

XML Format Plug-in

User's Guide

Version 3.4

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA, Inc. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with that agreement. No part of this guide may be reproduced or retransmitted in any form or by any means electronic, mechanical, or otherwise, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of BEA, Inc.

Copyright © 2001-2007 BEA, Inc. All rights reserved. All BEA Products are trademarks or registered trademarks of BEA, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

XML.....	5
TERMINOLOGY.....	5
CREATING AN XML FORMAT.....	6
FROM DESIGNER UI USING CREATION WIZARD	6
FROM COMMAND LINE USING SCHEMA2CAR UTILITY	24
XML EXTERNAL MESSAGE UI.....	25
BINDING XML SCHEMA TO DESIGNER	26
SIMPLE TYPE DEFINITION.....	27
<i>Atomic Type</i>	28
Type Mapping	28
Built-in atomic type.....	33
User-derived atomic type.....	36
Facets.....	40
<i>List Type</i>	43
<i>Union Type</i>	47
<i>Simple Type Designer Support</i>	48
COMPLEX TYPE DEFINITION.....	49
Aggregation.....	51
<i>Simple Content (Complex Type Definition)</i>	56
<i>Complex Content (Complex Type Definition)</i>	57
Type Substitution	62
Complex Type Designer Support.....	68
ATTRIBUTE GROUP DEFINITION	69
Attribute Group Designer Support	70
MODEL GROUP.....	71
Model Group Flattening Rules	73
Model Group Represented as Section.....	81
Model Group Definition Designer Support	82
SCHEMA ATTRIBUTE DECLARATION	83
Ignore Fixed Attributes (Schema Attribute Declaration)	85
Attribute Declaration Designer Support	85
SCHEMA ELEMENT DECLARATION	86
Ignore Fixed Elements.....	88
Element Substitution.....	88
Element References	90
Nil Value Elements	91
Element Declaration Designer Support.....	93
ANNOTATION	94
Annotation Designer Support	96
WILDCARD	96
Wildcard Designer Support	98
IDENTITY-CONSTRAINT DEFINITION	98
XML NAME MAPPING	98
XML Name of Element / Attribute to Designer Name.....	99
Model Group Name to Designer Name	100
Collisions and Conflicts.....	102
W3C XML SCHEMA SUPPORT	103
BINDING DTD TO DESIGNER.....	103

DTD ELEMENT DECLARATION	104
<i>Element with Character Data Only</i>	104
<i>Element with Standard Content</i>	105
<i>Element with Mixed Content</i>	107
<i>Element with Any Content</i>	108
<i>Element with No Content Allowed</i>	109
GROUP DECLARATION	110
<i>Sequence Declaration</i>	111
<i>Choice Declaration</i>	113
<i>Group Flattening Rules</i>	114
<i>Group Represented as Section</i>	118
DTD ATTRIBUTED DECLARATION	119
<i>Enumeration Type</i>	121
<i>Ignore Fixed Attributes (DTD AttributeDeclaration)</i>	122
DESIGNING XML STRUCTURE MANUALLY IN DESIGNER	123
FIELD REPRESENTATION	125
<i>Field Properties pane</i>	125
SECTION REPRESENTATION	127
<i>Section Properties pane</i>	128
ADDING ELEMENT	129
<i>Simple Type</i>	129
Non-repeating element	130
Repeating element	131
<i>Complex Type</i>	133
Simple Content (Complex Type)	133
Complex Content (Complex Type)	138
ADDING ATTRIBUTE	146
ADDING CHARACTER CONTENT	147
ADDING WILDCARD	149
<i>any element</i>	149
<i>anyAttribute element</i>	150
ADDING GROUP	152
RECONFIGURE SCHEMA	155
FORMAT OPTION	155
GENERAL	156
NAMESPACES	157
XML DECLARATION	161
XML PARSER OPTIONS	163
ENTITY REFERENCES	165
CATALOG SUPPORT	165
REPAIR FUNCTIONALITY IN XML PLUGIN	166
GLOSSARY	168

XML

XML Plugin allows you to parse and write XML data; it allows you to view your pre-defined schema (DTD/XSD) in an intuitive and user-friendly form by importing the same in the Designer UI and also allows you to manually create XML Schema from scratch.

See Also:

[Terminology](#)

[Creating an XML Format](#)

[XML External Message UI](#)

[Binding XML Schema to Designer](#)

[Binding DTD to Designer](#)

[Designing XML Structure manually in Designer](#)

[Reconfigure Schema](#)

[Format Option](#)

[Catalog Support](#)

[Repair functionality in XML plugin](#)

[Glossary](#)

Terminology

XML

The extensible Markup Language (XML) is a subset of SGML (the Structured Generalized Markup Language). It provides a uniform method for describing and exchanging structured data in an open, text-based format. An XML document (which is an instance of an XML Schema) is a tree of elements. An element may have a set of attributes, and may contain other elements, text, or a mixture thereof.

Schema

A schema contains a set of rules that constrains the structure and content of a XML document's components, i.e., its elements, attributes and text. A schema is, in other words, a specification of the syntax and semantics of a (potentially infinite) set of XML documents. An XML document is said to be valid with respect to a schema if, and only if, it satisfies the constraints specified in the schema.

DTD

Document Type Definition (DTD) is a schema for an XML document. It specifies what elements and attributes may be used in a particular type of XML document and what their structure and nesting may be.

W3C XSD (XML Schema)

The “XML Schema” language is an ongoing effort by the World Wide Web Consortium to supplant DTDs with a more flexible and powerful system to describe the structure of conforming XML documents, including provisions for defining data types.

See Also:

[Glossary](#)

[Creating an XML Format](#)

[XML](#)

Creating an XML Format

An XML message format can be created in the Designer in two ways

- By importing a Schema (DTD or W3C XSD) (Auto-creation)
- By manually adding various elements and setting properties.

A schema (DTD or W3C XSD) can be imported in the Designer either from the Designer UI or from the command line without even going to the Designer. In case the user imports a DTD, depending on his requirement, he has to change the Designer Type or XML Type of the elements / attributes.

See Also:

[Terminology](#)

[Glossary](#)

[From Designer UI using Creation Wizard](#)

[From command line using Schema2Car utility](#)

[XML External Message UI](#)

[Binding XML Schema to Designer](#)

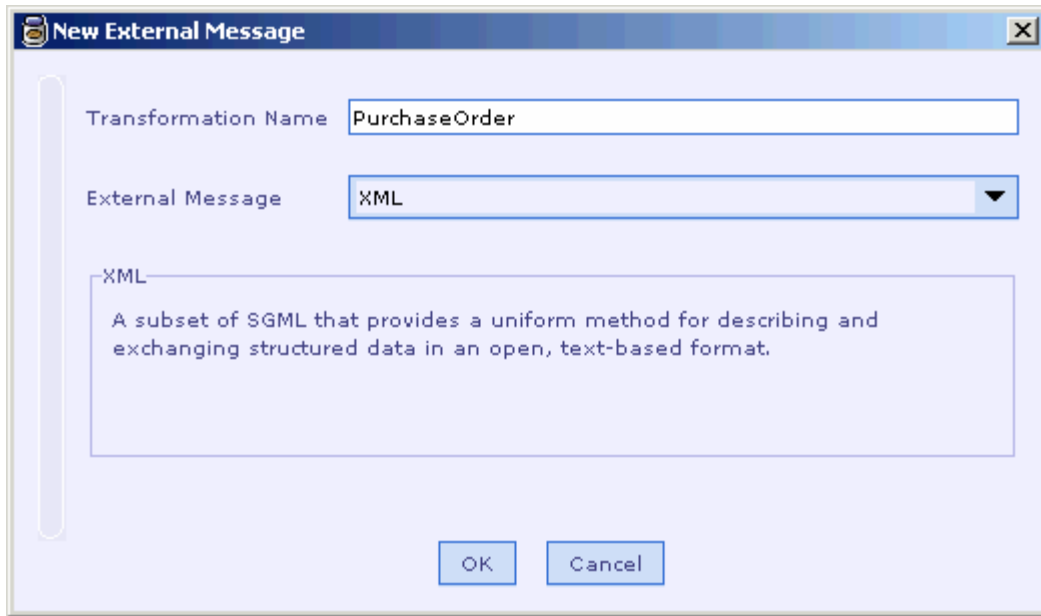
[Binding DTD to Designer](#)

[Designing XML Structure manually in Designer](#)

From Designer UI using Creation Wizard

There is a creation wizard that guides you through the schema import. To invoke the creation wizard, follow the steps given under:

1. Right-click the Cartridge node in the Designer and select the New External Message menu item from the context menu.
2. In the New External Message dialog that appears enter the Transformation Name and select XML from the External Message list box. Click OK.



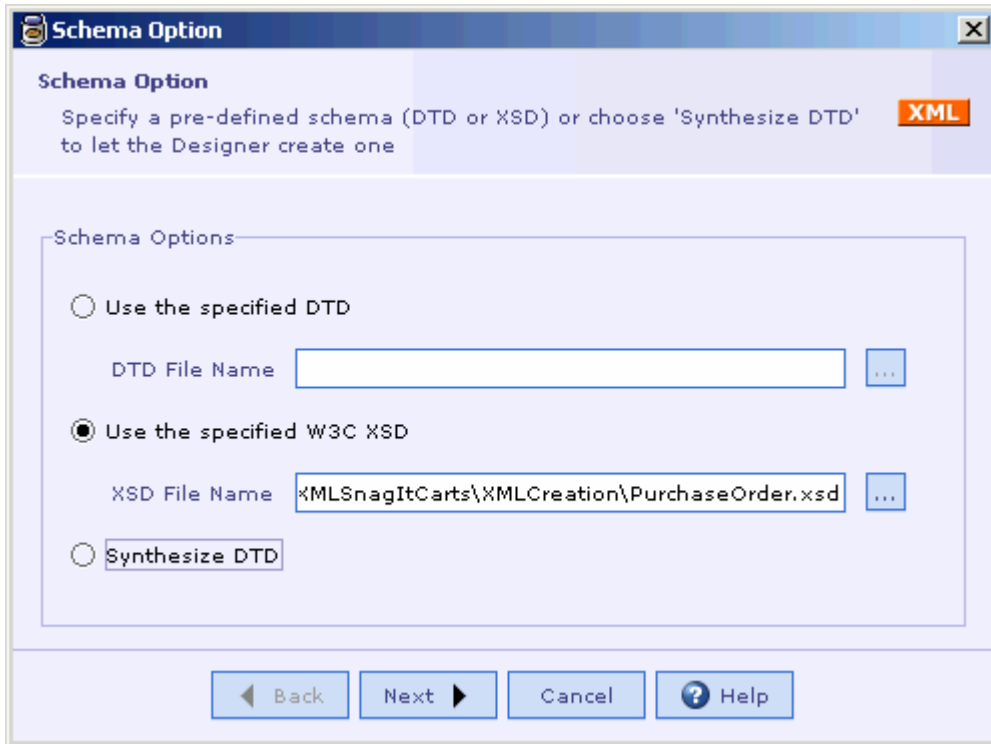
The various windows in the wizard are explained hereunder.

Schema Option window

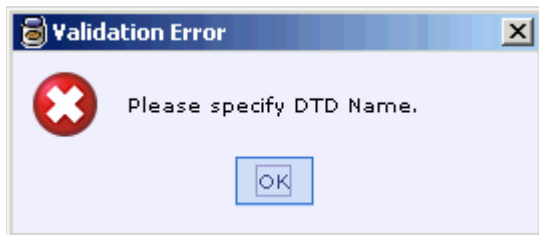
In the **Schema Option** dialog that appears, three options are available as listed below:

SI No.	Option	Description
i)	"Use the specified DTD"	Allows you to import a DTD.
ii)	"Use the specified W3C XSD"	Allows you to import a W3C XSD.
iii)	"Synthesize DTD"	<p>Allows you to manually design a XML message structure.</p> <p>Automatically generates a DTD (with the specified root element name) based on the XML structure created by you, on the cartridge being validated.</p>

If you select any of the first two options, you have to specify the path of the DTD / XSD file in the corresponding text box or alternatively, you can click the ellipsis button which would open the “Open” window from where you can select your required DTD / XSD file.

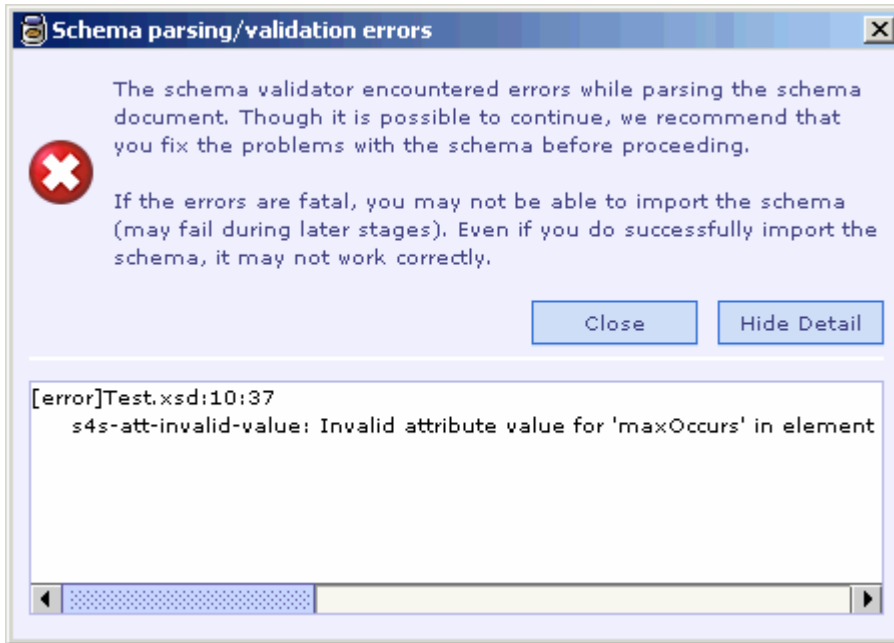


After selecting either of the first two options, if you click the “Next” button without specifying the DTD / XSD file, you will get the following error window.

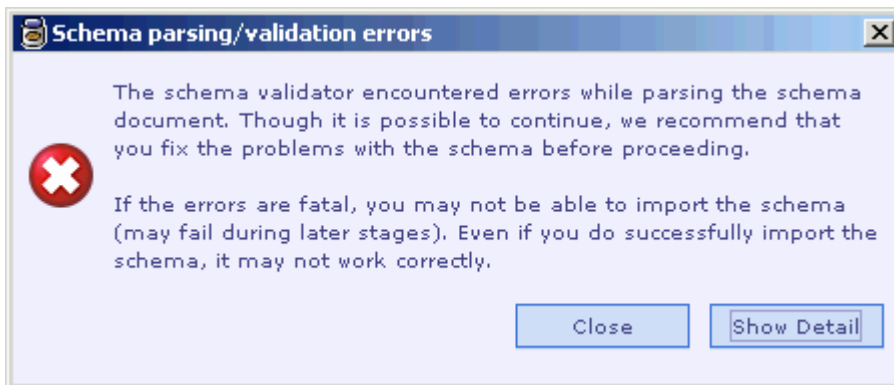


Schema Parsing/Validation errors window

The schema specified by you is automatically validated by the Designer. If you specify an invalid schema and click the “Next” button in the “Schema Option” window, the following error window with warning message would be generated:



If you click the “Hide Detail” button, in the above window, the pane showing the details of the error would be hidden and the “Hide Detail” button which is a toggle becomes “Show Detail” as seen in the following picture:



(Note that you can proceed with the creation wizard by clicking the “Close” button in the above window, however, in case of fatal errors in the schema, you would not be able to proceed further or the schema would not be imported. Also note that your schema should be in the same location as that of the cartridge in which it is imported, otherwise, it would result in code generation error.)

After properly selecting an option and specifying the required DTD / XSD file, if required, if you click the “Next” button in the “Schema Option” window, “Mode of Operation” window would be opened.

The path taken by the creation wizard for each of the above-mentioned three options is outlined below:

i) Use the specified DTD

Window Name	Purpose
Schema parsing/validation errors	To warn the user in case of invalid schema
Mode of Operation	Lets you choose between Batch Mode and Message Mode
Root Element	Gives information about root element / Lets you choose root element in case of multiple global elements in schema
Schema Import Options	Lets you customize your message structure. Only some options are applicable.
Header & Trailer (In case "Message Mode" is selected)	Lets you select the required Header / Trailer elements for your message.
Batch Mode (In case "Batch Mode" is selected)	Lets you select the required Body / Record / Header / Trailer elements for your message.
Finish	Provides information about the chosen DTD file, Header and Trailer element names if specified.

For details regarding DTD to XML bindings, refer the chapter ["Binding DTD to Designer"](#).

ii) Use the specified W3C XSD

Window Name	Purpose
Schema parsing/validation errors	To warn the user in case of invalid schema.
Mode of Operation	Lets you choose between Batch Mode and Message Mode.
Root Element	Gives information about root element / Lets you choose root element in case of multiple global elements in schema.
Schema Import Options	Lets you customize your message structure. All options are applicable.
Header & Trailer (In case "Message Mode" is selected)	Lets you select the required Header / Trailer elements for your message.
Batch Mode (In case "Batch Mode" is selected)	Lets you select the required Body / Record / Header / Trailer elements for your message.
Finish	Provides information about the chosen XSD file, Header and Trailer element names if specified

For details regarding XSD to XML bindings, refer the chapter ["Binding XML Schema to Designer"](#).

iii) Synthesize DTD

Window Name	Purpose
Mode of Operation	Lets you choose between Batch Mode and Message Mode
Message Mode Tags	Lets you specify the required names

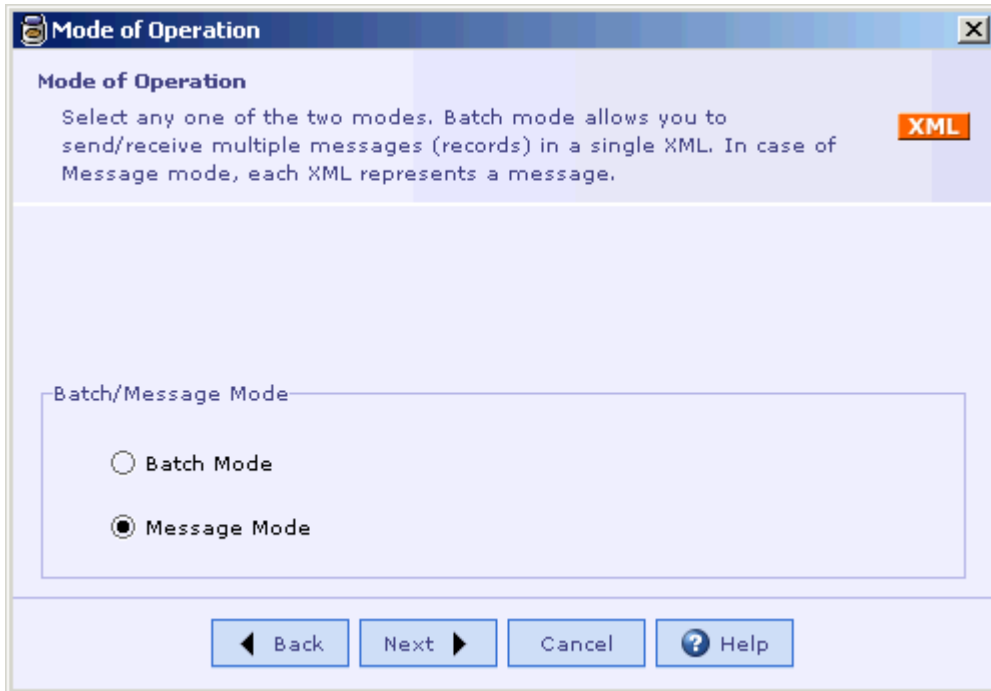
(In case "Message Mode" is selected) Batch Mode Tags (In case "Batch Mode" is selected)	for the Root / Header / Record / Trailer elements for your message Lets you specify the root element name and other options.
Finish	Displays the top-level structure of the DTD.

After creating an empty XML message structure using the above option and designing the XML structure by adding fields / sections, if you validate the cartridge, a DTD file with name as that of the root element name specified by you in the "Message Mode Tags" / "Batch Mode Tags" window is automatically generated in the same location as that of the cartridge. This DTD file is based on the XML structure created by you, and gets updated whenever you change the message structure in the Designer, save and validate the same.

For details regarding manual creation of XML message structure, refer the chapter ["Designing XML Structure manually in Designer"](#).

Mode of Operation window

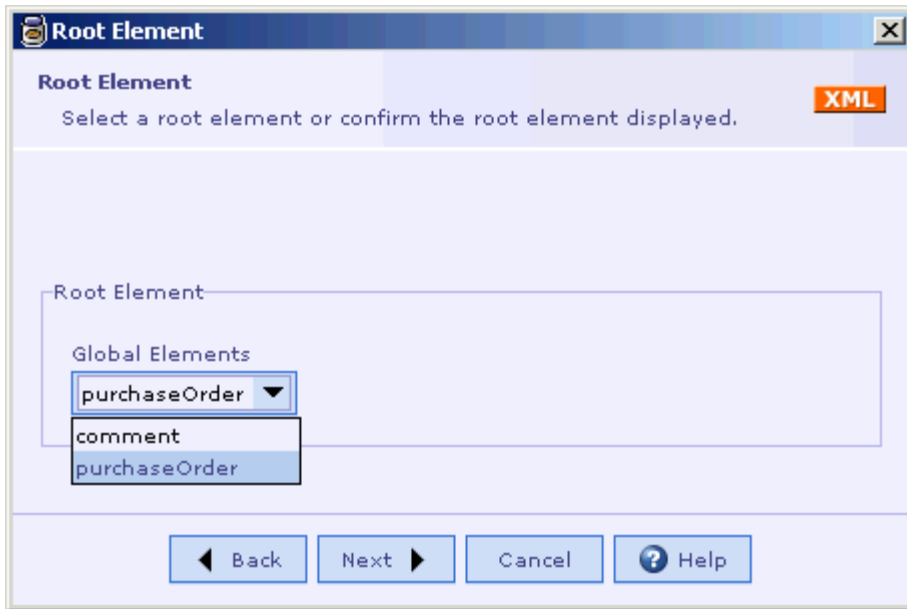
In the '**Mode of Operation**' window, you have two options to choose from viz., "Batch Mode" and "Message Mode". Batch Mode allows you to send/receive multiple records as part of a single message, whereas Message Mode allows you to send/receive single record as part of your message.



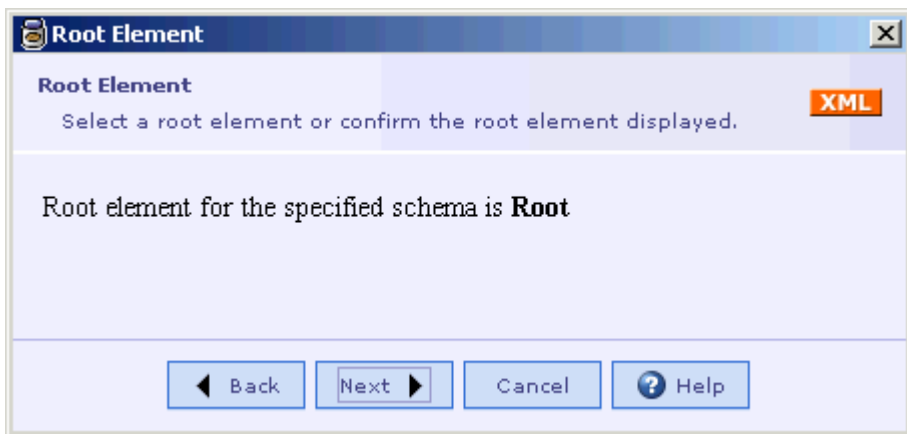
After selecting the required mode, if you click the “Next” button, “Root Element” window would be opened, provided you have selected the DTD / XSD option in the ‘Schema Option’ window. However, if you had selected the third option viz., “Synthesize DTD”, “Message Mode Tags” / “Batch Mode Tags” window would be opened depending on the mode selected. Note that the “Back” button is enabled in this window, clicking which you can go back to the “Schema Option” window.

Root Element window

This window appears in the creation wizard only if you had selected the option of importing a DTD / XSD in the ‘Schema Option’ window. This window has two different displays based on the selected schema. If your schema has more than one top-level element, they are displayed in the “Global Elements” list box, from which you can select the required root element as seen in the following picture:



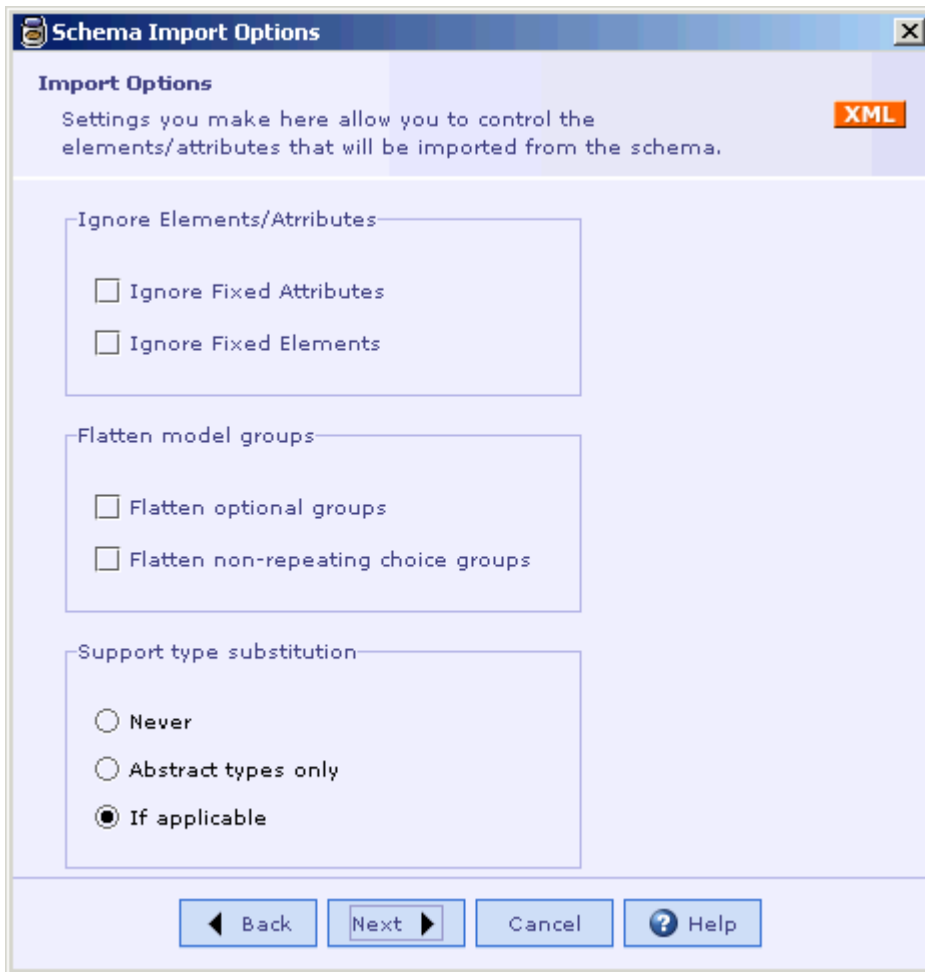
However, if your schema has just one top-level element, the same is automatically recognized as the root element and the information is displayed as seen under:



After selecting the required root element, if applicable, if you click the "Next" button, "Schema Import Options" window would be opened.

Schema Import Options window

This window appears in the creation wizard only if you had selected the option of importing a DTD / XSD in the 'Schema Option' window. This window provides you a set of options to customize your message structure. The options are categorized into three sections as seen in the following picture depicting the default selection:



i) Ignore Elements/Attributes

This section has the following two options:

- Ignore Fixed Attributes (Applicable for both DTD & XSD)

If you want to exclude fixed value attributes in your schema from getting imported in the message structure, you can check this checkbox. In this case, the fixed attributes will not find place in the imported message structure. Refer [Ignore Fixed Attributes](#) under the chapter “Attribute Declaration”.

- Ignore Fixed Elements (Applicable for XSD alone)

If you want to exclude the elements in your XSD for which the “Fixed” attribute is set, from getting imported in the message structure, you can check this checkbox. In this case, the fixed value elements will not find place in the imported message structure. Refer [Ignore Fixed Elements](#) under the chapter “Element Declaration”.

ii) Flatten model groups (Applicable for both DTD & XSD)

The options under this section let you reduce the number of sections in your message structure, which in turn would reduce the strain during mapping. This section has the following two options:

- Flatten optional groups

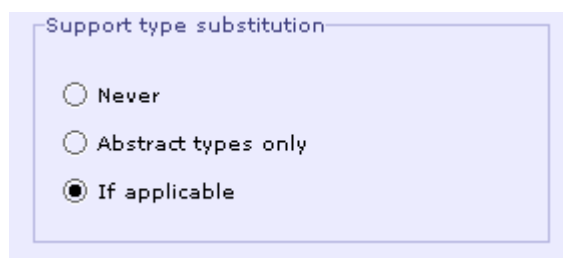
If you do not want the non-repeating optional groups in your schema to be represented as a section (which is the default behavior) in your imported message structure, you have to check this checkbox. If this option is selected, the group is removed and its child elements are aggregated to the parent of the group. For more details (specific to XSD), refer [“Optional group flattening”](#) under “Flattening Rules” chapter. Note that in case of the optional group being a Choice, you have to select the next option as well for this feature to work.

- Flatten non-repeating choice groups

If you do not want the non-repeating choice groups in your schema to be represented as a section in your message structure, you have to check this checkbox (as well as the previous checkbox). If this option along with the previous one is selected, the choice group is removed and its child elements are aggregated to the parent of the group. For more details (specific to XSD), refer [Choice group flattening](#) under “Flattening Rules” chapter.

iii) Support type substitution (Applicable only for XSD)

This feature allows the user to apply the feature of type substitution provided in the XSD. Three options are provided as seen in the following picture:



- Never

If this option is selected, then type substitution feature will not be supported even if it is applicable.

- Abstract types only

If this option is selected, then type substitution feature will be supported only for those elements whose complex type definition has the “abstract” attribute set to true.

- If applicable

This is the default option and is provided to support the type substitution feature for applicable elements.

For detailed explanation of each option, refer [“Type Substitution”](#).

After selecting the required import options, if you click the “Next” button, “Header & Trailer” window would be opened.

Header & Trailer window

This window appears in the creation wizard only if you had selected the option of importing a DTD / XSD in the ‘Schema Option’ window and also “Message Mode” option in the “Mode of Operation” window. In this window, you can select the required Header / Trailer elements for your message as seen in the following picture:



- Header

If you check the “Header” checkbox, the corresponding list box is enabled with the list of all top-level elements declared in your schema. You have to select the element that should appear in the Header section of the message.

- Trailer

If you check the “Trailer” checkbox, the corresponding list box is enabled with the list of all top-level elements declared in your schema. You have to select the element that should appear in the Trailer section of the message.

Note that checking the Header / Trailer checkboxes is optional and you can very well skip this step and proceed. If you click the “Next” button in this window, the “Finish” window, which is the final window in the creation wizard, would be opened.

Batch Mode window

This window appears in the creation wizard only if you had selected the option of importing a DTD / XSD in the ‘Schema Option’ window and also “Batch Mode” option in the “Mode of Operation” window. In this window, you can select the required Body / Record / Header / Trailer elements for your message as seen in the following picture:

Batch Mode

☒ Separate Body element that consists of records

Body Element Name:

Record Element Name:

☐ Root element consists of records

Record Element Name:

☒ Header

Header Element Name:

☒ Trailer

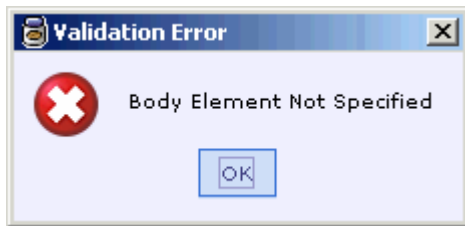
Trailer Element Name:

◀ Back Next ▶ Cancel ? Help

The options in this window are described below:

- Separate Body element that consists of records

In Batch mode, you can have multiple records. If the record elements in your schema are contained in an element other than the root element (say, "Body" element) then you have to select this option. If this option is selected, then the "Body Element Name" list box and the "Record Element Name" text box under this option are enabled. In the "Body Element Name" list box, all top-level elements declared in your schema are listed, from which you have to select the element name in your schema, which contains the record elements. In the "Record Element Name" textbox, you have to specify the name of your record element. These two inputs are mandatory for this option and trying to proceed to the next window without specifying them would generate the following error message window:



- Root element that consists of records

If the record elements in your schema are contained in the root element itself, you have to select this option. If this option is selected, then the "Record Element Name" text box under this option is enabled. In the "Record Element Name" textbox, you have to specify the name of your record element. This input is mandatory for this option and trying to proceed without specifying the same would generate "Validation Error" window saying, "Record Element not specified".

- Header

If you check the "Header" checkbox, the corresponding list box is enabled with the list of all top-level elements declared in your schema. You have to select the element that should appear in the Header section of the message. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate "Validation Error" window saying, "Header Element not specified".

- Trailer

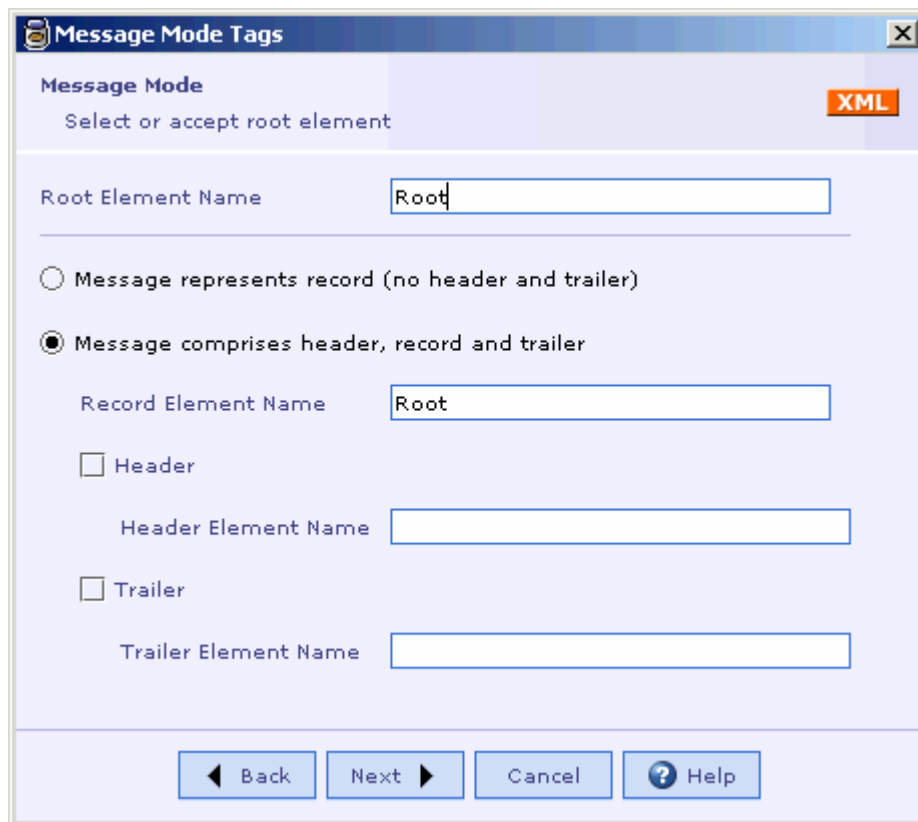
If you check the "Trailer" checkbox, the corresponding list box is enabled with the list of all top-level elements declared in your schema. You have to select the element that should appear in the Trailer section of the message. This input is

mandatory for this option and trying to proceed to the next window without specifying the same would generate "Validation Error" window saying, "Trailer Element not specified".

After selecting the required options, if you click the "Next" button in this window, the "Finish" window, which is the final window in the creation wizard, would be opened.

Message Mode Tags window

This window appears in the creation wizard only if you had selected the "Synthesize DTD" option in the 'Schema Option' window and also "Message Mode" option in the "Mode of Operation" window. In this window, you can specify the required names for the Root / Header / Record / Trailer elements for your message as seen in the following picture:



- Root Element Name

In this textbox, you have to specify the required name for the root element of your message. This input is mandatory and trying to proceed to the next window without specifying the same would generate "Validation Error" window saying, "Root Element not specified".

- Message represents record (no header and trailer)

If you do not want your message to have Header and Trailer information, you can select this option.

- Message comprises header, record and trailer

If you want your message to have Header and Trailer information, you can select this option. If this option is selected, then the “Record Element Name” textbox and the “Header” and “Trailer” check boxes get enabled.

- Record Element Name

In the “Record Element Name” text box, you have to specify the root tag of the data layer. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate “Validation Error” window saying, “Record Element not specified”.

- Header

If you want your message to have Header information, you can check this check box and specify the required name in the “Header Element Name” text box. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate “Validation Error” window saying, “Header Element not specified”.

- Trailer

If you want your message to have Trailer information, you can check this check box and specify the required name in the “Trailer Element Name” text box. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate “Validation Error” window saying, “Trailer Element not specified”.

After selecting the required options, if you click the “Next” button in this window, the “Finish” window, which is the final window in the creation wizard, would be opened.

Batch Mode Tags window

This window appears in the creation wizard only if you had selected the “Synthesize DTD” option in the ‘Schema Option’ window and also “Batch Mode” option in the “Mode of Operation” window. In this window, you can specify the root element name and also the other options as seen in the following picture:

Batch Mode Tags

Batch mode elements
Select the elements of a batch XML

Root Element Name

☐ Separate Body element that consists of records

Body Element Name

Record Element Name

☒ Root element consists of records

Record Element Name

☐ Header

Header Element Name

☐ Trailer

Trailer Element Name

◀ Back Next ▶ Cancel ? Help

- **Root Element Name**

In this textbox, you have to specify the required name for the root element of your message. This input is mandatory and trying to proceed to the next window without specifying the same would generate “Validation Error” window saying, “Root Element not specified”.

- **Separate Body element that consists of records**

In Batch mode, you have multiple records. If the record elements in your message are to be contained in an element other than the root element (say, “Body” element) then you have to select this option. If this option is selected, then the “Body Element Name” and the “Record Element Name” text boxes under this option are enabled. In the “Body Element Name” text box, you have to specify the element name, which should contain the record elements. In the “Record Element Name” textbox, you have to specify the required name for the record element. These two inputs are mandatory for this option and trying to proceed to the next window without specifying them would generate “Validation Error” window saying “Body Element not specified” and “Record Element not specified”.

- Root element consists of records

If the record elements in your message are to be contained in the root element itself, you have to select this option. If this option is selected, then the "Record Element Name" text box under this option is enabled. In the "Record Element Name" textbox, you have to specify the required name for your record element. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate "Validation Error" window saying, "Record Element not specified".

- Header

If you want your message to have Header information, you can check this check box and specify the required name in the "Header Element Name" text box. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate "Validation Error" window saying, "Header Element not specified".

- Trailer

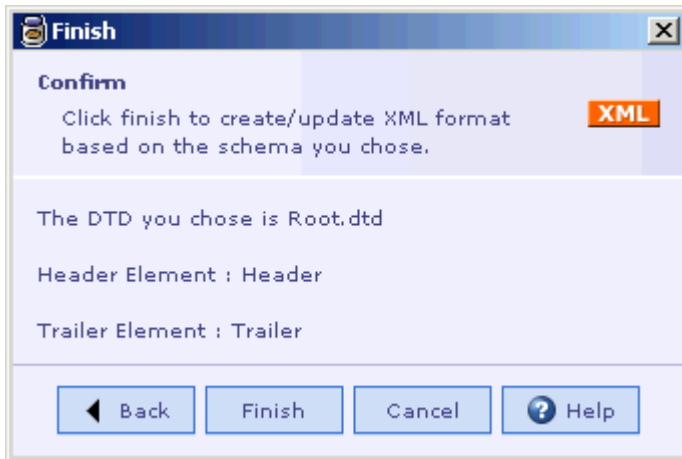
If you want your message to have Trailer information, you can check this check box and specify the required name in the "Trailer Element Name" text box. This input is mandatory for this option and trying to proceed to the next window without specifying the same would generate "Validation Error" window saying, "Trailer Element not specified".

After selecting the required options, if you click the "Next" button in this window, the "Finish" window, which is the final window in the creation wizard, would be opened.

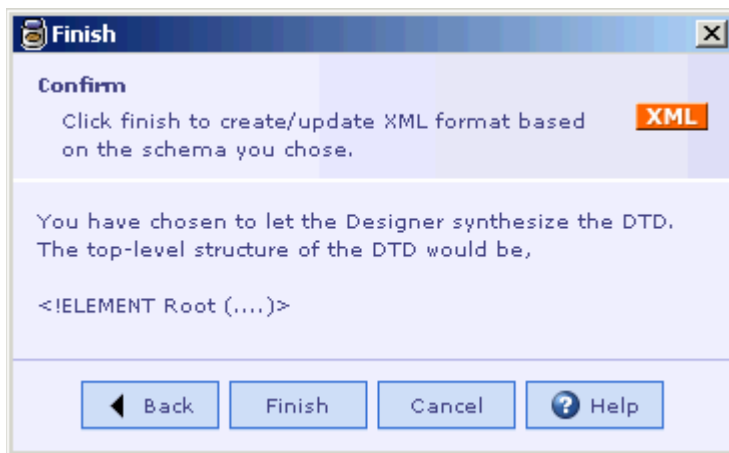
Finish window

This is the final window in the creation wizard and provides information about the chosen XSD / DTD file, Header and Trailer element names if specified and also the top level structure of the DTD if the "Synthesize DTD" option was chosen.

If the DTD/XSD option were chosen in the "Schema Option" window, the 'Finish' window would display information similar to the one shown in the following picture:



If “Synthesize DTD” option were chosen in the “Schema Option” window, the ‘Finish’ window would display information similar to the one shown in the following picture:



See Also:

[From command line using Schema2Car utility](#)
[Designing XML Structure manually in Designer](#)
[Binding XML Schema to Designer](#)
[Reconfigure Schema](#)
[Binding DTD to Designer](#)
[Format Option](#)

From Command Line using Schema2Car Utility

A command-line tool named “schema2car.bat” is available which lets you import the Schema from the command line without going to the Designer. This tool is available in the DESIGNER_HOME\tools directory.

This tool can simply be invoked by calling from the above directory:

schema2car [options] <schemafilename>

Options:

-root rootElementName

If there is more than one global element in your schema, you can use the above option to specify the required element to be taken as the root element.

Example:

```
schema2car -root purchaseOrder po.xsd
```

where "purchaseOrder" is the required root element and "po.xsd" is the schema file. When this command is executed, a cartridge by the name "po.car" (same name as that of the schema file) would be created in the location of the schema file. The cartridge would have an XML External Message node added to it with the same name as that of the XSD. This is equivalent to creating an XML External Message from the Designer UI using creation wizard with the default options.

Note:

- The generated cartridge needs to be compiled manually.
- The schema specified by you is automatically validated by the Designer. If you specify an invalid schema, appropriate error message would be thrown in the command line and the "Schema Option" window of the creation wizard would be opened.
- If your schema has more than one global element and if you omit to specify the "-root" option, then again, the "Schema Option" window of the creation wizard would be opened letting you specify the required root.

See Also:

[From Designer UI using Creation Wizard](#)
[Designing XML Structure manually in Designer](#)
[Binding XML Schema to Designer](#)
[Reconfigure Schema](#)
[Binding DTD to Designer](#)
[Format Option](#)

XML External Message UI

Once an XML external format has been created the following properties can be specified in the External Message UI.

- Format Name. The name of the external format. This is a mandatory property.
- Format Version. This is a mandatory property.
- Standard Name. The message name. This is an optional property.

- Standard Version. This is an optional property.
- Detailed Name. This is an optional property.
- Description. This is an optional property.

External - XML [ejb_jar]

Format Details

External Format: XML

Name: ejb_jar

Version: 1.0

Standard Details

Name:

Version:

Detailed Name:

Description:

Version Info ...

See Also:

[Creating an XML Format](#)
[Binding XML Schema to Designer](#)
[Binding DTD to Designer](#)
[Designing XML Structure manually in Designer](#)
[Format Option](#)

Binding XML Schema to Designer

The XML Plugin supports the W3C XML Schema Recommendation document. Roughly speaking, the XML Plugin maps a XML Schema type to a corresponding Designer type.

This chapter describes the Designer representation of each schema component and also the features of the schema component supported by the Designer.

Note that only the Schema components added under the Root Element of the XSD will be imported in the Designer UI & would have an equivalent representation in the Designer and not any other schema component added globally. Also, the root

element is not shown explicitly in the Designer & is shown in the 'General' tab of the 'Format Options' window.

You will find below a detailed list of the equivalent Designer representation for each Schema Component

See Also:

[Simple Type Definition](#)
[Complex Type Definition](#)
[Attribute Group Definition](#)
[Model Group](#)
[Attribute Declaration](#)
[Element Declaration](#)
[Annotation](#)
[Wildcard](#)
[Identity-constraint Definition](#)
[XML Name Mapping](#)
[W3C XML Schema support](#)
[Terminology](#)
[Glossary](#)

Simple Type Definition

Simple Type definitions provide for constraining character information item of element and attribute information items. There are three varieties of Simple Types viz., atomic types, list types and the union types.

The simple type definitions for atomic and list types can be categorized as:

- schema built-in data types
- user-derived data types

For union type, it is always user-derived.

As for now, there is no specific Designer representation for list and union types and elements of these types are represented similar to that of atomic type.

See Also:

[Atomic Type](#)
[List Type](#)
[Union Type](#)
[Simple Type Designer Support](#)

Atomic Type

The value of an atomic type is indivisible from XML Schema's perspective. For e.g., if the value for an atomic type is, say, "US", then no part of the value, such as the character "S", has any meaning by itself.

A schema component using an atomic simple type definition is typically represented as a field in Designer. This applies to both an element and an attribute using an atomic simple type definition. If the element is repeating, then it is represented as a section with a 'Value' field under it (since Designer does not support repeating fields).

Designer representation in a nutshell:

Simple Type Applied To	Designer Representation
Attribute	Field
Non Repeating Element	Field
Repeating Element	Section

An atomic simple type definition can be a built-in data type or a user-derived one. Conceptually, there is no difference between the Designer representations of the two. In case of schema built-in data types, the specified XML type is shown under the "XML Type" column. In case of user-derived types, XML built-in type specified in the top most level of the hierarchy is shown under the "XML Type" column.

See Also:

[Type Mapping](#)

[Built-in atomic type](#)

[User-derived atomic type](#)

[Facets](#)

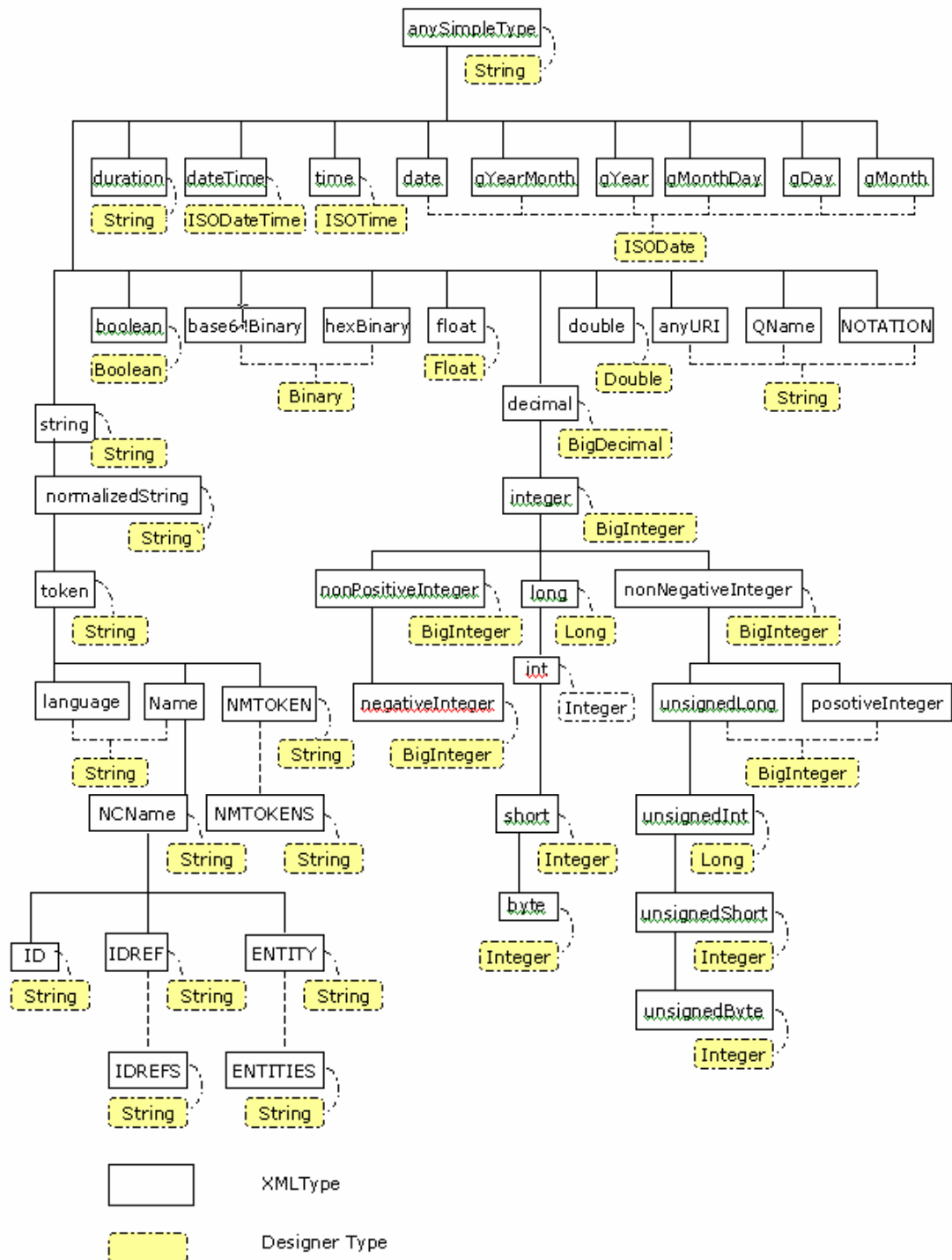
[List Type](#)

[Union Type](#)

[Simple Type Designer Support](#)

Type Mapping

The XML built-in hierarchy along with the corresponding Designer type is depicted in the following picture:



The above picture is represented in the form of tables as seen under:

Built-in Primitive data types

XML Schema Data type	Default Designer Data type	Other supported Designer Data type/s
string	String	Character
boolean	Boolean	-
decimal	BigDecimal	Double Float
float	Float	-
double	Double	-
duration	String	-
dateTime	ISODatetime	DateTime
time	ISOTime	TimeOnly
date	ISODate	DateOnly
gYearMonth	ISODate	String
gYear	ISODate	String
gMonthDay	ISODate	String
gDay	ISODate	String
gMonth	ISODate	String
hexBinary	Binary	-
base64Binary	Binary	-
anyURI	String	-
QName	String	-
NOTATION	String	-

The XML Schema Data type “anySimpleType”, which can be considered as the base type of all primitive types has its equivalent Designer data type as under:

XML Schema Data type	Designer Data type	Other supported Designer Data type/s
anySimpleType	String	-

Built-in Derived data types

XML Schema Data type	Designer Data type	Other supported Designer Data type/s
normalizedString	String	-
token	String	-
language	String	-
NMTOKEN	String	-
NMTOKENS	String	-
Name	String	-
NCName	String	-
ID	String	-
IDREF	String	-
IDREFS	String	-
ENTITY	String	-
ENTITIES	String	-
integer	BigInteger	Integer Long
nonPositiveInteger	BigInteger	Integer Long
negativeInteger	BigInteger	Integer

XML Schema Data type	Designer Data type	Other supported Designer Data type/s
		Long
long	Long	-
int	Integer	-
short	Integer	-
byte	Integer	-
nonNegativeInteger	BigInteger	Integer Long
unsignedLong	BigInteger	Integer Long
unsignedInt	Long	Integer BigInteger
unsignedShort	Integer	-
unsignedByte	Integer	-
positiveInteger	BigInteger	Integer Long

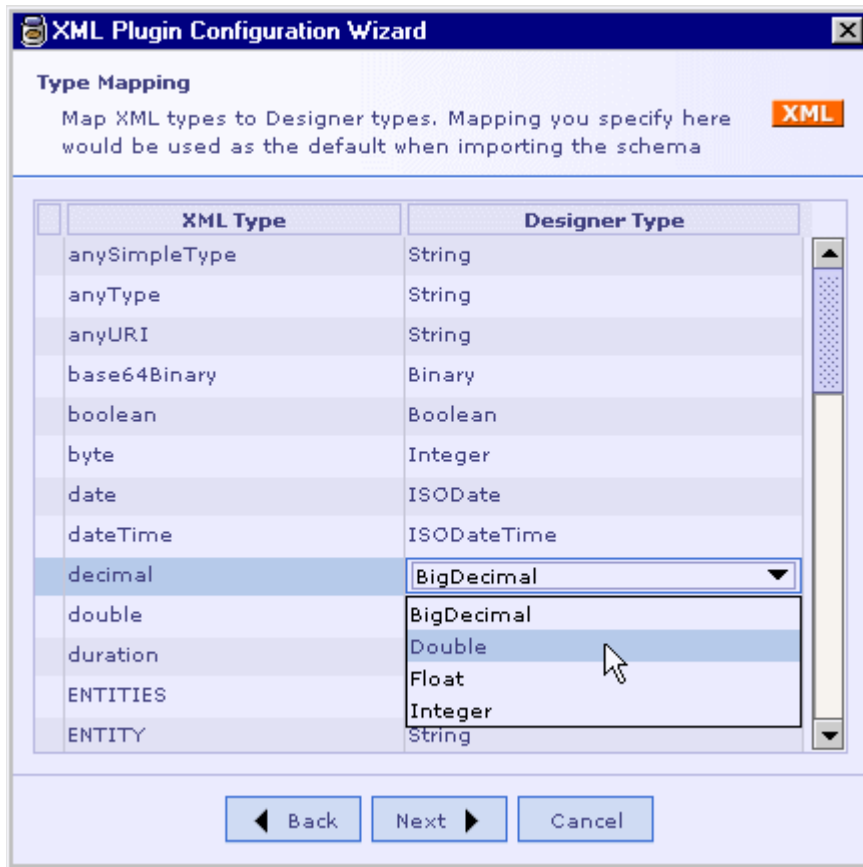
User-derived data types

In case of an element of user-derived type, the XML built-in type specified in the top most level of the hierarchy is taken as the XML type of the element and is mapped to the corresponding Designer type as seen in the above tables. Refer ["User-derived atomic type"](#).

Support for customizing type mapping

Using this feature you can change the default Designer type mapping for an XML type. All elements/attributes of that type (or its derived types) would be represented with a field of the corresponding Designer type.

This feature is particularly useful for mapping the 'decimal' XML type to Designer's Double type and the 'integer' XML type to Designer's Integer type, if you are sure that it always falls within the range. It is also possible to do this customization after importing, but it is very tedious.



See Also:

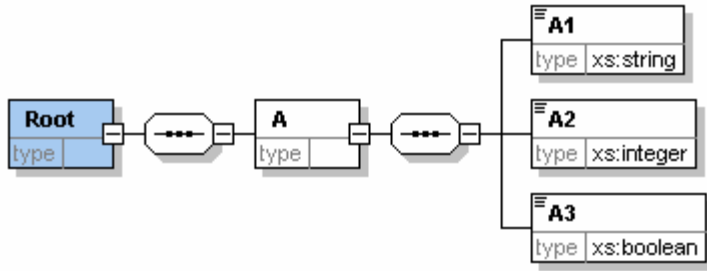
[Atomic Type](#)

Built-in atomic type

The primitive and derived data types listed in the above tables are the built-in data types.

Element

Consider the following schema where A1, A2 & A3 are elements with simple type definition and of built-in data types:



When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	A	Section	Element	
	A1	String	string	
	A2	BigInteger	integer	
	A3	Boolean	boolean	

Field Properties

XML Name
 Required ☒

Namespace
 Default Value

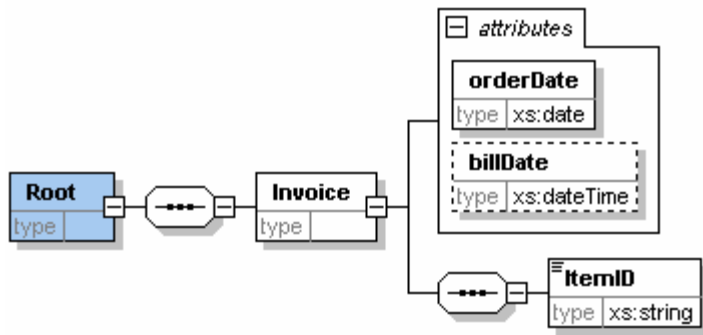
Node Type element

Designer Representation details

1. Elements 'A1', 'A2' and 'A3' are represented as fields (as they are non-repeating).
2. The built-in data type specified for each element in the schema is represented as it is under 'XML Type' column against each element.
3. The corresponding Designer type is represented under 'Type' column (Refer the [data type mapping tables](#) above).
4. The 'Node Type' of the elements 'A1', 'A2' and 'A3' is shown as "element" in the 'Field Properties' pane.
5. The XML Name of the element and its namespace are displayed in the corresponding text boxes in the "Field Properties" pane.

Attribute

Consider the following schema where the complex element 'Invoice' has two attributes 'orderDate' and 'billDate' of built-in data type "date" and "dateTime" respectively added to it.



When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Invoice	Section	Element	
	billDate	ISODateTime	dateTime	
	orderDate	ISODate	date	
	ItemID	String	string	

Field Properties	
XML Name	billDate
Required	<input type="checkbox"/>
Namespace	
Default Value	
Node Type	attribute
Facets	

Designer Representation details

- Attributes 'billDate' and 'orderDate' are represented as fields with Node Type as "attribute".
- The built-in data type specified for the attributes in the schema is represented as it is under 'XML Type' column against the corresponding attribute field.
- The corresponding Designer type is represented under 'Type' column (Refer the data type mapping tables above).
- The occurrence constraint is represented by the "Required" checkbox in the "Field Properties" pane. For the attribute, 'billDate', the checkbox would be unchecked & for the attribute 'orderDate', the checkbox would be checked.
- The XML Name of the attribute and its namespace (if applicable) are displayed in the corresponding text boxes in the "Field Properties" pane.

See Also:

[Atomic Type](#)

User-derived atomic type

User-derived data types are those derived data types that are defined by individual schema designers.

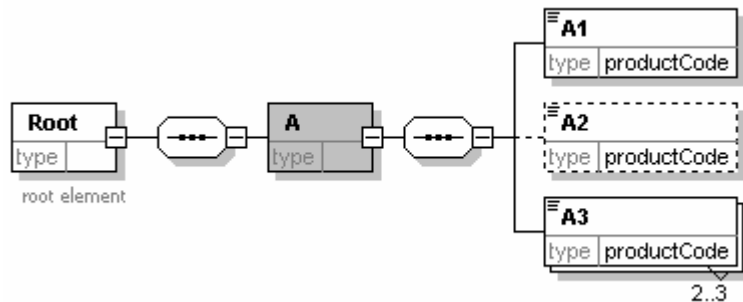
Element

Consider the following Simple Type definition schema component:

```
<xs:simpleType name="productCode">  
  <xs:restriction base="xs:string">  
    <xs:length value="8" fixed="true"/>  
  </xs:restriction>  
</xs:simpleType>
```

Here, “productCode” is a named Simple Type schema component, which is derived by restricting the base type “xs:string” by setting the ‘length’ facet.

Consider the following schema:



Here, we have 3 elements added under the Root with type as “productCode”, but with different occurrence constraints; A1 → minOcc = 1, maxOcc = 1; A2 → minOcc = 0, maxOcc = 1 and A3 → minOcc = 2, maxOcc = 3. When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	A	Section	Element	
	A1	String	string	
	A2	String	string	
	A3	Section	Element	
	Value	String	string	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Designer Representation details

1. Elements 'A1' and 'A2' are represented as fields as they are non-repeating (with Node Type as "element").
2. Element 'A3' is represented as a section as it is repeating, with a 'Value' field under it (whose Node Type is "Value").
3. The XML Type shown for these elements is the built-in data type "string" specified for the Simple Type definition 'productCode' which is the user-derived data type set for these elements. (Note that user-derived data type 'productCode', has no representation in the Designer)
4. The corresponding Designer type is represented under 'Type' column. (Refer the data type mapping tables above).
5. The occurrence constraints of the non-repeating elements, 'A1' and 'A2' in this case are represented by the "Required" checkbox in the "Field Properties" pane. For the element 'A1', the checkbox would be checked & for the element 'A2', the checkbox would be unchecked.
6. The occurrence constraints of the repeating element, 'A3' in this case is represented by the "Min Occurs" and "Max Occurs" combo boxes in the "Section Properties" pane as seen in the above picture.
7. The 'Node Type' for the elements 'A1' and 'A2' is shown as "element" in the 'Field Properties' pane. The Node Type for the 'Value' field of A3 is shown as "value" in the 'Field Properties' pane.

Attribute

Consider the following Simple Type definition (user-derived data type) schema component:

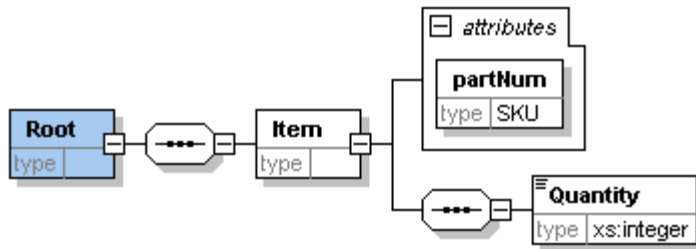
```
<xs:simpleType name="SKU">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
```

```

    </xs:restriction>
</xs:simpleType>

```

Consider the following schema where the complex element 'Item' has an attribute 'partNum' of user-derived data type 'SKU' added to it.






Above schema in text form:

```

<xs:element name="Root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Item">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Quantity" type="xs:integer"/>
          </xs:sequence>
          <xs:attribute name="partNum" type="SKU" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Item	Section	Element	
	partNum	String	string	
	Quantity	BigInteger	integer	

Field Properties

XML Name
Required ☒

Namespace
Default Value

Node Type attribute ▼

Facets

Designer Representation details

1. The attribute 'partNum' is represented as a field with 'Node Type' as "attribute".
2. The XML Type shown for the attribute is the built-in data type "string" specified for the Simple Type definition 'SKU' which is the user-derived data type set for the attribute. (Note that user-derived data type 'SKU' has no representation in the Designer).
3. The corresponding Designer type is represented under 'Type' column (Refer the data type mapping tables above).
4. The occurrence constraints are represented by the "Required" checkbox in the "Field Properties" pane. In this case, the checkbox would be checked as the attribute use is set as "required".

Note:

1. Simple Type definition schema component, by itself does not have a representation in the Designer. It can have a representation in the Designer if and only if an element of this type is added somewhere under the Root element.
2. The Designer representation of an element of simple type varies based on the occurrence constraints of that element.
3. User derived data type name has no equivalent Designer representation, and only the contents of that Simple Type definition schema component viz., the built-in data type used and the facets set have an equivalent representation in the Designer. Thus the data types 'productCode' and 'SKU' defined in the above examples have no equivalent Designer representation.

See Also:

[Atomic Type](#)

Facets

Facets set for an element or an attribute are represented in the Designer in a separate window. Let the Designer representation of a schema be as under:

Field Name	Type	XML Type	Description
A	Section	Element	
A1	String	string	
A2	String	string	
A3	Section	Element	
Abc Value	String	string	

Field Properties

XML Name

Required ☒

Namespace

Default Value

Node Type

Facets

In the above structure, when the field 'A1', 'A2' or "A3.Value" is selected, "Facets" button appear in the "Field Properties" pane, which when clicked, opens the "Facets" window depicting the facets if any set for that element.

Facets

Facets

Pattern

Enumeration

Facet	Value	Description
length	8	
minLength		
maxLength		
whiteSpace		

OK

Cancel

For ease of view, the representation of Facets is split across three tabs in the 'Facets' window as seen in the above picture.

Facets tab

Under the Facets tab, all applicable facets (except “pattern” and “enumeration”) for the XML type of a selected element or attribute are listed under the ‘Facet’ column and the value, if any set for the facet is shown under the respective ‘Value’ column. For e.g., in the above picture, under the ‘Facet’ column, the facets applicable (other than “pattern” and “enumeration”) of the XML type “string” are listed since the selected field, viz., A3.Value has XML type as “string”. The value “8” set for the length facet for the element A3 is shown under the ‘Value’ column.

A subset of the following facets would be listed in the ‘Facet’ column based on the XML type of the selected element / attribute.

Sl.No.	Facet list
1	length
2	minLength
3	maxLength
4	whiteSpace
5	maxExclusive
6	maxInclusive
7	minExclusive
8	minInclusive
9	totalDigits
10	fractionDigits

Pattern tab

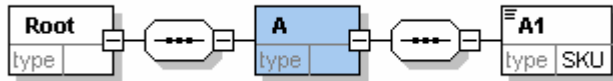
Pattern facet, if set for the XML type (if applicable) of an element or attribute is shown under this tab.

For e.g., consider the following Simple Type definition where ‘pattern’ facet is used:

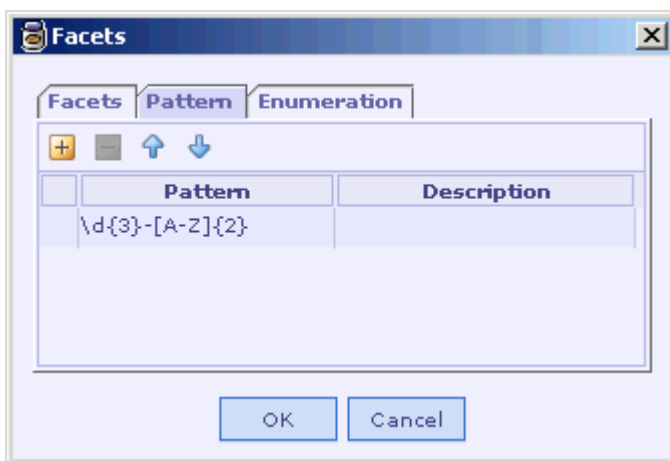
```
<xs:simpleType name="SKU">  
  <xs:restriction base="xs:string">  
    <xs:pattern value="\d{3}-[A-Z]{2}"/>  
  </xs:restriction>  
</xs:simpleType>
```

</xs:simpleType>

Let us suppose that an element A1 is added somewhere under the root element with type as "SKU" as shown under.



When the above schema is imported in the Designer, the pattern facet set for the Simple Type "SKU" would be represented in the "Facets" window under the "Pattern" tab as shown below:



Enumeration tab

Enumeration facet, if set for the XML type (if applicable) of an element or attribute is shown under this tab.

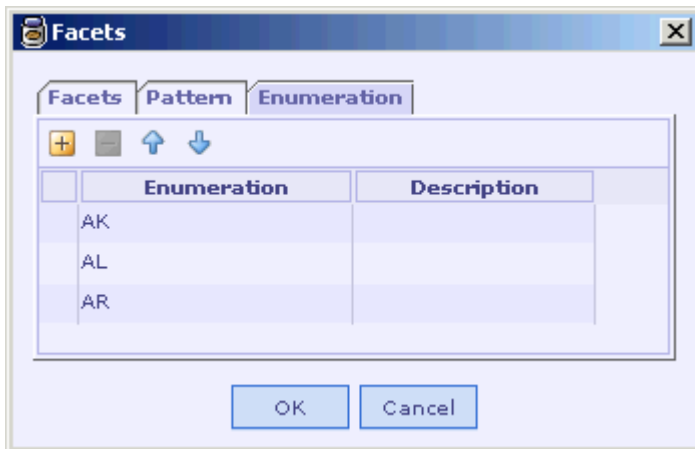
For e.g., consider the following Simple Type definition where 'enumeration' facet is used:

```
<xs:simpleType name="USState">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AK"/>
    <xs:enumeration value="AL"/>
    <xs:enumeration value="AR"/>
  </xs:restriction>
</xs:simpleType>
```

Let us suppose that an element A1 is added somewhere under the root element with type as "USState" as shown under.



When the above schema is imported in the Designer, the enumeration facet set for the Simple Type “USState” is represented in the “Facets” window under the “Enumeration” tab as shown below:



Note:

Facet representation for an attribute in the Designer is identical to that of an element.

See Also:

[List Type](#)

[Union Type](#)

[Simple Type Designer Support](#)

List Type

List types are comprised of sequences of atomic types and consequently the parts of a sequence (“the atoms”) themselves are meaningful, unlike an atomic type.

Representation of list type in Designer is somewhat similar to atomic type, except that the XML built-in type specified in the schema is ignored & in that place “anySimpleType” is used. Hence, facet constraints, if any, set and the data type constraint are not honored.

Built-in list type

XML Schema has three built-in list types, viz., NMTOKENS, IDREFS and ENTITIES. For e.g., an element of type NMTOKENS would be a white-space delimited list of NMTOKEN’s, such as “US UK FR”.

User-derived list type

You can also create new list types by derivation from existing atomic types.

Element without facet

Let us consider the case, where the list type is derived from an atomic type.

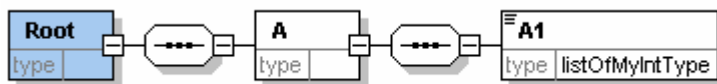
Consider the following atomic type definition.

```
<xs:simpleType name="myInteger">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="10000"/>
    <xs:maxInclusive value="99999"/>
  </xs:restriction>
</xs:simpleType>
```

Let "listOfMyIntType" be a list type derived from the above atomic type:

```
<xs:simpleType name="listOfMyIntType">
  <xs:list itemType="myInteger"/>
</xs:simpleType>
```

Consider the following schema where an element 'A1' is added under the Root element with type as "listOfMyIntType":



When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	A	Section	Element	
	A1	String	anySimpleType	

Designer Representation details

1. Element 'A1' is represented as a field as it is non-repeating (with Node Type as "element").
2. The XML Type of 'A1' is shown as "anySimpleType" irrespective of the built-in data type ("integer" in this case) specified for the Simple Type definition 'myInteger' which is the user-derived data type based on which the list data type is derived.

3. The corresponding Designer type ("string") is represented under 'Type' column.
4. The facets of the Simple Type definition 'myInteger' are not represented unlike the case of an atomic data type. (as the content of the element is taken as a whole and parsed and not as a sequence of atomic types)

Element with facet

Several facets such as length, minLength, maxLength, pattern and enumeration can be applied to list type.

Let us consider the case where the 'length' facet is applied.

Consider the following schema components:

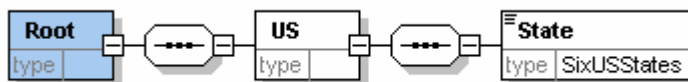
```
<xs:simpleType name="USState">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AK"/>
    <xs:enumeration value="AL"/>
    <xs:enumeration value="AR"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="USStateList">
  <xs:list itemType="USState"/>
</xs:simpleType>



<xs:simpleType name="SixUSStates">
  <xs:restriction base="USStateList">
    <xs:length value="6"/>
  </xs:restriction>
</xs:simpleType>
```

In the above, "USStateList" is a list type derived from the atomic simple type "USState". "SixUSStates" is a simple type derived by restricting "USStateList" to only six items, i.e., elements whose type is SixUSStates must have six items.

Consider the following schema:



Here, an element State is added under the Root element with type as "SixUSStates". When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	US	Section	Element	
	State	String	anySimpleType	

Field Properties


XML Name

Required
☒

Namespace

Default Value

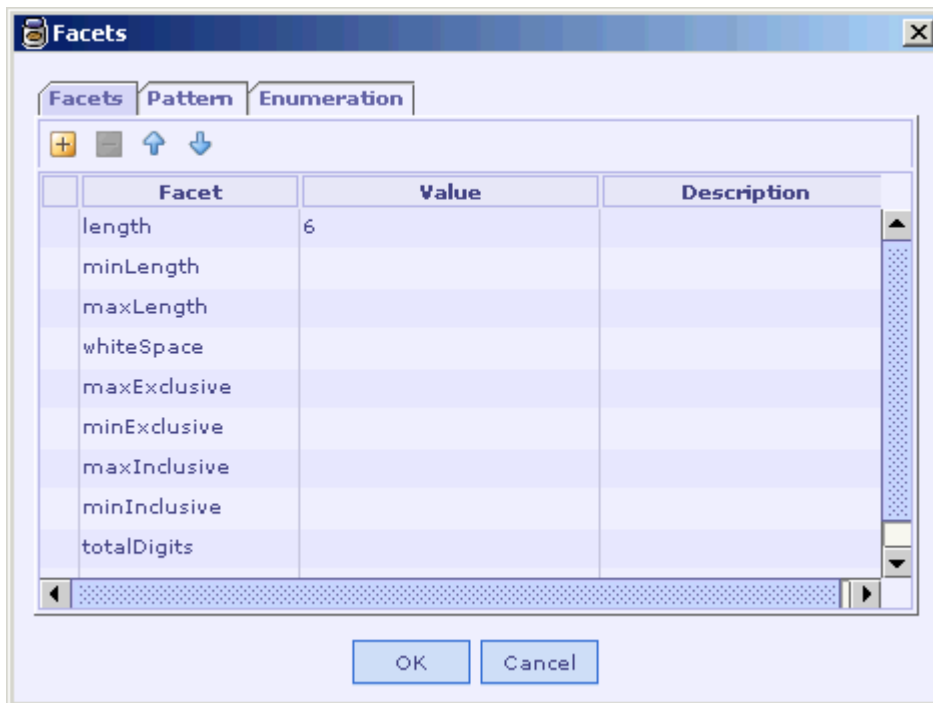
Node Type


element

Facets

Designer Representation details

1. Element 'State' is represented exactly similar to that of the previous case. Refer (i) to (iii) of [Designer Representation details](#) under "Element of user-derived list type".
2. The 'length' facet set for the Simple Type definition "SixUSStates" is represented in the Facets window as shown under:



Facet	Value	Description
length	6	
minLength		
maxLength		
whiteSpace		
maxExclusive		
minExclusive		
maxInclusive		
minInclusive		
totalDigits		

OK Cancel

See Also:

[Atomic Type](#)

[Union Type](#)

[Simple Type Designer Support](#)

Union Type

Atomic types and list types enable an element or an attribute value to be one or more instances of one atomic type. In contrast, a union type enables an element or attribute value to be one or more instances of one type drawn from the union of multiple atomic and list types.

Representation of union type in Designer is exactly identical to list type. Here also, the XML built-in type specified in the schema is ignored & in that place “anySimpleType” is used. Hence, facet constraints if any set and the data type constraint are not honored. Refer [List Type](#).

Consider the following schema components:

```
<xs:simpleType name="USState">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AK"/>
    <xs:enumeration value="AL"/>
    <xs:enumeration value="AR"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="myInteger">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="10000"/>
    <xs:maxInclusive value="99999"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="listOfMyIntType">
  <xs:list itemType="myInteger"/>
</xs:simpleType>

<xs:simpleType name="zipUnion">
  <xs:union memberTypes="USState listOfMyIntType"/>
</xs:simpleType>
```

In the above, “zipUnion” is a union type built from one atomic type namely “USState” and one list type namely “listOfMyIntType”.

Consider the following schema:



Here, an element “zips” is added under the Root element with type as “zipUnion”. When this schema is imported in the Designer, the Designer representation would be as under:

Field Name	Type	XML Type	Description
US	Section	Element	
zips	String	anySimpleType	

Field Properties

XML Name

Required
☒

Namespace

Default Value

Node Type

element ▼

Facets

Designer Representation details

Element ‘zips’ is represented exactly similar to that of the case in List type. Refer (i) to (iv) of Designer Representation details under “Element of user-derived list type”.

See Also:

[Atomic Type](#)

[List Type](#)

[Simple Type Designer Support](#)

Simple Type Designer Support

Property	Supported
{ name }	Yes
{ target namespace }	Yes
{ base type definition }	Yes
{ facets }	<p>“totalDigits” facet not supported for any applicable data type.</p> <p>Following facets are not supported for “duration” data type:</p> <p>maxInclusive, maxExclusive, minInclusive & maxInclusive.</p>
{ final }	Text

{ annotation }	Yes
{ variety }	atomic - Yes list - Yes (Facet constraint & Type constraint not supported) union - Yes (Facet constraint & Type constraint not supported)

See Also:

[Atomic Type](#)

[List Type](#)

[Union Type](#)

Complex Type Definition

A schema component using a complex type definition is typically represented as a section in Designer and the contents are shown as sub-elements. However, if the Complex Type has just text content and no attributes or sub-elements in it and is applied to a non-repeating element, it is represented as a field. If the complex type is represented as a section, its text content (if present) is represented as the 'Value' field under the section.

Designer representation in a nutshell:

Content of Complex Type	Applied to	Representation
Has Sub-elements with or without attributes	Repeating/Non-Repeating Element	Section
Has text and attributes & no sub-element	Repeating/Non-Repeating Element	Section
Has just text and no attribute or sub-element	Non-repeating element	Field
Has just text and no attribute or sub-element	Repeating element	Section with 'Value' field representing the text content

The following example would illustrate the above representation.

Consider the following three Complex Type definition schema components:

i) Complex Type schema component with sub-elements:

```

<xs:complexType name="HasElm">
  <xs:sequence>
    <xs:element name="Child1" type="xs:string"/>
    <xs:element name="Child2" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

```

ii) Complex Type schema component with text and attribute & no sub-element

```

<xs:complexType name="TextAndAttribute">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="Att1" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

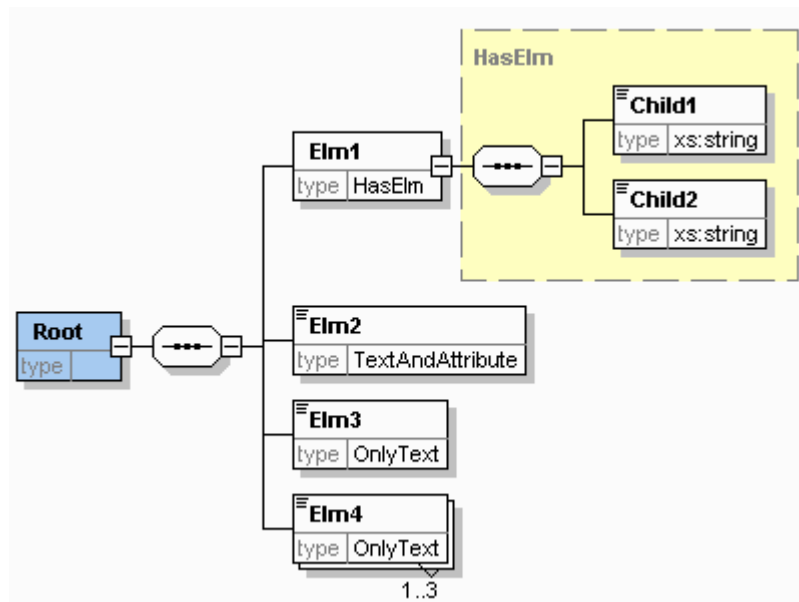
iii) Complex Type schema component with just text and no attribute or sub-element

```










<xs:complexType name="OnlyText">
  <xs:simpleContent>
    <xs:extension base="xs:string"/>
  </xs:simpleContent>
</xs:complexType>

```

Consider the following schema:



Here, we have used the three complex type definitions specified above for the elements added under the Root. Note that both 'Elm3' and 'Elm4' have the same type but 'Elm3' is non-repeating while 'Elm4' is repeating. When this schema is imported in the Designer, the Designer representation would be as under:

Field Name	Type	XML Type	Description
 Elm1	Section	Element	
 Child1	String	string	
 Child2	String	string	
 Elm2	Section	Element	
 Att1	String	string	
 Value	BigDecimal	decimal	
 Elm3	String	string	
 Elm4	Section	Element	
 Value	String	string	

Note that while 'Elm3' is represented as a field, 'Elm4' is represented as a section since it is repeating.

See Also:

[Aggregation](#)

[Simple Content](#)

[Complex Content](#)

[Type Substitution](#)

[Complex Type Designer Support](#)

Aggregation

Designer representation of an element of complex type definition is built based on the concept of aggregation, i.e. the contents of all the components referenced by the type are put to the type and eventually the type itself is aggregated to the element. A complex type schema component aggregates to Designer representation of all the schema components it references, i.e., for deriving Designer representation for an element of complex type definition, all the schema components that the complex type definition references are looked at and are aggregated to build the super-structure of the complex type. If the content of a complex type itself is an aggregate, we take those contents and aggregate it to the complex type. For e.g., if a complex type definition references to an attribute group, the contents of the attribute group are aggregated into the Designer representation of the complex type.

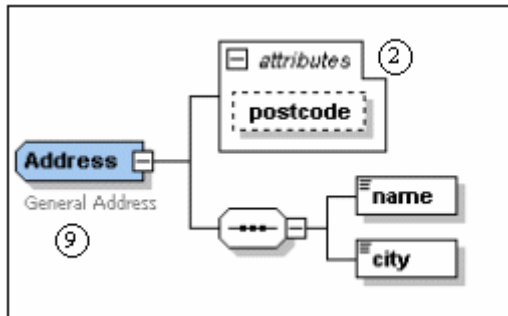
Contents that are aggregated:

Content of Complex Type	Aggregation procedure
Group	The contents of the group are aggregated. In some special cases the contents of the group cannot be aggregated and the group / model group itself is represented as section.
Model Group	
Attribute Group	The contents of the attribute group are aggregated. (Attribute Group never has a first-class representation)
Base type (if complex type is a derived one)	The contents of the base type are aggregated (including all ancestors' types).
Simple Element	<p>Always aggregated. (Have a first-class representation)</p> <p>The entire contents are aggregated as a whole without looking into them.</p> <p>Represented as a Field or a Section depending on its contents and occurrence.</p>
Complex Element	
Any element	
Attributes	
Any Attribute	
Annotation	
	Annotations for the complex type and its base types are accumulated.

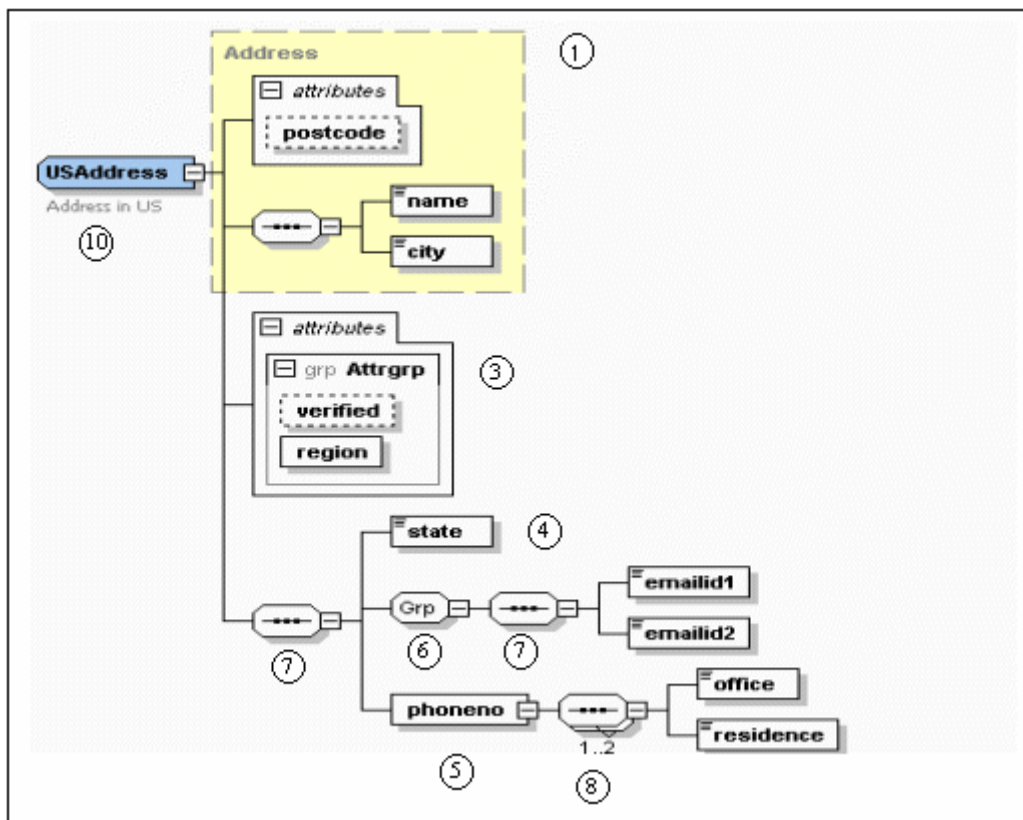
This aggregation is done only where it is possible. For e.g., if the content of a complex type is a model group, only where flattening of model group (Refer [“Model Group Flattening Rules”](#)) is applicable, the contents of the model group are pushed to the complex type, otherwise, the model group is retained as such and is represented as a section in the Designer.

Consider the following Complex Type schema component “USAddress”, which is derived from another Complex Type schema component viz. “Address”.

Address (Base Type)

















USAddress schema component



(The nos. in circles represent different contents allowed in a Complex Type and are referred in the "No." column in the following table ["Designer Representation Details"](#))

When an element, say "USAddress" of type "USAddress" is added under the Root element in a schema and when that schema is imported in the Designer, the equivalent Designer representation would be as under:

Field Name	Type	XML Type	Description
 USAddress	Section	Element	General Address Address in US
 name	String	string	
 city	String	string	
 region	String	string	
 verified	Boolean	boolean	
 postcode	String	string	
 state	String	string	
 zip	BigInteger	positiveInt...	
 emailid1	String	string	
 emailid2	String	string	
 phoneno	Section	Element	
 phonenoGroup 1	Section	Element	
 office	BigInteger	integer	
 residence	BigInteger	integer	

Designer Representation details

No	Content Type (Name)	Aggregated	Representation Details
1	Base Type (Address)	Yes	The base type 'Address' is aggregated & its contents 'name', 'city' & 'postcode' are pushed to the derived type 'USAddress'; They are represented as sub-fields of 'USAddress' element.
2	Attributes (postcode)	Yes	Attribute 'postcode' of 'Address' base type is aggregated to the base type & in turn, aggregated to the derived type as specified above.
3	Attribute Group (Attrgrp)	Yes	'Attrgrp' by itself has no representation, however, its contents viz., 'verified' & 'region' are aggregated to the complex type. They are represented as sub-fields of 'USAddress' element.
4	Simple Element (state)	Yes	Simple Element "state" is aggregated to the 'USAddress' type. It is represented as a sub-field of 'USAddress'

No	Content Type (Name)	Aggregated	Representation Details
			element.
5	Complex Element (phoneno)	Yes	Complex Element “phoneno” with its entire contents is aggregated to the ‘USAddress’ type. It is represented as a sub-section of ‘USAddress’ element.
6	Group (Grp)	Yes	The contents ‘emailid1’ & ‘emailid2’ of the group ‘Grp’ are aggregated to the ‘USAddress’ type (As “Flattening” is possible in this case). The group has no representation while its contents are represented as sub-fields of ‘USAddress’ element.
7	Non-repeating Model Group	Yes	Both model groups (represented by circle no.7) are flattened & their contents are aggregated to the ‘USAddress’ type. (As flattening possible here) The model groups have no representation while their contents are represented as sub-fields/sub-sections as the case may be.
8	Repeating Model Group	No	The model group is not flattened as flattening is not possible in this case. (Refer “Model Group represented as section”). The model group is represented as the section ‘phonenoGroup1’.
9 10	Annotation of Base & Derived type	Yes	Annotation of ‘Address’ type & ‘USAddress’ type are accumulated to the ‘USAddress’ type; Represented as Description for the ‘USAddress’ element.

See Also:

[Simple Content](#)
[Complex Content](#)
[Type Substitution](#)

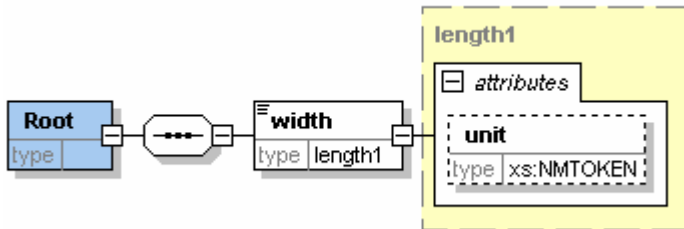
Simple Content (Complex Type Definition)

If a complex type is defined to be of simple content, then it can have only character data content (i.e. it can reference to a built-in data type) and/or attributes and can have no sub-elements in it. The simple content definition can use extension or restriction.

Consider the following Complex Type definition with Simple Content using extension:

```
<xs:complexType name="length1">
  <xs:simpleContent>
    <xs:extension base="xs:nonNegativeInteger">
      <xs:attribute name="unit" type="xs:NMTOKEN"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Consider the following schema:



Here, an element 'width' is added under the Root with type as "length1". When this schema is imported in the Designer, the Designer representation would be as under:

Field Name	Type	XML Type	Description
width	Section	Element	
unit	String	NMTOKEN	
Value	BigInteger	nonNegativeInteger	

Designer Representation details

1. The element 'width' is represented as a section (because it has attribute content) with the 'Value' field with 'Node Type' as 'value' representing its value information item. The base simple type is shown as the 'XML Type' of the 'Value' field.
2. The attribute 'unit' is represented as another sub-field of the 'width' section with 'Node Type' as "attribute".

See Also:

[Aggregation](#)

Complex Content (Complex Type Definition)

If a complex type is defined to be of complex content, then it can have sub-elements, attributes and/or character content in it. The complex content definition can use extension or restriction.

Mixed Content

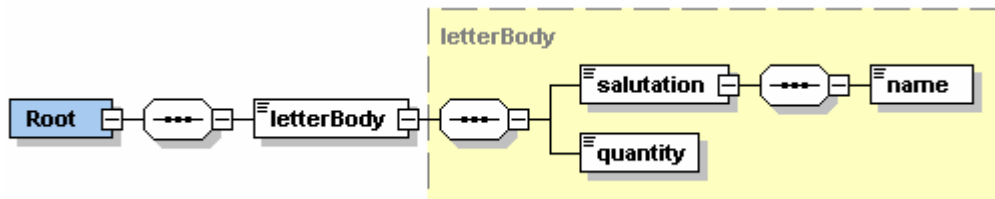
If a Complex Type definition is defined to be of mixed content (value for the attribute “mixed” is specified as “true”), then character data can appear alongside sub elements.

Consider the following Complex Type definition schema component:







```
<xs:complexType name="letterBody" mixed="true">
  <xs:sequence>
    <xs:element name="salutation">
      <xs:complexType mixed="true">
        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="quantity" type="xs:positiveInteger"/>
  </xs:sequence>
</xs:complexType>
```

As per the above definition, text can appear anywhere between an element of type “letterBody” and its child elements. Also, text can appear anywhere between the element “salutation” and its child element.

Consider the following schema:



Here, an element 'letterBody' is added under the Root with type as "letterBody". When this schema is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 letterBody	Section	Element	
 Value	String	string	
 salutation	Section	Element	
 Value	String	string	
 name	String	string	
 quantity	BigInte...	positiveInteger	

Designer Representation details

1. The applicability of text content (mixed = "true") for the elements "letterBody" and "salutation" is represented by the 'Value' field following those elements.
2. All the text content that can appear between the elements are merged and shown as a whole by the 'Value' field & hence, the position of the text content in the XML instance would not be retained as such.

Derivation by Restriction

A complex type derived by restriction is very similar to its base type, except that its declarations are more limited than the corresponding declarations in the base type. In fact, the values represented by the new type are a subset of the values represented by the base type (as is the case with restriction of simple types). In other words, an application prepared for the values of the base type would not be surprised by the values of the restricted type.

Consider the following Complex Type definition schema components:

```
<xs:complexType name="length1">
  <xs:sequence>
    <xs:element name="size" type="xs:nonNegativeInteger"/>
    <xs:element name="unit" type="xs:NMTOKEN" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="object" type="xs:string"/>
</xs:complexType>

<xs:complexType name="length2">
  <xs:complexContent>
    <xs:restriction base="length1">
      <xs:sequence>
        <xs:element name="size" type="xs:nonNegativeInteger"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

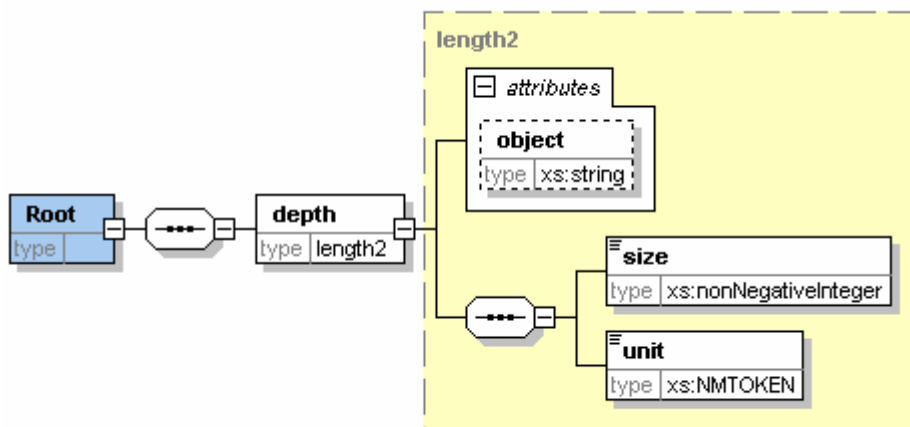
```

        <xs:element name="unit" type="xs:NMTOKEN" minOccurs="1"/>
    </xs:sequence>
</xs:restriction>
</xs:complexContent>
</xs:complexType>

```

Here 'length2' is a Complex Type definition schema component derived by restriction from the complex type 'length1'. In the definition of 'length2', we have provided a new (more restrictive) value for the minimum number of 'unit' element occurrences by setting minOccurs as 1. This change narrows the allowable number of 'unit' elements from a minimum of 0 (in the base type) to a minimum of 1 (in the derived type). Note that all elements of type 'length2' will also be acceptable as elements of type 'length1'.

Consider the following schema:



Here, an element 'depth' is added under the Root with type as "length2". When this schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	depth	Section	Element	
	att1	String	string	
	size	BigInteger	nonNegativeInteger	
	unit	String	NMTOKEN	

Field Properties

XML Name

unit

Required

☒

Namespace

http://bharath.com

Default Value

Node Type

element

Facets

The contents of the base type along with the restrictions set in the derived type are shown as the contents of an element of the derived type.

Designer Representation details

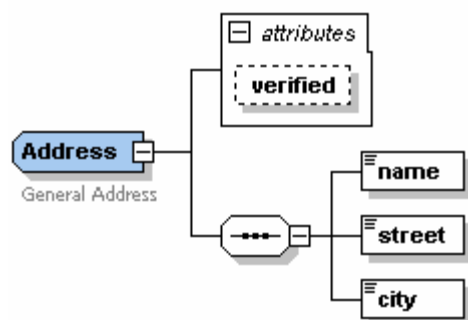
1. The element 'depth' is represented as a section with 'Node Type' as "element".
2. The element contents of the complex type definition viz, 'length2' (derived from 'length1') are represented as sub-fields of the section "depth" with 'Node Type' as "element".
3. The attribute of the base complex type definition 'length1' is aggregated to the derived type, (though not specifically mentioned in the derived type as schema definition does not require the same) and is represented as a sub-field of "depth" with 'Node Type' as "attribute".
4. The occurrence constraint "minOccurs = 1" set for the element "unit" is represented by the checking of the "Required" checkbox in the "Field Properties" pane. (also by the absence of the optional icon '?' near the field icon)

Derivation by Extension

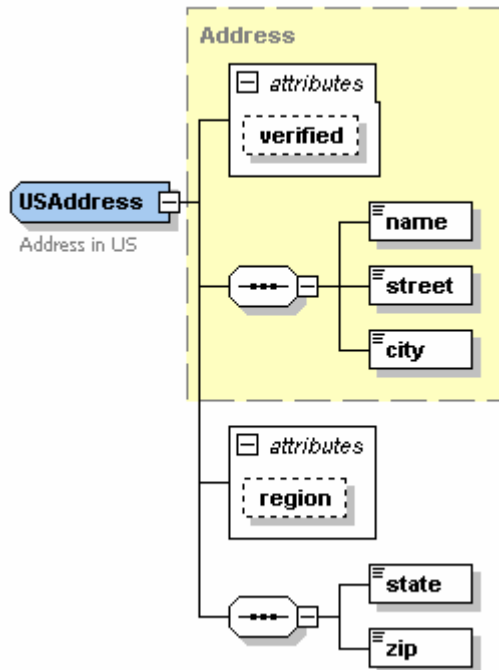
When a complex type is derived by extension, its effective content model is the content model of the base type plus the content model specified in the type derivation. Furthermore, the two content models are treated as two children of a sequential group.

Consider the following Complex Type definition schema components:

Address type (Base type)



USAddress type (Derived from 'Address' type by Extension)



The content model of 'USAddress' is the content model of 'Address' plus the declarations for 'state' and 'zip' elements, 'region' attribute and annotation.

When an element, say "USAddress" of type "USAddress" is added under the Root element in a schema and when that schema is imported in the Designer, the equivalent Designer representation would be as under:

Field Name	Type	XML Type	Description
USAddress	Section	Element	General Address Address in US
verified	Boolean	boolean	
name	String	string	
street	String	string	
city	String	string	
region	String	string	
state	String	string	
zip	BigInte...	positiveIn...	

The base type 'Address' is aggregated and its contents are aggregated to the derived type. The annotation of the base type and derived type are accumulated and shown as description for the element of derived type.

Designer Representation details

1. The model group (sequence) in the base type is flattened (Refer ["Model Group Flattening Rules"](#)) and the child elements viz. 'name', 'street' and 'city' are shown as the child elements of "USAddress" element with Node Type as "element".

2. The attribute 'verified' of the base type is aggregated to the derived type and shown as the sub-field of the "USAddress" element with 'Node Type' as "attribute".
3. The model group (sequence) added to the derived type is again flattened (as flattening is possible in this case) and the sub-elements viz. 'state' and 'zip' are shown as the child elements of "USAddress" element with 'Node Type' as "element". The attribute 'region' added for the derived type is again shown as a child element of "USAddress" element with 'Node Type' as "attribute".
4. Annotation of 'Address' type & 'USAddress' type are accumulated to the 'USAddress' type and represented as Description for the 'USAddress' element.

See Also:

[Aggregation](#)

[Simple Content](#)

[Type Substitution](#)

[Complex Type Designer Support](#)

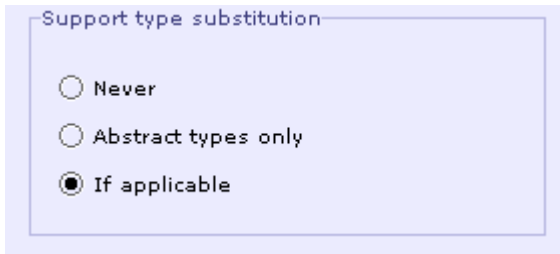
Type Substitution

Suppose there is a complex type definition by name "Address" and another complex type, say, "USAddress" is derived by extension from the "Address" type, XML Schema allows us to use instances of "USAddress" in place of instances of "Address". In other words, an instance document whose content conforms to the "USAddress" type will be valid if that content appears within the document at a location where an "Address" is expected (assuming the "USAddress" content itself is valid). To make this feature of XML Schema work, and to identify exactly which derived type is intended, the derived type must be identified in the instance document. The type is identified using the xsi:type attribute which is part of the XML Schema instance namespace. The value of this attribute is a 'QName'.

In Designer, wherever type substitution option is selected, an element of the base type (which is not abstract) is represented as a choice section with the element of base type and the element of all the derived types added as its children, meaning that any of the structures can be used and also the "type" attribute used to specify the desired type in the instance document is added as a sub-field of that section. If the base type is abstract, i.e., if the attribute abstract = "true" is set for the base type, then the children of the choice section will not have an element of the base type.

Type Substitution options

The option of applying type substitution in the instance document is left with the user by allowing him to select any of the three options in the "Schema Import Options" window as seen under:



If applicable

This is the default option and is provided to support the type substitution feature for applicable elements. If this option is selected, then the imported schema would have a Designer representation so as to support type substitution of applicable elements as described above.

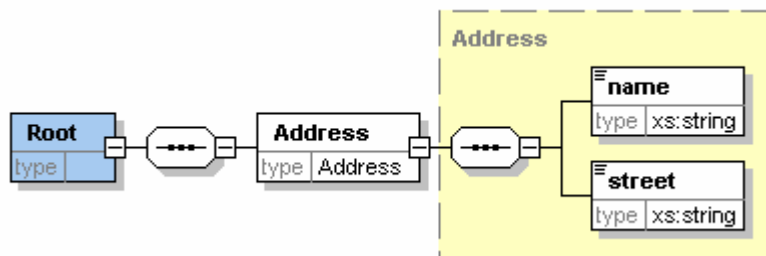
Consider the following schema components:

```
<xs:complexType name="Address">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="USAddress">
  <xs:complexContent>
    <xs:extension base="Address">
      <xs:sequence>
        <xs:element name="zip" type="xs:positiveInteger"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Here "Address" is a complex type definition and "USAddress" is another complex type definition derived by extension from "Address".

Consider the following schema:



Here there is an element added under the Root with type "Address". When this schema is imported in the Designer with the default import options (By default, the option "If applicable" is checked in the "Support type substitution" pane), the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Address	Section	Element	
	type	String	QName	
	Address	Section	Element	
	name	String	string	
	street	String	string	
	USAddress	Section	Element	
	name	String	string	
	street	String	string	
	zip	BigInteger	positiveInteger	

Field Properties

XML Name

Required
☒







Namespace

Default Value

Node Type

Designer Representation details




1. The "Address" element which is of the base type is represented as a choice section ('Compositor' as choice) with two sub-sections: "Address" section with structure conforming to an element of "Address" type and "USAddress" section with structure conforming to an element of "USAddress" type (derived type), meaning that an "Address" element in the instance document can use any one of the structures.
2. A sub-field "type" with Node Type as "attribute" and XML Type as "Qname" is added as a sub-field of the above choice section to represent the "xsi:type" attribute. The default value of this field is specified as "Address" (base type with the namespace), meaning that if no value is given for this attribute, it will be taken as the base type by default.
3. In the instance document, if the value of the type attribute is given as "USAddress", then the "Address" element can have the structure of "USAddress" element.
4. In this case, the "Address" type is not abstract. If it were so, then the Designer representation for the "Address" element would be as under:

Field Name	Type	XML Type
 Address	Section	Element
 type	String	QName
 USAddress	Section	Element
 name	String	string
 street	String	string
 zip	BigInteger	positiveInteger

Note that unlike the previous case, here, the “Address” section does not have the “Address” sub-section as its child. Also, in this case, no default value would be shown for the “type” attribute.

Never

If this option is selected, then type substitution feature will not be supported even if it is applicable. The above schema, when imported with this option selected would have the following Designer representation:

Field Name	Type	XML Type
 Address	Section	Element
 name	String	string
 street	String	string

Designer Representation details

The “Address” element is represented as a separate entity without any relation to the “USAddress” type (derived type).

Abstract types only

This option is provided to support type substitution for elements whose complex type definition has the “abstract” attribute set to true. If this option is selected, then type substitution feature will be supported only for elements of abstract types and not for the other elements, even if type substitution is applicable for them.

For e.g., consider the following complex type definition schema components:

```
<xs:complexType name="Address1" abstract="true">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="Address2" abstract="false">
```

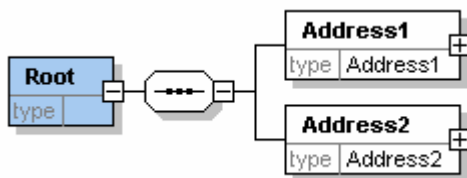
```

<xs:sequence>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="street" type="xs:string"/>
  <xs:element name="city" type="xs:string"/>
</xs:sequence>
</xs:complexType>

```

Here “Address1” is an abstract type while “Address2” is a non-abstract type. Let “USAddress” be a type derived by extension from “Address1” and “UKAddress” be a type derived by extension from “Address2”.

Consider the following schema:



When this schema is imported in the Designer with the option “Abstract types only” selected, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Address 1	Section	Element	
	type	String	QName	
	USAddress	Section	Element	
	name	String	string	
	street	String	string	
	zip	BigInteger	positiveInteger	
	Address2	Section	Element	
	name	String	string	
	street	String	string	
	city	String	string	

Section Properties			
XML Name	<input type="text" value="Address1"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	element ▼	Compositor	choice ▼

Designer Representation details

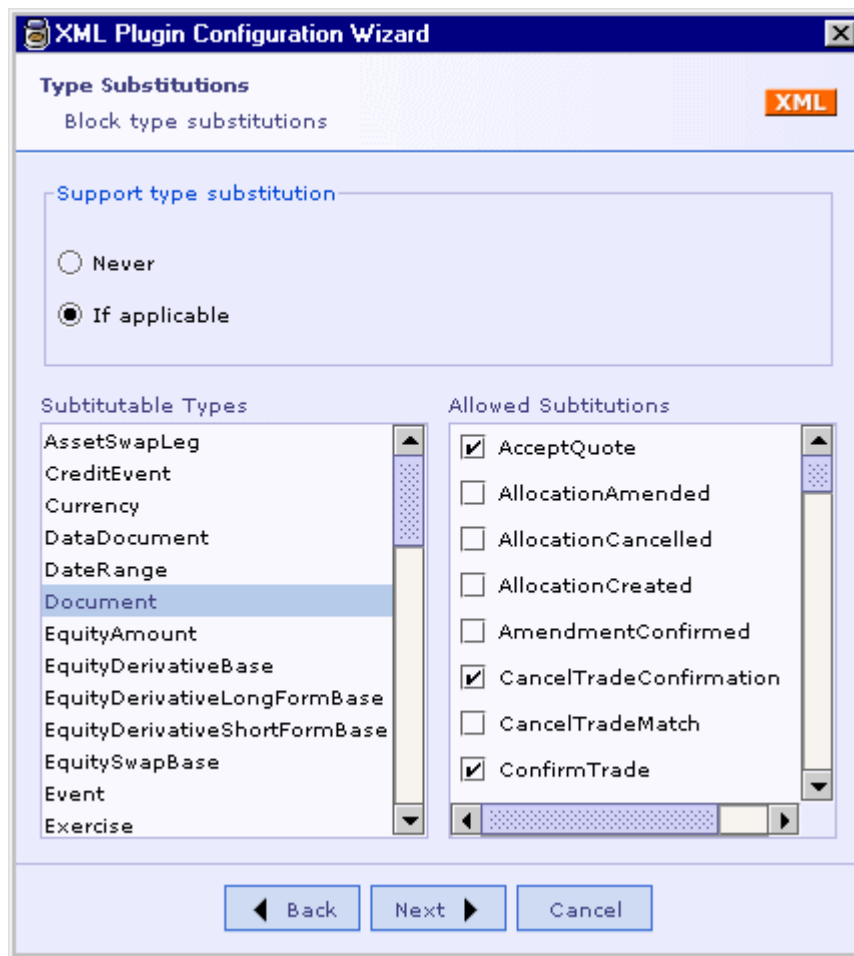
The “Address1” element whose type is abstract alone is represented in the way to support type substitution as discussed above, while the “Address2” element is

represented as a separate entity without reference to its derived type ("UKAddress" in this case)

Support for Controlling Type Substitution

While importing a schema you can now customize/change the way type substitution is applied for the global types in the schema. This feature is particularly useful if you want to block some of the allowed message types for a particular type.

While importing a schema, a page that displays all the substitutable types in the schema along with the possible substitutes for each type is shown.



To control the allowed substitutes,

1. Select the type you are interested in from the left list box.
2. The right pane (list) displays the possible substitutes for the selected type. All the types are checked by default. To block or disallow a type, uncheck it. You would

find that these types are not included in the imported representation. Hence these types of message will be rejected.

See Also:

[Aggregation](#)

[Simple Content](#)

[Complex Content](#)

[Complex Type Designer Support](#)

Complex Type Designer Support

Property	Supported
{ name }	Yes
{ target namespace }	Yes
{ base type definition }	Yes
{ derivation method }	restriction - Yes extension - Yes
{ final }	text
{ abstract }	Yes
{ attribute use pairs }	
{ attribute wildcard }	Yes
{ content type }	simple content - Yes complex Content - Yes mixed content - Yes For mixed content, all the text content are merged.
{ prohibited substitutions }	text
{ annotations }	Yes

See Also:

[Aggregation](#)
[Simple Content](#)
[Complex Content](#)
[Type Substitution](#)

Attribute Group Definition

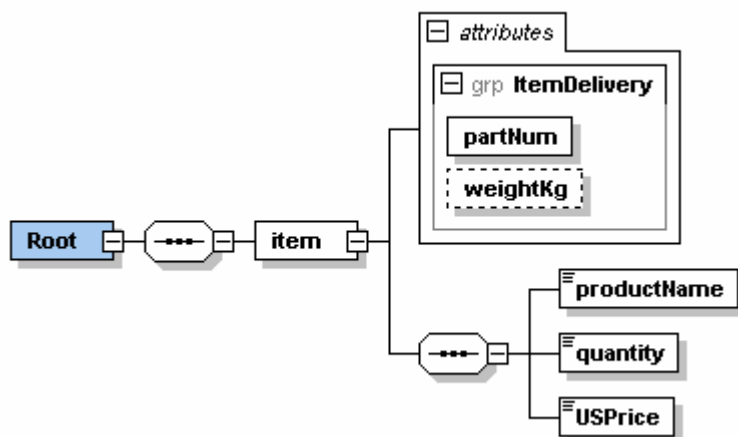
An attribute group definition is an association between a name and a set of attribute declarations, enabling re-use of the same set in several complex type definitions. Using an attribute group can improve the readability of schemas, and facilitates updating schemas because an attribute group can be defined and edited in one place and referenced in multiple definitions and declarations. Attribute group references must appear at the end of complex type definitions.

Attribute group schema component, by itself, has no default representation in the Designer. When an attribute group is referenced, each attribute in the attribute group definition becomes part of the referencing complex type definition, i.e., the contents of the attribute group are aggregated to the complex type in which the attribute group is referenced.

Consider the following Attribute Group definition schema component:







```
<xs:attributeGroup name="ItemDelivery">  
  <xs:attribute name="partNum" type="xs:string" use="required"/>  
  <xs:attribute name="weightKg" type="xs:decimal"/>  
</xs:attributeGroup>
```

Consider the following schema:



Here, an element 'item' is added under the Root and the attribute group 'ItemDelivery' is referenced in its complex type definition. When this schema is

imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 item	Section	Element	
 partNum	String	string	
 weightKg	BigDecimal	decimal	
 productName	String	string	
 quantity	BigInteger	positiveInteger	
 USPrice	BigDecimal	decimal	

Designer Representation details

The contents viz., 'partNum' and 'weightKg' of the attribute group 'ItemDelivery' are aggregated to the complex type definition of the 'item' element and are represented as the sub-fields of the 'item' element. The attribute group, by itself has no representation in the Designer.

See Also:

[Attribute Group Designer Support](#)

[Terminology](#)

[Glossary](#)

Attribute Group Designer Support

Property	Supported
{ name }	Attribute group name does not have a concrete Designer representation
{ target namespace }	
{ attribute use pairs }	
{ attribute wildcard }	Yes
{ annotation }	No

See Also:

[Attribute Group Definition](#)

Model Group

A model group is a constraint in the form of a grammar fragment that applies to lists of element information items. It is used to specify the order of occurrence of child elements of an element. There are three varieties of model group:

- Sequence
-- child elements must occur in the specified order.
- Conjunction (All)
-- child elements can occur in any order.
- Disjunction (Choice)
-- only one of the child elements must occur.

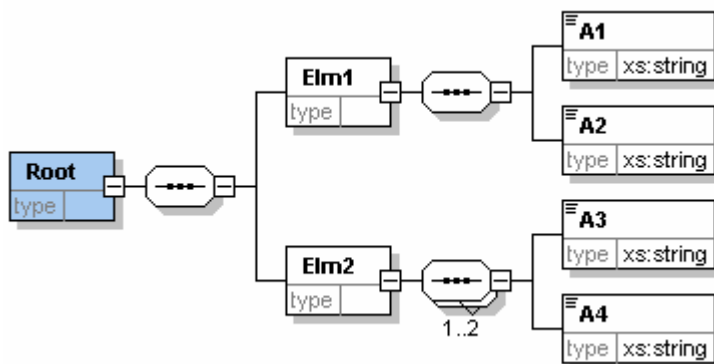
A model group can be named or anonymous. A named model group is one which is globally defined (xs:group) by associating a name and optional annotations with a model group. An anonymous model group is one which is locally defined inside a complex type definition. In case of named model group, by reference to the name, the entire model group can be incorporated into a referencing item.

Note:

The Designer representation of a named and anonymous model group is identical. In the following discussions and examples, we use the anonymous model group. However, it is equally applicable to a named model group as well.

Generally, the model group is flattened (Refer ["Model Group Flattening Rules"](#)) and its contents are aggregated to its parent. Thus, the model group has no explicit representation in the Designer. However, in some special cases such as when the model group is repeating, the contents of the model group cannot be aggregated and the model group itself is represented as a section. Refer ["Aggregation"](#).

Consider the following schema:



Here, we have three model groups, two of which are non-repeating and one is repeating with minOccurs = 1 & maxOccurs = 2. When this schema is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
Elm 1	Section	Element	
A1	String	string	
A2	String	string	
Elm2	Section	Element	
Elm2Group 1	Section	Element	
A3	String	string	
A4	String	string	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Designer Representation details

1. The model group under "Root" is flattened (has no explicit Designer representation) and its children viz. 'Elm1' and 'Elm2' are aggregated to the root element and represented at the root level. (Root element has no explicit representation in the Designer)
2. Similarly, the model group under 'Elm1' is also flattened and its contents viz., 'A1' and 'A2' are aggregated to the 'Elm1' element.
3. The compositor of the above model group namely "sequence" is also pushed to the parent element 'Elm1' and represented by the 'sequence' item set in the "Compositor" list box in the "Section Properties" pane.
4. The model group under 'Elm2' is not flattened since it is repeating and hence represented as a section 'Elm2Group1' with 'Node Type' as "group" and 'Compositor' as "sequence" with the contents viz., 'A3' and 'A4' shown as sub-fields of the section.

See Also:

[Model Group represented as section](#)
[Model Group Definition Designer Support](#)

Model Group Flattening Rules

In the following cases, the model group is flattened, i.e., the model group is removed and the contents and the compositor of the model group are pushed to its parent.

Single model group flattening

Where applicable

The following conditions should be satisfied for this rule to be applied:

- Element has a single model group as its child. (the model group may be 'sequence', 'choice' or 'all'.
- The model group is mandatory.
- The model group is non-repeating.

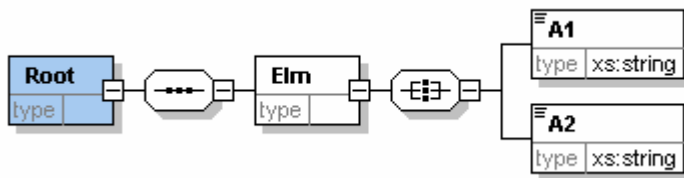
Flattening details

- The parent element (section) inherits the compositor of its child model group.
- The model group is removed.
- The child elements of the child model group are moved to its parent element, i.e. the parent section.




Note: No option need be selected

Example

Consider the following schema:



Here, the model group "all" satisfies all the three conditions mentioned above. When this schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Elm	Section	Element	
	A1	String	string	
	A2	String	string	


Section Properties

XML Name

Min Occurs
 ▼

Namespace

Max Occurs
 ▼

Node Type
 element ▼

Compositor
 ▼

Designer Representation details

1. The element 'Elm' (represented by section) inherits the compositor of its child model group. Consequently, the Compositor of the element 'Elm' is shown as "all". (Refer "Section Properties" pane)
2. The "all" model group is flattened (removed), i.e., it does not have any representation in the Designer.
3. The child elements of the "all" model group are moved to its parent element, i.e. the parent section. Thus, the child elements viz., 'A1' and 'A2' of the "all" model group are represented as the child elements of the parent section "Elm".

Remarks

This rule is always applied.

Mandatory Sequence Group Flattening

Where applicable

The following conditions should be satisfied for this rule to be applied:

- The model group is mandatory and non-repeating.
- The parent of the model group (element or another group) is *also a sequence*.

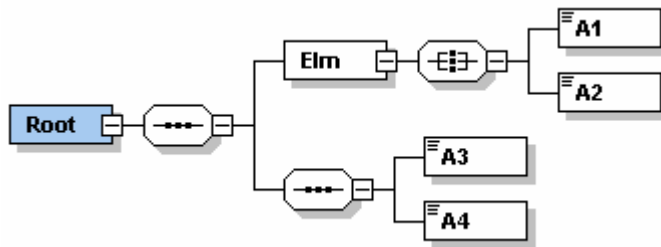
Flattening details

- The model group is removed.
- The child elements of the model group are moved to its parent (in its place) .

Note: No option need to be selected

Example

Consider the following schema:



Here, the “all” model group is mandatory & non-repeating and its parent element “Elm” has Compositor as “sequence” (before the model group was added to it). The “sequence” model group (with children as ‘A3’ & ‘A4’) is also mandatory & non-repeating & its parent is again a sequence model group. When this schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Elm	Section	Element	
	A1	String	string	
	A2	String	string	
	A3	String	string	
	A4	String	string	
Section Properties				
	XML Name	Elm	Min Occurs	1
	Namespace		Max Occurs	1
	Node Type	element	Compositor	all

Designer Representation details

1. The two model groups are flattened (removed), i.e., they don't have any representation in the Designer.
2. The child elements of the two model groups are moved to their corresponding parents, i.e. ‘A1’ and ‘A2’ of the “all” model group are represented as the child elements of the parent section “Elm” while ‘A3’ and ‘A4’ of the sequence model group are moved to the parent sequence and in turn moved to the root element.

Remarks

This rule is always applied.

Single Child Group Flattening

Where applicable

The following condition should be satisfied for this rule to be applied:

- The model group has a single child.

Flattening details

- The model group is removed
- The single child element of the model group is moved to its parent (in place of the group)
- The cardinality of the child is determined as specified in the following table:

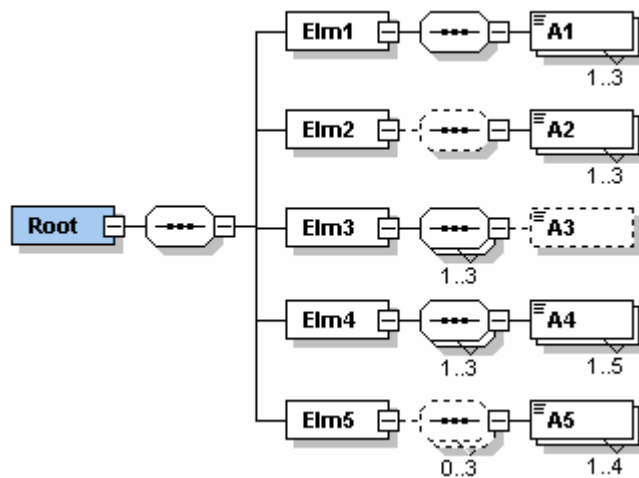
Cardinality of model group	Cardinality of the single child	Resultant cardinality of the single child
Mandatory & Non-repeating (1, 1)	Any cardinality say, (1, 3)	No change (1, 3)
Optional & Non-repeating (0, 1)	Min Occurs < 2 Max Occurs -- anything say, (1, 3)	Max Occurs is retained. Min Occurs becomes invariably zero. (0, 3)
Repeating (Optional or Mandatory)	Min Occurs < 2 Max Occurs -- anything	MinOccurs = Min Occurs of Model group x min occurs of child Max Occurs = max occurs of model group x max occurs of child
(1, 3)	(0, 1)	(0, 3)
(1, 3)	(1, 5)	(1, 15)

(0, 3)	(1, 4)	(0, 12)
--------	--------	---------

Note: No option need be selected

Example

Consider the following schema:



When this schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Elm1	Section	Element	
	A1	Section	Element	
	Elm2	Section	Element	
	A2	Section	Element	
	Elm3	Section	Element	
	A3	Section	Element	
	Elm4	Section	Element	
	A4	Section	Element	
	Value	String	string	
	Elm5	Section	Element	
	A5	Section	Element	

Section Properties			
XML Name	A4	Min Occurs	1
Namespace		Max Occurs	15
Node Type	element	Compositor	sequence

Designer Representation details

1. All the model groups are flattened (removed) & their single child is moved to the parent element.
2. The cardinality of the single child is determined as per the above table.

Remarks

1. This rule is always applied.
2. Note that when the model group is optional and repeating / non-repeating, if the min occurs of the single child is > 1 , the model group is not flattened and is retained as such as a section.

Optional group flattening

Where applicable

The following conditions should be satisfied for this rule to be applied:

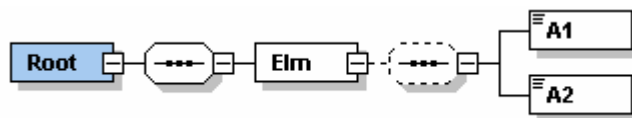
- The model group is optional and non-repeating.
- The model group has the compositor as "sequence" or "all".
- The option "Flatten optional groups" is selected in the "Schema Import Options" window.

Flattening details

- The cardinality of the model group is applied to its child elements. Since the model group is optional the child elements under it are invariably made optional.
- The model group is removed.
- The child elements under the model group are moved to the parent of the group (in place of the group).

Example

Consider the following schema:



When this schema is imported in the Designer with the option “Flatten optional groups” selected in the “Schema Import Options” window, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
Elm	Section	Element	
A1	String	string	
A2	String	string	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Designer Representation details

1. The min Occurs cardinality of the “sequence” model group is applied to its child elements viz., ‘A1’ and ‘A2’.
2. The “sequence” model group is flattened (removed), i.e., it does not have any representation in the Designer.
3. The child elements of the model group are moved to its parent, i.e. the parent section. Thus, the child elements viz., ‘A1’ and ‘A2’ of the “sequence” model group are represented as the child elements of the parent section “Elm”.

Remarks

- This rule is not applied by default. It is applied only when the option “Flatten optional groups” is selected in the “Schema Import Options” window. Else, the model group will not be flattened.
- There are both positive and negative effects in using this option. The positive effect is that the user can get rid of an extra section and thereby reduce the extra work during mapping. The negative effect is that some information is lost. In the above e.g., since the optional cardinality of the model group is applied to the child elements which are originally mandatory, the mandatory behavior of the child elements is lost.

Choice Group Flattening

Where applicable

The following conditions should be satisfied for this rule to be applied:

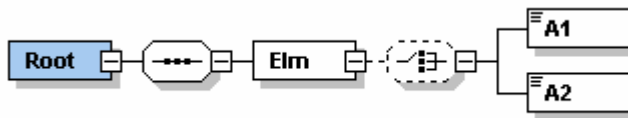
- The model group is non-repeating choice.
- Both the options “Flatten optional groups” and “Flatten non-repeating choice groups” are selected in the “Schema Import Options” window.

Flattening details

- The cardinality of the model group is applied to its child elements. Since the model group is optional the child elements under it are invariably made optional.
- The model group is removed.
- The child elements under the model group are moved to the parent of the group (in place of the group).

Example

Consider the following schema:



When this schema is imported in the Designer with both the options “Flatten optional groups” and “Flatten non-repeating choice groups” selected in the “Schema Import Options” window, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	Elm	Section	Element	
	A1	String	string	
	A2	String	string	

Section Properties

XML Name	<input type="text" value="Elm"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	element ▼	Compositor	sequence ▼

Designer Representation details

1. The min Occurs cardinality of the “choice” model group is applied to its child elements viz., ‘A1’ and ‘A2’.
2. The “choice” model group is flattened (removed), i.e., it does not have any representation in the Designer.

3. The child elements of the model group are moved to its parent, i.e. the parent section. Thus, the child elements viz., 'A1' and 'A2' of the "choice" model group are represented as the child elements of the parent section "Elm".

Remarks

- This rule is not applied by default. It is applied only when both the options "Flatten optional groups" and "Flatten non-repeating choice groups" are selected in the "Schema Import Options" window. Else, the model group will not be flattened.
- There are both positive and negative effects in using this option. The positive effect is that the user can get rid of an extra section and thereby reduce the extra work during mapping. The negative effect is that some information is lost. In the above e.g., since the optional cardinality of the model group is applied to the child elements which are originally mandatory, the mandatory behavior of the child elements is lost. Secondly, the "choice" compositor information itself is lost.

Optional behavior of Choice fields

The fields of a "choice" model group are made optional whether the "choice" model group is optional or not or whether it is flattened or not. This is due to conflict between the Designer and schema in interpreting the meaning of the word 'optional'.

See Also:

[Model Group represented as section](#)

[Model Group Definition Designer Support](#)

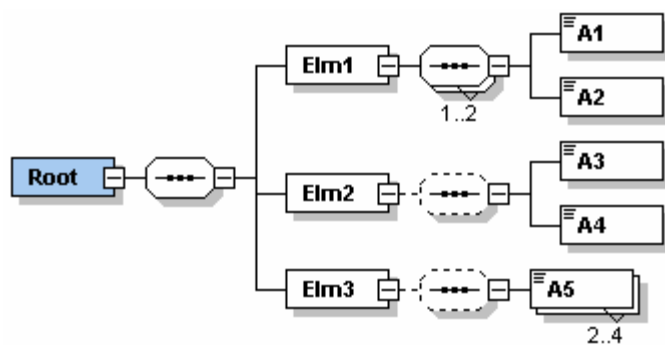
Model Group Represented as Section

In the following cases, the model group is not flattened, i.e., the model group is represented as a section.

1. Where the model group is repeating with more than one child
2. Where the model group is optional with more than one child
3. Where the optional model group has a single child whose minOccurs > 1

Example

Consider the following schema where the above three conditions are satisfied:



When this schema is imported in the Designer with the default import options, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
Elm 1	Section	Element	
Elm 1Group 1	Section	Element	
A1	String	string	
A2	String	string	
Elm 2	Section	Element	
Elm 2Group 1	Section	Element	
Elm 3	Section	Element	
Elm 3Group 1	Section	Element	

Section Properties			
XML Name	Elm1Group1	Min Occurs	1
Namespace		Max Occurs	2
Node Type	group	Compositor	sequence

Note that none of the model group is flattened and they are represented as a section.

See Also:

[Model Group Flattening Rules](#)

[Model Group Definition Designer Support](#)

Model Group Definition Designer Support

Property	Supported
{ name }	Yes (if model group is not flattened)
{ target namespace }	Yes (if model group is not flattened)

{ model group}	all, choice, sequence – Yes
{ annotation}	Yes (if model group is not flattened)

Property	Supported
{ compositor}	Yes
{ particles}	Yes
{ annotation}	Yes (if model group is not flattened)

See Also:

[Model Group Flattening Rules](#)

[Model Group represented as section](#)

Schema Attribute Declaration

An attribute declaration is an association between a name and a simple type definition, together with occurrence information and (optionally) a default value. Attributes may be declared locally or globally. Once declared, a global attribute can be referenced in one or more declarations using the 'ref' attribute. Attributes can be declared with a 'use' attribute to indicate whether the attribute is required, optional, or even prohibited. Default values of attributes are declared using the 'default' attribute. Default attribute values apply when attributes are missing. The 'fixed' attribute is used in attribute declarations to ensure that the attributes are set to particular values. Note that the concepts of a fixed value and a default value are mutually exclusive, and so it is an error for a declaration to contain both fixed and default attributes. Attribute references must appear at the end of complex type definitions.

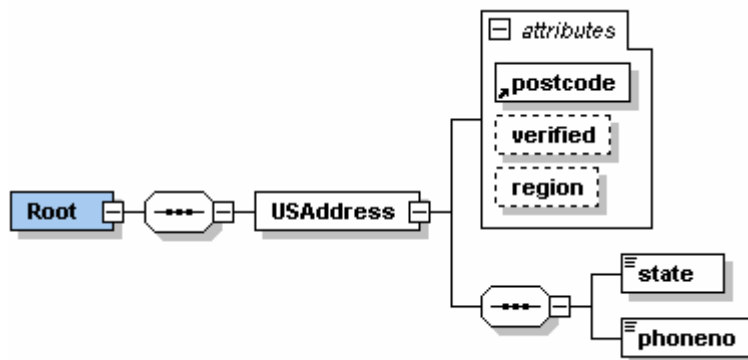
Attributes are always aggregated to the complex type in which they are declared and are represented as fields in the Designer. Refer ["Aggregation"](#).

Consider the following schema components:

```
<xs:complexType>
  . . .
  <xs:attribute ref="postcode" use="required"/>
  <xs:attribute name="verified" type="xs:boolean" use="optional" default="1"/>
  <xs:attribute name="region" type="xs:string" fixed="north"/>
</xs:complexType>

<xs:attribute name="postcode" type="xs:string"/>
```

Here two attributes are added locally inside the Complex Type definition and one is added globally & referenced. Consider the following schema:



When this schema is imported in the Designer with the default import options, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	USAddress	Section	Element	
	region	String	string	
	verified	Boolean	boolean	
	postcode	String	string	
	state	String	string	
	phoneno	BigInteger	positiveInteger	

Field Properties	
XML Name	verified
Required	<input type="checkbox"/>
Namespace	
Default Value	1
Node Type	attribute
Facets	

Designer Representation details

1. The locally defined as well as the referenced attributes are aggregated to the "USAddress" element.
2. The "use" attribute is represented by the "Required" checkbox in the "Field Properties" pane.
3. The "default" and "fixed" value, if set for the attributes are represented by "Default Value" text boxes in the "Field Properties" pane.
4. The sequential order of occurrence of attributes is not supported; i.e., though as per the schema the order of occurrence of the attributes is "postcode" followed by "verified" followed by "region", they are imported in the following order: "region", "verified" and "postcode".
5. Note that if the "use" for an attribute were set as "prohibited", then that attribute would not be imported in the Designer.

See Also:






[Ignore Fixed Attributes](#)

[Attribute Declaration Designer Support](#)

[Glossary](#)

Ignore Fixed Attributes (Schema Attribute Declaration)

“Ignore Fixed Attributes” option is provided in the “Schema Import Options” window. When this option is checked, the fixed value attributes (attributes for which ‘fixed’ attribute is specified) will not be imported in the Designer. This is particularly useful in cases where fixed value attributes are used to provide meta-info or additional information about the elements in the schema. In the above e.g., for the “region” attribute, the attribute fixed = “north” is specified. When the above schema is imported in the Designer with the option “Ignore Fixed Attributes” selected, the equivalent Designer representation would be as under:

	Field Name	Type	XML Type	Description
	 USAddress	Section	Element	
	 verified	Boolean	boolean	
	 postcode	String	string	
	 state	String	string	
	 phoneno	BigInteger	positiveIn...	

Note that the “region” attribute is not imported in this case.

See Also:

[Attribute Declaration Designer Support](#)

Attribute Declaration Designer Support

Property	Supported
{ name }	Yes
{ target namespace }	Yes
{ type definition }	Yes
{ scope }	Yes
{ value constraint }	Yes
{ annotation }	Yes

See Also:

[Ignore Fixed Attributes](#)

Schema Element Declaration

An element declaration is an association of a name with a type definition, either simple or complex, an optional default value and a possibly empty set of identity-constraint definitions. The association is either global or scoped to a containing complex type definition. Once declared, a global element can be referenced in one or more declarations using the “ref” attribute. The declaration of a global element also enables the element to appear at the top-level of an instance document. Cardinality constraints cannot be placed on global declarations, although they can be placed on local declarations that reference global declarations. In other words, global declarations cannot contain the attributes “minOccurs”, “maxOccurs” (where element is of complex type), or “use” (element is of simple type).

Default and Fixed attributes apply to an element of simple type definition. When an element is declared with a default value, the default value applies when the element appears without any content in the instance document and not when the element itself is missing. The “fixed” attribute is used in element declarations to ensure that the elements are set to particular values.

Elements are always aggregated to the complex type definition in which they are defined. An element is represented as a field or a section in the Designer depending on its contents and occurrence constraints. Refer [“Aggregation”](#).

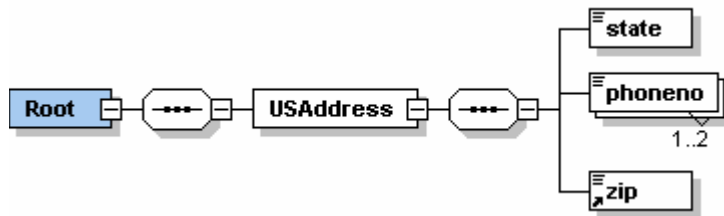
Consider the following schema components:

```
<xs:element name="USAddress">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="state" type="xs:string" default="PA"/>
      <xs:element name="phoneno" type="xs:positiveInteger" maxOccurs="2"/>
      <xs:element ref="zip"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="zip" type="xs:integer" fixed="95819"/>
```

Here “USAddress” is an element with complex type definition, while “state”, “phoneno” and “zip” are elements with simple type definition, where “zip” is a globally declared element and is referenced inside the “USAddress” element’s complex type definition. The “state” element has “default” attribute set; the element

“phoneno” has occurrence constraint set and the element “zip” has “fixed” attribute set. Consider the following schema:



When this schema is imported in the Designer with the default import options, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
USAddress	Section	Element	
state	String	string	
phoneno	Section	Element	
Value	BigInteger	positiveInteger	
zip	String	string	

Section Properties

XML Name: Min Occurs:

Namespace: Max Occurs:

Node Type: Compositor:

Designer Representation details

1. The complex type element “USAddress” is represented as a section. The locally defined as well as the referenced sub-elements are aggregated to the “USAddress” element and are represented as its children.
2. The elements “state” and “zip” are represented as sub-fields of “USAddress” section as they are non-repeating. The element “phoneno” is represented as a sub-section with a ‘Value’ field since it is repeating. All the three have ‘Node Type’ as “element”.
3. The “default” and “fixed” attributes set for the elements are represented by the “Default Value” text box in the “Field Properties” pane.
4. The occurrence constraints of the non-repeating elements (“state” and “zip”) are represented by the “Required” check box in the “Field Properties” pane while the occurrence constraints of the repeating element (“phoneno”) are represented by the “Min Occurs” and “Max Occurs” combo boxes in the “Section Properties” pane.





See Also:

[Ignore Fixed Elements](#)

[Element Substitution](#)
[Element References](#)
[Nil Value Elements](#)
[Element Declaration Designer Support](#)

Ignore Fixed Elements

“Ignore Fixed Elements” option is provided in the “Schema Import Options” window. When this option is checked, the elements for which ‘fixed’ attribute is specified will not be imported in the Designer. In the above e.g., for the element “zip”, the ‘fixed’ attribute is specified. When the above schema is imported in the Designer with the option “Ignore Fixed Elements” selected, the equivalent Designer representation would be as under:

	Field Name	Type	XML Type	Description
	USAddress	Section	Element	
	state	String	string	
	phoneno	Section	Element	
	Value	BigInteger	positiveInteger	

Note that the “zip” element is not imported in this case.

See Also:

[Element Substitution](#)
[Element References](#)
[Nil Value Elements](#)
[Element Declaration Designer Support](#)

Element Substitution

XML Schema provides a mechanism, called substitution groups, that allows elements to be substituted for other elements. More specifically, elements can be assigned to a special group of elements that are said to be substitutable for a particular named element called the head element. (Note that the head element as well as the substitutable elements must be declared as global elements). Elements in a substitution group must have the same type as the head element, or they can have a type that has been derived from the head element's type. The existence of a substitution group does not require any of the elements in that class to be used, nor does it preclude use of the head element. It simply provides a mechanism for allowing elements to be used interchangeably.

For e.g., consider the following element declaration schema components:

```
<xs:element name="comment" type="xs:string"/>
```



```
<xs:element name="shipComment" type="xs:string"
    substitutionGroup="comment"/>
```








```
<xs:element name="customerComment" type="xs:string"
    substitutionGroup="comment"/>
```

Here, the elements “shipComment” and “customerComment” are assigned to a substitution group whose head element is “comment”. This would mean that the elements “shipComment” and “customerComment” could be used in any place that we are able to use comment.

Consider the following schema component:

```
<xs:element name="item" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="productName" type="xs:string"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="USPrice" type="xs:decimal"/>
      <xs:element ref="comment"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Here the element “comment” is referenced. When this “item” element is added somewhere under the Root element of a schema and when that schema is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type
 item	Section	Element
 productName	String	string
 quantity	BigInteger	positiveInteger
 USPrice	BigDecimal	decimal
 [?] comment	String	string
 [?] customerComment	String	string
 [?] shipComment	String	string

Designer Representation details

1. The head element viz., “comment” and the elements of the substitution group viz., “customerComment” and “shipComment” are shown as optional fields under the “Items” element.
2. Basically they are considered as children of a choice section and the choice section is flattened and the elements are aggregated to the “item” element.
3. Note that due to the above flattening, the choice functionality is lost.


See Also:[Ignore Fixed Elements](#)[Element References](#)[Nil Value Elements](#)[Element Declaration Designer Support](#)

Element References

Sometimes, in the schema definition, it is preferable to use an existing element rather than declare a new element. In such cases, the 'ref' attribute is used in the element declaration. The value of "ref" attribute must reference a [global element](#) and it enables the referenced element to appear in the instance document in the context of the referencing declaration.

In the Designer, there is a specific representation for a reference element provided the referred element is of complex type and the element is referred more than once. This specific representation is applicable to the second and subsequent usage of the referred element, and not to the first usage.

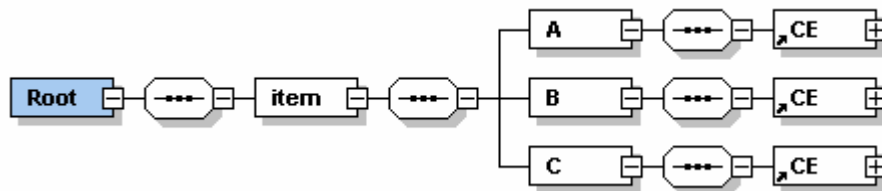
Reference Element representation

The reference element is represented in the Designer by a circle icon super-imposed on the section (folder) icon . Also, the full path of the first occurrence of the referred element is shown as description.










For e.g., let 'CE' be a complex type element declared globally with the following definition:

```
<xs:element name="CE">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ce1" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Consider the following schema, where the element 'CE' is referred in three places:



When the above schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	 item	Section	Element	
	 A	Section	Element	
	 CE	Section	Element	
	 ce1	String	string	
	 B	Section	Element	
	 CE	Section	Element	Ref section CE in Root.Group0.item.Group1.A.Group2
	 C	Section	Element	
	 CE	Section	Element	Ref section CE in Root.Group0.item.Group1.A.Group2
	 ce1	String	string	

Designer Representation details

1. The first reference of 'CE' element (added as child of 'A') is shown in the usual manner.
2. The second and third references of 'CE' element (added as child of 'B' and 'C') are shown as reference elements. See "Reference Element representation".

Note:

1. This representation is particularly useful in cases where you refer an element in a recursive manner, i.e., where a child element refers to a parent element.
2. In case the referred element is of simple type, all usages of the referred element are represented in the normal (and identical) fashion (provided there are no naming conflicts), i.e. as a field.

See Also:

[Ignore Fixed Elements](#)

[Element Substitution](#)

[Nil Value Elements](#)

[Element Declaration Designer Support](#)

Nil Value Elements

XML Schema's nil mechanism enables an element to appear with or without a non-nil value. This mechanism involves an "out of band" nil signal. In other words, there is no actual nil value that appears as element content, instead there is an attribute to indicate that the element content is nil. To allow an element to have nil value, you



have to set the attribute “nillable” to true in the element declaration and in the instance document, to represent that the element has a nil value, you have to set the “xsi:nil” attribute to true. Note that the nil mechanism applies only to element values, and not to attribute values. An element with xsi:nil = “true” attribute set, may not have any element content but it may still carry attributes.

A mandatory element, for which the nillable attribute is set to true is represented in the Designer as an optional element.


For e.g., consider the following schema:

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="A" type="xs:positiveInteger"/>
      <xs:element name="B" type="xs:positiveInteger" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above schema, we have two elements ‘A’ and ‘B’ added under the root element. Both elements are mandatory and have the same data type. For the element ‘B’ alone, the ‘nillable’ attribute is set to true. When the above schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	A	BigInteger	positiveInteger	
	B	BigInteger	positiveInteger	

Field Properties

XML Name	<input type="text" value="B"/>	Required	<input type="checkbox"/>
Namespace	<input type="text"/>	Default Value	<input type="text"/>
Node Type	 element ▼	<input type="button" value="Facets"/>	

Designer Representation details

The mandatory element ‘B’ is shown as an optional element.

Consider the following instance document where the attribute “xsi:nil” attribute is set to true for the element ‘B’.

```
<?xml version="1.0" encoding="UTF-8"?>
  <root xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\..\NillableElement.xsd">
    <A>1</A>
    <B xsi:nil = "true"></B>
  </root>
```

The corresponding “Parse and Write” output would be as under:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <A>1</A>
    <B xsi:nil="true"/>
</root>
```

Note that though the element ‘B’ is mandatory, setting the ‘xsi:nil’ attribute to true enables us to use the element with nil content.

See Also:

[Ignore Fixed Elements](#)

[Element Substitution](#)

[Element References](#)

[Element Declaration Designer Support](#)

Element Declaration Designer Support

Property	Supported
{ name }	Yes
{ target namespace }	Yes
{ type definition }	Yes
{ scope }	Yes
{ value constraint }	Yes
{ nillable }	
{ identity-constraint definitions }	No

{ substitution group affiliation}	Yes
{ substitution group exclusions}	
{ disallowed substitutions}	
{ abstract}	Yes
{ annotation}	Yes

See Also:

[Ignore Fixed Elements](#)

[Element Substitution](#)

[Element References](#)

[Nil Value Elements](#)

Annotation

An annotation is information for human and/or mechanical consumers. Annotation of schemas and schema components, with material for human or computer consumption, is provided for by allowing application information and human information at the beginning of most major schema elements, and anywhere at the top level of schemas. The content of an annotation may be a “documentation” and / or an “appinfo”. “documentation” element is the recommended location for human readable material. The “appinfo” element can be used to provide information for tools, stylesheets and other applications.

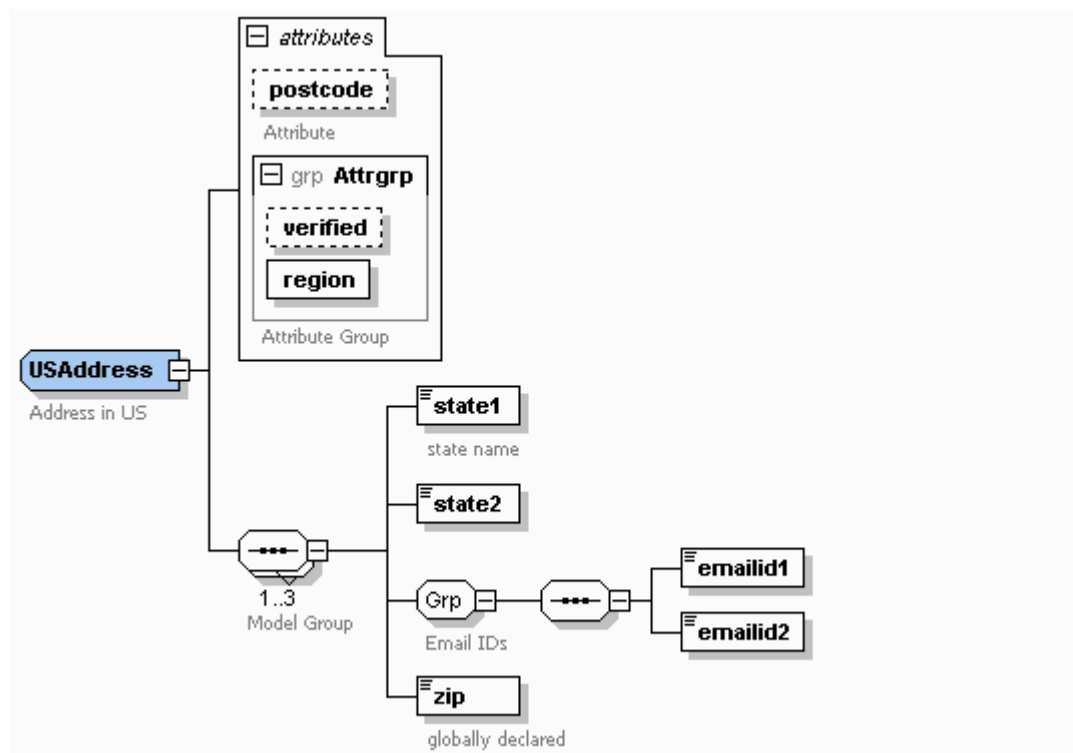
Annotation can be specified for various schema components like Element declaration, Complex Type definition, simpleType definition, Attribute, Attribute Group, etc.

The “documentation” element of the Annotation is shown as Description in the Designer. The Designer representation of annotation added to various schema components is explained in the following table.











Schema Component to which annotation is added	Designer Representation
Only for Base Type & not for Derived type	Shown as description for element of derived type
Both for Base Type & Derived type	Both annotations are accumulated & shown as description for element of derived type

Global Simple Type definition	Description of element of that simple type, provided not overwritten at the element level.
Element (both local & global)	Description of the element
Attribute	Description of the attribute
Attribute Group	No representation (information is lost)
Model Group	If the group is flattened, no representation. If not flattened, represented as the description of the equivalent section

Consider the following complex type definition:



Here “state1” & “state2” are elements of type “USState” which is a simple type definition with annotation (documentation) as “State in US”. For “state1” another annotation is specified at the element level. (Annotations specified for the schema components are displayed under each schema component in the above picture). When an element “USAddress” is added under the root with type as “USAddress” and when that schema is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 USAddress	Section	Element	Address in US
 region	String	string	
 verified	Boolean	boolean	
 postcode	String	string	Attribute
 USAddressGroup 1	Section	Element	Model Group
 state1	String	string	state name
 state2	String	string	State in US
 emailid1	String	string	
 emailid2	String	string	
 zip	BigInteger	positiveInteger	globally declared

Designer Representation details

1. For the element "state1", the annotation of its type "USState" is overwritten by its own annotation.
2. The annotation of attribute group is lost as the attribute group is flattened.
3. The annotation of the model group "Grp" is lost since it is flattened while the annotation of the repeating model group is represented as description against the section "USAddressGroup1".

See Also:

[Annotation Designer Support](#)

[Binding XML Schema to Designer](#)

Annotation Designer Support

Property	Supported
{ application information }	No
{ user information }	Yes (where schema component is not flattened)

See Also:

[Annotation](#)

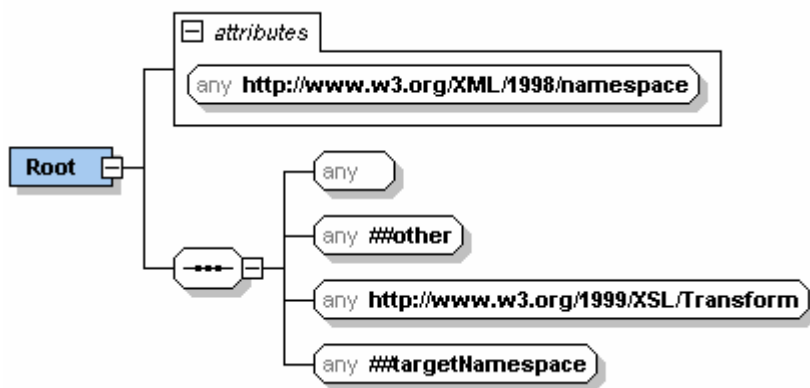
Wildcard

A wildcard provides for validation of attribute and element information items dependent on their namespace name, but independently of their local name.

Consider the following complex type definition schema component where the four basic types of wildcard and one attribute wildcard are added.

```
<xs:complexType>
  <xs:sequence>
    <xs:any processContents="skip"/>
    <xs:any namespace="##other" processContents="lax"/>
    <xs:any namespace="http://www.w3.org/1999/XSL/Transform"/>
    <xs:any namespace="##targetNamespace"/>
  </xs:sequence>
  <xs:anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
</xs:complexType>
```

Consider the following schema:



Here the Root element has the above mentioned complex type definition. When this schema is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	anyAttribute	Section	Element	
	any	Section	Element	
	any2	Section	Element	
	any3	Section	Element	
	any4	Section	Element	

Section Properties

XML Name

any

Min Occurs

1

Namespace

http://www.w3.org/1999/XS

Max Occurs

1

Node Type

= any

Compositor

sequence

Designer Representation details

1. The “any” elements are represented as sections (the names “any2”, “any3” and “any4” are auto-generated to avoid naming collision) with ‘Node Type’ as “any”.
2. The “anyAttribute” is also represented as a section with the same name with ‘Node Type’ as “anyAttribute”.
3. The value of the “namespace” attribute specified for the ‘any’ elements are displayed in the “Namespace” list box in the “Section Properties” pane.

See Also:

[Wildcard Designer Support](#)

[Binding XML Schema to Designer](#)

Wildcard Designer Support

Property	Supported
{ namespace constraint }	Yes
{ process contents }	Designer does not validate contents, be it skip, lax or strict.
{ annotation }	No

See Also:

[Wildcard](#)

Identity-Constraint Definition

Identity-constraint definition components provide for uniqueness and reference constraints with respect to the contents of multiple elements and attributes. The three kinds of identity-constraint definitions are xs:key, xs:keyref and xs:unique. Identity-constraint definition is not supported in the Designer.

See Also:

[Simple Type Definition](#)

[Complex Type Definition](#)

[Attribute Group Definition](#)

XML Name Mapping

Generally, the XML name is mapped as it is to the Designer name. In special cases as listed below, the Designer applies either conversion mechanism to convert XML

name to Designer acceptable names, or it coins new names to represent unnamed XML schema components:

1. Where the XML name is not a valid Designer name
2. Where there are name collisions and
3. Where anonymous model groups are not flattened.

However, only the Designer name is modified and the XML name is retained as such.

See Also:

[XML Name of Element / Attribute to Designer Name](#)

[Model Group Name to Designer Name](#)

[Collisions and conflicts](#)

XML Name of Element / Attribute to Designer Name

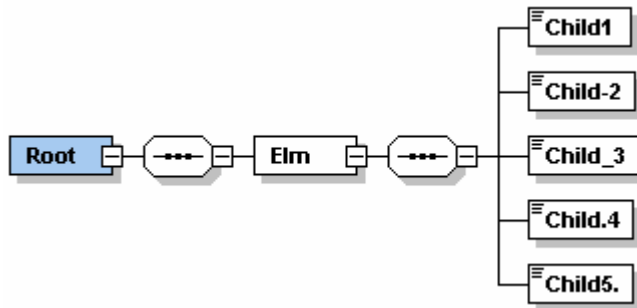
Generally, when a schema is imported in the Designer, the XML name specified for an element / attribute in the schema is mapped as it is to the corresponding field / section name in the Designer. In case, the XML name has characters not accepted by the Designer, they are converted to “underscore”. However, the XML name of the element / attribute is retained as such, i.e., the “XML Name” text box in the “Field Properties” / “Section Properties” pane shows the same value for the element / attribute names as specified in the schema. The following table explains the mapping from XML Name to Designer Name:

XML Name of element / attribute	Designer Name (Field Name column)	Remarks
Child1	Child1	No punctuation used. Hence Designer Name is same as XML name.
Child-2	Child-2	‘Hyphen’ accepted in Designer Name. Hence, Designer Name is same as XML name.
Child_3	Child_3	‘Underscore’ accepted in Designer Name. Hence, Designer Name is same as XML name.
Child.4	Child_4	‘Dot’ not accepted in Designer Name. It is converted to ‘underscore’. (XML Name not changed)
Child5.	Child5_	‘Period’ not accepted in Designer

		Name. It is converted to 'underscore'. (XML Name not changed)
--	--	--

The following pictures explain the above table visually:

XSD:



Designer equivalent:

Field Name	Type	XML Type
Elm	Section	Element
Child1	String	string
Child-2	String	string
Child_3	String	string
Child_4	String	string
Child5_	String	string

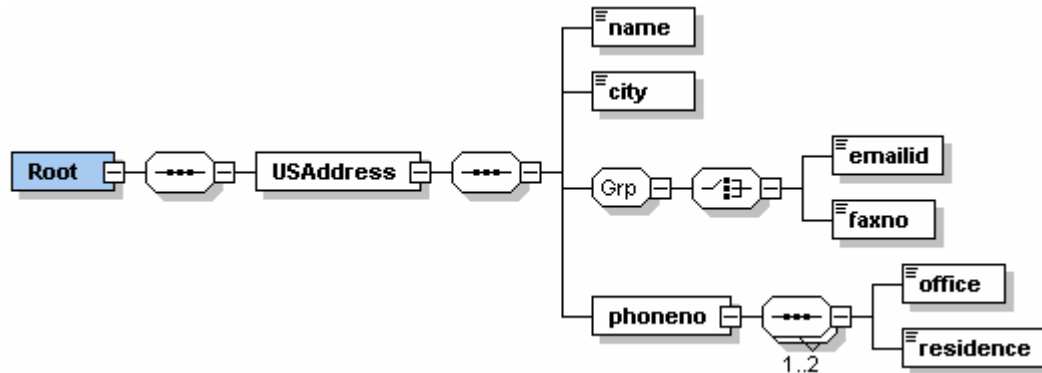
See Also:

[Model Group Name to Designer Name](#)
[Collisions and conflicts](#)

Model Group Name to Designer Name

There are named (xs: group) and anonymous model groups in XSD. During schema import, the Designer automatically flattens (removes) these model groups wherever applicable ,also provides the user, the option to flatten the model group in certain cases in the “Schema Import Options” window. Wherever the model group is not flattened, the computation of the equivalent Designer name for the model group depends on whether the model group is named or anonymous. If the model group is named, the name is mapped as it is to the corresponding section in the Designer. Where the model group is an anonymous one, a new name is invented for the corresponding section depending on the name of the parent element. The following pictures throw light on the above discussion:

XSD:



Designer equivalent: (Above XSD imported with default options)

Field Name	Type	XML Type
USAddress	Section	Element
name	String	string
city	String	string
Grp	Section	Element
emailid	String	string
faxno	String	string
phoneno	Section	Element
phonenoGroup 1	Section	Element
office	BigInteger	integer
residence	BigInteger	integer

In the above XSD, leaving the model group following the “Root” element, there are three model groups; two are anonymous while one is named. The anonymous model group following the “USAddress” element is flattened by default; the named model group “Grp” is not flattened & hence represented by a section with the same name as found in the schema; the anonymous model group following the “phoneno” element is not flattened as it is repeating & is represented by a section with the name “phonenoGroup1” which name is arrived at by concatenating the two strings “phoneno” and “Group1”, where “phoneno” is the name of the parent element while “Group1” indicates that this is the first non-flattened model group occurring under the section “phoneno”. The subsequent anonymous and non-flattened model groups would be named “phonenoGroup2”, “phonenoGroup3” and so on.

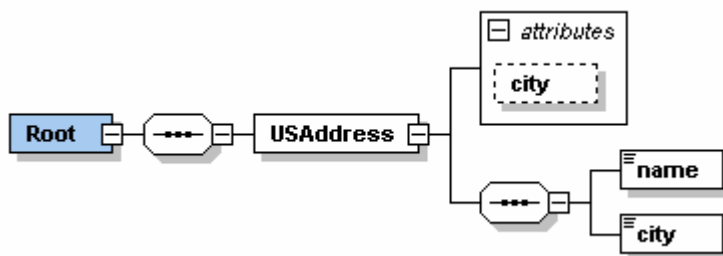
See Also:

[XML Name of Element / Attribute to Designer Name Collisions and conflicts](#)

Collisions and Conflicts

During mapping of XML Name to Designer name, conflicts may arise in some cases. For e.g., an XML component can have two distinct sub-components with same XML name. However, the Designer does not permit duplicate field / section names at the same level (there being only two possible representations for any XML element in the Designer, they being a field or a section). Note that in the Designer though a field can be represented as an element, attribute or value, depending on the “Node Type” selected in the “Field Properties” pane, all the three represent only a field internally. To overcome naming conflicts, during import of a schema, the Designer automatically adjusts some field / section names to avoid validation errors. However, the XML name is not altered.

For e.g., consider the following schema:



In the above schema, the “USAddress” element has a sub-element with XML name as “city” and also an attribute with the same XML name. When this schema is imported in the Designer, the Designer representation would be as under:

	Field Name	Type	XML Type	Description
	USAddress	Section	Element	
	city	String	string	
	name	String	string	
	city2	String	string	

Field Properties

XML Name

city

Required

☒

Namespace

Default Value

Node Type

element

Facets

Here, the attribute as well as the sub-elements is represented as sub-field of the section “USAddress” and they are at the same level. Hence, if the same name were retained for the sub-element “city”, it would have resulted in duplicate field names and consequently would result in validation error. To avoid this, during import, the Designer automatically maps the conflicting name to a new name. In this case, the XML name “city” of the sub-element is represented as “city2” in the Designer.

However, the XML name is not modified. (See [“Field Properties”](#) pane). Note that this name change would in no way affect the parsing or writing of a valid XML instance, i.e., during runtime, the parser would expect a “city” sub-element and not a “city2” sub-element.

See Also:

[XML Name of Element / Attribute to Designer Name](#)

[Model Group Name to Designer Name](#)

W3C XML Schema Support

Click the following links to view the Designer support details of various W3C XML Schema components.

[Simple Type Designer Support](#)

[Complex Type Designer Support](#)

[Attribute Group Designer Support](#)

[Model Group Definition Designer Support](#)

[Attribute Declaration Designer Support](#)

[Element Declaration Designer Support](#)

[Annotation Designer Support](#)

[Wildcard Designer Support](#)

[Identity Constraint definition](#)

[Terminology](#)

[Glossary](#)

Binding DTD to Designer

The XML Plugin allows you to import a DTD in the Designer. Roughly speaking, the XML Plugin maps a DTD component to a corresponding Designer type. This chapter describes the Designer representation of each DTD component and also the features of the DTD component supported by the Designer. You will find below a detailed list of the equivalent Designer representation for each DTD Component

See Also:

[DTD Element declaration](#)

[Group declaration](#)

[Attribute declaration](#)

[Terminology](#)

[Glossary](#)

DTD Element Declaration

If you use a DTD to describe the XML message structure, every element used in a valid XML instance must be declared in that DTD using the ELEMENT declaration, which consists of an element name followed by a description of the element's contents. Based on the content, elements in XML documents can be categorized into five heads. The following sections describe the DTD structure to describe these element types and their corresponding Designer representation.

The types of elements that are possible are

[Element with Character Data only](#)

[Element with Standard Content](#)

[Element with Mixed Content](#)

[Element with Any Content](#)

[Element with No Content Allowed](#)

See Also:

[Group declaration](#)

[DTD Attribute declaration](#)



Element with Character Data Only

This type of element can contain only parsed character data and not any other content such as (sub-elements or attributes). The keyword #PCDATA describes basic elements that contain parsed character data. A DTD component defining an element with "character data only" is typically represented as a field in Designer. However, in case where the element is repeating, it is represented as a section.

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (child1, child2*)>
<!ELEMENT child1 (#PCDATA)>
<!ELEMENT child2 (#PCDATA)>
```

Here, we have 2 elements added under the Root, which allow just character data. The element "child1" is non-repeating while "child2" is repeating. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	child1	String	string	
	child2	Section	Element	
	Value	String	string	


Field Properties

XML Name

Required ☒

Namespace

Default Value

Node Type  element ▼

Designer Representation details

1. The non-repeating element "child1" is represented as a field with 'Node Type' as "element" while the repeating element "child2" is represented as a section with 'Node Type' as "element" with the "Value" sub-field added.
2. The character content of "child1" is represented by the field itself (data type "string" in the 'XML Type' column in the table).
3. The character content of "child2" is represented by the "Value" sub-field whose 'Node Type' is "value".
4. The XML name of the element is displayed in the 'XML Name' textbox.
5. The mandatory / optional behavior of the non-repeating element "child1" is represented by the checking / unchecking of the 'Required' checkbox in the "Field Properties" pane.
6. The occurrence constraints of the repeating element "child2" is represented by the value in the 'Min Occurs' and 'Max Occurs' combo boxes in the "Section Properties" pane.

See Also:

[Element with Standard Content](#)
[Element with Mixed Content](#)
[Element with Any Content](#)
[Element with No Content Allowed](#)

Element with Standard Content

An element with standard content can have one or more child elements associated with it and also it can have attributes. Specifically, it should not contain text. A DTD component defining an element with standard content is represented in the Designer as a section with sub-fields (with Node Type as "element") / sub-sections representing the sub-elements and sub-fields (with Node Type as "attribute") representing the attributes.

Consider the following DTD:

```

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (item)>
<!ELEMENT item (productName, quantity)>
<!ELEMENT productName (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ATTLIST item weightKg CDATA #IMPLIED
              partNum CDATA #REQUIRED>

```

Here, the element “item” has child elements as well as attributes as it’s content. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
item	Section	Element	
weightKg	String	string	
partNum	String	string	
productName	String	string	
quantity	String	string	

Section Properties

XML Name
Min Occurs

Namespace
Max Occurs

Node Type element
Compositor

Designer Representation details

1. The element “item” is represented as a section with ‘Node Type’ as “element” with sub-fields added to represent the sub-elements and attributes.
2. In this case, since the child elements of the “item” element are non-repeating, they are represented as sub-fields with ‘Node Type’ as “element”. (repeating child elements would be represented as sections)
3. The attributes of the “item” element are represented as sub-fields with ‘Node Type’ as “attribute”.
4. The XML name of the “item” element is displayed in the ‘XML Name’ text box.
5. The occurrence constraints of the “item” element is represented by the value set in the ‘Min Occurs’ and ‘Max Occurs’ combo boxes in the “Section Properties” pane.
6. The order of occurrence of the child elements is represented by the item set in the ‘Compositor’ list box. In this case, since the child elements’ occurrence is of sequential order (child elements are separated by comma), “sequence” item is selected in the list box.

See Also:

[Element with Character Data only](#)

[Element with Mixed Content](#)

[Element with Any Content](#)

[Element with No Content Allowed](#)





Element with Mixed Content

An element with mixed content can contain both text and other elements. A DTD component defining an element with mixed content is represented in the Designer as a section with sub-fields (with Node Type as “element”) / sub-sections representing the sub-elements and another sub-field (with Node Type as “value”) to represent the text content.

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (inventory)>
<!ELEMENT inventory (#PCDATA | item)*>
<!ELEMENT item (#PCDATA)>
```

Here, the element “inventory” has child element as well as text (#PCDATA) as its content. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 inventory	Section	Element	
 Value	String	string	
 item	Section	Element	
 Value	String	string	


Section Properties

XML Name

Min Occurs

Namespace

Max Occurs

Node Type  element

Compositor

Designer Representation details

1. The element “inventory” is represented as a section with ‘Node Type’ as “element” and ‘Compositor’ as “sequence”, with sub-sections added to represent the sub-element and a sub-field added to represent the text content.

2. In this case, since the child element "item" of the "inventory" element is repeating, it is represented as sub-section with 'Node Type' as "element".
3. The text content of the "inventory" element is represented by the "Value" sub-field with 'Node Type' as "value" and with 'Required' checkbox checked.
4. Note that all the text content that can appear between the elements are merged and shown as a whole by the 'Value' field & hence, the position of the text content in the XML instance would not be retained as such.

See Also:

[Element with Character Data only](#)

[Element with Standard Content](#)

[Element with Any Content](#)

[Element with No Content Allowed](#)

Element with Any Content

An element with any content can contain any type of content, including standard content, text, and mixed content. In DTD, you declare an 'any content' element with the keyword "ANY". A DTD component defining an element with any content is represented in the Designer as a section with a sub-field (with 'Node Type' as "value") representing the applicability of text content, a sub-section (with 'Node Type' as "any") to represent the applicability of any element content and another sub-section (with 'Node Type' as "anyAttribute") to represent the applicability of any attribute content.

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (main)>
<!ELEMENT main ANY>
```

Here, the element "main" is declared with the keyword "ANY" which means that the "main" element can contain any type of content. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	main	Section	Element	
	Value	String	string	
	anyAttribute	Section	Element	
	any	Section	Element	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Designer Representation details

1. The element "main" is represented as a section with 'Node Type' as "element" and 'Compositor' as "sequence".
2. A sub-field "Value" with 'Node Type' as "value" is added to the "main" section to represent the applicability of text content.
3. A sub-section "anyAttribute" with 'Node Type' as "anyAttribute" is added to the "main" section to represent the applicability of any attribute content.
4. Another sub-section "any" with 'Node Type' as "any" is added to the "main" section to represent the applicability of any element content.
5. The occurrence constraints of both the sub-sections are set as Min Occurs = 0 & Max Occurs = Unbounded.

See Also:

[Element with Character Data only](#)
[Element with Standard Content](#)
[Element with Mixed Content](#)
[Element with No Content Allowed](#)

Element with No Content Allowed


An element with no content allowed is an empty element, which does not have, nor allows content. In DTD, you declare empty elements with the EMPTY keyword. A DTD component defining an element with empty content is represented in the Designer as a section without any sub-section or sub-field. However, an empty element can allow attributes, and in such cases, the attributes are represented as sub-fields of the section with 'Node Type' set as "attribute".

Consider the following DTD:


```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (br)>
```

<!ELEMENT br EMPTY>

Here, the “br” element is declared with EMPTY keyword meaning that it is an empty element. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 br	Section	Element	

Section Properties

XML Name	<input type="text" value="br"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	 element ▼	Compositor	sequence ▼

Designer Representation details

1. The element “br” is represented as an empty section with ‘Node Type’ as “element” and ‘Compositor’ as “sequence”.
2. The XML name of the element is displayed in the ‘XML Name’ text box.

See Also:

[Element with Character Data only](#)
[Element with Standard Content](#)
[Element with Mixed Content](#)
[Element with Any Content](#)

Group Declaration

When multiple child elements are specified, you use groups to combine them together. In DTD, you use open and close parentheses to represent a group. The elements added inside a group fall under two categories viz., sequence and choice, based on the order of their occurrence. In addition, the grouped sets can be qualified with ?, * and + suffixes to indicate the cardinality (occurrence constraints) of the group.

Generally, the group is flattened (Refer [“Group Flattening Rules”](#)) and its contents are aggregated to its parent and hence, a group has no explicit representation in the Designer and the order indicator functionality of the child elements enclosed in the group is represented by the ‘Compositor’ property of the parent element. However, in cases where “Flattening” is not possible, such as when the group is repeating, the

contents of the group cannot be aggregated and the group is represented as a section.

See Also:

[Sequence declaration](#)

[Choice declaration](#)

[Group Flattening Rules](#)

[Group represented as section](#)

Sequence Declaration

Sequence indicates that the child elements must occur in a specific order. In DTD, you use comma to represent a sequence, i.e., if you want the child elements to appear in a specific sequence, you separate them with commas.

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (contact_details)>
<!ELEMENT contact_details (account_id, name, phone, email)>
<!ELEMENT account_id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

In the above declaration, the “contact_details” element has multiple child elements and hence they are represented as a group (inside the () parentheses). Also, the child elements of the “contact_details” element are separated by commas which indicates that in the corresponding XML instance, the child elements should occur in the order in which they are specified i.e., “account_id” element followed by “name” element, followed by “phone” element and so on. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	contact_details	Section	Element	
	account_id	String	string	
	name	String	string	
	phone	String	string	
	email	String	string	

Section Properties

XML Name
Min Occurs
 ▼

Namespace
Max Occurs
 ▼

Node Type
 ▼
Compositor
 ▼

Designer Representation details

1. The group enclosing the child elements of the “contact_details” element is flattened (has no explicit Designer representation) since it is non-repeating, and its children are aggregated to the “contact_details” element and represented as sub-fields. (Root element has no explicit representation in the Designer)
2. The sequence functionality (represented by ‘commas’ in the DTD declaration) is pushed to the parent element “contact_details” and represented by the ‘sequence’ item set in ‘Compositor’ list box in the “Section Properties” pane.
3. Here, the occurrence constraints of the group and the parent element are identical (both being mandatory & non-repeating) and are represented by the ‘Min Occurs’ and ‘Max Occurs’ combo boxes in the “Section Properties” pane.
4. In the above e.g., suppose the group was declared as optional using the ? qualifier, i.e., if the declaration of “contact_details” element was as follows `<!ELEMENT contact_details (account_id, name, phone, email)?>` and, if you had enabled the “Flatten optional groups” option in “Schema Import Options” window during import, the corresponding Designer representation would be as under:

Field Name	Type	XML Type
contact_details	Section	Element
[?] account_id	String	string
[?] name	String	string
[?] phone	String	string
[?] email	String	string

Here, the occurrence constraints (Min Occurs = 0 & Max Occurs = 1) of the group are pushed to the child elements.

See Also:

[Choice declaration](#)





Choice Declaration

Choice indicates that only one child element must occur. In DTD, you use the pipe symbol | to represent a choice, i.e., if you want only one of the child elements to appear, you separate them with pipe symbol.


Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (contact_details)>
<!ELEMENT contact_details (phone | email | website)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT website (#PCDATA)>
```

In the above declaration, the child elements of the “contact_details” element are separated by pipe symbol, which indicates that only one among the child elements should occur in the corresponding XML instance. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 contact_details	Section	Element	
 phone	String	string	
 email	String	string	
 website	String	string	

Section Properties

XML Name	<input type="text" value="contact_details"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	 element ▼	Compositor	choice ▼

Designer Representation details

1. The group enclosing the child elements of the “contact_details” element is flattened (has no explicit Designer representation) since it is non-repeating, and its children are aggregated to the “contact_details” element and represented as sub-fields. (Root element has no explicit representation in the Designer)
2. The choice functionality (represented by ‘pipe symbol’ in the DTD declaration) is pushed to the parent element “contact_details” and represented by the ‘choice’ item set in ‘Compositor’ list box in the “Section Properties” pane. And also, all the

child elements of the "contact_details" element are made optional to represent the choice behavior in the Designer.

3. Here, the occurrence constraints of the group and the parent element are identical (both being mandatory & non-repeating) and are represented by the 'Min Occurs' and 'Max Occurs' combo boxes in the "Section Properties" pane.
4. In the above e.g., suppose the group was declared as optional using the ? qualifier, i.e., if the declaration of "contact_details" element was as follows
<!ELEMENT contact_details (phone | email | website)?>
5. and if you had enabled the "Flatten optional groups" option in "Schema Import Options" window during import, the corresponding Designer representation would be similar to the one above. On the other hand if you import using the default options (without specifying flattening option) the group would have an explicit representation as section in the Designer.

See Also:

[Sequence declaration](#)

[Group Flattening Rules](#)

[Group represented as section](#)

Group Flattening Rules

In the following cases, the group is flattened, i.e., the group has no explicit representation in the Designer and the contents and the order indicator of the child elements of the group are pushed to its parent.

Mandatory and Non-repeating group flattening

Where applicable

The following condition should be satisfied for this rule to be applied:

- The group is mandatory and non-repeating

Flattening details

- The parent element (section) inherits the order indicator (sequence, choice) of the child elements enclosed in the group.
- The group has no explicit Designer representation.
- The child elements enclosed in the group are moved to its parent element, i.e. the parent section.

Example

Consider the following DTD:

```

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (contact_details)>
<!ELEMENT contact_details (account_id, name)>
<!ELEMENT account_id (#PCDATA)>
<!ELEMENT name (#PCDATA)>

```

Here, the group (represented by open & close parentheses) is mandatory & non-repeating (no qualifier used) and the child elements enclosed in the group are separated by comma, meaning the “sequence” order indicator. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
contact_details	Section	Element	
account_id	String	string	
name	String	string	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Designer Representation details

1. The group is flattened, i.e., it does not have any explicit representation in the Designer.
2. The child elements enclosed in the group are moved to the parent element and represented as the child elements of the parent section “contact_details” and the order indicator (‘sequence’ in this case) of the child elements is inherited by the parent element. (Compositor property for the “contact_details” element is set as ‘sequence’).

Remarks

1. This rule is always applied.
2. In case the order indicator of the child elements enclosed in the group is choice, then the child elements are made optional. Refer “Optional behavior of Choice fields”.

Optional group flattening

Where applicable

The following condition should be satisfied for this rule to be applied:

- The group is optional and non-repeating
- The option “Flatten optional groups” is selected in the “Schema Import Options” window.

Flattening details




- The cardinality of the group is applied to the child elements enclosed in the group. Since the model group is optional, the child elements under it are invariably made optional.
- The group has no explicit Designer representation.
- The child elements enclosed in the group are moved to its parent element, i.e. the parent section.

Example


Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (contact_details)>
<!ELEMENT contact_details (email, website)?>
<!ELEMENT email (#PCDATA)>
<!ELEMENT website (#PCDATA)>
```

Here, the group (represented by open & close parentheses) is optional & non-repeating (qualifier ? used) and the child elements enclosed in the group are separated by comma, meaning the “sequence” order indicator. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
 contact_details	Section	Element	
 email	String	string	
 website	String	string	

Section Properties

XML Name	<input type="text" value="contact_details"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	 element ▼	Compositor	<input type="text" value="sequence"/> ▼

Designer Representation details

1. The group is flattened, i.e., it does not have any explicit representation in the Designer.
2. The child elements enclosed in the group are moved to the parent element and represented as the child elements of the parent section "contact_details" and the order indicator ('sequence' in this case) of the child elements is inherited by the parent element. (Compositor property for the "contact_details" element is set as 'sequence').
3. The optional behavior of the group is applied to the child elements viz., 'email' and 'website' enclosed in the group by making them optional.

Remarks

1. This rule is not applied by default. It is applied only when the option "Flatten optional groups" is selected in the "Schema Import Options" window. Else, the group will not be flattened.
2. There are both positive and negative effects in using this option. The positive effect is that the user can get rid of an extra section and thereby reduce the extra work during mapping. The negative effect is that some information is lost. In the above e.g., since the optional behavior of the group is applied to the child elements which are originally mandatory, the mandatory behavior of the child elements is lost.

Optional behavior of Choice fields

If the order indicator used for the elements in a group is a choice (pipe symbol used), then the child elements of the group are invariably made optional, whether the group is optional or not or whether it is flattened or not. This is due to conflict between the Designer and DTD in interpreting the meaning of the word 'optional'.

Example

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (contact_details)>
<!ELEMENT contact_details (phone | email)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Here, the group (represented by open & close parentheses) is mandatory & non-repeating (no qualifier used) and the child elements enclosed in the group are separated by pipe symbol, meaning the 'choice' order indicator. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	contact_details	Section	Element	
	phone	String	string	
	email	String	string	

Section Properties

XML Name	<input type="text" value="contact_details"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	element ▼	Compositor	choice ▼

Note that the child elements “phone” and “email” are made optional though they are mandatory as per the DTD.

See Also:

[Sequence declaration](#)

[Choice declaration](#)

[Group represented as section](#)

Group Represented as Section

In the following cases, the group is not flattened, i.e., the group is represented as a section.

1. Where the group has more than one child and is repeating.
2. Where the group has more than one child and is optional.

Example

Consider the following DTD where the above two conditions are satisfied:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (contact_details)>
<!ELEMENT contact_details (email, website)*>
<!ELEMENT email (email1 | email2)?>
<!ELEMENT email1 (#PCDATA)>
<!ELEMENT email2 (#PCDATA)>
<!ELEMENT website (#PCDATA)>
```

When this DTD is imported in the Designer with the default import options, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
contact_details	Section	Element	
contact_detailsGroup 1	Section	Element	
email	Section	Element	
emailGroup 1	Section	Element	
email1	String	string	
email2	String	string	
website	String	string	

Section Properties

XML Name
Min Occurs
 ▼

Namespace
Max Occurs
 ▼

Node Type
 ▼
Compositor
 ▼

Note that both the groups are not flattened and they are represented as a section.

See Also:

[Sequence declaration](#)

[Choice declaration](#)

[Group Flattening Rules](#)

DTD AttributeDeclaration

If you use DTD to describe XML structure, all attributes used in a valid XML document must be declared in that DTD. You define a list of attributes for an element with the ATTLIST assignment. You can assign multiple attributes to an element using a single assignment. Attribute declarations have an attribute name, an attribute type and a default usage. Attribute types fall into one of three categories: string types, tokenized types, and enumerated types. For e.g., CDATA is a string type while ENUMERATION is an enumerated type. For string and tokenized attributes, default usages can be set with the keywords #IMPLIED, #FIXED and #REQUIRED which are described in the following table:

Keyword	Attribute Usage	Default Value
#IMPLIED	Optional attributes	No default value is assigned or allowed
#FIXED	Fixed-value attributes	Default value is assigned as provided and necessary
#REQUIRED	Mandatory attributes	No default value is assigned or

		allowed.
-	Literal value attributes	Default value is assigned as provided and necessary.

Default attribute values apply when attribute are missing in the related document.

When a DTD is imported, attributes are always aggregated to the element for which they are declared and are represented as fields in the Designer. Refer ["Aggregation"](#).

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (purchase_order)>
<!ELEMENT purchase_order (date, item)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!--ATTLIST purchase_order cust_rep CDATA #IMPLIED
shipping CDATA #REQUIRED
prepaid CDATA #FIXED "yes"
store NMTOKEN "Seattle"
-->
```

In the above DTD, the purchase_order element has four attributes added to it. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

Field Name	Type	XML Type	Description
purchase_order	Section	Element	
cust_rep	String	string	
shipping	String	string	
prepaid	String	string	
store	String	string	
date	String	string	
item	String	string	

Field Properties

XML Name
Required ☐

Namespace
Default Value

Node Type attribute ▼

Facets

Designer Representation details

1. All the attributes are aggregated to the "purchase_order" element and are represented as its sub-fields with 'Node Type' set as "attribute" in the "Field Properties" pane.
2. The mandatory / optional behavior of the attribute is represented by the checking / unchecking of the 'Required' check box in the "Field Properties" pane. The attribute "cust_rep" is optional; hence "Required" check box would be unchecked for it while for the mandatory attribute "shipping" the "Required" check box would be checked.
3. The default value "yes" set for the Fixed attribute "prepaid" is represented by the value set in the "Default Value" text box in the "Field Properties" pane.
4. The default value "Seattle" set for the literal value attribute "store" is also similarly represented.

See Also:

[Enumeration type](#)

[Ignore Fixed Attributes](#)

[DTD Element declaration](#)

[Group declaration](#)

Enumeration Type

For enumerated type attribute, the list of possible values specified in the DTD is represented in the Designer in the table under the "Enumeration" tab of the "Facets" window, which is opened by clicking the "Facets" button in the "Field Properties" pane.

Consider the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Root (purchase_order)>
<!ELEMENT purchase_order (date, item)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ATTLIST purchase_order type (walkin | phone | catalog) "walkin"
>
```

In the above DTD, the purchase_order element has an enumeration attribute "type". The type attribute has three name tokens that define the valid values for the attribute. The default value "walkin" is assigned as a literal value enclosed in quotation marks. When this DTD is imported in the Designer, the corresponding Designer representation would be as under:

	Field Name	Type	XML Type	Description
	purchase_order	Section	Element	
	type	String	string	
	date	String	string	
	item	String	string	

Field Properties

XML Name
Required ☐

Namespace
Default Value

Node Type attribute ▼

The default value is represented in the “Default Value” text box in the “Field Properties” pane. To view the list of enumerations, you should click the “Facets” button, which would open the “Facets” window as seen below:

Facets

Facets
Pattern
Enumeration

+
-
↑
↓

Enumeration	Description
walkin	
phone	
catalog	

Note that the list mentioned in the DTD finds place in the table in the “Enumeration” tab.

See Also:

[Ignore Fixed Attributes](#)

Ignore Fixed Attributes (DTD AttributeDeclaration)

“Ignore Fixed Attributes” option is provided in the [“Schema Import Options”](#) window. When this option is checked, the attributes for which usage is specified as #FIXED will not be imported in the Designer during DTD import. This is particularly useful in cases where fixed attributes are used to provide meta-info or additional information about the elements in the schema. In the first e.g. given above, for the “prepaid” attribute, usage is set as #FIXED. When the DTD in the first example under the chapter “Attribute declaration” given above is imported in the Designer with the

option “Ignore Fixed Attributes” selected, the equivalent Designer representation would be as under:

Field Name	Type	XML Type
purchase_order	Section	Element
cust_rep	String	string
shipping	String	string
store	String	string
date	String	string
item	String	string

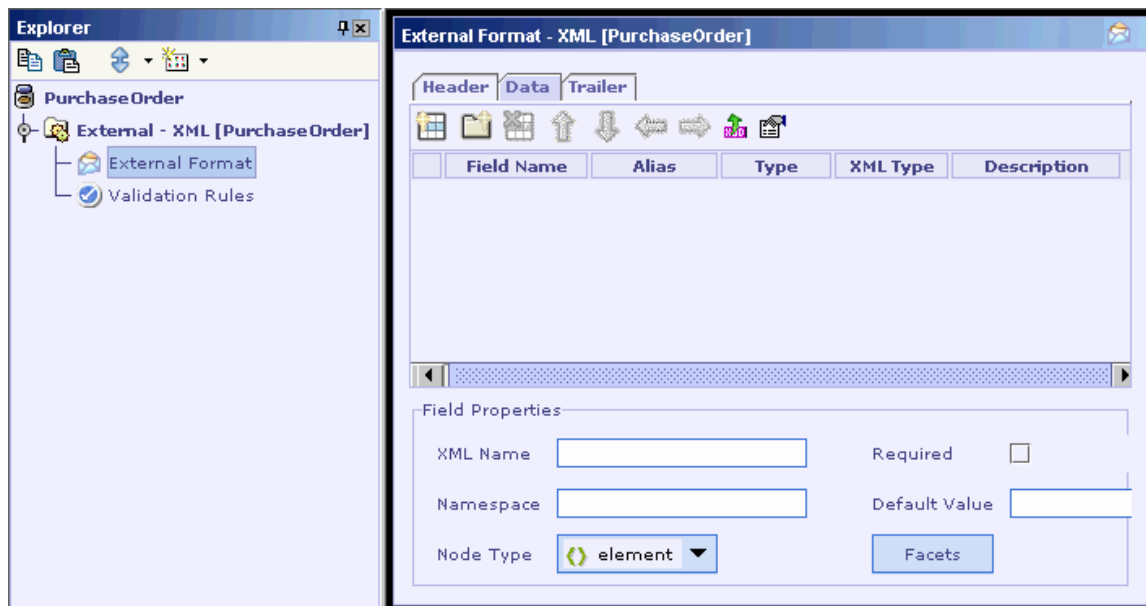
Note that the “prepaid” attribute is not imported in this case.

See Also:

[Enumeration type](#)

Designing XML Structure Manually in Designer

When you select the “Synthesize DTD” option in the [“Schema Option” window](#) and select the required options in the subsequent windows of the creation wizard (Refer [“iii\) Synthesize DTD”](#)), an empty XML message format is created as shown below:



In the Explorer window, the newly created XML Message format is displayed with two child nodes viz., External Format and Validation Rules. The External Format User Interface has three segments viz., Header, Data and Trailer, where you design your

XML message structure by adding fields / sections and setting properties for the same. The tool bar that lets you add field / section and set format options is shown under:



Tool Bar Icon	Function
	Adds a new field
	Adds a new section
	Removes a field / section
	Moves the selection field / section up by one step
	Moves the selection field / section down by one step
	Moves the selected field / section left
	Moves the selected field / section right
	Lets you reconfigure the schema
	Lets you specify various format options

To add an XML element with different types of content and XML attribute, you have to add a field / section in the Designer and set the "Node Type" and "Compositor" in the "Properties" pane accordingly. The following sections explain the possible XML component representations of a Field and a Section.

See Also:

[Field representation](#)

[Section representation](#)

[Adding Element](#)

[Adding Attribute](#)

[Adding character content](#)

[Adding Wildcard](#)

[Adding Group](#)

Field Representation

A Designer field can represent an XML element / attribute / character data based on the selected “Node Type”


Node Type of the Field	XML component
element	Non-repeating element with character data or text
attribute	Attribute
Value	character data / text of mixed content element

When a field is selected in the UI, the “[Field Properties](#)” pane is displayed, which is described hereunder:

See Also:

[Section representation](#)

Field Properties pane

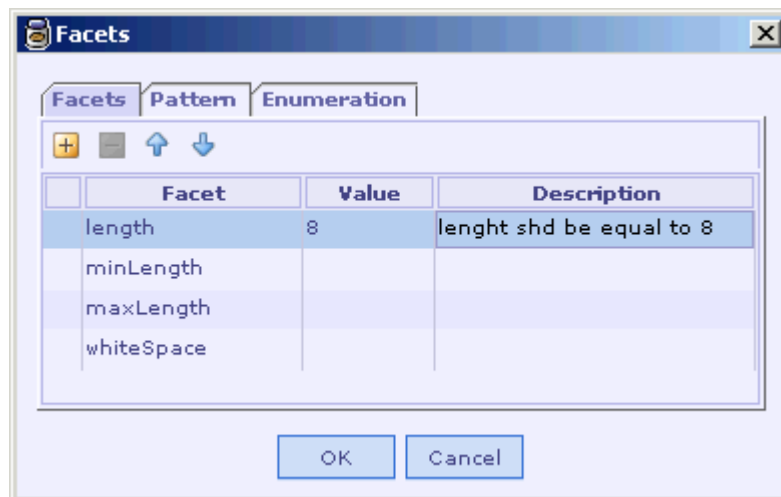


- **XML Name** text box is used to specify the required name of the XML element / attribute. If no value is specified here, the Designer name is taken as the XML name.
- **Namespace** text box is used to specify the namespace for namespace qualified element / attribute.
- **Node Type** list box is used to specify the required XML representation of the selected field. There are three items in the list viz., element, attribute and value each representing different XML components as detailed in the table under [“Field representation”](#).





- **Required** check box is used to specify the optional / mandatory behavior of an element / attribute.
- **Default Value** text-box is used to specify the default / fixed value of the XML element / attribute.
- **Facets** button, when clicked, opens the “Facets” window, where you can set the required facet for the element. The window has three tabs as explained under:

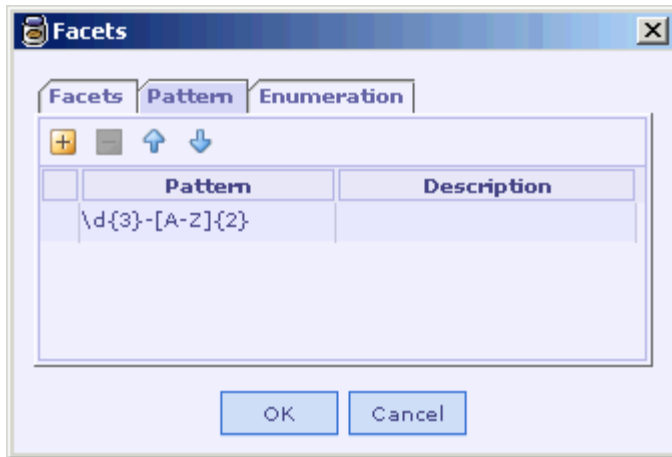
‘Facets’ tab

Under the ‘Facets’ tab, all applicable facets (except “pattern” and “enumeration”) for the XML type of a selected element or attribute are listed under the ‘Facet’ column. You can set the required value under the respective ‘Value’ column and also specify description if required under the ‘Description’ column.



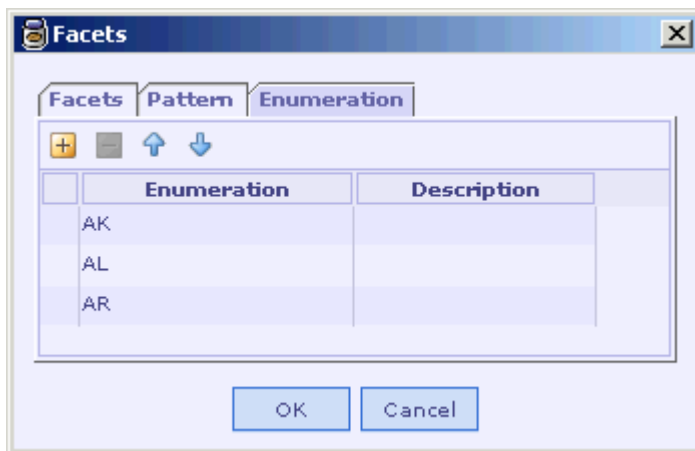
‘Pattern’ tab

Under the ‘Pattern’ tab, pattern facet for the XML Type of the selected element / attribute (if applicable) can be specified under the ‘Pattern’ column and description if required can be specified under the ‘Description’ column. Click the ‘Add’ button  available in the tool bar to add a ‘pattern’ facet. You can remove an added facet using the ‘Remove’ button  which is enabled when a facet is added. You can move the added facets in the table one step up and down using the “Move Selection Up”  and “Move Selection Down”  buttons.



'Enumeration' tab

Under the 'Enumeration' tab, enumeration facet for the XML Type of the selected element / attribute (if applicable) can be specified under the 'Enumeration' column and description if required can be specified under the 'Description' column. Use the buttons available in the tool bar to add / remove / move up & down the added enumeration facets.



See Also:

[Field representation](#)

Section Representation

A Designer section can represent an XML element / group / anyElement / anyAttribute based on the selected "Node Type"

Node Type of the Section	XML component
element	Element with sub-elements, with & without attributes / Element with text & attribute / Repeating element
group	group (sequence / choice / all)
Any	any Element
anyAttribute	any Attribute

When a section is selected in the UI, the “[Section Properties](#)” pane is displayed, which is described hereunder:

See Also:

[Field representation](#)

[Section Properties pane](#)

Section Properties pane

The screenshot shows the 'Section Properties' pane with the following settings:

- XML Name:** item
- Namespace:** (empty)
- Node Type:** element (selected from a dropdown menu)
- Min Occurs:** 1 (selected from a dropdown menu)
- Max Occurs:** Unbounded (selected from a dropdown menu)
- Compositor:** sequence (selected from a dropdown menu)

- **XML Name** text box is used to specify the required name of the XML element. If no value is specified here, the Designer name is taken as the XML name.
- **Namespace** text box is used to specify the namespace for namespace qualified element.
- **Node Type** list box is used to specify the required XML representation of the selected section. There are four items in the list viz., element, group, any and anyAttribute each representing different XML components as detailed in the table under “[Section representation](#)”.
- **Min Occurs, Max Occurs** combo boxes are used to specify the occurrence constraints for the XML element.

- **Compositor** list box is used to specify the compositor which determines the order of occurrence of child elements of an XML element. There are three elements in this list box viz., sequence, choice and all.

sequence

Indicates that the child elements should appear in sequential order as specified.

choice

Indicates that only one of the child elements should appear.

all

Indicates that the child elements may appear in any order.

We have seen what a field and a section added in the Designer mean in the XML's perspective. Now let us see how to design a XML message structure containing various XML components in the Designer.

See Also:

[Section Representation](#)

Adding Element

To add an element, you have to add a field / section based on the element content and its occurrence constraints. An element can be of [simple](#) / [complex](#) type and can be repeating / non-repeating.

See Also:

[Adding Attribute](#)

[Adding character content](#)

[Adding Wildcard](#)

[Adding Group](#)

Simple Type

An element is said to be of simple type if it can contain only character data and no sub-elements or attributes. A simple type element can be [repeating](#) or [non-repeating](#). In the Designer, to add a simple type element, you have to add a field / section based on its occurrence. If the element is non-repeating, you have to add a field; if it is repeating, you have to add a section.

See Also:

[Complex Type](#)


Non-repeating element

To add a non-repeating simple type element, follow the steps detailed below:


1. Add a field by clicking the “Add New Field” icon & specify the field name.
2. In the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	element
XML Name text box	required XML name for the element
Required check box	Mandatory / Optional behavior of the element
Namespace text box	Namespace for the element, if required
Default Value text box	Default value for the element, if required

3. In the XML Type column, specify the required data type for the element.
4. If you want to set facets, click the “Facets” button in the “Field Properties” pane to open the “Facets” window and specify required facets for that element.
5. If required, specify description about the field in the “Description” column.

Field Name	Type	XML Type	Description
 name	String	string	

Field Properties

XML Name	<input type="text" value="name"/>	Required	<input checked="" type="checkbox"/>
Namespace	<input type="text"/>	Default Value	<input type="text"/>
Node Type	 element ▼	<input type="button" value="Facets"/>	

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (name)>
<!ELEMENT name (#PCDATA)>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="name" type="xs:string"/>
```

See Also:

[Simple Type](#)

[Repeating element](#)

Repeating element

To add a repeating simple type element, follow the steps detailed below:

1. Add a section by clicking the “Add New Section” icon & specify the section name.
2. In the “Section Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	element
XML Name text box	required XML name for the element
Namespace text box	Namespace for the element, if required
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the element
Compositor list box	sequence

3. Add a sub-field under the above section, specify a name (typically named as “Value” as it represents the value of the XML element) and in the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	value
XML Name text box	Not Applicable
Required check box	Checked
Namespace text box	Not Applicable
Default Value text box	Default value for the element, if required

4. In the XML Type column for the above added field, specify the required data type for the element.
5. If you want to set facets for the element, click the "Facets" button in the "Field Properties" pane to open the "Facets" window and specify required facets for that element.
6. If required, specify description about the section & the sub-field in the "Description" column.

	Field Name	Type	XML Type	Description
	name	String	string	
	comment	Section	Element	
	Abc Value	String	string	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Equivalent DTD

The relevant DTD equivalent (See "comment" element) for the above structure is:

```
<!ELEMENT Root (name, comment*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
```

See Also:

[Adding Element](#)

Complex Type

An element is said to be of complex type if it can contain sub-elements, attributes as well as character data. The content of a Complex Type can be [simple](#) / [complex](#). A complex type element is typically represented as a section in the Designer except in one special case (Refer ["Character Data Only"](#)).

See Also:

[Simple Type](#)

Simple Content (Complex Type)

An element of complex type with simple content (Text-only content) can have just character data or character data and attributes, but it cannot have sub-elements.


Character Data Only

To add an element of complex type with simple content that has just character data and no other content, you have to add a field / section based on the occurrence constraints of the element. If the element is non-repeating, you have to add a field and if it is repeating, you have to add a section. The non-repeating option is the only case, where an element of complex type is represented as a field. (Note that in XSD, an element with just character data can be of simple type or complex type, whereas in DTD, there is no separate type definition and an element with just character data has only one possible representation).

Non-repeating Element

To add a non-repeating element with just character data, follow the steps explained in the section ["Non-repeating element"](#) under "Simple Type".

Designer representation of a non-repeating complex type element with just character data.

	Field Name	Type	XML Type	Description
	internationalPrice	String	string	

Field Properties


XML Name

Required
☒

Namespace

Default Value

Node Type


element
▼

Facets

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (internationalPrice)>
<!ELEMENT internationalPrice (#PCDATA)>
```

Equivalent XSD


The relevant XSD equivalent for the above structure is:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Repeating Element

To add a repeating element with just character data, follow the steps explained in the section ["Repeating Element"](#) under "Simple Type".

Designer representation of a repeating complex type element with just character data.

Field Name	Type	XML Type	Description
 internationalPrice	Section	Element	
Abc Value	String	string	

Section Properties


XML Name

Min Occurs
 ▼

Namespace

Max Occurs
 ▼

Node Type


element
▼

Compositor
 ▼

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (internationalPrice)* >
<!ELEMENT internationalPrice (#PCDATA)>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="internationalPrice" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Character Data and Attributes

To add an element of complex type with simple content that has character data as well as attributes, you have to invariably add a section whether the element is repeating or non-repeating and add sub-fields to represent the attributes and character data. To add such an element follow the steps detailed below:

1. Add a section by clicking the "Add New Section" icon & specify the section name.
2. In the "Section Properties" pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	element

XML Name text box	required XML name for the element
Namespace text box	Namespace for the element, if required
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the element, if required
Compositor list box	sequence

3. To add attribute, add a sub-field under the above section, specify a name, and in the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	attribute
XML Name text box	required XML name for the attribute
Required check box	Mandatory / Optional behavior of the attribute
Namespace text box	Namespace for the attribute, if required
Default Value text box	Default value for the attribute, if required

4. In the XML Type column for the above added attribute field, specify the required data type for the attribute.
5. If you want to set facets for the attribute, click the “Facets” button in the “Field Properties” pane to open the “Facets” window and specify required facets for that attribute.
6. To represent the character data, again add a sub-field under the above section, specify a name (typically named as “Value” as it represents the value of the XML element) and in the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	value

XML Name text box	Not Applicable
Required check box	Checked
Namespace text box	Not Applicable
Default Value text box	Default value for the element, if required

7. In the XML Type column for the above added field, specify the required data type for the element.
8. If you want to set facets for the element, click the "Facets" button in the "Field Properties" pane to open the "Facets" window and specify required facets for that element.
9. If required, specify description about the element & attribute in the corresponding "Description" column.

	Field Name	Type	XML Type	Description
	name	String	string	
	comment	Section	Element	
	Abc Value	String	string	
	internationalPrice	Section	Element	
	currency	String	string	
	Abc Value	BigDecimal	decimal	

Section Properties

XML Name
Min Occurs

Namespace
Max Occurs

Node Type
Compositor

Equivalent DTD

The relevant DTD equivalent (see "internationalPrice" element) for the above structure is:

```
<!ELEMENT Root (name, comment*, internationalPrice)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT internationalPrice (#PCDATA)>
<!ATTLIST internationalPrice currency CDATA #IMPLIED >
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="currency" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

See Also:

[Complex Type](#)

Complex Content (Complex Type)

An element of complex type with complex content can have sub elements, attributes as well as character data. Based on the content, an element of complex content can be further categorized as element having standard content, mixed content, any content or empty content.

Standard Content

Complex Types with standard content have only elements or elements and attributes associated with them. To add such an element in the Designer, you have to add a section to represent the parent element, sub-fields / sub-sections to represent the child elements and sub-fields (with Node Type "Attribute") to represent the attributes.

To add an element of standard content that has both child elements and attributes, follow the steps detailed below:

1. Add a section by clicking the "Add New Section" icon & specify the section name.
2. In the "Section Properties" pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	element
XML Name text box	required XML name for the

	element
Namespace text box	Namespace for the element, if required
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the element, if required
Compositor list box	sequence

For each attribute, add a sub-field under the above added section, specify a name and set the required properties in the “Field Properties” pane. Refer the [table](#) in the previous section.

To add the child elements, add a sub-field or sub-section depending on whether the child element is repeating or non-repeating and also based on its content. Refer the sections explained above to determine whether to add a field or section to represent the child element. In the following case, fields (with Node Type as “element”) are added to represent the child elements.

	Field Name	Type	XML Type	Description
	item	Section	Element	
	weightKg	BigDecimal	decimal	
	partNum	String	string	
	productName	String	string	
	quantity	BigInteger	positiveInteger	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```

<!ELEMENT Root (item)>
<!ELEMENT item (productName, quantity)>
<!ELEMENT productName (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ATTLIST item weightKg CDATA #IMPLIED
              partNum CDATA #REQUIRED>

```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="productName" type="xs:string"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
    </xs:sequence>
    <xs:attribute name="partNum" type="xs:string" use="required"/>
    <xs:attribute name="weightKg" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
```

Mixed Content

Complex Types with mixed content have a combination of elements and text associated with them and may contain attribute definitions. To add such an element in the Designer, you have to add a section to represent the parent element, sub-fields (with Node Type as “element”) / sub-sections to represent the child elements, sub-fields (with Node Type as “attribute”) to represent the attributes if any, and another sub-field (with node type as “value”) to represent the text content.

To add an element with mixed content without any attributes, follow the steps detailed below:

1. Add a section by clicking the “Add New Section” icon & specify the section name.
2. In the “Section Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	element
XML Name text box	required XML name for the element
Namespace text box	Namespace for the element, if required
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the element, if required





Compositor list box	sequence
---------------------	----------

To represent the text content of the element, add a sub-field under the above section, specify a name and in the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	value
XML Name text box	Not Applicable
Required check box	Checked
Namespace text box	Not Applicable
Default Value text box	Not Applicable

In the XML Type column for the above added field, select “string”.

To add the child elements, add a sub-field or sub-section depending on whether the child element is repeating or non-repeating and also based on its content. Refer the sections explained above to determine whether to add a field or section to represent the child element. In the following case, a section (with Node Type as “element”) is added to represent the repeating “item” child element.

Field Name	Type	XML Type	Description
 inventory	Section	Element	
 Value	String	string	
 item	Section	Element	
 Value	String	string	

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (inventory)>
<!ELEMENT inventory (#PCDATA | item)*>
<!ELEMENT item (#PCDATA)>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```

<xs:element name="inventory">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="item" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Any Content

Complex Types with any content are unconstrained, i.e., they can contain any type of content say, text, sub-element, attribute, etc. To add such an element in the Designer, you have to add a section to represent the element, a sub-field (with Node Type as “value”) to represent the applicability of text content, a sub-section (with Node Type as “any”) to represent the applicability of child elements and another sub-section (with Node Type as “anyAttribute”) to represent the applicability of attributes.

To add an element with any content, follow the steps detailed below:

1. Add a section by clicking the “Add New Section” icon & specify the section name.
2. In the “Section Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	Element
XML Name text box	required XML name for the element
Namespace text box	Namespace for the element, if required
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the element, if required
Compositor list box	Sequence

To represent the applicability of text content for the element, add a sub-field under the above section, specify a name and in the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	Value
XML Name text box	Not Applicable
Required check box	Unchecked
Namespace text box	Not Applicable
Default Value text box	If required

To represent the applicability of any attribute content for the element, add a sub-section under the above section, specify a name and in the “Section Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	anyAttribute
XML Name text box	required XML name for the anyAttribute
Namespace text box	required Namespace for the anyAttribute
Min Occurs, Max Occurs combo boxes	Min Occurs = 0 Max Occurs = Unbounded
Compositor list box	sequence

To represent the applicability of any sub-element content for the element, add another sub-section under the above section, specify a name and in the “Section Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	any
XML Name text box	required XML name for the any element
Namespace text box	required Namespace for the any element

Min Occurs, Max Occurs combo boxes	Min Occurs = 0 Max Occurs = Unbounded
Compositor list box	sequence

Field Name	Type	XML Type	Description
item	Section	Element	
weightKg	BigDecimal	decimal	
partNum	String	string	
productName	String	string	
quantity	BigInteger	positiveInteger	

Section Properties

XML Name	<input type="text" value="item"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	element ▼	Compositor	sequence ▼

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (main)>
<!ELEMENT main ANY>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="main" type="xs:anyType"/>
```

Empty Content

Complex Types with no content associated with them are empty, meaning they don't have, nor do they allow, content. They can, however, have attributes. To add such an element in the Designer, you have to add a section to represent the element and sub-fields (with Node Type as "attribute") to represent the attributes if any.

To add an element with empty content, follow the steps detailed below:

1. Add a section by clicking the "Add New Section" icon & specify the section name.

2. In the “Section Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	element
XML Name text box	required XML name for the element
Namespace text box	Namespace for the element, if required
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the element, if required
Compositor list box	sequence

If you want the element to contain attributes, add the same by adding sub-fields under the above added section with Node Type as “attribute” and setting the required properties. Refer the [table](#) given earlier for details. In the following case, we have an empty element, which allows attributes.

Field Name	Type	XML Type	Description
internationalPrice	Section	Element	
currency	String	string	
value	String	string	

Section Properties

XML Name
Min Occurs ▼

Namespace
Max Occurs ▼

Node Type element ▼
Compositor ▼

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```

<!ELEMENT Root (internationalPrice)>
<!ELEMENT internationalPrice EMPTY>
<!ATTLIST internationalPrice currency CDATA #IMPLIED
                        value CDATA #IMPLIED>

```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="internationalPrice">
  <xs:complexType>
    <xs:attribute name="currency" type="xs:string"/>
    <xs:attribute name="value" type="xs:decimal"/>
  </xs:complexType>
</xs:element>
```

See Also:

[Complex Type](#)



Adding Attribute

To add an Attribute in the Designer, you have to add a field and set the Node Type as "attribute".

To add an attribute, follow the steps detailed below:

1. Add a field by clicking the "Add New Field" icon & specify the field name.
2. In the "Field Properties" pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	attribute
XML Name text box	required XML name for the attribute
Required check box	Mandatory / Optional behavior of the attribute
Namespace text box	Namespace for the attribute, if required
Default Value text box	Default value for the attribute, if required

	Field Name	Type	XML Type	Description
	item	Section	Element	
	partNum	String	string	

Field Properties


XML Name

Required
☒

Namespace

Default Value

Node Type


attribute

Facets

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (item)>
<!ELEMENT item EMPTY>
<!ATTLIST item partNum CDATA #REQUIRED >
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="item">
  <xs:complexType>
    <xs:attribute name="partNum" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Note:

To add an "attributeGroup", which is nothing but a list of attributes, just add each attribute in the "attributeGroup" following the steps detailed above.

See Also:

[Adding Element](#)
[Adding character content](#)
[Adding Wildcard](#)
[Adding Group](#)

Adding character content

To represent the character data content of a repeating simple type element or a complex type element where character data is applicable, follow the steps detailed below:

1. Under the parent element (section), add a sub-field by clicking the “Add New Field” icon & specify the field name.
2. In the “Field Properties” pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	value
XML Name text box	Not Applicable
Required check box	Checked
Namespace text box	Not Applicable
Default Value text box	Default value for the element (simple type), if required

3. In case of repeating simple type element, if you want to set facets for the element, click the “Facets” button in the “Field Properties” pane to open the “Facets” window and specify required facets for that element.

Field Name	Type	XML Type	Description
comment	Section	Element	
Abc Value	String	string	

Field Properties

XML Name
Required ☒

Namespace
Default Value

Node Type Abc value ▼

Facets

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (comment+)>
<!ELEMENT comment (#PCDATA)>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

<xs:element name="comment" type="xs:string" maxOccurs="unbounded"/>

See Also:

[Adding Element](#)
[Adding Attribute](#)
[Adding Wildcard](#)
[Adding Group](#)

Adding Wildcard

A wildcard provides for validation of attribute and element information items dependent on their namespace name, but independently of their local name. This concept is relevant to XSD. The two wildcard schema components are "[any](#)" and "[anyAttribute](#)" elements.

See Also:

[Adding Element](#)
[Adding Attribute](#)
[Adding character content](#)
[Adding Group](#)

any element

In general, an "any" element specifies that any well-formed XML is permissible in a type's content model. It enables element content according to namespaces. To add an "any" element in the Designer, you have to add a section with "Node Type" as "any".

To add an "any" element, follow the steps detailed below:

1. Add a section by clicking the "Add New Section" icon & specify the section name.
2. In the "Section Properties" pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	any
XML Name text box	any
Namespace text box	value of "namespace" attribute of the "any" element (##any, ##other, etc.)

Min Occurs, Max Occurs combo boxes	Occurrence constraints for the "any" element, if required
Compositor list box	sequence

It is pertinent to note here that unlike the previous cases, the value specified in the "Namespace" text box for an "any" element does not represent the namespace of the "any" element, but it represents the value of the "namespace" attribute of the "any" element.

Field Name	Type	XML Type	Description
any	Section	Element	

Section Properties

XML Name	<input type="text" value="any"/>	Min Occurs	<input type="text" value="1"/> ▼
Namespace	<input type="text" value="##any"/>	Max Occurs	<input type="text" value="1"/> ▼
Node Type	<input type="text" value="any"/> ▼	Compositor	<input type="text" value="sequence"/> ▼

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:any namespace="##any"/>
```

See Also:

[anyAttribute element](#)

anyAttribute element



"anyAttribute" element enables attributes to appear in elements. In contrast to an "any" element, "anyAttribute" cannot constrain the number of attributes that may appear in an element. To add an "anyAttribute" element in the Designer, you have to add a section with "Node Type" as "anyAttribute".

To add an "anyAttribute" element, follow the steps detailed below:

1. Under the required parent element, "Elm" in this case (added as section), add a sub-section by clicking the "Add New Section" icon & specify the sub-section name.
2. In the "Section Properties" pane, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	anyAttribute
XML Name text box	anyAttribute
Namespace text box	value of "namespace" attribute of the "anyAttribute" element (##any, ##local, ##other, etc.)
Min Occurs, Max Occurs combo boxes	Min Occurs = 0 Max Occurs = Unbounded (because, no. of attributes can't be constrained)
Compositor list box	sequence

Here also, as in the case of "any" element, the value specified in the "Namespace" text box for the "anyAttribute" element does not represent the namespace of the "anyAttribute" element, but it represents the value of the "namespace" attribute of the "anyAttribute" element.

Field Name	Type	XML Type	Description
 Elm	Section	Element	
 anyAttribute	Section	Element	

Section Properties

XML Name	<input type="text" value="anyAttribute"/>	Min Occurs	<input type="text" value="0"/> ▼
Namespace	<input type="text" value="##any"/>	Max Occurs	<input type="text" value="Unbounded"/> ▼
Node Type	<input type="text" value="anyAttribute"/> ▼	Compositor	<input type="text" value="sequence"/> ▼

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:anyAttribute namespace = "##any"/>
```

See Also:

[any element](#)

Adding Group

To determine the order of occurrence of child elements of an XML element, you may add a group (sequence / choice / all). (In XSD, two types of groups, viz., named (group) and anonymous are available; in DTD, open and close parentheses () represent a group, a comma between two elements indicates “sequence” while the pipe symbol indicates “choice” and the concept of “all” is not applicable for DTD). In the Designer, you have two options to represent a group’s functionality:

- i) You can explicitly add a section under the parent element to represent the group and set the compositor;
- ii) You can make the parent element itself determine the order of occurrence of its children by setting the “Compositor” property for the same.

The latter option (called [“Flattening”](#)), has its own pros and cons; the advantage being, you can reduce the complexity of the message structure; the disadvantage being, there can be loss of information and Flattening is not applicable in some cases.

To explicitly add a group (Option (i) discussed above), follow the steps detailed below:

- Under the parent element (section) where you want to add a group, add a sub-section by clicking the “Add New Section” icon & specify the required section name. Note that this sub-section will have no representation in the corresponding XML instance, as it is not an XML element.
- In the “Section Properties” pane for the added sub-section, specify the properties as shown in the following table:

Property Name	Value to be specified
Node Type list box	group
XML Name text box	Not Applicable
Namespace text box	Not Applicable
Min Occurs, Max Occurs combo boxes	Occurrence constraints for the group
Compositor list box	required compositor (sequence, choice or all)

- Under the above sub-section, add the child elements of the parent element by adding a field or section depending on whether the child element is repeating or

non-repeating and also based on its content. Refer the sections explained above to determine whether to add a field or section to represent the child element. In the following case, fields (with Node Type as “element”) are added to represent the non-repeating child elements with character data.

- If the Compositor of the group is “choice”, make all the child elements as optional to represent the “choice” behavior in the Designer.

Field Name	Type	XML Type	Description
ContactDetails	Section	Element	
account_id	String	string	
extra_info	Section	Element	
extra_infoGroup 1	Section	Element	
fax_no	BigInteger	positiveInteger	
email_id	String	string	
phone_no	BigInteger	positiveInteger	

Section Properties

XML Name
Min Occurs

Namespace
Max Occurs

Node Type

group

Compositor

choice

Equivalent DTD

The relevant DTD equivalent for the above structure is:

```
<!ELEMENT Root (ContactDetails)>
<!ELEMENT ContactDetails (account_id, extra_info, phone_no) >
<!ELEMENT account_id (#PCDATA)>
<!ELEMENT extra_info (fax_no | email_id)?>
<!ELEMENT fax_no (#PCDATA)>
<!ELEMENT email_id (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Equivalent XSD

The relevant XSD equivalent for the above structure is:

```
<xs:element name="ContactDetails">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="account_id" type="xs:string"/>
      <xs:element name="extra_info">
        <xs:complexType>
```

```

<xs:choice minOccurs="0">
  <xs:element name="fax_no" type="xs:positiveInteger"/>
  <xs:element name="email_id" type="xs:string"/>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="phone_no" type="xs:positiveInteger"/>
</xs:sequence>
</xs:complexType>
</xs:element>

```

Note:

To represent the group's functionality by the element itself (Option (ii) discussed above), just change the compositor of the "extra_info" element to "choice" and make the child elements optional.

Field Name	Type	XML Type	Description
ContactDetails	Section	Element	
account_id	String	string	
extra_info	Section	Element	
fax_no	BigInteger	positiveInteger	
email_id	String	string	
phone_no	BigInteger	positiveInteger	

Section Properties

XML Name

Min Occurs
▼

Namespace

Max Occurs
▼

Node Type
 element ▼


Compositor
choice ▼

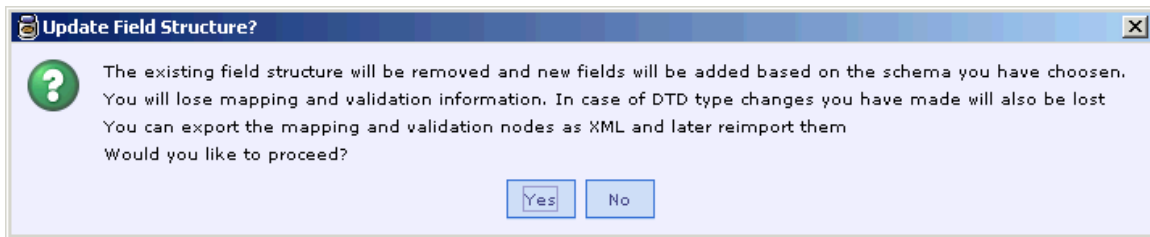
However, in this case, the information regarding the optional behavior of the choice group is lost, as the group is not explicitly added.

See Also:

- [Adding Element](#)
- [Adding Attribute](#)
- [Adding character content](#)
- [Adding Wildcard](#)

Reconfigure Schema

If your schema has changed after you last imported, you can re-import the schema by using the functionality of the 'Reconfigure Schema'  button in the 'External Format' UI toolbar. When you click this button, the ["Schema Option" window](#) of the [creation wizard](#) would be opened which guides you through the various steps of the XML message creation process. In the last window of the creation wizard viz., "Finish", when you click the "Finish" button, the "Update Field Structure" confirmation window would be opened:



If you click the 'Yes' button in the above window, the existing message structure would be overwritten based on the newly selected schema. If you click the 'No' button, there won't be any change to the existing message structure.

Note:

If you reconfigure the schema of a XML external message, the mapping information corresponding to this external message would be lost. Also, in case of DTD, the type changes if any, made manually for the elements / attributes in the imported XML message structure would be lost.


See Also:

[Creating an XML Format](#)

[Binding XML Schema to Designer](#)

Format Option

The Designer gives information about the imported XML message structure and also allows the user to customize parsing and writing options by providing various options in the "XML Format Option" window. This setting is optional and is just for convenience. You can later change the properties as per your choice.

Click on the 'Format Options' button  in the 'External Format' UI toolbar to bring up the 'XML Format Option' dialog box that has five tabs as described below.

See Also:

[General](#)

[Namespaces](#)

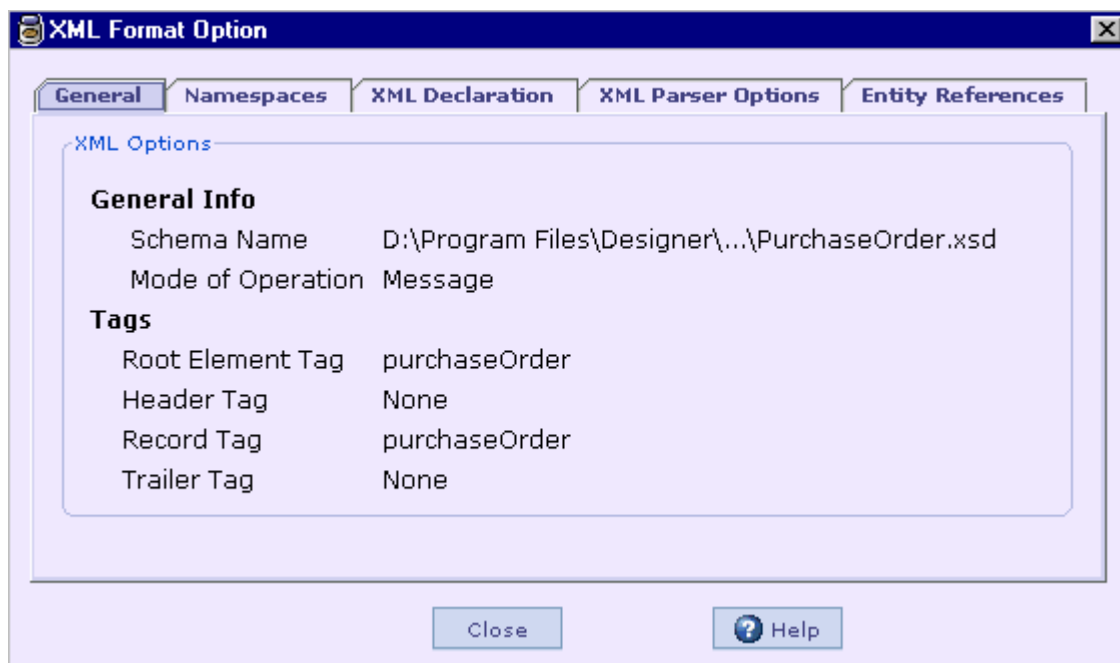
[XML Declaration](#)

[XML Parser Options](#)

[Entity References](#)

General

Under this tab, information about the XML message structure such as the name of the imported schema and the options such as mode of operation, name of root / header / record / trailer elements specified in the windows of the creation wizard are listed. The information displayed under this tab is not editable.



General Info

1. Schema Name
2. Here, the name of the schema (XSD / DTD) with full path, which is imported to get the current XML message structure in the Designer, is displayed. In case of "Synthesize DTD" option selected in the "Schema Option" window, it will read as "Synthesized" meaning that the schema (DTD) is synthesized.

3. Mode Of Operation

Here, the mode of operation (Batch / Message) selected in the ["Mode Of Operation"](#) window of the creation wizard is displayed.

Tags

Here, the root element name specified in the [“Root Element”](#) window of the creation wizard and the Header / Trailer / Record element names specified in the [“Header & Trailer”](#) / [“Batch Mode”](#) window of the creation wizard are displayed.

See Also:

[Namespaces](#)

[XML Declaration](#)

[XML Parser Options](#)

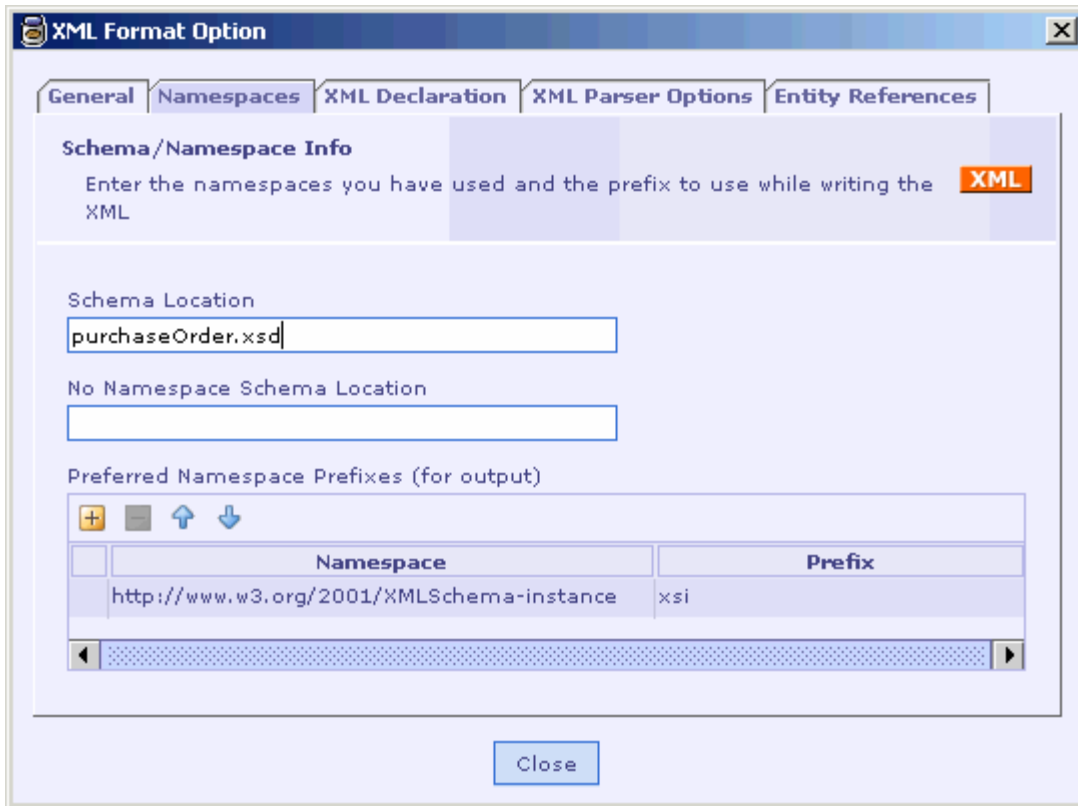
[Entity References](#)

Namespaces

The options listed under this tab are used to customize the output with regard to namespace and schema location. If the user decides not to declare a specific target namespace, he can specify where the schema for a document is located using the XML schema instance namespace. The XML schema instance namespace defines two key attributes for locating schemas: “noNamespaceSchemaLocation” and “schemaLocation”. Under this tab, you can specify three inputs as described below:

1. Schema Location

Typically, you would use the “schemaLocation” attribute where you set a default namespace for a schema and there’s no target namespace. In the “Schema Location” text box, the user can specify the value for the “schemaLocation” attribute. The value specified here would be displayed as the value of the “schemaLocation” attribute of the root element in the generated output.

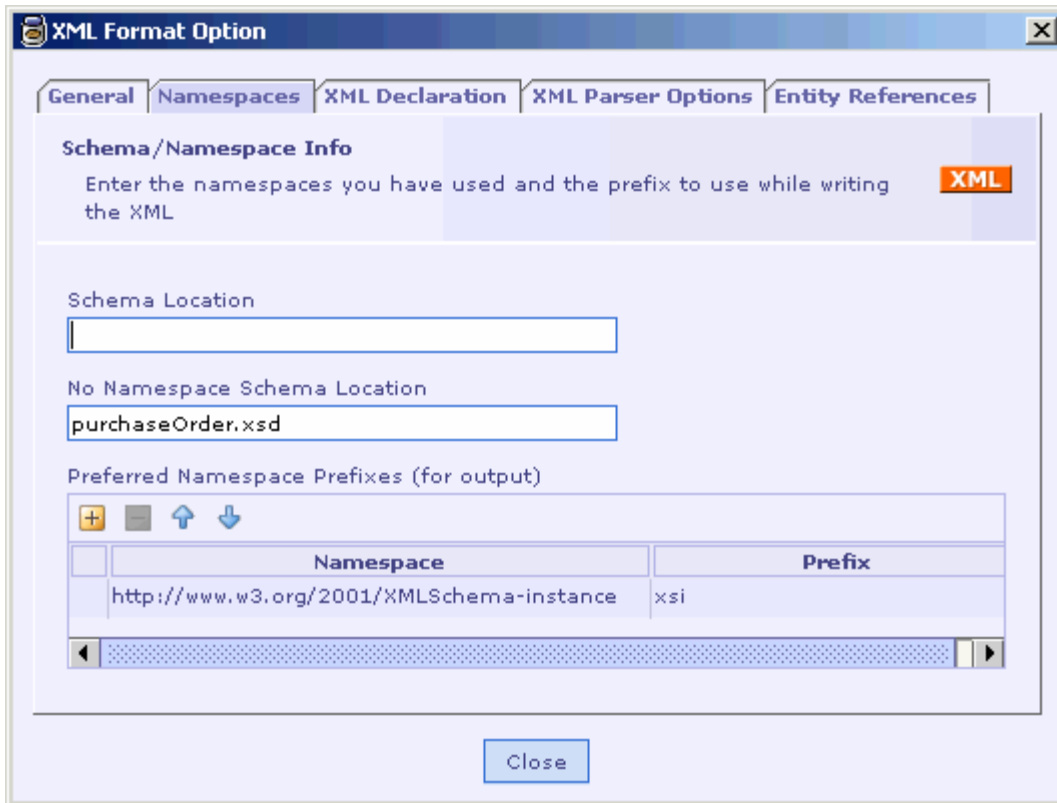


In the case depicted in the above picture, we have specified the value for the “schemaLocation” attribute as “purchaseOrder.xsd”. This would find place in the Root element of the generated “write” output as seen under:

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="purchaseOrder.xsd">
. . .
</Root>
```

2. No Namespace Schema Location

Using “noNamespaceSchemaLocation” attribute, you can make a direct reference to a schema location when there is no namespace target defined for the document. In the “No Namespace Schema Location” text box, the user can specify the value for the “noNamespaceSchemaLocation” attribute. The value specified here would be displayed as the value of the “noNamespaceSchemaLocation” attribute of the root element in the generated “write” output.



In the case depicted in the above picture, we have specified the value for the “noNamespaceSchemaLocation” attribute as “purchaseOrder.xsd”. This would find place in the Root element of the generated “write” output as seen under:

```
<?xml version="1.0" encoding="UTF-8"?>

<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="purchaseOrder.xsd">
. . .
</Root>
```

3. Preferred Namespace Prefixes (for output)

In this table, the target namespace, the schema namespace and any other name space specified in the imported schema would be displayed. The user can add / remove namespace by using the “Add” and “Remove” buttons in the tool bar of the table and also set his preferred prefix for the namespace. The namespaces specified here will be listed in the top of the output XML document. The parser will try to honor the namespaces specified here. In cases where it is not possible, the same would be ignored. If schema has target namespace, it would be automatically set in the output XML document. In case, where namespace prefix is not specified, it would be auto-generated. The three possible cases are explained below:

Case 1: Target namespace with default prefix

Preferred Namespace Prefixes (for output)

Namespace	Prefix	Specify At Root
http://www.any.com	targetnamespace	<input checked="" type="checkbox"/>
http://www.w3.org/2001/XMLSchema-instance	xsi	<input checked="" type="checkbox"/>

In the above case, after the schema is imported, the target namespace set in the schema is displayed under the "Namespace" column and a default prefix "targetnamespace" is displayed under the "Prefix" column. In this case, the value of the target namespace is shown as the value of the "xmlns" attribute in the Root element of the generated "write" output as seen under:

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns="http://www.any.com">  
...  
</Root>
```

Case 2: Target namespace with user-defined prefix

Suppose the user specifies his own preferred prefix in place of the "targetnamespace" value under the Prefix column as shown in the following picture, that prefix would find place in the generated "write" output.

Preferred Namespace Prefixes (for output)

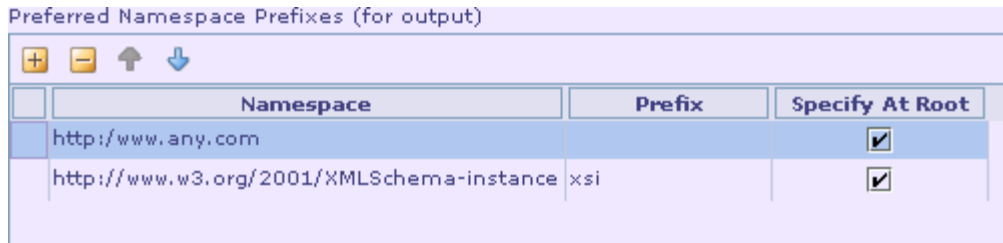
Namespace	Prefix	Specify At Root
http://www.any.com	po	<input checked="" type="checkbox"/>
http://www.w3.org/2001/XMLSchema-instance	xsi	<input checked="" type="checkbox"/>

In the above case, the prefix "po" is used in place of the default value "targetnamespace". This finds place in the generated "write" output as seen under:

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<po:Root xmlns:po="http://www.any.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance">  
...  
</po:Root>
```


Case 3: Target namespace with no prefix

Suppose the user omits to specify any value in the “Prefix” column as shown in the following picture, a prefix “ns1” would be auto-generated and that prefix would find place in the generated “write” output.



Namespace	Prefix	Specify At Root
http://www.any.com		<input checked="" type="checkbox"/>
http://www.w3.org/2001/XMLSchema-instance	xsi	<input checked="" type="checkbox"/>

In the above case, no prefix is used for the targetnamespace. A prefix “ns1” is auto-generated and finds place in the generated “write” output as seen under:

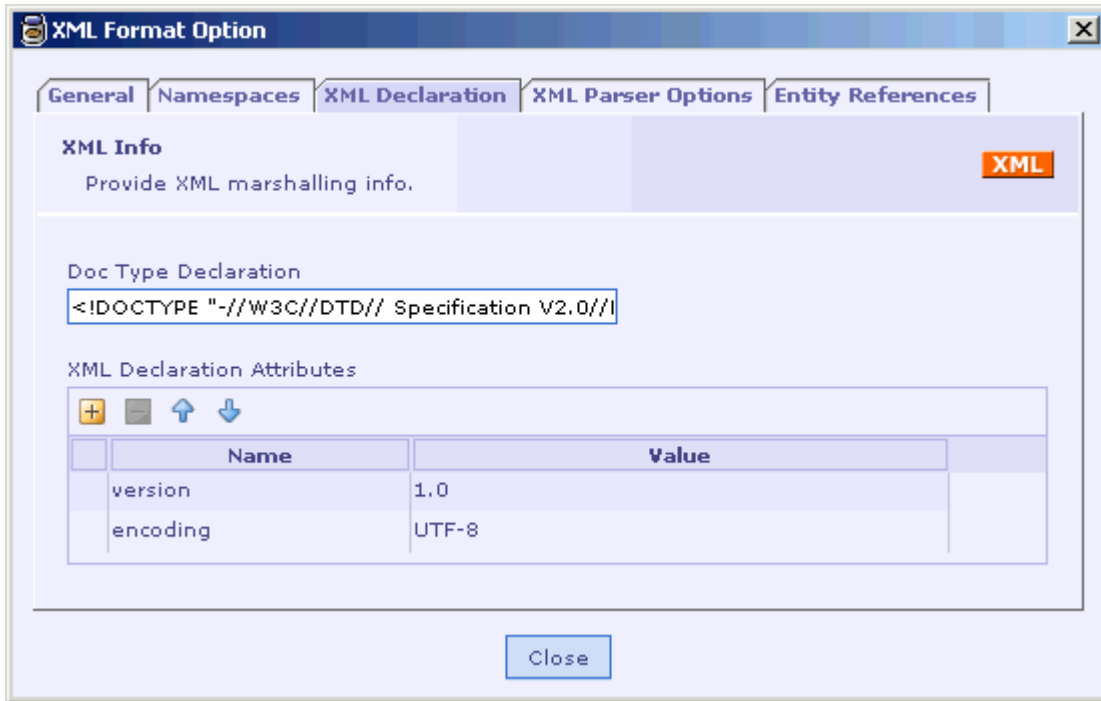
```
<?xml version="1.0" encoding="UTF-8"?>  
  
<ns1:Root xmlns:ns1="http://www.any.com"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
...  
</ns1:Root>
```

See Also:

[General](#)
[XML Declaration](#)
[XML Parser Options](#)
[Entity References](#)

XML Declaration

The options listed under this tab are used to customize the output with regard to DOCTYPE declaration and XML declaration.



- Doc Type Declaration

In DTD, the DOCTYPE declaration follows the XML declaration. The DOCTYPE declaration is a container for all DTD assignments. In external DTDs, the DOCTYPE assignment contains a Uniform Resource Identifier that identifies the location of the DTD. You can specify the required DOCTYPE declaration in the "Doc Type Declaration" textbox and the value specified here would be displayed in the DOCTYPE declaration of the generated "write" output as seen under:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE "-//W3C//DTD// Specification V2.0//EN" "/XML/1998/06/xmlspec-v20.dtd">
<purchase_order>
. . .
</purchase_order>
```

- XML Declaration Attributes

In this table, by default, the two XML declaration attributes viz., version & encoding are listed. The user has the option to add / remove any XML declaration attribute by using the "Add" and "Remove" buttons in the tool bar of the table. The attributes and the values specified here would find place in the XML declaration of the generated "write" output.

XML Declaration Attributes	
<div> <div>+</div> <div>-</div> <div>↑</div> <div>↓</div> </div>	
Name	Value
version	1.0
encoding	UTF-8
standalone	no

In the above case, we have added the “standalone” attribute. All the attributes find place in the XML declaration of the generated output as seen below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<purchase_order>
. . .
</purchase_order>
```

See Also:

[General](#)

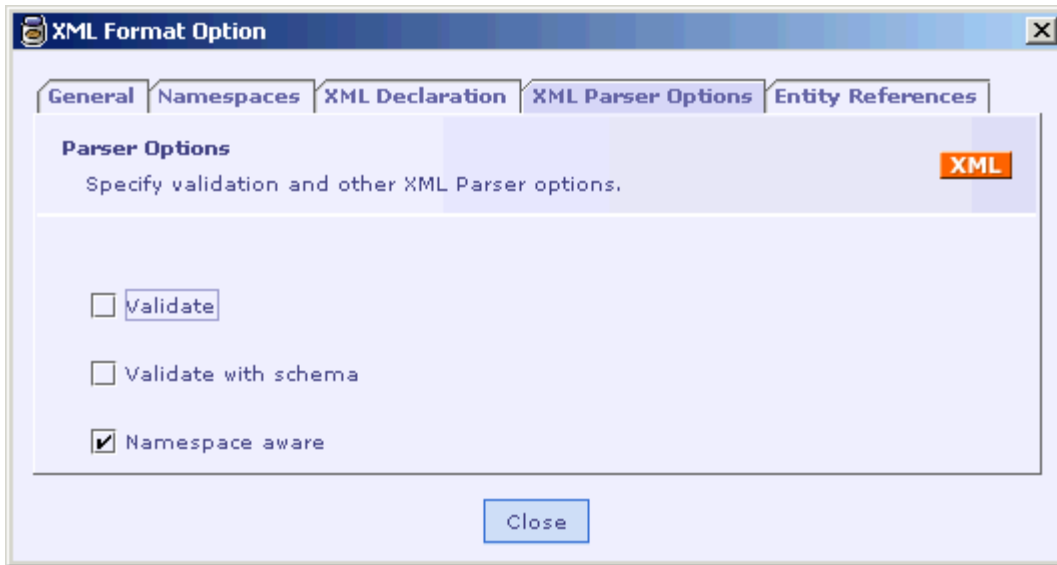
[Namespaces](#)

[XML Declaration](#)

[Entity References](#)

XML Parser Options

Under this tab, the user can specify external validation options and also the option of enabling or disabling namespace processing by the XML parser. These options are relevant for input and applicable to the XML parser (for e.g., Xerces). The validation options are applicable when the XML message structure is imported using a DTD or XSD. There exist a significant number of scenarios that do not require validation and / or cannot afford the overhead of schema validation. Typically, schema validation is disabled to improve processing speed and/or to be able to process documents containing invalid or incomplete content. An application must be allowed to disable schema validation checking during runtime. To achieve this, these options are provided.



1. Validate

If you check the “Validate” checkbox, the XML parser would validate the XML instance with the corresponding DTD if available. If it were unchecked, no schema validation would be done by the XML parser even if a corresponding DTD schema were available.

2. Validate with schema

If you check the “Validate with schema” checkbox, the XML parser would validate the XML instance with the corresponding XSD if available. If it were unchecked, no schema validation would be done by the XML parser even if a corresponding XSD schema were available.

Note:

By default, these two checkboxes would be unchecked. If you check these, you have to specify the list of schemas that define the structure of the imported XML message structure in the table provided in the [“Entity References”](#) tab.

3. Namespace aware

If your document uses namespaces, to enable / disable namespace processing, you have to check / uncheck the “Namespace aware” checkbox. If this were unchecked, the XML parser would not recognize namespaces and an element name “po:order” (with namespace prefix as “po”) would be taken as one with name “po:order” and not as “order”. When you create the XML message structure by importing a DTD, this would be unchecked by default and in case of XSD; this would be checked by default.

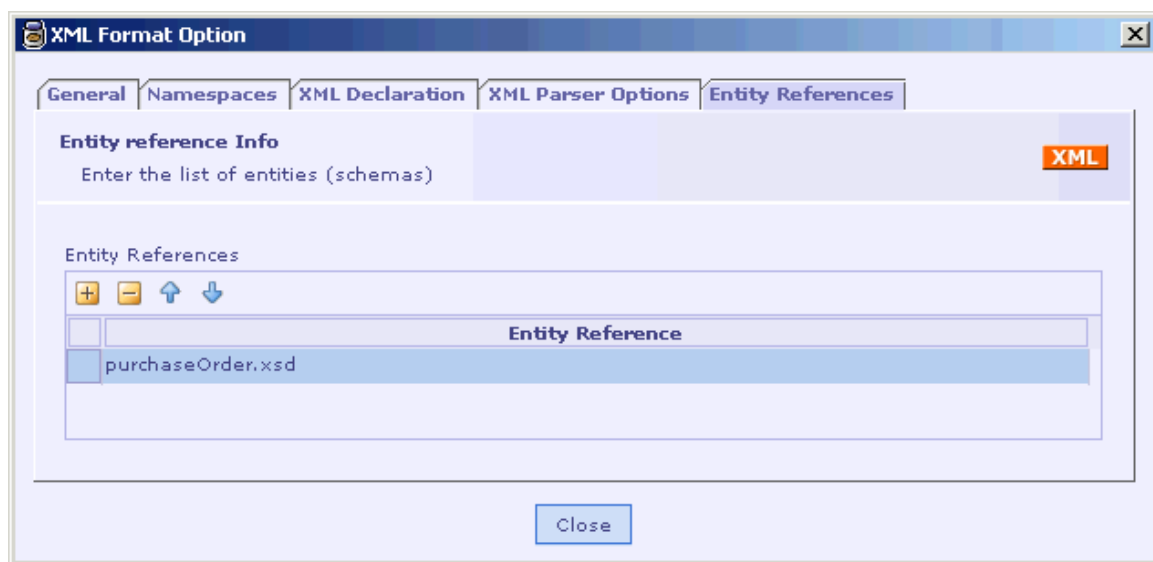
See Also:

[Format Option](#)

[Entity References](#)

Entity References

If the user decides to validate the XML instance using the XML parser, i.e., check the conformity of the XML instance with the corresponding schema (DTD / XSD) by enabling the corresponding checkbox provided in the [“XML Parser Options”](#) tab, he has to list the XSDs / DTDs directly or indirectly used for importing the current message structure in the Designer, in the table provided in the “Entity References” tab. The entities specified here are used during the external validation process.



See Also:

[General](#)

[Namespaces](#)

[XML Declaration](#)

[XML Parser Options](#)

Catalog Support

Schemas, especially those that are developed by a standard body, often contain references to external resources, such as “<http://www.w3.org/2001/xml.xsd>” or “XMLSchema.dtd”. To import those schemas, you have to either modify a schema to fix the references, or you have to make sure that the network connection is available when you import it. But both approaches are problematic.

We have added a XML catalog, which turns on the use of catalog-based entity resolver. This will allow you to essentially “redirect” references to your local copies of resources without touching the schema files.

The catalog of local schemas (XSD and DTDs) is in ‘Volante\config\xml\catalog’ directory. It also has resolver-catalog.xml where you specify the link between the system/public id to the local file.

For example the following would instruct the Designer to use xml.xsd in this directory instead of fetching it from the http location.

```
<system systemId="http://www.w3.org/2001/xml.xsd"
    uri="xml.xsd"/>
```

So if the user’s schema refers to “<http://www.w3.org/2001/xml.xsd>”, the local copy will be used. Users can update this catalog based on their requirements.

The concept of catalog has been standardized by Oasis and most XML based products support it. A catalog resolver library was available in Apache site, which we integrated with Designer.

See Also:

[Format Option](#)

Repair Functionality in XML Plugin

XML Plugin supports repair functionality, i.e., it supports generating output even if an exception is encountered at any phase of message transformation. However, the generation of output depends on the nature of exception encountered and the phase of message transformation. The correctness of the output is also not guaranteed.

For e.g., let us consider that during the parsing phase of message transformation, an exception is encountered which is non-fatal. Then, along with the exception, the parsed output also gets produced (though its correctness is not guaranteed).

This feature is supported only during certain phases of message transformation. For e.g., if an exception were encountered during the actual transformation phase (mapping), no output would be generated (null output generated) as all exceptions encountered in this phase are fatal in nature.

Some non-fatal error cases are described below:

1. Violation of facet constraints in simple type definitions.

For e.g., let us consider that you have set the length facet for a simple type element 'A' of type string as '5', but while giving data, you give the value for 'A' as "abcd". Then during the parsing phase the following exception and output would be generated:

Exception:

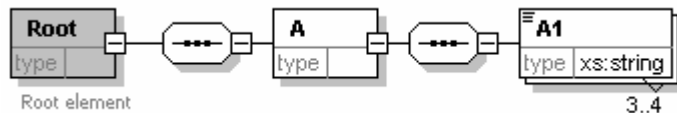
```
<TransformException>
  <Type>TransformException</Type>
  <Message>The value 'abcd' of element 'A' is not facet-valid with respect to
    length '5'</Message>
  <Severity>error</Severity>
  . . .
</TransformException>
```

Parsed Output:

```
<DataObject>
  <A>abcd</A>
</DataObject>
```

2. Violation of content specification in complex type definitions (local structural constraint), for e.g., if the occurrence constraint of a child element is violated.

For e.g., let us consider the following schema:



We have set occurrence constraints for the child element A1 of the complex element A as: minOccurs="3" maxOccurs="4".

If your XML file has, say, only 2 occurrences of the element 'A1', then during the parsing phase the following exception and output would be generated:

Exception:

```
<TransformException>
  <Type>TransformException</Type>
  <Message>The number of elements in the section 'A.A1' is less than
    '3'.</Message>
  . . .
</TransformException>
```

Parsed Output:

```
<DataObject>
  <A>
    <A1>
      <Value>one</Value>
    </A1>
    <A1>
      <Value>two</Value>
    </A1>
  </A>
</DataObject>
```

See Also:

[Creating an XML Format](#)

Glossary

Element

Elements are the basic components of an XML document. An XML document consists of individual elements that are marked by start and end tags. Tags contain the name of the element, so that they can be distinguished from one another more easily. An element can contain sub-elements, attributes and text.

any element

'any' element is one whose content is unconstrained, i.e., it can contain any type of content say, text, sub-element, attribute, etc. This is useful when you are developing new schemas and are not sure of the exact document structure that you want to use.

Attribute

Attributes are used to further specify additional information that augments the data of an element. They are specified in the start-tag of an element. They always have a name and a value and the syntax is: name = "value".

AnyAttribute

'anyAttribute' element enables any type of attributes to appear in elements. In contrast to an 'any' element, 'anyAttribute' cannot constrain the number of attributes that may appear in an element.

Group

Groups are used to structure related sets of elements.

Model Group

A model group is used to specify the order of occurrence of child elements of an element.

Compositor

A compositor specifies the relationship between two components in a content model. 'sequence', 'choice' and 'all' are the three compositors that can be used in a content model.

Sequence

'sequence' compositor indicates that the child elements must occur in the specified order.

Choice

'choice' compositor indicates that only one of the child elements must occur.

All

'all' compositor indicates that the child elements can occur in any order.

Namespace

Namespaces are used to prevent naming conflicts when like-named elements and attributes are associated with different types of data structures and are used in the same document.

Facet

Facets are applied to data types to distinguish those aspects of one data type which differ from other data types. For e.g., for a "string" data type, you can apply the facets 'length', 'minLength', 'maxLength' and 'whiteSpace'.

Flattening

Flattening is the process of removing a model group and pushing the contents of a model group to the parent of the group. There is an option provided in the Designer during schema import to enable flattening in some cases where it is not carried out by default, and is particularly useful in reducing the number of sections in the message structure.

Aggregation

Aggregation is the process where the contents of all the components referenced by the schema component type are put to the type and eventually the type itself is aggregated to the element to which it is set.

Global elements / attributes

Global elements, and global attributes, are created by declarations that appear as the children of the schema element. Once declared, a global element or a global attribute can be referenced in one or more declarations using the “ref” attribute.

See Also:

[Terminology](#)

[Creating an XML Format](#)