# DataInterchange v3.1 Technical XML Implementation Guide

## XML PTF Overview

DataInterchange development has created this PTF in response to the immediate demand to support XML processing. The DI XML processor enhancement for DI 3.1 should be viewed as a short-term solution for implementing XML translation. The enhancement is supplied via host PTF and DI Client Fixpak 6 for DI 3.1 customers to process XML data more easily. The initial enhancement is for MVS batch processing and is available for both the DataInterchange MVS and DataInterchange MVS/CICS products. Processing for CICS environment is planned to follow soon after. The PTF will allow translation for XML data in both directions. Application data to XML for send or outbound processing, and XML to application data for receive or inbound processing, are both supported. It is implemented by converting an XML DTD to an "EDI standard". The "EDI standard" is a representation of the XML data using an EDI-type syntax that is understood by DI. A DTD conversion utility (Windows application) will be used to define an "EDI standard" from the user's DTD. The generated standard has a structure similar to the XML document.

Note: This document is not a formal IBM publication. It is a technical document written by DataInterchange development to help customers implement XML using this PTF.

### Prerequisite Software

The following software will be new prerequisites for the customer to install this PTF:

- OS/390 V2R6
- XML Tool kit for OS/390 ("non-priced" software) http://www.s390.ibm.com/xml/

### Acknowledgments

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved. This software is distributed under Apache Software License, Version 1.1 . A copy of this license is included in the DataInterchange Client installation directory (LICENSE.TXT).

## Converting an XML DTD

### Overview

To implement XML processing using DI, the first step is to convert the DTD using the DTD conversion utility (DTDConvert) . The DTD conversion utility is a GUI standalone program available with DI Client Fixpak6, which reads an XML DTD and generates an EDI standard representation of the XML structure. The EDI standard can be used by DataInterchange to perform normal mapping and translation functions.

The utility will convert each DTD file that you select. From the DTD file and other user input, it will generate two output files:

- An XML dictionary. This file contains information that allows DI to correlate the XML elements and attributes with the "EDI standard" segments and data elements.
- An import file for DI Client. This file is imported into DI Client to define the EDI standard in the database.

You can invoke the DTD conversion utility by selecting the corresponding shortcut or by double-clicking on the DTDCONVERT.EXE file located in the DataInterchange install directory. Then fill in the fields requested by the DTD conversion utility. The Browse button can be used to locate and identify the XML DTD file that will be used as input to the utility. The DTD Name and Root Element Name fields are required. After entering the required data and any optional data you need, press the OK button to begin converting the DTD. During conversion processing, a status window is displayed. The status window displays messages pertinent to the conversion, including identifying the names of the two output files produced by the utility. You can close the status window using its Close button. After the conversion of a DTD has been completed, you can convert another DTD by specifying the necessary information in the DTD conversion utility. Once you have completed all conversions, close the utility by pressing the Close button.

Note: The DTD conversion utility is supported on the Windows 98, Windows NT, and Windows 2000 platforms. It has successfully been used with Windows 95 in a test environment. However, Windows 95 is not supported because some underlying software is not supported in that environment.

## Input fields

The DTD conversion utility includes on-line help and will take the following as user input:

**DTD Name** (required) - The name of the DTD to be converted. The DTD name is selected by using the "Browse" button to locate and select a DTD file. The maximum length for the DTD file name is 63 characters (including the extension). File names longer than 63 characters will be truncated.

**Root element name** (required) - The XML element name for the root element. The root element is an element that completely contains all other elements of the document. The maximum length is 64 characters. Only valid XML element name characters are permitted in this field (see note below).

**Standard name** (optional) - The name of the EDI standard to be generated from this DTD. If not specified, the first 8 alphanumeric characters of the root element name are used. The maximum length is 8 characters. Only letters and numbers are permitted in this field.

**Transaction name** (optional) - This defines the EDI standard transaction name that will be associated with this DTD. If not specified, the first 6 characters of the root element name are used. The maximum length is 6 characters. Only valid XML element name characters are permitted in this field (see note below).

**Sender Qualifier element** (optional) - The XML element name, attribute, or path that defines the sender qualifier for the standard. The maximum length is 64 characters. Only valid XML element name characters (see note below), and the characters "/" and "$" to indicate path or attribute names, are permitted in this field.

For inbound processing, the value from the sender qualifier is used as the interchange sender qualifier. If the Sender Qualifier element is not specified, or if the specified element, attribute or path is not found in the XML data, the value "ZZ" will be used as the default sender qualifier.

For outbound processing, this field is not used, and the interchange sender qualifier will always be "ZZ".

**Sender ID element** (optional) - The XML element name, attribute, or path that defines the sender ID for the standard. The maximum length is 64 characters. Only valid XML element name characters (see note below), and the characters "/" and "$" to indicate path or attribute names, are permitted in this field.

For inbound processing, the value from the sender ID element is used as the interchange sender. If the Sender ID element is not specified, or if the specified element, attribute or path is not found in the XML data, the value "XMLPROC" will be used as the default sender ID.

For outbound processing, this field is not used, and the interchange sender ID will always be "XMLPROC".

**Receiver Qualifier element** (optional) - The XML element name, attribute, or path that defines the receiver qualifier for the standard. The maximum length is 64 characters. Only

valid XML element name characters (see note below), and the characters “/” and “$” to indicate path or attribute names, are permitted in this field.

For inbound processing, the value from the receiver qualifier is used as the interchange receiver qualifier. If the Receiver Qualifier element is not specified, or if the specified element, attribute or path is not found in the XML data, the value "ZZ" will be used as the default receiver qualifier.

For outbound processing, this field is not used, and the interchange receiver qualifier will always be “ZZ”.

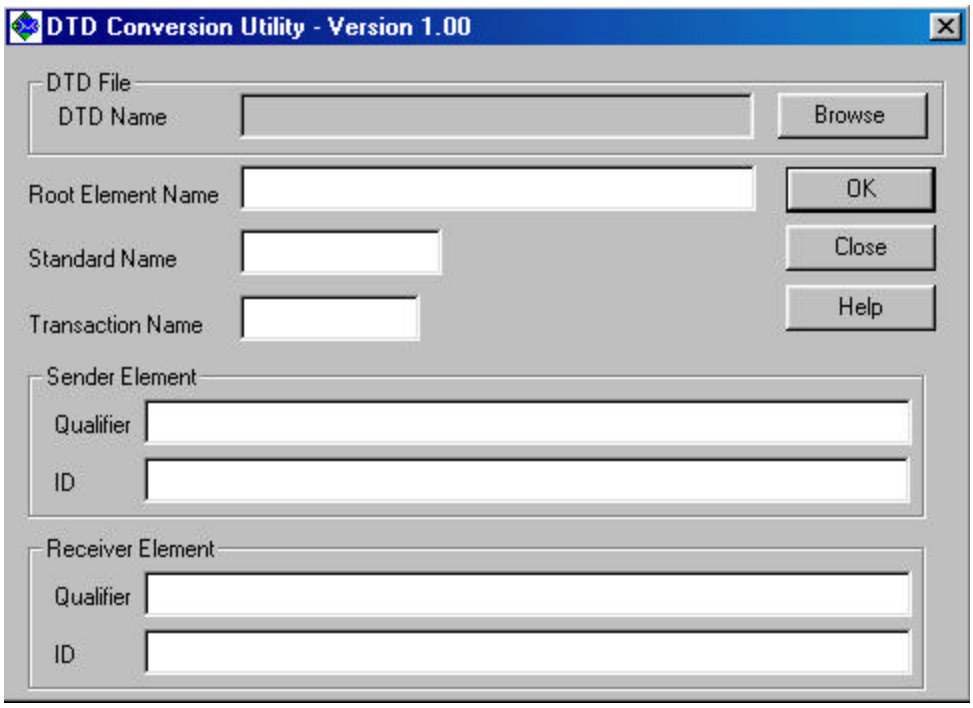
**Receiver ID element** (optional) - The XML element name, attribute, or path that defines the receiver ID for the standard. The maximum length is 64 characters. Only valid XML element name characters (see note below), and the characters “/” and “$” to indicate path or attribute names, are permitted in this field.

For inbound processing, the value from the receiver ID element is used as the interchange receiver. If the Receiver ID element is not specified, or if the specified element, attribute or path is not found in the XML data, the value "XMLRCVR" will be used as the default receiver ID.

For outbound processing, this field is not used, and the interchange receiver ID will always be “XMLRCVR”.

Note: The valid characters for XML elements includes letters, numbers, and the following special characters: period (.), dash (-), underscore (_), and colon (:).

Below is a screen shot from the DTD conversion utility:




## Utility Output files

After converting your DTD, two new files will be created in the same directory as the DTD you selected:

- DI Client import file:
  The DI import file is used to import the new EDI standard into DI Client. The filename will be in the format *stdname*.EIF, where *stdname* is the standard name you specified. If you did not specify a standard name, the standard name is taken from the first 8 alphanumeric characters of the root element.

  This file is already in DI Client format and will not need to be converted. For instructions on how to import a standard into DI Client, please refer to the DI Client documentation.

- XML dictionary file:
  The XML dictionary is used for runtime processing. It is used to correlate the XML element and attribute names with EDI standard segments and data elements. The filename will be in the format *stdname*.DIC, where *stdname* is the standard name you specified. If you did not specify a standard name, the standard name is taken from the first 8 alphanumeric characters of the root element.

This should be uploaded to your host environment and placed either in a PDS member or a HFS file. The naming of the host file name for the XML dictionary is discussed more in the section on "XML Dictionary resolution".

You will also still have the DTD file that you converted. The DTD file is optional for runtime processing. It can be used to validate the inbound or outbound XML data and/or specify default XML attribute values. Whether DI uses the DTD or not depends on the validation level for the transaction. If you want DI to process the DTD during the XML parsing, this file should be uploaded to your host environment and placed in a PDS member or HFS file. The naming for this file is discussed more in the section on "XML DTD resolution".

The DTD and XML dictionary files can be considered library type files for XML processing.

**Specifying Sender and Receiver information**

To identify the sender and receiver ID and qualifier fields using the DTD conversion utility, you simply enter the name of the XML element that contains the value. For example, if the XML data contains the following:

```
<Header>
    <From>SenderName</From>
    <To>ReceiverName</To>
</Header>
```

You would just enter the values "From" and "To" in the Sender ID and Receiver ID fields. Note that these fields, like all XML elements names, are case sensitive. This example does not have any qualifiers, so you would leave those blank.

Sometimes the sender and receiver information is in an attribute. The format to specify an attribute is "elementName$attributeName". For example, if the XML data contains:

```
<Header>
    <From type="DUNS">SenderName</From>
    <To type="DUNS">ReceiverName</To>
</Header>
```

You would specify the sender and receiver IDs as in the first example. For the sender and receiver qualifiers you would specify:

- Sender qualifier: "From$type"
- Receiver qualifier: "To$type"

Attribute names, like element names, are case sensitive.

In some cases, the sender and receiver information is specified not just by a unique element, but also depends on the context in which the element occurs. (i.e. The meaning depends on the parent or ancestor elements). Take the following XML as an example:

```
<Header>
      <From>
        <Credential domain="DUNS">
          <Identity>942888711</Identity>  <!-- Sender ID -->
        </Credential>
      </From>
      <To>
        <Credential domain="Name">
          <Identity>XMLTEST1</Identity>  <!-- Receiver ID -->
        </Credential>
      </To>
    </Header>
```

In this example the sender and receiver IDs are both located in Identity elements, and the qualifiers occur in the **domain** attribute of the Credential elements. But, the Credential and Identity elements occur within both the From and To elements so we will need to identify the path to sender and receiver. You only need to include as much path information as required to uniquely identify the element. We will use the To structure for the receiver ID and the From structure for the sender ID. In the DTD conversion utility you would enter the following:

| | |
|---|---|
| **Sender ID element** | From/Credential/Identity |
| **Sender qualifier element** | From/Credential$domain |
| **Receiver ID element** | To/Credential/Identity |
| **Receiver qualifier element** | To/Credential$domain |

Other notes on specifying the sender and receiver information:

- The maximum path length for each sender and receiver element field is 64 characters.
- If the values in the XML data for the sender and receiver ID are longer than 35 characters, they will be truncated to 35 characters
- If the values in the XML data for the sender and receiver qualifier are longer than 4 characters, they will be truncated to 4 characters

## Diagnosing DTD Parsing Errors

Sometimes the DTD conversion utility will detect a syntax error, or issue a warning while trying to parse the DTD. This section is intended to help you understand the messages that the utility displays, so you can make any corrections that are needed.

First, a short explanation of how the utility parses the DTD file:

1. First the utility creates short buffer of XML data, containing an XML declaration (“<?xml...”), a document type declaration (“<!DOCTYPE..."), and an empty root element. It uses the root element and the DTD name that you provide as input to create this XML data.
2. Then the utility passes the XML data to a parser component, which will parse the data.
3. If the XML data and the referenced DTD file are well-formed (syntactically correct), then the parser returns information about elements and attributes defined by the DTD. The utility will use this information to generate the XML dictionary file and the DI Client import file.
4. If the data is not syntactically correct, the parser component will pass error messages to the utility, which will be displayed in the status window.

If the parser detects a syntax error, the status window will display messages that look something like the following:

> Fatal error parsing the DTD - File: bad.dtd, line: 6, column: 2
> Message text: Invalid document structure

The first line tells you the name of the DTD file, as well as the line and column number where the parser detected the error. Note that this is the position that the parser **detected** the error, not necessarily where the error is. For example, if you are missing the end of a comment in the DTD, the parser will tell you where it first detected the error, not where you meant for the comment to end.

The second line displays the error message that was returned from the parser. The words "Message text:" are not part of the parser error message, but are generated by the utility.

In some cases, you may see the file name "XMLBuffer". This means that the error was detected in the buffer of data that the utility generated. This could be caused by something such as a syntactically bad root element name, or an unreadable DTD file. To see the buffer of XML data that was created (XMLBuffer), you can look at the trace file that the utility creates. This file is named "xmltrc”, and is located in the same directory as the DTD. (Note: The “xmltrc” file may contain messages that start with “Ignoring validation error in....” These are a result of using an empty root element, and do not indicate a problem.)

There are also other warning messages that may be generated by the utility, even if the DTD is parsed without errors. These are issued when the utility finds something about the element structure that may or may not indicate a problem. Warning messages are generated for the following conditions:

- If an element is declared as a descendent of itself, the following message is displayed:

  > Warning : Element "*name*" declared as a descendant of itself in the DTD file
  > The nested instance of element "*name*" is ignored

An example of this is if element A has child elements B and C. However, element B has child A as one of its child elements. In this case, the utility will ignore the nested occurrence of A, and omit it from the list of children for element B. A will not appear as a child of B in the EDI standard that is generated, and errors may occur if XML data is received that contains A as a child element of B. If incoming data does not use the nested occurrence of the element, and you do not need to map data to it, then this warning should not cause a problem.

- If the parser indicates that the root element has content model “Any”, the following message is displayed:

  Warning : Root element "*name*" has content model "Any"
  Element "*name*" might not be declared in the DTD file

  There are two cases where the parser will indicate that the root element has content model “Any”. One case is that the DTD actually declares the element to by type “Any”. Although this is legal, this type of DTD is not very useful for mapping, since it does not define any structure for the DTD. The other case that may cause this is that the root element specified in the input field is not declared in the DTD file. (Note: XML element names are case sensitive.) In this case, the parser assumes that it should be type “Any”.

  If you see this warning, you should verify that the root element name is correct for the DTD (including any case sensitivity). If not, correct it and retry. If the root element name really is declared as type “Any”, then you will not be able to do much useful mapping for this DTD.

# Mapping

## Overview

Mapping XML data is similar to mapping EDI data using DI Client. To map between data format and XML, just map the data format fields to the data elements that correspond to the appropriate XML elements and attributes. To map a data format field to an XML element value, map it to the first data element of the segment that corresponds to the XML element. To map a data format field to an attribute of an XML element, map it to the corresponding data element on the segment for the XML element. The full XML element names are provided in the segment descriptions, and the attribute names are included in the data element descriptions.

Most of the segment names are based on the name of the XML element that they represent. However, there will be some segments and loops you will not recognize. These segments start with "$SQ" and "$CH", and are created by the DTD conversion utility to represent certain types of complex DTD element definitions.

Many segments do not contain any data elements. These segments are used to indicate the start and end of loops and are automatically qualified with occurrence 1 when you map inner loops using DI Client. The XML and translation processing will handle these segments.

Since mapping for XML data is to and from an EDI standard representation of the XML data, all DI mapping functions apply such as Boolean logic, segment, loop, element qualification, accumulators, application control fields, translation/validation tables, rawdata, and C&D application data formats.

## EDI Standard Representation of XML data

There are some special rules used to equate the XML elements and attributes with the standard segments and data elements. The segment and data element descriptions contain the element names and attributes that they correspond to, but a general understanding will help you determine, find and understand these better. The following rules apply:

- Each XML element corresponds to an EDI segment.

  The element name is truncated to 6 characters to create the segment ID. If the truncation causes duplicate segment names, then a sequence number is used for the last character(s) to ensure uniqueness. The segment description includes the full XML element name.

- If the XML element is declared as type "#PCDATA", "ANY", or mixed content (a mixture of #PCDATA and child elements), then the element value(s) correspond to the first data element on the segment.

When receiving these types of elements, all element values within the element are concatenated and placed into the first data element on the segment. The tags and attributes of any nested child elements for types mixed and ANY are discarded. (#PCDATA elements do not have nested child elements.)

When sending these types of elements, the first data element of the segment is used as the element value.

The description for the first data element includes the name of the XML element, and also its content model, either "Any" or "PCData/Mixed".

- If the XML element is declared as type "EMPTY", then the first data element is not used since no element value is allowed.

  If the element has attributes (usually the case for EMPTY elements), the first data element on this segment is the special data element "UNUSED", which should not be mapped. If the element does not have any attributes, then the segment does not have any data elements.

- If the XML element has any attributes, then the attributes correspond to data elements 2-n on the segment.

  The description for each data element includes the XML element name, followed by a "$" character, followed by the attribute name. For example, "elementName$attrName".

- If the XML element has child elements, then a loop is defined for it.

  The first segment of the loop is the segment that corresponds to the XML element as described above. The first data element of this segment is the special data element "UNUSED", which should not be mapped. If the XML element has attributes, they appear as elements 2-n as normal. The start segment does not need to be mapped, unless you want to map attribute values. The last segment of the loop has the same segment id as the first segment, plus a "$" at the end. The end segment does not need to be mapped. The start and end segments will be written if any of the nested segments within the loop contain data.

**Sequences**

A sequence means that the child elements are defined as a sequence of elements. For example: <!ELEMENT A (B,C*)> . This means element A consists of element B, followed by 0 or more occurrences of element C. The XML data might appear something like the following:

```
<A>
    <B>B-Value</B>
    <C>C-value</C>
    <C>C-value2</C>
</A>
```

In this case, element A is defined as a loop, since it has children B and C. Each child in the sequence is defined as a segment within the loop. Each segment is marked as either mandatory or optional, and as either repeating or non-repeating, depending on how the element is defined in the sequence.

In this example, segment B would be mandatory and not repeating, since the DTD states that element B must occur exactly one time in the sequence. Segment C would be defined as optional and repeating, since the DTD states that it occurs 0 or more times.

If B and/or C had other child elements, then they would appear as a nested loop (or loops) within loop A. Loops may be defined as repeating, but may not be defined as mandatory. In this case, it is up to the person performing the mapping to understand the restrictions imposed by the DTD and make sure that some data within loop B is mapped in order to generate valid XML data. Otherwise, the data would not validate properly against the DTD.

**Choices**

A choice means that the child elements are defined as a choice of elements. For example: <!ELEMENT A (B|C)> . This means element A consists of either element B or element C. The XML data might appear something like the following:

```
<A>
    <B>B-Value</B>
</A>
        - or -
<A>
    <C>C-Value</C>
</A>
```

Again, element A is defined as a loop, since it has children B and C. Each child in the sequence is defined as a segment (or loop) within the loop. Each segment (or loop) in a choice is marked as optional, and not repeating. It is up to the person performing the mapping to understand the structure of the XML data and only map one of the child segments or loops.

**Special Sequences and Choices**

Some types of XML sequences and choices require special handling to represent them in an EDI type structure. These cases are:

- Repeating sequences
- Repeating choices
- A sequence within a choice
- A choice within a sequence

For each of these, some special segments will be generated and inserted in the EDI standard transaction representation. These will start with "$CH" (for repeating choices or a choice within a sequence) or "$SQ" (for repeating sequences or a sequence within a choice). Segments beginning with $SQ and $CH will have no data elements.

**Mapping example**

As an example, look at the following XML data structure:

```
<ItemIn quantity="6" lineNumber="1">
     <ItemID>
        <SupplierPartID>pn12345</SupplierPartID>
        <SupplierPartAuxiliaryID>EA</SupplierPartAuxiliaryID>
     </ItemID>
</ItemIn>
<ItemIn quantity="12" lineNumber="2">
     <ItemID>
        <SupplierPartID>823488664ZZZ</SupplierPartID>
        <SupplierPartAuxiliaryID>345602840800</SupplierPartAuxiliaryID>
     </ItemID>
</ItemIn>
```

Now if we look at the sections:

<ItemIn quantity="6" lineNumber="1"> **Loop with attributes `(quantity,lineNumber)`**

<ItemID> **Loop**

| **Segment** | **Element** | |
|---|---|---|
| <SupplierPartID> | pn12345 | </SupplierPartID> |
| <SupplierPartAuxiliaryID> | EA | </SupplierPartAuxiliaryID> |

</ItemID> **End Loop**

</ItemIn> **End Loop**

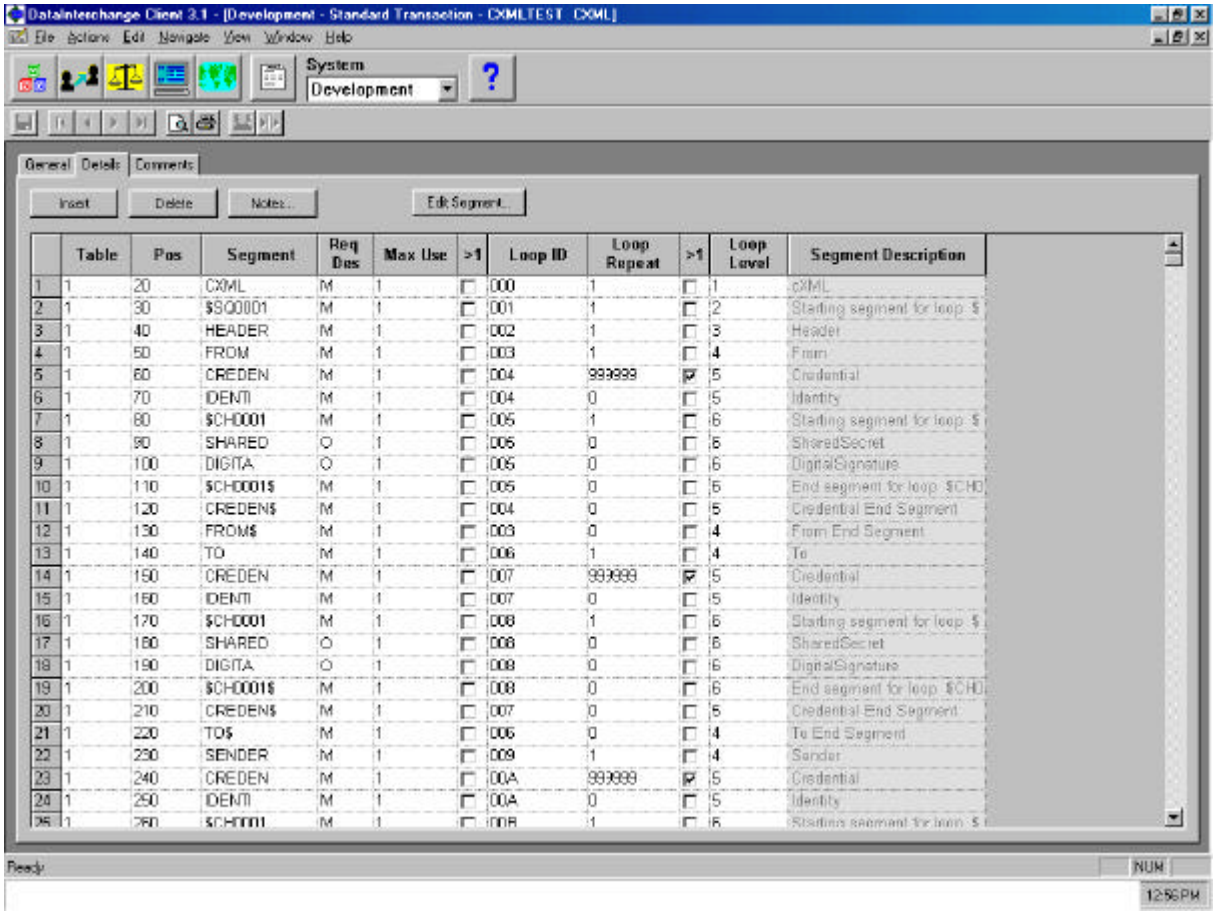
```
<ItemIn quantity="12" lineNumber="2">
     <ItemID>
        <SupplierPartID>823488664ZZZ</SupplierPartID>
        <SupplierPartAuxiliaryID>345602840800</SupplierPartAuxiliaryID>
     </ItemID>
</ItemIn>
```

Let's begin with the inner most structures SupplierPartID and SupplierPartAuxiliaryID identified under the heading **Segment**. These are considered child elements of ItemID. Each child of ItemID, SupplierPartID and SupplierPartAuxiliaryID, has only one child, identified under **Element**. The SupplierPartID and SupplierPartAuxiliaryID are represented in the EDI standard as segments or repeating segments with one element. The segment ID is generated from the XML element name. The element ID is the segment ID + sequence number. Since SupplierPartID and SupplierPartAuxiliaryID are child elements of ItemID, ItemID is represented in the EDI standard as a loop.

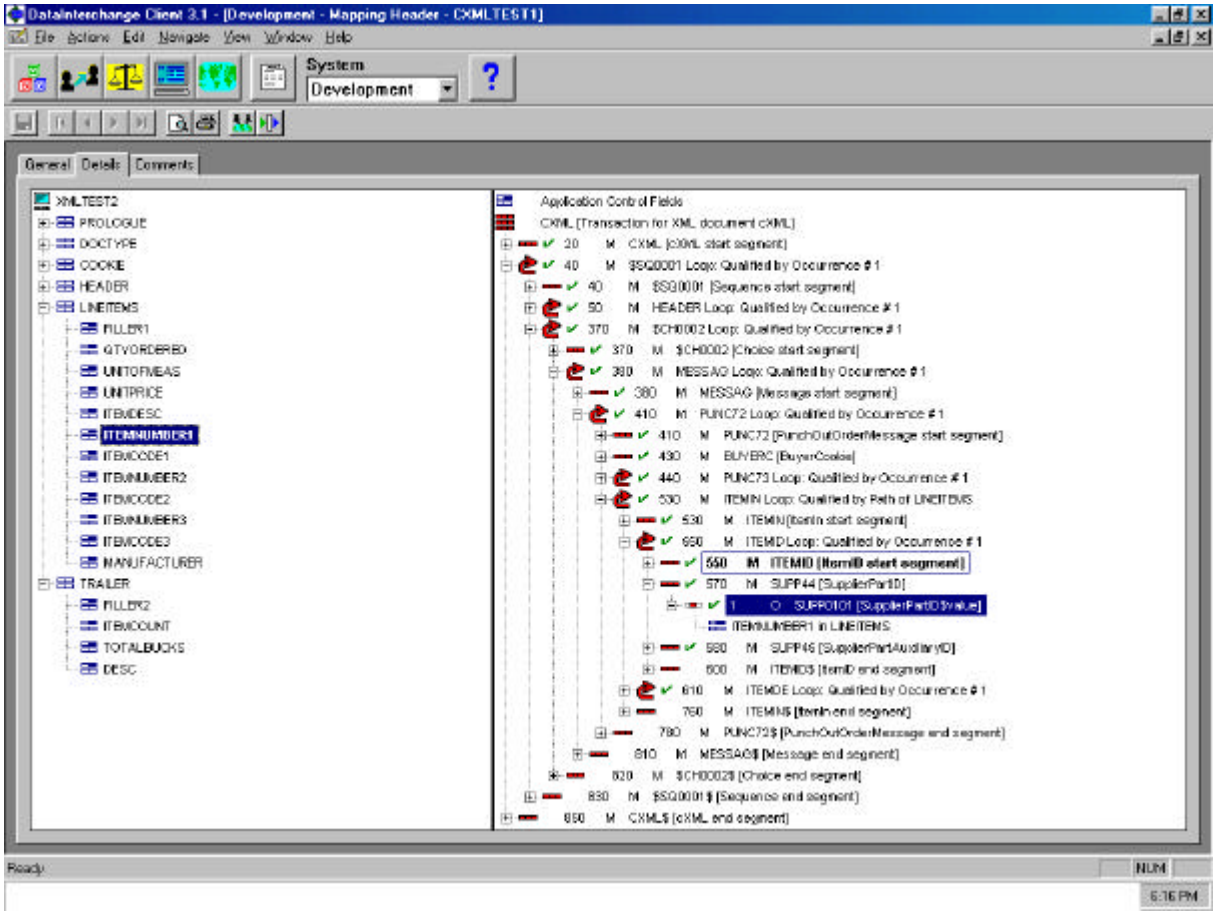
The ItemIn tag is an element with multiple ItemID child elements and also has attributes. Attributes are represented in the EDI standard as data elements with element IDs of segment ID + sequence number and a description of segment ID + '$' + XML attribute name.

In this example the ItemIn loop is equivalent to a line item loop and may be multiple occurrence qualified using a structure or record. If the SupplierPartID element is mapped without qualifying the ItemID loop, DI Client will automatically qualify the ItemID loop with occurrence 1.

Below is an example of an EDI standard transaction representation for an XML DTD :



Below is an example of mapping the SupplierPartID with our previous XML example:




## Overriding the default XML prolog

For outbound processing a new mapping keyword, DIPROLOG, is available to override the XML prolog declaration generated during XML processing. DIPROLOG is a special DI variable and can be created using literal data or from your application data. There is no need to issue an &USE with the DIPROLOG variable.

Below is an example of the default prolog without DTD validation:

```
<?xml version="1.0"?>
```

If DTD validation is specified (validation level = 2), a DOCTYPE declaration is added to the prolog which includes the root element and DTD name. Below is a sample default prolog with DTD validation:

```
<?xml version="1.0"?>
<!DOCTYPE cXML SYSTEM "cXML.dtd">
```

To override the generated prolog declaration with literal values, add the following mapping commands using your literal data:

**&SET DIPROLOG** <?xml version="1.0" encoding="iso-8859-1" ?>
**&SAVE DIPROLOG,*,*** <!DOCTYPE order SYSTEM “order.dtd”>

**Control String generation**

When mapping is complete, generate the control string using DI Client. If you are using DI Client in “stand-alone” mode (not client server), you will need to export the control string along with any trading partner profile or usage setup and upload/import to the host system.

The EDIFACT envelope standard must be loaded into the host system.

## Inbound Translation Process (XML to Data Format)

### Overview

The inbound translation process will be similar to the current EDI to data format (ADF) process flow. The DI Utility will identify XML data as input via a new PERFORM keyword XML(Y/N) which will default to N if not specified. If XML(Y) is specified the Utility will call the XML preprocessor service to convert the XML data into an EDI standard format, including an EDIFACT envelope, and write it to the XML work file XMLWORK. The Utility will continue the inbound process flow as normal through the translation process, and pass the XMLWORK file as the FILEID parm to translation services.

The level of XML validation will be determined by the VALIDATE keyword on the Utility PERFORM command.

- 0 - Indicates that external DTD references should be ignored.
- 1 - Indicates the DTD specified on the DOCTYPE declaration should be processed (e.g.. default attributes, entity references, etc.), but no DTD validation should be done.
- 2 - Indicates full validation against DTD.

The default is 1. All XML data must be well-formed to be processed. Validation level 1 and 2 requires the XMLDTD keyword to be used on the Utility PERFORM commands and that the DTD files are on the host system.

XML processor errors will be written to a file XMLERR and will signal the Utility to also log and write an error message to the Audit trail report. The XML data containing errors will be written to the XMLEXCP file.

XML processor trace messages will be written out to XMLTRC file if allocated. This file is for DI development use only. This file should typically only be allocated when doing problem determination. Allocating this file during normal translation will reduce performance.

Transaction store and management reporting will have no changes and may be used with current implementation. XML data will be represented in EDI standard format. Since Transaction Store will be usable and the XML data is stored in EDI standard format, other outbound commands are supported. The interchange control numbers are generated from the system clock to avoid duplicate envelopes.

## Trading Partner Setup

Since the XML data will be converted to EDI data with an EDIFACT envelope, trading partner identification works the same as with current EDI processing. If a sender ID and/or receiver ID was identified with the DTD conversion utility, the EDIFACT sender ID and receiver ID will contain these values from the XML data.

With traditional inbound EDI processing, the interchange sender ID and qualifier are used to locate a DI trading partner profile, and the profile along with the transaction ID is used to locate the map. The receiver ID can be used for minimal trading partner setup. Please see the DI Administrator's Guide for more information on minimal trading partners. For XML processing the setup is the same as for EDI processing. You set up the trading partner profile and attach a trading partner usage to the map. Some trading partner usage fields do not apply to XML data such as application sender/receiver, functional acknowledgement fields, decryption, and authentication. These EDI type fields should not be used for XML processing and could produce unexpected results.

If the sender and receiver ID and qualifier elements were identified with the DTD conversion utility, the XML processor will retrieve the values from these elements (or attributes) and place them into the interchange header. The sender and receiver ID will each be truncated to 35 characters. The qualifiers will each be truncated to 4 characters.

If a sender ID was not identified with the DTD conversion utility, the sender ID will default to XMLPROC with qualifier ZZ. A trading partner profile and trading partner usage is required for these defaults. If a receiver ID was not identified, the receiver ID will default to XMLRCVR with qualifier ZZ. A trading partner profile and trading partner usage is required for these defaults as needed for minimal trading partner resolution. Please see the DI Administrator's Guide for more information on minimal trading partner setup.

## PERFORM Commands

The following PERFORM commands can be used to process inbound XML data:

PERFORM DEENVELOPE
PERFORM TRANSLATE TO APPLICATION
PERFORM DEENVELOPE AND TRANSLATE

**New PERFORM Keywords**

The following new keywords may be used on the PERFORM commands to control how the inbound XML data is processed:

**XML (Y/N)** - Required for XML processing. The default is N.

- Y - Indicates that input data is XML.
- N - Indicates normal processing (no XML processing)**.**

**XMLVALIDATE(0/1/2)** - Optional. All XML data must be well-formed to be processed. The default is 1.

- 0 - Indicates that external DTD references should be ignored.
- 1 - Indicates that if a DTD is specified on the DOCTYPE declaration, it should be processed (e.g.. default attributes, entity references, etc.), but no DTD validation should be done.
- 2 - Indicates full validation against DTD.

**XMLSTDID()** - Optional for XML processing. Identifies the standard ID that was created with the DTD conversion utility. If this keyword is used, the specified standard ID is used to convert each XML document to a corresponding "EDI standard" envelope and transaction. If this keyword is not specified, then the first 8 alphanumeric characters of the root element name are used as the standard ID for each XML document. The maximum length is 8 characters. Also see the XMLDICT keyword.

**MULTIDOCS(Y/N)** - Optional for XML processing. The default is N.

- Y - Indicates the XML input file contains multiple documents. If Y is specified, the input message must be in EBCDIC and each document must begin with an XML declaration (<?xml...).
- N - Indicates the XML input file contains one document.

**XMLEBCDIC(Y/N)** - Optional for XML processing. The default is Y.

- Y - Indicates the XML data is to be interpreted as EBCDIC.
- N - Indicates that the encoding on XML declaration should be used.

See the section on “XML Encoding Considerations” for more detail.

**XMLSEGINP(Y/N) -** Optional for XML processing. The default is Y.

- Y - Indicates that the record boundaries in the input XML data should be treated as line breaks.
- N - Indicates that the record boundaries in the input XML data should be ignored.

This keyword is ignored for XMLEBCDIC(N). See the section on "Other Considerations" for more detail.

**XMLDTDS()** - Required for XML DTD processing. Identifies the PDS or HFS path where the XML DTD members are located. The maximum length is 64. Please see the "XML DTD Resolution" section of this document for more information.

**XMLDICT()** - Required for XML processing. Identifies the PDS or HFS path where the XML dictionary files generated by the DTD conversion utility are located. The maximum length is 64. Please see the "XML Dictionary Resolution" section of this document for more information.

**Other Considerations**

If you are using XMLEBCDIC(Y), you can also use the XMLSEGINP(Y/N) keyword to control whether the record boundaries on your input XML data are treated as line breaks or ignored.

- If you use XMLSEGINP(Y), then a newline character is assumed at the end of each input record in the XML data. This will give more information for any error messages generated by the XML parser, since it will include more accurate line and column information. However, this requires that the record boundaries only occur where whitespace characters are valid, such as between elements.
- If you use XMLSEGINP(N), then record breaks are ignored and the data is treated as a continuous stream of characters. This allows record breaks to occur anywhere in the data. However, any parser errors will not accurately show the line number, since the entire document is treated as if it were a single line.

For XMLEBCDIC(N), the record boundaries are always ignored, and line numbers are based on the carriage-return or newline characters that appear in the data.

The Interchange Sender ID and Qualifier will default to XMLPROC and ZZ if no sender ID is present in the XML data. This means a Trading Partner profile and Trading Partner receive usage for sender ID XMLPROC must be defined and attached to the XML mapping. The interchange and transaction control numbers will be based on the system clock to avoid duplicate envelopes in Transaction store.

The PERFORM RECEIVE command will receive EDI data only. It is a two step process to receive XML data using IBM Global Network (IGN):

PERFORM RECVFILE

And

PERFORM DEENVELOPE AND TRANSLATE.

It is also a two step process to translate from XML to an EDI standard such as X12 or EDIFACT. First you use this XML processor to translate from XML to application data. Then you translate the application data to EDI .

## Outbound Translation Process (Data Format to XML)

### Overview

The outbound process is similar to the current data format (ADF) to EDI process flow. The translation process will determine the mapping to use with the current trading partner usage lookup. An XML map will be identified using envelope type "L" for the EDI standard transaction. (This will be created when the EDI standard is generated by the DTD conversion utility.) If envelope type is L, each new transaction will force an interchange break to build all EDIFACT envelope segments. The XML processor will convert the standard data to XML data.

XML validation will be determined by the trading partner usage overrides.

- 0 - Will indicate that no checking should be done on the outbound XML data.
- 1 - Will verify that the outbound XML data is well-formed. If a DTD is specified in the DIPROLOG variable, it will be processed for default attributes, parameter entity references, etc. The XML data will not be checked against the DTD for validity.
- 2 - Will validate the outbound XML data against the DTD, in addition to verifying that it is well-formed. This requires the XMLDTD keyword to be used on the Utility PERFORM commands.

Note that the validation of the outbound XML data will cause additional processing, so it may impact performance.

The Trading Partner Profile definition flag (segmented output = Y) will cause XML data to be written out in an easy to read format with line breaks and indentation. If segmented output = N, the data is written as a continuous stream, with no blanks or line breaks inserted.

XML processor errors and status messages will be written to a file XMLERR. If an XML processing error occurs, the processor will signal the Utility to also log and write an error message to the Audit trail report. If the XML data is generated by the XML processor, but an error is found during the validation, the invalid XML data will be written to the XMLEXCP file.

XML processor trace messages will be written out to the XMLTRC file if allocated. This file is for DI development use only. It should typically only be allocated when doing problem determination. Allocating this file during normal translation will reduce performance.

Transaction store and management reporting will have no changes and may be used with current implementation. XML data will be represented in EDI standard format. Since the transaction store will be usable and the XML data is stored in EDI standard format, other outbound commands are supported. The interchange sender and qualifier will be XMLPROC and ZZ. Interchange control numbers will be updated based on trading partner similar to current EDI processing.

## Trading Partner Setup

Since the application data will be converted to EDI data with an EDIFACT envelope as the first step, trading partner identification works the same as with current EDI processing. The internal trading partner ID identified on the RAW data format or the DI 'C' record will be used along with the data format ID to locate the mapping.

For XML processing the interchange sender ID and qualifier will always be XMLPROC and ZZ The receiver ID will be populated using the trading partner profile member information for traditional processing. Minimal trading partner setup may be used. Please see DI Administrator's Guide for more information on minimal trading partners. For XML processing the setup is the same as for EDI processing. You set up the trading partner profile and attach a trading partner usage to the mapping. Some trading partner usage fields do not apply to XML data such as application sender/receiver, functional acknowledgement fields, encryption, security, and authentication. These EDI type fields should not be used for XML processing and could produce unexpected results.

The envelope type field on the trading partner usage override must be 'L' for XML. The envelope type indicates this is an XML trading partner. There is no envelope profile for XML.

## PERFORM Commands

The following PERFORM commands can be used to process outbound XML data:

PERFORM TRANSLATE TO STANDARD
PERFORM ENVELOPE
PERFORM TRANSLATE AND ENVELOPE

## New PERFORM Keywords

The following new keywords may be used on the PERFORM commands to control how the outbound XML data is processed:

**XMLDTDS()** - Required for XML DTD processing. Identifies the PDS or HFS path where the XML DTD members are located. The maximum length is 64. Please see the "XML DTD resolution" section of this document for more information.

**XMLDICT()** - Required for XML processing. Identifies the PDS or HFS path where the XML dictionary files generated by the DTD conversion utility are located. The maximum length is 64. Please see the "XML Dictionary Resolution" section of this document for more information.

**Other Considerations**

The PERFORM SEND command will send EDI data only. It is a two step process to send XML data using IBM Global Network (IGN).

PERFORM TRANSLATE AND ENVELOPE

And

PERFORM SENDFILE

A new mapping variable DIPROLOG will allow the user to map an override for the default XML prolog generated by the XML processor. You can use this if you want to customize the XML prolog information, such as the encoding type or the DOCTYPE declaration. The maximum number of characters that can be saved is approximately 900.

To map from an EDI standard such as X12 or EDIFACT to XML format, there are two options:

- You can map the EDI data to a data format (ADF) using host or DI Client mapping. With this option you will need to map the XML prolog, starting, and ending tags as well as control any formatting wanted such as line breaks and indentation.
- The second option is to use this XML processor to translate from EDI to application data followed by application data to XML translation.

## Other Topics

### XML Dictionary Resolution

The XML dictionary is used by the XML processor to correlate the XML elements and attributes with the EDI standard segments and data elements. It is one of the two files that are generated by the DTD conversion utility when you convert a DTD. The file will be saved in the same directory as the DTD being converted. The filename will be in the format *standard***.DIC**, where *standard* is the standard name being generated.

This file must be uploaded to the host so the XML processor can access it during translation. It should be uploaded using text mode, since it is generated as an ASCII text file. It may be stored on the host as either a PDS member, or as an HFS file. If it is stored as a PDS member, the member name should be the same as the standard name. If it is stored as an HFS file, the filename should be in the format */path/standard***.DIC**, where *path* is the HFS directory for the file, and *standard* is the standard name (in uppercase).

The XMLDICT keyword on the PERFORM command is used to specify the location of this file for the XML processor. If the XMLDICT value starts with a slash ("/") character, the dictionary is assumed to be an HFS file, and the XMLDICT value specifies the directory. If the XMLDICT value does not start with a slash ("/") character, the dictionary is assumed to be a PDS member, and the XMLDICT value specifies the fully qualified PDS name.

For inbound processing, the XMLSTDID keyword identifies the EDI standard (dictionary) name to be used. If the XMLSTDID keyword is not used, the first 8 alphanumeric characters of the root element in the XML data will be used as the EDI standard (dictionary) name.

For outbound processing, the dictionary name is defined in the map (the standard dictionary name used for the map).

For example, if the dictionary name is **CXML** and is located in the PDS EDI.XML.DICT then the dictionary file would be in EDI.XML.DICT(CXML). Your PERFORM command would look something like this:

PERFORM TRANSLATE AND ENVELOPE WHERE XMLDICT(EDI.XML.DICT)

PERFORM DEENVELOPE AND TRANSLATE WHERE XMLSTDID(CXML)
XMLDICT(EDI.XML.DICT)

If the dictionary name CXML is located in HFS directory /u/ediuser, it would be in file /u/ediuser/CXML.DIC. The PERFORM command would look something like:

PERFORM TRANSLATE AND ENVELOPE WHERE XMLDICT(/u/ediuser)

PERFORM DEENVELOPE AND TRANSLATE WHERE XMLSTDID(CXML)
XMLDICT(/u/ediuser)

Note that for HFS files, the path and filenames are case sensitive.

**XML DTD Resolution**

External DTDs can be parsed along with the XML data. The DTD can be used to validate that the XML data conforms to the DTD, and can also be used to resolve things such as default attribute values and parameter entity references. If you want the parser to process an external DTD along with the XML data, then you must upload the DTD file to the host. The file may be stored as either a PDS member or an HFS file.

For send processing (data format to XML), the DTD should be uploaded as text. For receive processing (XML to data format), if you specify XMLEBCDIC(Y) (or use the default of Y), the DTD file should be uploaded as text. If you specify XMLEBCDIC(N), then the DTD file must contain an XML declaration ("<?xml...>") that includes the appropriate encoding type for the file. See the section on XML Encoding Considerations for more information.

If DTD processing is to be done as part of the XML parsing (based on the validation level), then the following processing takes place whenever the parser finds a reference to an external DTD file:

1. When an external DTD reference is found in the XML data, all path information (such as a URL path or file path information) is removed from the DTD name, leaving only a base DTD name.
2. If a DTD alias file (DD:DTDALIAS) is allocated, then the base name is first looked up in the alias file. (See below for more information.)
3. If the base name is found in the alias file, then the alias name is combined with the XMLDTDS value to determine the filename of the DTD file. The search for the base name in the alias file is not case sensitive.
4. If no alias is found (or the DTDALIAS file is not allocated), then the base DTD name is combined with the XMLDTDS value to determine the filename of the DTD file. This DTD file is used by the XML parser in place of the URL specified on the DOCTYPE declaration.

Normally, the XML processor will just take the DTD name (without file or URL path information) and combine it with the XMLDTDS value provided on the PERFORM command. If the XMLDTDS value starts with a "/", then it is assumed to be an HFS path, and the base name is appended to the path. Otherwise, the XMLDTDS value is assumed to be a PDS, and the first 8 characters of the base name (minus the extension) are assumed to be the member name. For example, if the XML data contains the following DOCTYPE declaration:

**<!DOCTYPE PurchaseOrder SYSTEM "http://xyz.org/xml/dtds/MyPO.dtd">**

The DTD name would be resolved as follows:

1. Remove the URL path information, leaving **MyPO.dtd** as the base name.
2. Look up **MyPO.dtd** in the DTDALIAS file. For this example, we will assume it is not found.
3. If the XMLDTDS value starts with a slash ("/"), then the base name is appended to the XMLDTDS value. If XMLDTDS(/u/ediuser/mydtds) is specified, then the DTD file name would be **/u/ediuser/mydtds/MyPO.dtd**.
4. If the XMLDTDS value does not start with a slash, then the base name (without the extension) is assumed to be a PDS name. If XMLDTDS(EDIUSER.MYDTDS) is specified, then the DTD file name would be **EDIUSER.MYDTDS(MYPO)**.

This default behavior may cause conflicts in certain cases, such as with long DTD names or DTD names that differ only in the extension. To resolve conflicts caused by long DTD names, the DTDALIAS file allows you to specify an alias for a DTD name. For example, if you are keeping your DTD files in a PDS, but you have two DTD files: **LongDTDName1.dtd** and **LongDTDName2.dtd** using the first 8 characters would yield the same member name for both: **LONGDTDN**. The DTDALIAS file allows you to specify different member names for each of these. The format is the DTD (base) name followed by one or more blanks, followed by the alias name (Note that HFS file names are case sensitive. PDS member names are not case sensitive.) . For example:

**longdtdname1.dtd LONGN1**
**longdtdname2.dtd LONGN2**

As an example, if you used the above DTDALIAS file, specified the parameter XMLDTDS(EDIUSER.DTDS) on your PERFORM command, and your XML data contained the following DOCTYPE declaration:

```
<!DOCTYPE po SYSTEM "http://xyz.org/xml/dtds/LongDTDName1.dtd">
```

The parser would resolve the DTD as follows:

1. Remove the URL path information from the DTD name, resulting in a base name of **LongDTDName1.dtd**.
2. Look up this base name in the DTDALIAS file. Since this search is not case sensitive, it would find the first entry, and result in an alias name of **LONGN1**.
3. Combine the alias name with the XMLDTDS value. Since **EDIUSER.DTDS** does not start with a '/', it is assumed to be a PDS. Therefore, the member **EDIUSER.DTDS(LONGN1)** will be processed as the DTD file for this XML document.

For outbound processing, the DTD name is in the XML dictionary created by the DTD conversion utility. If you use validation level 2, the DTD name is retrieved from the XML dictionary and used to create the default DOCTYPE declaration. If you use validation level 1, the DTD name will not be included in the default DOCTYPE declaration, but if you override the

default prolog using the DIPROLOG variable and specify a DTD, it will be processed. If you use validation level 0, outbound XML data is not validated using a DTD.

**Utility JCL**

The DI XML processor library (EDI.V3R1M0.SEDIXML1) and XML4C library (such as EDI.XML.SIXMMOD1) from the XML Toolkit must be included in the Utility JCL. For example:

```
//STEPLIB      DD DSN=EDI.V3R1M0.SEDILMD1,DISP=SHR
//             DD DSN=EDI.V3R1M0.SEDIXML1,DISP=SHR
//             DD DSN=DB93.DSNEXIT,DISP=SHR
//             DD DSN=DB93.DSNLOAD,DISP=SHR
//             DD DSN=EDI.SNA131.LOADLIB,DISP=SHR
//             DD DSN=EDI.XML.SIXMMOD1,DISP=SHR
```

The following files may be allocated for XML processing:

- **XMLWORK** - Required. This is a work file or temporary file used for XML inbound and outbound processing. The file allocation for this is similar to FFSWORK. The DataInterchange Utility and XML processor open this file for OUTPUT and INPUT and it should always be empty before the job begins. Use the JCL DISP=OLD to clear this file. It is a prime candidate for a virtual I/O data set.
- **XMLERR** - Required. Error and status messages from the XML processor are written to this file. The record length should be long enough to hold any messages from the XML processor (typically LRECL=255 is adequate), or some messages may be truncated. If you do not allocate this file in your JCL, then you will not be able to see these messages. The file allocation for this is similar to PRTFILE. The XML processor opens the file for OUTPUT. It will normally start writing from the beginning of this file for each new PERFORM command in your job step. Use the JCL DISP options to control whether the file is cleared or appended to. If you are executing multiple PERFORM commands, you may want to use DISP=MOD.
- **XMLTRC** - Optional. If allocated, it contains XML processing trace messages. The file allocation for this is similar to EDITRACE. The XML processor opens the file for OUTPUT. It will normally start writing from the beginning of this file for each transaction on outbound, and each XML input file on inbound. Use the JCL DISP options to control whether the file is cleared or appended to.
- **XMLEXCP** - Required. XML data that is in error is written to this file. If this file is not allocated in the JCL, the XML data that is in error will be discarded. The file allocation for this is similar to FFSEXCP. The XML processor opens the file for OUTPUT when processing the first transaction, and starts writing from the beginning of the file. It then opens the file for extend for each subsequent transaction in the PERFORM command. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. The file should be allocated the same as your XML input file for inbound processing. If you are executing multiple PERFORM commands, you may want to use DISP=MOD.

These files are used in a similar fashion as the related allocation files. The recommendation is to allocate these files the same as the related files.

## Encoding Considerations

The XML processor uses the XML Toolkit for OS/390 to parse the XML data. This parser supports many different character encodings, and follows the XML standards for auto detection of the character encoding. However, this can sometimes cause problems when dealing with EBCDIC data.

According to the XML standard, if the XML document is not in UTF-8 (similar to ASCII for most commonly used characters) or UTF-16 format, it **must** begin with an XML encoding declaration (<?xml...>). For EBCDIC data, the *encoding=* attribute must be present and indicate which encoding type is in use. If the XML data was generated as ASCII data, then converted to EBCDIC, it is likely that the *encoding=* attribute would not be added or updated to reflect the EBCDIC conversion.

Additionally, the newline (EBCDIC x'15') character is not recognized as a valid white space character by the XML standard. However, on MVS this is often inserted into files as a record separator by editors, upload applications, and other MVS applications.

To resolve both of these problems, the XML processor in DI can instruct the parser to override the default (auto detected) encoding type for the XML document with a special encoding type: "ebcdic-xml-us". This causes the parser to ignore the *encoding=* attribute in the XML declaration, and use this special type instead. This encoding type is based on the ibm1140 codepage, and will also treat any newline characters as a carriage-return (x'0A'), which is considered a valid XML whitespace character.

When receiving XML data, the XMLEBCDIC(Y) keyword (which is also the default) instructs the XML processor to override the default encoding type with the special "ebcdic-xml-us' encoding type. This applies to any external DTDs processed, as well as to the XML data itself. This allows the XML processor to handle XML data and DTDs that are in EBCDIC format, but may contain new line characters and/or do not include the encoding type in the XML declaration (or contains an incorrect encoding type). If XMLEBCDIC(N) is specified, then the XML processor will determine the encoding type (for both the data and external DTDs) based on the normal XML auto detection rules. You should typically use XMLEBCDIC(N) if your input XML data is in a format other than EBCDIC, such as ASCII or UTF-16. Since the XMLEBCDIC value is also used to control the interpretation of the DTD files, using XMLEBCDIC(N) also requires that your DTDs contain an XML declaration (<?xml...>) with the proper encoding type.

When sending XML data, the XML data is always generated as EBCDIC. The special "ebcdic-xml-us" encoding type will always be used to check the XML data (and process any external DTDs). No encoding type is specified in the default XML prolog. If you want to send

the data in some other format, such as ASCII or UTF-16, the translation must be done outside of DI. In many cases, it may be done by the transport mechanism, such as FTP, that you use to send the data to another system. If you need to specify an *encoding=* attribute in your XML declaration, you may override the default prolog using the DIPROLOG mapping variable as described in the Mapping section.

Additional information on encoding considerations and the XML Toolkit for OS/390 is on the XML Toolkit for OS/390 web site (http://www.s390.ibm.com/xml/) under the "Usage" section.

## XML Processor Messages and Codes

The following error and status messages are generated by the XML processor:
All messages are in the form: XP*nnnnS*, where:

*nnnn* is replaced by the message number

*S* tells the severity of the message

**I** means that the message is informational only, and is used to give status or to give additional information about other messages.

**W** means that the message describes a warning. The current document is still considered valid, but an unusual condition was found that may result in unexpected output.

**E** means that the message describes an error. The current document is considered invalid.

**T** means that the message describes a severe error that prevents any further processing by the XML processor.

Messages resulting from the XML processor can be found in the XMLERR file. XML data that is in error is written to the XMLEXCP file.

**XP0001I Starting XML processing (*processtype*)**

**Explanation:** The XML processor is starting to process a document or file. The *processtype* indicates whether it is using the preprocessor to convert XML to data format, or if it using the postprocessor to convert data format to XML.

**User Response:** No action required.

**XP0002I Ending XML processing (*processtype*) - status=*status***

**Explanation:** The XML processor has finished processing a document or file, and this message describes the final status. The *processtype* indicates whether it is using the preprocessor to convert XML to data format, or if it using the postprocessor to convert data format to XML.

When processing a single document, the ***status*** indicates the status for the document. Status 0 means the document was successfully processed; status 4 means that warnings were found; status 8 or 12 means that the document was not processed successfully.

When processing multiple input documents in an XML input file, the *status* tells the overall status for the set. Status 0 means that all documents in the file were successfully processed; status 4 means some documents were successful, but others had errors; status 8 means that all documents failed; status 12 means that a serious error occurred, and the XML processor could not continue.

**User Response:** No action required for status 0. For other status levels, check the other messages to determine the cause of the error(s).

**XP0011I Processing document *docnum*, (*size* bytes)**

**Explanation:** When processing multiple input documents in an XML input file, this message tells which document in the file is being processed. The *docnum* indicates the document number within the file (sequential, starting with 1), and the *size* tells the size of the document in bytes.

**User Response:** No action required.

**XP0012I Document *docnum* processed - status=*status***

**Explanation:** When processing multiple input documents in an XML input file, this message tells the status for each document. The *docnum* indicates the document number within the file (sequential, starting with 1), and the *status* tells whether the document was successfully processed. Status 0 means the document was successfully processed; status 4 means that warnings were found; status 8 or 12 means that the document was not processed successfully.

**User Response:** No action required for status 0. For other status levels, check the other messages to determine the cause of the error(s).

**XP0099T Fatal exception - *description***

**Explanation:** A serious error occurred in the XML processor, such as an out of memory condition or a protection exception. The XML processor could not continue.

**User Response:** If the description indicates that it was a memory exception, check to make sure that the process has adequate memory available for the size of the XML input file. If you have a large number of documents in your input file you may be able to process them by splitting the file into smaller pieces.

If the description indicates a "signal" or some other type of exception, then you will probably need to contact your system administrator or your support center.

**XP0101E Error reading XML input file *filename***

**Explanation:** The XML processor could not read the specified *filename* containing the input XML data.

**User Response:** Make sure that the DD name for the input file on your PERFORM command is correct, is allocated in the JCL, and that the job has read access to the file.

**XP0102E Unable to write to the DD:XMLEXCP file**

**Explanation:** The XML processor tried to write invalid XML data to the XMLEXCP file, but was unsuccessful.

**User Response:** Make sure that the XMLEXCP file is allocated in the JCL, and that the job has write access to the file. This error may also occur if you are using segmented input or output, but the record length of the XMLEXCP file is too short for the records.

**XP0103E Unable to write to the DD:XMLWORK file**

**Explanation:** The XML processor tried to write the XML or EDI standard data to the XMLWORK file, but was unsuccessful.

**User Response:** Make sure that the XMLWORK file is allocated in the JCL, and that the job has write access to the file.

**XP0199E File error *operation* file *filename***

**Explanation:** The XML processor encountered an I/O error trying to access the specified file. The *operation* indicates whether it failed opening, reading, or writing to the file. The *filename* tells which file was trying to be accessed.

**User Response:** Make sure that the specified file is allocated in the JCL, and that the job has the appropriate access to the file.

**XP0200I Using XML dictionary file *filename***

**Explanation:** The XML processor is using the file specified by *filename* as the XML dictionary for the current document.

**User Response:** No response is required. This is for information purposes only.

**XP0201E Error reading XML dictionary file *filename***

**Explanation:** The XML processor could not open or read the XML dictionary file. The *filename* tells which file the XML processor was trying to use as the XML dictionary.

**User Response:** Make sure that the specified file exists, and that the job has read access to the file. Also, make sure that your XMLDICT keyword has the right PDS name or HFS path (Note: HFS paths and filenames are case sensitive.) If the XML processor is trying to use the wrong XML dictionary for inbound processing, you may need to specify the XMLSTDID keyword.

**XP0202E Element *element* not found in XML dictionary**

**Explanation:** The specified element from the XML data could not be found in the XML dictionary file. The *element* tells which XML element was not found in the XML dictionary file.

**User Response:** Make sure that the correct XML dictionary file is being used for the data. Some likely causes and solutions are:

- If the XML processor is trying to use the wrong XML dictionary file, you may need to specify the correct dictionary in the XMLSTDID keyword. If no XMLSTDID keyword is specified, the dictionary name is taken from the root element, which may not be the dictionary name you want.
- If the specified element is not in the DTD file that you converted, you will need to either correct the XML data to match the DTD, or reconvert an updated DTD file that contains the specified element. Note that if you update and reconvert the DTD file, a new EDI standard will be generated. You may need to migrate your map to the new standard or remap it.

**XP0203E Error in XML dictionary - *filename***

**Explanation:** The format of the XML dictionary file is invalid. The *filename* tells the name of the file that the XML processor was trying to use.

**User Response:** Make sure that the specified file is actually an XML dictionary file that was generated by the DTD conversion utility. Also, make sure that the file was uploaded to the host as a text file (not binary), so the proper ASCII-EBCDIC conversion took place and the record structure was maintained. If you suspect that the file was edited or corrupted, you may need to upload the dictionary file to the host again.

**XP0204E Segment *segment* not found in XML dictionary**

**Explanation:** The specified segment from the EDI standard data could not be found in the XML dictionary file. The *segment* tells which segment from the EDI standard data was not found.

**User Response:** Make sure that the correct XML dictionary file is being used for the data. If an updated DTD has been converted and mapped, then the new XML dictionary file must also be uploaded to the host.

**XP0205E Invalid syntax in standard data**

**Explanation:** The EDI standard data generated by DI and used as XML processor input is syntactically invalid.

**User Response:** Check your map to verify that the data was mapped correctly. If you cannot resolve the problem, contact your support center.

**XP0206E XML dictionary does not match data for element "*childName*", parent "*parentName*"**

**Explanation:** The XML data does not match the content specification that is in the XML dictionary file. The XML data may be invalid, or the dictionary may need to be updated to reflect changes to the DTD.

Normally, the XML processor does not check the order of the XML elements, check for missing elements, etc. However, when processing special loops such as $SQ and $CH loops, it needs to check the order of the elements more closely. This error indicates that the elements, or the order of the elements in the XML data does not match what was expected based on the XML dictionary.

**User Response:** Check the XML data in the XMLEXCP file. It may be helpful to process this document with validation level 2, since the XML parser may give additional information about why the data does not match the content specification from the DTD file. If the XML document does not pass the DTD validation, correct it and retry. If the document passes the DTD validation, but still does not match the content specification in the XML dictionary file, make sure that the XML dictionary file was generated from the same DTD file that is used for validation. If the DTD file has changed, you may need to regenerate the XML dictionary file and Client import file from the new DTD.

**XP0300E The generated XML data is invalid.**

**Explanation:** The XML data generated by the XML processor is invalid. The XML data will be written to the XMLEXCP file. If the document is using validation level 1, this error means that the data is not well-formed. If the document is using validation level 2, this means that the data did not validate correctly against the DTD. This error is not issued when is using validation level 0, since no validation is performed.

**User Response:** Check the XML data in the XMLEXCP file, along with the other error messages for the document, to determine why the data is invalid. This is typically caused by a mapping error. Some common errors include:

- Errors in the prolog override (DIPROLOG mapping variable).
- Mapping invalid values to XML attributes.
- Not mapping required elements.
- Mapping more than one element of a choice.
- Using an incorrect DTD to validate the data.

**XP0301E Invalid XML data written to the DD:XMLEXCP file**

**Explanation:** The invalid XML data was written to the XMLEXCP file. If the document is using validation level 1 (or 0 for inbound), this error means that the data is not well-formed. If the document is using validation level 2, this means that the data did not validate correctly against the DTD.

**User Response:** You can use the data in the XMLEXCP file, along with other error messages generated for the document, to help determine what the error was. For inbound processing, you can correct the invalid XML documents in the XMLEXCP file, then reprocess the corrected data**.**

**XP0302W Warning issued by XML parser in file *filename*, line *linenum*, column *columnnum***

**Explanation:** The XML parser detected a warning condition in the specified file, line number, and column number. Warning conditions occur when the parser detects an unusual condition, which may or may not represent a problem. Another message will follow this one that contains the text returned by the parser.

If the *filename* value is "xmlbuffer", this means that the warning condition was detected in the current XML document that was being processed. (The document is in memory, so it does not have a real filename.) In some cases, the *filename* may refer to an external DTD file if the warning condition is detected in the DTD file.

The *linenum* and *columnnum* values tell the line and column position where the warning condition was detected. If you are using segmented input or output, the line numbers will normally correspond to the records in the file. If you are not using segmented input or output, the line and column numbers will not reflect the record position in the file, since the record boundaries are ignored.

**User Response:** The document is still processed, but you may want to check your input or output data to verify that the warning condition does not indicate a more significant problem.

**XP0303E Error found by XML parser in file *filename*, line *linenum*, column *columnnum***

**Explanation:** The XML parser detected an error condition in the specified file, line number, and column number. Error conditions typically indicate a problem validating the document against a DTD file. Another message will follow this one that contains the text returned by the parser.

If the *filename* value is "xmlbuffer", this means that the error condition was detected in the current XML document that was being processed. (The document is in memory, so it does not have a real filename.) In some cases, the *filename* may refer to an external DTD file if the error condition is detected in the DTD file.

The *linenum* and *columnnum* values tell the line and column position where the error condition was detected. If you are using segmented input or output, the line numbers will normally correspond to the records in the file. If you are not using segmented input or output, the line and column numbers will not reflect the record position in the file, since the record boundaries are ignored.

**User Response:** You can use the data in the XMLEXCP file to help determine what the error was. For inbound processing, you can correct the invalid XML documents in the XMLEXCP file, then reprocess the corrected data.

**XP0304E Fatal error found by XML parser in file *filename*, line *linenum*, column *columnnum***

**Explanation:** The XML parser detected a "fatal" error condition in the specified file, line number, and column number. Fatal error conditions from the parser typically indicate that the document is not well-formed (i.e. syntactically incorrect). Another message will follow this one that contains the text returned by the parser.

If the *filename* value is "xmlbuffer", this means that the error condition was detected in the current XML document that was being processed. (The document is in memory, so it does not have a real filename.) In some cases, the *filename* may refer to an external DTD file if the error condition is detected in the DTD file.

The *linenum* and *columnnum* values tell the line and column position where the error condition was detected. If you are using segmented input or output, the line numbers will normally correspond to the records in the file. If you are not using segmented input or output, the line and column numbers will not reflect the record position in the file, since the record boundaries are ignored.

**User Response:** You can use the data in the XMLEXCP file to help determine what the error was. For inbound processing, you can correct the invalid XML documents in the XMLEXCP file, then reprocess the corrected data.

**XP0305E No root node found in XML data**

**Explanation:** The XML processor was unable to access the root node in the XML document for an inbound XML document.

**User Response:** Check to make sure that the XML document contains a root node. You can correct the invalid XML document in the XMLEXCP file, then reprocess the corrected data.

**XP0306E Fatal error during XML parser initialization**

**Explanation:** The XML processor was unable to initialize the XML parser. This typically is caused by an installation error in the XML Toolkit for OS/390. Another message will follow this one that contains the text returned by the parser initialization function.

**User Response:** Verify that the XML Toolkit for OS/390 is correctly installed.

**XP0307E Fatal exception during XML parser processing**

**Explanation:** A serious error occurred within the XML parser while trying to parse the document. Another message will follow this one which contains the text returned by the parser. That error will provide more detail on the cause of the error.

**User Response:** If possible, correct the error indicated by the parser message text. If you continue to have problems, contact your system administrator or your support center.

**XP0310I Message text: *text***

**Explanation:** When the XML parser (XML Toolkit for OS/390) detects any type of error or warning, it returns some message text to the XML processor. This message follows other error or warning messages, and will include the text returned by the parser to provide more detail on the cause of the problem.

**User Response:** Use this message text to help diagnose error or warning conditions.

**XP0350I Using DTD file *filename***

**Explanation:** The XML processor is using the file specified by *filename* as the DTD file for the current document.

**User Response:** No response is required. This is for information purposes only. This information may be useful if you need to diagnose DTD problems.

**XP0999E Internal error:** ***error description***

**Explanation:** The XML processor detected an internal error, such as unexpected parameters from DataInterchange or an invalid state within the XML processor. The *error description* gives more information about the error that was detected.

**User Response:** Contact your system administrator or support center.

# XML QUICK REFERENCE

1. Convert the DTD using the conversion utility. This will create an EDI standards import file for DI Client, and an XML dictionary file to be used by the XML processor.

2. Import the EDI standards import file (generated by the utility) into DI Client.

3. Upload the XML dictionary file and DTD file to MVS (PDS member or HFS file).

4. Map the data format to the EDI standard (send map), or the EDI standard to the data format (receive map) using DI Client.

5. Generate a control string for the map.

6. Set up trading partner and usage information.

7. Update/Create DI Utility JCL.

8. Update/Create DTDALIAS file if necessary.

9. Invoke the DI Utility to do the translation and specify some new keywords on the PERFORM command(s).

10. When doing send (data format to XML) processing, DI will translate the data format data to the EDI standard format using the send map, then invoke the XML processor to convert from the EDI standard format to XML.

11. When doing receive (XML to data format) processing, DI will invoke the XML processor to convert the XML data to the EDI standard format, then translate the EDI standard data to the data format using the receive map.