver la lista de características y funciones clave de RaptorXML, consulte el apartado Características. En el apartado Especificaciones compatibles se enumeran todas las especificaciones con las que cumple RaptorXML. Para más información visite el sitio web de Altova.

Esta documentación
La presente documentación está incluida en la aplicación y también está disponible en el sitio web de Altova. Tenga en cuenta que el explorador Chrome tiene una restricción que no permite expandir las entradas de la tabla de contenido cuando la documentación se abre localmente. Sin embargo, si abre la documentación desde un servidor web, la tabla de contenido funciona correctamente en Chrome.

La presente documentación se divide en varias secciones:

- Introducción a RaptorXML (la presente sección)
- Instalar RaptorXML
- Interfaz de la línea de comandos
- Interfaz Java
- Interfaz COM/.NET
- Información sobre motores XSLT y XQuery
- Funciones de extensión XSLT y XQuery

Última actualización: 07/11/2013

1.1 Ediciones e interfaces

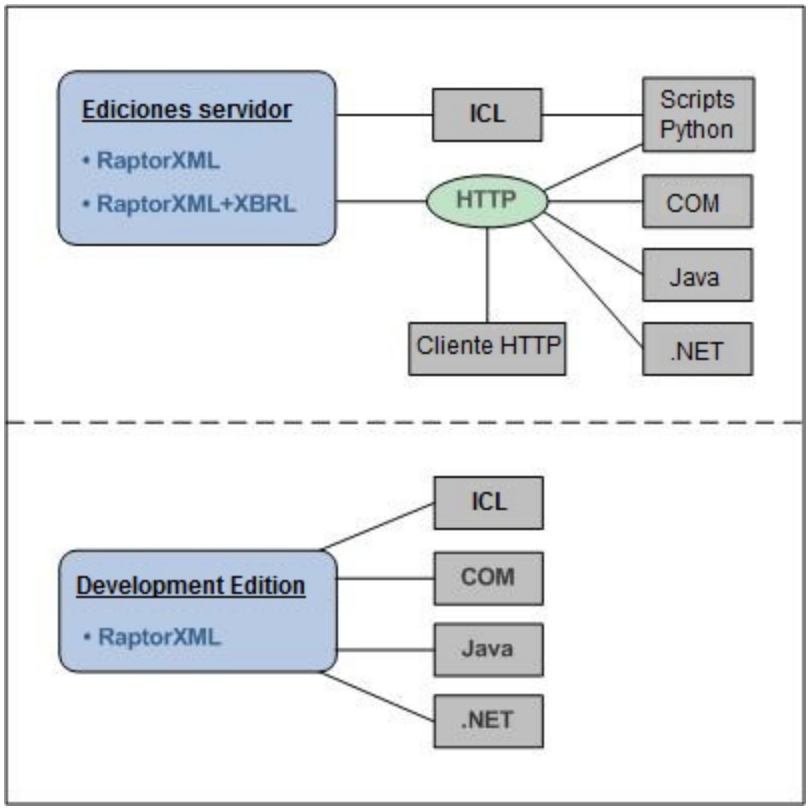
Altova ofrece tres ediciones distintas de RaptorXML:

- ***RaptorXML Server:*** un rapidísimo motor de procesamiento XML compatible con XML, XML Schema, XSLT, XPath y XQuery, entre otros estándares.
- ***RaptorXML+XBRL Server:*** ofrece todas las características de RaptorXML Server y funciones de procesamiento y validación compatibles con todos los estándares XBRL.
- ***RaptorXML Development Edition:*** disponible para sistemas Windows. Se puede descargar por separado desde el sitio web de Altova y se puede activar con la licencia de los productos de Altova MissionKit (XMLSpy, MapForce y StyleVision). Al igual que RaptorXML(+XBRL) Server, puede utilizarse para validaciones y transformaciones XML, XSLT y XQuery, pero sus funciones son limitadas en comparación con RaptorXML(+XBRL) Server. RaptorXML Development Edition:
  - Solamente se puede ejecutar una instancia de su binario al mismo tiempo en el mismo equipo.
  - Solamente es compatible con equipos Windows (no es compatible con Windows Server, Linux ni Mac OS X).
  - No es compatible con multi-threading ni procesamiento por lotes (procesamiento de múltiples archivos)
  - No es compatible con la representación de gráficos
  - No es compatible con XBRL
  - Solamente está disponible en una versión de 32 bits

Interfaces
Puede acceder a RaptorXML Development Edition a través de varias interfaces:

- Una **interfaz de la línea de comandos** (en adelante ***ILC***) *(en todas las ediciones)*
- Una **interfaz COM** para sistemas Windows *(en todas las ediciones)*
- Una **interfaz .NET** para sistemas Windows *(en todas las ediciones)*
- Una **interfaz Java** para sistemas Windows, Linux y Mac OS *(en todas las ediciones)*

El diagrama que aparece a continuación muestra cómo se accede a las dos ediciones servidor (RaptorXML Server y RaptorXML+XBRL Server) y a la edición limitada RaptorXML Development Edition a través de las diferentes interfaces.



Observe que las interfaces COM, Java y .NET usan el protocolo HTTP para conectarse a las ediciones servidor. Los scripts Pyton se pueden enviar a las ediciones servidor a través de la interfaz HTTP y de la línea de comandos.

Interfaz de la línea de comandos (ILC)
Permite validar XML (y otros documentos), transformar XSLT y ejecutar XQuery desde la línea de comandos. Para más información y aprender a usarla consulte la sección Interfaz de la línea de comandos.

Interfaz COM
Puede usar RaptorXML a través de la interfaz COM y, por tanto, puede ser utilizada por aplicaciones y lenguajes de script compatibles con COM. La compatibilidad con al interfaz COM está implementada para interfaces sin formato e interfaces de envío. Los datos de entrada se pueden suministrar como archivos o como cadenas de texto en scripts y en los datos de la aplicación.

Interfaz Java
Las funciones de RaptorXML también están disponibles como clases Java que se pueden usar en programas Java. Por ejemplo, hay clases Java que ofrecen características de validación XML, transformación XSLT y ejecución de XQuery.

Interfaz .NET
RaptorXML ofrece un archivo DLL construido como contenedor de RaptorXML que permite a los usuarios de .NET conectarse a las funciones de RaptorXML. Además, RaptorXML ofrece un ensamblado de interoperabilidad principal firmado por Altova. Los datos de entrada se

pueden suministrar como archivos o como cadenas de texto en scripts y en los datos de la aplicación.

1.2 Requisitos del sistema

RaptorXML es compatible con estos sistemas operativos:

- Windows Server 2008 R2 o superior
- Windows XP con Service Pack 3, Windows 7, Windows 8 o superior
- Linux (CentOS 6, RedHat 6, Debian 6 y Ubuntu 12.04 o superior)
- Mac OS X 10.7 o superior

RaptorXML(+XBRL) Server es compatible con equipos de 32 y 64 bits. Estos son los núcleos basados en conjuntos de instrucciones x86 y amd64 (x86-64): Intel Core i5, i7, XEON E5. RaptorXML Development Edition solamente está disponible en una versión de 32 bits.

Para usar RaptorXML a través de una interfaz COM el usuario debe tener privilegios para usar la interfaz COM, es decir, para registrar la aplicación y ejecutar las aplicaciones y scripts pertinentes.

1.3 Características

RaptorXML ofrece todas las funciones que aparecen a continuación. La mayoría de las funciones corresponden a la línea de comandos y a la interfaz COM. La principal diferencia es que la interfaz COM en Windows permite construir documentos a partir de cadenas de texto con el código de aplicación o de script (en lugar de hacer referencia a archivos XML, DTD, esquemas XML, XSLT o XQuery).

Validación XML

- Valida documentos XML con esquemas XML y DTD internos o externos.
- Revisa el formato de documentos XML, DTD, XML Schema, XSLT y XQuery.

Transformaciones XSLT

- Transforma XML usando documentos XSLT 1.0, 2.0 o 3.0 suministrados por el usuario.
- Los documentos XML y XSLT se pueden suministrar en forma de archivpo (por su URL) o, en el caso de la interfaz COM, en forma de cadena de texto.
- Los resultados se devuelven en forma de archivo (en la ubicación elegida por el usuario) o, en el caso de la interfaz COM, en forma de cadena de texto.
- Los parámetros XSLT se pueden suministrar a través de la línea de comandos o de la interfaz de COM.
- Las funciones de extensión de Altova, así como las funciones de extensión Java y .NET, permiten un procesamiento más especializado. Por ejemplo, permiten crear ciertas características como gráficos y códigos de barras en los documentos de salida.

Ejecución de XQuery

- Ejecuta documentos XQuery 1.0 y 3.0.
- Los documentos XQuery y XML se pueden suministrar en forma de archivo (por su URL) o, en el caso de la interfaz COM, en forma de cadena de texto.
- Los resultados se devuelven en forma de archivo (en la ubicación elegida por el usuario) o, en el caso de la interfaz COM, en forma de cadena de texto.
- Las variables XQuery externas se pueden suministrar a través de la línea de comandos o de la interfaz de COM.
- Opciones de serialización: codificación de salida, método de codificación (es decir, si el resultado es en XML, XHTML, HTML o texto), omisión de la declaración XML y sangría.

Características de alto rendimiento

- Optimizaciones de código de altísimo rendimiento
  - Implementaciones nativas de conjuntos de instrucciones
  - Versión de 32 y 64 bits
- Bajísima superficie de memoria
  - Representación en memoria de XML Information Set extremadamente compacta
  - Validación de instancias por transmisión por secuencias
- Características compatibles con múltiples plataformas
- Código altamente adaptable para informática en paralelo y equipos multi-CPU/multinúcleo
- Carga, validación y procesamiento en paralelo

Características para desarrolladores

- Avanzadas funciones de generación de informes de errores
- Modo servidor Windows y modo demonio Unix (a través de opciones de la línea de comandos)
- Intérprete Python 3.x para scripting
- API de COM en la plataforma Windows
- API de Java en todas las plataformas
- Funciones de extensión XPath, Java, .NET, XBRL, etc.
- Serialización de secuencias de datos
- Servidor HTTP integrado con API de validación REST

Para más información consulte el apartado Especificaciones compatibles y visite el sitio web de Altova.

1.4 Especificaciones compatibles

RaptorXML es compatible con todas estas especificaciones.

Recomendaciones del W3C
Sitio web: World Wide Web Consortium (W3C)

- Extensible Markup Language (XML) 1.0 (Fifth Edition)
- Extensible Markup Language (XML) 1.1 (Second Edition)
- Namespaces in XML 1.0 (Third Edition)
- Namespaces in XML 1.1 (Second Edition)
- XML Information Set (Second Edition)
- XML Base (Second Edition)
- XML Inclusions (XInclude) Version 1.0 (Second Edition)
- XML Linking Language (XLink) Version 1.0
- XML Schema Part 1: Structures Second Edition
- XML Schema Part 2: Datatypes Second Edition
- W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures
- W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes
- XPointer Framework
- XPointer xmlns() Scheme
- XPointer element() Scheme
- XML Path Language (XPath) Version 1.0
- XSL Transformations (XSLT) Version 1.0
- XML Path Language (XPath) 2.0 (Second Edition)
- XSL Transformations (XSLT) Version 2.0
- XQuery 1.0: An XML Query Language (Second Edition)
- XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)
- XML Path Language (XPath) 3.0

Borradores y recomendaciones candidatas del W3C
Sitio web: World Wide Web Consortium (W3C)

- XSL Transformations (XSLT) Version 3.0
- XQuery 3.0: An XML Query Language
- XPath and XQuery Functions and Operators 3.0

Estándares OASIS
Sitio web: OASIS Standards

- XML Catalogs V 1.1 - OASIS Standard V1.1

Altova RaptorXML Development Edition 2014

Instalar RaptorXML Development Edition

2 Instalar RaptorXML Development Edition

Esta sección describe el procedimiento para instalar y configurar RaptorXML Development Edition correctamente. Incluye varios apartados con información sobre:

- Cómo instalar RaptorXML y asignarle licencias en sistemas Windows
- Cómo usar los catálogos XML.
- Cómo trabajar con los recursos globales de Altova.

RaptorXML tiene opciones especiales que admiten el uso de catálogos XML y de recursos globales de Altova, características que mejoran la portabilidad y modularidad del entorno en el que se trabaja.

2.1 Instalación y configuración en Windows

Esta sección explica cómo instalar RaptorXML Development Edition y asignarle licencias en sistemas Windows.

Instalación en Windows

- Requisitos del sistema
- Instalación e instaladores
- Ubicación de la carpeta de aplicación

Asignación de licencias en Windows

2.1.1 Instalación en Windows

Temas de este apartado:

- Requisitos del sistema
- Instaladores e instalación
- Carpeta de la aplicación

Requisitos del sistema

- Windows Server 2003, 2008 R2 o superior
- Windows XP with Service Pack 3, Windows 7, Windows 8 o superior

---

Instaladores e instalación
Desde el sitio web de Altova puede descargar el instalador de RaptorXML Development Edition, que instala RaptorXML Development Edition y realiza los registros pertinentes. El ejecutable de RaptorXML Development Edition se guarda por defecto en esta carpeta:

```
<CarpetaArchivosPrograma>\Altova\RaptorXMLDevelopment2014\RaptorXMLDev.exe
```

El instalador realiza todos los registros necesarios para usar RaptorXML Development Edition desde la interfaz de COM, como interfaz Java y en el entorno .NET. Esto también incluye registrar el ejecutable deRaptorXML Development Edition como objeto servidor COM, instalar `RaptorXMLLib.dll` (para su uso en la interfaz Java) en el directorio `WINDIR\system32\` y añadir el archivo `Altova.RaptorXML.dll` a la biblioteca de referencia de .NET.

Ubicación de la carpeta de aplicación
La aplicación se instala en esta carpeta:

Windows XP	`C:\Archivos de programa\Altova\`
Windows Vista, Windows 7/8	`C:\Archivos de programa\Altova\`
Versión de 32 bits en un sistema operativo de 64 bits	`C:\Archivos de programa (x86)\Altova\`

2.1.2 Asignación de licencias en Windows

Cuando invoque el comando `RaptorXMLDev` por primera vez, aparece el cuadro de diálogo "Activación del software". Introduzca los datos de su licencia en este cuadro de diálogo y haga clic en **Guardar**. Si la licencia es válida, el diálogo desaparece y puede empezar a usar RaptorXML Development Edition.

RaptorXML Development Edition puede activarse (es decir, desbloquearse) con una licencia válida de Altova MissionKit (que incluye XMLSpy, StyleVision y MapForce) o de uno de estos tres productos independientes de Altova. Es decir, no necesita comprar una especial para RaptorXML Development Edition.

2.2 Catálogos XML

El mecanismo de catalogación XML permite recuperar archivos de carpetas locales, lo cual incrementa la velocidad global de procesamiento y mejora la portabilidad de los documentos (porque solo se tienen que cambiar los identificadores URI de los archivos de catálogo). Para más información consulte el apartado Cómo funcionan los catálogos.

Las herramientas XML de Altova usan un mecanismo de catalogación para acceder rápidamente a los archivos más utilizados, como esquemas XML y DTD. El usuario puede personalizar y ampliar este mecanismo de catalogación, que se describe en el apartado Mecanismo de catalogación XML de Altova. En el apartado Variables para ubicaciones del sistema se enumeran variables Windows para las ubicaciones más corrientes. Estas variables se pueden usar en los archivos de catálogo para encontrar las carpetas más utilizadas.

Esta sección se divide en varios apartados:

- Cómo funcionan los catálogos
- Mecanismo de catalogación XML de Altova
- Variables para ubicaciones de sistemas Windows

Para más información consulte la especificación XML Catalogs.

2.2.1 Cómo funcionan los catálogos

Temas de este apartado:

La función de los catálogos es redireccionar llamadas a recursos remotos a una URL local. Esto se consigue mediante asignaciones en el archivo de catálogo entre identificadores públicos o de sistemas, identificadores URI o partes de identificadores y la URL local correspondiente.

Asignaciones entre identificadores públicos y de sistema y URL locales
Durante la lectura de la declaración DOCTYPE de una DTD en un archivo XML, el identificador público o de sistema de la declaración encuentra el recurso necesario. Si el identificador selecciona un recurso remoto o si el identificador no es un localizador, entonces se puede asignar a un recurso local mediante una entrada en el catálogo.

Por ejemplo, este archivo SVG:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg>
  ...
</svg>
```

Su identificador público es: `-//W3C//DTD SVG 1.1//EN`
Su identificador de sistema es: `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`

En el catálogo se puede añadir una entrada para asignar el identificador público a una URL local. Por ejemplo:

```
<public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
```

O una entrada para asignar el identificador de sistema a una URL local. Por ejemplo:

```
<system systemId="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" uri="
schemas/svg/svg11.dtd"/>
```

Si se usa el identificador público o de sistema que aparece en el catálogo, entonces se usa la URL a la que está asignado. Las rutas relativas se resuelven con referencia a un atributo `xml:base` en el elemento de redirección del catálogo. La URL base de reserva es la URL del archivo de catálogo. Por el contrario, si no se usa el identificador público o de sistema que aparece en el catálogo, entonces se usará la URL del documento XML (en nuestro ejemplo sería la URL `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`).

Asignaciones entre rutas de archivo relativas/absolutas, direcciones URL o nombres y URL locales
El elemento **`uri`** se puede usar para asignar una ruta de archivo relativa/absoluta, una

dirección URL o un nombre a una URL local. Por ejemplo:

- `<uri name="doc.xslt" uri="C:\Docs\doc.xslt"/>`
- `<uri name="U:\Docs\2013\doc.xslt" uri="C:\Docs\doc.xslt"/>`
- `<uri name="http://www.altova.com/schemas/doc.xslt" uri="C:\Docs\doc.xslt"/>`
- `<uri name="foo" uri="C:\Docs\doc.xslt"/>`

Cuando se encuentra el valor de `name`, este se asigna al recurso especificado en el atributo `uri`. Con un catálogo distinto, el mismo nombre se podría asignar a un recurso diferente. Por ejemplo:

```
xsi:schemaLocation="http://www.altova.com/schemas/orgchart OrgChart.xsd"
```

Por lo general, la parte URI del valor del atributo (en negrita) es una ruta a la ubicación real del esquema. Sin embargo, si se hace referencia al esquema a través de un catálogo, no es necesario que la parte URI apunte a un esquema XML real, aunque el esquema debe existir para que el atributo `xsi:schemaLocation` siga siendo válido desde el punto de vista léxico. Por ejemplo, el valor `foo` sería suficiente para la parte URI del valor del atributo `xsi:schemaLocation` (en vez de `Orgchart.xsd`). El esquema está ubicado dentro del catálogo gracias a la parte de espacio de nombres del valor del atributo `xsi:schemaLocation`. En el ejemplo anterior la parte de espacio de nombres es `http://www.altova.com/schemas/orgchart`.

En el catálogo la entrada siguiente encontraría el esquema por la parte de espacio de nombres.

```
<uri name="http://www.altova.com/schemas/orgchart" uri="
C:\MySchemas\OrgChart.xsd"/>
```

Para más información sobre estos elementos consulte la especificación XML Catalogs.

2.2.2 Mecanismo de catalogación XML de Altova

Temas de este apartado:

- El archivo de catálogo raíz `RootCatalog.xml` contiene los archivos de catálogo en los que busca RaptorXML.
- Los archivos catálogo de extensión `CoreCatalog.xml`, `CustomCatalog.xml` y `Catalog.xml`.
- Subconjunto de catálogos compatible.

RootCatalog.xml
RaptorXML busca por defecto en el archivo `RootCatalog.xml` (*ver más abajo*) la lista de archivos de catálogo que debe usar. El catálogo raíz `RootCatalog.xml` está en esta carpeta:

```
<CarpetaArchivosPrograma>\Altova\RaptorXMLDevelopment2014
```

Para usar otro archivo como catálogo raíz, utilice la opción `--catalog` de la línea de comandos, el método `setCatalog` de la interfaz Java o el método `Catalog` de la interfaz COM.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         xmlns:spy="http://www.altova.com/catalog_ext"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog
Catalog.xsd">

  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/
CustomCatalog.xml"/>
  <nextCatalog catalog="CoreCatalog.xml"/>

  <!-- Incluir todos los catálogos situados en la carpeta Schemas del primer
nivel de directorios -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas" catalog="
catalog.xml" spy:depth="1"/>

  <!-- Incluir todos los catálogos situados en la carpeta XBRL del primer
nivel de directorios -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL" catalog="
catalog.xml" spy:depth="1"/>
</catalog>
```

El resto de archivos de catálogo se enumeran dentro de un elemento `nextCatalog` y puede añadir tantos archivos como quiera. RaptorXML busca en todos los archivos de catálogo y resuelve las asignaciones que hay en ellos.

En el fragmento de código anterior puede observar una referencia directa a dos catálogos: `CoreCatalog.xml` y `CustomCatalog.xml`. Además se hace referencia a los catálogos llamados `catalog.xml` que están en el primer nivel de subcarpetas de las carpetas `Schemas` y `XBRL`. (El valor de la variable `%AltovaCommonFolder%` se explica en el apartado Variables para ubicaciones de sistema.)

Los archivos de catálogo de `Altova Common Folder` asignan los identificadores públicos y de sistema predefinidos de los esquemas más utilizados (como XML Schema y XHTML) a identificadores URI que apuntan a las copias locales de los esquemas correspondientes. Estos

esquemas se instalan en la carpeta `Altova Common Folder` durante la instalación de RaptorXML.

CoreCatalog.xml, CustomCatalog.xml y Catalog.xml
Los archivos de catálogo `CoreCatalog.xml` y `CustomCatalog.xml` se enumeran en `RootCatalog.xml`:

- `CoreCatalog.xml` contiene ciertas asignaciones propias de Altova necesarias para encontrar esquemas en la carpeta `Altova Common Folder`.
- `CustomCatalog.xml` es un archivo esqueleto donde puede crear sus propias asignaciones. En `CustomCatalog.xml` puede crear asignaciones para cualquier esquema que necesite y que no esté en los archivos de catálogo de la carpeta `Altova Common Folder`. Para ello debe utilizar elementos compatibles del mecanismo de catalogación OASIS (*ver más abajo*).
- Hay varios archivos `Catalog.xml` dentro de las carpetas de esquemas o taxonomías XBRL de la carpeta `Altova Common Folder` y cada uno de estos archivos asigna identificadores públicos/de sistema a identificadores URI que apuntan a copias locales de los esquemas correspondientes.

Tanto `CoreCatalog.xml` como `CustomCatalog.xml` están en la carpeta `<`*`CarpetaArchivosPrograma`*`>\Altova\RaptorXMLDevelopment2014\etc`. Cada archivo `catalog.xml` está en una carpeta de esquema y estas carpetas de esquema están a su vez dentro de las carpetas `%AltovaCommonFolder%\Schemas` y `%AltovaCommonFolder%\XBRL`.

Subconjunto de catálogos compatible
Cuando cree entradas en un archivo de catálogo utilizado por RaptorXML, solamente debería usar los elementos de la especificación OASIS que aparecen a continuación. Consulte la especificación XML Catalogs para obtener más información.

- `<public publicId="IDPúblico del Recurso" uri="URL del archivo local"/>`
- `<system systemId="IDdeSistema del Recurso" uri="URL del archivo local "/>`
- `<uri name="nombreArchivo" uri="URL del archivo identificado por el nombre de archivo"/>`
- `<rewriteURI uriStartString="InicioDeCadena del URI que se debe rescribir" rewritePrefix="Cadena que debe sustituir a InicioDeCadena"/>`
- `<rewriteSystem systemIdStartString="InicioDeCadena del IDdeSistema" rewritePrefix="Cadena de sustitución para encontrar el recurso localmente"/>`

Cuando no exista un identificador público, puede asignar el identificador de sistema directamente a una URL con ayuda del elemento `system`. También puede asignar un URI a otro URI usando el elemento `uri`. Los elementos `rewriteURI` y `rewriteSystem` sirven para rescribir la parte inicial de un URI o de un identificador de sistema, respectivamente. Esto permite reemplazar el inicio de una ruta de archivo y, por tanto, apuntar a otro directorio.

Nota: todos los elementos pueden tomar el atributo `xml:base`, que se usa para especificar el URI base del elemento. Si no hay ningún elemento con `xml:base`, el URI base será el URI del archivo de catálogo.

Para obtener más información sobre estos elementos consulte la especificación XML Catalogs.

2.2.3 Variables para ubicaciones de sistemas Windows

En los archivos de catálogo puede usar variables de entorno Shell para señalar la ruta de acceso a varias ubicaciones del sistema Windows. Estas son las variables de entorno Shell compatibles:

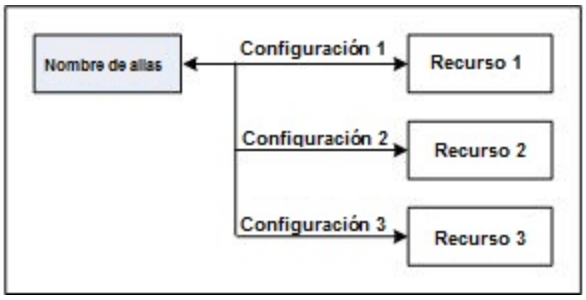
%AltovaCommonFolder%	C:\Archivos de programa\Altova\Common2014
%DesktopFolder%	Ruta de acceso completa de la carpeta Escritorio del usuario actual.
%ProgramMenuFolder%	Ruta de acceso completa de la carpeta del menú **Programas** del usuario actual.
%StartMenuFolder%	Ruta de acceso completa de la carpeta del menú **Inicio** del usuario actual.
%StartUpFolder%	Ruta de acceso completa de la carpeta **Inicio** del usuario actual.
%TemplateFolder%	Ruta de acceso completa de la carpeta de plantillas del usuario actual.
%AdminToolsFolder%	Ruta de acceso completa del directorio del sistema de archivos que almacena las herramientas administrativas del usuario actual.
%AppDataFolder%	Ruta de acceso completa de la carpeta Datos de programa del usuario actual.
%CommonAppDataFolder%	Ruta de acceso completa del directorio de archivos que contiene datos del programa de todos los usuarios.
%FavoritesFolder%	Ruta de acceso completa de la carpeta Favoritos del usuario actual.
%PersonalFolder%	Ruta de acceso completa de la carpeta personal del usuario actual.
%SendToFolder%	Ruta de acceso completa de la carpeta SendTo del usuario actual.
%FontsFolder%	Ruta de acceso completa de la carpeta Fuentes del sistema.
%ProgramFilesFolder%	Ruta de acceso completa de la carpeta Archivos de programa del usuario actual.
%CommonFilesFolder%	Ruta de acceso completa de la carpeta Common files del usuario actual.
%WindowsFolder%	Ruta de acceso completa de la carpeta Windows del usuario actual.
%SystemFolder%	Ruta de acceso completa de la carpeta System del usuario actual.
%LocalAppDataFolder%	Ruta de acceso completa al directorio del sistema de archivos que sirve como repositorio de datos para aplicaciones locales (no roaming).
%MyPicturesFolder%	Ruta de acceso completa a la carpeta Mis imágenes.

2.3 Recursos globales

Temas de este apartado:

- Recursos globales: ¿qué son?
- Recursos globales: ¿cómo se usan?

¿Qué son los recursos globales?
Un archivo de recurso global de Altova asigna un alias a varios recursos mediante configuraciones diferentes, tal y como muestra el diagrama que aparece a continuación. La idea es poder cambiar de alias para acceder a recursos distintos, dependiendo de la configuración elegida.



Los recursos globales se definen desde las herramientas de Altova (como Altova XMLSpy, por ejemplo) y se guardan en un archivo XML de recursos globales. RaptorXML puede usar estos recursos globales como datos de entrada. Para ello necesita el nombre y la ubicación del archivo de recursos globales, así como el alias y la configuración que debe usar.

La ventaja de usar recursos globales es que puede cambiar de recurso con solo cambiar el nombre de la configuración. En RaptorXML, esto significa que al usar un valor diferente de la opción `--globalresourcesconfig | --gc`, se puede usar un recurso global distinto (*ver ejemplo que aparece más abajo*).

¿Cómo se utilizan los recursos globales con RaptorXML?
Para especificar el uso de un recurso global como entrada para un comando de RaptorXML es obligatorio usar estos parámetros en la interfaz de la línea de comandos:

- El archivo XML de recursos globales (opción `--globalresourcesfile | --gr`)
- La configuración necesaria (opción `--globalresourcesconfig | --gc`)
- El alias, que se puede especificar directamente en la ILC cuando sea necesario un nombre de archivo. También puede estar dentro del archivo XML en el que RaptorXML busca un nombre de archivo (como en un atributo `xsi:schemaLocation`, por ejemplo).

Por ejemplo, si quiere transformar `entrada.xml` con `transform.xslt` en `salida.html`, lo normal sería usar estos comandos en la ILC usando los nombres de archivo:

```
raptorxmldev xslt --input=entrada.xml --output=salida.html transform.xslt
```

No obstante, si tiene una definición de recurso global para el alias `MiEntrada` que apunta al recurso de archivo `PrimeraEntrada.xml` por medio de una configuración llamada `PrimeraConfig`, podría usar el alias `MiEntrada` en la línea de comandos:

```
raptorxmldev xslt --input=altova://file_resource/MiEntrada
--gr=C:\MisRecursosGlobales.xml --gc=PrimeraConfig --output=Salida.html
transform.xslt
```

Ahora imagine que tiene otro recurso de archivo, por ejemplo `SegundaEntrada.xml`, que apunta al alias `MiEntrada` por medio de una configuración llamada `SegundaConfig`, entonces puede usar este otro recurso con solo cambiar la opción `--gc` del comando anterior:

```
raptorxmldev xslt --input=altova://file_resource/MiEntrada
--gr=C:\MisRecursosGlobales.xml --gc=SegundaConfig --output=Salida.html
transform.xslt
```

Nota: en el ejemplo anterior se usó un recurso de archivo. Los recursos de archivo deben llevar el prefijo **`altova://file_resource/`**. También puede usar recursos globales que sean carpetas. Para identificar un recurso de carpeta, utilice el prefijo: **`altova://folder_resource/NombreAlias`**. No olvide que en la interfaz de la línea de comandos puede usar recursos de carpeta como parte de la ruta de acceso. Por ejemplo: **`altova://folder_resource/NombreAlias/entrada.xml`**.

Altova RaptorXML Development Edition 2014

Interfaz de la línea de comandos (ILC)

3 Interfaz de la línea de comandos (ILC)

El ejecutable de RaptorXML que sirve para trabajar con la interfaz de la línea de comandos (ILC) se instala por defecto en esta carpeta:

```
<CarpetaArchivosPrograma>\Altova\RaptorXMLDevelopment2014\bin\raptorxmldev
.exe
```

Uso
Esta es la sintaxis de la línea de comandos:

```
raptorxmldev --h | --help | --version | <comando> [opciones] [argumentos]
```

raptorxmldev	Llama a la aplicación
--h | --help	Muestra el texto de ayuda
--version	Muestra el número de versión de la aplicación
<comando>	Comando que se debe ejecutar (*ver la lista que aparece más abajo*). Cada comando (con sus argumentos y opciones respectivos) se describe y explica en los apartados de esta sección.
[opciones]	Opciones de un comando. Cada apartado de esta sección describe un comando y sus opciones. Además, el apartado Opciones enumera todas las opciones de la línea de comandos.
[argumentos]	Argumentos de un comando. Cada apartado de esta sección describe un comando y sus argumentos.

Comandos de la ILC
A continuación enumeramos todos los comandos de la ILC de RaptorXML ordenados según su función (algunos aparecen en más de una categoría). Estos comandos se describen uno a uno en los apartados de esta sección, junto con sus opciones y argumentos.

Comandos de validación

valdtd | dtd	Valida un documento DTD.
valxml-withdtd | xml	Valida un documento XML con una DTD.
valxml-withxsd | xsi	Valida un documento XML con un esquema XML.
valxquery	Valida un documento XQuery.
valxsd | xsd	Valida un esquema XML del W3C.
valxslt	Valida un documento XSLT.
valany	Valida cualquier tipo de documento de los mencionados anteriormente. El tipo de documento se detecta de forma automática.

Comandos para comprobar el formato XML

wfxml	Comprueba si un documento XML tiene un formato correcto.
wfdtd	Comprueba si un documento DTD tiene un formato correcto.
wfany	Comprueba si el documento (ya sea XML o DTD) tiene un formato correcto.

Comandos XSLT

xslt	Realiza una transformación con un archivo XSLT dado.
valxslt	Valida un documento XSLT.

Comandos XQuery

xquery	Ejecuta un XQuery con un archivo XQuery dado.
valxquery	Valida un documento XQuery.

3.1 Comandos para validar XML, DTD, XSD

Los comandos de validación XML sirven para validar este tipo de documentos:

- *XML:* los documentos de instancia XML se validan con una DTD (valxml-withdtd | xml) o con un esquema XML 1.0/1.1 (valxml-withxsd | xsi).
- *DTD:* los DTD se revisan para ver si su formato correcto y confirmar que no tienen errores (valdtd | dtd).
- *XSD:* los esquemas XML del W3C (XSD) se validan según las reglas de la especificación XML Schema (valxsd | xsd).

valxml-withdtd | xml	Valida un documento de instancia XML con una DTD.
valxml-withxsd | xsi	Valida un documento de instancia XML con un esquema XML.
valdtd | dtd	Valida un documento DTD.
valxsd | xsd	Valida un documento de esquema XML del W3C (XSD).
valany	Valida cualquier documento, ya sea XML, DTD o XSD. Este comando también se usa para validar documentos XBRL (de instancia o taxonomía), XSLT y XQuery. El tipo de documento suministrado se detecta automáticamente.

Nota: también se pueden validar instancias XBRL, taxonomías XBRL, documentos XSLT y documentos XQuery. Estos comandos de validación se describen en estas secciones: Comandos de validación XBRL, Comandos XSLT y Comandos XQuery.

3.1.1 valxml-withdtd (xml)

El comando **valxml-withdtd | xml** valida un documento XML de instancia con una DTD.

```
raptorxmldev valxml-withdtd | xml [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* es el documento XML que debe validarse. Si existe una referencia a una DTD en el documento, no es necesario usar la opción `--dtd`.

Ejemplos

- **raptorxmldev** valxml-withdtd --dtd=c:\MiDTD.dtd c:\Test.xml
- **raptorxmldev** xml c:\Test.xml
- **raptorxmldev** xml --verbose=true c:\Test.xml

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones del comando valxml-withdtd

--dtd=*ARCHIVO*
--namespaces=true|false

Opciones de procesamiento

--streaming=true|false

Opciones de catalogación

--catalog=*ARCHIVO*
--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false
--gr, --globalresourcefile=*ARCHIVO*
--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado
--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*
--verbose=true|false

Opciones de ayuda y versión

--h, --help

--version

3.1.2 valxml-withxsd (xsi)

El comando **valxml-withxsd | xsi** valida un documento XML de instancia con las especificaciones XML Schema Definition Language (XSD) 1.0 y 1.1 del W3C.

```
raptorxmldev valxml-withxsd | xsi [opciones] ArchivoEntrada
```

El argumento **ArchivoEntrada** suministra el documento XML que debe validarse. La opción --schemalocation-hints=true|false indica si la referencia XSD del documento XML debe utilizarse o no, siendo `true` su valor predeterminado (es decir, se utiliza). La opción --xsd=*ARCHIVO* especifica qué esquema se utiliza.

Nota: si usa la opción `--script` para ejecutar scripts Python, no olvide especificar la opción `--streaming=false`.

Ejemplos

- **raptorxmldev** `valxml-withxsd --schemalocation-hints=false --xsd=c:\MiXSD.xsd c:\SinRefXSD.xml`
- **raptorxmldev** `xsi c:\SinRefXSD.xml`

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones del comando valxml-withxsd

--assessment-mode=skip|lax|strict

Opciones de XML Schema

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|
prefer-namespace

--xsd=*ARCHIVO*

--xsd-version=1.0|1.1|detect

Opciones para instancias XML

--xinclude=true|false

--xml-mode=wf|id|valid

Opciones de procesamiento

--streaming=true|false
--script=*ARCHIVO*

Opciones de catalogación

--catalog=*ARCHIVO*
--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false
--gr, --globalresourcefile=*ARCHIVO*
--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado
--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*
--verbose=true|false

Opciones de ayuda y versión

--h, --help
--version

3.1.3 valdtd (dtd)

El comando **valdtd | dtd** valida un documento DTD con la especificación XML 1.0 o XML 1.1.

```
raptorxmldev valdtd | dtd [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* es el documento DTD que debe validarse.

Ejemplos

- **raptorxmldev** valdtd c:\Test.dtd
- **raptorxmldev** dtd --verbose=true c:\Test.dtd

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

--gr, --globalresourcefile=*ARCHIVO*

--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*

--verbose=true|false

Opciones de ayuda y versión

--h, --help

--version

3.1.4 valxsd (xsd)

El comando **valxsd | xsd** valida un documento de esquema XML (documento XSD) con la especificación XML Schema Definition Language (XSD) 1.0 o 1.1 del W3C. Tenga en cuenta que lo que se valida con la especificación XML Schema es un esquema XML y no un documento XML de instancia con un esquema XML.

```
raptorxmldev valxsd | xsd [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* suministra el documento de esquema XML que debe validarse. La opción --xsd-version=1.0|1.1|detect indica la versión XSD con la que debe validarse, siendo `1.0` su valor predeterminado.

Ejemplos

- **raptorxmldev** valxsd c:\Test.xsd
- **raptorxmldev** xsd --verbose=true c:\Test.xsd

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones de XML Schema

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|
prefer-namespace

--xsd-version=1.0|1.1|detect

Opciones de procesamiento

--script=*ARCHIVO*

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

3.1.5 valany

El comando **valany** valida un documento XML, DTD o esquema XML con sus especificaciones correspondientes. El tipo de documento se detecta automáticamente.

```
raptorxmldev valany [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* suministra el documento que debe validarse. Recuerde que como argumento del comando se puede suministrar solamente un documento. El tipo de documento se detecta automáticamente.

Ejemplos

- **raptorxmldev** valany c:\Test.xml
- **raptorxmldev** valany --errorformat=text c:\Test.xml

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones de XML Schema

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|
prefer-namespace

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

--gr, --globalresourcefile=*ARCHIVO*

--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*

--verbose=true|false

Opciones de ayuda y versión

--h, --help

--version

3.2 Comandos para comprobar el formato

Los comandos de comprobación de formato sirven para comprobar si el formato de documentos XML y DTD es correcto.

wfxml	Comprueba si el documento XML tiene un formato correcto.
wfdtd	Comprueba si el documento DTD tiene un formato correcto.
wfany	Comprueba si el documento XML o DTD tiene un formato correcto. El tipo se detecta automáticamente.

3.2.1 wfxml

El comando **wfxml** revisa un documento XML y comprueba si su formato es correcto según la especificación XML 1.0 o XML 1.1.

```
raptorxmldev wfxml [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* es el documento XML cuyo formato debe comprobarse.

Ejemplos

- **raptorxmldev** wfxml c:\Test.xml
- **raptorxmldev** wfxml --verbose=true c:\Test.xml

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones de procesamiento

--streaming=true|false
--dtd=*ARCHIVO*
--namespaces=true|false

Opciones de catalogación

--catalog=*ARCHIVO*
--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false
--gr, --globalresourcefile=*ARCHIVO*
--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado
--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*
--verbose=true|false

Opciones de ayuda y versión

--h, --help
--version

3.2.2 wfdtd

El comando **wfdtd** revisa un documento DTD y comprueba si su formato es correcto según la especificación XML 1.0 o XML 1.1.

```
raptorxmldev wfdtd [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* es el documento DTD cuyo formato debe comprobarse.

Ejemplos

- **raptorxmldev** wfdtd c:\Test.dtd
- **raptorxmldev** wfdtd --verbose=true c:\Test.dtd

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

--gr, --globalresourcefile=*ARCHIVO*

--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*

--verbose=true|false

Opciones de ayuda y versión

--h, --help

--version

3.2.3 wfany

El comando **wfany** comprueba si el formato de un documento XML o DTD es correcto según la especificación correspondiente. El tipo de documento se detecta automáticamente.

```
raptorxmldev wfany [opciones] ArchivoEntrada
```

El argumento *ArchivoEntrada* es el documento cuyo formato debe comprobarse. Recuerde que como argumento del comando se puede suministrar solamente un documento. El tipo de documento se detecta automáticamente.

Ejemplos

- **raptorxmldev** wfany c:\Test.xml
- **raptorxmldev** wfany --errorformat=text c:\Test.xml

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

--gr, --globalresourcefile=*ARCHIVO*

--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

--error-format=text|shortxml|longxml

Opciones para mensajes

--log-output=*ARCHIVO*

--verbose=true|false

Opciones de ayuda y versión

--h, --help

--version

3.3 Comandos XSLT

Estos son los comandos XSLT:

- xslt: para transformar documentos XML con un documento XSLT
- valxslt: para validar documentos XSLT

Los argumentos y opciones de cada comando se describen en los siguientes apartados.

3.3.1 xslt

El comando **xslt** toma un archivo XSLT como único argumento y lo utiliza para transformar un archivo XML de entrada y generar un archivo de salida. Los archivos de entrada y salida se especifican como opciones.

```
raptorxmldev xslt [opciones] Archivo-XSLT
```

El argumento *Archivo-XSLT* es la ruta de acceso y el nombre del archivo XSLT que se debe usar para la transformación. Se necesita un archivo XML de entrada (--input) o el punto de entrada de una plantilla con nombre (--template-entry-point). Si no especifica la opción del documento de salida --output, se genera un resultado estándar. Puede usar XSLT 1.0, 2.0 o 3.0. La versión predeterminada es XSLT 3.0.

Ejemplos

- **raptorxmldev** xslt --input=c:\Test.xml --output=c:\Salida.xml c:\Test.xslt
- **raptorxmldev** xslt --template-entry-point=PlantillaInicial --output=c:\Salida.xml c:\Test.xslt
- **raptorxmldev** xslt --input=c:\Test.xml --output=c:\Salida.xml --param date=//node/@att1 --p=title:'cadenasinespacios' --param=title:"'cadena con espacios'" --p=amount:456 c:\Test.xslt

---

Opciones

Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones propias del comando xslt

--indent-characters=*VALOR*

--input=*ARCHIVO*

--output=*ARCHIVO*

--p, --param=*CLAVE:VALOR*

--streaming-serialization-enabled=true|false

Opciones compartidas por los comandos xslt y valxslt

--chartext-disable=true|false

--dotnetext-disable=true|false

--javaext-barcode-location=*ARCHIVO*

--javaext-disable=true|false

--template-entry-point=*VALOR*

--template-mode=*VALOR*

--xslt-version=1|2|3

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

--gr, --globalresourcefile=*ARCHIVO*

--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

Opciones para mensajes

--verbose=true|false

Opciones para instancias XML

--xinclude=true|false

--xml-mode=wf|id|valid

Opciones de XML Schema

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|
prefer-namespace

--xsd-version=1.0|1.1|detect

Opciones de ayuda y versión

--h, --help

--version

3.3.2 valxslt

El comando **valxslt** toma un archivo XSLT como único argumento y lo valida.

```
raptorxmldev valxslt [opciones] Archivo-XSLT
```

El argumento *Archivo-XSLT* es la ruta de acceso y el nombre del archivo XSLT que debe validarse. El archivo se valida con la especificación XSLT 1.0, 2.0 o 3.0. La versión predeterminada es XSLT 3.0.

Ejemplos

- **raptorxmldev** valxslt c:\Test.xslt
- **raptorxmldev** valxslt --xslt-version=2 c:\Test.xslt

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

Opciones compartidas por los comandos xslt y valxslt

--chartext-disable=true|false
--dotnetext-disable=true|false
--javaext-barcode-location=*ARCHIVO*
--javaext-disable=true|false
--template-entry-point=*VALOR*
--template-mode=*VALOR*
--xslt-version=1|2|3

Opciones de catalogación

--catalog=*ARCHIVO*
--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false
--gr, --globalresourcefile=*ARCHIVO*
--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

Opciones para mensajes

--verbose=true|false

Opciones para instancias XML

--xinclude=true|false

--xml-mode=wf|id|valid

Opciones de XML Schema

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|
prefer-namespace

--xsd-version=1.0|1.1|detect

Opciones de ayuda y versión

--h, --help

--version

3.4 Comandos XQuery

Estos son los comandos XQuery:

- xquery: para ejecutar documentos XQuery y, si se quiere, con un documento de entrada
- valxquery: para validar documentos XQuery

Los argumentos y opciones de cada comando se describen en los siguientes apartados.

3.4.1 xquery

El comando **xquery** toma un archivo XQuery como único argumento y lo ejecuta con un archivo de entrada opcional para generar un archivo de salida. Los archivos de entrada y salida se especifican como opciones.

```
raptorxmldev xquery [opciones] Archivo-XQuery
```

El argumento *Archivo-XQuery* es la ruta de acceso y el nombre del archivo XQuery que debe ejecutarse. Puede usar tanto XQuery 1.0 como 3.0, pero la versión predeterminada es 3.0.

Ejemplos

- **raptorxmldev** xquery --output=c:\Salida.xml c:\TestConsulta.xq
- **raptorxmldev** xquery --input=c:\Entrada.xml --output=c:\Salida.xml --var company=Altova -var date=2006-01-01 c:\TestConsulta.xq
- **raptorxmldev** xquery --input=c:\Entrada.xml --output=c:\Salida.xml --xparam source=" doc('c:\test\books.xml')//book "
- **raptorxmldev** xquery --output=c:\Salida.xml --omit-xml-declaration=false --output-encoding=ASCII c:\TestConsulta.xq

---

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

***Opciones propias del comando* xquery**

--indent-characters=*VALOR*
--input=*ARCHIVO*
--output=*ARCHIVO*
--output-encoding=*VALOR*
--output-indent=true|false
--output-method=xml|html|xhtml|text
--p, --param=*CLAVE:VALOR*

***Opciones compartidas por los comandos* xquery *y* valxquery**

--omit-xml-declaration=true|false
--xquery-version=1|3

Opciones de catalogación

--catalog=*ARCHIVO*
--user-catalog=*ARCHIVO*

Opciones para recursos globales

3.4.2 valxquery

El comando **valxquery** toma un archivo XQuery como único argumento y lo valida.

```
raptorxmldev valxquery [opciones] Archivo-XQuery
```

El argumento *Archivo-XQuery* es la ruta de acceso y el nombre del archivo XQuery que debe validarse.

Ejemplos

- **raptorxmldev** valxquery c:\Test.xquery
- **raptorxmldev** valxquery --xquery-version=1 c:\Test.xquery

---

Opciones
Haga clic en una opción o en el título de su categoría para leer su descripción.

- El valor booleano de la opción se establece en `true` si no se especifica ningún valor para la opción.
- El valor de la opción se puede especificar sin comillas en todos los casos excepto en dos: (i) cuando la cadena de valor contiene espacios o (ii) cuando en la descripción de la opción se especifica que las comillas son necesarias.

***Opciones compartidas por los comandos* xquery *y* valxquery**

--omit-xml-declaration=true|false

--xquery-version=1|3

Opciones de catalogación

--catalog=*ARCHIVO*

--user-catalog=*ARCHIVO*

Opciones para recursos globales

--enable-globalresources=true|false

--gr, --globalresourcefile=*ARCHIVO*

--gc, --globalresourceconfig=*VALOR*

Opciones para errores

--error-limit=*N*|ilimitado

Opciones para mensajes

--verbose=true|false

Opciones para instancias XML

--xinclude=true|false

--xml-mode=wf|id|valid

Opciones de XML Schema

--xsd-version=1.0|1.1|detect

Opciones de ayuda y versión

--h, --help

--version

3.5 Comandos de ayuda y licencias

En esta sección describimos dos características importantes de RaptorXML Development Edition:

- Ayuda: cómo obtener información sobre los comandos disponibles o sobre los argumentos y opciones de un comando.
- Licencias: cómo asignar licencias a RaptorXML.

3.5.1 Ayuda

El comando **help** toma un solo argumento: el nombre del comando para el que desea obtener información adicional. Muestra la sintaxis del comando y otra información importante para poder ejecutar el comando correctamente.

```
raptorxmldev help Comando
```

Nota: si ejecuta el comando **help** sin ningún argumento, aparece la ayuda sobre todos los comandos de la ILC, cada uno con una breve descripción.

Ejemplo

```
raptorxmldev help valany
```

Este comando contiene un argumento, el comando `valany`. Cuando se ejecuta este comando, aparece información de ayuda sobre el comando `valany`.

La opción `--help`
También puede ver la información de ayuda de un comando si usa la opción `--help` con dicho comando. Por ejemplo, si usamos la opción `--help` con el comando `valany`:

```
raptorxmldev valany --help
```

conseguimos el mismo resultado que cuando usamos el comando `help` con el argumento `valany`:

```
raptorxmldev help valany
```

En ambos casos aparece información de ayuda sobre el comando `valany`.

3.5.2 Licencias

Cuando se invoca un comando de RaptorXML Development Edition por primera vez, aparece el cuadro de diálogo de activación del software. Introduzca los datos de su licencia en este cuadro de diálogo y haga clic en **Guardar**. Si la licencia es válida, el cuadro de diálogo desaparece y puede empezar a usar RaptorXML Development Edition.

La licencia de su Altova MissionKit o de su producto independiente de Altova (XMLSpy, StyleVision o MapForce) sirve para trabajar con RaptorXML Development Edition. Por tanto, al introducir la licencia de estos productos en el cuadro de diálogo, se activa (se desbloquea) RaptorXML Development Edition. En otras palabras, para trabajar con RaptorXML Development Edition no necesita una licencia especial.

3.6 Opciones

En esta sección describimos todas las opciones de la ILC, organizadass por funciones. Para ver qué opción debe usar con cada comando, consulte la descripción del comando correspondiente.

- Catálogos
- Errores
- Recursos globales
- Ayuda y versión
- Mensajes
- Procesamiento
- Documento XML
- Validación XML
- Esquemas XML (XSD)
- XQuery
- XSLT
- Archivos ZIP

3.6.1 Catálogos

[--catalog, --user-catalog]

--catalog=*ARCHIVO*
Especifica la ruta de acceso absoluta a un archivo de catálogo que no está en el archivo de catálogo raíz instalado. El valor predeterminado es la ruta de acceso absoluta del archivo de catálogo raíz instalado.

--user-catalog=*ARCHIVO*
Especifica la ruta de acceso absoluta a un catálogo XML que debe utilizarse junto con el catálogo raíz.

3.6.2 Errores

[--error-limit, --error-format]

--error-limit=*N*|ilimitado
Especifica el límite de errores. Esta opción es útil para limitar el uso del procesador durante la validación. Cuando se alcanza el límite de error, se detiene la validación.
Valor predeterminado: `100`.

--error-format=text|shortxml|longxml
Especifica el formato de la salida de error. Los valores posibles son formatos de texto, XML y XML detallado (`longxml`).
Valor predeterminado: `text`.

3.6.3 Recursos globales

[--enable-global-resources, --globalresourcefile, --globalresourceconfig]

--enable-globalresources=true|false
Habilita la función de recursos globales.
Valor predeterminado: `false`.

--gr, --globalresourcefile=*ARCHIVO*
Especifica el archivo de recursos globales (y habilita los recursos globales).

--gc, --globalresourceconfig=*VALOR*
Especifica la configuración activa del recurso global (y habilita los recursos globales).

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.4 Ayuda y versión

[--help, --version]

--h, --help
Muestra el texto de ayuda para el comando. Por ejemplo `valany --h`. (Otra opción es usar el comando `help` con un argumento. Por ejemplo: `help valany`.)

--version
Muestra el número de versión de RaptorXML. Si se utiliza con un comando, escriba la opción `--version` antes del comando.

3.6.5 Mensajes

[--log-output, --verbose]

--log-output=*ARCHIVO*
Escribe el mensaje de salida en la URL de archivo indicada, en lugar de en la consola. Compruebe que la ILC tiene permiso de escritura en la ubicación de destino.

--verbose=true|false
Si el valor es `true`, RaptorXML genera información adicional durante la validación. Valor predeterminado es `false`.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.6 Procesamiento

[--listfile, --script, --streaming-serialization-enabled, --streaming]

--listfile=true|false
Si el valor es `true`, el argumento *ArchivoEntrada* del comando se entiende como un archivo de texto que contiene un nombre de archivo por línea. Otra opción es enumerar los archivos en la ILC, separados por un espacio. No obstante, recuerde que las ILC tienen un límite de caracteres. Además, no olvide que la opción `--listfile` solamente afecta a los argumentos y no a las opciones.
Valor predeterminado: `false`

--script=*ARCHIVO*
Una vez finalizada la validación, ejecuta el script Python.

--streaming=true|false
Habilita la validación de transmisión por secuencias. En el modo de transmisión por secuencias, el almacenamiento de datos en memoria se reduce al mínimo y el procesamiento es más rápido. El inconveniente es que puede que no esté disponible cierta información que podría necesitar más adelante, como el modelo de datos del documento XML, por ejemplo. Si quiere evitar esto, debería deshabilitar el modo de transmisión por secuencias (dándole el valor `false` a la opción `--streaming`). Cuando use la opción `--script` con el comando `valxml-withxsd`, aconsejamos deshabilitar la transmisión por secuencias.
Valor predeterminado: `true`.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.7 Instancias XML

[--xinclude, --xml-mode]

--xinclude=true|false
Habilita la compatibilidad con inclusiones XML (XInclude). Si el valor es `false`, los elementos XInclude `include` se ignoran.
Valor predeterminado: `false`.

--xml-mode=wf|id|valid
Especifica el modo de procesamiento XML que debe utilizarse: `wf`=comprobación de formato ; `id`=comprobación de formato con ID/IDREF; `valid`=validación.
Valor predeterminado: `wf`.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.8 Validación de instancias XML

[--dtd, --xsd, --namespaces, --assessment-mode]

--dtd=*ARCHIVO*
Especifica el documento DTD externo que debe utilizarse para la validación. Si en el documento XML hay una referencia a una DTD externa, esta opción de la ILC reemplaza a la referencia externa.

--xsd=*ARCHIVO*
Especifica qué esquemas XML deben utilizarse para la validación de documentos XML. Si quiere especificar más de un esquema, añada la opción varias veces.

--namespaces=true|false
Habilita el procesamiento preparado para espacios de nombres. Esta opción es muy útil si quiere buscar en la instancia XML errores resultantes de espacios de nombres erróneos.
Valor predeterminado: `false`.

--assessment-mode=lax|strict
Especifica el modo de evaluación de la validez del esquema, según se define en las especificaciones XSD. El documento XML de instancia se validará en función del modo especificado en esta opción.
Valor predeterminado: `strict`.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.9 Esquemas XML (XSD)

[--schemalocation-hints, --schema-imports, --schema-mapping, --xsd-version]
[Nota sobre schemaLocation-hints]

--schemalocation-hints=load-by-schemalocation|load-by-namespace|load-combining-both|ignore

- **Valor predeterminado**: `load-by-schemalocation`. Este valor toma la URL de la ubicación del esquema de los atributos `xsi:schemaLocation` y `xsi:noNamespaceSchemaLocation` de los documentos XML.
- El valor `load-by-namespace` toma la parte de espacio de nombres del atributo `xsi:schemaLocation` y una cadena vacía en el caso del atributo `xsi:noNamespaceSchemaLocation` y encuentra el esquema por medio de una asignación de catálogo.
- Si usa el valor `load-combining-both` y el espacio de nombres o la URL tienen una asignación de catálogo, se usa dicha asignación. Si ambos tienen asignaciones de catálogo, el valor de la opción `--schema-mapping` decide qué asignación se utiliza. Si ni el espacio de nombres ni la URL tiene una asignación de catálogo, se usa la URL.
- El valor `ignore` ignora los atributos `xsi:schemaLocation` y `xsi:noNamespaceSchemaLocation`.

--schema-imports=load-by-schemalocation|load-preferring-schemalocation|load-by-namespace|load-combining-both|license-namespace-only

Esta opción indica el comportamiento de los elementos `xs:import`. Cada uno de estos elementos tiene un atributo opcional `namespace` y un atributo opcional `schemaLocation`: `<import namespace="unEspacioNombres" schemaLocation="unaURL">`. El comportamiento de los atributos se modifica usando estos valores en la opción:

- `load-by-schemalocation`: el valor del atributo `schemaLocation` se utiliza para buscar el esquema, teniendo en cuenta las asignaciones de catálogo. Si está presente el atributo `namespace`, se importa el espacio de nombres (con licencia).
- `load-preferring-schemalocation`: si está presente, se utiliza el atributo `schemaLocation` teniendo en cuenta las asignaciones de catálogo. Si no está presente el atributo `schemaLocation`, entonces se usa el valor del atributo `namespace` a través de las asignaciones de catálogo. Este es el **valor predeterminado** de la opción `--schema-imports`.
- `load-by-namespace`: el valor del atributo `namespace` se utiliza para buscar el esquema por medio de una asignación de catálogo.
- Si se usa `load-combining-both` y la parte de espacio de nombres o la parte URL tiene una asignación de catálogo, entonces se usa la asignación. Si ambos atributos tienen asignaciones de catálogo, entonces el valor de la opción `--schema-mapping` decide qué asignación se utiliza. Si no hay ninguna asignación de catálogo, se utiliza el atributo `schemaLocation`.
- `license-namespace-only`: se importa el espacio de nombres. No se importa el documento de esquema.

--schema-mapping=prefer-schemalocation|prefer-namespace

Si la opción `--schemalocation-hints` o la opción `--schema-imports` tiene el valor `load-combining-both` y si las partes de espacio de nombres y URL pertinentes tienen asignaciones de catálogo, entonces el valor de la opción especifica cuál de las dos asignaciones se utiliza (la asignación del espacio de nombres o de la URL: el valor `prefer-schemalocation` se refiere a la asignación de la URL). El valor `prefer-schemalocation` es el **valor predeterminado**.

--xsd-version=1.0|1.1|detect

Especifica qué versión de la especificación Schema Definition Language (XSD) del W3C se debe usar.

El **valor predeterminado** es `1.0`.

Esta opción también puede ser útil si quiere ver en qué aspectos no es compatible un esquema 1.0 con la especificación 1.1. El valor `detect` es una característica de Altova. Permite detectar la versión del esquema XML (`1.0` o `1.1`) leyendo el valor del atributo `vc:minVersion` del elemento `<xs:schema>` del documento. Si el valor del atributo `@vc:minVersion` es `1.1`, se entiende que la versión del esquema es `1.1`. Si el atributo tiene otro valor que no sea `1.1` (o si no está presente el atributo `@vc:minVersion`), se entiende que la versión del esquema es `1.0`.

Nota sobre la opción `--schemaLocation-hints`
Los documentos de instancia pueden usar sugerencias para indicar la ubicación del esquema. Para ello se usan dos atributos:

- `xsi:schemaLocation` para documentos de esquema con espacios de nombres de destino. El valor del atributo es un par de elementos: el primero es un espacio de nombres y el segundo es una URL que encuentra el esquema. El nombre del espacio de nombres debe coincidir con el espacio de nombres de destino del esquema.

```
<document xmlns="http://www.altova.com/schemas/test03"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.altova.com/schemas/test03
Test.xsd">
```

- `xsi:noNamespaceSchemaLocation` para documentos de esquema sin espacios de nombres de destino. El valor del atributo es la URL del esquema. El documento de esquema referenciado no puede tener un espacio de nombres de destino.

```
<document xmlns="http://www.altova.com/schemas/test03"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="Test.xsd">
```

La opción `--schemalocation-hints` especifica cómo usar estos dos atributos, sobre todo cómo trabajar con la información del atributo `schemaLocation` (*ver la descripción de la opción*

que aparece más arriba). Recuerde que RaptorXML Development Edition considera que la parte de espacio de nombres del valor de `xsi:noNamespaceSchemaLocation` es una cadena vacía.

Las sugerencias sobre la ubicación del esquema también se pueden incluir en una instrucción de importación `import` de un esquema XML.

```
<import namespace="unEspacioNombres" schemaLocation="unaURL">
```

En la instrucción de importación `import`, las sugerencias se pueden hacer por medio de un espacio de nombres asignado a un esquema en el archivo de catálogo o con una URL directamente, en el atributo `schemaLocation`. La opción `--schema-imports` especifica cómo se selecciona la ubicación del esquema.

3.6.10 XQuery

Opciones propias del comando `xquery`
[--indent-characters, --input, --output, --output-encoding, --output-indent, --output-method, --param]

`--indent-characters=`*`VALOR`*
Especifica la cadena de caracteres que debe usarse como sangría.

`--input=`*`ARCHIVO`*
La URL del archivo XML que debe transformarse.

`--output=`*`ARCHIVO`*
La URL del archivo de salida principal. Por ejemplo, en caso de tener varios archivos HTML de salida, el archivo de salida principal será la ubicación del archivo HTML del punto de entrada. Si no se especifica la opción `--output`, se genera un resultado estándar.

`--output-encoding=`*`VALOR`*
El valor del atributo `encoding` del documento de salida. Son valores válidos todos los nombres del registro de juego de caracteres IANA.
Valor predeterminado: `UTF-8`.

`--output-indent=true|false`
Si el valor es `true`, la sangría del documento de salida seguirá su estructura jerárquica. Si el valor es `false`, el documento de salida no tendrá sangría jerárquica.
Valor predeterminado: `false`.

`--output-method=xml|html|xhtml|text`
Especifica el formato de salida.
Valor predeterminado: `xml`.

`--p, --param=`*`CLAVE:VALOR`*
Especifica el valor de un parámetro externo. En el documento XQuery los parámetros externos se declaran con la declaración `declare variable` seguida de un nombre de variable y después la palabra clave `external` seguida del punto y coma final. Por ejemplo:
`declare variable $foo as xs:string external;`

Al usar la palabra clave `external`, `$foo` se convierte en parámetro externo y su valor se pasa en tiempo de ejecución desde una fuente externa. El parámetro externo recibe un valor con el comando de la ILC. Por ejemplo: `--param=foo:'MiNombre'`

En la descripción anterior, *`CLAVE`* es el nombre de parámetro externo y *`VALOR`* es su valor, dado como expresión XPath. Los nombres de parámetro utilizados en la ILC deben declararse en el documento XQuery. Si se pasan valores a varios parámetros externos en la ILC, cada parámetro debe llevar una opción `--param` distinta. Si la expresión XPath contiene espacios, entonces debe estar entre comillas dobles.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

Opciones compartidas por los comandos `xquery` y `valxquery`
[--omit-xml-declaration, --xquery-version]

--omit-xml-declaration=true|false
Opción de serialización que especifica si la declaración XML se omite en el resultado o no. Si el valor es `true`, el documento de salida no tendrá una declaración XML. Si el valor es `false`, se incluye una declaración XML en el documento de salida.
Valor predeterminado: `false`.

--xquery-version=1|3
Indica si el procesador XQuery debe usar XQuery 1.0 o 3.0.
Valor predeterminado: `3`.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.11 XSLT

Opciones propias del comando `xslt`
[--indent-characters, --input, --output, --param, --streaming-serialization-enabled]

`--indent-characters=`*`VALOR`*
Especifica la cadena de caracteres que debe usarse como sangría.

`--input=`*`ARCHIVO`*
La URL del archivo XML que debe transformarse.

`--output=`*`ARCHIVO`*
La URL del archivo de salida principal. Por ejemplo, en caso de tener varios archivos HTML de salida, el archivo de salida principal será la ubicación del archivo HTML del punto de entrada. Si no se especifica la opción `--output`, se genera un resultado estándar.

`--p, --param=`*`CLAVE:VALOR`*
Especifica un parámetro global de la hoja de estilos. *`CLAVE`* es el nombre del parámetro y *`VALOR`* es una expresión XPath que da un valor al parámetro. Los nombres de parámetro utilizados en la ILC deben declararse en la hoja de estilos. Si usa más de un parámetro, debe usar el modificador `--param` antes de cada parámetro. Si la expresión XPath incluye espacios, entonces debe ir entre comillas dobles, tanto si el espacio está en la expresión propiamente dicha o en un literal de cadena de la expresión. Por ejemplo:

```
raptorxmldev xslt --input=c:\Test.xml --output=c:\Salida.xml --param
date=//node/@att1 --p=title:'cadenasinespacios' --param=title:"'cadena con
espacios'" --p=amount:456 c:\Test.xslt
```

`--streaming-serialization-enabled=true|false`
Habilita la serialización de secuencias de datos.
Valor predeterminado: `true`.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

Opciones compartidas entre los comandos `xslt` y `valxslt`
[--chartext-disable, --dotnetext-disable, --javaext-barcode-location, --javaext-disable,, --template-entry-point, --template-mode, --xslt-version]

`--chartext-disable=true|false`
Deshabilita las extensiones de gráficos.
Valor predeterminado: `false`
Opción no disponible en RaptorXML Development Edition.

`--dotnetext-disable=true|false`
Deshabilita las extensiones .NET.
Valor predeterminado: `false`

`--javaext-barcode-location=`*`ARCHIVO`*
Especifica la ubicación del archivo de extensión de código de barras

`--javaext-disable=true|false`
Deshabilita las extensiones Java.

Valor predeterminado: `false`

`--template-entry-point=`*`VALOR`*
Indica el nombre de una plantilla con nombre de la hoja de estilos XSLT que sirve de punto de entrada de la transformación.

`--template-mode=`*`VALOR`*
Especifica el modo de plantilla que debe usarse para la transformación.

`--xslt-version=1|2|3`
Especifica si el procesador XSLT debe usar XSLT 1.0, XSLT 2.0 o XSLT 3.0.
Valor predeterminado: `3`

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

3.6.12 Archivos ZIP

[--recurse]

--recurse=true|false

Esta opción se utiliza para seleccionar ficheros dentro de un archivo ZIP. Si el valor es `true`, el argumento *ArchivoEntrada* del comando seleccionará el fichero seleccionado también en los subdirectorios.

Por ejemplo: `test.zip|zip\test.xml` seleccionará los ficheros llamados `test.xml` en todos los subdirectorios de la carpeta ZIP. Si quiere puede usar los caracteres comodín `*` y `?` . Por ejemplo: `*.xml` seleccionaría todos los ficheros de la carpeta ZIP que tengan la extensión `.xml`.

El valor predeterminado de esta opción es `false`.

Opción no disponible en RaptorXML Development Edition.

Nota: si no se especifica un valor para la opción, el valor booleano de la opción se establece en `true`.

Altova RaptorXML Development Edition 2014

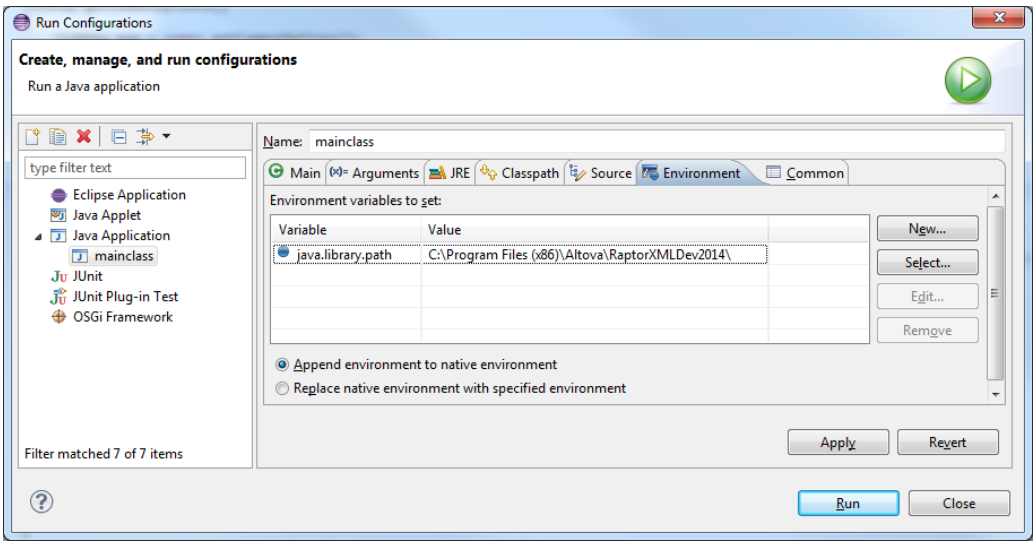
Interfaz Java

4 Interfaz Java

A la API de RaptorXML también se puede acceder desde código Java. Para ello es necesario que las bibliotecas que aparecen más abajo residan en el parámetro classpath. Esta bibliotecas se instalan en la carpeta de instalación.

- `RaptorXMLDev.jar`: la biblioteca que se comunica con la aplicación nativa usando JNI para RaptorXML Development Edition.
- `RaptorXMLDevJava.dll`: la biblioteca nativa que contiene las funciones de procesamiento de RaptorXML Development Edition.
- `RaptorXMLDev_JavaDoc.zip`: un archivo Javadoc que contiene la documentación de ayuda para la API de Java.

Nota: para poder usar la API de Java, los archivos Jar y DLL deben estar en el parámetro classpath. No recomendamos copiar el DLL desde su carpeta de instalación porque usa archivos de catálogo que se cargan usando rutas de acceso relativas. Si se producen excepciones debido a la falta de dependencias de bibliotecas, añada el directorio de instalación a la ruta de acceso del sistema antes de ejecutar el intérprete de Java. Esto se puede hacer desde la línea de comandos igual que hace la muestra instalada (lea el archivo `BuildAndRun.bat`, situado en la carpeta `examples/API` de la carpeta de la aplicación de RaptorXML Development Edition). Desde Eclipse puede añadir la ruta de acceso a la variable `java.library.path` con la opción *Run Configurations* de la sección *Environment* (*imagen siguiente*).



Resumen sobre la interfaz Java
La API de Java viene empaquetada en el paquete `com.altova.raptorxml`. La clase RaptorXML ofrece un método de punto de entrada llamado `getFactory()`, que devuelve objetos `RaptorXMLFactory`. De modo que se puede crear una instancia de `RaptorXMLFactory` con la llamada `RaptorXML.getFactory()`.

La interfaz `RaptorXMLFactory` ofrece métodos para obtener objetos del motor para realizar tareas de validación y procesamiento (como transformaciones XSLT).

Nota: el método `getFactory` devuelve fábricas diferentes dependiendo de la edición de RaptorXML que esté instalada. La edición RaptorXML Server devuelve el objeto de fábrica correspondiente de RaptorXML Server, mientras que la edición RaptorXML Development devuelve el objeto de fábrica de RaptorXML Development Edition.

La interfaz pública de `RaptorXMLFactory` se describe en este fragmento:

```
public interface RaptorXMLFactory
{
   public XMLValidator getXMLValidator();
   public XQuery getXQuery();
   public XSLT getXSLT();
   public void setGlobalCatalog(String catalog);
   public void setUserCatalog(String catalog);
   public void setGlobalResourcesFile(String file);
   public void setGlobalResourceConfig(String config);
   public void setErrorFormat(ENUMErrorFormat format);
   public void setErrorLimit(int limit);
   public void setReportOptionalWarnings(boolean report);
}
```

Para más información consulte la descripción de `RaptorXMLFactory` y de la correspondiente interfaz Java. También puede consultar un ejemplo de proyecto Java.

4.1 Ejemplo de proyecto Java

El fragmento de código Java que aparece más adelante muestra cómo acceder a las funciones básicas. El código tiene varias partes:

- Busca la carpeta de ejemplos y crea una instancia de objeto COM de RaptorXML
- Valida un archivo XML
- Realiza una transformación XSLT y devuelve el resultado en forma de cadena de texto
- Procesa un documento XQuery y devuelve el resultado en forma de cadena de texto
- Ejecuta el proyecto

Estas funciones básicas se incluyen en los archivos de la carpeta `examples/API` de la carpeta de aplicación de RaptorXML Development Edition.

```
public class RunRaptorXML
{
    // Buscar muestras instaladas con el producto
    // (está dos niveles más arriba de examples/API/Java)
    // NOTA: quizás sea necesario cambiar esta ruta de acceso
       static final String strExamplesFolder = System.getProperty("user.dir")
+ "/../../" ;

       static com.altova.raptorxml.RaptorXMLFactory rxml;
```

```
       static void ValidateXML() throws
com.altova.raptorxml.RaptorXMLException
       {
           com.altova.raptorxml.XMLValidator xmlValidator =
rxml.getXMLValidator();
           System.out.println("RaptorXML Java - XML validation");
           xmlValidator.setInputXMLFromText( "<!DOCTYPE root [ <!ELEMENT root
(#PCDATA)> ]> <root>simple input document</root>" );
              if( xmlValidator.isWellFormed() )
                  System.out.println( "The input string is well-formed" );
              else
                  System.out.println( "Input string is not well-formed: " +
xmlValidator.getLastErrorMessage() );

              if( xmlValidator.isValid() )
                  System.out.println( "The input string is valid" );
              else
                  System.out.println( "Input string is not valid: " +
xmlValidator.getLastErrorMessage() );
       }
```

```
       static void RunXSLT() throws com.altova.raptorxml.RaptorXMLException
       {
           System.out.println("RaptorXML Java - XSL Transformation");
           com.altova.raptorxml.XSLT xsltEngine = rxml.getXSLT();
           xsltEngine.setInputXMLFileName( strExamplesFolder + "simple.xml" );
           xsltEngine.setXSLFileName( strExamplesFolder + "transform.xsl" );
           String result = xsltEngine.executeAndGetResultAsString();
           if( result == null )
               System.out.println( "Transformation failed: " +
```



```
xsltEngine.getLastErrorMessage() );
            else
                System.out.println( "Result is " + result );
        }
```

```
        static void RunXQuery() throws com.altova.raptorxml.RaptorXMLException
        {
            System.out.println("RaptorXML Java - XQuery execution");
            com.altova.raptorxml.XQuery xqEngine = rxml.getXQuery();
            xqEngine.setInputXMLFileName( strExamplesFolder + "simple.xml" );
            xqEngine.setXQueryFileName( strExamplesFolder + "CopyInput.xq" );
            System result = xqEngine.executeAndGetResultAsString();
            if( result == null )
                System.out.println( "Execution failed: " +
xqEngine.getLastErrorMessage() );
            else
                System.out.println( "Result is " + result );
        }
```

```
        public static void main(String[] args)
        {
            try
            {
                rxml = com.altova.raptorxml.RaptorXML.getFactory();
                rxml.setErrorLimit( 3 );

                ValidateXML();
                RunXSLT();
                RunXQuery();
            }

            catch( com.altova.raptorxml.RaptorXMLException e )
            {
                e.printStackTrace();
            }

        }

}
```

4.2 Interfaces de RaptorXML para Java

A continuación resumimos las interfaces Java de la API de RaptorXML. Para más información consulte los apartados correspondientes.

- `RaptorXMLFactory`
  Crea una instancia de objeto COM de RaptorXML nueva por medio de una llamada nativa y ofrece acceso a los motores de RaptorXML.
- `XMLValidator`
  Interfaz para el motor de validación XML.
- `XSLT`
  Interfaz para los motores XSLT.
- `XQuery`
  Interfaz para los motores XQuery.
- `RaptorXMLException`
  Interfaz para el método `RaptorXMLException`.

4.2.1 RaptorXMLFactory

`public interface` **`RaptorXMLFactory`**

Descripción
Use `RaptorXMLFactory()` para crear una nueva instancia de objeto COM de RaptorXML. Esto ofrece acceso a los motores de RaptorXML. `RaptorXMLFactory` y el objeto COM de RaptorXML tienen una relación de iguales. Es decir, en adelante las llamadas a la función `get <NOMBREMOTOR>()` devolverán interfaces para la misma instancia del motor.

Más abajo describimos primero los métodos de la interfaz `RaptorXMLFactory` y después sus enumeraciones.

Métodos
Los métodos de la clase se describen por orden alfabético. En la tabla se organizan por grupos para facilitar la consulta.

Motores	Errores, advertencias	Catálogos y recursos globales
getXMLValidator	setErrorFormat	setGlobalCatalog
getXQuery	setErrorLimit	setUserCatalog
getXSLT	setReportOptionalWarnings	setGlobalResourceConfig
		setGlobalResourcesFile

Información sobre el producto	
getProductName	Is64Bit
getProductNameAndVersion	getAPIMajorVersion
getMajorVersion	getAPIMinorVersion
getMinorVersion	getAPIServicePackVersion
getServicePackVersion	

Subir | Métodos | Enumeraciones

`getAPIMajorVersion`
`public int` **`getAPIMajorVersion()`**
Devuelve la versión principal de la API en forma de entero. La versión principal de la API puede ser distinta a la versión principal del producto si la API está conectada a otro servidor.
Devuelve: un entero que es la versión principal de la API.

Subir | Métodos | Enumeraciones

getAPIMinorVersion
`public int` **`getAPIMinorVersion()`**
Devuelve la versión secundaria de la API en forma de entero. La versión secundaria de la API puede ser distinta a la versión secundaria del producto si la API está conectada a otro servidor.
Devuelve: un entero que es la versión secundaria de la API.

Subir | **Métodos** | **Enumeraciones**

getAPIServicePackVersion
`public int` **`getAPIServicePackVersion()`**
Devuelve la versión del service pack de la API en forma de entero. La versión del service pack de la API puede ser distinta a la versión del service pack del producto si la API está conectada a otro servidor.
Devuelve: un entero que es la versión del service pack de la API.

Subir | **Métodos** | **Enumeraciones**

getMajorVersion
`public int` **`getMajorVersion()`**
Devuelve la versión principal del producto en forma de entero. Ejemplo: para `Altova RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `16` (la diferencia entre la versión principal (`2014`) y el año inicial `1998`). Emite la excepción `RaptorXMLException` en caso de error.
Devuelve: un entero que es la versión principal del producto.

Subir | **Métodos** | **Enumeraciones**

getMinorVersion
`public int` **`getMinorVersion()`**
Devuelve la versión secundaria del producto en forma de entero. Ejemplo: para `Altova RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `2` (tomado del número de versión secundaria `r2`). Emite la excepción `RaptorXMLException` en caso de error.
Devuelve: un entero que es la versión secundaria del producto.

Subir | **Métodos** | **Enumeraciones**

getProductName
`public string` **`getProductName()`**
Devuelve el nombre del producto en forma de cadena de texto. Ejemplo: para `Altova RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `Altova RaptorXML Development Edition`. Emite la excepción `RaptorXMLException` en caso de error.
Devuelve: una cadena de texto que es el nombre del producto.

Subir | **Métodos** | **Enumeraciones**

getProductNameAndVersion
`public string` **`getProductNameAndVersion()`**
Devuelve la versión del service pack del producto en forma de entero. Ejemplo: para `Altova RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `Altova RaptorXML Development Edition 2014r2sp1(x64)`. Emite la excepción `RaptorXMLException` en caso de error.
Devuelve: una cadena de texto que es el nombre y la versión del producto.

Subir | **Métodos** | **Enumeraciones**

getServicePackVersion
`public int` **`getServicePackVersion()`**
Devuelve la versión del service pack del producto en forma de entero. Ejemplo: para `RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `1` (del número de versión del service pack `sp1`). Emite la excepción `RaptorXMLException` en caso de error.
Devuelve: un entero que es la versión del service pack del producto.

Subir | **Métodos** | **Enumeraciones**

getXMLValidator
`public XMLValidator` **`getXMLValidator()`**
Recupera el validador XML.
Devuelve: una instancia nueva de `XMLValidator` de este `RaptorXMLFactory`.

Subir | **Métodos** | **Enumeraciones**

getXQuery
`public XQuery` **`getXQuery()`**
Recupera el motor XQuery.
Devuelve: una instancia nueva de `XQuery` de este `RaptorXMLFactory`.

Subir | **Métodos** | **Enumeraciones**

getXSLT
`public XSLT` **`getXSLT()`**
Recupera el motor XSLT.
Devuelve: una instancia nueva de `XSLT` de este `RaptorXMLFactory`.

Subir | **Métodos** | **Enumeraciones**

is64Bit
`public boolean` **`is64Bit()`**
Comprueba si la aplicación es un ejecutable de 64 bits. Ejemplo: para `Altova RaptorXML`

`Development Edition 2014r2sp1(x64)`, devuelve `true`. Emite la excepción `RaptorXMLException` en caso de error.
Devuelve: el booleano `true` si la aplicación es de 64 bits y `false` si no lo es.

Subir | Métodos | Enumeraciones

setErrorFormat
`public void setErrorFormat(ENUMErrorFormat format)`
Define el formato de los errores de RaptorXML como uno de los literales `ENUMErrorFormat` (Text, ShortXML, LongXML).
Parámetros: `format`: almacena el valor del literal `ENUMErrorFormat` seleccionado.

Subir | Métodos | Enumeraciones

setErrorLimit
`public void setErrorLimit(int limit)`
Define el límite de errores de validación de RaptorXML.
Parámetros: `limit`: es de tipo `int` y especifica el número de errores que se deben comunicar antes de detener la ejecución. Utilice `-1` para definir `limit` como ilimitado (es decir, para comunicar todos los errores). El valor predeterminado es `100`.

Subir | Métodos | Enumeraciones

setGlobalCatalog
`public void setGlobalCatalog(String catalog)`
Define la ubicación en forma de URL del archivo de catálogo principal (de punto de entrada).
Parámetros: `catalog`: la cadena suministrada debe ser una URL absoluta que dé la ubicación exacta del archivo de catálogo principal que debe utilizarse.

Subir | Métodos | Enumeraciones

setGlobalResourceConfig
`public void setGlobalResourceConfig(String config)`
Establece la configuración activa del recurso global.
Parámetros: `config`: es de tipo `String` y especifica el nombre de la configuración utilizada por el recurso global activo.

Subir | Métodos | Enumeraciones

setGlobalResourcesFile
`public void setGlobalResourcesFile(String file)`
Establece la ubicación en forma de URL del archivo XML de recursos globales.
Parámetros: `file`: la cadena suministrada debe ser una URL absoluta que dé la ubicación exacta del archivo XML de recursos globales.

Subir | Métodos | Enumeraciones

setReportOptionalWarnings

```
public void setReportOptionalWarnings(boolean report)
```

Habilita/deshabilita la notificación de advertencias. Un valor `true` habilita las advertencias; `false` las deshabilita.

Parámetros: `report`: toma el booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setUserCatalog

```
public void setUserCatalog(String catalog)
```

Establece la ubicación en forma de URL del archivo de catálogo personal del usuario.

Parámetros `catalog`: la cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo de catálogo personal que se debe usar.

Subir | Métodos | Enumeraciones

Enumeraciones
ENUMErrorFormat

ENUMErrorFormat

```
public enum ENUMErrorFormat {
       eFormatText
       eFormatShortXML
       eFormatLongXML }
```

`ENUMErrorFormat` puede tomar uno de estos literales de enumeración: `eFormatText`, `eFormatShortXML`, `eFormatLongXML`. Estos literales definen el formato de los mensajes de error (`eLongXML` aporta los mensajes de error más detallados). El literal predeterminado es `eFormatText`.

Utilizado por (Interfaz::Método):

RaptorXMLFactory	setErrorFormat

Subir | Métodos | Enumeraciones

4.2.2 XMLValidator

```
public interface XMLValidator
```

Descripción
Valida el documento XML, esquema o DTD suministrados. La validación de documentos XML se puede hacer con DTD o esquemas XML internos o externos. También comprueba si los documentos tienen un formato XML correcto. A continuación describimos primero los métodos de la interfaz y después sus enumeraciones.

Métodos
Los métodos de la clase se describen por orden alfabético. En la tabla se organizan por grupos para facilitar la consulta.

Procesamiento	Archivos de entrada	XML Schema
isValid(ENUM type)	setInputXMLFileName	setSchemaImports
isValid	setInputXMLFromText	setSchemalocationHints
isWellFormed(ENUM type)	setInputXMLFileCollection	setSchemaMapping
isWellFormed	setInputXMLTextCollection	setXSDVersion
getLastErrorMessage	setSchemaFileName	
setAssessmentMode	setSchemaFromText	**XML**
setPythonScriptFile	setSchemaFileCollection	setEnableNamespaces
setStreaming	setSchemaTextCollection	setXincludeSupport
	setDTDFileName	setXMLValidationMode
	setDTDFromText	

Subir | Métodos | Enumeraciones

getLastErrorMessage
```
public String getLastErrorMessage()
```
Recupera el último mensaje de error del motor de validación XML.
Devuelve: una cadena que es el último mensaje de error del motor de validación XML.

Subir | Métodos | Enumeraciones

isValid
```
public boolean isValid(ENUMValidationType type)
```
Devuelve el resultado de validar el documento XML, esquema o DTD. El tipo de documento se indica con el parámetro `type`, que toma como valor un literal de `ENUMValidationType`. El resultado es `true` si el documento es válido y `false` si no lo es. Si ocurre un error, se emite

una excepción RaptorXMLException. Use el método getLastErrorMessage para obtener más información.
Parámetros: type: un literal de ENUMValidationType que especifica si se valida un esquema XML, un DTD o un documento XML con un esquema XML o si se valida un documento XML con un documento DTD.
Devuelve: el booleano true si el documento es válido y false si no lo es.

Subir | Métodos | Enumeraciones

isValid
public boolean **isValid()**
Devuelve el resultado de validar el documento suministrado.
Devuelve: el booleano true si el documento es válido y false si no lo es.

Subir | Métodos | Enumeraciones

isWellFormed
public boolean **isWellFormed(**ENUMWellformedCheckType type**)**
Devuelve el resultado de la comprobación de formato XML del documento XML o DTD. El tipo de documento cuyo formato se comprueba viene dado por el parámetro type, que toma como valor un literal de ENUMWellformedCheckType. Si el formato es correcto el resultado es true y false si no lo es. Si ocurre un error, se emite una excepción RaptorXMLException. Use el método getLastErrorMessage para obtener más información.
Parámetros: type: un literal de ENUMWellformedCheckType que especifica si el documento cuyo formato se comprueba es un XML o un DTD.
Devuelve: el booleano true si el formato es correcto y false si no lo es.

Subir | Métodos | Enumeraciones

isWellFormed
public boolean **isWellFormed()**
Devuelve el resultado de la comprobación de formato XML del documento XML o DTD. Si el formato es correcto el resultado es true y false si no lo es.
Devuelve: el booleano true si el formato es correcto y false si no lo es.

Subir | Métodos | Enumeraciones

setAssessmentMode
public void **setAssessmentMode(**ENUMAssessmentMode mode**)**
Establece el modo de evaluación de la validación XML (Strict/Lax), que se define en el parámetro mode, que toma como valor un literal de ENUMAssessmentMode.
Parámetros: mode: un literal de ENUMAssessmentMode que especifica si la validación debe strict o lax o si se debe pasar por alto.

Subir | Métodos | Enumeraciones

setDTDFileName
`public void` **`setDTDFileName(`**`String filePath`**`)`**
Establece la ubicación en forma de URL del documento DTD que se debe usar para la validación.
Parámetros: `filePath`: cadena de texto que debe ser una URL absoluta que dé la ubicación exacta del archivo DTD que se debe usar.

Subir | Métodos | Enumeraciones

setDTDFromText
`public void` **`setDTDFromText(`**`String dtdText`**`)`**
Aporta el contenido del documento DTD en forma de texto.
Parámetros: `dtdText`: cadena de texto es el documento DTD que se debe usar para la validación.

Subir | Métodos | Enumeraciones

setEnableNamespaces
`public void` **`setEnableNamespaces(`**`boolean enable`**`)`**
Habilita el procesamiento preparado para espacios de nombres. Es útil si quiere buscar errores en la instancia XML derivados del uso incorrecto de espacios de nombres. El valor `true` habilita el procesamiento preparado para espacios de nombres; `false` lo deshabilita. El predeterminado es `false`.
Parámetros: `support`: toma el booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setInputXMLFileCollection
`public void` **`setInputXMLFileCollection(`**`Collection<?> fileCollection`**`)`**
Aporta la colección de archivos XML que se usará como datos de entrada. Los archivos se identifican con su dirección URL. *Recuerde que el procesamiento de múltiples archivos no es compatible con la edición Development Edition.*
Parámetros: `fileCollection`: colección de cadenas de texto, cada una de las cuales es la URL absoluta de un archivo XML de entrada.

Subir | Métodos | Enumeraciones

setInputXMLFileName
`public void` **`setInputXMLFileName(`**`String filePath`**`)`**
Establece la ubicación en forma de URL del documento XML que se debe validar.
Parámetros: `filePath`: cadena de texto que debe ser una URL absoluta que dé la ubicación exacta del archivo XML.

Subir | Métodos | Enumeraciones

setInputXMLFromText
`public void setInputXMLFromText(String inputText)`
Aporta el contenido del documento XML que se debe validar.
Parámetros: `inputText`: la cadena suministrada es el contenido del documento XML que se debe validar.

Subir | Métodos | Enumeraciones

setInputXMLTextCollection
`public void setInputXMLTextCollection(Collection<?> stringCollection)`
Aporta el contenido de varios archivos XML que se usarán como datos de entrada. *Recuerde que el procesamiento de múltiples archivos no es compatible con la edición Development Edition.*
Parámetros: `stringCollection`: colección de cadenas de texto, cada una de las cuales es el contenido de un archivo XML.

Subir | Métodos | Enumeraciones

setPythonScriptFile
`public void setPythonScriptFile(String file)`
Establece la ubicación en forma de URL del archivo de script Python.
Parámetros: `file`: cadena de texto que debe ser una URL absoluta que dé la ubicación exacta del archivo Python. *Recuerde que los scripts Python no son compatibles con al edición Development Edition.*

Subir | Métodos | Enumeraciones

setSchemaFileCollection
`public void setSchemaFileCollection(Collection<?> fileCollection)`
Aporta la colección de archivos XML que se usarán como esquemas XML externos. Los archivos se identifican por sus URL. *Recuerde que el procesamiento de múltiples archivos no es compatible con la edición Development Edition.*
Parámetros: `fileCollection`: colección de cadenas de texto, cada una de las cuales es la URL absoluta de un archivo de esquema XML.

Subir | Métodos | Enumeraciones

setSchemaFileName
`public void setSchemaFileName(String filePath)`
Establece la ubicación en forma de URL del documento de esquema XML que se debe usar.
Parámetros: `filePath`: cadena de texto que debe ser una URL absoluta que dé la ubicación exacta del archivo de esquema XML.

Subir | Métodos | Enumeraciones

setSchemaFromText
`public void setSchemaFromText(String schemaText)`
Aporta el contenido del documento XML que se debe usar.
Parámetros: `schemaText`: cadena de texto que es el contenido del documento de esquema XML que se debe usar.

Subir | Métodos | Enumeraciones

setSchemaImports
`public void setSchemaImports(ENUMSchemaImports opt)`
Indica cómo ocuparse de las importaciones de esquema dependiendo del valor de los atributos de los elementos `xs:import`. La opción se indica mediante el literal de `ENUMSchemaImports` seleccionado.
Parámetros: `opt`: almacena el literal de `ENUMSchemaImports` que determina qué se hace con las importaciones de esquemas. Para más información consulte la descripción de `ENUMSchemaImports`.

Subir | Métodos | Enumeraciones

setSchemalocationHints
`public void setSchemalocationHints(ENUMLoadSchemalocation opt)`
Especifica el mecanismo que se debe usar para encontrar el esquema. El mecanismo viene dado por el literal de `ENUMLoadSchemalocation` seleccionado.
Parámetros: `opt`: almacena el literal de `ENUMLoadSchemalocation` que determina el mecanismo que se debe utilizar. Para más información consulte la descripción de `ENUMLoadSchemalocation`.

Subir | Métodos | Enumeraciones

setSchemaMapping
`public void setSchemaMapping(ENUMSchemaMapping opt)`
Establece la asignación que se debe usar para encontrar el esquema. La asignación viene dada por el literal de `ENUMSchemaMapping` seleccionado.
Parámetros: `opt`: almacena el literal de `ENUMSchemaMapping`. Para más información consulte la descripción de `ENUMSchemaMapping`.

Subir | Métodos | Enumeraciones

setInputSchemaTextCollection
`public void setInputSchemaTextCollection(Collection<?> stringCollection)`
Aporta el contenido de varios documentos de esquema XML. *Recuerde que el procesamiento de múltiples archivos no es compatible con la edición Development Edition.*
Parámetros: `stringCollection`: colección de cadenas de texto, cada una de las cuales es el

contenido de un documento de esquema XML.

Subir | Métodos | Enumeraciones

setStreaming
`public void setStreaming(boolean support)`
Habilita la validación de transmisión por secuencias. En el modo de transmisión por secuencias, los datos almacenados en memoria se minimizan y el procesamiento es más rápido.
Parámetros: `support`: el valor `true` habilita la validación de transmisión por secuencias; `false` la deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setXincludeSupport
`public void setXIncludeSupport(boolean support)`
Habilita o deshabilita el uso de elementos `XInclude`. El valor `true` lo habilita; `false` lo deshabilita. El valor predeterminado es `false`.
Parámetros: `support`: el valor booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setXMLValidationMode
`public void setXMLValidationMode(ENUMXMLValidationMode mode)`
Establece el modo de validación XML, que es un literal de enumeración de `ENUMXMLValidationMode`.
Parámetros: `mode`: un literal de enumeración de `ENUMXMLValidationMode` que determina si se valida el documento o si se comprueba su formato XML.

Subir | Métodos | Enumeraciones

setXSDVersion
`public void setXSDVersion(ENUMXSDVersion version)`
Establece la versión de XML Schema que se debe usar para validar el documento XML.
Parámetros: `version`: un literal de enumeración de `ENUMXSDVersion` que establece la versión de XML Schema.

Subir | Métodos | Enumeraciones

Enumeraciones
ENUMAssessmentMode

ENUMLoadSchemalocation
ENUMSchemaImports
ENUMSchemaMapping
ENUMValidationMode
ENUMValidationType
ENUMWellformedCheckType
ENUMXSDVersion

Subir | Métodos | Enumeraciones

ENUMAssessmentMode

```
public enum ENUMAssessmentMode {
       eAssessmentModeLax
       eAssessmentModeStrict    }
```

ENUMAssessmentMode toma uno de estos literales: eAssessmentModeLax, eAssessmentModeStrict. Definen si la validación debe ser de tipo estricto (strict) o permisivo (lax).

Utilizado por (Interfaz::Método):

XMLValidator	setAssessmentMode

Subir | Métodos | Enumeraciones

ENUMLoadSchemalocation

```
public enum ENUMLoadSchemalocation {
       eLoadBySchemalocation
       eLoadByNamespace
       eLoadCombiningBoth
       eLoadIgnore    }
```

ENUMLoadSchemalocation contiene el literal de enumeración que especifica el mecanismo para encontrar el esquema. La selección se basa en el atributo schema location del documento de instancia XML. Este atributo puede ser xsi:schemaLocation o xsi:noNamespaceSchemaLocation.

- eLoadBySchemalocation utiliza la URL del atributo schema location del archivo XML. Este literal de enumeración es el valor predeterminado.
- eLoadByNamespace utiliza la parte de espacio de nombres de xsi:schemaLocation y una cadena vacía en el caso de xsi:noNamespaceSchemaLocation para encontrar el esquema a través de la asignación de catálogos.
- eLoadCombiningBoth: si el URL del espacio de nombres o de schema location tiene una asignación de catálogo, se usa la URL de schema location.
- eLoadCombiningBoth: se pasan por alto tanto el atributo xsi:schemaLocation como el atributo xsi:noNamespaceSchemaLocation.

Utilizado por (Interfaz::Método):

XMLValidator	setSchemalocationHints
XSLT	setSchemalocationHints

Subir | Métodos | Enumeraciones

ENUMSchemaImports

```
public enum ENUMSchemaImports {
       eSILoadBySchemalocation
       eSILoadPreferringSchemalocation
       eSILoadByNamespace
       eSILoadCombiningBoth
       eSILicenseNamespaceOnly   }
```

`ENUMSchemaImports` contiene el literal de enumeración que define el comportamiento de los elementos `xs:import` del esquema, cada uno de los cuales tiene un atributo opcional `namespace` y un atributo opcional `schemaLocation`.

- `eSILoadBySchemalocation` usa el valor del atributo `schemaLocation` para encontrar el esquema, teniendo en cuenta las asignaciones de catálogo. Si está presente el atributo `namespace`, se importa el espacio de nombres.
- `eSILoadPreferringSchemalocation`: si está presente el atributo `schemaLocation`, entonces se usa teniendo en cuenta las asignaciones de catálogo. Si no está presente el atributo `schemaLocation`, entonces se usa el valor del atributo `namespace` mediante la asignación de catálogo. Este literal de enumeración es el valor predeterminado.
- `eSILoadByNamespace` usa el valor del atributo `namespace` para encontrar el esquema por medio de una asignación de catálogo.
- `eSILoadCombiningBoth`: si la URL de `namespace` o la URL de `schemaLocation` tiene una asignación de catálogo, ésta se utiliza. Si ambas tienen asignaciones de catálogo, entonces el valor de `ENUMSchemaMapping` decide qué asignación se utiliza. Si ninguna URL tiene una asignación de catálogo, entonces se usa la URL de `schemaLocation`.
- `eSILicenseNamespaceOnly`: se importa el espacio de nombres. No se importa ningún documento de esquema.

Utilizado por (Interfaz::Método):

XMLValidator	setSchemaImports
XSLT	setSchemaImports

Subir | Métodos | Enumeraciones

ENUMSchemaMapping

```
public enum ENUMSchemaMapping {
       eSMPreferSchemalocation
       eSMPreferNamespace   }
```

`ENUMSchemaMapping` contiene el literal de enumeración que especifica si se selecciona el espacio de nombres o la ubicación del esquema.

- `eSMPreferNamespace`: selecciona el espacio de nombres.
- `eSMPreferSchemalocation`: selecciona la ubicación del esquema. Es el valor predeterminado.

Utilizado por (Interfaz::Método):

XMLValidator	setSchemaMapping
XSLT	setSchemaMapping

Subir | Métodos | Enumeraciones

ENUMXMLValidationMode

```
public enum ENUMXMLValidationMode {
       eProcessingModeValid
       eProcessingModeWF  }
```

`ENUMXMLValidationMode` contiene el literal de enumeración que especifica el tipo de validación XML que se debe llevar a cabo (validación o comprobación de formato XML).

- `eProcessingModeValid`: establece el modo de procesamiento XML en `validation`.
- `eProcessingModeValid`: establece el modo de procesamiento XML en `wellformed`. Es el valor predeterminado.

Utilizado por (Interfaz::Método):

XMLValidator	setXMLValidationMode
XSLT	setXMLValidationMode
XQuery	setXMLValidationMode

Subir | Métodos | Enumeraciones

ENUMValidationType

```
public enum ENUMValidationType {
       eValidateAny
       eValidateXMLWithDTD
       eValidateXMLWithXSD
       eValidateDTD
       eValidateXSD }
```

`ENUMValidationType` contiene el literal de enumeración que especifica si los documentos XML se deben validar con un documento DTD o XSD.

- `eValidateAny`: el tipo de documento se detecta automáticamente.
- `eValidateXMLWithDTD`: valida el documento XML con un archivo DTD.
- `eValidateXMLWithXSD`: valida el documento XML con un archivo XSD.
- `eValidateDTD`: valida un documento DTD.
- `eValidateXSD`: valida un documento XSD.

Utilizado por (Interfaz::Método):

XMLValidator	isValid

Subir | Métodos | Enumeraciones

ENUMWellformedCheckType

```
public enum ENUMWellformedCheckType {
       eWellformedAny
       eWellformedXML
       eWellformedDTD   }
```

`ENUMWellformedCheckType` contiene el literal de enumeración que especifica de qué tipo es el

documento cuyo formato XML debe comprobarse.

- eWellformedAny: el tipo de documento se detecta automáticamente.
- eWellformedXML: comprueba el formato XML de un documento XML.
- eWellformedDTD: comprueba el formato XML de un documento DTD.

Utilizado por (Interfaz::Método):
XMLValidator isWellformed

Subir | Métodos | Enumeraciones

ENUMXSDVersion

```
public enum ENUMXSDVersion {
       eXSDVersionAuto
       eXSDVersion10
       eXSDVersion11    }
```

ENUMXSDVersion contiene el literal de enumeración que especifica la versión de XML Schema.

- eXSDVersionAuto: la versión se detecta automáticamente dependiendo del valor del atributo vc:minVersion del documento XSD. Si el atributo vc:minVersion del documento XSD tiene el valor 1.1, se considera que el documento tiene la versión XSD 1.1. Si este atributo tiene cualquier otro valor o si el atributo no está en el documento, se entiende que el documento tiene la versión XSD 1.0.
- eXSDVersion10: la versión de XML Schema para la validación se establece en XML Schema 1.0.
- eXSDVersion11: la versión de XML Schema para la validación se establece en XML Schema 1.1.

Utilizado por (Interfaz::Método):
XMLValidator setXSDVersion
XSLT setXSDVersion
XQuery setXSDVersion

Subir | Métodos | Enumeraciones

4.2.3 XSLT

```
public interface XSLT
```

Descripción
Transforma datos XML usando el documento XSLT 1.0, 2.0 o 3.0 dado. Los documentos XML y XSLT se pueden aportar en forma de archivo (con una URL) o de cadena de texto. El resultado se devuelve en forma de archivo (en la ubicación de destino dada) o como cadena de texto. Puede indicar parámetros XSLT y habilitar funciones de extensión de Altova para un procesamiento avanzado (p. ej. generación de gráficos). El documento XSLT también se puede validar. Cuando las cadenas de texto de entrada se deben interpretar como URL, deben utilizarse rutas de acceso absolutas.

A continuación describimos primero los métodos de la interfaz `XSLT` y después sus enumeraciones.

Métodos
Los métodos de la clase se describen por orden alfabético. En la tabla se organizan por grupos para facilitar la consulta.

Procesamiento	XSLT
isValid	setVersion
execute	setXSLFileName
executeAndGetResultAsString	setXSLFromText
executeAndGetResultAsStringWithBaseOutputURI	setaddExternalParameter
getLastErrorMessage	setclearExternalParametersList
setIndentCharacters	setInitialTemplateMode
setStreamingSerialization	setNamedTemplateEntryPoint

XML Schema	XML	Extensiones
setSchemaImports	setInputXMLFileName	setChartExtensionsEnabled
setSchemalocationHints	setInputXMLFromText	setDotNetExtensionsEnabled
setSchemaMapping	setLoadXMLWithPSVI	setJavaExtensionsEnabled
setXSDVersion	setXincludeSupport	setJavaBarcodeExtensionLocation
	setXMLValidationMode	

Subir | **Métodos** | **Enumeraciones**

---

addExternalParameter

```
public void addExternalParameter(String name, String value)
```

Añade el nombre y valor de un parámetro externo nuevo. Cada parámetro externo y su valor

debe especificarse en una llamada distinta al método. Los parámetros deben declararse en el documento XML. Como los valores de parámetros son expresiones XPath, los valores de parámetros que sean cadenas deben ir entre comillas simples.
Parámetros: `name`: almacena el nombre del parámetro, que es un QName, como cadena de texto. `value`: almacena el valor del parámetro como cadena de texto.

Subir | Métodos | Enumeraciones

clearExternalParameterList
`public void` **clearExternalVariableList()**
Borra la lista de parámetros externos creada con el método `AddExternalParameter`.

Subir | Métodos | Enumeraciones

execute
`public boolean` **execute(**`String outputFile`**)**
Ejecuta la transformación XSLT de acuerdo con la especificación XSLT indicada en `ENUMXSLTVersion` (véase el método `setVersion`) y guarda el resultado en el archivo de salida indicado en el parámetro `outputFile`. Si ocurre un error, se emite una excepción `RaptorXMLException`. Use el método `getLastErrorMessage` para obtener más información.
Parámetros: `outputFile`: una cadena que indica la ubicación (ruta de acceso y nombre de archivo) del archivo de salida.
Devuelve: el booleano `true` si la ejecución finaliza correctamente y `false` si no se puede completar la ejecución.

Subir | Métodos | Enumeraciones

executeAndGetResultAsString
`public String` **executeAndGetResultAsString()**
Ejecuta la transformación XSLT de acuerdo con la especificación XSLT indicada en `ENUMXSLTVersion` (véase el método `setVersion`) y devuelve el resultado en forma de cadena de texto. Si ocurre un error, se emite una excepción `RaptorXMLException`. Use el método `getLastErrorMessage` para obtener más información.
Devuelve: una cadena que es el resultado de la transformación XSLT.

Subir | Métodos | Enumeraciones

executeAndGetResultAsStringWithBaseOutputURI
`public String` **executeAndGetResultAsStringWithBaseOutputURI(**`String baseURI`**)**
Ejecuta la transformación XSLT de acuerdo con la especificación XSLT indicada en `ENUMXSLTVersion` (véase el método `setVersion`) y devuelve el resultado en forma de cadena de texto en la ubicación definida por el URI base. Si ocurre un error, se emite una excepción `RaptorXMLException`. Use el método `getLastErrorMessage` para obtener más información.
Parámetros: `baseURI`: una cadena que indica el URI.
Devuelve: una cadena que es el resultado de la transformación XSLT.

Subir | Métodos | Enumeraciones

getLastErrorMessage
`public String` **`getLastErrorMessage()`**
Recupera el último mensaje de error del motor XSLT.
Devuelve: una cadena que es el último mensaje de error del motor XSLT.

Subir | Métodos | Enumeraciones

isValid
`public boolean` **`isValid()`**
Devuelve el resultado de validar el documento XSLT de acuerdo con la especificación XSLT indicada en `ENUMXSLTVersion` (véase el método `setVersion`). El resultado es `true` si el documento es válido y `false` si no lo es. Si ocurre un error, se emite una excepción `RaptorXMLException`. Use el método `getLastErrorMessage` para obtener más información.
Devuelve: el booleano `true` si el documento es válido y `false` si no lo es.

Subir | Métodos | Enumeraciones

setChartExtensionsEnabled
`public void` **`setChartExtensionsEnabled(`**`boolean enable`**`)`**
Habilita o deshabilita las funciones de extensión de Altova para gráficos. *Esta característica no es compatible con la edición Development Edition*.
Parámetros: `enable`: el valor `true` habilita las extensiones para gráficos; `false` las deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setDotNetExtensionsEnabled
`public void` **`setDotNetExtensionsEnabled(`**`boolean enable`**`)`**
Habilita o deshabilita las funciones de extensión .NET.
Parámetros: `enable`: el valor `true` habilita las extensiones .NET; `false` las deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setJavaBarcodeExtensionLocation
`public void` **`setJavaBarcodeExtensionLocation(`**`String path`**`)`**
Especifica la ubicación del archivo de extensión para códigos de barras. Consulte este anexo para obtener más información.
Parámetros: `path`: la cadena indicada debe ser una URL absoluta que dé la ubicación base del archivo que se debe usar.

Subir | Métodos | Enumeraciones

setJavaExtensionsEnabled
`public void setJavaExtensionsEnabled(boolean enable)`
Habilita o deshabilita las funciones de extensión Java.
Parámetros: `enable`: el valor `true` habilita las extensiones Java; `false` las deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setIndentCharacters
`public void setIndentCharacters(String chars)`
Establece el carácter que se usará como sangría en el documento de salida.
Parámetros: `chars`: almacena el carácter de sangría.

Subir | Métodos | Enumeraciones

setInitialTemplateMode
`public void setInitialTemplateMode(String mode)`
Establece el nombre del modo de plantilla inicial. El procesamiento comenzará a partir de las plantillas que tengan este valor de modo. La transformación debe iniciarse después de asignar los documentos XML y XSLT.
Parámetros: `mode`: el nombre del modo de plantilla inicial, en forma de cadena de texto.

Subir | Métodos | Enumeraciones

setInputXMLFileName
`public void setInputXMLFileName(String xmlFile)`
Establece la ubicación en forma de URL del documento XML que se debe transformar.
Parámetros: `xmlFile`: la cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo XML que se debe utilizar.

Subir | Métodos | Enumeraciones

setInputXMLFromText
`public void setInputXMLFromText(String xmlText)`
Aporta el contenido del documento XML de entrada como texto.
Parámetros: `xmlText`: la cadena indicada son los datos XML que se deben procesar.

Subir | Métodos | Enumeraciones

setLoadXMLWithPSVI

```
public void setLoadXMLWithPSVI(boolean load)
```
Habilita o deshabilita la opción de carga y uso del conjunto Post Schema Validation Infoset (PSVI). Si se carga el PSVI, la información obtenida del esquema se puede usar para calificar datos del documento XML. El valor `true` habilita la carga de PSVI; `false` la deshabilita.
Parámetros: `load`: toma el booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setNamedTemplateEntryPoint
```
public void setNamedTemplateEntryPoint(String template)
```
Aporta el nombre de la plantilla con nombre con la que se debe iniciar el procesamiento.
Parámetros: `template`: el nombre de la plantilla con nombre, en forma de cadena de texto.

Subir | Métodos | Enumeraciones

setSchemaImports
```
public void setSchemaImports(ENUMSchemaImports opt)
```
Especifica cómo se deben realizar las importaciones de esquema en función del valor de los atributos de los elementos `xs:import`. Esto se especifica con el literal de `ENUMSchemaImports` seleccionado.
Parámetros: `opt`: almacena el literal de `ENUMSchemaImports` que determina cómo se deben realizar las importaciones de esquema.

Subir | Métodos | Enumeraciones

setSchemalocationHints
```
public void setSchemalocationHints(ENUMLoadSchemalocation opt)
```
Especifica el mecanismo que se debe usar para encontrar el esquema. El mecanismo viene dado por el literal de `ENUMLoadSchemalocation` seleccionado.
Parámetros: `opt`: almacena el literal de `ENUMLoadSchemalocation` que determina el mecanismo que se debe utilizar para encontrar el esquema.

Subir | Métodos | Enumeraciones

setSchemaMapping
```
public void setSchemaMapping(ENUMSchemaMapping opt)
```
Establece qué asignación se usa para encontrar el esquema. La asignación viene dada por el literal de `ENUMSchemaMapping` seleccionado.
Parámetros: `opt`: almacena el literal de `ENUMSchemaMapping`.

Subir | Métodos | Enumeraciones

setStreamingSerialization
```
public void setStreamingSerialization(boolean support)
```

Habilita la serialización de secuencias de datos. En el modo de transmisión por secuencias, los datos almacenados en memoria se minimizan y el procesamiento es más rápido.
Parámetros: `support`: el valor `true` habilita la serialización de secuencias de datos; `false` la deshabilita.

Subir | Métodos | Enumeraciones

setVersion
`public void setVersion(EnumXSLTVersion version)`
Establece la versión XSLT que se debe usar para el procesamiento (validación o transformación XSLT).
Parámetros: `version`: almacena un literal de `EnumXSLTVersion`: `eVersion10`, `eVersion20` o `eVersion30`.

Subir | Métodos | Enumeraciones

setXincludeSupport
`public void setXIncludeSupport(boolean support)`
Habilita o deshabilita el uso de elementos `XInclude`. El valor `true` habilita el uso de elementos XInclude; `false` lo deshabilita. El valor predeterminado es `false`.
Parámetros: `support`: toma el booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setXMLValidationMode
`public void setXMLValidationMode(ENUMXMLValidationMode mode)`
Establece el modo de validación XML, que es un literal de enumeración de `ENUMXMLValidationMode`.
Parámetros: `mode`: es un literal de enumeración de `ENUMXMLValidationMode`.

Subir | Métodos | Enumeraciones

setXSDVersion
`public void setXSDVersion(ENUMXSDVersion version)`
Establece la versión de XML Schema con la que se debe validar el documento XML.
Parámetros: `version`: es un literal de enumeración de `ENUMXSDVersion`.

Subir | Métodos | Enumeraciones

setXSLFileName
`public void setXSLFileName(String xslFile)`
Establece la ubicación en forma de URL del documento XSLT que se debe usar para la transformación.
Parámetros: `xslFile`: la cadena indicada debe ser una URL absoluta que aporta la ubicación

exacta del archivo XSLT.

Subir | Métodos | Enumeraciones

setXSLFromText
`public void setXSLFromText(String xslText)`
Aporta el contenido del documento XSLT como texto.
Parámetros: `xslText`: la cadena indicada es el documento XSLT que se debe usar para la transformación.

Subir | Métodos | Enumeraciones

Enumeraciones
ENUMXSLTVersion

ENUMXSLTVersion

```
public enum ENUMXSLTVersion {
       eVersion10
       eVersion20
       eVersion30 }
```

`ENUMXSLTVersion` toma uno de estos literales de enumeración: `eVersion10`, `eVersion20`, `eVersion30`. Determinan la versión XSLT que se debe usar para el procesamiento (validación o transformación XSLT).

Utilizado por (Interfaz::Método):
XSLT setVersion

Subir | Métodos | Enumeraciones

4.2.4 XQuery

```
public interface XQuery
```

Descripción
Ejecuta documentos XQuery 1.0 y 3.0 usando el motor de RaptorXML. Los documentos XQuery y XML pueden darse como archivos (por medio de una URL) o como cadenas de texto. El resultado se devuelve en forma de archivo (en la ubicación de destino indicada) o como cadena de texto. Puede indicar variables XQuery externas y habilitar diferentes opciones de serialización. El documento XQuery también se puede validar. Cuando las cadenas de texto de entrada se deben interpretar como URL, deben utilizarse rutas de acceso absolutas.

A continuación describimos primero los métodos de la interfaz XQuery y después sus enumeraciones.

Métodos
Los métodos de la clase se describen por orden alfabético. En la tabla se organizan por grupos para facilitar la consulta.

Procesamiento	XML	XQuery
isValid	setInputXMLFileName	setVersion
execute	setInputXMLFromText	setXQueryFileName
executeAndGetResultAsString	setLoadXMLWithPSVI	setXQueryFromText
getLastErrorMessage	setXincludeSupport	addExternalVariable
	setXMLValidationMode	clearExternalVariableList
	setXSDVersion	

Opciones de serialización	Extensiones
setIndentCharacters	setChartExtensionsEnabled
setOutputEncoding	setDotNetExtensionsEnabled
setOutputIndent	setJavaExtensionsEnabled
setOutputMethod	
setOutputOmitXMLDeclaration	

Subir | Métodos | Enumeraciones

addExternalVariable
```
public void addExternalVariable(String name, String value)
```
Añade el nombre y valor de una variable externa nueva. Cada variable externa y su valor se debe especificar en una llama distinta al método. Las variables se deben declarar en el documento XQuery (y la declaración de tipo es opcional). Sea cual sea la declaración de tipo para la variable externa en el documento XQuery, el valor de la variable dado como argumento

de `AddExternalVariable` no tiene por qué ser un delimitador (como las comillas, por ejemplo).
Parámetros: `name`: almacena el nombre de la variable, que es un QName, como cadena de texto. `value`: almacena el valor de la variable como cadena de texto.

Subir | Métodos | Enumeraciones

clearExternalVariableList
`public void` **clearExternalVariableList()**
Borra la lista de variables externas creadas con el método `AddExternalVariable`.

Subir | Métodos | Enumeraciones

execute
`public boolean` **execute(**`String outputFile`**)**
Ejecuta la transformación XQuery de acuerdo con la especificación XQuery indicada en `ENUMXQueryVersion` (véase el método `setVersion`) y guarda el resultado en el archivo de salida indicado en el parámetro `outputFile`.
Parámetros: `outputFile`: una cadena que indica la ubicación (ruta de acceso y nombre de archivo) del archivo de salida.
Devuelve: el booleano `true` si la ejecución finaliza correctamente, `false` si no.

Subir | Métodos | Enumeraciones

executeAndGetResultAsString
`public String` **executeAndGetResultAsString()**
Ejecuta la transformación XQuery de acuerdo con la especificación XQuery indicada en `ENUMXQueryVersion` (véase el método `setVersion`) y devuelve el resultado en forma de cadena de texto.
Devuelve: una cadena que es el resultado de la ejecución de XQuery.

Subir | Métodos | Enumeraciones

getLastErrorMessage
`public String` **getLastErrorMessage()**
Recupera el último mensaje de error del motor XQuery.
Devuelve: una cadena que es el último mensaje de error del motor XQuery.

Subir | Métodos | Enumeraciones

isValid
`public boolean` **isValid()**
Devuelve el resultado de validar el documento XQuery de acuerdo con la especificación XQuery indicada en `ENUMXQueryVersion` (véase el método `setVersion`). El resultado es `true`

si el documento es válido y `false` si no lo es. Si ocurre un error, se emite una excepción `RaptorXMLException`. Use el método `getLastErrorMessage` para obtener más información.
Devuelve: el booleano `true` si el documento es válido, `false` si no lo es.

Subir | Métodos | Enumeraciones

setChartExtensionsEnabled
`public void setChartExtensionsEnabled(boolean enable)`
Habilita o deshabilita las funciones de extensión de Altova para gráficos. *Esta característica no es compatible con la edición Development Edition.*
Parámetros: `enable`: el valor `true` habilita las extensiones para gráficos; `false` las deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setDotNetExtensionsEnabled
`public void setDotNetExtensionsEnabled(boolean enable)`
Habilita o deshabilita las funciones de extensión .NET.
Parámetros: `enable`: el valor `true` habilita las extensiones .NET; `false` las deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setIndentCharacters
`public void setIndentCharacters(String chars)`
Establece el carácter que se usará como sangría en el documento de salida.
Parámetros: `chars`: almacena el carácter de sangría.

Subir | Métodos | Enumeraciones

setInputXMLFileName
`public void setInputXMLFileName(String xmlFile)`
Establece la ubicación en forma de URL del documento XML que se debe usar para la ejecución de XQuery.
Parámetros: `xmlFile`: la cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo XML que se debe usar.

Subir | Métodos | Enumeraciones

setInputXMLFromText
`public void setInputXMLFromText(String xmlText)`
Aporta el contenido del documento XML de entrada como texto.
Parámetros: `xmlText`: la cadena indicada son los datos XML que se deben procesar.

Subir | Métodos | Enumeraciones

setJavaExtensionsEnabled
`public void setJavaExtensionsEnabled(boolean enable)`
Habilita o deshabilita las funciones de extensión Java.
Parámetros: `enable`: el valor `true` habilita las extensiones Java; `false` las deshabilita. El valor predeterminado es `true`.

Subir | Métodos | Enumeraciones

setLoadXMLWithPSVI
`public void setLoadXMLWithPSVI(boolean load)`
Habilita o deshabilita la opción de carga y uso del conjunto Post Schema Validation Infoset (PSVI). Si se carga el PSVI, la información obtenida del esquema se puede usar para calificar datos del documento XML. El valor `true` habilita la carga de PSVI; `false` la deshabilita.
Parámetros: `load`: toma el booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setOutputEncoding
`public void setOutputEncoding(String encoding)`
Establece la codificación del documento de salida.
Parámetros: `encoding`: usa un nombre de codificación IANA oficial como `UTF-8`, `UTF-16`, `US-ASCII`, `ISO-8859-1` como cadena de texto.

Subir | Métodos | Enumeraciones

setOutputIndent
`public void setOutputIndent(boolean indent)`
Habilita o deshabilita la aplicación de sangría en el documento de salida.
Parámetros: `indent`: el valor `true` habilita la sangría; `false` la deshabilita.

Subir | Métodos | Enumeraciones

setOutputMethod
`public void setOutputMethod(String outputMethod)`
Especifica la serialización del documento de salida.
Parámetros: `outputMethod`: los valores válidos son `xml | xhtml | html | text`, dados como cadenas de texto. El valor predeterminado es `xml`.

Subir | Métodos | Enumeraciones

setOutputOmitXMLDeclaration
`public void setOutputOmitXMLDeclaration(boolean omit)`

Habilita o deshabilita la inclusión de la declaración XML en el documento de salida.
Parámetros: `omit`: el valor `true` omite la declaración; `false` la incluye. El valor predeterminado es `false`.

Subir | Métodos | Enumeraciones

setVersion
`public void setVersion(EnumXQueryVersion version)`
Establece la versión de XQuery que se debe usar para el procesamiento (validación o ejecución de XQuery).
Parámetros: `version`: almacena un literal de enumeración de `EnumXQueryVersion`: `eVersion10` o `eVersion30`. El valor predeterminado es `eVersion30ml`.

Subir | Métodos | Enumeraciones

setXincludeSupport
`public void setXIncludeSupport(boolean support)`
Habilita o deshabilita el uso de elementos XInclude. El valor `true` habilita el uso de elementos XInclude; `false` lo deshabilita. El valor predeterminado es `false`.
Parámetros: `support`: toma el booleano `true` o `false`.

Subir | Métodos | Enumeraciones

setXMLValidationMode
`public void setXMLValidationMode(ENUMXMLValidationMode mode)`
Establece el modo de validación XML, que es un literal de enumeración de `ENUMXMLValidationMode`.
Parámetros: `mode`: es un literal de enumeración de `ENUMXMLValidationMode`.

Subir | Métodos | Enumeraciones

setXQueryFileName
`public void setXQueryFileName(String queryFile)`
Establece la ubicación en forma de URL del archivo XQuery que debe ejecutarse.
Parámetros: `queryFile`: la cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo XML que se debe usar.

Subir | Métodos | Enumeraciones

setXQueryFromText
`public void setXQueryFromText(String queryText)`
Aporta el contenido del documento XQuery en forma de texto.
Parámetros: `queryText`: la cadena dada es el documento XQuery que se debe procesar.

Subir | Métodos | Enumeraciones

setXSDVersion
`public void setXSDVersion(ENUMXSDVersion version)`
Establece la versión de XML Schema con la que se debe validar el documento XML.
Parámetros: `version`: es un literal de enumeración de `ENUMXSDVersion`.

Subir | Métodos | Enumeraciones

Enumeraciones
ENUMXQueryVersion

ENUMXQueryVersion

```
public enum ENUMXQueryVersion {
       eVersion10
       eVersion30 }
```

`ENUMXQueryVersion` toma uno de estos literales de enumeración: `eVersion10`, `eVersion30`. Determinan la versión de XQuery que se debe usar para el procesamiento (ejecución o validación).

Utilizado por (Interfaz::Método):
`XQuery` `setVersion`

Subir | Métodos | Enumeraciones

4.2.5 RaptorXMLException

```
public interface RaptorXMLException
```

Descripción
Tiene un solo método que genera la excepción.

RaptorXMLException
```
public void RaptorXMLException(String message)
```
Genera una excepción con información sobre el error que se produce durante el procesamiento.

Parámetros: `message`: una cadena con información sobre el error.

Altova RaptorXML Development Edition 2014

Interfaces COM y .NET

5 Interfaces COM y .NET

Dos interfaces, una sola API
Las interfaces COM y .NET de RaptorXML Development Edition utilizan una sola API: la API de COM/.NET API para RaptorXML Development Edition. La interfaz .NET está construida como un contenedor alrededor de la interfaz COM.

Puede usar RaptorXML con:

- Lenguajes de scripting como JavaScript, a través de la interfaz COM
- Lenguajes de programación como C#, a través de la interfaz .NET

Apartados de esta sección
Esta sección se divide en varios apartados:

- Notas sobre la interfaz COM: funcionamiento general de la interfaz COM y pasos necesarios para trabajar con ella.
- Notas sobre la interfaz .NET: cómo preparar el entorno de desarrollo para trabajar con la interfaz .NET.
- Lenguajes de programación: fragmentos de código en los lenguajes de programación más frecuentes, que muestran cómo se llama a las funciones de RaptorXML.
- Referencia de API: documentación del modelo de objetos, de los objetos y de las propiedades de la API.

5.1 Notas sobre la interfaz COM

Durante la instalación, RaptorXML Development Edition se registra automáticamente como objeto de servidor COM. Esto permite invocar a RaptorXML Development Edition desde otras aplicaciones y lenguajes de scripting compatibles con el uso de llamadas COM. Si quiere cambiar la ubicación del paquete de instalación de RaptorXML Development Edition, es mejor desinstalar RaptorXML Development Edition y volver a instalarlo en su nueva ubicación. De este modo el programa de instalación se encarga del proceso de registro.

Comprobar si RaptorXML Development Edition se registró correctamente
Si RaptorXML Development Edition se registró correctamente, el registro incluye las clases `RaptorXMLDev.Application`. Estas clases suelen estar en `HKEY_LOCAL_MACHINE\SOFTWARE\Classes`.

Código de ejemplo
El ejemplo de código VBScript muestra cómo se puede usar la API de RaptorXML a través de su interfaz COM. Para este fragmento de código también ofrecemos un archivo de ejemplo en la carpeta `examples/API` de la carpeta de aplicación de RaptorXML.

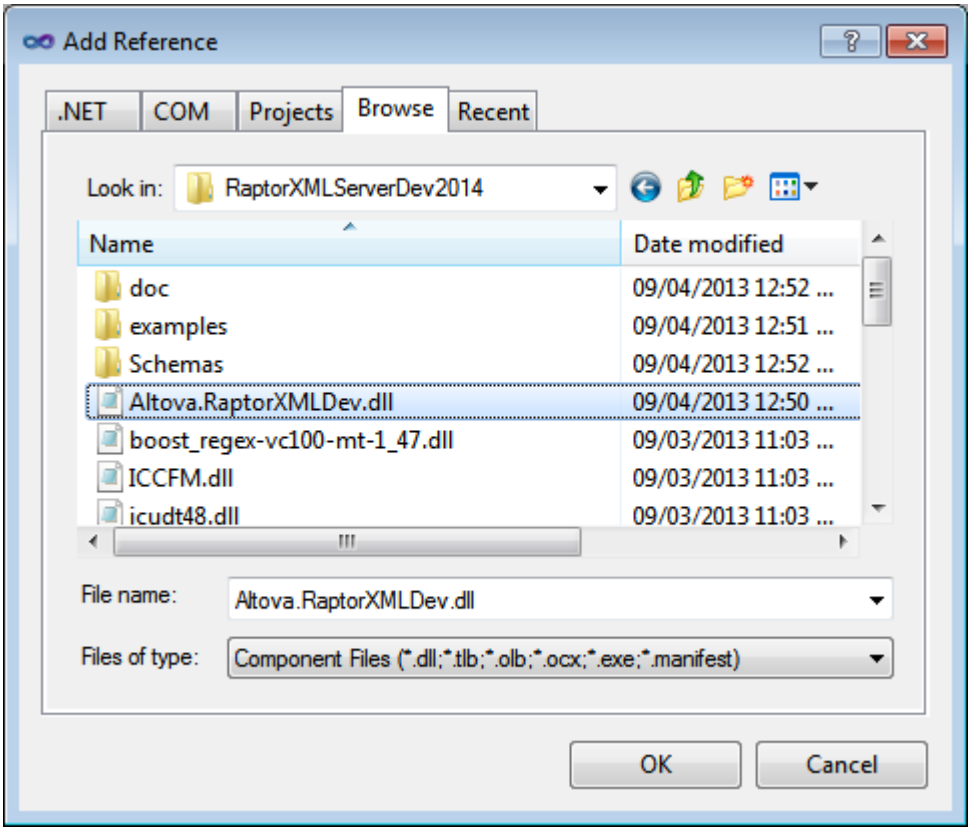
5.2 Notas sobre la interfaz NET

La interfaz .NET está construida como contenedor alrededor de la interfaz COM de RaptorXML. Se ofrece como ensamblado de interoperabilidad principal firmado por Altova y usa el espacio de nombres `Altova.RaptorXMLDev`.

Añadir al proyecto Visual Studio .NET una referencia al DLL de RaptorXML
Para poder usar RaptorXML en su proyecto .NET debe añadir una referencia al DLL de RaptorXML (`Altova.RaptorXMLDev.dll`) en el proyecto. El paquete de instalación de RaptorXML Development Edition contiene un archivo DLL firmado y llamado `Altova.RaptorXMLDev.dll`, que se añadirá automáticamente al caché global de ensamblados cuando se instale RaptorXML usando el programa de instalación de RaptorXML. (Su ubicación suele ser la carpeta `C:\WINDOWS\assembly`.)

Para añadir una referencia a este DLL en un proyecto .NET:

1. Abra el proyecto .NET y haga clic en **Proyecto | Añadir referencia**. Aparece el cuadro de diálogo "Añadir referencia" (*imagen siguiente*).



2. En la pestaña *Browse* vaya a la carpeta: *<RaptorXML application folder>*, seleccione el DLL de RaptorXML `Altova.RaptorXMLDev.dll` y haga clic en **Aceptar**.
3. Seleccione el comando **View | Object Browser** para ver los objetos de la API de RaptorXML.

Cuando `Altova.RaptorXMLDev.dll` esté a disposición de la interfaz .NET y RaptorXML esté

registrado como objeto servidor COM, las funciones de RaptorXML estarán disponibles en el proyecto .NET.

Nota: RaptorXML se registra automáticamente como objeto servidor COM durante la instalación. No es necesario registrarlo a mano.

Nota: si recibe un error de acceso, compruebe que tiene los permisos necesarios. Vaya a Servicios de componentes y otorge permisos a la misma cuenta que ejecuta el grupo de aplicaciones que contiene RaptorXML.

Código de ejemplo

El ejemplo de código C# y de código Visual Basic .NET muestran cómo se puede usar la API de RaptorXML a través de su interfaz .NET. Para estos ejemplos de código también ofrecemos archivos de muestra en la carpeta `examples/API` de la carpeta de aplicación de RaptorXML.

5.3 Lenguajes de programación

No todos los lenguajes de programación ofrecen la misma compatibilidad con COM y .NET. A continuación puede ver algunos ejemplos para los lenguajes de programación más frecuentes. Los fragmentos de código que aparecen en este apartado explican cómo acceder a funciones básicas. Estas funciones básicas están incluidas en los archivos de la carpeta `examples/API` de la carpeta de aplicación RaptorXML Development Edition.

VBScript
VBScript se puede usar para acceder a la API de COM de RaptorXML Development Edition. El ejemplo de código VBScript demuestra estas funciones básicas:

- Conectarse a la API de COM de RaptorXML Development Edition
- Validar un archivo XML
- Realizar una transformación XSLT
- Ejecutar un XQuery

C#
C# se puede usar para acceder a la API de .NET de RaptorXML Development Edition. El ejemplo de código C# demuestra cómo acceder a la API para utilizar estas funciones básicas:

- Conectarse a la API de .NET de RaptorXML Development Edition
- Validar un archivo XML
- Realizar una transformación XSLT
- Ejecutar un XQuery

Visual Basic .NET
Visual Basic.NET solo difiere de C# en la sintaxis. El acceso a la API de .NET funciona exactamente igual. El ejemplo de código Visual Basic demuestra estas funciones básicas:

- Conectarse a la API de .NET de RaptorXML Development Edition
- Validar un archivo XML
- Realizar una transformación XSLT
- Ejecutar un XQuery

Este apartado contiene estos ejemplos:

Para la interfaz COM

- Un ejemplo de código VBScript

Para la interfaz .NET

- Un ejemplo de código C#
- Un ejemplo de código Visual Basic

5.3.1 Ejemplo de COM: VBScript

Este ejemplo de código VBScript se divide en varias partes:

- Preparación e inicialización del objeto COM de RaptorXML
- Validación de un archivo XML
- Transformación XSLT y devolución del resultado en forma de cadena de texto
- Ejecución de un XQuery y devolución del resultado en forma de cadena de texto
- Preparación de la secuencia de ejecución del script y su punto de entrada

---

```
' El objeto COM de RaptorXML
option explicit

dim objRaptor
dim sExampleFolder


sub Init
	Dim sObjectName
	' Buscar las muestras instaladas con el producto (está dos niveles más
arriba, si la interfaz se inició desde examples/API/Java)
	sExampleFolder = left ( WScript.ScriptFullName, ( len (
WScript.ScriptFullName ) ) - ( len( WScript.ScriptName ) ) ) & "..\..\"

	' ¿Esta es la edición Development Edition?
	if ( InStr( sExampleFolder, "\RaptorXMLServerDev" ) > 0 ) then
		sObjectName = "DevelopmentEdition"
	else
		sObjectName = "Server"
	end if

	objRaptor = Null
	On Error Resume Next
	' Intentar cargar el objeto COM de 32 bits; no emitir ninguna excepción
si no se encuentra el objeto
	Set objRaptor = WScript.GetObject( "", "RaptorXML." & sObjectName )
	On Error Goto 0
	if ( IsNull( objRaptor ) ) then
		' Intentar cargar el objeto COM de 64 bits (se emite una
excepción si el objeto no se encuentra)
		Set objRaptor = WScript.GetObject( "", "RaptorXML_x64." &
sObjectName )
	end if
	objRaptor.ErrorLimit = 3
	objRaptor.ReportOptionalWarnings = true
end sub
```

```
' Validar un archivo
sub ValidateXML
	dim objXMLValidator, bResult
	Set objXMLValidator = objRaptor.GetXMLValidator

	objXMLValidator.InputXMLFromText = "<!DOCTYPE root [ <!ELEMENT root
(#PCDATA)> ]> <root>simple input document</root>"
	bResult = objXMLValidator.IsWellFormed
	if ( bResult ) then
		MsgBox( "La cadena de entrada tiene un formato XML correcto" )
	else
		MsgBox( objXMLValidator.LastErrorMessage )
```

```
	end if
	bResult = objXMLValidator.IsValid
	if ( bResult ) then
		MsgBox( "La cadena de entrada es válida" )
	else
		MsgBox( objXMLValidator.LastErrorMessage )
	end if
end sub
```

```
' Realizar una transformación; devolver el resultado en forma de cadena de
texto
sub RunXSLT
	dim objXSLT
	set objXSLT = objRaptor.GetXSLT
	objXSLT.InputXMLFileName = sExampleFolder & "simple.xml"
	objXSLT.XSLFileName = sExampleFolder & "transformar.xsl"
	MsgBox( objXSLT.ExecuteAndGetResultAsString() )
end sub
```

```
' Ejecutar un XQuery; devolver el resultado en forma de cadena de texto
sub RunXQuery
	dim objXQ
	set objXQ = objRaptor.GetXQuery()
	objXQ.InputXMLFileName = sExampleFolder & "simple.xml"
	objXQ.XQueryFileName = sExampleFolder & "CopyInput.xq"
	MsgBox( objXQ.ExecuteAndGetResultAsString() )
end sub
```

```
' Realizar todas las funciones de muestra
sub main
	Init
	ValidateXML
	RunXSLT
	RunXQuery
end sub

' Punto de entrada del script; ejecutar la función principal
main
```

5.3.2 Ejemplo de .NET: C#

Este ejemplo de código C# se divide en varias partes:

- Preparación e inicialización del objeto .NET de RaptorXML
- Validación de un archivo XML
- Transformación XSLT y devolución del resultado en forma de cadena de texto
- Ejecución de XQuery y devolución del resultado en forma de cadena de texto
- Preparación de la secuencia de ejecución del código y de su punto de entrada

---

```
using System;
using System.Text;
using Altova.RaptorXMLDev;

namespace RaptorXMLRunner
{
    class Program
    {
      static ApplicationClass app;
      static string exampleFolder;
      static void Init()
      {
      // Ruta del EXE:
<dirInstalación>\examples\API\C#\RaptorXMLRunner\bin\<config>
      // Subir 5 niveles para buscar los archivos de ejemplo
         exampleFolder = System.IO.Path.GetDirectoryName( System.Reflection.
Assembly.GetExecutingAssembly().Location ) + "\\..\\..\\..\\..\\..\\";

      // Asignar un objeto AltovaXML Application
         app = new ApplicationClass();
         app.ErrorLimit = 3;
         app.ReportOptionalWarnings = true;
      }
```

```
      // Validar un archivo
         static void ValidateXML()
         {
             XMLValidator objXMLValidator = app.GetXMLValidator();
             objXMLValidator.InputXMLFromText = "<!DOCTYPE root [ <!ELEMENT
root (#PCDATA)> ]> <root>simple input document</root>";
             if ( objXMLValidator.IsWellFormed() )
                Console.WriteLine( "La cadena de entrada tiene un formato XML
correcto" );
             else
                Console.WriteLine( objXMLValidator.LastErrorMessage );

             if ( objXMLValidator.IsValid() )
                Console.WriteLine( "la cadena de entrada es válida" );
             else
                Console.WriteLine( objXMLValidator.LastErrorMessage );
         }
```

```
      // Realizar una transformación; devolver el resultado como cadena de
texto
         static void RunXSLT()
         {
             XSLT objXSLT = app.GetXSLT();
             objXSLT.InputXMLFileName = exampleFolder + "simple.xml";
             objXSLT.XSLFileName = exampleFolder + "transform.xsl";
             Console.WriteLine( objXSLT.ExecuteAndGetResultAsString() );
         }
```

```
      // Ejecutar un XQuery; devolver el resultado como cadena de texto
         static void RunXQuery()
         {
             XQuery objXQuery = app.GetXQuery();
             objXQuery.InputXMLFileName = exampleFolder + "simple.xml";
             objXQuery.XQueryFileName = exampleFolder + "CopyInput.xq";
             Console.WriteLine( objXQuery.ExecuteAndGetResultAsString() );
         }
```

```
         static void Main(string[] args)
         {
             try
             {
                 Init();
                 ValidateXML();
                 RunXSLT();
                 RunXQuery();
             }
             catch (System.Exception ex)
             {
                 Console.WriteLine( ex.Message );
                 Console.WriteLine( ex.ToString() );
             }
         }
    }
}
```

5.3.3 Ejemplo de .NET: Visual Basic .NET

Este ejemplo de código Visual Basic se divide en varias partes:

- Preparación e inicialización del objeto .NET de RaptorXML
- Validación de un archivo XML
- Transformación XSLT y devolución del resultado en forma de cadena de texto
- Ejecución de XQuery y devolución del resultado en forma de cadena de texto
- Preparación de la secuencia de ejecución del código y de su punto de entrada

---

```
Option Explicit On
Imports Altova.RaptorXMLDev

Module RaptorXMLRunner


   Dim objRaptor As ApplicationClass
   Dim sExampleFolder As String

   Sub Init()
   ' la ruta de acceso del ejecutable debería ser
<dirInstalación>\examples\API\C#\RaptorXMLRunner\bin\<config> ; subir 5
niveles para encontrar los archivos de ejemplo

   Dim exeURI As System.Uri
       exeURI = New
System.Uri(System.Reflection.Assembly.GetExecutingAssembly().CodeBase)
       sExampleFolder = System.IO.Path.GetDirectoryName(exeURI.LocalPath) &
"\..\..\..\..\..\"

       objRaptor = New ApplicationClass()
       objRaptor.ErrorLimit = 3
       objRaptor.ReportOptionalWarnings = True
   End Sub
```

```
   ' Validar un archivo
     Sub ValidateXML()
     Dim objXMLValidator As XMLValidator
         objXMLValidator = objRaptor.GetXMLValidator()

         objXMLValidator.InputXMLFromText = "<!DOCTYPE root [ <!ELEMENT root
(#PCDATA)> ]> <root>simple input document</root>"

         If (objXMLValidator.IsWellFormed()) Then
             Console.WriteLine("La cadena de entrada tiene un formato XML
correcto")
         Else
             Console.WriteLine(objXMLValidator.LastErrorMessage)
         End If
         If (objXMLValidator.IsValid()) Then
             Console.WriteLine("La cadena de entrada es válida")
         Else
             Console.WriteLine(objXMLValidator.LastErrorMessage)
         End If
     End Sub
```

```
   ' Realizar una transformación; devolver el resultado en forma de cadena de
```

```
texto
       Sub RunXSLT()
          Dim objXSLT As XSLT
          objXSLT = objRaptor.GetXSLT()
          objXSLT.InputXMLFileName = sExampleFolder & "simple.xml"
          objXSLT.XSLFileName = sExampleFolder & "transform.xsl"
          Console.WriteLine(objXSLT.ExecuteAndGetResultAsString())
     End Sub
```

```
   ' Ejecutar un XQuery; devolver el resultado en forma de cadena de texto
     Sub RunXQuery()
         Dim objXQ As XQuery
         objXQ = objRaptor.GetXQuery()
         objXQ.InputXMLFileName = sExampleFolder & "simple.xml"
         objXQ.XQueryFileName = sExampleFolder & "CopyInput.xq"
         Console.WriteLine(objXQ.ExecuteAndGetResultAsString())
     End Sub
```

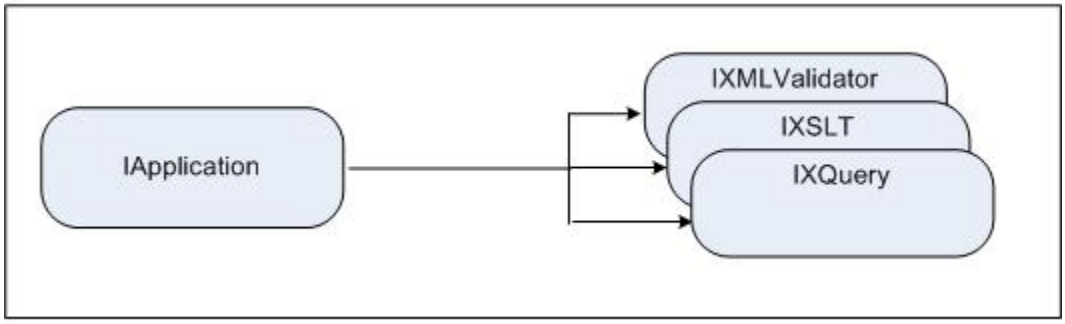
```
   ' Punto de entrada; realizar todas las funciones de muestra
     Sub Main()
         Init()
         ValidateXML()
         RunXSLT()
         RunXQuery()
     End Sub

End Module
```

5.4 Referencia de API

Esta sección describe la especificación de la API: su modelo de objetos y los detalles de sus interfaces y enumeraciones.

El punto de partida para usar las funciones de RaptorXML es la interfaz `IApplication`. Este objeto contiene los objetos que ofrecen las funciones de RaptorXML: validación XML, transformaciones XSLT y procesamiento de documentos XQuery. El modelo de objetos de la API de RaptorXML aparece en este esquema:



La jerarquía del modelo de objetos aparece más abajo y las interfaces se describen una por una en los apartados siguientes. Los métodos y propiedades de cada interfaz también se describen en detalle.

```
-- IApplication
  |-- IXMLValidator
  |-- IXSLT
  |-- IXQuery
```

5.4.1 Interfaces

Las interfaces definidas son estas:

IApplication
IXMLValidator
IXSLT
IXQuery

IApplication

La interfaz **IApplication** ofrece métodos para devolver interfaces del motor de RaptorXML correspondiente: del validador XML, del motor XSLT o del motor XQuery. Las propiedades definen los parámetros de la interfaz.

Métodos
IXMLValidator
IXSLT
IXQuery

Propiedades	
APIMajorVersion	Is64Bit
APIMinorVersion	MajorVersion
APIServicePackVersion	MinorVersion
ErrorFormat	ProductName
ErrorLimit	ProductNameAndVersion
GlobalCatalog	ReportOptionalWarnings
GlobalResourceConfig	ServicePackVersion
GlobalResourcesFile	UserCatalog

Métodos

Los métodos de la interfaz **IApplication** devuelven interfaces del motor de RaptorXML correspondiente: del validador XML, del motor XSLT o del motor XQuery.

IXMLValidator **GetXMLValidator()** [Subir | Métodos | Propiedades]
Devuelve una instancia del motor del validador XML.

IXSLT **GetXSLT()** [Subir | Métodos | Propiedades]
Devuelve una instancia del motor XSLT.

IXQuery **GetXQuery()** [Subir | Métodos | Propiedades]
Devuelve una instancia del motor XQuery.

Propiedades
Las propiedades de la interfaz **IApplication** se describen a continuación por orden alfabético. La tabla organiza las propiedades por grupos para facilitar su consulta. Recuerde que las cadenas de entrada que se deben interpretar como direcciones URL deben indicar rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

Errores y advertencias	Catálogos	Recursos globales
ErrorFormat	GlobalCatalog	GlobalResourceConfig
ErrorLimit	UserCatalog	GlobalResourcesFile
ReportOptionalWarnings		

Información sobre el producto	
ProductName	Is64Bit
ProductNameAndVersion	APIMajorVersion
MajorVersion	APIMinorVersion
MinorVersion	APIServicePackVersion
ServicePackVersion	

int **APIMajorVersion** [Subir | Métodos | Propiedades]
Devuelve la versión principal de la API como número entero. La versión principal de la API puede ser distinta a la versión principal del producto si la API está conectada a otro servidor.

int **APIMinorVersion** [Subir | Métodos | Propiedades]
Devuelve la versión secundaria de la API como número entero. La versión secundaria de la API puede ser distinta a la versión secundaria del producto si la API está conectada a otro servidor.

int **APIServicePackVersion** [Subir | Métodos | Propiedades]
Devuelve la versión del service pack de la API como número entero. La versión del service pack de la API puede ser distinta a la versión del service pack del producto si la API está conectada a otro servidor.

ENUMErrorFormat **ErrorFormat** [Subir | Métodos | Propiedades]
Establece el formato de los errores de RaptorXML y es un literal de ENUMErrorFormat (Text | ShortXML | LongXML).

---

uint **ErrorLimit** [Subir | Métodos | Propiedades]
Configura el límite de errores de validación de RaptorXML. Es de tipo uint. Si se alcanza el límite de errores, la ejecución se detiene. El valor predeterminado es 100.

---

string **GlobalCatalog** [Subir | Métodos | Propiedades]
Indica la ubicación del archivo de catálogo principal (punto de entrada). La cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo de catálogo que se debe usar.

---

string **GlobalResourceConfig** [Subir | Métodos | Propiedades]
Indica la configuración activa del recurso global que se debe utilizar.

---

string **GlobalResourcesFile** [Subir | Métodos | Propiedades]
Indica el archivo de recurso global. La cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo de recurso global que se debe usar.

---

bool **Is64Bit** [Subir | Métodos | Propiedades]
Comprueba si la aplicación es un ejecutable de 64 bits. *Ejemplo:* para Altova RaptorXML Development Edition 2014r2sp1(x64), devuelve true.

---

int **MajorVersion** [Subir | Métodos | Propiedades]
Devuelve la versión principal del producto como número entero. *Ejemplo:* para Altova RaptorXML Development Edition 2014r2sp1(x64), devuelve 16 (la diferencia entre la versión principal (2014) y el año inicial 1998).

---

int **MinorVersion** [Subir | Métodos | Propiedades]
Devuelve la versión secundaria del producto como número entero. *Ejemplo:* para Altova RaptorXML Development Edition 2014r2sp1(x64), devuelve 2 (del número de versión secundaria r2).

---

string **ProductName** [Subir | Métodos | Propiedades]
Devuelve el nombre del producto como cadena de texto. *Ejemplo:* para Altova RaptorXML Development Edition 2014r2sp1(x64), devuelve Altova RaptorXML Development Edition.

---

string **ProductNameAndVersion** [Subir | Métodos | Propiedades]

Devuelve el nombre y la versión del producto como cadena de texto. *Ejemplo:* para `Altova RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `Altova RaptorXML Development Edition 2014r2sp1(x64)`.

`bool` **ReportOptionalWarnings** [Subir | Métodos | Propiedades]
Habilita o deshabilita la notificación de advertencias. El valor `true` habilita las advertencias; `false` las deshabilita.

`int` **ServicePackVersion** [Subir | Métodos | Propiedades]
Devuelve la versión del service pack del producto como número entero. *Ejemplo:* para `RaptorXML Development Edition 2014r2sp1(x64)`, devuelve `1` (del número del service pack `sp1`).

`string` **UserCatalog** [Subir | Métodos | Propiedades]
Indica, en forma de URL, la ubicación del archivo de catálogo definido por el usuario. La cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo de catálogo del usuario que se debe usar.

IXMLValidator

La interfaz **`IXMLValidator`** ofrece métodos para probar:

- La validez de un documento XML, DTD o XSD: `IsValid`. los documentos XML se pueden validar con un DTD o con un esquema XML y las referencia al DTD/esquema pueden estar dentro del documento XML o en el código.
- Si el formato de un documento XML es correcto: `IsWellFormed`.

Ambos métodos devuelven el valor booleano `TRUE` o `FALSE`. Las propiedades definen los parámetros de la interfaz.

Métodos
IsValid
IsWellFormed

Propiedades		
AssessmentMode	InputXMLFromText	SchemalocationHints
DTDFileName	LastErrorMessage	SchemaMapping
DTDFromText	PythonScriptFile	SchemaTextArray
EnableNamespaces	SchemaFileArray	Streaming
InputFileArray	SchemaFileName	XincludeSupport
InputTextArray	SchemaFromText	XMLValidationMode
InputXMLFileName	SchemaImports	XSDVersion

Métodos

Los dos métodos de la interfaz **IXMLValidator** son **IsValid** y **IsWellFormed**. Prueban si el documento especificado es válido y si tiene un formato XML correcto respectivamente. Ambos métodos devuelven el valor booleano `true` o `false`.

`bool` **IsValid(**ENUMValidationType `nType`**)** [Subir | Métodos | Propiedades]

- Devuelve el resultado de la validación indicada con el valor de ENUMValidationType. Devuelve `true` si el documento es válido y `false` si no lo es.
- `nType` es el valor de ENUMValidationType. El tipo de validación indica si el XML se debe validar con un DTD o un XSD. O si se debe validar un DTD o un XSD. El valor predeterminado es `eValidateAny`, lo cual indica que RaptorXML debe determinar el tipo de documento automáticamente.
- Si se produce un error durante la ejecución, se emite una excepción `RaptorXMLException`. Use la operación LastErrorMessage para obtener más información.

`bool` **IsWellFormed(**ENUMWellformedCheckType `nType`**)** [Subir | Métodos | Propiedades]

- Devuelve el resultado de la comprobación de formato XML indicada con el valor de ENUMWellformedCheckType. Devuelve `true` si el formato es correcto y `false` si no lo es.
- `nType` es el valor de ENUMWellformedCheckType. Su valor indica si se debe revisar el formato de un documento XML o de un documento DTD. El valor predeterminado es `eWellformedAny`.
- Si se produce un error durante la ejecución, se emite una excepción `RaptorXMLException`. Use la operación LastErrorMessage para obtener más información.

Propiedades

Las propiedades de la interfaz **IXMLValidator** se describen a continuación por orden alfabético. La tabla organiza las propiedades por grupos para facilitar su consulta. Recuerde que las cadenas de entrada que deben interpretarse como direcciones URL deben dar rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

Archivos de datos	Archivos de esquema	Procesamiento
InputFileArray	DTDFileName	AssessmentMode
InputTextArray	DTDFromText	EnableNamespaces
InputXMLFileName	SchemaFileArray	LastErrorMessage
InputXMLFromText	SchemaFileName	PythonScriptFile
	SchemaFromText	Streaming
	SchemaImports	XincludeSupport
	SchemalocationHints	XMLValidationMode
	SchemaMapping	XSDVersion
	SchemaTextArray	

ENUMAssessmentMode **AssessmentMode** [Subir | Métodos | Propiedades]
Establece el modo de evaluación del validador XML (strict o lax), según lo indicado por los literales de ENUMAssessmentMode.

`string` **DTDFileName** [Subir | Métodos | Propiedades]
Indica qué documento DTD externo se debe usar para la validación. La cadena indicada debe ser una URL absoluta que dé la ubicación base del DTD que se debe usar.

`string` **DTDFromText** [Subir | Métodos | Propiedades]
Ofrece todo el DTD en forma de cadena de texto.

`bool` **EnableNamespaces** [Subir | Métodos | Propiedades]
Habilita el procesamiento preparado para espacios de nombres. Es útil si quiere buscar errores en la instancia XML derivados del uso incorrecto de espacios de nombres. El valor `true` habilita el procesamiento preparado para espacios de nombres; `false` lo deshabilita. El predeterminado es `false`.

`object` **InputFileArray** [Subir | Métodos | Propiedades]
Ofrece una lista de las direcciones URL de los archivos XML que se deben usar como datos de entrada. La propiedad ofrece un objeto que contiene, como cadenas de texto, las URL absolutas de los archivos XML.

`object` **InputTextArray** [Subir | Métodos | Propiedades]
Ofrece una lista de las direcciones URL de los archivos de texto que se deben usar como datos de entrada. La propiedad ofrece un objeto que contiene, como cadenas de texto, las URL absolutas de los archivos de texto.

`string` **InputXMLFileName** [Subir | Métodos | Propiedades]

Indica qué archivo XML se debe validar. La cadena indicada debe ser una URL absoluta que dé la ubicación base del archivo XML que se debe usar.

`string` **InputXMLFromText** [Subir | Métodos | Propiedades]
Suministra, como cadena de texto, el contenido del documento XML que se debe validar.

`string` **LastErrorMessage** [Subir | Métodos | Propiedades]
Recupera el último mensaje de error del motor de RaptorXML como cadena de texto.

`string` **PythonScriptFile** [Subir | Métodos | Propiedades]
Especifica el archivo de script Python que incluye un procesamiento adicional del archivo XML o XSD indicado para la validación. La cadena indicada debe ser una URL absoluta que dé la ubicación base del script de Python.

`object` **SchemaFileArray** [Subir | Métodos | Propiedades]
Ofrece una lista de las URL de los archivos XSD que se deben usar como esquemas XML externos. La propiedad ofrece un objeto que contiene, como cadenas de texto, las URL absolutas de los archivos de esquema XML.

`string` **SchemaFileName** [Subir | Métodos | Propiedades]
Indica qué esquema XML externo se debe usar para la validación. La cadena indicada debe ser una URL absoluta que dé la ubicación base del archivo de esquema XML que se debe usar.

`string` **SchemaFromText** [Subir | Métodos | Propiedades]
Suministra, como cadena de texto, el contenido del archivo de esquema XML que se debe usar para la validación.

ENUMSchemaImports **SchemaImports** [Subir | Métodos | Propiedades]
Indica qué se debe hacer con las importaciones de esquema, dependiendo del valor de los atributos de los elementos `xs:import`. Esto viene dado por el literal de ENUMSchemaImports seleccionado.

ENUMLoadSchemalocation **SchemalocationHints** [Subir | Métodos | Propiedades]
Especifica el mecanismo que se debe usar para encontrar el esquema. El mecanismo viene dado por el literal de ENUMLoadSchemalocation seleccionado.

ENUMSchemaMapping **SchemaMapping** [Subir | Métodos | Propiedades]
Establece qué asignación se usa para encontrar el esquema. La asignación viene dada por el literal de ENUMSchemaMapping seleccionado.

`object` **SchemaTextArray** [Subir | Métodos | Propiedades]
Ofrece un conjunto de cadenas de texto, que son los archivos XSD que se deben usar como esquemas XML externos. La propiedad ofrece un objeto que contiene, como cadenas de texto, la cadena de texto correspondiente a cada archivo XSD.

`bool` **Streaming** [Subir | Métodos | Propiedades]
Habilita la validación por transmisión de secuencias de datos. En el modo de transmisión por secuencias, los datos almacenados en memoria se minimizan y el procesamiento es más rápido. El valor `true` habilita la validación por transmisión de secuencias; `false` la deshabilita. El valor predeterminado es `true`.

`bool` **XincludeSupport** [Subir | Métodos | Propiedades]
Habilita el uso de elementos `XInclude`. El valor `true` habilita los elementos XInclude; `false` los deshabilita. El valor predeterminado es `false`.

ENUMXMLValidationMode **XMLValidationMode** [Subir | Métodos | Propiedades]
Establece el modo de validación XML (validación o comprobación de formato XML). El modo es el indicado por el literal de ENUMXMLValidationMode.

ENUMXSDVersion **XSDVersion** [Subir | Métodos | Propiedades]
Indica la versión de XML Schema que se debe usar para validar el documento XML. Sus valores son los literales de ENUMXSDVersion.

IXSLT

La interfaz **IXSLT** ofrece métodos y propiedades para ejecutar una transformación XSLT 1.0, 2.0 o 3.0. Los resultados se pueden guardar en un archivo o devolverse como cadena de texto. La interfaz también permite pasar parámetros XSLT a la hoja de estilos XSLT. Las direcciones URL de los archivo XML y XSLT se pueden dar como cadenas de texto a través de las propiedades de la interfaz. Otra opción es construir documentos XML y XSLT dentro del código como cadenas de texto.

Nota: Cuando las cadenas de texto de entrada se deben interpretar como URL, deben utilizarse rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

Nota: Puede usar el motor XSLT 2.0 o 3.0 de RaptorXML en el modo compatible con versiones anteriores para procesar hojas de estilos XSLT 1.0. No obstante, el resultado puede ser diferente al que se obtendría con el motor XSLT 1.0.

Métodos
IsValid
Execute

ExecuteAndGetResultAsString
ExecuteAndGetResultAsStringWithBaseOutputURI
AddExternalParameter
ClearExternalParameterList

Propiedades		
ChartExtensionsEnabled	JavaBarcodeExtensionLocation	SchemaMapping
DotNetExtensionsEnabled	JavaExtensionsEnabled	StreamingSerialization
EngineVersion	LastErrorMessage	XincludeSupport
IndentCharacters	LoadXMLWithPSVI	XMLValidationMode
InitialTemplateMode	NamedTemplateEntryPoint	XSDVersion
InputXMLFileName	SchemaImports	XSLFileName
InputXMLFromText	SchemalocationHints	XSLFromText

Métodos

Los métodos de la **IXSLT** se describen a continuación. Recuerde que las cadenas de entrada que deben interpretarse como direcciones URL deben dar rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

Métodos
IsValid
Execute
ExecuteAndGetResultAsString
ExecuteAndGetResultAsStringWithBaseOutputURI
AddExternalParameter
ClearExternalParameterList

bool **IsValid()** [Subir | Métodos | Propiedades]

- Devuelve el resultado de validar la hoja de estilos XSLT de acuerdo con la especificación XSLT indicada en ENUMXSLTVersion (ver la propiedad EngineVersion). El resultado es `true` si el XSLT es válido y `false` si no lo es.
- Si ocurre un error, se emite una excepción `RaptorXMLException`. Use la operación LastErrorMessage para obtener más información.

`bool` **`Execute(string bstrResultFileName)`** [Subir | Métodos | Propiedades]

- Ejecuta la transformación XSLT de acuerdo con la especificación XSLT indicada en `ENUMXSLTVersion` (ver la propiedad `EngineVersion`) y guarda el resultado en un archivo de salida.
- El archivo de salida viene dado por `bstrResultFileName`, que es una cadena que indica la URL del archivo de salida.
- El resultado es `true` si la transformación finaliza correctamente y `false` si se producen errores.
- Si ocurre un error, se emite una excepción `RaptorXMLException`. Use la operación `LastErrorMessage` para obtener más información.

`string` **`ExecuteAndGetResultAsString()`** [Subir | Métodos | Propiedades]

- Ejecuta la transformación XSLT de acuerdo con la especificación indicada en `ENUMXSLTVersion` (ver la propiedad `EngineVersion`) y devuelve el resultado de la transformación como cadena de texto.
- Si ocurre un error durante la transformación, se emite una excepción `RaptorXMLException`. Use la operación `LastErrorMessage` para obtener más información.

`string` **`ExecuteAndGetResultAsStringWithBaseOutputURI(`**`string bstrBaseURI`**`)` [Subir | Métodos | Propiedades]

- Ejecuta la transformación XSLT de acuerdo con la especificación indicada en `ENUMXSLTVersion` (ver la propiedad `EngineVersion`) y devuelve el resultado de la transformación como cadena de texto en la ubicación definida por el URI base (la cadena `bstrBaseURI`).
- Si ocurre un error durante la transformación, se emite una excepción `RaptorXMLException`. Use la operación `LastErrorMessage` para obtener más información.

`void` **`AddExternalParameter(`**`string bstrName, string bstrValue`**`)` [Subir | Métodos | Propiedades]

- Añade el nombre y el valor de un parámetro externo: `bstrName` y `bstrValue` son cadenas.
- Cada parámetro externo y su valor se puede especificar en una llamada distinta al método. Los parámetros se deben declarar en el documento XSLT y si quiere puede añadir una declaración de tipo. Sea cual sea la declaración de tipo del documento XSLT, no es necesario ningún delimitador concreto si el valor del parámetro se indica con `AddExternalParameter`.

`void` **ClearExternalParameterList()** [Subir | Métodos | Propiedades]

- Borra la lista de parámetros externos creada con el método AddExternalParameter.

Propiedades

Las propiedades de la interfaz **IXSLT** se describen a continuación por orden alfabético. La tabla organiza las propiedades por grupos para facilitar su consulta. Recuerde que las cadenas de entrada que deben interpretarse como direcciones URL deben dar rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

XML	XSLT	Schema
InputXMLFileName	EngineVersion	SchemaImports
InputXMLFromText	XSLFileName	SchemalocationHints
LoadXMLWithPSVI	XSLFromText	SchemaMapping
XincludeSupport		XSDVersion
XMLValidationMode		
Procesamiento	**Extensiones**	
IndentCharacters	ChartExtensionsEnabled	
InitialTemplateMode	DotNetExtensionsEnabled	
LastErrorMessage	JavaBarcodeExtensionLocation	
NamedTemplateEntryPoint	JavaExtensionsEnabled	
StreamingSerialization		

`bool` **ChartExtensionsEnabled** [Subir | Métodos | Propiedades]
Habilita o deshabilita las funciones de extensión de Altova para gráficos. El valor `true` habilita las extensiones para gráficos; `false` las deshabilita. El valor predeterminado es `true`.

`bool` **DotNetExtensionsEnabled** [Subir | Métodos | Propiedades]
Habilita o deshabilita las funciones de extensión de Visual Studio .NET. El valor `true` habilita las extensiones .NET; `false` las deshabilita. El valor predeterminado es `true`.

ENUMXSLTVersion **EngineVersion** [Subir | Métodos | Propiedades]
Indica qué versión XSLT se debe usar (1.0, 2.0 o 3.0). El valor de la propiedad es un literal de ENUMXSLTVersion.

`string` **IndentCharacters** [Subir | Métodos | Propiedades]

Establece el carácter que debe utilizarse para la sangría.

`string` **InitialTemplateMode** [Subir | Métodos | Propiedades]
Establece el modo inicial para el procesamiento XSLT. Se procesarán aquellas plantillas cuyo valor de modo sea igual a la cadena indicada.

`string` **InputXMLFileName** [Subir | Métodos | Propiedades]
Especifica la ubicación del archivo XML que se debe transformar. La cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo XML que se debe usar.

`string` **InputXMLFromText** [Subir | Métodos | Propiedades]
Suministra, como cadena de texto, el contenido del archivo XML que se debe transformar.

`string` **JavaBarcodeExtensionLocation** [Subir | Métodos | Propiedades]
Especifica la ubicación del archivo de extensiones para códigos de barras. Consulte este apartado para obtener más información. La cadena indicada debe ser una URL absoluta que dé la ubicación base del archivo que se debe usar.

`bool` **JavaExtensionsEnabled** [Subir | Métodos | Propiedades]
Habilita o deshabilita las extensiones Java. El valor `true` habilita las extensiones Java; `false` las deshabilita. El valor predeterminado es `true`.

`string` **LastErrorMessage** [Subir | Métodos | Propiedades]
Recupera el último mensaje de error del motor de RaptorXML como cadena de texto.

`bool` **LoadXMLWithPSVI** [Subir | Métodos | Propiedades]
Habilita o deshabilita la opción de carga y uso del conjunto Post Schema Validation Infoset (PSVI). Si se carga el PSVI, la información obtenida del esquema se puede usar para calificar datos del documento XML. El valor `true` habilita la carga de PSVI; `false` la deshabilita.

`string` **NamedTemplateEntryPoint** [Subir | Métodos | Propiedades]
Especifica el nombre, como cadena de texto, de la plantilla con nombre que debe utilizarse como punto de entrada de la transformación.

ENUMSchemaImports **SchemaImports** [Subir | Métodos | Propiedades]
Indica qué se debe hacer con las importaciones de esquemas, dependiendo del valor de los atributos de los elementos `xs:import`. Esto viene dado por el literal de ENUMSchemaImports seleccionado.

ENUMLoadSchemalocation **SchemalocationHints** [Subir | Métodos | Propiedades]
Especifica el mecanismo que se debe usar para encontrar el esquema. El mecanismo viene dado por el literal de ENUMLoadSchemalocation seleccionado.

ENUMSchemaMapping **SchemaMapping** [Subir | Métodos | Propiedades]
Establece qué asignación se usa para encontrar el esquema. La asignación viene dada por el literal de ENUMSchemaMapping seleccionado.

---

bool **StreamingSerialization** [Subir | Métodos | Propiedades]
Habilita la serialización de secuencias de datos. En el modo de transmisión por secuencias, los datos almacenados en memoria se minimizan y el procesamiento es más rápido. El valor `true` habilita la serialización de secuencias de datos; `false` la deshabilita.

---

bool **XincludeSupport** [Subir | Métodos | Propiedades]
Habilita el uso de elementos `XInclude`. El valor `true` habilita el uso de XInclude; `false` lo deshabilita. El valor predeterminado es `false`.

ENUMXMLValidationMode **XMLValidationMode** [Subir | Métodos | Propiedades]
Establece el modo de validación XML (validación o comprobación de formato XML). El modo es el indicado por el literal de ENUMXMLValidationMode.

ENUMXSDVersion **XSDVersion** [Subir | Métodos | Propiedades]
Indica la versión de XML Schema que se debe usar para validar el documento XML. Sus valores son los literales de ENUMXSDVersion.

---

string **XSLFileName** [Subir | Métodos | Propiedades]
Especifica qué archivo XSLT se debe usar para la transformación. La cadena indicada debe ser una URL absoluta que dé la ubicación del archivo XSLT que se debe usar.

---

string **XSLFromText** [Subir | Métodos | Propiedades]
Ofrece, en forma de cadena de texto, el contenido del documento XSLT que se debe usar para la transformación.

IXQuery

La interfaz **IXQuery** ofrece métodos y propiedades para ejecutar un documento XQuery 1.0 o XQuery 3.0. Los resultados se pueden guardar en un archivo o devolverse como cadena de texto. La interfaz también permite pasar variables XQuery externas al documento XQuery. Las URL de los archivos XQuery y XML se pueden dar como cadenas de texto a través de las propiedades de la interfaz. Otra opción es construir los documentos XML y XQuery dentro del código como cadenas de texto.

Note: Cuando las cadenas de texto de entrada se deben interpretar como URL, deben utilizarse rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

Métodos
IsValid
Execute
ExecuteAndGetResultAsString
AddExternalVariable
ClearExternalParameterList

Propiedades			
ChartExtensionsEnabled	InputXMLFromText	OutputEncoding	XMLValidationMode
DotNetExtensionsEnabled	JavaBarcodeExtensionLocation	OutputIndent	XQueryFileName
EngineVersion	JavaExtensionsEnabled	OutputMethod	XQueryFromText
IndentCharacters	LastErrorMessage	OutputOmitXMLDeclaration	XSDVersion
InputXMLFileName	LoadXMLWithPSVI	XincludeSupport	

Métodos

Los métodos de la **IXQuery** se describen a continuación. Recuerde que las cadenas de entrada que deben interpretarse como direcciones URL deben dar rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

Métodos
IsValid
Execute
ExecuteAndGetResultAsString
AddExternalVariable
ClearExternalParameterList

bool **IsValid()** [Subir | Métodos | Propiedades]

- Devuelve el resultado de validar el documento XQuery de acuerdo con la especificación XQuery indicada en ENUMXQueryVersion (ver la propiedad EngineVersion). El resultado es `true` si el documento es válido y `false` si no lo es.

- Si ocurre un error, se emite una excepción `RaptorXMLException`. Use la operación `LastErrorMessage` para obtener más información.

`bool` **`Execute(string bstrOutputFile)`** [Subir | Métodos | Propiedades]

- Ejecuta el documento XQuery de acuerdo con la especificación XQuery indicada en `ENUMXQueryVersion` (ver la propiedad `EngineVersion`), y guarda el resultado en un archivo de salida.
- El archivo de salida viene dado por `bstrOutputFile`, que es una cadena que indica la URL del archivo de salida.
- El valor booleano `true` se devuelve si la operación finaliza correctamente y `false` si hay errores.
- Si ocurre un error durante la transformación, se emite una excepción `RaptorXMLException`. Use la operación `LastErrorMessage` para obtener más información.

`string` **`ExecuteAndGetResultAsString()`** [Subir | Métodos | Propiedades]

- Ejecuta la transformación XQuery de acuerdo con la especificación XQuery indicada en `ENUMXQueryVersion` (ver la propiedad `EngineVersion`) y devuelve el resultado de la transformación como cadena de texto.
- Si ocurre un error durante la transformación, se emite una excepción `RaptorXMLException`. Use la operación `LastErrorMessage` para obtener más información.

`void` **`AddExternalVariable(`**`string bstrName, string bstrValue`**`)`** [Subir | Métodos | Propiedades]

- Añade el nombre y valor de una variable externa: `bstrName` y `bstrValue` son cadenas de texto.
- Cada variable externa y su valor deben indicarse en una llamada distinta al método. Las variables deben declararse en el documento XQuery y si quiere puede incluir una declaración de tipo. Sea cual sea la declaración de tipo del documento XQuery, no es necesario incluir ningún delimitador especial si el valor de la variable se indica por medio de `AddExternalVariable`.

`void` **`ClearExternalVariableList()`** [Subir | Métodos | Propiedades]

- Borra la lista de variables externas creadas con el método `AddExternalVariable`.

Propiedades

Las propiedades de la interfaz **IXQuery** se describen a continuación por orden alfabético. La tabla organiza las propiedades por grupos para facilitar su consulta. Recuerde que las cadenas de entrada que deben interpretarse como direcciones URL deben dar rutas de acceso absolutas. Si se usa una ruta de acceso relativa, debe definir en el módulo de llamada un mecanismo para resolver la ruta de acceso relativa.

XML	XQuery	Procesamiento	Extensiones
InputXMLFileName	EngineVersion	IndentCharacters	ChartExtensionsEnabled
InputXMLFromText	XQueryFileName	LastErrorMessage	DotNetExtensionsEnabled
LoadXMLWithPSVI	XQueryFromText	OutputEncoding	JavaBarcodeExtensionLocation
XincludeSupport		OutputIndent	JavaExtensionsEnabled
XMLValidationMode		OutputMethod	
XSDVersion		OutputOmitXMLDeclaration	

`bool` **ChartExtensionsEnabled** [Subir | Métodos | Propiedades]
Habilita o deshabilita las funciones de extensión de Altova para gráficos. El valor `true` habilita las extensiones para gráficos; `false` las deshabilita. El valor predeterminado es `true`.

`bool` **DotNetExtensionsEnabled** [Subir | Métodos | Propiedades]
Habilita o deshabilita las funciones de extensión Visual Studio .NET. El valor `true` habilita las extensiones .NET; `false` las deshabilita. El valor predeterminado es `true`.

ENUMXQueryVersion **EngineVersion** [Subir | Métodos | Propiedades]
Indica qué versión de XQuery se debe usar (1.0 o 3.0). El valor de la propiedad es un literal de ENUMXQueryVersion.

`string` **IndentCharacters** [Subir | Métodos | Propiedades]
Establece el carácter que debe utilizarse para la sangría.

`string` **InputXMLFileName** [Subir | Métodos | Propiedades]
Indica la ubicación del archivo XML que se debe procesar. La cadena indicada debe ser una URL absoluta que dé la ubicación exacta del archivo XML que se debe usar.

`string` **InputXMLFromText** [Subir | Métodos | Propiedades]

Suministra, como cadena de texto, el contenido del archivo XML que se debe procesar.

`string` **JavaBarcodeExtensionLocation** [Subir | Métodos | Propiedades]
Especifica la ubicación del archivo de extensión para códigos de barras. Para más información consulte este apartado. La cadena indicada debe ser una URL absoluta que dé la ubicación base del archivo que se debe usar.

`bool` **JavaExtensionsEnabled** [Subir | Métodos | Propiedades]
Habilita o deshabilita las extensiones Java. El valor `true` habilita las extensiones Java; `false` las deshabilita. El valor predeterminado es `true`.

`string` **LastErrorMessage** [Subir | Métodos | Propiedades]
Recupera el último mensaje de error del motor de RaptorXML como cadena de texto.

`bool` **LoadXMLWithPSVI** [Subir | Métodos | Propiedades]
Habilita o deshabilita la opción de carga y uso del conjunto Post Schema Validation Infoset (PSVI). Si se carga el PSVI, la información obtenida del esquema se puede usar para calificar datos del documento XML. El valor `true` habilita la carga de PSVI; `false` la deshabilita.

`string` **OutputEncoding** [Subir | Métodos | Propiedades]
Establece la codificación del documento de salida. Use un nombre de codificación IANA oficial (p. ej. `UTF-8`, `UTF-16`, `US-ASCII`, `ISO-8859-1`) como cadena de texto.

`bool` **OutputIndent** [Subir | Métodos | Propiedades]
Habilita o deshabilita la sangría en el documento de salida. El valor `true` habilita la sangría; `false` la deshabilita.

`string` **OutputMethod** [Subir | Métodos | Propiedades]
Especifica la serialización del documento de salida. Los valores válidos son: `xml | xhtml | html | text`. El valor predeterminado es `xml`.

`bool` **OutputOmitXMLDeclaration** [Subir | Métodos | Propiedades]
Habilita o deshabilita la inclusión de la declaración XML en el documento de salida. El valor `true` omite la declaración; `false` la incluye. El valor predeterminado es `false`.

`bool` **XincludeSupport** [Subir | Métodos | Propiedades]
Habilita o deshabilita el uso de elementos `XInclude`. El valor `true` habilita el uso de XInclude; `false` lo deshabilita. El valor predeterminado es `false`.

ENUMXMLValidationMode **XMLValidationMode** [Subir | Métodos | Propiedades]
Establece el modo de validación XML (validación o comprobación de formato XML). El modo es el indicado por el literal de ENUMXMLValidationMode.

---

string **XQueryFileName** [Subir | Métodos | Propiedades]
Indica qué archivo XQuery se debe usar. La cadena indicada debe ser una URL absoluta que dé la ubicación del archivo XQuery que se debe usar.

---

string **XQueryFromText** [Subir | Métodos | Propiedades]
Suministra, como cadena de texto, el contenido del archivo XQuery que se debe usar.

ENUMXSDVersion **XSDVersion** [Subir | Métodos | Propiedades]
Indica la versión de XML Schema que se debe usar para validar el documento XML. Sus valores son los literales de ENUMXSDVersion.

5.4.2 Enumeraciones

Las enumeraciones definidas son estas:

ENUMAssessmentMode
ENUMErrorFormat
ENUMLoadSchemalocation
ENUMQueryVersion
ENUMSchemaImports
ENUMSchemaMapping
ENUMValidationType
ENUMWellformedCheckType
ENUMXMLValidationMode
ENUMXQueryVersion
ENUMXSDVersion
ENUMXSLTVersion

ENUMAssessmentMode

Descripción

Contiene literales de enumeración que definen el modo de evaluación del validador XML: `Strict` o `Lax`.

Utilizada por

Interfaz	*Operación*
IXMLValidator	AssessmentMode

Literales de la enumeración

```
eAssessmentModeStrict        = 0
eAssessmentModeLax           = 1
```

eAssessmentModeStrict
Establece el modo de evaluación de la validez del esquema en `Strict`. Es el valor predeterminado.

eAssessmentModeLax
Establece el modo de evaluación de la validez del esquema en `Lax`.

ENUMErrorFormat

Descripción
Contiene literales de enumeración que especifican el formato de los errores de salida.

Utilizada por

Interfaz	*Operación*
IApplication	ErrorFormat

Literales de la enumeración

```
eFormatText             = 0
eFormatShortXML         = 1
eFormatLongXML          = 2
```

eFormatText
Establece el formato de los errores de salida en `Text`. Es el valor predeterminado.

eFormatShortXML
Establece el formato de los errores de salida en `ShortXML`. Este formato es una versión abreviada del formato `LongXML`.

eFormatLongXML
Establece el formato de los errores de salida en `LongXML`. Este formato es el que ofrece información más detallada.

ENUMLoadSchemalocation

Descripción
Contiene literales de enumeración que indican cómo determinar la ubicación del esquema.

Utilizada por

Interfaz	*Operación*
IXBRL	SchemalocationHints
IXMLValidator	SchemalocationHints
IXSLT	SchemalocationHints

Literales de la enumeración

```
eSHLoadBySchemalocation          = 0
eSHLoadByNamespace               = 1
eSHLoadCombiningBoth             = 2
eSHLoadIgnore                    = 3
```

eSHLoadBySchemalocation
Asigna a Load Schemalocation el valor `LoadBySchemalocation`. Usa la URL de schema location de los atributos `xsi:schemaLocation` y `xsi:noNamespaceSchemaLocation` de los documentos XML o XBRL de instancia. Es el **valor predeterminado**.

eSHLoadByNamespace
Asigna a Load Schemalocation el valor `LoadByNamespace`. Use la parte de espacio de nombres del atributo `xsi:schemaLocation` (en el caso de `xsi:noNamespaceSchemaLocation` es una cadena vacía) y busca el esquema a través de la asignación de catálogo.

eSHLoadCombiningBoth
Asigna a Load Schemalocation el valor `CombiningBoth`. Si el espacio de nombres o la URL tienen una asignación de catálogo, esta asignación de catálogo se utiliza. Si ambas tienen una asignación de catálogo, entonces es el valor del parámetro `ENUMSchemaMapping` lo que decide cuál de las asignaciones se utiliza. Si ninguna tiene una asignación de catálogo, se usa la URL.

eSHLoadIgnore
Asigna a Load Schemalocation el valor `LoadIgnore`. Si el valor del parámetro es `eSHLoadIgnore`, se ignoran los atributos `xsi:schemaLocation` y `xsi:noNamespaceSchemaLocation`.

ENUMQueryVersion

Descripción
Contiene literales de enumeración que especifican la versión XQuery que se debe usar: XQuery 1.0 o 3.0.

Literales de la enumeración

```
eXQVersion10        = 1
eXQVersion30        = 3
```

eXQVersion10
Establece que la versión XQuery es XQuery 1.0.

eXQVersion30
Establece que la versión XQuery es XQuery 3.0.

ENUMSchemaImports

Descripción

Contiene los literales de enumeración que definen el comportamiento de los elementos `xs:import`. El elemento `xs:import` tiene los atributos opcionales `namespace` y `schemaLocation`.

Utilizada por

Interfaz	*Operación*
IXBRL	SchemaImports
IXMLValidator	SchemaImports
IXSLT	SchemaImports

Literales de la enumeración

```
eSILoadBySchemalocation                = 0
eSILoadPreferringSchemalocation        = 1
eSILoadByNamespace                     = 2
eSICombiningBoth                       = 3
eSILicenseNamespaceOnly                = 4
```

eSILoadBySchemalocation

Asigna a Schema Import el valor `LoadBySchemalocation`. El valor del atributo `schemaLocation` se usa para buscar el esquema, teniendo en cuenta las asignaciones de catálogo. Si está presente el atributo `namespace`, se importa el espacio de nombres.

eSILoadPreferringSchemalocation

Asigna a Schema Import el valor `LoadPreferringSchemalocation`. Si está presente el atributo `schemaLocation`, éste se utiliza teniendo en cuenta las asignaciones de catálogo. Si no está presente el atributo `schemaLocation`, se usa el valor del atributo `namespace` a través de una asignación de catálogo. Este literal es el **valor predeterminado** de la enumeración.

eSILoadByNamespace

Asigna a Schema Import el valor `LoadByNamespace`. El valor del atributo `namespace` se utiliza para encontrar el esquema a través de una asignación de catálogo.

eSICombiningBoth

Asigna a Schema Import el valor `CombiningBoth`. Si el atributo `namespace` o `schemaLocation` tiene una asignación de catálogo, esta asignación de catálogo se utiliza. Si ambos tienen una asignación de catálogo, entonces es el valor del parámetro `ENUMSchemaMapping` lo que decide cuál de las asignaciones se utiliza. Si ninguna tiene una asignación de catálogo, se usa el valor del atributo `schemaLocation` (que debería ser una URL).

eSILicenseNamespaceOnly
Asigna a Schema Import el valor `LicenseNamespaceOnly`. El espacio de nombres se importa. No se importa ningún documento de esquema.

ENUMSchemaMapping

Descripción
Contiene los literales de enumeración que definen cuál de las dos asignaciones de catálogo se prefiere: Esta enumeración sirve para eliminar ambigüedades en `ENUMLoadSchemalocation` y `ENUMSchemaImports`.

Utilizada por

Interfaz	*Operación*
IXBRL	SchemaMapping
IXMLValidator	SchemaMapping
IXSLT	SchemaMapping

Literales de la enumeración

```
eSMPreferSchemalocation          = 0
eSMPreferNamespace               = 1
```

eSMPreferSchemalocation
Establece que se debe seleccionar la URL de schema location.

eSMPreferNamespace
Establece que se debe seleccionar el espacio de nombres.

ENUMValidationType

Descripción
Contiene literales de enumeración que definen qué tipo de documento se debe validar.

Utilizada por

Interfaz	*Operación*
IXMLValidator	IsValid

Literales de la enumeración

```
eValidateAny                = 0
eValidateXMLWithDTD         = 1
eValidateXMLWithXSD         = 2
eValidateDTD                = 3
eValidateXSD                = 4
```

eValidateAny
Establece que el tipo de validación es `Any`. Esto valida el documento después de detectar su tipo automáticamente. *(La validación de varios documentos no es compatible con la edición Development Edition.)*

eValidateXMLWithDTD
Establece que el tipo de validación es `XMLWithDTD`. Especifica que el documento XML debe validarse con un documento DTD. *(La validación de varios documentos no es compatible con la edición Development Edition.)*

eValidateXMLWithXSD
Establece que el tipo de validación es `XMLWithXSD`. Especifica que el documento XML debe validarse con un esquema XML. *(La validación de varios documentos no es compatible con la edición Development Edition.)*

eValidateDTD
Establece que el tipo de validación es `ValidateDTD`. Especifica que debe validarse un documento DTD. *(La validación de varios documentos no es compatible con la edición Development Edition.)*

eValidateXSD
Establece que el tipo de validación es `ValidateXSD`. Especifica que debe validarse un documento de XML Schema. *(La validación de varios documentos no es compatible con la edición Development Edition.)*

ENUMWellformedCheckType

Descripción
Contiene los literales de enumeración que definen el tipo de documento cuyo formato XML debe comprobarse: XML o DTD.

Utilizada por

Interfaz	*Operación*
IXMLValidator	IsWellFormed

Literales de la enumeración

```
eWellFormedAny          = 0
eWellFormedXML          = 1
eWellFormedDTD          = 2
```

eWellformedAny
Establece que el tipo de comprobación de formato es `Any`. Comprueba el formato XML del documento tras detectar automáticamente si se trata de un documento XML o DTD. (*La comprobación del formato de varios documentos a la vez no es compatible con la edición Development Edition.)*

eWellformedXML
Establece que el tipo de comprobación de formato es `XML`. Comprueba si formato del documento XML se ajusta a la especificación XML 1.0 o XML 1.1. (*La comprobación del formato de varios documentos a la vez no es compatible con la edición Development Edition.)*

eWellformedDTD
Establece que el tipo de comprobación de formato es `DTD`. Comprueba si formato del documento DTD es correcto. (*La comprobación del formato de varios documentos a la vez no es compatible con la edición Development Edition.)*

ENUMXMLValidationMode

Descripción
Contiene literales de enumeración que definen el modo de procesamiento XML que se debe usar: validación o comprobación de formato.

Utilizada por

Interfaz	*Operación*
IXMLValidator	XMLValidationMode
IXQuery	XMLValidationMode
IXSLT	XMLValidationMode

Literales de la enumeración

```
eXMLValidationModeWF            = 0
eXMLValidationModeID            = 1
eXMLValidationModeValid         = 2
```

eXMLValidationModeWF
Establece que el modo de procesamiento XML es `Wellformed`. Es el valor predeterminado.

eXMLValidationModeID
Solo para uso interno.

eXMLValidationModeValid
Establece que el modo de procesamiento XML es `Validation`.

ENUMXQueryVersion

Descripción
Contiene literales de enumeración que especifican qué versión de XQuery se debe usar: XQuery 1.0 o 3.0.

Utilizada por

Interfaz	*Operación*
IXQuery	EngineVersion

Literales de la enumeración

```
eXQVersion10        = 1
eXQVersion30        = 3
```

eXQVersion10
Establece que la versión XQuery es XQuery 1.0.

eXQVersion30
Establece que la versión XQuery es XQuery 3.0. Es el valor predeterminado.

ENUMXSDVersion

Descripción
Contiene literales de enumeración que indican qué versión de XML Schema se debe usar para la validación: XSD 1.0 o 1.1.

Utilizada por

Interfaz	*Operación*
IXMLValidator	XSDVersion
IXQuery	XSDVersion
IXSLT	XSDVersion

Literales de la enumeración

```
eXSDVersionAuto          = 0
eXSDVersion10            = 1
eXSDVersion11            = 2
```

eXSDVersionAuto
Establece que la versión de XML Schema que se debe usar es `Auto-detect`. La versión XSD se detecta automáticamente tras analizar el documento XSD. Si el atributo `vc:minVersion` del documento XSD tiene el valor `1.1`, se considera que el documento tiene la versión XSD 1.1. Si este atributo tiene cualquier otro valor o si el atributo no está en el documento, se entiende que el documento tiene la versión XSD 1.0.

eXSDVersion10
Establece que la versión de XML Schema que se debe usar es XML Schema 1.0.

eXSDVersion11
Establece que la versión de XML Schema que se debe usar es XML-Schema 1.1.

ENUMXSLTVersion

Descripción
Contiene literales de enumeración que especifican qué versión XSLT se debe usar: XSLT 1.0, 2.0 o 3.0.

Utilizada por

Interfaz	*Operación*
IXSLT	EngineVersion

Literales de la enumeración

```
eVersion10          = 1
eVersion20          = 2
eVersion30          = 3
```

eVersion10
Establece que la versión XSLT que se debe usar es XSLT 1.0.

eVersion20
Establece que la versión XSLT que se debe usar es XSLT 2.0.

`eVersion30`
Establece que la versión XSLT que se debe usar es XSLT 3.0.

Altova RaptorXML Development Edition 2014

Información sobre motores XSLT y XQuery

6 Información sobre motores XSLT y XQuery

Los motores XSLT y XQuery de RaptorXML siguen las especificaciones del W3C y, por tanto, son más estrictos que otros motores anteriores de Altova, como los de las versiones antiguas de XMLSpy y del predecesor de RaptorXML, el procesador descatalogado AltovaXML. Por consiguiente, RaptorXML señala algunos errores leves que antes no se notificaban en la versión anterior de estos motores.

Por ejemplo:

- Se notifica un error de tipo (`err:XPTY0018`) si el resultado de un operador de ruta de acceso contiene tanto nodos como no nodos.
- Se notifica un error de tipo (`err:XPTY0019`) si `E1` en una expresión XPath `E1/E2` no da como resultado una secuencia de nodos.

Si encuentra este tipo de errores, modifique el documento XSLT/XQuery o el documento de instancia según corresponda.

Esta sección describe características relacionadas con la implementación de los motores e incluye estos apartados:

6.1 XSLT 1.0

El motor XSLT 1.0 de RaptorXML cumple con la recomendación XSLT 1.0 del 16 de noviembre de 1999 y con la recomendación XPath 1.0 del 16 de noviembre de 1999, ambas del W3C.

Nota sobre la implementación
Cuando el atributo `method` de `xsl:output` tiene el valor HTML o si selecciona de forma predeterminada el formato de salida HTML, los caracteres especiales del archivo XML o XSLT se insertan en el documento HTML como referencias de caracteres HTML. Por ejemplo, el carácter ` ` (la referencia de carácter decimal para un espacio de no separación) se inserta como ` ` en el código HTML.

6.2 XSLT 2.0

Temas de este apartado:

Especificaciones
El motor XSLT 2.0 de RaptorXML cumple con la recomendación XSLT 2.0 del 23 de enero de 2007 y la recomendación XPath 2.0 del 14 de diciembre de 2010, ambas del W3C.

Compatibilidad con versiones antiguas
El motor XSLT 2.0 es compatible con versiones previas. Esto solamente es relevante cuando se utiliza el motor XSLT 2.0 (parámetro de la interfaz de la línea de comandos `--xslt=2`) para procesar una hoja de estilos XSLT 1.0. Tenga en cuenta que los resultados obtenidos con el motor XSLT 1.0 pueden ser diferentes a los obtenidos con el motor XSLT 2.0 en modo de compatibilidad con versiones antiguas.

Espacios de nombres
En su hoja de estilos XSLT 2.0 debe declarar estos espacios de nombres para poder usar los constructores de tipo y las funciones disponibles en XSLT 2.0. Los prefijos que aparecen a continuación son los que se suelen usar, pero puede usar otros prefijos si quiere.

Espacio de nombres	Prefijo	URI del espacio de nombres
Tipos XML Schema	`xs:`	`http://www.w3.org/2001/XMLSchema`
Funciones XPath 2.0	`fn:`	`http://www.w3.org/2005/xpath-functions`

Estos espacios de nombres se suelen declarar en el elemento `xsl:stylesheet` o en el elemento `xsl:transform`:

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

Es necesario tener en cuenta que:

- El motor XSLT 2.0 utiliza el espacio de nombres Funciones XPath 2.0 y XQuery 1.0 como **espacio de nombres de funciones predeterminado**. Esto significa que puede usar funciones XPath 2.0 y XSLT 2.0 en su hoja de estilos sin prefijos. Si declara el espacio de nombres Funciones XPath 2.0 en su hoja de estilos con un prefijo, podrá usar el prefijo asignado en la declaración.
- Cuando se usan constructores de tipo y tipos del espacio de nombres XML Schema, el prefijo utilizado en la declaración de espacio de nombres se debe usar en la llamada al constructor de tipo (por ejemplo, `xs:date`).
- Algunas funciones XPath 2.0 se llaman igual que algunos tipos de datos de XML Schema. Por ejemplo, las funciones XPath `fn:string` y `fn:boolean` y los tipos de datos de XML Schema `xs:string` y `xs:boolean`. Por tanto, si usa la expresión `string('Hello')`, la expresión se evalúa como `fn:string('Hello')` y no como `xs:string('Hello')`.

Compatibilidad con esquemas
El motor XSLT 2.0 está preparado para esquemas de modo que puede usar tipos de esquema definidos por el usuario y la instrucción `xsl:validate`.

Comportamiento propio de esta implementación
Más abajo puede ver cómo se ocupa el motor XSLT 2.0 de algunos aspectos del comportamiento de las funciones XSLT 2.0 relacionadas con esta implementación.

`xsl:result-document`
También son compatibles estas codificaciones específicas de Altova: `x-base16tobinary` y `x-base64tobinary`.

`function-available`
Esta función mira si hay funciones del ámbito disponibles (XSLT 2.0, XPath 2.0 y funciones de extensión).

`unparsed-text`
El atributo `href` acepta (i) rutas de acceso relativas para archivos que estén en la carpeta del URI base y (ii) rutas de acceso absolutas con o sin el protocolo `file://`. También son compatibles estas codificaciones específicas de Altova: `x-binarytobase16` y `x-binarytobase64`.

`unparsed-text-available`
El atributo `href` acepta (i) rutas de acceso relativas para archivos que estén en la carpeta del URI base y (ii) rutas de acceso absolutas con o sin el protocolo `file://`. También son compatibles estas codificaciones específicas de Altova: `x-binarytobase16` y `x-binarytobase64`.

Nota: estos valores de codificación estaban implementados en el ya descatalogado AltovaXML pero ya no se utilizan (son obsoletos): `base16tobinary`, `base64tobinary`, `binarytobase16` y `binarytobase64`.

Funciones XPath 2.0
Consulte el apartado Funciones XQuery 1.0 y XPath 2.0.

6.3 XSLT 3.0

El motor XSLT 3.0 de RaptorXML cumple con el borrador de trabajo XSLT 2.0 del 10 de julio de 2012 y con la recomendación XPath 2.0 del 8 de enero de 2013, ambos del W3C.

El motor XSLT 3.0 tiene las mismas características de implementación que el motor XSLT 2.0. Pero además ofrece las nuevas funciones nuevas `xsl:evaluate`, `xsl:try` y `xsl:catch` y es compatible con funciones y operadores XPath y XQuery 3.0 y con la especificación XPath 3.0.

Nota: estas instrucciones XSLT 3.0 no son compatibles por ahora:

```
xsl:accept
xsl:accumulator
xsl:accumulator-rule
xsl:assert
xsl:break
xsl:context-item
xsl:expose
xsl:fork
xsl:iterate
xsl:map
xsl:map-entry
xsl:merge
xsl:merge-action
xsl:merge-key
xsl:merge-source
xsl:mode
xsl:next-iteration
xsl:next-match
xsl:on-completion
xsl:override
xsl:package
xsl:stream
xsl:use-package
```

6.4 XQuery 1.0

Temas de este apartado:

Especificaciones compatibles
El motor XQuery 1.0 de RaptorXML cumple con la recomendación XQuery 1.0 del 14 de diciembre de 2010 del W3C. El estándar XQuery concede libertad a la hora de implementar muchas características. A continuación explicamos cómo se implementaron estas características en el motor XQuery 1.0 de RaptorXML.

Compatibilidad con esquemas
El motor XQuery 1.0 está preparado para esquemas.

Codificación
El motor XQuery 1.0 es compatible con las codificaciones de caracteres UTF-8 y UTF-16.

Espacios de nombres
Se predefinen estos URI de espacios de nombres y sus enlaces asociados.

Espacio de nombres	Prefijo	URI del espacio de nombres
Tipos XML Schema	`xs:`	`http://www.w3.org/2001/XMLSchema`
Schema instance	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
Funciones integradas	`fn:`	`http://www.w3.org/2005/xpath-functions`

Funciones locales	`local:`	`http://www.w3.org/2005/xquery-local-functions`

Es importante tener en cuenta que:

- El motor XQuery 1.0 entiende que los prefijos de la tabla anterior están enlazados con los correspondientes espacios de nombres.
- Como el espacio de nombres de funciones integradas (ver tabla) es el espacio de nombres de funciones predeterminado de XQuery, no es necesario usar el prefijo `fn:` cuando se invocan funciones integradas (por ejemplo, `string("Hello")` llamará a la función `fn:string`). No obstante, el prefijo `fn:` se puede utilizar para llamar a una función integrada sin necesidad de declarar el espacio de nombres en el prólogo de la consulta (por ejemplo: `fn:string("Hello")`).
- Puede cambiar el espacio de nombres de funciones predeterminado declarando la expresión `default function namespace` en el prólogo de la consulta.
- Cuando use tipos del espacio de nombres XML Schema, puede usar el prefijo `xs:` sin necesidad de declarar los espacios de nombres de forma explícita ni enlazar estos prefijos a los espacios de nombres en el prólogo de la consulta. (Ejemplo: `xs:date` y `xs:yearMonthDuration`.) Si quiere usar otros prefijos para el espacio de nombres de XML Schema, estos se deben declarar en el prólogo de la consulta. (Ejemplo: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Recuerde que los tipos de datos `untypedAtomic`, `dayTimeDuration` y `yearMonthDuration` se movieron del espacio de nombres XPath Datatypes al espacio de nombres XML Schema (es decir, ahora es `xs:yearMonthDuration`.)

Si se asignaron mal los espacios de nombres para funciones, constructores de tipo, pruebas de nodo, etc., se emite un error. Sin embargo, recuerde que algunas funciones se llaman igual que los tipos de datos de esquema (p. ej. `fn:string` y `fn:boolean`.) (Se definen `xs:string` y `xs:boolean`.) El prefijo del espacio de nombres determina si se usa la función o el constructor de tipo.

Documento XML de origen y validación

Los documentos XML que se utilizan para ejecutar un documento XQuery con el motor XQuery 1.0 deben tener un formato XML correcto. Sin embargo, no es necesario que sean válidos con respecto a un esquema XML. Si el archivo no es válido, el archivo no válido se carga sin información de esquema. Si el archivo XML está asociado a un esquema externo y es válido con respecto a dicho esquema, se genera información posterior a la validación de esquema, que se utilizará para evaluar la consulta.

Comprobación de tipos estática y dinámica

En la fase de análisis estático se revisan aspectos de la consulta como la sintaxis, si existen referencias externas (p. ej. para módulos), si las funciones y variables que se invocan están definidas, etc. Si se detecta un error en la fase de análisis estático, se notifica y la ejecución se interrumpe.

La comprobación dinámica de tipos se realiza en tiempo de ejecución, cuando la consulta se ejecuta. Si un tipo no es compatible con los requisitos de una operación, se emite un error. Por ejemplo, la expresión `xs:string("1") + 1` devuelve un error porque la operación de suma no

se puede llevar a cabo en un operando de tipo `xs:string`.

Módulos biblioteca
Los módulos biblioteca almacenan funciones y variables para poder volver a utilizarlas. El motor XQuery 1.0 es compatible con el uso de módulos almacenados en un **solo archivo XQuery externo**. Dicho archivo de módulo debe incluir una declaración `module` en su prólogo que apunte a un espacio de nombres de destino Por ejemplo:

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

Todas las funciones y variables declaradas en el módulo pertenecen al espacio de nombres asociado al módulo. El módulo se importa en un archivo XQuery con la instrucción `import module` del prólogo de la consulta. La instrucción `import module` solamente importa funciones y variables declaradas directamente en el archivo de módulo biblioteca. Por ejemplo:

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

Funciones externas
Las funciones externas son incompatibles con el motor XQuery 1.0, es decir, todas las expresiones que usen la palabra clave `external`. Por ejemplo:

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

Intercalaciones
La intercalación predeterminada es la intercalación de puntos de código Unicode, que compara las cadenas de texto según sus puntos de código Unicode. Otras intercalaciones compatibles son las intercalaciones ICU que se enumeran aquí. Para usar una intercalación concreta, indique su URI tal y como aparece en la lista de intercalaciones compatibles. Las comparaciones de cadenas de texto, incluidas las comparaciones para las funciones `fn:max` y `fn:min`, se harán según la intercalación especificada. Si no se indica la opción de intercalación, se utiliza la intercalación de puntos de código Unicode predeterminada.

Precisión de tipos numéricos

- El tipo de datos `xs:integer` es de precisión arbitraria, es decir, puede representar un número de dígitos cualquiera.
- El tipo de datos `xs:decimal` tiene un límite de 20 dígitos después del punto decimal.
- Los tipos de datos `xs:float` y `xs:double` tienen una precisión limitada de 15 dígitos.

Compatibilidad con instrucciones XQuery
La instrucción `Pragma` no es compatible. Si se encuentra, se ignora y en su lugar se evalúa la expresión de reserva.

Compatibilidad con funciones XQuery
Para más información sobre el comportamiento de las funciones XQuery 1.0, consulte el apartado Funciones XPath 2.0 y XQuery 1.0.

6.5 XQuery 3.0

El motor XQuery 3.0 de RaptorXML cumple con la recomendación XQuery 3.0 del 8 de enero de 2013 del W3C y es compatible con Funciones XPath y XQuery 3.0

Tiene las mismas características de implementación que el motor XQuery 1.0.

Nota: estas características no son compatibles por ahora:

Cláusula Group By en expresiones FLWOR (3.10.7 Cláusula Group By).
Cláusulas Tumbling Window y Sliding Window en expresiones FLWOR (3.10.4 Cláusula Window).
Cláusula Count en expresiones FLWOR (3.10.6 Cláusula Count).
Allowing empty en la cláusula 3.10.2 For, para funciones similares a las combinaciones externas de SQL.
Expresiones Try/Catch (3.15 Expresiones Try/Catch).
Constructores de espacios de nombres calculados (3.9.3.7 Computed Namespace Constructors).
Declaraciones de salida (2.2.4 Serialización).
Anotaciones (4.15 Anotaciones).
Aserciones de función en pruebas de función.

6.6 Funciones XQuery 1.0 y XPath 2.0

Temas de este apartado:

Especificaciones
Las funciones XQuery 1.0 y XPath 2.0 compatibles con RaptorXML cumplen con la recomendación XQuery 1.0 and XPath 2.0 Functions and Operators del 14 de diciembre de 2010 del W3C.

Aspectos generales
A continuación encontrará una lista del comportamiento de ciertas funciones propias de esta implementación. Es importante tener en cuenta que:

- El espacio de nombres predeterminado de las funciones cumple con las normas del estándar. Por tanto, puede llamar a las funciones sin usar un prefijo.
- En general, si una función espera una secuencia de un elemento como argumento y se suministra una secuencia con más de un elemento, se devuelve un error.
- Todas las comparaciones de cadenas se hacen usando la intercalación de puntos de código Unicode.
- Los resultados que son QName se serializan de esta forma `[prefijo:]nombrelocal`.

Precisión de xs:decimal
El término *precisión* hace referencia al número de dígitos del número y la especificación exige un mínimo de 18 dígitos. Para las operaciones de división que dan un resultado de tipo `xs:decimal`, la precisión es de 19 dígitos después del punto decimal sin redondear.

Uso horario implícito
Cuando hay que comparar dos valores `date`, `time` o `dateTime`, es necesario conocer el uso horario de los valores que se comparan. Cuando el uso horario no se conoce de forma explícita, se usa el uso horario implícito, que se toma del reloj del sistema y su valor se puede comprobar con la función `fn:implicit-timezone()`.

Intercalaciones

La intercalación predeterminada es la intercalación de puntos de código Unicode, que compara las cadenas de texto según sus puntos de código Unicode. Otras intercalaciones compatibles son las intercalaciones ICU que aparecen más abajo. Para usar una intercalación determinada, aporte su URI tal y como aparece en la lista de intercalaciones compatibles. Las comparaciones de cadenas de texto (incluidas las necesarias para las funciones `fn:max` y `fn:min`) se harán según la intercalación especificada. Si no se especifica la opción de intercalación, se usa la intercalación de puntos de código Unicode predeterminada.

Idioma	Identificadores URI
`da`: danés	`da_DK`
`de`: alemán	`de_AT, de_BE, de_CH, de_DE, de_LI, de_LU`
`en`: inglés	`en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW`
`es`: español	`es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE`
`fr`: francés	`fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG`
`it`: italiano	`it_CH, it_IT`
`ja`: japonés	`ja_JP`
`nb`: noruego Bokmal	`nb_NO`
`nl`: holandés	`nl_AW, nl_BE, nl_NL`
`nn`: noruego Nynorsk	`nn_NO`
`pt`: portugués	`pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST`
`ru`: ruso	`ru_MD, ru_RU, ru_UA`
`sv`: sueco	`sv_FI, sv_SE`

Eje del espacio de nombres
El eje del espacio de nombres se dejó de utilizar en XPath 2.0. Sin embargo, RaptorXML es compatible con el uso de ejes de espacio de nombres. Para acceder a la información sobre el espacio de nombres con mecanismos de XPath 2.0, utilice las funciones `fn:in-scope-prefixes()`, `fn:namespace-uri()` y `fn:namespace-uri-for-prefix()`.

`base-uri`

- Si se usan entidades externas en el documento XML de origen y si se especifica un nodo de la entidad externa como argumento de entrada de la función `base-uri()`, se utiliza el URI base del documento XML y no el URI base de la entidad externa.
- El URI base de un nodo del documento XML se puede modificar usando el atributo

`xml:base`.

---

collection

- El argumento es un URI relativo que se resuelve con el URI base actual.
- Si el URI resuelto identifica un archivo XML, este archivo XML se trata como un catálogo que hace referencia a una colección de archivos. Este archivo debe tener este formato:

```
<collection>
    <doc href="uri-1" />
    <doc href="uri-2" />
    <doc href="uri-3" />
 </collection>
```

  Se cargan los archivos a los que hacen referencia los atributos `href` y sus nodos de documento se devuelven en forma de secuencia.
- Si el URI resuelto no encuentra ningún archivo XML con la estructura de catálogo descrita, la cadena del argumento (que admite comodines como `?` y `*`) se utiliza como cadena de búsqueda. Se cargan los archivos XML cuyo nombre coincida con la expresión de búsqueda y sus nodos de documento se devuelven en forma de secuencia (*ver ejemplos más abajo*).
- Ejemplo XSLT: la expresión `collection("c:\MyDocs\*.xml")//Title` devuelve una secuencia de todos los elementos `DocTitle` del archivo `.xml` de la carpeta `MyDocs`.
- Ejemplo XQuery: la expresión `{for $i in collection(c:\MyDocs\*.xml) return element doc{base-uri($i)}}` devuelve los URI base de todos los archivos `.xml` de la carpeta `MyDocs`, donde cada URI está dentro de un elemento `doc`.
- La intercalación predeterminada está vacía.

---

current-date, current-dateTime, current-time

- La fecha y hora actual se toma del reloj del sistema.
- El uso horario se toma del uso horario implícito que se obtiene al evaluar el contexto. El uso horario implícito se toma del reloj del sistema.
- El uso horario siempre se especifica en el resultado.

---

doc

Solamente se emite un error si no hay ningún archivo XML disponible en la ubicación especificada o si el archivo no tiene un formato XML correcto. El archivo se valida si hay un esquema disponible. Si el archivo no es válido, el archivo no válido se carga sin información de esquema.

---

id

En un documento con formato XML correcto pero no válido que contiene dos o más elementos con el mismo valor ID, se devuelve el primer elemento del orden de documento.

---

in-scope-prefixes

En el documento XML solamente se puede anular la declaración de espacios de nombres predeterminados. Sin embargo, incluso al anular la declaración de un espacio de nombres predeterminado de un nodo de elemento, se devuelve el prefijo para el espacio de nombres predeterminado, que es una cadena de longitud cero.

---

normalize-unicode

Los formatos de normalización compatibles son NFC, NFD, NFKC y NFKD.

---

resolve-uri

- Si se omite el segundo argumento, opcional, el URI que se debe resolver (el primer argumento) se resuelve con el URI base del contexto estático, que es el URI de la hoja de estilos XSLT o el URI base dado en el prólogo del documento XQuery.
- El URI relativo (el primer argumento) se anexa después del último "/" de la notación de ruta de acceso de la notación del URI base.
- Si el valor del primer argumento es la cadena de longitud cero, se devuelve el URI base del contexto estático y este URI incluye el nombre de archivo del documento del que se deriva el URI base del contexto estático (p. ej. el archivo XSLT o XML).

---

static-base-uri

El URI base del contexto estático es el URI base de la hoja de estilos XSLT o el URI base especificado en el prólogo del documento XQuery.

6.7 Funciones XQuery y XPath 3.0

Las funciones y operadores XPath y XQuery 3.0 compatibles con RaptorXML cumplen con la recomendación del W3C sobre Funciones y operadores XPath y XQuery 3.0 del 21 de mayo de 2013.

Tienen las mismas características que las funciones XQuery 1.0 y XPath 2.0.

Altova RaptorXML Development Edition 2014

Funciones de extensión XSLT y XQuery

7 Funciones de extensión XSLT y XQuery

En los lenguajes de programación, como Java y C#, hay varias funciones predefinidas que no están disponibles como funciones XQuery/XPath ni como funciones XSLT. Un buen ejemplo son las funciones matemáticas disponibles en Java, como `sin()` y `cos()`. Si estas funciones estuvieran a disposición de los diseñadores de hojas de estilos XSLT y consultas XQuery, el ámbito de aplicación de las hojas de estilos y de las consultas sería mayor y sería más fácil crearlas.

Los motores XSLT y XQuery utilizados en los productos de Altova admiten el uso de funciones de extensión en Java y .NET. También admiten el uso de scripts MSXSL para XSLT y las funciones de extensión de Altova.

Tenga en cuenta que, excepto algunas funciones de extensión de Altova para XSLT, a casi todas las funciones de extensión que aparecen en este anexo se les llama desde expresiones XPath. Este anexo describe cómo usar funciones de extensión y scripts MSXSL en sus hojas de estilos XSLT y en sus documentos XQuery. El anexo se divide en estos apartados:

- Funciones de extensión de Altova
- Funciones de extensión Java
- Funciones de extensión .NET
- Scripts MSXSL para XSLT

Los dos principales aspectos que se tienen en cuenta en estos apartados son: (i) cómo se llama a las funciones de cada biblioteca y (ii) qué reglas se siguen para convertir los argumentos de una llamada a función en el formato de entrada de la función y qué reglas se siguen para la conversión de retorno (el resultado de la función se convierte en objeto de datos XSLT/XQuery).

Requisitos
Para poder trabajar con funciones de extensión, en el equipo que ejecuta la transformación XSLT o la ejecución XQuery debe tener instalado (o debe tener acceso a) un entorno de ejecución Java (para acceder a funciones Java) y .NET Framework 2.0 (mínimo, para acceder a funciones .NET).

7.1 Funciones de extensión de Altova

Las funciones de extensión de Altova están en el espacio de nombres

```
http://www.altova.com/xslt-extensions
```

y en esta sección se presentan con el prefijo

```
altova:
```

que se supone estará enlazado al espacio de nombres señalado.

Las funciones de extensión de Altova se pueden usar tanto en un contexto XPath como en un contexto XSLT (es decir, como funciones XPath y como funciones XSLT respectivamente). Asegúrese de utilizar la función de extensión en el contexto adecuado (XPath o XSLT).

Las funciones de extensión descritas en esta sección son compatibles con la versión actual de su producto Altova. No obstante, tenga en cuenta que en futuras versiones del producto algunas funciones pueden dejar de ser compatibles o su comportamiento puede cambiar. Por tanto, consulte siempre la documentación del producto para conocer el funcionamiento de estas funciones en cada versión del producto.

Funciones XPath
Puede usar estas funciones en contextos XPath:

- `altova:generate-auto-number()`
- `altova:reset-auto-number()`
- `altova:get-temp-folder()`
  - Funciones para añadir una duración a xs:dateTime y obtener xs:dateTime
  - Funciones para añadir una duración a xs:date y obtener xs:date
  - Funciones par añadir una duración a xs:time y obtener xs:time
  - Funciones para quitar la zona horaria de la fecha/hora actual
  - Funciones para obtener el día de la semana en forma de número entero

Funciones XSLT
Puede usar estas funciones en un contexto XSLT, igual que las funciones XSLT 2.0 `current-group()` o `key()`:

- **`altova:evaluate()`**
- **`altova:distinct-nodes()`**
- **`altova:encode-for-rtf()`**
- **`altova:xbrl-labels()`**
- **`altova:xbrl-footnotes()`**

Funciones para códigos de barras
Las funciones de extensión de Altova para códigos de barras permiten generar códigos de barras y guardarlos en el los documentos de salida generados a partir de hojas de estilos XSLT.

7.1.1 Funciones XSLT generales

Las funciones XSLT descritas en este apartado son compatibles con la versión actual de su producto Altova. No obstante, tenga en cuenta que en futuras versiones del producto algunas funciones pueden dejar de ser compatibles o su comportamiento puede cambiar. Por tanto, consulte siempre la documentación del producto para conocer el funcionamiento de estas funciones en cada versión del producto.

Recuerde que las funciones de extensión de Altova están en el **espacio de nombres de funciones de extensión de Altova**

```
http://www.altova.com/xslt-extensions
```

y en esta sección se presentan con el prefijo

```
altova:
```

que se supone estará enlazado al espacio de nombres señalado.

Estas funciones generales están disponibles en un contexto XSLT, más o menos como las funciones XSLT 2.0 `current-group()` o `key()`:

- altova:evaluate()
- altova:distinct-nodes()
- altova:encode-for-rtf()
- altova:xbrl-labels()
- altova:xbrl-footnotes()

Nota: las funciones XBRL de Altova solamente están disponibles en los productos de Altova compatibles con XBRL.

Funciones para contextos XSLT
Estas funciones se pueden usar en contextos XSLT.

altova:evaluate()
Toma una expresión XPath, pasada en forma de cadena, como argumento obligatorio. Devuelve el resultado de la expresión evaluada.

```
altova:evaluate(XPathExp as xs:string)
```

Por ejemplo:

```
altova:evaluate('//Name[1]')
```

Aquí, la expresión `//Name[1]` se pasa como cadena si se escribe entre comillas simples. La función `altova:evaluate` devuelve el contenido del primer elemento `Name` del documento.

La función `altova:evaluate` puede tomar otros argumentos (opcionales). Estos argumentos son los valores de las variables de nombre `p1`, `p2`, `p3`... `pN` que se pueden usar en la expresión XPath.

```
altova:evaluate(XPathExp as xs:string [, p1value ... pNvalue])
```


donde

- los nombres de variable deben tener el formato `pX`, siendo `X` un número entero
- la secuencia de los argumentos de la función, a partir del segundo argumento, corresponde a la secuencia de las variables `p1` a `pN`. De modo que el segundo argumento será el valor de la variable `p1`, el tercero será el valor de la variable `p2` y así sucesivamente.
- los valores de variable deben ser de tipo `item*`

Por ejemplo:

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
<xsl:value-of select="altova:evaluate( $xpath, 10, 20, 'hi' )" />
```
Resultado: "`hi 20 10`*"*

Observe que en el ejemplo anterior:

- el segundo argumento de la expresión `altova:evaluate` es el valor asignado a la variable `$p1`, el tercer argumento es el valor asignado a la variable `$p2` y así sucesivamente.
- el cuarto argumento de la función es un valor de cadena, al estar escrito entre comillas simples.
- el atributo `select` del elemento `xs:variable` suministra la expresión XPath. Puesto que la expresión debe ser de tipo `xs:string`, se escribe entre comillas simples.

Aquí tiene más ejemplos:

```
<xsl:variable name="xpath" select="'$p1'" />
<xsl:value-of select="altova:evaluate( $xpath, //Name[1] )" />
```
Resultado: el valor del primer elemento `Name`*.*

```
<xsl:variable name="xpath" select="'$p1'" />
<xsl:value-of select="altova:evaluate( $xpath, '//Name[1]' )" />
```
Resultado: "`//Name[1]`*"*

La función de extensión `altova:evaluate()` es muy práctica cuando una expresión XPath de la hoja de estilos XSLT contiene partes que se deben evaluar de forma dinámica. Por ejemplo, imagine que el usuario selecciona un criterio de ordenación y este criterio se almacena en el atributo `UserReq/@sortkey`. En la hoja de estilos podría tener esta expresión:

```
<xsl:sort select="altova:evaluate(../UserReq/@sortkey)" order="ascending"/
>
```

La función `altova:evaluate()` lee el atributo `sortkey` del elemento secundario `UserReq` del primario del nodo de contexto. Digamos que el valor del atributo `sortkey` es `Price`. En ese caso, la función `altova:evaluate()` devuelve `Price`, que se convierte en el valor del atributo `select`:

```
<xsl:sort select="Price" order="ascending"/>
```

Si esta instrucción `sort` aparece dentro del contexto de un elemento llamado `Order`, entonces los elementos `Order` se ordenan según el valor de los secundarios `Price`. Otra opción es que, si el valor de `@sortkey` fuera `Date`, por ejemplo, entonces los elementos `Order` se ordenarían según el valor de los secundarios `Date`. Es decir, el criterio de ordenación para `Order` se selecciona del atributo `sortkey` en tiempo de ejecución. Esto no sería posible con una expresión como:

```
<xsl:sort select="../UserReq/@sortkey" order="ascending"/>
```

En este caso, el criterio de ordenación sería el propio atributo `sortkey`, no `Price` ni `Date` (ni

otro contenido actual de `sortkey`).

En la función de extensión `altova:evaluate()` puede usar estas variables:

- Variables estáticas: `<xsl:value-of select="$i3, $i2, $i1" />`
  Resultado: el valor de las tres variables.

- Expresiones XPath dinámicas con variables dinámicas:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate( $xpath, 10, 20, 30 )" />
  ```
  Resultado: "30 20 10"

- Expresiones XPath dinámicas sin variables dinámicas:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate( $xpath )" />
  ```
  Resultado: "No variable defined for $p3."

Nota: el contexto estático incluye espacios de nombres, tipos y funciones (pero no variables) del entorno desde donde se llama a la función. El URI base y el espacio de nombres predeterminado se heredan.

`altova:distinct-nodes()`
La función `altova:distinct-nodes()` toma un conjunto de nodos como entrada y devuelve el mismo conjunto menos los nodos que tienen el mismo valor. La comparación se realiza por medio de la función XPath/XQuery `fn:deep-equal`.

```
altova:distinct-nodes( $arg as node()* )  as node()*
```

`altova:encode-for-rtf()`
La función `altova:encode-for-rtf()` convierte la cadena de entrada en código para RTF.

```
altova:encode-for-rtf( $inputstr as xs:string?,
$preserveallwhitespace as xs:boolean,
$preservenewlines as xs:boolean) as xs:string
```

Los espacios en blanco y las líneas nuevas se conservan dependiendo del valor booleano especificado en sus respectivos parámetros.

`altova:xbrl-labels()`
La función `altova:xbrl-labels()` toma dos argumentos de entrada: un nombre de nodo y la ubicación del archivo de taxonomía que incluye el nodo. La función devuelve las etiquetas XBRL asociadas al nodo de entrada.

```
altova:xbrl-labels( $name as xs:QName, $file as xs:string )  as node()*
```

`altova:xbrl-footnotes()`
La función `altova:footnotes()` toma un nodo como argumento de entrada y devuelve el conjunto de los nodos de nota al pie XBRL a los que hace referencia el nodo de entrada.

```
altova:footnotes( $arg as node() )  as node()*
```


7.1.2 Funciones XPath generales

Las funciones de uso general que aparecen en este apartado se pueden usar en contextos XPath y son compatibles con la versión actual de su producto Altova. No obstante, tenga en cuenta que en futuras versiones del producto algunas funciones pueden dejar de ser compatibles o su comportamiento puede cambiar. Por tanto, consulte siempre la documentación del producto para conocer el funcionamiento de estas funciones en cada versión del producto.

Recuerde que las funciones de extensión de Altova están en el **espacio de nombres de funciones de extensión de Altova**

```
http://www.altova.com/xslt-extensions
```

y en esta sección se presentan con el prefijo

```
altova:
```

que se supone estará enlazado al espacio de nombres señalado.

Estas funciones de uso general están disponibles en contextos XPath:

- altova:generate-auto-number()
- altova:reset-auto-number()
- altova:get-temp-folder()

Funciones para contextos XPath
Estas funciones se pueden usar en contextos XPath.

altova:generate-auto-number(id as xs:string, start-with as xs:double, increment as xs:double, reset-on-change as xs:string)
Genera una serie de números que tienen el ID especificado. Se especifica el entero inicial y el incremento.

---

altova:reset-auto-number(id as xs:string)
Esta función restablece la numeración automática de la serie de numeración automática especificada con el argumento de Id. La serie vuelve a empezar por el entero inicial de la serie (ver la función `altova:generate-auto-number`).

---

altova:get-temp-folder as xs:string
Obtiene la carpeta temporal.

7.1.3 Funciones de fecha y hora (XPath)

Las funciones de fecha y hora de Altova se pueden usar en contextos XPath y permiten procesar datos almacenados en tipos de datos XML Schema de fecha y hora. Estas funciones se pueden usar con el **motor XPath 3.0** de Altova.

Recuerde que las funciones de extensión de Altova están en el **espacio de nombres de funciones de extensión de Altova**

http://www.altova.com/xslt-extensions

y en esta sección se presentan con el prefijo

altova:

que se supone estará enlazado al espacio de nombres señalado.

Estas funciones de fecha y hora están disponibles en contextos XPath y a continuación aparecen agrupadas según su funcionamiento.

- Agregar una duración a un xs:dateTime y obtener un xs:dateTime [add-years-to-dateTime | add-months-to-dateTime | add-days-to-dateTime | add-hours-to-dateTime | add-minutes-to-dateTime | add-seconds-to-dateTime]
- Agregar una duración a un xs:date y obtener un xs:date [add-years-to-date | add-months-to-date | add-days-to-date]
- Agregar una duración a un xs:time y obtener un xs:time [add-hours-to-time | add-minutes-to-time | add-seconds-to-time]
- Quitar la zona horaria de la fecha/hora actual [current-dateTime-no-TZ | current-date-no-TZ | current-time-no-TZ]
- Obtener el día de la semana en forma de número entero [weekday-from-dateTime | weekday-from-dateTime | weekday-from-date | weekday-from-date]

Agregar duraciones a `xs:dateTime`
Estas funciones añaden una duración a **`xs:dateTime`** y devuelven **`xs:dateTime`**. El tipo `xs:dateTime` tiene un formato léxico de este tipo `2014-01-24T13:30:30`.

`altova:`**`add-years-to-dateTime(`**`xs:dateTime, xs:integer`**`)`** como `xs:dateTime`
Añade una duración en años a `xs:dateTime`. El segundo parámetro es el número de años que se añaden al `xs:dateTime` dado en el primer parámetro. El resultado es de tipo `xs:dateTime`.

`altova:`**`add-months-to-dateTime(`**`xs:dateTime, xs:integer`**`)`** como `xs:dateTime`
Añade una duración en meses a `xs:dateTime`. El segundo parámetro es el número de meses que se añaden al `xs:dateTime` dado en el primer parámetro. El resultado es de tipo `xs:dateTime`.

`altova:`**`add-days-to-dateTime(`**`xs:dateTime, xs:integer`**`)`** como `xs:dateTime`
Añade una duración en días a `xs:dateTime`. El segundo parámetro es el número de días que se añaden al `xs:dateTime` dado en el primer parámetro. El resultado es de tipo `xs:dateTime`.

`altova:add-hours-to-dateTime(xs:dateTime, xs:integer)` como `xs:dateTime`
Añade una duración en horas a `xs:dateTime`. El segundo parámetro es el número de horas que se añaden al `xs:dateTime` dado en el primer parámetro. El resultado es de tipo `xs:dateTime`.

`altova:add-minutes-to-dateTime(xs:dateTime, xs:integer)` como `xs:dateTime`
Añade una duración en minutos a `xs:dateTime`. El segundo parámetro es el número de minutos que se añaden al `xs:dateTime` dado en el primer parámetro. El resultado es de tipo `xs:dateTime`.

`altova:add-seconds-to-dateTime(xs:dateTime, xs:integer)` como `xs:dateTime`
Añade una duración en segundos a `xs:dateTime`. El segundo parámetro es el número de segundos que se añaden al `xs:dateTime` dado en el primer parámetro. El resultado es de tipo `xs:dateTime`.

Agregar duraciones a `xs:date`

Estas funciones añaden una duración a **`xs:date`** y devuelven **`xs:date`**. El tipo `xs:date` tiene un formato léxico de este tipo `2014-01-24`.

`altova:add-years-to-date(xs:date, xs:integer)` como `xs:date`
Añade una duración en años a `xs:date`. El segundo parámetro es el número de años que se añaden al `xs:date` dado en el primer parámetro. El resultado es de tipo `xs:date`.

`altova:add-months-to-date(xs:date, xs:integer)` como `xs:date`
Añade una duración en meses a `xs:date`. El segundo parámetro es el número de meses que se añaden al `xs:date` dado en el primer parámetro. El resultado es de tipo `xs:date`.

`altova:add-days-to-date(xs:date, xs:integer)` como `xs:date`
Añade una duración en días a `xs:date`. El segundo parámetro es el número de días que se añaden al `xs:date` dado en el primer parámetro. El resultado es de tipo `xs:date`.

Agregar duraciones a `xs:time`

Estas funciones añaden una duración a **`xs:time`** y devuelven **`xs:time`**. El tipo `xs:time` tiene un formato léxico de este tipo `23:00:00+03:00:00`.

`altova:add-hours-to-time(xs:time, xs:integer)` como `xs:time`
Añade una duración en horas a `xs:time`. El segundo parámetro es el número de horas que se añaden al `xs:time` dado en el primer parámetro. El resultado es de tipo `xs:time`.

`altova:add-minutes-to-time(xs:time, xs:integer)` como `xs:time`

Añade una duración en minutos a `xs:time`. El segundo parámetro es el número de minutos que se añaden al `xs:time` dado en el primer parámetro. El resultado es de tipo `xs:time`.

`altova:add-seconds-to-time(xs:time, xs:integer)` como `xs:time`
Añade una duración en segundos a `xs:time`. El segundo parámetro es el número de segundos que se añaden al `xs:time` dado en el primer parámetro. El resultado es de tipo `xs:time`.

Quitar la zona horaria del tipo de datos
Estas funciones quitan la zona horaria de los valores **`xs:dateTime`**, **`xs:date`** o **`xs.time`** actuales respectivamente. Recuerde que la diferencia entre `xs:dateTime` y `xs:dateTimeStamp` es que `dateTimeStamp` necesita la parte de zona horaria obligatoriamente y `dateTime` puede llevarla o no. El formato de un valor `xs:dateTimeStamp` es `CCYY-MM-DDThh:mm:ss.sss±hh:mm` o `CCYY-MM-DDThh:mm:ss.sssZ`.

`altova:current-dateTime-no-TZ()` como `xs:dateTimeStamp`
Quita la parte de zona horaria de `current-dateTime()` (que es la hora y fecha actual del reloj del sistema) y devuelve un valor `xs:dateTimeStamp`.

`altova:current-date-no-TZ()` como `xs:date`
Quita la parte de zona horaria de `current-date()` (que es la fecha actual del reloj del sistema) y devuelve un valor `xs:date`.

`altova:current-time-no-TZ()` como `xs:time`
Quita la parte de zona horaria de `current-time()` (que es la hora actual del reloj del sistema) y devuelve un valor `xs:time`.

Obtener el día de la semana de `xs:dateTime` o de `xs:date`
Estas funciones devuelven el día de la semana (en forma de número entero) de un `xs:dateTime` o `xs:date`. Los días de la semana se numeran del 1 al 7. En el formato europeo, la semana empieza el lunes (`=1`). En los demás formatos la semana empieza el domingo (`=1`).

`altova:weekday-from-dateTime(xs:dateTime)` as `xs:integer`
Devuelve el día de la semana en forma de número entero. Los días de la semana se numeran del 1 al 7 empezando por `Domingo=1`. Si necesita usar el formato europeo (donde `Lunes=1`), utilice la otra firma de esta función (*ver siguiente función*).

`altova:weekday-from-dateTime(xs:dateTime, xs:integer)` as `xs:integer`
Devuelve el día de la semana en forma de número entero. Si el segundo parámetro (número entero) es `0`, entonces los días de la semana se numeran del 1 al 7 empezando por `Domingo=1`. Si no hay un segundo parámetro o si el segundo parámetro es un número entero distinto a `0`, entonces se numera `Lunes=1`.

`altova:weekday-from-date(xs:date)` as `xs:integer`

Devuelve el día de la semana en forma de número entero. Los días de la semana se numeran del 1 al 7 empezando por `Domingo=1`. Si necesita usar el formato europeo (donde `Lunes=1`), utilice la otra firma de esta función (*ver siguiente función*).

`altova:weekday-from-date(xs:date, xs:integer)` as `xs:integer`
Devuelve el día de la semana en forma de número entero. Si el segundo parámetro (número entero) es `1`, entonces los días de la semana se numeran del 1 al 7 empezando por `Lunes=1`. Si no hay un segundo parámetro o si el segundo parámetro es un número entero distinto a `1`, entonces se numera `Domingo=1`.

7.1.4 Funciones para códigos de barras (XPath)

Los motores XSLT de Altova usan bibliotecas Java de terceros para crear códigos de barras. A continuación enumeramos las clases y los métodos públicos utilizados. Las clases se empaquetan en `AltovaBarcodeExtension.jar`, que está en la carpeta `C:\Archivos de programa\Altova\CommonYYYY\jar`.

Las bibliotecas Java utilizadas están en las subcarpetas de la carpeta `C:\Archivos de programa\Altova\CommonYYYY\jar`:

- `barcode4j\barcode4j.jar` (Sitio web: http://barcode4j.sourceforge.net/)
- `zxing\core.jar` (Sitio web: http://code.google.com/p/zxing/)

Los archivos de licencia están también en estas carpetas.

Paquete `com.altova.extensions.barcode`
El paquete `com.altova.extensions.barcode` se utiliza para generar la mayoría de códigos de barras.

Se utilizan estas clases:

```
public class BarcodeWrapper
    static BarcodeWrapper newInstance( String name, String msg, int dpi, int
orientation, BarcodePropertyWrapper[] arrProperties )
    double getHeightPlusQuiet()
    double getWidthPlusQuiet()
    org.w3c.dom.Document generateBarcodeSVG()
    byte[] generateBarcodePNG()
    String generateBarcodePngAsHexString()
```

`public class` **`BarcodePropertyWrapper`** *Utilizada para almacenar las propiedades del código de barras que se establecerán más tarde de forma dinámica*

```
    BarcodePropertyWrapper( String methodName, String propertyValue )
    BarcodePropertyWrapper( String methodName, Integer propertyValue
)
    BarcodePropertyWrapper( String methodName, Double propertyValue )
    BarcodePropertyWrapper( String methodName, Boolean propertyValue )
    BarcodePropertyWrapper( String methodName, Character propertyValue )
    String getMethodName()
    Object getPropertyValue()
```

`public class` **`AltovaBarcodeClassResolver`** *Registra la clase* `com.altova.extensions.barcode.proxy.zxing.QRCodeBean` *para el bean* `qrcode`*, además de las clases registradas por* `org.krysalis.barcode4j.DefaultBarcodeClassResolver`.

Paquete `com.altova.extensions.barcode.proxy.zxing`
El paquete `com.altova.extensions.barcode.proxy.zxing` se utiliza para generar códigos de barras de tipo QRCode.

Se utilizan estas clases:

`class` **`QRCodeBean`**

- *Extiende* `org.krysalis.barcode4j.impl.AbstractBarcodeBean`
- *Crea una interfaz* `AbstractBarcodeBean` *para* `com.google.zxing.qrcode.encoder`


```
    void generateBarcode(CanvasProvider canvasImp, String msg)
    void setQRErrorCorrectionLevel(QRCodeErrorCorrectionLevel level)
    BarcodeDimension calcDimensions(String msg)
    double getVerticalQuietZone()
    double getBarWidth()

class QRCodeErrorCorrectionLevel Nivel de corrección de errores para QRCode
    static QRCodeErrorCorrectionLevel byName(String name)
    "L" = ~7% correction
    "M" = ~15% correction
    "H" = ~25% correction
    "Q" = ~30% correction
```

Plantilla XSLT

A continuación puede ver un ejemplo de plantilla XSLT que ilustra el uso de funciones para códigos de barras en una hoja de estilos XSLT.

```
<?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:altova="http://www.altova.com"
   xmlns:altovaext="http://www.altova.com/xslt-extensions"
   xmlns:altovaext-barcode="java:com.altova.extensions.barcode.BarcodeWrapper"

   xmlns:altovaext-barcode-property="
java:com.altova.extensions.barcode.BarcodePropertyWrapper">
   <xsl:output method="html" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
      <html>
         <head><title/></head>
         <body>
            <img alt="barcode" src="{altovaext:get-temp-folder()}barcode.png
"/>
         </body>
      </html>
      <xsl:result-document
         href="{altovaext:get-temp-folder()}barcode.png"
         method="text" encoding="base64tobinary" >
         <xsl:variable name="barcodeObject"
            select="
altovaext-barcode:newInstance(&apos;Code39&apos;,string(&apos;some
value&apos;),
            96,0, (altovaext-barcode-property:new( &apos;setModuleWidth&apos;,
25.4 div 96 * 2 ) ) )"/>
         <xsl:value-of select="
xs:base64Binary(xs:hexBinary(string(altovaext-barcode:generateBarcodePngAsHexS
tring($barcodeObject)) ))"/>
      </xsl:result-document>
   </xsl:template>
</xsl:stylesheet>
```

7.2 Funciones de extensión Java

Puede usar una función de extensión Java dentro de una expresión XPath o XQuery para invocar un constructor Java o llamar a un método Java (estático o de instancia).

Un campo de una clase Java se trata como un método sin argumentos. Un campo puede ser estático o de instancia. Más adelante describimos cómo se accede a los campos estáticos y de instancia.

Este apartado tiene varias partes:

- Java: constructores
- Java: métodos estáticos y campos estáticos
- Java: métodos de instancia y campos de instancia
- Tipos de datos: XSLT/XQuery a Java
- Tipos de datos: Java a XSLT/XQuery

Formato de la función de extensión
La función de extensión de la expresión XPath/XQuery debe tener este formato `prefijo:nombreFunción()`.

- La parte `prefijo:` identifica la función de extensión como función Java. Lo hace asociando la función de extensión con una declaración de espacio de nombres del ámbito, cuyo URI debe empezar por `java:` (*ver ejemplos más abajo*). La declaración de espacio de nombres debe identificar una clase Java, por ejemplo: `xmlns:myns="java:java.lang.Math"`. Sin embargo, también puede ser simplemente: `xmlns:myns="java"` (sin los dos puntos), dejando la identificación de la clase Java a la parte `nombreFunción()` de la función de extensión.
- La parte `nombreFunción()` identifica el método Java al que se llama y presenta los argumentos para el método (*ver ejemplos más abajo*). Sin embargo, si el URI de espacio de nombres identificado por la parte `prefijo:` no identifica una clase Java (*ver punto anterior*), entonces la clase Java debe identificarse en la parte `nombreFunción()`, antes de la clase y separada de la clase por un punto (*ver el segundo ejemplo XSLT que aparece más abajo*).

Nota: la clase a la que se llama debe estar en la ruta de acceso de clase del equipo.

Ejemplo de código XSLT
Aquí ofrecemos dos ejemplos de cómo se puede llamar a un método estático. En el primer ejemplo, el nombre de la clase (`java.lang.Math`) se incluye en el URI de espacio de nombres y, por tanto, no puede estar en la parte `nombreFunción()`. En el segundo ejemplo, la parte `prefijo:` presenta el prefijo `java:` mientras que la parte `nombreFunción()` identifica la clase y el método.

```
<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)" />
```

El método nombrado en la función de extensión (`cos()`) debe coincidir con el nombre de un método estático público de la clase Java nombrada (`java.lang.Math`).

Ejemplo de código XQuery
Aquí puede ver un ejemplo de código XQuery similar al código XSLT anterior:


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

Clases Java definidas por el usuario

Si creó sus propias clases Java, a los métodos de estas clases se les llama de otra manera, dependiendo de: (i) si a las clases se accede por medio de un archivo JAR o de un archivo de clases y (ii) si estos archivos están en el directorio actual (el directorio del documento XSLT o XQuery). Para más información consulte los apartados Archivos de clases definidos por el usuario y Archivos Jar definidos por el usuario. Recuerde que debe especificar las rutas de acceso de los archivos de clases que no están en el directorio actual y de todos los archivos JAR.

7.2.1 Archivos de clases definidos por el usuario

Si se accede a las clases por medio de un archivo de clases, entonces hay cuatro posibilidades:

- El archivo de clases está en un paquete. El archivo XSLT/XQuery está en la misma carpeta que el paquete Java. (*Ver ejemplo más abajo*.)
- El archivo de clases no está en un paquete. El archivo XSLT/XQuery está en la misma carpeta que el archivo de clases. (*Ver ejemplo más abajo*.)
- El archivo de clases está en un paquete. El archivo XSLT/XQuery está en una carpeta cualquiera. (*Ver ejemplo más abajo*.)
- El archivo de clases no está en un paquete. El archivo XSLT/XQuery está una carpeta cualquiera. (*Ver ejemplo más abajo*.)

Imaginemos que tenemos un archivo de clases que no está en un paquete y que está en la misma carpeta que el documento XSLT/XQuery. En este caso, puesto que en la carpeta se encuentran todas las clases, no es necesario especificar la ubicación del archivo. La sintaxis que se utiliza para identificar una clase es esta:

```
java:nombreClase
```

donde

`java:` indica que se está llamando a una función definida por el usuario (por defecto se cargan las clases Java del directorio actual)
`nombreClase` es el nombre de la clase del método elegido

La clase se identifica en un URI de espacio de nombres y el espacio de nombres se usa como prefijo para la llamada al método.

El archivo de clases está en un paquete. El archivo XSLT/XQuery está en la misma carpeta que el paquete Java
El código que aparece a continuación llama al método `getVehicleType()` de la clase `Car` del paquete `com.altova.extfunc`. El paquete `com.altova.extfunc` está en la carpeta `JavaProject`. El archivo XSLT también está en la carpeta `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>
```

El archivo de clases no está en un paquete. El archivo XSLT/XQuery está en la misma carpeta que el archivo de clases
El código que aparece a continuación llama al método `getVehicleType()` de la clase `Car` del paquete `com.altova.extfunc`. El archivo de clases `Car` está en esta carpeta: `JavaProject/com/altova/extfunc`. El archivo XSLT también está en la carpeta

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>
```

El archivo de clases está en un paquete. El archivo XSLT/XQuery está en una carpeta cualquiera

El código que aparece a continuación llama al método `getCarColor()` de la clase `Car` del paquete `com.altova.extfunc`. El paquete `com.altova.extfunc` está en la carpeta `JavaProject`. El archivo XSLT está en otra carpeta cualquiera. En este caso debe especificarse la ubicación del paquete dentro del URI como una cadena de consulta. La sintaxis es esta:

```
java:nombreClase[?ruta=uri-del-paquete]
```

donde

`java:` indica que se está llamando a una función Java definida por el usuario
`uri-del-paquete` es el URI del paquete Java
`nombreClase` es el nombre de la clase del método elegido

La clase se identifica en un URI de espacio de nombres y el espacio de nombres se usa como prefijo para la llamada al método. El ejemplo de código que aparece a continuación explica cómo se accede a un archivo de clases que está ubicado en un directorio que no es el directorio actual.

```
<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>
```

El archivo de clases no está en un paquete. El archivo XSLT/XQuery está una carpeta cualquiera

El código que aparece a continuación llama al método `getCarColor()` de la clase `Car` del

paquete `com.altova.extfunc`. El paquete `com.altova.extfunc` está en la carpeta `JavaProject`. El archivo XSLT está en otra carpeta cualquiera. En este caso debe especificarse la ubicación del paquete dentro del URI como una cadena de consulta. La sintaxis es esta:

```
java:nombreClase[?ruta=uri-del-archivoClases]
```

donde

`java:` indica que se está llamando a una función Java definida por el usuario
`uri-del-archivoClases` es el URI de la carpeta donde se ubica el archivo de clases
`nombreClase` es el nombre de la clase del método elegido

La clase se identifica en un URI de espacio de nombres y el espacio de nombres se usa como prefijo para la llamada al método. El ejemplo de código que aparece a continuación explica cómo se accede a un archivo de clases que está ubicado en un directorio que no es el directorio actual.

```
<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>
```

Nota: cuando se presenta una ruta de acceso por medio de una función de extensión, la ruta de acceso se añade al ClassLoader.

7.2.2 Archivos JAR definidos por el usuario

Si se accede a las clases por medio de un archivo JAR, entonces se debe especificar el URI del archivo JAR usando esta sintaxis:

```
xmlns:claseEspacioNombres="java:nombreClase?ruta=jar:uri-del-archivoJar!/"
```

Para la llamada al método se usa el preifjo del URI de espacio de nombres que identifica la clase: `claseEspacioNombres:método()`

En la sintaxis anterior:

`java:` indica que se está llamando a una función de Java
`nombreClase` es el nombre de la clase definida por el usuario
`?` es el separador entre el nombre de la clase y la ruta de acceso
`ruta=jar:` indica que se ofrece una ruta de acceso a un archivo JAR
`uri-del-archivoJar` es el URI del archivo JAR
`!/` es el delimitador final de la ruta de acceso
`claseEspacioNombres:método()` es la llamada al método

Otra opción es dar el nombre de la clase con la llamada al método. Por ejemplo:

```
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()
```

Y aquí puede ver un ejemplo de XSLT que usa un archivo JAR para llamar a una función de extensión Java:

```
<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/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
       <xsl:variable name="myCar" select="car:Car1.new('red')" />
       <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

Nota: cuando se presenta una ruta de acceso por medio de una función de extensión, la ruta de acceso se añade al ClassLoader.

7.2.3 Java: constructores

Una función de extensión se puede usar para llamar a un constructor Java. A todos los constructores se les llama con la pseudofunción `new()`.

Si el resultado de una llamada a un constructor Java se puede convertir de manera implícita a tipos de datos XPath/XQuery, entonces la llamada a la función de extensión Java devuelve una secuencia que es un tipo de datos XPath/XQuery. Si el resultado de una llamada a un constructor Java no se puede convertir a un tipo de datos XPath/XQuery adecuado, entonces el constructor crea un objeto Java contenido con un tipo que es el nombre de la clase que devuelve ese objeto Java. Por ejemplo, si se llama a un constructor para la clase `java.util.Date` (`java.util.Date.new()`), entonces se devuelve un objeto que tiene el tipo `java.util.Date`. Puede que el formato léxico del objeto devuelto no coincida con el formato léxico de un tipo de datos XPath y, por tanto, su valor debe convertirse al formato léxico del tipo de datos XPath pertinente y después al tipo de datos XPath.

Puede hacer dos cosas con el objeto Java creado por un constructor:

- Puede asignar el objeto a una variable:
```
<xsl:variable name="currentdate" select="date:new()" xmlns:date="
java:java.util.Date" />
```
- Puede pasar el objeto a una función de extensión (*ver métodos de instancia y campos de instancia*):
```
<xsl:value-of select="date:toString(date:new())" xmlns:date="
java:java.util.Date" />
```


7.2.4 Java: métodos estáticos y campos estáticos

La llamada a un método estático la hace directamente su nombre Java y se hace presentando los argumentos para el método. A los campos estáticos (es decir, los métodos que no toman argumentos), como los campos de valor constante `E` y `PI`, se accede sin especificar ningún argumento.

Ejemplos de código XSLT
Aquí puede ver varios ejemplos de cómo se llama a métodos y campos estáticos:

```
<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)" />
```

Observe que las funciones de extensión anteriores tienen el formato `prefijo:nombreFunción()`. En los tres ejemplos anteriores, el prefijo es `jMath:`, que está asociado al URI de espacio de nombres `java:java.lang.Math`. (El URI de espacio de nombres debe empezar por `java:`. En los ejemplos anteriores se extiende para contener el nombre de la clase (`java.lang.Math`).) La parte `nombreFunción()` de las funciones de extensión debe coincidir con el nombre de una clase pública (p. ej. `java.lang.Math`) seguido del nombre de un método estático público con sus argumentos (como `cos(3.14)`) o de un campo estático público (como `PI()`).

En los tres ejemplos anteriores, el nombre de la clase se incluyó en el URI de espacio de nombres. Si no estuviera en el URI de espacio de nombres, se incluiría en la parte `nombreFunción()` de la función de extensión. Por ejemplo:

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

Ejemplo de XQuery
Un ejemplo de XQuery similar sería:

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

7.2.5 Java: métodos de instancia y campos de instancia

A un método de instancia se le pasa un objeto Java como primer argumento de la llamada a método. Dicho objeto Java suele crearse usando una función de extensión (por ejemplo, una llamada a un constructor) o un parámetro o una variable de hoja de estilos. Un ejemplo de código XSLT de este tipo sería:

```
<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>
```

En el ejemplo anterior el valor del nodo `enrollment/@type` se crea de la siguiente manera:

1. Se crea un objeto con un constructor para la clase `java.util.Date` (con el constructor `date:new()`).
2. Este objeto Java se pasa como argumento del método `jlang.Object.getClass`.
3. El objeto que obtiene el método `getClass` se pasa como argumento al método `jlang.Object.toString`.

El resultado (el valor de `@type`) será una cadena con este valor: `java.util.Date`.

En teoría, un campo de instancia es diferente de un método de instancia porque al campo de instancia no se pasa como argumento un objeto Java propiamente dicho. En su lugar se pasa como argumento un parámetro o variable. Sin embargo, el parámetro o la variable puede contener el valor devuelto por un objeto Java. Por ejemplo, el parámetro `CurrentDate` toma el valor que devolvió un constructor para la clase `java.util.Date`. Este valor se pasa después como argumento al método de instancia `date:toString` a fin de suministrar el valor de `/enrollment/@date`.

7.2.6 Tipos de datos: XPath/XQuery a Java

Cuando se llama a una función Java desde dentro de una expresión XPath/XQuery, el tipo de datos de los argumentos de la función es importante a la hora de determinar a cuál de las clases Java que tienen el mismo nombre se llama.

En Java se siguen estas reglas:

- Si hay más de un método Java con el mismo nombre, pero cada método tiene un número diferente de argumentos, entonces se selecciona el método Java que mejor se ajusta al número de argumentos de la llamada a función.
- Los tipos de datos de cadena, numéricos y booleanos de XPath/XQuery (*ver lista más abajo*) se convierten de forma implícita en el tipo de datos Java correspondiente. Si el tipo XPath/XQuery suministrado se puede convertir a más de un tipo Java (p. ej. `xs:integer`), entonces se selecciona el tipo Java que se declaró para el método seleccionado. Por ejemplo, si el método Java al que se llama es `fx(decimal)` y el tipo de datos XPath/XQuery suministrado es `xs:integer`, entonces `xs:integer` se convierte en el tipo de datos Java `decimal`.

La tabla que aparece a continuación enumera las conversiones implícitas de los tipos de cadena, numéricos y booleanos XPath/XQuery en tipos de datos Java.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitivo), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double y sus clases contenedoras, como `java.lang.Integer`
`xs:float`	`float` (primitivo), `java.lang.Float`, `double` (primitivo)
`xs:double`	`double` (primitivo), `java.lang.Double`
`xs:decimal`	`float` (primitivo), `java.lang.Float`, `double` (primitivo), `java.lang.Double`

Los subtipos de los tipos de datos XML Schema de la tabla anterior (que se usan en XPath y XQuery) también se convierten en los tipos Java correspondientes al tipo antecesor del subtipo.

En algunos casos quizás no sea posible seleccionar el método Java correcto usando la información dada. Por ejemplo, imagine que:

- El argumento presentado es un valor `xs:untypedAtomic` de 10 y está destinado al método `mimétodo(float)`.
- Sin embargo, hay otro método en la clase que toma un argumento de otro tipo de datos: `mimétodo(double)`.
- Puesto que los métodos tienen el mismo nombre y el tipo suministrado (`xs:untypedAtomic`) se puede convertir correctamente tanto en `float` como en `double`, es posible que `xs:untypedAtomic` se convierta en `double` en lugar de en `float`.
- Por consiguiente, el método seleccionado no será el método necesario y quizás no produzca el resultado esperado. Una solución es crear un método definido por el usuario con un nombre diferente y usar ese método.

Los tipos que no aparecen en la lista anterior (p. ej. `xs:date`) no se convertirán y generarán un error. No obstante, tenga en cuenta que en algunos casos, es posible crear el tipo Java necesario usando un constructor Java.

7.2.7 Tipos de datos: Java a XPath/XQuery

Cuando un método Java devuelve un valor y el tipo de datos del valor es un tipo de cadena, numérico o booleano, entonces se convierte en el tipo de datos XPath/XQuery correspondiente. Por ejemplo, los tipos de datos Java `java.lang.Boolean` y `boolean` se convierten en `xsd:boolean`.

Las matrices unidimensionales devueltas por las funciones se extienden en una secuencia. Las matrices multidimensionales no se convierten y, por tanto, deberían ser contenidas.

Cuando se devuelve un objeto Java contenido o un tipo de datos que no es de cadena, numérico ni booleano, puede garantizar la conversión del tipo XPath/XQuery necesario usando primero un método Java (p. ej. `toString`) para convertir el objeto Java en una cadena. En XPath/XQuery la cadena se puede modificar para ajustarse a la representación léxica del tipo necesario y convertirse después en dicho tipo (usando la expresión `cast as`, por ejemplo).

7.3 Funciones de extensión .NET

Si trabaja en la plataforma .NET desde un equipo Windows, puede usar funciones de extensión escritas en cualquier lenguaje .NET (p. ej. C#). Una función de extensión .NET se puede usar dentro de una expresión XPath/XQuery para invocar un constructor, una propiedad o un método (estático o de instancia) de una clase .NET.

A una propiedad de una clase .NET se le llama usando la sintaxis `get_NombrePropiedad()`.

Este apartado tiene varias partes:

- .NET: constructores
- .NET: métodos estáticos y campos estáticos
- .NET: métodos de instancia y campos de instancia
- Tipos de datos: XSLT/XQuery en .NET
- Tipos de datos: .NET en XSLT/XQuery

Formato de la función de extensión
La función de extensión de la expresión XPath/XQuery debe tener este formato `prefijo:nombreFunción()`.

- La parte `prefijo:` está asociada a un URI que identifica la clase .NET.
- La parte `nombreFunción()` identifica el constructor, la propiedad o el método (estático o de instancia) dentro de la clase .NET y, si es necesario, suministra los argumentos.
- El URI debe empezar por `clitype:` (que identifica la función como función de extensión .NET).
- El formato `prefijo:nombreFunción()` de la función de extensión se puede usar con clases del sistema y con clases de un ensamblado cargado. No obstante, si se tiene que cargar una clase, será necesario suministrar parámetros que contengan la información necesaria.

Parámetros
Para cargar un ensamblado se usan estos parámetros:

`asm`	El nombre del ensamblado que se debe cargar.
`ver`	El número de versión (máximo cuatro enteros separados por puntos).
`sn`	El símbolo de clave del nombre seguro del ensamblado (16 dígitos hexadecimales).
`from`	Un URI que da la ubicación del ensamblado (DLL) que se debe cargar. Si el URI es relativo, es relativo al archivo XSLT o XQuery. Si está presente este parámetro, se ignoran los demás parámetros.
`partialname`	El nombre parcial del ensamblado. Se suministra a `Assembly.LoadWith.PartialName()`, que intentará cargar el ensamblado. Si está presente el parámetro `partialname`, se ignoran los demás parámetros.
`loc`	La configuración regional, por ejemplo, `en-US`. La configuración predeterminada es `neutral`.

Si el ensamblado se debe cargar desde un archivo DLL, use el parámetro `from` y omita el parámetro `sn`. Si el ensamblado se debe cargar desde el caché general de ensamblados (GAC), use el parámetro `sn` y omita el parámetro `from`.

Debe insertar un signo de interrogación final antes del primer parámetro y los parámetros deben separarse con un punto y coma (;). El nombre de parámetro da su valor con un signo igual (=), como en el ejemplo que aparece más abajo.

Ejemplos de declaraciones de espacios de nombres

Esto es un ejemplo de una declaración de espacio de nombres en XSLT que identifica la clase del sistema `System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

Esto es un ejemplo de una declaración de espacio de nombres en XSLT que identifica la clase que se debe cargar como `Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

Esto es un ejemplo de una declaración de espacio de nombres en XQuery que identifica la clase del sistema `MyManagedDLL.testClass`:. Existen dos tipos de clases:

1. Cuando el ensamblado se carga desde el GAC:
   ```
   declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
   ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
   ```

2. Cuando el ensamblado se carga desde el archivo DLL (ver las referencias parciales y completas):
   ```
   declare namespace
cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
   Projects/extFunctions/MyManagedDLL.dll;

   declare namespace
   cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
   ```

Ejemplo de código XSLT

Aquí puede ver un ejemplo de código XSLT que llama a funciones de la clase del sistema `System.Math`:

```
<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>
```

La declaración de espacio de nombres del elemento `math` asocia el prefijo `math:` al URI `clitype:System.Math`. La parte inicial `clitype:` del URI indica que lo que sigue identifica una clase del sistema o una clase cargada. El prefijo `math:` de las expresiones XPath asocia las funciones de extensión al URI (y, por extensión, a la clase) `System.Math`. Las funciones de extensión identifican métodos en la clase `System.Math` y presenta argumentos cuando es necesario.

Ejemplo de código XQuery

Aquí puede ver un fragmento de código XQuery similar al ejemplo anterior:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

Tal y como ocurre con el código XSLT anterior, la declaración de espacio de nombres identifica la clase .NET, en este caso una clase del sistema. La expresión XQuery identifica el método al que se debe llamar y presenta el argumento.

7.3.1 .NET: constructores

Una función de extensión se puede usar para llamar a un constructor .NET. A todos los constructores se les llama con la pseudofunción `new()`. Si hay más de un constructor para una clase, entonces se selecciona el constructor que más se ajusta al número de argumentos suministrados. Si no se encuentra ningún constructor que coincida con los argumentos suministrados, entonces se genera el error "`No constructor found`".

Constructores que devuelven tipos de datos XPath/XQuery
Si el resultado de una llamada a un constructor .NET se puede convertir de forma implícita en tipos de datos XPath/XQuery, entonces la función de extensión .NET devuelve una secuencia que es un tipo de datos XPath/XQuery.

Constructores que devuelven objetos .NET
Si el resultado de una llamada a un constructor .NET no se puede convertir a un tipo de datos XPath/XQuery adecuado, entonces el constructr crea un objeto .NET contenido con un tipo que es el nombre de la clase que devuelve dicho objeto. Por ejemplo, si se llama al constructor para la clase `System.DateTime` (con `System.DateTime.new()`), entonces se devuelve un objeto que tiene un tipo `System.DateTime`.

Puede que el formato léxico del objeto devuelto no coincida con el formato léxico de un tipo de datos XPath. En estos casos, el valor devuelto (i) debe convertirse al formato léxico del tipo de datos XPath pertinente y (ii) debe convertirse en el tipo de datos XPath necesario.

Se pueden hacer tres cosas con un objeto .NET creado con un constructor:

- Se puede usar dentro de una variable:
```
<xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
xmlns:date="clitype:System.DateTime" />
```
- Se puede pasar a una función de extensión (ver *Métodos de instancia y campos de instancia*):
```
<xsl:value-of select="date:ToString(date:new(2008, 4, 29))" xmlns:date
="clitype:System.DateTime" />
```
- Se puede convertir en un tipo de cadena, numérico o booleano:
```
<xsl:value-of select="xs:integer(data:get_Month(date:new(2008, 4, 29))
" xmlns:date="clitype:System.DateTime" />
```


7.3.2 .NET: métodos estáticos y campos estáticos

La llamada a un método estático la hace directamente su nombre y se hace presentando los argumentos para el método. El nombre usado en la llamada debe ser el mismo que un método estático público de la clase especificada. Si el nombre del método y el número de argumentos que se dio en la llamada a función coincide con algún método de la clase, entonces los tipos de los argumentos presentados se evalúan para encontrar el resultado ideal. Si no se encuentra ninguna coincidencia, se emite un error.

Nota: un campo de una clase .NET se trata como si fuera un método sin argumentos. Para llamar a una propiedad se usa la sintaxis `get_nombrePropiedad()`.

Ejemplos

Este ejemplo de código XSLT muestra una llamada a un método con un argumento (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

Este ejemplo de código XSLT muestra una llamada a un campo (que se trata como si fuera un método sin argumentos) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="
clitype:System.Double"/>
```

Este ejemplo de código XSLT muestra una llamada a una propiedad (la sintaxis es `get_nombrePropiedad()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')" xmlns:string="
clitype:System.String"/>
```

Este ejemplo de código XQuery muestra una llamada a un método con un argumento (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

7.3.3 .NET: métodos de instancia y campos de instancia

Un método de instancia es un método al que se le pasa un objeto .NET como primer argumento de la llamada al método. Este objeto .NET se suele crear usando una función de extensión (por ejemplo, una llamada a un constructor) o un parámetro o una variable de una hoja de estilos. Un ejemplo de código XSLT para este tipo de método sería:

```
<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>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

En el ejemplo anterior, se usó un constructor `System.DateTime` (`new(2008, 4, 29)`) para crear un objeto .NET de tipo `System.DateTime`. Este objeto se creó dos veces, una vez como valor de la variable `releasedate`, y otra vez como primer y único argumento del método `System.DateTime.ToString()`. Al método de instancia `System.DateTime.ToString()` se le llama dos veces, ambas con el constructor `System.DateTime` (`new(2008, 4, 29)`) como primer y único argumento. En una de estas instancias, se usó la variable `releasedate` para obtener el objeto .NET.

Métodos de instancia y campos de instancia
La diferencia entre un método de instancia y un campo de instancia es solo teórica. En un método de instancia, se pasa directamente un objeto .NET como argumento. En un campo de instancia, se pasa un parámetro o una variable (aunque el parámetro o la variable puede contener un objeto .NET). Por ejemplo, en el código del ejemplo anterior, la variable `releasedate` contiene un objeto .NET y esta es la variable que se pasa como argumento de `ToString()` en el segundo constructor de elemento `date`. Por tanto, la instancia `ToString()` del primer elemento `date` es un método de instancia, mientras que la segunda se considera un campo de instancia. El resultado es el mismo en ambos casos.

7.3.4 Tipos de datos: XPath/XQuery a .NET

Cuando se usa una función de extensión .NET dentro de una expresión XPath/XQuery, los tipos de datos de los argumentos de la función son importantes para determinar a cuál de los métodos .NET que tienen el mismo nombre se está llamando.

En .NET se siguen estas normas:

- Si en una clase hay varios métodos que tienen el mismo nombre, solamente se pueden seleccionar los métodos que tienen el mismo número de argumentos que la llamada a función.
- Los tipos de datos de cadena, numéricos y booleanos XPath/XQuery (*ver lista más abajo*) se convierten de forma implícita en el tipo de datos .NET correspondiente. Si el tipo XPath/XQuery suministrado se puede convertir en más de un tipo .NET (p. ej. `xs:integer`), entonces se selecciona el tipo .NET que se declaró para el método seleccionado. Por ejemplo, si el método .NET al que se está llamando es `fx(double)` y el tipo de datos XPath/XQuery suministrado es `xs:integer`, entonces se convierte `xs:integer` en el tipo de datos .NET `double`.

La tabla que aparece a continuación enumera las conversiones implícitas de los tipos de cadena, numéricos y booleanos XPath/XQuery en tipos de datos .NET.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Los subtipos de los tipos de datos XML Schema de la tabla anterior (que se usan en XPath y XQuery) también se convierten en los tipos .NET correspondientes al tipo antecesor del subtipo.

En algunos casos quizás no sea posible seleccionar el método .NET correcto usando la información dada. Por ejemplo, imagine que:

- El argumento presentado es un valor `xs:untypedAtomic` de 10 y está destinado al método `mimétodo(float)`.
- Sin embargo, hay otro método en la clase que toma un argumento de otro tipo de datos: `mimétodo(double)`.
- Puesto que los métodos tienen el mismo nombre y el tipo suministrado (`xs:untypedAtomic`) se puede convertir correctamente tanto en `float` como en `double`, es posible que `xs:untypedAtomic` se convierta en `double` en lugar de en `float`.
- Por consiguiente, el método seleccionado no será el método necesario y puede que no produzca el resultado esperado. Una solución es crear un método definido por el usuario con un nombre diferente y usar ese método.

Los tipos que no aparecen en la lista anterior (p. ej. `xs:date`) no se convertirán y generarán un error.

7.3.5 Tipos de datos: .NET a XPath/XQuery

Cuando un método .NET devuelve un valor y el tipo de datos del valor es un tipo de cadena, numérico o booleano, entonces se convierte en el tipo de datos XPath/XQuery correspondiente. Por ejemplo, el tipo de datos .NET `decimal` se convierte en `xsd:decimal`.

Cuando se devuelve un objeto .NET o un tipo de datos que no es de cadena, numérico ni booleano, puede garantizar la conversión del tipo XPath/XQuery necesario usando primero un método .NET (p. ej. `System.DateTime.ToString()`) para convertir el objeto .NET en una cadena. En XPath/XQuery la cadena se puede modificar para ajustarse a la representación léxica del tipo necesario y convertirse después en dicho tipo (usando la expresión `cast as`, por ejemplo).

7.4 Scripts MSXSL para XSLT

El elemento `<msxsl:script>` contiene funciones y variables definidas por el usuario a las que se puede llamar desde dentro de expresiones XPath en la hoja de estilos XSLT. El elemento `<msxsl:script>` es un elemento de nivel superior, es decir, debe ser un elemento secundario de `<xsl:stylesheet>` o `<xsl:transform>`.

El elemento `<msxsl:script>` debe estar en el espacio de nombres `urn:schemas-microsoft-com:xslt` (*ver ejemplo más abajo*).

Lenguaje de scripting y espacio de nombres

El lenguaje de scripting utilizado dentro del bloque se especifica en el atributo `language` del elemento `<msxsl:script>` y el espacio de nombres que se debe usar para las llamadas a función desde expresiones XPath se identifica con el atributo `implements-prefix`:

```
<msxsl:script language="lenguaje-de-scripting" implements-prefix="prefijo-
espacioNombres-usuario">

   función-1 o variable-1
   ...
   función-n o variable-n

</msxsl:script>
```

El elemento `<msxsl:script>` interactúa con Windows Scripting Runtime, de modo que dentro del elemento `<msxsl:script>` solamente se pueden usar lenguajes que estén instalados en el equipo. Para poder usar scripts MSXSL es necesario tener **instalada la plataforma .NET Framework 2.0 (o superior)**. Por tanto, los lenguajes de scripting .NET se pueden usar dentro del elemento `<msxsl:script>`.

El atributo `language` admite los mismos valores que el atributo `language` del elemento HTML `<script>`. Si no se especifica el atributo `language`, entonce se asume Microsoft JScript por defecto.

El atributo `implements-prefix` toma un valor que es un prefijo de un espacio de nombres declarado dentro del ámbito. Este espacio de nombres suele ser un espacio de nombres de usuario que se reservó para una biblioteca de funciones. Todas las funciones y variables definidas dentro del elemento `<msxsl:script>` están en el espacio de nombres identificado por el prefijo indicado en el atributo `implements-prefix`. Cuando se llama a una función desde dentro de una expresión XPath, el nombre de función completo debe estar en el mismo espacio de nombres que la definición de función.

Ejemplo

Aquí puede ver un ejemplo de una hoja de estilos XSLT que usa una función definida dentro de un elemento `<msxsl:script>`.

```
<?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>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Tipos de datos

Los valores de los parámetros que se pasan dentro y fuera del bloque de script solamente pueden ser tipos de datos XPath. Esta restricción no afecta a los datos que se pasan las funciones y variables situadas dentro del bloque de script.

Ensamblados

Puede importar un ensamblado al script usando el elemento `msxsl:assembly`. El ensamblado se identifica con un nombre o un URI. El ensamblado se importa cuando se compila la hoja de estilos. Aquí puede ver cómo se usa el elemento `msxsl:assembly`:

```
<msxsl:script>
    <msxsl:assembly name="miEnsamblado.nombreEnsamblado" />
    <msxsl:assembly href="rutaDelEnsamblado" />

    ...

</msxsl:script>
```

El nombre de ensamblado puede ser un nombre completo, como:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

o un nombre abreviado, como `"miEnsamblado.Draw"`.

Espacios de nombres

Puede declarar espacios de nombres con el elemento `msxsl:using`. Esto permite escribir las clases del ensamblado en el script sin sus espacios de nombres, lo cual le permitirá ahorrar mucho tiempo. Aquí puede ver cómo se usa el elemento `msxsl:using` para declarar espacios de nombres.

```
<msxsl:script>
    <msxsl:using namespace="ENmiEnsamblado.NombreEspaciodenombres" />
```



```
    ...
</msxsl:script>
```

El valor del atributo `namespace` es el nombre del espacio de nombres.

Índice

C

D

E

F

I

L

M

R

S

T

V

W

X