XML Toolkit for z/OS

User's Guide

SA22-7932-03

XML Toolkit for z/OS

User's Guide

SA22-7932-03

Note
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 41.

Fourth Edition, 2004

This edition applies to Version 1 Release 7of XML Toolkit for z/OS (5655-J51) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About this document

This document provides the information you need to use V1.7.0 of XML Toolkit for z/OS™. It contains instructions on how to use the following components:

- XML Parser, C++™ Edition
- XSLT Processor, C++ Edition

Note:: XML Parser, Java Edition and XML Processor, Java Edition are not supported in Toolkit V1.7.0.

Information on how to use Toolkit V1.6.0 and Toolkit V1.5.0 is available from the *XML Toolkit for z/OS and OS/390 User's Guide V1R6* and the *XML Toolkit for z/OS and OS/390 User's Guide V1R5*. Both documents can be downloaded from the following Web site:

`http://www-1.ibm.com/servers/eserver/zseries/software/xml/`

Who should use this User's Guide?

This document is for application programmers, system programmers, and end users working on a z/OS system and using the Toolkit.

This document assumes that readers are familiar with the z/OS system and with the information for z/OS and its accompanying products.

The Toolkit home page,

`http://www.ibm.com/servers/eserver/zseries/software/xml/`

offers information about the Toolkit releases, the Program Directory, and installation instructions.

What is in the User's Guide?

This document describes how to use the Toolkit XML Parser, C++ Edition and XSLT Processor, C++ Edition. Using the document, you will:

- Receive an introduction to XML and the XML Toolkit.
  - Read about XML and its implications in todays world.
  - Learn about the components of the XML Toolkit.
  - Learn about the API's that are implemented by the Toolkit.
  - Read about the process of validation.
  - Discover how to access data sets using XML.
- Understand how to use the XML Parser, C++ Edition in the XML Toolkit.
  - Learn about the C++ XML parser.
  - Review samples of how to use the C++ XML parser.
- Understand how to use the XSLT Processor, C++ Edition in the XML Toolkit.
  - Learn about the C++ XSLT processor.
  - Review samples of how to use the C++ XSLT processor.
- Find out where you can learn more about the XML Toolkit and all its components.

Summary of changes

Summary of Changes
for SA22-7932-03
XML Toolkit Version 1 Release 7

This document contains information previously presented in *XML Toolkit for z/OS User's Guide*, SA22-7932-02, which supports XML Toolkit Version 1 Release 6.

New Information

- New information added to each ″How to use...″ chapter.

Changed Information

Deleted Information

- Removed ″How to use the XML Parser, Java Edition″ chapter.
- Removed ″How to use the XML Processor, Java Edition″ chapter.
- Removed ″How to use the XSLT Processor, Java Edition command line utility″ chapter.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of Changes
for SA22-7932-02
XML Toolkit Version 1 Release 6

This document contains information previously presented in *XML Toolkit for z/OS and OS/390® User's Guide*, SA22-7932-01, which supports XML Toolkit Version 1 Release 5.

New Information

- New information added to each ″How to use...″ chapter.

Changed Information

- Changed information in the ″Introduction″ chapter.
- Restored the ″How to Use the XSLT Processor for z/OS, C++ Edition″ chapter.
- Changed the name of the chapter ″How to use the Lotus-XSL Java command line utility″ to ″How to use the XML Toolkit command line utilities″.

Deleted Information

Summary of Changes
for SA22-7932-01
XML Toolkit Version 1 Release 5

This document contains information previously presented in *XML Toolkit for z/OS and OS/390 User's Guide*, SA22-7932-00, which supports XML Toolkit Version 1 Release 4.

New Information

- Added ″How to use the Lotus-XSL Java command line utility″ chapter.
- Added ″Where to go for more information″ section.
- New information added to ″Introduction″ chapter.
- New information added to each ″How to use...″ chapter.

Changed Information

- Export statements changed in section ″ Building sample applications on the z/OS UNIX® environment″ in Chapter 2.

Deleted Information

- Removed the ″How to Use the XSLT Processor for z/OS, C++ Edition″ chapter.

Chapter 1. Introduction

Why XML?

XML allows you to tag data in a way that is similar to how you tag data when creating an HTML file. XML incorporates many of the successful features of HTML, but was also developed to address some of the limitations of HTML. XML tags may be user-defined through a schema for later validation, which can either be a Document Type Definition (DTD) or a document written in the XML Schema language. In addition, namespaces can help ensure you have unique tags for your XML document. The syntax of XML has more restrictions than HTML, but this results in faster and cheaper browsing. The ability to create your own tagging structure gives you the power to categorize and structure data for both ease of retrieval and ease of display. XML is already being used for publishing, as well as for data storage and retrieval, data interchange between heterogeneous platforms, data transformations, and data displays. As it evolves and becomes more powerful, XML may allow for single-source data retrieval **and** data display.

The benefits of using XML vary but, overall, marked-up data and the ability to read and interpret that data provide the following benefits:

- With XML, applications can more easily read information from a variety of platforms. The data is platform-independent, so now the sharing of data between you and your customers can be simplified.
- Companies that work in the business-to-business (B2B) environment are developing DTDs and schemas for their industry. The ability to parse standardized XML documents gives business products an opportunity to be exploited in the B2B environment.
- XML data can be read even if you do not have a detailed picture of how that data is structured. Your clients will no longer need to go through complex processes to update how to interpret data that you send to them because the DTD or schema gives the ability to understand the information.
- Changing the content and structure of data is easier with XML. The data is tagged so you can add and remove elements without impacting existing elements. You will be able to change the data without having to change the application.

However, despite all the benefits of using XML, there are some things to be aware of. First of all, working with marked up data can be additional work when writing applications because it physically requires more pieces to work together. Given the benefits of using XML, this additional work up front can reduce the amount of work needed to make a change in the future. Second, although it is a recommendation developed by the World Wide Web Consortium (W3C), XML, along with its related technologies and standards including Schema, XPath, and DOM/SAX APIs, are still a developing technology.

APIs

DOM

The Document Object Model (DOM) specification is an object-based interface developed by the World Wide Web Consortium (W3C) that builds an XML document as a tree structure in memory. An application accesses the XML data through the tree in memory, which is a replication of how the data is actually structured. The DOM also allows you to dynamically traverse and update the XML document.

DOM uses a set of C/C++ or Java APIs to interact with the XML data.

The DOM API is ideal when you want to manage XML data or access a complex data structure repeatedly. The DOM API:

- Builds the data as a tree structure in memory.
- Parses an entire XML document at one time.
- Allows applications to make dynamic updates to the tree structure in memory.
- Allows applications to randomly access any item in the memory tree structure.
- Allows applications to generate an XML document by starting with an empty tree, populating it with the desired data, and then serializing it as an XML character document.

Using the DOM API preserves the structure of the document (and the relationship between elements) and does the parsing up front so that you do not have to do the parsing process over again each time you access a piece of data. If you choose to validate your document, you can be assured that the syntax of the data is valid when you are working with it. However, the DOM API requires additional memory to be allocated and freed, initialized and read, translating to increased machine cycles. In addition, the DOM API is, by nature, a four-step process:

1. The application invokes the parser, passing it an XML document.
2. The parser parses the entire document and builds a DOM tree structure in memory.
3. Completion status is returned to the application.
4. The application utilizes DOM APIs to access and optionally modify data in the DOM tree.

The following is a schematic of the DOM parsing model.

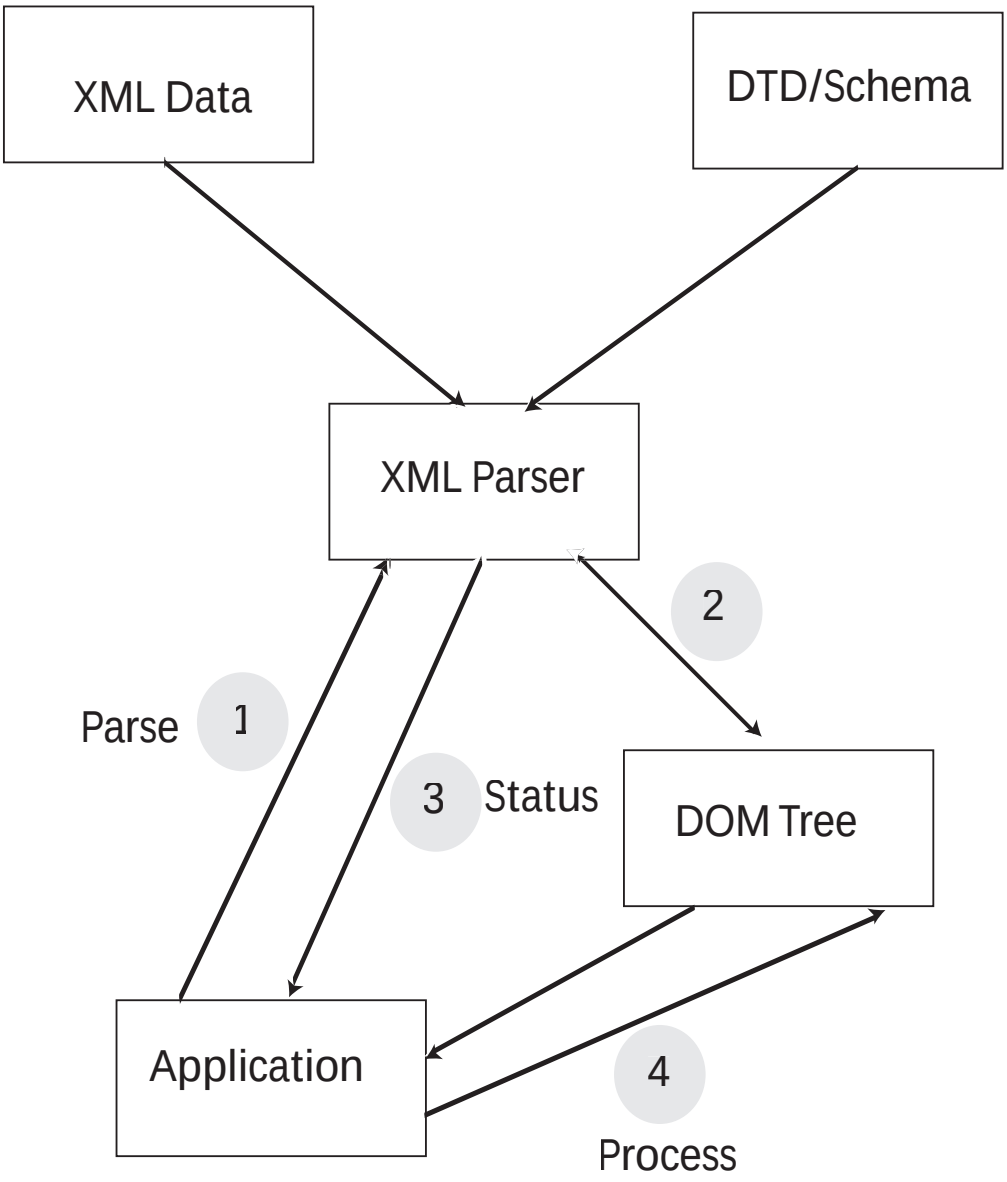


Figure 1. DOM Parsing Model

For information on the Toolkit support for DOM APIs, see the Interfaces and Specifications chart for Toolkit Parser on page 11.

SAX

The Simple API for XML (SAX) specification is an event-based interface developed by members of the XML-DEV mailing list. It uses the parser to access XML data as a series of events in a straight line, which means that the parser finds information in the XML document without retaining state (or context) information.

When writing applications using the SAX specification, you will use a set of C/C++ or Java APIs to interact with the XML data.

The SAX API can provide faster and less costly processing of XML data when you do not need to access all of the data in an XML document. Part of the reason for this performance benefit not seen in DOM arises from the fact that SAX places

more burden on the application than does DOM. Often, applications that might naturally tend to be inclined to use DOM, instead use SAX and work around its limitations, in order to take advantage of those performance benefits. The SAX API does the following:

- Accesses data through a series of events, eliminating the need to build a tree structure in memory.
- Assists the application in determining the most efficient way to build an internal model.
- Allows you to access a small number of elements at one time rather than an entire document.

The SAX API is best for applications that need access to a subset of the data and do not need to understand its relationship to surrounding elements. SAX is also ideal for information that is both generated by and readable by a machine. However, SAX can only traverse the XML document in a single pass, which makes it more expensive when you want to access data repeatedly from an XML document. When it comes to saving needed information from the document or keeping its own understanding of relationships between elements (if that is important), SAX places more burden on the application than does DOM.

The SAX parsing model is a three-step process:

1. The application invokes the parser passing it an XML document. It also passes in the addresses of event handlers for the various SAX events.
2. The parser parses the document, calling the application's event handlers for each token encountered in the XML document.
3. When the document is complete, control is returned to the application.

The following is a schematic of the SAX parsing model.

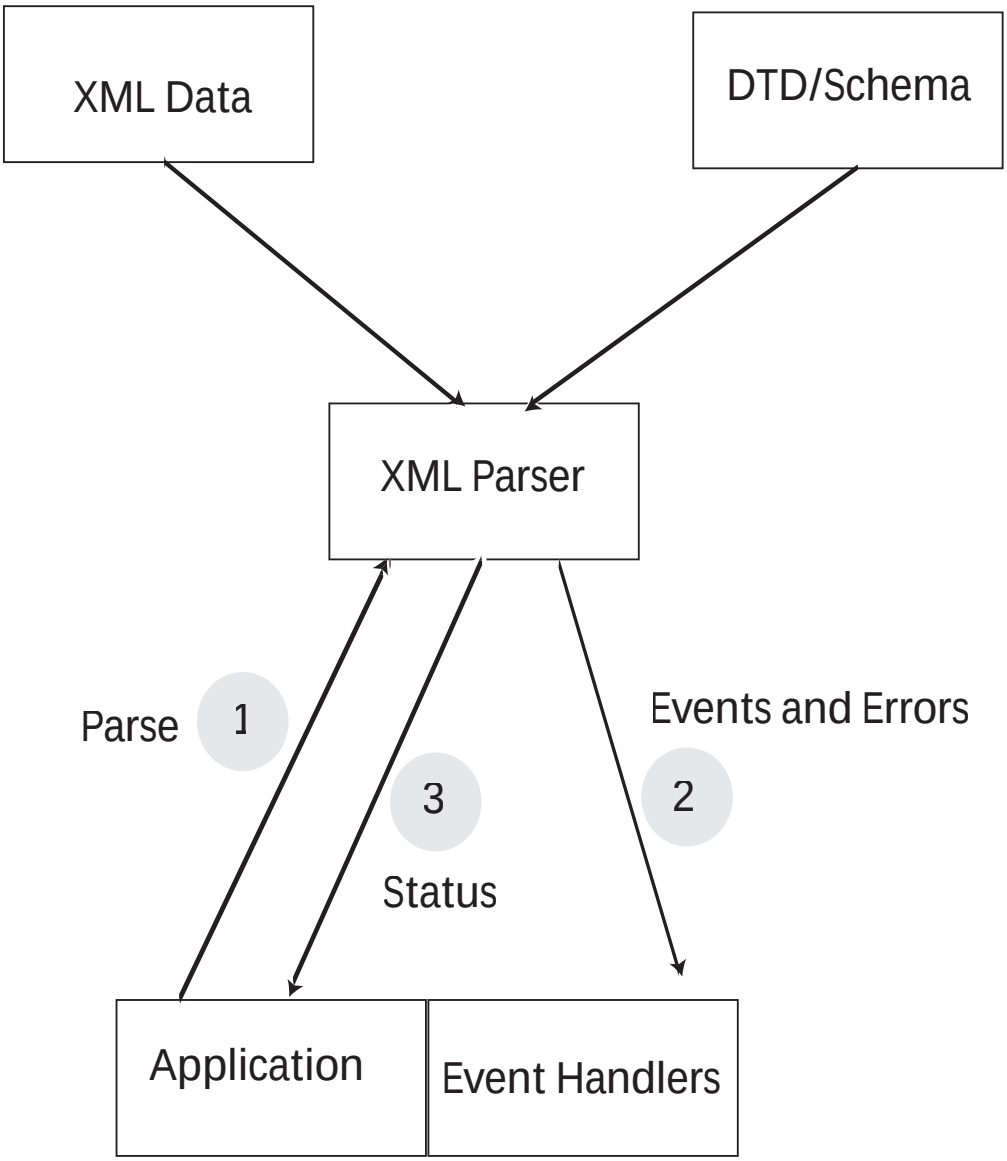


Figure 2. SAX Parsing Model

For information on the Toolkit support for SAX APIs, see the Interfaces and Specifications chart for Toolkit Parser on page 11.

DOM vs SAX

The DOM and SAX APIs can each parse documents efficiently given appropriate conditions. The following table summarizes and compares the characteristics of the DOM API with those of the SAX API:

Table 1. DOM vs SAX

	DOM	**SAX**
Type of Interface	Object based	Event based
Object Model	Created automatically	Must be created by application
Element Sequencing	Preserved	Can be preserved or not, depending on the application

Table 1. DOM vs SAX (continued)

	DOM	SAX
Speed of Initial Data Retrieval	Slower	Faster
Stored Information	Better for complex structures	Better for simple structures
Validation	Optional	Optional
Ability to update XML document	Yes (in memory)	No

XPath

XPath is a language for addressing parts of an XML document, designed to be used by XSLT and other XML-related technologies. It provides basic facilities for manipulation of strings, numbers and booleans. XPath is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern). For information on the Toolkit support for XPath, see the Interfaces and Specifications chart for Toolkit Processors on page 12.

Validation

A valid document is one that follows the XML syntax and also conforms to the rules of an associated DTD or XML Schema. (A well-formed document is one that follows the XML syntax.)

Validation is the process of comparing an XML document with a specified DTD or XML Schema. It ensures that the document uses only those tags that have been defined in the DTD or XML Schema as well as ensuring that it conforms to the element rules specified in the DTD or XML Schema.

Validation of an XML document is expensive in terms of machine cycles. If the document is received from a reliable source and the format of the document has been predetermined, validation may not be necessary. However, using validation ensures that only elements defined in the DTD or XML Schema are used and, therefore, the structure of the XML document remains consistent.

If you do not want to validate the document each time you access data, you can, as an example, code an application so that it may reject tags that it does not recognize and takes an appropriate error path. If you do this, you may want to use validation during testing and initial implementation of a new version of an application or temporarily until the source of a document has been accredited.

The following table summarizes the expected results of validation:

Table 2. Expected Validation Results

	Validate Against a DTD or XML Schema	Do Not Validate
Document Is Valid	Once validation is completed, parsing continues.	Validation is ignored and processing continues.
Document Is Not Valid	Validation will result in an error response that will help you determine the error. Parsing is discontinued.	Validation is ignored and processing continues.

How to access XML data

The XML parser (as well as the XSLT processor) was designed to utilize the Uniform Resource Identifiers (URI) standard to access files. This standard is described in RFC 2396 . Most APIs that need to access data support both absolute URIs and relative URIs, aside from the following exception: The XSLT processor output parameter only supports relative URIs.

How to access data sets

The URI design is based on a hierarchical file system naming scheme. Traditional MVS data set naming schemes do not fit directly within this scheme so some adaptation has been required in order to be able to access data sets from XML. Fortunately, there is a precedent for accessing data sets when running a C++ program from UNIX and that is to prefix the data set name with '//'. Here is an example:

```
//'USER1.SAMPLE.XML(PERSON1)'
//SAMPLE.XML(PERSON1)
```

The '//' tells a C++ program running from the UNIX APIs to look for the name as a data set rather than in an HFS. The single quotes tell it not to add on the user's default high level qualifier. In addition, if you are running from a batch or started task environment, the data set can be accessed via DD statements in the JCL. For example:

```
//DD:SAMPFILE(PERSON1)
```

Where the following DD is also defined in the JCL:

```
//SAMPFILE DD   DSN=USER1.SAMPLE.XML,DISP=SHR,
//      VOL=SER=BPXLK2,UNIT=3390
```

All the examples above will access the same member of the same data set.

Relative URIs

In the cases where relative URIs are allowed, these data set definitions can be used instead of the traditional hierarchical file system parameters. Using this format, there is no 'path' distinction as in a hierarchical system. Here is an example invocation of the SAXCount sample program passing a data set name:

```
SAXCount '//sample.xml(person1)'
```

The quotes are needed so that UNIX doesn't see the parentheses as errors. MVS also has a convention of adding a default high level qualifier if one is present. If you don't want to have the default high level qualifier added on, use single quotes around the data set name and double quotes around the whole parameter:

```
SAXCount "//'user1.sample.xml(person1)'"
```

When used in batch, JCL requires single quotes around the parameters, so you must use a pair of single quotes:

```
PARM='/  //''user1.sample.xml(person2)'''
```

You may have noticed that there is an extra '/' in the beginning of the parameters. This is required by JCL to separate run-time options from the parameters.

Absolute URIs

Data sets can also be specified using absolute URIs. (Note: The XSLT Processor, C++ Edition does not support absolute URIs). Since the convention for accessing data sets is to start with a '//' and this convention is also used to distinguish the absolute URIs with host names, you can only specify an absolute URI using the host format. The host name itself is still optional. Here are some examples:

```
SAXCount 'file:////sample.xml(person1)'
     SAXCount 'file://localhost//sample.xml(person1)'
     SAXCount "file:////'user1.sample.xml(person1)'"
     PARM='/ file:////''user1.sample.xml(person2)'''
```

In addition, when using XML in a batch or started task environment, you can use the **//DD:** format to access a data set that is defined via a DD statement. The following is an example:

```
SAXCount 'file:////dd:sampfile(person1)'
```

Considerations when using the Xalan C++ commands

Most APIs that need to access data support both absolute URIs and relative URIs. The following are some known exceptions:

- The Xalan output parameter only supports relative URIs.
- The Xalan input parameters (for the XML file and the XSL file) support all URIs for UNIX files, but only absolute URIs for MVS data sets.

DTDs, Schema and other embedded files

The conventions described above also apply to files which are referenced within XML documents, such as DTDs. Here is an example xml DOCTYPE statement to access a data set:

```
DOCTYPE personnel SYSTEM "file:////'USER1.SAMPLE.DTD(PERSON1)'"
```

Encoding issues

The promise of XML is that it is portable and works on all platforms. Making this work effectively and efficiently requires program design that takes into account the specific situation pertinent to a particular application (for example, where the document originates, where it is likely to be processed, the performance requirements, the throughput requirements, where the document is stored and how it is likely to be accessed). Proper encoding of XML documents will require thought and consideration at application design time.

The following information is intended to give application programmers guidance on how to deal with encoding of XML documents on z/OS.

Encoding and XML

This section presents the encoding rules in a simple and straightforward manner as background to the discussion of encoding of XML on z/OS. It is not intended to reproduce the detail of the XML 1.0 specification or to cover every possible case.

The XML standard defines encoding fairly rigorously. If the document is not in UTF-8 or UTF-16, the encoding of the document must be specified via the

encoding= attributes on the processing instruction. Also, even though it is possible for the encoding specified via the transport protocol to override the encoding declaration, it is strongly advised that the actual encoding of the document match the encoding specified on the encoding= attribute. Problems can occur if the document is converted from one code page to another without the *encoding=* attribute being changed. There are places where conversion takes place without the knowledge of the application programmer. Examples of these include file transfer using ftp (File Transfer Program) without the binary option and storing files in a database using DRDA.

Whenever possible, avoid letting these types of conversions take place so that mismatches do not occur. The XML parser for z/OS converts the document to Unicode for processing and is capable of handling many different code pages. Also, converting from one code page to another can cause loss of data if there are code points in the original code page that are not present in the target code page. Avoiding conversion prior to calling the parser results in the most efficient (from a performance perspective) and least error-prone solution. Conversion is expensive and if the document is converted before the parser is invoked, two conversions actually occur - once from the original code page and once to Unicode within the parser. Therefore, use the binary option on ftp and equivalent file transfer mechanisms.

XML is intended to be a portable data format. The truly portable encoding is Unicode. Therefore whenever possible, it is best to use Unicode as the encoding for XML documents. However, not all platforms provide easy to use facilities for handling Unicode. As a compromise, ASCII is another portable encoding that is better supported via facilities. It is recommended that XML documents intended for use on other platforms be encoded in US ASCII or UTF-8 or UTF-16. This also provides performance benefits because the XML parser is optimized for these encodings.

XML and z/OS

The current XML W3C recommendation (XML 1.0) specification defines CR (Carriage Return), LF (Line Feed), and the combination CR-LF (Carriage Return followed by Line Feed) as acceptable white space characters. These characters are to be converted to LF by the XML parser. Unfortunately, the XML 1.0 specification does not define NEL (New Line or Next Line) as acceptable.

This presents a problem on z/OS because the most common end-of-line character on z/OS is NEL. The C '\n' string converts to NEL, and editors and file I/O routines in the C runtime insert NEL to indicate end-of-line in byte oriented file systems like the HFS (Hierarchical File System). Therefore, if the XML document is created using C or C++ and the application programmer does not do any special programming to avoid it, the line ending character will be NEL. This is not recommended for XML documents because by nature, they are intended to be portable. The NEL is common on z/OS but not on other platforms and therefore is not portable.

Unfortunately, this means that the application programmer has to be aware of this fact and program around it. There are two options available to programmers writing code to create XML documents.

1. The simplest way to create portable XML documents is to use iconv() to convert them to ASCII or Unicode before sending them out of the application program. The runtime function iconv() will convert the NEL to LF in ASCII and the problem is therefore avoided.

2. Another option is to define a literal for LF and use it instead of the string '\n' to create line breaks. This approach works if the file will not be edited or otherwise manipulated on z/OS (remember, most mainframe editors insert NEL characters!). Also, if this file is edited on z/OS, the document will appear to be a single line (since there aren't any NEL characters in it) and therefore will not be very readable.

Note: This is not an issue in the native MVS environment where file systems are record oriented and typically do not require end of line characters.

If you need to edit or view the file on z/OS, it is best to convert it to ASCII and then use viascii (available at z/OS Unix Tools) to edit it.

For the other case, where the program is processing a received XML document, the situation is more complex. The fastest (and in some cases, the simplest) solution is to not convert the file into EBCDIC. If the file is in ASCII or Unicode, then it will have LF as the end-of-line indicator and there won't be any problem with the line ending. However, this is much more complex for a z/OS application program to deal with. Depending on the specific situation (for example, development/test vs production), conversion may or may not be required. However, the recommendation to avoid conversion if at all possible, still holds, especially in a production environment where the cost of conversion can be prohibitive. For development/test situations, where the file may have to be viewed or edited for debugging purposes, conversion may be the right answer. The parser converts all the data into Unicode so converting the data to EBCDIC after parsing is required. At this point, only data that is required needs to be converted, rather than the entire XML document. Note that converting small strings may be less efficient than converting larger strings. Also, handling Unicode or ASCII data in a z/OS program does require care in programming and isn't always simple. All these factors need to be considered in a set of trade-offs when designing the application.

If the file is in EBCDIC and has been created or modified on a z/OS system, then the line ending character is typically a NEL. The z/OS XML Parser, C++ Edition will accept XML documents that have a NEL as a line termination character. Even though these are non-compliant XML documents, the parser will normalize the line-endings to LF. However, these documents are non-compliant and may not be accepted by parsers on other platforms. In general, EBCDIC is not a portable encoding so IBM does not recommend using EBCDIC for XML documents going between platforms or on the Internet.

Note: XML 1.1, a W3C candidate recommendation updating the XML standard, does support NEL

Avoiding conversion

Most transport protocols have mechanisms to avoid conversions. Here are some of the more common products used for transport and the options to turn off conversion (if they exist). Detailed descriptions of these options and their uses are in the documentation associated with each product.

File Transfer Program (FTP)
The binary option prevents FTP from converting the file.

MQSeries
Do not specify MQGMO_CONVERT option on the MQGET call.

DRDA It is not possible to turn off conversion except by using 'FOR BIT DATA' but

this can have other side effects. The DB2 XML Extender has filters that convert LF to NEL and vice versa to ensure that the document is correct.

XML Toolkit for z/OS

The XML Toolkit for z/OS (Toolkit) provides the base infrastructure to integrate vertical/industry-specific data formats, structures, schemas, and metadata to ensure industry compliance of data representation and content. Some of its key uses include categorizing and tagging data for exchange in disparate environments, as well as transforming ad hoc unstructured data to XML records, enabling you to search, cross-reference, and share records.

The Toolkit includes the XML Parser, C++ Edition (XML Parser, Java Edition is not supported in Toolkit V1.7.0). The XML Parser, C++ Edition is a port of IBM's XML4C parser. It is tested and packaged for use on z/OS. XML4C is based on open source code from the Xerces Apache project of the Apache Software Foundation.

In addition to the parser, Toolkit V1.7.0 also includes the XSLT Processor, C++ Edition (the XSLT Processor, Java Edition is not included in this release of the Toolkit). The XSLT Processor, C++ Edition is a port of IBM's XSLT4C XSLT processor (formerly known as LotusXSL-C++). It is tested and packaged for use on z/OS. The processor is an implementation of the W3C recommendations for XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0. XSLT4C is based on open source code from the Xalan Apache project of the Apache Software Foundation. It allows users to transform XML documents into other XML documents, HTML, or text, and run on multiple platforms.

The Toolkit includes z/OS world-class service and support.

For more information about the Toolkit product, visit the Toolkit Web site at:
`http://www.ibm.com/servers/eserver/zseries/software/xml/`

The following two tables presents a quick summary of the major features found in the XML Toolkit for z/OS. Symbols in the tables have the following meaning:

- ″-″: feature absent;
- ″S″: completely supported;
- ″P″: subset;
- ″X″: experimental.

Table 3. Interfaces and Specifications for the Toolkit Parser

Interfaces and Specifications	**C++ Edition parser**
DOM 1.0	S
DOM 2.0	S
DOM 3.0	P,X
SAX 1.0	S
SAX 2.0	S
IDOM	see below
XML 1.0	S
XML 1.1	X
XML Namespaces 1.0	S
XML Namespaces 1.1	X

Table 3. Interfaces and Specifications for the Toolkit Parser (continued)

Interfaces and Specifications	C++ Edition parser
XML Schema 1.0	S
JAXP 1.0	—
JAXP 1.1	—
JAXP 1.2	—

Note: The experimental IDOM was renamed in XML Toolkit V1.5.0 and is now called the recommended DOM C++ Binding. The former DOM interfaces are now deprecated.

Table 4. Interfaces and Specifications for the Toolkit Processor

Interfaces and Specifications	C++ Edition processor
XSL Transformations	S
XPATH 1.0	S
TRaX	—
XSLTC	N/A

Sample applications are provided with the Toolkit to help demonstrate its features. The procedures required to set up and configure these sample applications for MVS and z/OS UNIX environments are described in the chapters that follow.

Toolkit packaging strategy

The Toolkit V1.7.0 package contains three levels of the Toolkit: V1.7.0, V1.6.0, and V1.5.0. This packaging strategy was developed to compensate for the lack of upward compatibility between Toolkit releases. By making three levels of the Toolkit available to customers (current level plus 2 back levels), support is available to customers using a level of the Toolkit other than the current level. For example, if Toolkit V1.7.0 is the current level, the Toolkit package will have included in it levels V1.7.0, V1.6.0, and V1.5.0. .A customer who may be installing a product that requires Toolkit V1.6.0 can still obtain and install V1.6.0 (assuming they don't already have it) because it is included in the current V1.7.0 Toolkit package.

Toolkit support for both z/OS UNIX System Services and MVS environments

The Toolkit supports applications running on both z/OS UNIX System Services and MVS environments. To better understand how the Toolkit provides this support, you need to recognize the differences between these two types of environments, and the applications supported on them. The following table provides an introductory comparison of these two environments:

Table 5. z/OS UNIX vs. MVS

	z/OS UNIX Environment	MVS Environment
Parallel Processing Model	pthreads	tasks
JES Batch Processing	posix enabled batch	non-posix batch
File access	HFS or data sets	data sets only
Command environment	UNIX shell or BPXBATCH	TSO or batch

For more information on how these environments compare, visit the following Web page:

`http://www.ibm.com/servers/eserver/zseries/zos/unix/release/ncomp2.html`

Chapter 2. How to use the XML Parser, C++ Edition

Samples have been provided to demonstrate the features of the XML Parser, C++ Edition. These samples use simple applications written on top of the SAX and DOM API's. See "Why XML?" on page 1 for more information on the APIs. The following samples can be found in the samples directory:

SAXCount
counts the elements, attributes, spaces and characters in an XML file

SAX2Count
same as SAXCount, except uses SAX 2.0

SAXPrint
parses an XML file and prints it out

SAX2Print
same as SAXPrint, except uses SAX 2.0

DOMCount
counts the elements, attributes, spaces and characters in an XML file

DOMPrint
parses an XML file and prints it out

MemParse
parses XML in a memory buffer, outputting the number of elements and attributes

Redirect
redirects the input stream for external entities

PParse
demonstrates progressive parsing

StdInParse
demonstrates streaming XML data from standard input

EnumVal
shows how to enumerate the markup declarations in a DTD validator

SEnumVal
shows how to enumerate the markup declarations in a Schema validator

CreateDOMDocument
creates a DOM tree in memory from scratch

Rule: These samples are only examples of how to exploit the XML Parser, C++ Edition. You will need to modify your own applications accordingly.

Pre-built versions of the samples for the z/OS UNIX environment are included in the Toolkit. These can be used to illustrate XML concepts, validate XML documents, and validate DTDs and schemas during development. See "Using the sample applications" on page 16 section for instructions on how to use these pre-built versions. Also, source code is provided in the Toolkit for all of the samples to aid developers in getting started with their applications.

The procedures for building and using your built samples differ depending on the target environment. The procedures for building your samples are outlined in sections "Building sample applications for the z/OS UNIX Environment" on page 18 and "Building sample applications for the MVS Environment" on page 20. The

procedures for using your built samples are outlined in sections “Using your sample applications on the z/OS UNIX Environment” on page 20 and “Using your sample applications on the MVS Environment” on page 22

The XML Parser, C++ Edition component is installed in `/usr/lpp/ixm/IBM/xml4c-5_4` by default. It contains the following sub-directories:

/doc contains online APIs and design documentation

/include
used for building samples

/lib used for running the parser code

/bin used for the samples

In addition to the sub-directories, the Toolkit includes the following data sets:

hlq.SIXMLOD1
used for running the parser code in an MVS environment

hlq.SIXMEXP
used to build applications for an MVS environment

Using the sample applications

Note: The pre-built samples can be run in a z/OS UNIX command environment.

Before running the samples, you must ensure that several environment variables are set properly. First, set up an environment variable to point to the location where the XML Toolkit, C++ Parser component was installed:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Then type in the following command statements:

```
export LIBPATH=$XERCESCROOT/lib:$LIBPATH
export PATH=$XERCESCROOT/bin:$PATH
```

You are now set to run the sample applications. For example, to run the DOMPrint application from the *$XERCESCROOT/bin* directory, type the following command statement:

```
cd $XERCESCROOT/samples/data
DOMPrint -v=always -wenc=IBM-1047-s390 -wfpp=on personal.xml
```

This sample application will then parse the personal.xml file, construct the DOM tree, and invoke DOMWriter::writeNode() to serialize the resultant DOM tree back to an XML stream. The following is a sample output from DOMPrint:

```
cd $XERCESCROOT/samples/data
DOMPrint -v=always -wenc=IBM-1047-s390 -wfpp=on personal.xml

<?xml version="1.0" encoding="ibm-1047-s390"?>

<!DOCTYPE personnel SYSTEM "personal.dtd">
<!-- @version: -->
<personnel>
```

```
<person id="Big.Boss">
 <name><family>Boss</family> <given>Big</given></name>
 <email>chief@foo.com</email>
 <link subordinates="one.worker two.worker three.worker four.worker five.worker"></link>
</person>

<person id="one.worker">
 <name><family>Worker</family> <given>One</given></name>
 <email>one@foo.com</email>
 <link manager="Big.Boss"></link>
</person>

<person id="two.worker">
 <name><family>Worker</family> <given>Two</given></name>
 <email>two@foo.com</email>
 <link manager="Big.Boss"></link>
</person>

<person id="three.worker">
 <name><family>Worker</family> <given>Three</given></name>
 <email>three@foo.com</email>
 <link manager="Big.Boss"></link>
</person>

<person id="four.worker">
 <name><family>Worker</family> <given>Four</given></name>
 <email>four@foo.com</email>
 <link manager="Big.Boss"></link>
</person>

<person id="five.worker">
 <name><family>Worker</family> <given>Five</given></name>
 <email>five@foo.com</email>
 <link manager="Big.Boss"></link>
</person>

</personnel>
```

Help for each of the samples can be displayed by using the `-?` parameter. For example, to display help for the `MemParse` sample, type the following:

```
MemParse -?
```

This will display the following text:

```
Usage:
MemParse [options]
This program uses the SAX Parser to parse a memory buffer
containing XML statements, and reports the number of
elements and attributes found.

Options:
-v=xxx Validation scheme [always | never | auto*].
-n Enable namespace processing. Defaults to off.
-s Enable schema processing. Defaults to off.
```

```
-f Enable full schema constraint checking. Defaults to off.
-? Show this help.

* = Default if not provided explicitly.
```

Rule: In order to run the samples, the XML Parser, C++ Edition requires the run-time library provided by Language Environment, SCEERUN, to be made available in the program search order. The best way to do that is by adding SCEERUN data set in the LNKLST. If you do not wish to add SCEERUN to the LNKLST, access SCEERUN data set through STEPLIB.

z/OS UNIX Environment

Building sample applications for the z/OS UNIX Environment

Before being able to build the provided samples, the system environment must be configured correctly. Doing so requires the use of the GNU make utility (gmake). To download gmake go to:

`http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html#gmake`

After you have downloaded gmake, issue the following command against the install file:

```
pax -rzf
```

This will place the gmake program into the `/bin` directory (the `/bin` directory was created by the `pax` command). For additional information on using gmake, see the IBM redbook Open Source Software for OS/390 UNIX, SG24-5944 available online at:

`http://www-1.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html/`

Please note that all references to gmake refer to the GNU make utility.

Product files are required to build the XML Parser, C++ Edition on z/OS UNIX. These files and their descriptions are displayed in the following table:

Table 6. Product Files Required to Build Sample XML Applications for z/OS UNIX Environments

Product file name	Product file description
files in the `include` directory	C++ header files contained in the `include` directory. These are required in order to compile application code.
libxerces-c2_4_0.x	The definition side-deck contained in the `lib` directory that describes the XML Parser, C++ Edition external functions and the variables. This is required in order to bind application code.

Rule: Any application that is to invoke the XML Parser, C++ Edition parser under the z/OS UNIX System Services environment must include libxerces-c2_4_0.x when they bind. The binder uses the definition side-deck to resolve references to functions and variables defined in libxerces-c2_4_0.dll.

The next thing you need to do is set the XML4C root path. To set it correctly, issue the following command statement:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Now, you need to obtain access to a copy of the samples directory to which you have write access. Unless you are a superuser, you normally will not have write access to the samples subdirectory that was shipped with the product. In this case, you will need to do the following:

1. Create a new directory that you have write access to, for example:

   ```
   cd $HOME
   mkdir mysamples
   ```

2. Set a new environment variable that contains the full path to this new directory, as follows:

   ```
   export XERCESCOUT=$HOME/mysamples
   ```

3. Copy the samples directory to your directory:

   ```
   cp -r /usr/lpp/ixm/IBM/xml4c-5_4/samples $XERCESCOUT
   ```

Since the `XERCESCOUT` environment variable is set, that copy of the samples subdirectory will be used. The binary files will be stored in a ″bin″ subdirectory.

After you have copied the samples directory, you need to set up environment variables. This is done through the following sequence:

```
unset _CXX_CXXSUFFIX
export CXX=c++
export CXXFLAGS="-2"
```

If debugging is desired, the *-g* option can be used instead of the *-2* option in the `export CXXFLAGS` statement.

Once the environment variables have been properly set, Makefiles must be created. The directory in which you create the Makefiles depends on where you are building the samples. If you have set the `XERCESCOUT` environment variable, type the following:

```
cd $XERCESCOUT/samples
configure
```

Finally, to build the samples, type the following in the directory in which you created the Makefiles:

```
export _CXX_CXXSUFFIX=cpp
export _CXX_CCMODE=1
gmake
```

After issuing the gmake command, the build process is completed. The samples are built into the `$XERCESCOUT/bin` directory. Proceed to the next section to see how to run your newly built sample applications.

Using your sample applications on the z/OS UNIX Environment

Library files are required to run XML Parser, C++ Edition on z/OS UNIX. These files can be found in the *$XERCESCROOT/lib* directory. The file names and their descriptions are displayed in the following table:

Table 7. Library Files Required to Run Sample XML Applications on z/OS UNIX

Library File Name	Library File Description
libxerces-c2_4_0.dll	the XML Parser, C++ Edition library file
libicudata26.1.dll, libicuuc26.1.dll, libicudata_stub26.1.dll	ICU library files

Before running the samples, you must ensure that several environment variables are set properly. First, set up an environment variable to point to the location where the XML Parser, C++ Edition component was installed:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Then type in the following command statements:

```
export LIBPATH=$XERCESCROOT/lib:$LIBPATH
```

Then set the PATH to locate the samples you have just built:

```
export PATH=$XERCESCOUT/bin:$PATH
```

You are now set to run your sample applications. For example, to run the SAXCount application from the *$XERCESCOUT/bin* directory, type the following command statement:

```
SAXCount $XERCESCROOT/samples/data/personal.xml
```

This sample application will then count the number of elements, attributes, spaces and characters in the XML file personal.xml .

MVS Environment

Building sample applications for the MVS Environment

Before being able to build the provided samples, the system environment must be configured correctly. Doing so requires the use of the GNU make utility (gmake). To download gmake go to:

```
http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html#gmake
```

After you have downloaded gmake, issue the following command against the install file:

```
pax -rzf
```

This will place the gmake program into the `/bin` directory (the `/bin` directory was created by the `pax` command). For additional information on using gmake, see the IBM redbook *Open Source Software for OS/390 UNIX*, SG24-5944 available online at:

http://www-1.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html

Please note that all references to gmake refer to the GNU make utility.

Product files are required to build the XML Parser, C++ Edition on MVS. These files and their descriptions are displayed in the following table:

Table 8. Product Files Required to Build Sample XML Applications for MVS Environments

Product file name	Product file description	Data set name
files in the `include` directory	C++ header files contained in the `include` directory. These are required in order to compile application code.	
IXM4C54X	definition side-deck that describes the XML Parser, C++ Edition functions and the variables	*hlq*.SIXMEXP

Rule: Any application that is to invoke the XML Parser, C++ Edition parser under the native MVS environment must include the IXM4C54 definition side-deck when they bind. The binder uses the definition side-deck to resolve references to functions and variables defined in the IXM4C54.

To be able to run the sample applications, you must first allocate a data set to hold the executables. The following is an example of a data set allocation:

```
userid.SAMPLES.rel.LOAD, 100 tracks on 3390, Record format:U,
Record Length: 0, Block size: 32760, ORG: PDSE,
Directory blocks: 0
```

Rule: If you are building samples from multiple releases, you will need a unique PDSE for each release. For example, SAMPLES.**V160** .LOAD for samples from Toolkit V1.6.0 and SAMPLES.**V170** .LOAD for samples from Toolkit V1.7.0.

The next thing you need to do is set the XML4C root path. To set it correctly, issue the following command statement:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Now, you need to obtain access to a copy of the samples directory to which you have write access. Unless you are a superuser, you normally will not have write access to the samples subdirectory that was shipped with the product. In this case, you will need to do the following:

1. Create a new directory that you have write access to, for example:

   ```
   cd $HOME
   mkdir mysamples
   ```

2. Set a new environment variable that contains the full path to this new directory, as follows:

   ```
   export XERCESCOUT=$HOME/mysamples
   ```

3. Copy the samples directory to your directory:

```
cp -r /usr/lpp/ixm/IBM/xml4c-5_4/samples $XERCESCOUT
```

Since the XERCESCOUT environment variable is set, that copy of the samples subdirectory will be used. The binary files are stored in the MVS data set pointed to by the LOADMOD environment variable. If XERCESCOUT is not set, the copy of the samples in the HFS that the product resides in will be used, and the binary files will be stored there.

After you have copied the samples directory, you need to set up environment variables. This is done through the following sequence:

```
export LOADMOD=userid.SAMPLES.rel.LOAD
export LOADEXP=hlq.SIXMEXP
export OS390BATCH=1
unset _CXX_CXXSUFFIX
export CXX=c++
export CXXFLAGS="-2"
```

If debugging is desired, the *-g* option can be used instead of the *-2* option in the `export CXXFLAGS` statement.

Once the environment variables have been properly set, Makefiles must be created. Type the following:

```
cd $XERCESCOUT/samples
configure
```

.

You are now ready to build the samples. Type the following in the directory in which you created the Makefiles:

```
export _CXX_CXXSUFFIX=cpp
export _CXX_XSUFFIX_HOST=SIXMEXP
export _CXX_CCMODE=1
gmake
```

After you have issued the gmake command, the build process is completed. The built samples are placed into the *userid*.SAMPLES.*rel*.LOAD data set. Proceed to the next section to see how to run your newly built sample applications.

Using your sample applications on the MVS Environment

Library files are required to run XML Parser, C++ Edition on MVS. The following table is a list of library files required, a short description of the files, and the data set names of where these files are located.

Table 9. Library Files Required to Run Sample XML Applications on MVS

Library file name	Library file description	Library data set name
IXM4C54	XML Parser, C++ Edition library file	*hlq*.SIXMLOD1
IXMI26UC	ICU library file (explicitly loaded by IXM4C54)	*hlq*.SIXMLOD1

Table 9. Library Files Required to Run Sample XML Applications on MVS (continued)

Library file name	Library file description	Library data set name
IXMI26DA	ICU library file (explicitly loaded by IXM4C54)	*hlq*.SIXMLOD1
IXMI26D1	ICU library file (explicitly loaded by IXM4C54)	*hlq*.SIXMLOD1

Before you run the samples, you must make sure that you have access to the library, SIXMLOD1. You can ask your system programmer to install SIXMLOD1 in LNKLST. If the SIXMLOD1 data set cannot be placed in LNKLST, you can STEPLIB the data set for each application that requires it. You can invoke the samples from TSO or a JCL job. For example, you can submit the following JCL to run SAXCount.

```
//USERJOB  JOB  MSGLEVEL=(1,1),CLASS=A
//TEST     EXEC PGM=SAXCOUNT,
//*  HFS file input
//   PARM='//usr/lpp/ixm/IBM/xml4c-5_4/samples/data/personal.xml'
//*
//*  DDNAME input
//*  PARM='///DD:XMLDATA(PERSONAL)'
//*  PARM='DD:XMLDATA(PERSONAL)'
//*
//*  Data set input
//*  PARM='"//''USERID.XML.DATA(PERSONAL)''"'
//*  PARM='"//XML.DATA(PERSONAL)"'
//*
//STEPLIB  DD   DSN=hlq.SIXMLOD1,DISP=SHR
//         DD   DSN=userid.SAMPLES.rel.LOAD,DISP=SHR
//*XMLDATA DD   DSN=userid.XML.DATA,DISP=SHR
/*
```

Multi-threading considerations

The following are multi-threading considerations for the XML Parser, C++ Edition.

Using UNIX pthreads

Within a program, an instance of the parser may be used without restriction from a single thread, or an instance of the parser can be accessed from multiple threads, provided the application guarantees that only one thread has entered a method of the parser at any one time.

When two or more parser instances exist in a process, the instances can be used concurrently, without external synchronization. That is, in an application containing two parsers and two threads, one parser can be running within the first thread concurrently with the second parser running within the second thread.

Similar rules apply to XML4C DOM documents. Multiple document instances may be concurrently accessed from different threads, but any given document instance can only be accessed by one thread at a time.

DOMStrings allow multiple concurrent readers. All DOMString const methods are thread safe, and can be concurrently entered by multiple threads. Non-const

DOMString methods, such as `appendData()`, are not thread safe and the application must guarantee that no other methods (including const methods) are executed concurrently with them.

The application also needs to guarantee that only one thread has entered either the method `XMLPlatformUtils::Initialize()` or the method `XMLPlatformUtils::Terminate()` at any one time.

Using MVS multi-tasking

Care must be taken when using the parser in a multi-tasking environment within a single address space. Each task that wishes to use a parser must initialize its own parser environment via a call to `XMLPlatformUtils::Initialize()`. It follows then that each task must have its own parser instance and cannot share parser data structures, such as DOMString.

Chapter 3. How to use the XSLT Processor, C++ Edition

Samples have been provided to demonstrate the features of the XSLT Processor, C++ Edition. These samples use simple applications written on top of the SAX, DOM, and Xalan API's. See "Why XML?" on page 1 for more information on the APIs. The following samples can be found in the samples directory:

CompileStylesheet
: use a compiled stylesheet to perform a series of transformations

DocumentBuilder
: programmatically constructs an XML document, applies the `foo.xsl` stylesheet to this document, and writes the output to `foo.out`

ExternalFunctions
: implements, installs, and illustrates the usage of three extension functions

ParsedSourceWrappers
: performs a transformation with input in the form of a pre-built XercesDOM or XalanSourceTree

SerializeNodeSet
: serializes the node set returned by the application of an XPath expression to an XML document

SimpleTransform
: uses the `foo.xsl` stylesheet to transform `foo.xml`, and writes the output to `foo.out`

SimpleXPathAPI
: uses the XPathEvaluator interface to evaluate an XPath expression from the specified context node of an XML file and displays the nodeset returned by the expression

SimpleXPathCAPI
: uses the XPathEvaluator C interface to evaluate an XPath expression and displays the string value returned by the expression

StreamTransform
: processes character input streams containing a stylesheet and an XML document, and writes the transformation output to a character output stream

TraceListen
: trace events during a transformation

TransformToXercesDOM
: performs a simple transformation but puts the result in a Xerces DOMDocument

UseStylesheetParam
: set a stylesheet parameter that the stylesheet uses during the transformation

XalanTransform
: uses the XalanTransformer class and the associated C++ API to apply an XSL stylesheet file to an XML document file and write the transformation output to either an output file or to a stream

XalanTransformerCallback
: returns transformation output in blocks to a callback function, which writes the output to a file

XPathWrapper
use this sample to find out what a given XPath expression returns from a given context node in an XML file

Rule: These samples are only examples of how to exploit the XSLT Processor, C++ Edition. You will need to modify your own applications accordingly.

Pre-built versions of the samples for the z/OS UNIX environment are included in the Toolkit. These can be used to illustrate XML concepts, validate XML documents, and validate DTDs and schemas during development. See “Using the sample applications” on page 26 section for instructions on how to use these pre-built versions.

The procedures for building and using your built samples differ depending on the target environment. The procedures for building your samples are outlined in sections “Building sample applications for the z/OS UNIX Environment” on page 27 and “Building sample applications for the MVS Environment” on page 30. The procedures for using your built samples are outlined in sections “Using your sample applications on the z/OS UNIX Environment” on page 29 and “Using your sample applications on the MVS Environment” on page 32

The XSLT Processor, C++ Edition component is installed in `/usr/lpp/ixm/IBM/xslt4c-1_7` by default. It contains the following sub-directories:

/doc contains online APIs and design documentation

/include
used for building samples

/lib used for running the processor code

/bin used for the samples

In addition to the sub-directories, the Toolkit includes the following data sets:

hlq.SIXLMOD1
used for running the processor code in an MVS environment

hlq.SIXMEXP
used to build applications for an MVS environment

Using the sample applications

Set up an environment variable to point to the location where the XSLT Processor, C++ Edition component was installed:

```
export XALANCROOT=/usr/lpp/ixm/IBM/xslt4c-1_7
```

You also need to set up an environment variable to point to the location where the XML Parser, C++ Edition component was installed. Here is how you do that:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Next, type in the following command statements:

```
export LIBPATH=$XALANCROOT/lib:$XERCESCROOT/lib:$LIBPATH
export PATH=$XALANCROOT/bin:$PATH
```

You must now copy the sample files to a temporary directory. Here is how you do that:

```
mkdir $HOME/xslsamples
cd $HOME/xslsamples
cp $XALANCROOT/samples/SimpleTransform/foo.* .
```

You are now set to run the sample applications. For example, to run the SimpleTransform application, in the $XALANCROOT/samples/SimpleTransform/ directory type the following:

```
SimpleTransform
```

This sample application will then use the `foo.xsl` stylesheet to transform `foo.xml`, and write the output to `foo.out`. The pre-built samples can be run in a z/OS UNIX command environment.

Rule: In order to run the samples, the XSLT Processor, C++ Edition requires the run-time library provided by Language Environment, SCEERUN, to be made available in the program search order. The best way to do that is by adding SCEERUN data set in the LNKLST. If you do not wish to add SCEERUN to the LNKLST, access SCEERUN data set through STEPLIB.

z/OS UNIX Environment

Building sample applications for the z/OS UNIX Environment

Next, the system environment must be configured correctly. Doing so requires the use of the GNU make utility (gmake). To download gmake go to:

`http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html#gmake`

After you have downloaded gmake, issue the following command against the install file:

```
pax -rzf
```

This will place the gmake program into the `/bin` directory (the `/bin` directory was created by the `pax` command). For additional information on using gmake, see the IBM redbook Open Source Software for OS/390 UNIX, SG24-5944 available online at:

`http://www-1.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html`

Please note that all references to gmake refer to the GNU make utility.

Product files are required to build the XSLT Processor, C++ Edition on z/OS UNIX. These files and their descriptions are displayed in the following table:

Table 10. Product Files Required to Build Sample XML Applications for z/OS UNIX Environments

Product file name	Product file description
libxslt4c.1_7_0.x	the definition side-deck that describes the XSLT Processor, C++ Edition functions and the variables

Table 10. Product Files Required to Build Sample XML Applications for z/OS UNIX Environments (continued)

Product file name	Product file description
libxerces-c2_4_0.x	The definition side-deck contained in the `lib` directory that describes the XSLT Processor, C++ Edition external functions and the variables. This is required in order to bind application code.

Rule: Any application that is to invoke the XSLT Processor, C++ Edition parser under the z/OS UNIX System Services environment must include libxalan-c1_7_0.x and libxerces-c2_4_0.x when they bind. The binder uses the definition side-deck to resolve references to functions and variables defined in libxalan-c1_7_0.dll and libxerces-c2_4_0.dll.

Now set up an environment variable to point to the location where the XSLT Processor, C++ Edition component was installed:

```
export XALANCROOT=/usr/lpp/ixm/IBM/xslt4c-1_7
```

You also need to set up an environment variable to point to the location where the XML Parser, C++ Edition component was installed. Here is how you do that:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Next, you need to obtain access to a copy of the samples directory to which you have write access. Unless you are a superuser, you normally will not have write access to the samples subdirectory that was shipped with the product. In this case, you will need to do the following:

1. Create a new directory that you have write access to, for example:

   ```
   cd $HOME
   mkdir mysamples
   ```

2. Set a new environment variable that contains the full path to this new directory, as follows:

   ```
   export XALANCOUT=$HOME/mysamples
   ```

3. Copy the samples directory to your directory:

   ```
   cp -r /usr/lpp/ixm/IBM/xslt4c-1_7/samples $XALANCOUT
   ```

Since the `XALANCOUT` environment variable is set, that copy of the samples subdirectory will be used. The binary files will be stored in a ″bin″ subdirectory.

After you have copied the samples directory, you need to set up environment variables. This is done through the following sequence:

```
export CXX=c++
export CXXFLAGS="-2"
```

If debugging is desired, the *-g* option can be used instead of the *-2* option in the `export CXXFLAGS` statement.

Next, you need to set up some information for the non-XPLINK Standard C++ Library sidedeck. Here is how you do that:

```
export _CXX_PSYSIX=\
"{_PLIB_PREFIX}.SCEELIB(C128N)":\
"{_CLIB_PREFIX}.SCLBSID(IOC,IOSTREAM,COMPLEX,COLL)"
```

Rule:: All three segments of the above example must be entered on the same command line.

where {_PLIB_PREFIX} and {_CLIB_PREFIX} are set to a default (for example, CEE and CBC, respectively) during custom installation, or using user overrides.

Finally, to build the samples, type the following in the directory in which you created the Makefiles:

```
export _CXX_CXXSUFFIX=cpp
export _CXX_CCMODE=1
cd $XALANCOUT/samples
gmake
```

After issuing the gmake command, the build process is completed. The samples are built into the `$XALANCOUT/bin` directory.

Using your sample applications on the z/OS UNIX Environment

Library files are required to run XSLT Processor, C++ Edition on z/OS UNIX. These files can be found in the *$XALANCROOT/lib* and *$XERCESCROOT/lib* directories. The file names and their descriptions are displayed in the following table:

Table 11. Library Files Required to Run Sample XML Applications on z/OS UNIX

Library File Name	Library File Description
libxslt4c.1_7_0.dll, libxslt4cMessages.1_7_0.dll	XSLT Processor, C++ Edition library files
libxerces-c2_4_0.dll	XML Parser, C++ Edition library file
libicuuc26.1.dll, libicudata_stub26.1.dll, libicui18n26.1.dll	ICU library files

Before running the samples, you must ensure that several environment variables are set properly. First, set up an environment variable to point to the location where the XSLT Processor, C++ Edition component was installed:

```
export XALANCROOT=/usr/lpp/ixm/IBM/xslt4c-1_7
```

You also need to set up an environment variable to point to the location where the XML Parser, C++ Edition component was installed. Here is how you do that:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Then type in the following command statements:

```
export LIBPATH=$XALANCROOT/lib:$XERCESCROOT/lib:$LIBPATH
export ICU_DATA=$XERCESCROOT/lib
```

Then set the PATH to locate the samples you have just built:

```
export PATH=$XALANCOUT/bin:$PATH
```

You are now set to run your sample applications. For example, to run the SimpleTransform application from the *$XALANCOUT/bin* directory, type the following command statement:

```
cd $XALANCOUT/samples/SimpleTransform
SimpleTransform
```

This sample application will then use the `foo.xsl` stylesheet to transform `foo.xml`, and write the output to `foo.out`.

MVS Environment

Building sample applications for the MVS Environment

Before being able to build the provided samples, the system environment must be configured correctly. Doing so requires the use of the GNU make utility (gmake). To download gmake go to:

`http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html#gmake`

After you have downloaded gmake, issue the following command against the install file:

```
pax -rzf
```

This will place the gmake program into the `/bin` directory (the `/bin` directory was created by the `pax` command). For additional information on using gmake, see the IBM redbook *Open Source Software for OS/390 UNIX*, SG24-5944 available online at:

`http://www-1.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html`

Please note that all references to gmake refer to the GNU make utility.

Product files are required to build the XSLT Processor, C++ Edition on MVS. These files and their descriptions are displayed in the following table:

Table 12. Product Files Required to Build Sample XML Applications for MVS Environments

Product file name	Product file description	Data set name
files in the `include` directory	C++ header files contained in the `include` directory. These are required in order to compile application code.	
IXMLC17X	Definition side-deck that describes the XSLT Processor, C++ Edition functions and variables	*hlq*.SIXMEXP
IXM4C54X	definition side-deck that describes the XML Parser, C++ Edition functions and the variables	*hlq*.SIXMEXP

Rule: Any application that is to invoke the XSLT Processor, C++ Edition parser under the native MVS environment must include the IXMLC17X and IXM4C54X definition side-decks when they bind. The binder uses the definition side-decks to resolve references to functions and variables defined in the IXMLC17 and IXM4C54.

To be able to run the sample applications, you must first allocate a data set to hold the executables. **If you have already allocated a data set for XML Parser, C++ Edition, skip this step.** The following is an example of a data set allocation:

```
userid.SAMPLES.rel.LOAD, 150 tracks on 3390, Record format:U,
Record Length: 0, Block size: 32760, ORG: PDSE,
Directory blocks: 0
```

Rule: If you are building samples from multiple releases, you will need a unique PDSE for each release. For example, SAMPLES.**V160** .LOAD for samples from Toolkit V1.6.0 and SAMPLES.**V170** .LOAD for samples from Toolkit V1.7.0.

Next, you must ensure that several environment variables are set properly. First, set up an environment variable to point to the location where the XSLT Processor, C++ Edition component was installed:

```
export XALANCROOT=/usr/lpp/ixm/IBM/xslt4c-1_7
```

You also need to set up an environment variable to point to the location where the XML Parser, C++ Edition component was installed. Here is how you do that:

```
export XERCESCROOT=/usr/lpp/ixm/IBM/xml4c-5_4
```

Then, you need to obtain access to a copy of the samples directory to which you have write access. Unless you are a superuser, you normally will not have write access to the samples subdirectory that was shipped with the product. In this case, you will need to do the following:

1. Create a new directory that you have write access to, for example:

   ```
   cd $HOME
   mkdir mysamples
   ```

2. Set a new environment variable that contains the full path to this new directory, as follows:

   ```
   export XALANCOUT=$HOME/mysamples
   ```

3. Copy the samples directory to your directory:

   ```
   cp -r /usr/lpp/ixm/IBM/xslt4c-1_7/samples $XALANCOUT
   ```

Since the `XALANCOUT` environment variable is set, that copy of the samples subdirectory will be used. The binary files are stored in the MVS data set pointed to by the LOADMOD environment variable.

After you have copied the samples directory, you need to set up environment variables. This is done through the following sequence:

```
export LOADMOD=userid.SAMPLES.rel.LOAD
export LOADEXP=hlq.SIXMEXP
export OS390BATCH=1
export CXX=c++
export CXXFLAGS="-2"
```

If debugging is desired, the *-g* option can be used instead of the *-2* option in the `export CXXFLAGS` statement.

Next, you need to set up some information for the non-XPLINK Standard C++ Library sidedeck. Here is how you do that:

```
export _CXX_PSYSIX=\
"{_PLIB_PREFIX}.SCEELIB(C128N)":\
"{_CLIB_PREFIX}.SCLBSID(IOC,IOSTREAM,COMPLEX,COLL)"
```

Rule:: All three segments of the above example must be entered on the same command line.

where {_PLIB_PREFIX} and {_CLIB_PREFIX} are set to a default (for example, CEE and CBC, respectively) during custom installation, or using user overrides. You are now ready to build the samples. The following sequence shows how to build the samples:

```
export _CXX_CXXSUFFIX=cpp
export _CXX_XSUFFIX_HOST=SIXMEXP
export _CXX_CCMODE=1
gmake
```

After you have issued the gmake command, the build process is now completed. The built samples are placed into the *userid*.SAMPLES.*rel*.LOAD data set.

Using your sample applications on the MVS Environment

Library files are required to run XSLT Processor, C++ Edition on MVS. The following table is a list of library files required, a short description of the files, and the data set names of where these files are located.

Table 13. Library Files Required to Run Sample XML Applications on MVS

Library file name	Library file description	Library data set name
IXMLC17	XSLT Processor, C++ Edition library file	*hlq*.SIXMLOD1
IXMMSG17	XSLT Processor, C++ Edition message handling	*hlq*.SIXMLOD1
IXM4C54	XML Parser, C++ Edition library file	*hlq*.SIXMLOD1
IXMI26UC	ICU library file	*hlq*.SIXMLOD1
IXMI26DA	ICU library file	*hlq*.SIXMLOD1
IXMI26D1	ICU library file	*hlq*.SIXMLOD1
IXMI26IN	ICU library file	*hlq*.SIXMLOD1

Before you run the samples, you must make sure that you have access to the library, SIXMLOD1. You can ask your system programmer to install SIXMLOD1 in LNKLST. If the SIXMLOD1 data set cannot be placed in LNKLST, you can STEPLIB

the data set for each application that requires it. You can invoke the samples from TSO or a JCL job. For example, you can submit the following JCL to run TRACELSN.

```
//USERJOB  JOB  MSGLEVEL=(1,1),CLASS=A
//TEST1    EXEC PGM=TRACELSN,
//*  HFS file input
//   PARM='/-tt'
//STEPLIB  DD   DSN=hlq.SIXMLOD1,DISP=SHR
//         DD   DSN=userid.SAMPLES.rel.LOAD,DISP=SHR
```

Chapter 4. How to use the XML Toolkit command line utilities

How to use the XSLT Processor, C++ Edition command line utility

To perform a transformation, you can call the XSLT Processor, C++ Edition from the command line. Xalan is a simple executable providing a command-line interface for performing XSLT transformations .The following describes how you can use Xalan to perform transformations:

1. Set *XALANCROOT* to be `/usr/lpp/ixm/IBM/xslt4c-1_7`
2. Set *XERCESCROOT* to be `/usr/lpp/ixm/IBM/xml4c-5_4`
3. Set the *PATH* to include `$XALANCROOT/bin`
4. Set the *LIBPATH* to include `$XALANCROOT/lib:$XERCESCROOT/lib`

Then from the command line, type the following:

```
Xalan
```

or

```
Xalan -?
```

to show all the options. The following is an example of the Xalan command line:

```
Xalan -o foo.out
         $XALANCROOT/samples/SimpleTransform/foo.xml
         $XALANCROOT/samples/SimpleTransform/foo.xsl
```

Rule:: All three segments of the above example must be entered on the same command line.

Here is a sample job for the Xalan command (IXMXAL17):

```
//XALAN1   JOB  REGION=0M,NOTIFY=&SYSUID
//STEP1    EXEC PGM=IXMXAL15
//  PARM='/-e ibm-1047-s390 -o DD:OUTFILE DD:INXML DD:INXSL'
//STEPLIB  DD   DSN=hlq.SIXMLOD1,DISP=SHR,
//INXML    DD   DSN=USER1.FOO.XML,DISP=SHR
//INXSL    DD   DSN=USER1.FOO.XSL,DISP=SHR
//OUTFILE  DD   DSN=USER1.FOO.OUT,DISP=SHR
//*
```

The following table lists the flags and arguments the Xalan executable can take (the flags are case insensitive) :

Table 14. Flags and Arguments for the Xalan Executable

-a (Use stylesheet processing instruction, not the stylesheet argument)
-e encoding (Force the specified encoding for the output)
-i integer (Indent the specified amount)
-m (Omit the META tag in HTML output)
-o filename (Write transformation result to this file (rather than to the console))
-p name expr (Set a stylesheet parameter with this expression)
-u name expr (Disable escaping of URLs in HTML output)
-v (Validate the XML source document)

Table 14. Flags and Arguments for the Xalan Executable (continued)

- (A dash as the 'source' argument reads from stdin. A dash as the 'stylesheet' argument reads from stdin. ('-' cannot be used for both arguments.))
-?(Show all options)

There is another XSLT Processor, C++ Edition command line utility available called testXSLT. Like Xalan, this command line utility can perform transformations. However, unlike Xalan, it has additional options which can be used to help debug stylesheets during development. The following describes how you can use testXSLT to perform transformations:

1. Set *XALANCROOT* to be `/usr/lpp/ixm/IBM/xslt4c-1_7`
2. Set *XERCESCROOT* to be `/usr/lpp/ixm/IBM/xml4c-5_4`
3. Set the *PATH* to include `$XALANCROOT/bin`
4. Set the *LIBPATH* to include `$XALANCROOT/lib:$XERCESCROOT/lib`

You can now call the testXSLT executable with the appropriate flags and arguments or enter

```
testXSLT -h
```

to show all the options. The following command line, for example, includes the **-IN**, **-XSL**, and**-OUT** flags with their accompanying arguments; the XML source document, the XSL stylesheet, and the output file:

```
testXSLT -IN $XALANCROOT/samples/SimpleTransform/foo.xml
         -XSL $XALANCROOT/samples/SimpleTransform/foo.xsl
         -OUT foo.out
```

Rule:: All three segments of the above example must be entered on the same command line.

Also, here is a sample job for the testXSLT command (IXMTST17):

```
//TSTXSLT1 JOB  REGION=0M,NOTIFY=&SYSUID
//STEP1    EXEC PGM=IXMTST15
//  PARM='/-IN FILE:////FOO.XML -XSL FILE:////FOO.XSL -OUT FOO.OUT'
//STEPLIB  DD   DSN=hlq.SIXMLOD1,DISP=SHR,
//*
```

The following table lists the flags and arguments the testXSLT executable can take (the flags are case insensitive) :

Table 15. Flags and Arguments for the testXSLT Executable

-IN inputXMLURL
-XSL XSLTransformationURL
-OUT outputFileName
-H(Display list of command line options)
-? (Display list of command line options)
-V (Version info)
-QC (Quiet Pattern Conflicts Warnings)
-Q (Quiet Mode)
-INDENT (Number of spaces to indent each level in output tree — default is 0)
-VALIDATE (Validate the XSL and XML input — default is not to validate)

Table 15. Flags and Arguments for the testXSLT Executable (continued)

-TT (Trace the templates as they are being called)
-TG (Trace each result tree generation event)
-TS (Trace each selection event)
-TTC (Trace the template children as they are being processed)
-XML (Use XML formatter and add XML header)
-NH (Don't write XML header) *The -XML flag must be set before use
-HTML (Use HTML formatter)
-NOINDENT (turns off HTML indenting) *The -HTML flag must be set before use
-STRIPCDATA (Strip CDATA sections of their brackets, but do not escape) *The -XML or -HTML flag must be set before use
-ESCAPECDATA (Strip CDATA sections of their brackets, and escape) *The -XML or -HTML flag must be set before use
-TEXT (Use simple Text formatter)
-DOM (Test for well-formed output — format to DOM then to XML for output)
-XST (Format to Xalan source tree, then to XML for output)
-XD (Use Xerces DOM instead of Xalan source tree)
-DE (Disable built-in extension functions)
-EN (Specify the namespace URI for Xalan extension functions; the default is `http://xml.apache.org/xslt`)
-PARAM name expression (Set a stylesheet parameter)

Chapter 5. Where to go for more information

For more information on XML Toolkit for z/OS, visit the XML Toolkit Web site at:
`http://www.ibm.com/servers/eserver/zseries/software/xml/`

For additional information on the Apache XML project, visit the Apache Web site at:
`http://xml.apache.org/`

There are also two redbooks that you may find informative:

- *Using XML on z/OS and OS/390 for Application Integration*, which contains information on how to integrate XML technology with business applications on z/OS. This document can be accessed from the following link: http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246285.html?Open
- *XML on z/OS and OS/390: Introduction to a Service-Oriented Architecture*, which provides a general introduction to the XML Toolkit in the first half, followed by a comprehensive introduction to services–oritented architecture (SOA) and Web Services. This document can be accessed from the following link: http://www.redbooks.ibm.com/redbooks/pdfs/sg246826.pdf

Notices

This information was developed for products and services offered in the U.S.A.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, New York 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2–31 Roppongi 3–chrome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms used in this book are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- Language Environment
- MVS
- OS/390
- zSeries
- z/OS

UNIX is a registered trademark of The Open Group in the United States and other countries.

The following terms may be trademarks or service marks of others:

Java	Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
UNIX	UNIX is a registered trademark of The Open Group in the United States and other countries.
Xerces	The Apache Software Foundation
Xalan	The Apache Software Foundation

Index

W

X

Z

Readers' Comments — We'd Like to Hear from You

XML Toolkit for z/OS
User's Guide

Publication No. SA22-7932-03

We appreciate your comments about this publication. Feel free to comment on specific errors or omissions, accuracy, organisation, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

For general questions, please call ″Hello IBM″ (phone number 01803/313233).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Comments:

Thank you for your support.

To submit your comments:

- Send your comments to the address on the reverse side of this form.
- Send your comments via email to: mhvrcfs@us.ibm.com

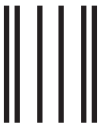
If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SA22-7932-03

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5655–J51

Printed in USA

SA22-7932-03