# WebSphere Data Interchange v3.2
# XML Schema Implementation Guide

## XML Schema CSD

As part of this Fix Pack and CSD, WebSphere Data Interchange (WDI) development has included support for XML schemas.

This document is not a formal IBM publication. It is a technical document written by WDI development to help customers use XML schemas with this CSD.

### Overview

XML schemas are quickly replacing DTDs as a way to define the structure of XML documents. Most new XML grammars are now being defined in terms of W3C XML Schema, not as DTDs. These schemas provide significantly more function than DTDs, particularly in areas such as data type validation. For more information on W3C XML Schema, see the W3C web site: http://www.w3.org/XML/Schema.

The changes included in this CSD and Fix pack allow you to:

- Import an XML schema into WDI Client, similar to the way you currently import a DTD.
- Use an XML schema as the source and/or target document of a map, similar to the way you currently use a DTD.
- Allow you to validate a source document against a schema, similar to the way that WDI currently validates XML input against a DTD.

### Importing an XML schema

Importing an XML schema using the WDI Client is similar to importing a DTD. However, instead of selecting from files of type "XML DTD file (*.dtd)", select from files of type "XML schema file (*.xsd)". After you have selected the XML schema file, you will be asked for the dictionary name, root element, and description, just as you are with DTD files. WDI Client will process files with the extension ".dtd" as a DTD file, and files with the extension ".xsd" as an XML schema.

If the schema contains a "targetNamespace" attribute, then the specified target namespace is associated with the imported schema in the database. See the section on namespace processing for more information.

If the schema contains "xmlns" attributes to identify namespaces, and the namespaces do not already exist in the namespace table, the new namespaces will be added at the time the file is imported. The prefix and other information associated with these entries may be changed later if you want. For more information, see the section on namespace processing.

## Working with XML schemas

In the XML functional area, the XML schemas are displayed on a separate tab from the DTDs. An XML dictionary can contain both schemas and DTDs. The WDI Client determines whether it is a schema or DTD when the file is imported.

A dictionary may not contain both a schema and a DTD with the same name. If you import a schema into a dictionary that has an existing DTD by the same name, the new schema will replace the DTD. Similarly, if you import a DTD into a dictionary that already has a schema with the same name, the DTD will replace the schema. For example, if you have an existing DTD named "MYPO", and import file "MYPO.XSD" into the same dictionary, the new schema will replace the existing DTD.

All information that can be associated with a DTD can also be associated with a schema. This includes the root element, description, and the sender/receiver fields. Schemas also have an additional field, the "target namespace". The target namespace is the namespace that the schema describes. It is typically specified in the schema by the *targetNamespace* attribute. When a schema is imported, WDI Client scans the data, and if a *targetNamespace* attribute is found, it will save that value as the target namespace. Normally, there is no need to change this value, and setting it incorrectly may cause errors in the map.

## Mapping using XML schemas

Creating a data transformation map using an XML schema is similar to creating a map with a DTD. (XML schemas are not supported for send and receive maps.) You still select "XML" as your source and/or target syntax, and select the XML dictionary you want to use. When it is time to select the source or target document, you can choose from a list of all DTDs and schemas in the dictionary. A "Grammar Type" column identifies whether each document is a DTD or a schema.

The tree structure for the schema is similar to that of a DTD. Some of the additional information that is provided by a schema, such as data types and repetition counts of child elements are also included as part of the element descriptions.

Special processing and commands have been added to support schemas that use XML namespaces. See the section on "Namespace Processing" for more information.

A new *Create* command has been added that allows you to create an empty XML element with no attributes if needed. Previously, you had to create an attribute or a value in order to create the XML element. This command can be used for DTDs, as well as for schemas.

**Create**

The **Create** command is used to create a specified compound element in the target data. The **Create** command uses the following format:

**Create(***targetpath***)**

Where:

*targetpath* - identifies the path in the target document definition that is to be created. *targetpath* must be a compound map node, such as an XML element. To create simple nodes such as an attribute or an element value, the assignment command or MapFrom/MapTo commands should be used.

Some special content types that are defined by schemas have limitations on their mapping capabilities. These include:

- xsd:anyType – This means that this element can contain any type of content, including child elements, simple elements, or mixed content. WDI only allows this to be mapped as if it were a simple *string* element.
- xsd:any – This means that any child element may appear in this position, sometimes subject to namespace restrictions. WDI does not allow you to map to or from this element.
- xsd:anyAttribute – This means that any attribute(s) may appear on this element. WDI does not allow you to map to or from this attribute.
- Substitution groups – This means that one element may be substituted for another. WDI does not allow mapping to or from the substitution group elements.

## Translation using XML schemas

Translating data using an XML schema is similar to translating using a DTD. First you create the map as described above. Then you translate the data using the PERFORM TRANSFORM command.

Two new keywords have been added for the PERFORM TRANSFORM command:

**XMLNS**

| | |
|---|---|
| **Y** | Do namespace processing for input XML data. |
| **N** | Do not do namespace processing for input XML data. This is the default. |

If you are doing schema validation, or if the input document is based on an XML schema that uses namespaces, you should set XMLNS(Y). See the section on namespace processing for more information.

**XMLSCHEMAVAL**

| | |
|---|---|
| **Y** | Validate the data against an XML schema |
| **N** | Do not validate the data against an XML schema. This is the default. |
| **A** | Validate the data against an XML schema only if one is specified. If no schema is specified, do not attempt to validate against a schema. |

The XML schema to validate against should be in the directory specified by the XMLDTDS keyword.

**XML schema validation is not supported when using CICS Transaction Server 1.3.** This is because the older XML parser used for Transaction Server 1.3 (from XML Toolkit V1R2) based its schema support on an early draft of the W3C schema specification. This early implementation does not match the final W3C recommendation. If you attempt to validate against an XML schema when using Transaction Server 1.3 and XML Toolkit V1R2, this will typically result in validation errors even if the data is correct according to the W3C schema recommendation.

Due to the way in which the parser constructs content models for elements with complex content, specifying large values for the minOccurs or maxOccurs attributes may cause a stack overflow or very poor performance in the parser. Large values for minOccurs should be avoided and unbounded should be used instead of a large value for maxOccurs.

If you set XMLSCHEMAVAL to Y(es) or A(uto), DTD validation is also done if a DTD is specified in the XML data. If the XML document specifies both an XML schema and a DTD, WDI does not allow you to validate only against the schema, but ignore the DTD.

## Namespace processing

Namespaces allow the use of multiple XML schema definitions within an XML document or building a grammar from several different schemas, by providing a way to resolve name conflicts between the schemas. For example, one schema may define an **address** element to be a person name, that includes street, city, state, and zipcode elements. Another may define **address** to be an e-mail address that contains a simple string. By using different namespaces, these can be uniquely identified.

For example, to show that element **address** belongs to a particular namespace, a schema and/or instance document defines a prefix for the namespace. For example, it might define **xmlns:po="http://example.com/ns/POExample"**. Then when the qualified element appears in the data, the element would appear as **<po:address>**, to show that it is part of that namespace. The **po** prefix in this example is just a shorthand way to represent the namespace **"http://example.com/ns/POExample"**. Even though the schema might use prefix **po**, instance documents may use different prefixes, for example **mypo**, **purch**, or even define it as the default namespace, so no prefix is used for this element. As long as these were defined to the **"http://example.com/ns/POExample"** namespace in the instance documents they should all be considered equivalent.

The mapping and translation for XML schemas is "namespace aware." This means that it recognizes that the prefix represents a namespace, and gives it special treatment instead of just treating it as part of the element name. Since the schema translation and mapping is namespace-aware, elements with the same name and namespace would be considered to be equivalent, even if they used different prefixes.

## Namespace Table

The namespace table is a new tab on the Client XML functional area. It provides information that is associated with each namespace URI. Each namespace table entry is associated with an XML dictionary. Besides the namespace URI and dictionary name, each entry includes the following information for the URI when it is used in that dictionary.

- **Description** – This is a text description for the namespace. It is only used to help the user identify the namespace, and is not used for mapping or translation.
- **Prefix** – This is the prefix that will be used for the namespace when qualified elements and attributes are displayed in the map, and when the elements and attributes are written in the XML output. If no prefix is specified, or if no namespace entry exists for a namespace URI, then the element or attribute is written without a prefix (i.e., using the default namespace).
- **Schema location** – This specifies the schema location that is used if the namespace URI is used in a **SetSchemaLocation** command.

When a schema is imported, the schema is scanned for any **xmlns** attributes. If an **xmlns** attribute is found namespace for a namespace URI that is not already defined for the dictionary, a new namespace table entry is created using the prefix defined for the attribute. The description and schema location can be added later if needed.

The prefix and schema location are not included in the internal map representation, so the map does not need to be recompiled if these values are changed.

## Target Namespace

The target namespace tells which namespace a schema describes. It is identified in the schema by the **targetNamespace** attribute.

When a schema is imported, the schema is scanned for a **targetNamespace** attribute. If this attribute is found, the target namespace for the schema is set from the attribute value. The target namespace associated with the schema can be changed, but typically should not be unless the import was not able to find it correctly. Setting the target namespace to a value that does not match the **targetNamespace** attribute can prevent WDI Client from parsing the schema correctly, which will result in errors when trying to map the document.

**Namespace Processing for Input XML documents**
You must specify **XMLNS(Y)** on your PERFORM command if you are doing schema validation, and/or your XML schema uses namespace qualified elements/attributes. Namespace processing is required for schema validation, because the attributes that help locate the schema definitions (**xsi:schemaLocation** and **xsi:noNamespaceSchemaLocation**) use namespace qualification. If your source or target document is a schema that uses qualified elements and/or attributes, namespace processing is required so the internal names used by the translation process will match the internal names used in the map.

**Namespace Processing for Output XML Documents**

When generating output XML documents, qualified elements and attributes use the prefix specified by the namespace table entry for the URI. If no prefix is specified, or if no namespace entry exists for a namespace URI, then the element or attribute is written without a prefix (i.e., using the default namespace).

Some special attributes are sometimes needed in the XML output to identify the namespaces used and the schema location(s). Some new mapping commands have been added to specify these. These commands are only used for XML output, and ignored for EDI and data format output.

**SetNoNSSchemaLocation(location)**

Where:
**location** is a character expression that indicates the schema location for the output.

If this command is included, an attribute of the following form will be created on the root element of the XML output:
**xsi:noNamespaceSchemaLocation=”location”**

For example: **noNamespaceSchemaLocation(“myschema.xsd”)**
Would generate the following:
**xsi:noNamespaceSchemaLocation=”myschema.xsd”**

Note: The **xsi** prefix will be changed if you have a different prefix specified in the namespace table for http://www.w3.org/2001/XMLSchema-instance. If you do not have a namespace table entry for http://www.w3.org/2001/XMLSchema-instance the default prefix **xsi** will be used. An appropriate

**xmlns:prefix=**http://www.w3.org/2001/XMLSchema-instance attribute will automatically be generated when this command is used.

If there is more than one occurrence of this command, only the last value is used.

**SetSchemaLocation(URI)**

Where:
**URI** is a URI from the namespace table.

If this command is included, an attribute of the following form will be created on the root element of the XML output:
**xsi:schemaLocation="URI location"**

The specified URI is looked up in the namespace table, and the location is taken from that entry. For example:
**SetSchemaLocation("http://www.ibm.com/schema/example")**
Would generate the following (assuming the namespace table defines **example.xsd** as the location for namespace **http://www.ibm.com/schema/example**):
**xsi:schemaLocation="http://www.ibm.com/schema/example example.xsd"**

Note: The **xsi** prefix will be changed if you have a different prefix specified in the namespace table for http://www.w3.org/2001/XMLSchema-instance. If you do not have a namespace table entry for http://www.w3.org/2001/XMLSchema-instance the default prefix **xsi** will be used. An appropriate **xmlns:prefix=**http://www.w3.org/2001/XMLSchema-instance attribute will automatically be generated when this command is specified.

This command may be specified multiple times in the map. If there is more than one occurrence of this command, then a namespace/location pair will be created for each command.

**SetNamespace(URI)**

Where:
**URI** is a URI from the namespace table.

If this command is included, an attribute of the following form will be created on the root element of the XML output:
**xmlns:prefix="URI"**

The specified URI is looked up in the namespace table, and the prefix is taken from that entry. For example: **xmlns("http://www.ibm.com/schema/example")**

Would generate the following (assuming the namespace table defines **xmp** as the prefix for namespace **http://www.ibm.com/schema/example**):

```
xmlns:xmp=”http://www.ibm.com/schema/example”
```

If there is more than one occurrence of this command, then an attribute will be created for each command.