

XML Toolkit for z/OS



User's Guide

XML Toolkit for z/OS



User's Guide

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 99.

Seventh Edition, 2008

This edition applies to Version 1 Release 9 of XML Toolkit for z/OS (5655-J51) and to all subsequent releases and modifications until otherwise indicated in new editions.

This is a major revision of SA22-7932-05.

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.

© Copyright International Business Machines Corporation 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
Tables	vii
About this document	ix
Who should use this User's Guide?	ix
What is in the User's Guide?	ix
Summary of changes	xi
Chapter 1. Introduction	1
Why XML?	1
APIs	1
DOM	1
SAX	3
DOM vs SAX	5
XPath	6
Validation	6
XML Toolkit for z/OS	7
z/OS specific parser classes	8
Deprecated DOM support	11
Toolkit packaging strategy	11
Toolkit support for both z/OS UNIX System Services and MVS environments	12
Chapter 2. How to access XML data	13
How to access data sets	13
Relative URIs	13
Absolute URIs	14
Considerations when using the Xalan C++ commands	14
DTDs, Schema and other embedded files	14
Chapter 3. Encoding issues	15
Encoding and XML	15
XML and z/OS	16
Avoiding conversion	17
Chapter 4. How to use Toolkit XPLINK support	19
Using Toolkit XPLINK support	19
Building an XPLINK application	19
Running an XPLINK application	20
Chapter 5. How to use the XML Parser, C++ Edition	21
Using the sample applications	22
Rule for running non-XPLINK samples	24
Rule for running XPLINK samples	24
z/OS UNIX Environment	24
Building sample applications for the z/OS UNIX Environment	24
Using your sample applications on the z/OS UNIX Environment	27
MVS Environment	28
Building sample applications for the MVS Environment	28
Using your sample applications on the MVS Environment	30
Multi-threading considerations	31
Using UNIX pthreads	32

	Using MVS multi-tasking	32
I	Chapter 6. How to use z/OS specific parser classes	33
I	Using a SAX2XMLReader class	33
I	Constructing a zXercesDOMParser	38
I	Constructing a DOMBuilder	44
I	Using samples for the z/OS specific parser classes	49
	Chapter 7. How to use the XSLT Processor, C++ Edition	51
	Using the sample applications	52
	Rule for running non-XPLINK samples	53
	Rule for running XPLINK samples	53
	z/OS UNIX Environment	53
	Building sample applications for the z/OS UNIX Environment	53
	Using your sample applications on the z/OS UNIX Environment	56
	MVS Environment.	57
	Building sample applications for the MVS Environment	57
	Using your sample applications on the MVS Environment	60
	Chapter 8. How to use the XML Toolkit command line utilities	63
	How to use the XSLT Processor, C++ Edition command line utility	63
	Chapter 9. Where to go for more information.	67
	Appendix A. Building samples for native MVS using JCL	69
	Building XML Parser, C++ Edition samples for native MVS using JCL	69
	Building XSLT Processor, C++ Edition samples for native MVS using JCL	71
	Appendix B. Calling XML Parser, C++ Edition from COBOL	75
	Source code samples	75
	Compilation instructions	83
	Setup instructions	84
	Appendix C. Parser environment and instance reuse	85
	Appendix D. Accessibility	97
	Using assistive technologies	97
	Keyboard navigation of the user interface	97
	z/OS information	97
	Notices	99
	Trademarks.	100
	Index	101

Figures

1.	DOM Parsing Model	3
2.	SAX Parsing Model	5
I 3.	Open source parsing model	9
I 4.	z/OS parsing model	10
5.	Non-valid XML file to be processed via DDNAME	82
6.	JCL to compile, bind and run the sample code	83

Tables

1.	DOM vs SAX.	5
2.	Expected Validation Results	6
3.	Interfaces and Specifications for the Toolkit Parser	7
4.	Interfaces and Specifications for the Toolkit Processor	8
5.	z/OS UNIX vs. MVS.	12
6.	Product Files Required to Build Sample XML Applications for z/OS UNIX Environments.	25
7.	Library Files Required to Run Sample XML Applications on z/OS UNIX.	27
8.	Product Files Required to Build Sample XML Applications for MVS Environments	28
9.	Library Files Required to Run Sample XML Applications on MVS	30
10.	SAX2XMLReader APIs and behaviors in z/OS	34
11.	SAX2 features and behaviors in z/OS	36
12.	SAX2 properties and behaviors in z/OS	38
13.	zAbstractDOMParser APIs and behaviors in z/OS.	39
14.	zXercesDOMParser APIs and behaviors in z/OS	43
15.	DOMBuilder APIs and behaviors in z/OS	46
16.	DOMImplementation APIs and behaviors in z/OS	47
17.	DOMBuilder features and behaviors in z/OS.	48
18.	Xerces features and behaviors in z/OS.	48
19.	Xerces properties and behaviors in z/OS	49
20.	Product Files Required to Build Sample XML Applications for z/OS UNIX Environments.	54
21.	Library Files Required to Run Sample XML Applications on z/OS UNIX.	56
22.	Product Files Required to Build Sample XML Applications for MVS Environments	58
23.	Library Files Required to Run Sample XML Applications on MVS	60
24.	Flags and Arguments for the Xalan Executable.	63
25.	Flags and Arguments for the testXSLT Executable	64

About this document

This document provides information you need to use V1.9.0 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 XSLT Processor, Java Edition are not supported in Toolkit V1.9.0.

It also provides information on the z/OS specific classes in the XML Parser, C++ Edition which utilize the z/OS XML System Services component to parse XML documents. When the z/OS specific classes are used, a portion of the parse is performed by z/OS XML System Services, which can take advantage of the more competitive zAAP specialty engines (if present).

Information on how to use Toolkit V1.8.0 and Toolkit V1.7.0 is available from the *XML Toolkit for z/OS User's Guide V1R8* and the *XML Toolkit for z/OS User's Guide V1R7*. 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, and how to utilize the z/OS XML System Services parser. Using the document, you will:

- Receive an introduction to XML and the XML Toolkit.
 - Read about XML and its implications in today's world.
 - Learn about the components of the XML Toolkit.
 - Learn about the APIs 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 z/OS specific parser classes that utilize the z/OS XML System Services component.
 - Learn about the z/OS specific parser classes.

- | - Review the APIs associated with the z/OS specific parser classes.
- | - Review the provided samples that use the z/OS specific parser classes.
- | • 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 SA22-7932-06 XML Toolkit Version 1 Release 9

This document is a refresh of *XML Toolkit for z/OS User's Guide*, SA22-7932-05, which supports XML Toolkit Version 1 Release 9.

New Information

- An introduction to new support that allows XML Toolkit users performing non-validating parsing to indicate that z/OS z/OS XML System Services be used as an underlying parsing technology: "z/OS specific parser classes" on page 8
- Chapter 6, "How to use z/OS specific parser classes," on page 33

Changed Information

Deleted Information

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 SA22-7932-05 XML Toolkit Version 1 Release 9

This document contains information previously presented in *XML Toolkit for z/OS User's Guide*, SA22-7932-04, which supports XML Toolkit Version 1 Release 8.

New Information

- Three new appendixes added: Appendix A, "Building samples for native MVS using JCL," on page 69, Appendix B, "Calling XML Parser, C++ Edition from COBOL," on page 75, and Appendix C, "Parser environment and instance reuse," on page 85.
- New information added to each "How to use..." chapter.

Changed Information

Deleted Information

Summary of Changes for SA22-7932-04 XML Toolkit Version 1 Release 8

This document contains information previously presented in *XML Toolkit for z/OS User's Guide*, SA22-7932-03, which supports XML Toolkit Version 1 Release 7.

New Information

- XPLINK support: Chapter 4, "How to use Toolkit XPLINK support," on page 19
- Deprecated DOM support: "Deprecated DOM support" on page 11
- New information added to each "How to use..." chapter.

Changed Information

Deleted Information

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.

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++ 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.

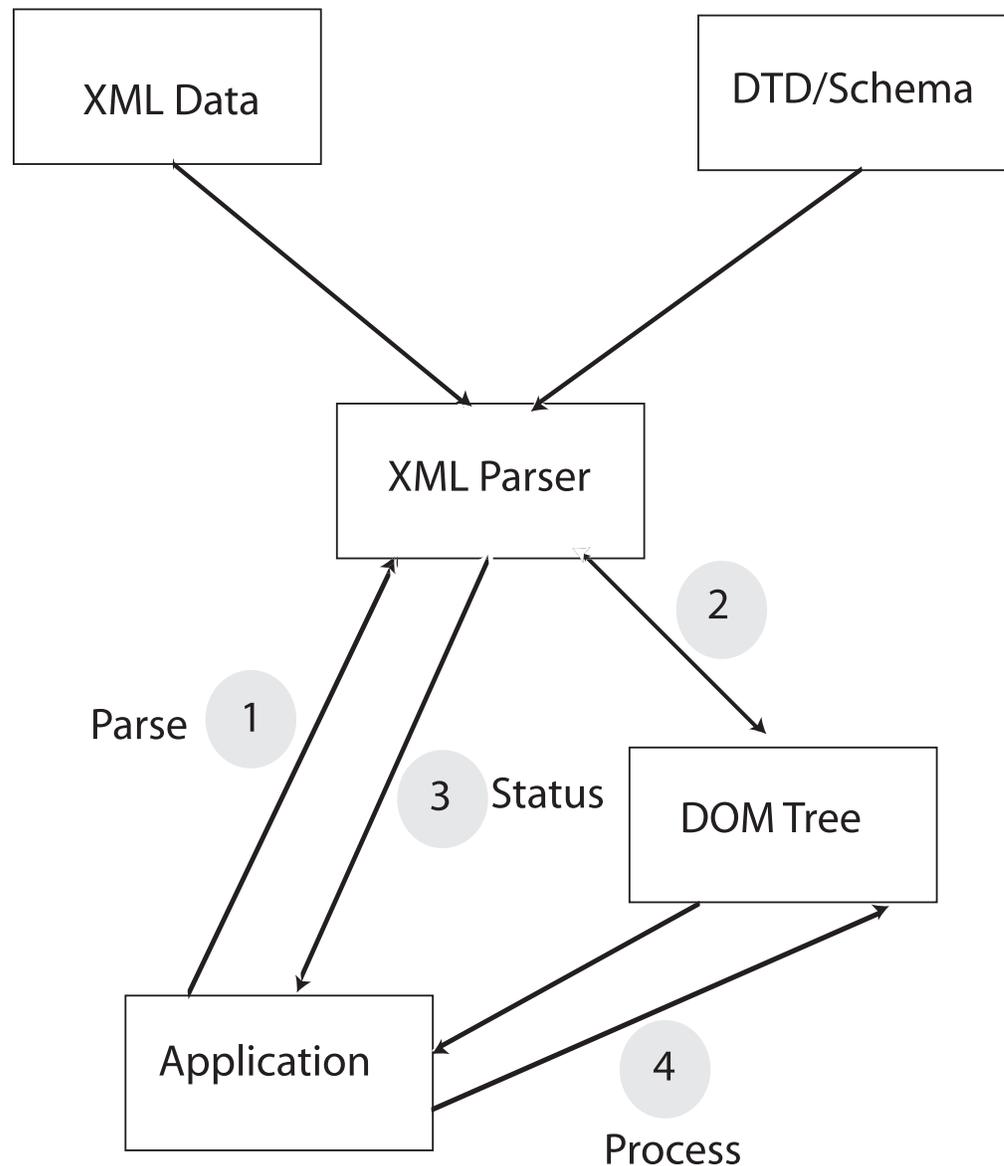


Figure 1. DOM Parsing Model

For information on the Toolkit support for DOM APIs, see the Interfaces and Specifications chart for Toolkit Parser on page 7.

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++ 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.

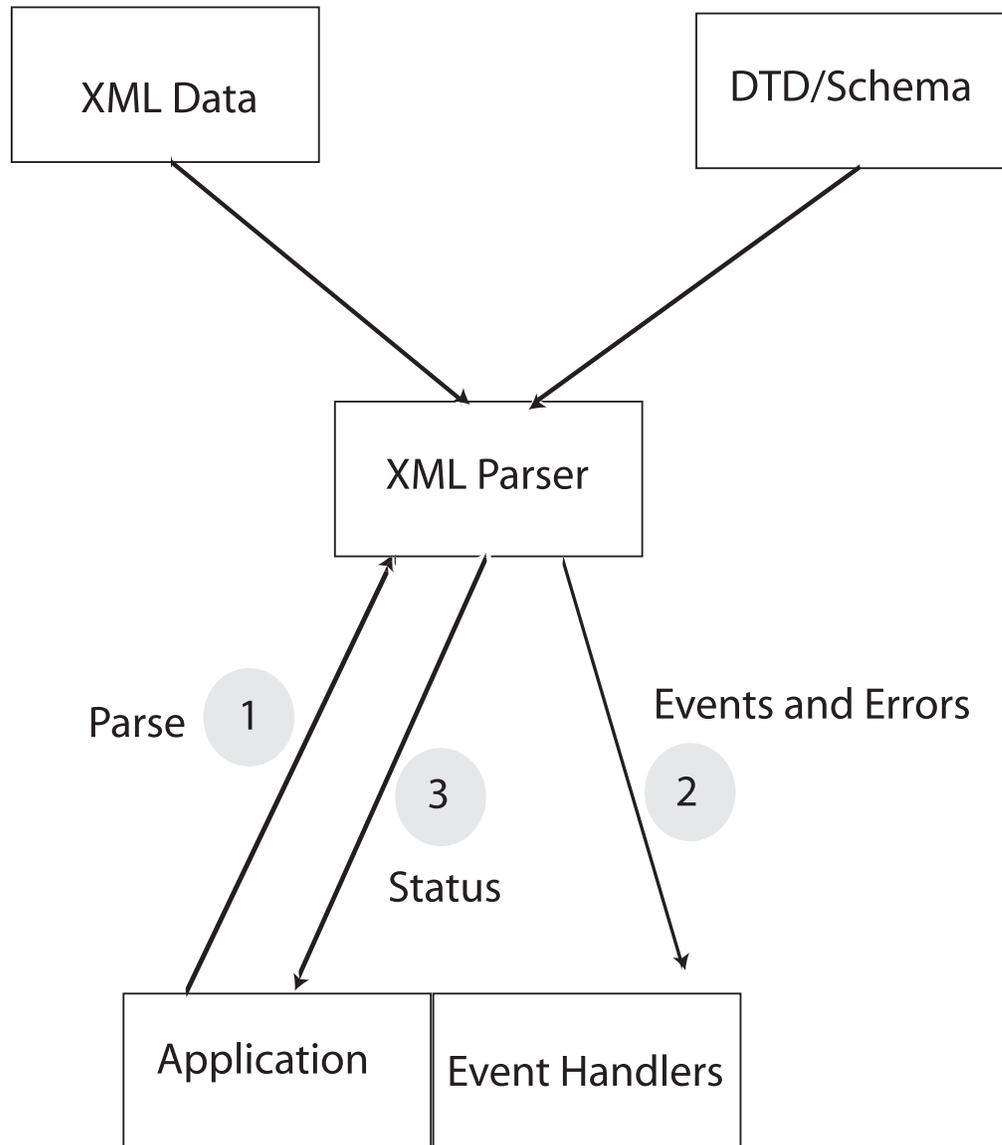


Figure 2. SAX Parsing Model

For information on the Toolkit support for SAX APIs, see the Interfaces and Specifications chart for Toolkit Parser on page 7.

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 8.

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.

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. The XML Parser, Java Edition is no longer supported in the Toolkit as equivalent support exists in the Java SDK 1.4 and later versions. 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.

The XML Parser, C++ Edition has a set of classes closely resembling the standard SAX2 and DOM classes with slight alterations. These changes allow the XML Parser, C++ Edition to utilize the z/OS XML System Services component to parse XML documents. When these z/OS specific classes are used, a portion of the parse is performed by z/OS XML System Services, gaining a raw performance improvement, along with the ability to take advantage of the more competitive zAAP specialty engines (if present).

In addition to the parser, Toolkit V1.9.0 also includes the XSLT Processor, C++ Edition. The XSLT Processor, Java Edition is no longer supported in the Toolkit as equivalent support exists in the Java SDK 1.4 and later versions. 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

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

Interfaces and Specifications	C++ Edition parser
XML 1.0	S
XML 1.1	S
XML Namespaces 1.0	S
XML Namespaces 1.1	S
XML Schema 1.0	S

Table 4. Interfaces and Specifications for the Toolkit Processor

Interfaces and Specifications	C++ Edition processor
XSL Transformations	S
XPATH 1.0	S
XML 1.1	S
XML Namespaces 1.1	S

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.

z/OS specific parser classes

The capability exists within the XML Parser, C++ Edition that allows an application to take advantage of the z/OS XML System Services component. A set of z/OS specific parser classes have been implemented in the XML Parser, C++ Edition to provide this ability. These classes were created to closely mimic the existing SAX2 and DOM interfaces. They allow many applications to exploit the improved cost and performance characteristics of the z/OS XML System Services component with minimal changes to their code.

The figure below illustrates the flow when using the existing embedded open source parser. Applications may continue to use this existing parser functionality without modifications to their code.

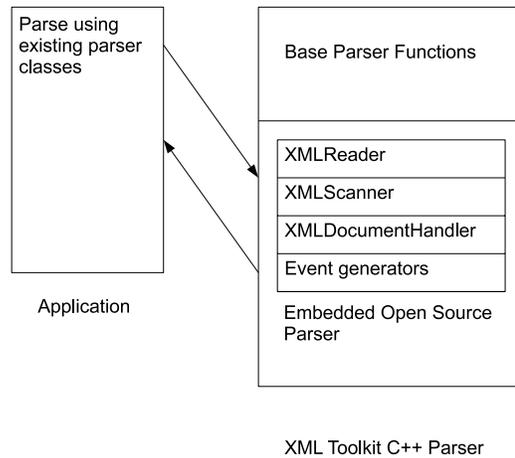


Figure 3. Open source parsing model

When an application does what is referred to as a “Parse using existing parser classes” in the application box, it will use the existing embedded open source parser to parse the XML document. This consists of an XMLReader, an XML Scanner (which is configurable), the XMLDocumentHandler interface which feeds the results back to the application via an event generator that will either generate SAX callback events or will create a DOM tree in memory.

The figure below illustrates the flow when using the new capability provided by the z/OS specific parser classes. In order to take advantage of this functionality, minor code modifications are required.

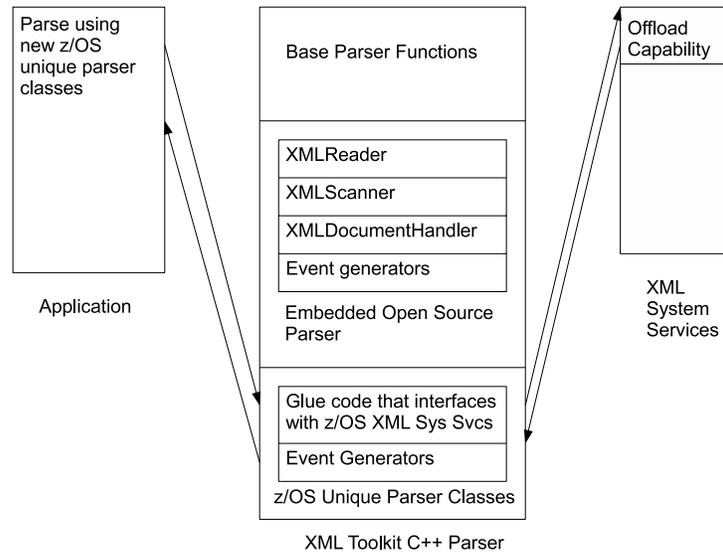


Figure 4. z/OS parsing model

When an application does what is referred to a “Parse using new z/OS specific parser classes”, it will invoke the z/OS XML System Services component to parse the XML document. The document is passed to z/OS XML System Services in one or more input buffers and will be returned in one or more output buffers. As output buffers are consumed by the new classes, the event generator will generate either SAX2 callbacks or create a DOM tree in memory.

Note: This implementation does not have a configurable scanner nor does it implement the XMLDocumentHandler interface.

There are some benefits to using the classes that take advantage of z/OS XML System Services:

- z/OS XML System Services can take advantage of redirection of work to zAAP specialty engines, if present.
- Significant performance improvements are possible.

There are some significant differences in behavior in the z/OS specific parser classes that must be considered when deciding if you can use these. Not all of the functionality of the existing base parser classes exists in the new z/OS specific parser classes. The functionality that is most widely used is implemented where possible given that we must conform to the behavior of the z/OS XML System Services component:

- z/OS release 1.7 or higher is required.
- Validation via either an internal or external DTD or a schema is not supported.
- The z/OS XML System Services component is a namespace compliant parser only. You may not specify an option to make the parser non-namespace compliant.
- There is no implementation of the XML Entity Resolver. z/OS XML System Services will resolve entity references that are defined in the internal DTD and

substitute the replacement text for the entity. However, there is no notification given to the application that this has happened.

- There is no ability to force namespaces to be URI conformant.
- Different error messages will be displayed when an XML document contains an error.
 - The new error messages provide an offset to the character in error, not a line and column number.
 - In most cases, the return and reason codes from z/OS XML System Services are displayed as opposed to a specific description of the error.
- SAX1 and deprecated DOM are not supported.
- The SAX2XMLFilter class is not supported.
- The XMLDocumentHandler interface is not implemented.
- The z/OS XML System Services C/C++ APIs, which are called by the z/OS specific parser classes, are compiled XPLINK for optimum performance. You should build your application using XPLINK and bind it with the XPLINK version of the XML Parser, C++ Edition DLLs, otherwise you must set the following environment variable: :

```
export _CEE_RUNOPTS="XPLINK(ON)"
```

If you mix non-XPLINK and XPLINK compiled code, there will be a significant performance degradation during execution. It is strongly recommended that when using the new z/OS specific parser classes, that only XPLINK compiled code is used. This will provide the best possible performance. If, however, your application cannot be compiled XPLINK, you should use the non-XPLINK version of the parser to avoid this performance penalty.

For more information on the z/OS specific parser classes, including the APIs, examples, and sample programs, see Chapter 6, “How to use z/OS specific parser classes,” on page 33. More information on the z/OS XML System Services parser, including return and reason code descriptions, can be found in the *z/OS XML System Services User's Guide and Reference* located at <http://www.ibm.com/servers/eserver/zseries/zos/xml/>.

Deprecated DOM support

The previously deprecated DOM code has been removed from the main XML Toolkit Parser DLL to reduce the DLL's footprint. If you want to use the deprecated DOM code, you will need to include the new sidedeck for the deprecated DOM code. The DLL and sidedeck names are listed in Table 6 on page 25 and Table 8 on page 28.

Toolkit packaging strategy

The Toolkit V1.9.0 package contains three levels of the Toolkit: V1.9.0, V1.8.0, and V1.7.0. 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 V1.9.0 is the current level, the Toolkit package will have included in it levels V1.9.0, V1.8.0, and V1.7.0. A customer who may be installing a product that requires Toolkit V1.8.0 can still obtain and install V1.8.0 (assuming they don't already have it) because it is included in the current V1.9.0 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 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. The following is an 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 interfaces that need to access data support both absolute URIs and relative URIs. The following are some known exceptions:

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

For more information on the Xalan command, see Chapter 8, "How to use the XML Toolkit command line utilities," on page 63. For more information on Xalan itself, see "XML Toolkit for z/OS" on page 7.

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)''"
```

Chapter 3. 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 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, because these documents are non-compliant, they 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 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.

Chapter 4. How to use Toolkit XPLINK support

Extra Performance Linkage (XPLINK) is a relatively new call linkage between programs that have the potential for a significant performance increase when used in an environment of frequent calls between small functions or subprograms. The Object Oriented aspect of C++ causes much C++ code to fall into this category.

An XPLINK copy of the XML Parser, C++ Edition and the XSLT Processor, C++ Edition library files and sidedecks are provided for customers in addition to the non-XPLINK versions. A listing of these files along with the build and run steps required to use XPLINK are presented in the following chapters: Chapter 5, "How to use the XML Parser, C++ Edition," on page 21 and Chapter 7, "How to use the XSLT Processor, C++ Edition," on page 51.

Using Toolkit XPLINK support

Under certain circumstances, it may be appropriate to use the new XPLINK Toolkit code. If you have an existing application that is pure XPLINK and you need to do XML parsing, you should see a performance improvement if you were previously using your XPLINK application with the non-XPLINK XML Parser, C++ Edition. This is because when you use your XPLINK application with the non-XPLINK XML Parser, C++ Edition, you incur a significant performance penalty each time you call the XML Parser, C++ Edition. This performance penalty is a result of having to run through additional code to convert the XPLINK stack structure and register conventions to the format that the non-XPLINK XML Parser, C++ Edition expected. When the XML Parser, C++ Edition finishes converting, there is additional overhead (another performance penalty) restoring the XPLINK environment upon return. As a result, you should see a significant benefit by calling an XPLINK XML Parser, C++ Edition from your XPLINK application, since you'll be avoiding the performance penalties of having to do the XPLINK to non-XPLINK, and then back XPLINK code conversions.

If your application is non-XPLINK, then you should continue to use the non-XPLINK XML Parser, C++ Edition. Calling the XPLINK XML Parser, C++ Edition from a non-XPLINK application will most likely perform worse than if you continue to use the non-XPLINK XML Parser, C++ Edition.

If you can convert your non-XPLINK application to be 100 percent XPLINK, then you should see a significant benefit using the XPLINK XML Toolkit code.

For best results, you want all of the code you are calling to be built using XPLINK. For example, you would not want to bind an XPLINK application with the XPLINK XML Parser, C++ Edition main DLL (libxml4c5_6_0.xplink.dll) and the non-XPLINK deprecated DOM DLL (libxml4c-depdom5_6_0.dll).

For more information on XPLINK, refer to the *z/OS Language Environment Programming Guide* Chapter 3, "Using Extra Performance Linkage (XPLINK).

Building an XPLINK application

In order to build an XPLINK application, you need to specify the XPLINK compiler option (-wc,XPLINK) during compilation. When link-editing your application, you must use the DFSMS binder and specify the XPLINK binder option (-w1,XPLINK). You also need to include the XPLINK sidedeck on the bind step.

The samples provided in the Toolkit are pre-built non-XPLINK. If you want to build an XPLINK version of them, you can set an environment variable that has been added in support of XPLINK. The environment variable, once set, will apply the correct compiler and binder options in the Makefile and the correct XPLINK sidedeck will also be included. To build an XPLINK copy of the samples, do the following:

```
export OS390_XPLINK=1
```

For the XSLT Processor, C++ Edition samples there is an extra step needed to pickup the correct version of the Standard C++ library (see Chapter 7, “How to use the XSLT Processor, C++ Edition,” on page 51 for more information).

Running an XPLINK application

If the initial program you call is compiled XPLINK, then Language Environment will initialize the enclave as an XPLINK environment. If your initial program is non-XPLINK, and you are calling an XPLINK program later on, then you need to specify the XPLINK(ON) runtime option so that calls may be made between XPLINK and non-XPLINK programs.

When you build the samples and use the OS390_XPLINK environment variable, XPLINK(ON) runtime options never needs to be set.

Chapter 5. 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

PSVIWriter

exposes the underlying PSVI of the parsed XML file

SCMPrint

parses an XSD file and prints information about the Schema Component Model

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 22 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.

Rule: The prebuilt samples shipped with the Toolkit are the non-XPLINK versions. If you want to use the XPLINK versions of the samples, then you must build your own copy of them.

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 “z/OS UNIX Environment” on page 24 and “Building sample applications for the MVS Environment” on page 28. The procedures for using your built samples are outlined in sections “Using your sample applications on the z/OS UNIX Environment” on page 27 and “Using your sample applications on the MVS Environment” on page 30

The XML Parser, C++ Edition component is installed in `/usr/lpp/ixm/IBM/xml4c-5_6` 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 XERCECROOT=/usr/lpp/ixm/IBM/xml4c-5_6
```

Then type in the following command statements:

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

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

```
cd $XERCECROOT/samples/data
DOMPrint -v=always -wenc=IBM-1047-s390 -wfpp=on personal.xml
(use "-v=auto" for personal-schema.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:

```
<?xml version="1.0" encoding="IBM-1047-s390" standalone="no" ?>
<!DOCTYPE personnel SYSTEM "personal.dtd">
<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"/>
  </person>

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

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

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

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

  <person id="five.worker">
    <name>
      <family>Worker</family>
```

```
    <given>Five</given>
  </name>
  <email>five@foo.com</email>
  <link manager="Big.Boss"/>
</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 for running non-XPLINK samples

In order to run the non-XPLINK 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 this 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.

Rule for running XPLINK samples

In order to run the XPLINK samples, the XML Parser, C++ Edition requires the run-time libraries provided by Language Environment, SCEERUN and SCEERUN2, to be made available in the program search order. The best way to do this is by adding the SCEERUN and SCEERUN2 data sets into the LNKLST. If you do not wish to add SCEERUN and SCEERUN2 to the LNKLST, access SCEERUN and SCEERUN2 data sets 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.
non-XPLINK product files	
libxml4c5_6_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.
libxml4c-depdom5_6_0.x	The definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs.
XPLINK product files	
libxml4c5_6_0.xplink.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 use XPLINK to bind application code.
libxml4c-depdom5_6_0.xplink.x	The definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs and compiling with XPLINK.

Rules for invoking the XML Parser, C++ Edition in z/OS UNIX

Any application that is to invoke the XML Parser, C++ Edition parser under the z/OS UNIX System Services environment must include libxml4c5_6_0.x, (or libxml4c5_6_0.xplink.x if using XPLINK), when they bind. The binder uses the definition side-deck to resolve references to functions and variables defined in libxml4c5_6_0.dll (libxml4c5_6_0.xplink.dll if using XPLINK).

If you are using the deprecated DOM APIs, you need to include the libxml4c-depdom5_6_0.x sidedeck, or for XPLINK applications, the libxml4c-depdom5_6_0.xplink.x sidedeck.

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

```
export XERCECROOT=/usr/lpp/ixm/IBM/xml4c-5_6
```

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 XERCECOUT=$HOME/mysamples
```

3. Copy the samples directory to your directory:

```
cp -r /usr/lpp/ixm/IBM/xml4c-5_6/samples $XERCECOUT
```

Since the XERCECOUT 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.

The next statement is only required for building XPLINK samples:

```
export OS390_XPLINK=1
```

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 XERCECOUT environment variable, type the following:

```
cd $XERCECOUT/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 `$XERCECOUT/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 `$XERCECROOT/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
non-XPLINK library files	
<code>libxml4c5_6_0.dll</code>	XML Parser, C++ Edition library file
<code>libxml4c5_6_0-depdom.dll</code>	library file for the previously deprecated DOM API
<code>libcudata33.0.dll</code> , <code>libcudata_stub33.0.dll</code> , <code>libcuc33.0.dll</code> , <code>libcui18n33.0.dll</code>	ICU library files
XPLINK library files	
<code>libxml4c5_6_0.xplink.dll</code>	XML Parser, C++ Edition library file
<code>libxml4c-depdom5_6_0.xplink.dll</code>	library file for the previously deprecated DOM API
<code>libcudata33.0.xplink.dll</code> , <code>libcudata_stub33.0.xplink.dll</code> , <code>libcuc33.0.xplink.dll</code> , <code>libcui18n33.0.xplink.dll</code>	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 XERCECROOT=/usr/lpp/ixm/IBM/xml4c-5_6
```

Then type in the following command statements:

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

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

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

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

```
SAXCount $XERCECROOT/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/bpxalty1.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.	
non-XPLINK product files		
IXM4C56X	Definition side-deck that describes the XML Parser, C++ Edition functions and the variables.	hlq.SIXMEXP
IXMDD56X	Definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs.	hlq.SIXMEXP
XPLINK product files		
IXM4C6XX	Definition side-deck that describes the XML Parser, C++ Edition functions and the variables. This is required in order to use XPLINK to bind application code.	hlq.SIXMEXP
IXMDD6XX	Definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs and XPLINK.	hlq.SIXMEXP

Rules for invoking the XML Parser, C++ Edition in native MVS

Any application that is to invoke the XML Parser, C++ Edition parser under the native MVS environment must include either of the following definition side-decks when they bind: IXM4C56X (for non-XPLINK applications) or IXM4C6XX (for XPLINK applications). The binder uses the definition side-deck to resolve references to functions and variables defined in the IXM4C56 or IXM4C6X. In addition to the above, any applications that wish to use previously deprecated DOM APIs must also include either of the following definition side-decks: IXMDD56X (for non-XPLINK applications) or IXMDD6XX (for XPLINK applications). The binder uses the definition side-deck to resolve references to functions and variables defined in the IXMDD56 or IXMDD6X.

Rules for building samples in native MVS

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, 500 tracks on 3390, Record format:U,  
|          Record Length: 0, Block size: 32760, ORG: PDSE,  
|          Directory blocks: 0
```

You should allocate a minimum of 500 tracks.

If you are building samples from multiple releases, you will need a unique PDSE for each release, for example: **SAMPLES.V180** .LOAD for samples from Toolkit V1.8.0 and **SAMPLES.V190** .LOAD for samples from Toolkit V1.9.0.

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

```
export XERCESSROOT=/usr/lpp/ixm/IBM/xml4c-5_6
```

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 XERCESSOUT=$HOME/mysamples
```

3. Copy the samples directory to your directory:

```
cp -r /usr/lpp/ixm/IBM/xml4c-5_6/samples $XERCESSOUT
```

Since the XERCESSOUT 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 XERCECOUT 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 LOAEXP=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.

The next statement is only required if your building XPLINK samples:

```
export OS390_XPLINK=1
```

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

```
cd $XERCECOUT/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
non-XPLINK library files		
IXM4C56	XML Parser, C++ Edition library file	hlq.SIXMLOD1
IXMDD56	previously deprecated DOM API library file	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
non-XPLINK library files		
IXMI33UC	ICU library file (explicitly loaded by IXM4C56)	hlq.SIXMLOD1
IXMI33DA	ICU library file	hlq.SIXMLOD1
IXMI33D1	ICU library file	hlq.SIXMLOD1
IXMI33IN	ICU library file	hlq.SIXMLOD1
XPLINK library files		
IXM4C6X	XML Parser, C++ Edition library file	hlq.SIXMLOD1
IXMDD6X	previously deprecated DOM API library file	hlq.SIXMLOD1
IXMI33XC	ICU library file (explicitly loaded by IXM4C6X)	hlq.SIXMLOD1
IXMI33XA	ICU library file	hlq.SIXMLOD1
IXMI33X1	ICU library file	hlq.SIXMLOD1
IXMI33XN	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 SAXCount.

```
//USERJOB JOB MSGLEVEL=(1,1),CLASS=A
//TEST EXEC PGM=SAXCOUNT,
/* HFS file input
// PARM='//usr/lpp/ixm/IBM/xml4c-5_6/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 6. How to use z/OS specific parser classes

This topic describes how to use the z/OS specific parser classes that will utilize z/OS XML System Services to parse an XML document, instead of the embedded open source parser. Examples are provided to illustrate how to convert existing code to use this function. Samples are provided to enable use of this function. The APIs, features, and properties supported are listed along with any z/OS unique behaviors for SAX2XMLReader, zXercesDOMParser, and DOMBuilder. For a complete description of the APIs, features and properties refer to the HTML documentation provided with the product.

Using a SAX2XMLReader class

In order to use the new, z/OS specific parser classes to parse XML files using SAX2, you will need to create an instance of the SAX2XMLReader class. This is the same base class you would use to parse an XML document using the embedded open source parser. However, instead of using the SAX2 XMLReaderFactory::createXMLReader() method, you will use a new z/OS specific zXMLReaderFactory::createXMLReader() method. This will give you access to the z/OS specific parser classes that will use the z/OS XML System Services to parse the XML document instead of the embedded open source parser.

The example below shows the code you need in order to create an instance of SAX2XMLReader that will use the new z/OS specific parser classes. Changes from the existing SAX2 implementation are in bold.

```
#include <xercesc/sax2/SAX2XMLReader.hpp>
#include <xercesc/zparsers/zXMLReaderFactory.hpp>
#include <xercesc/sax2/DefaultHandler.hpp>
#include <xercesc/util/XMLString.hpp>

#ifdef XERCES_NEW_IOSTREAMS
#include <iostream>
#else
#include <iostream.h>
#endif

XERCES_CPP_NAMESPACE_USE

int main (int argc, char* args[]) {

    try {
        XMLPlatformUtils::Initialize();
    }
    catch (const XMLException& toCatch) {
        char* message = XMLString::transcode(toCatch.getMessage());
        cout << "Error during initialization! :\n";
        cout << "Exception message is: \n"
             << message << "\n";
        XMLString::release(&message);
        return 1;
    }

    char* xmlFile = "x1.xml";
    SAX2XMLReader* parser = zXMLReaderFactory::createXMLReader();
    parser->setFeature(XMLUni::fgSAX2CoreValidation, false);
    parser->setFeature(XMLUni::fgSAX2CoreNamespaces, true);
    DefaultHandler* defaultHandler = new DefaultHandler();
    parser->setContentHandler(defaultHandler);
    parser->setErrorHandler(defaultHandler);
}
```

```

try {
    parser->parse(xmlFile);
}
catch (const XMLException& toCatch) {
    char* message = XMLString::transcode(toCatch.getMessage());
    cout << "Exception message is: \n"
         << message << "\n";
    XMLString::release(&message);
    return -1;
}
catch (const SAXParseException& toCatch) {
    char* message = XMLString::transcode(toCatch.getMessage());
    cout << "Exception message is: \n"
         << message << "\n";
    XMLString::release(&message);
    return -1;
}
catch (...) {
    cout << "Unexpected Exception \n" ;
    return -1;
}

delete parser;
delete defaultHandler;
return 0;
}

```

The next section lists the SAX2XMLReader APIs and any unique behaviors for the z/OS specific parser classes.

Table 10. SAX2XMLReader APIs and behaviors in z/OS

SAX2XMLReader APIs	Unique class behavior in z/OS environment
virtual ContentHandler * getContentHandler() const =0	None
virtual const XMLCh * getURIText (unsigned int urild) const =0	None
virtual unsigned int getSrcOffset () const =0	Not supported, SAXNotSupportedException thrown.
virtual DTDHandler * getDTDHandler () const =0	Always returns NULL
virtual Grammar * getRootGrammar ()=0	Always returns NULL Grammar
virtual EntityResolver * getEntityResolver () const =0	Always returns NULL EntityResolver
virtual ErrorHandler * getErrorHandler () const =0	None
virtual bool getFeature(const XMLCh *const name) const =0	None
virtual void * getProperty (const XMLCh *const name) const =0	None
virtual void setContentHandler(ContentHandler *const handler)=0	None
virtual void setDTDHandler (DTDHandler *const handler)=0	Not supported, SAXNotSupportedException thrown.
virtual void setEntityResolver(EntityResolver *const resolver)=0	Only a NULL resolver is supported. Otherwise SAXNotSupportedException thrown.
virtual void setErrorHandler (ErrorHandler *const handler)=0	None

Table 10. SAX2XMLReader APIs and behaviors in z/OS (continued)

SAX2XMLReader APIs	Unique class behavior in z/OS environment
virtual void setFeature (const XMLCh *const name, const bool value)=0	None
virtual void parse (const InputSource &source)=0	None
virtual void parse (const XMLCh *const systemId)=0	None
virtual void parse (const char *const systemId)=0	None
virtual DeclHandler * getDeclarationHandler () const =0	Always returns a NULL DeclHandler
virtual LexicalHandler * getLexicalHandler () const =0	None
virtual void setDeclarationHandler (DeclHandler *const handler)=0	Only a NULL handler is supported. Otherwise SAXNotSupportedException thrown.
virtual PSVIHandler* getPSVIHandler() const	Always returns a NULL PSVIHandler
virtual void setPSVIHandler(PSVIHandler* const handler)	Only a NULL handler is supported. Otherwise SAXNotSupportedException thrown.
virtual XMLEntityResolver* getXMLEntityResolver() const	Always returns a NULL XMLEntityResolver
virtual void setPSVIHandler(PSVIHandler* const handler)	Only a NULL handler is supported. Otherwise SAXNotSupportedException thrown.
virtual void setXMLEntityResolver(XMLEntityResolver* const resolver)	Only a NULL XMLEntityResolver is supported. Otherwise SAXNotSupportedException thrown.
virtual void setLexicalHandler(LexicalHandler *const handler)=0	None
virtual XMLValidator * getValidator () const =0	Always returns a NULL XMLValidator
virtual int getErrorCount () const =0	Always returns either 0 or 1
virtual bool getExitOnFirstFatalError () const =0	Always returns TRUE
virtual bool getValidationConstraintFatal () const =0	Always returns TRUE
virtual Grammar * getGrammar (const XMLCh *const nameSpaceKey)=0	Always returns a NULL Grammar
virtual void setValidator (XMLValidator *valueToAdopt)=0	Not supported, throws SAXNotSupportedException
virtual void setExitOnFirstFatalError (const bool newState)=0	Only newState = TRUE is supported, FALSE will throw SAXNotSupportedException

Table 10. SAX2XMLReader APIs and behaviors in z/OS (continued)

SAX2XMLReader APIs	Unique class behavior in z/OS environment
virtual void setValidationConstraintFatal (const bool newState)=0	Only newState = TRUE is supported, FALSE will throw SAXNotSupportedException
virtual bool parseFirst (const XMLCh *const systemId, XMLPScanToken &toFill)=0	None
virtual bool parseFirst (const char *const systemId, XMLPScanToken &toFill)=0	None
virtual bool parseFirst (const InputSource &source, XMLPScanToken &toFill)=0	None
virtual bool parseNext (XMLPScanToken &token)=0	None
virtual void parseReset (XMLPScanToken &token)=0	None
virtual Grammar * loadGrammar (const InputSource &source, const short grammarType, const bool toCache=false)=0	Not supported, throws SAXNotSupportedException
virtual Grammar * loadGrammar (const XMLCh *const systemId, const short grammarType, const bool toCache=false)=0	Not supported, throws SAXNotSupportedException
virtual Grammar * loadGrammar (const char *const systemId, const short grammarType, const bool toCache=false)=0	Not supported, throws SAXNotSupportedException
virtual void resetCachedGrammarPool()=0	Not supported, throws SAXNotSupportedException
virtual void setInputBufferSize (const size_t bufferSize)	This setting is ignored
virtual void installAdvDocHandler (XMLDocumentHandler *const toInstall)=0	None
virtual bool removeAdvDocHandler (XMLDocumentHandler *const toRemove)=0	None

The following table lists the SAX2 features and any unique behaviors for the z/OS specific parser classes. These features may be set using the SAX2XMLReader::setFeature(const XMLCh* const, const bool). The current settings of a feature may be queried with the function bool SAX2XMLReader::getFeature(const XMLCh* const).

Table 11. SAX2 features and behaviors in z/OS

SAX2 features	Unique class behavior in z/OS environment
http://xml.org/sax/features/namespaces	Only TRUE is supported, FALSE will throw SAXNotSupportedException
http://xml.org/sax/features/namespace-prefixes	None
http://xml.org/sax/features/validation	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/validation/dynamic	None
http://apache.org/xml/features/validation/schema	Only FALSE is supported, TRUE will throw SAXNotSupportedException

Table 11. SAX2 features and behaviors in z/OS (continued)

SAX2 features	Unique class behavior in z/OS environment
http://apache.org/xml/features/validation/schema-full-checking	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/nonvalidating/load-external-dtd	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/continue-after-fatal-error	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/validation-error-as-fatal	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/validation/use-cachedGrammarInParse	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/validation/cache-grammarFromParse	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/standard-uri-conformant	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/calculate-src-ofs	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/validation/identity-constraint-checking	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/generate-synthetic-annotations	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/validate-annotations	Only FALSE is supported, TRUE will throw SAXNotSupportedException
http://apache.org/xml/features/schema/ignore-annotations	Only TRUE is supported, FALSE will throw SAXNotSupportedException
http://apache.org/xml/features/disable-default-entity-resolution	Only TRUE is supported, FALSE will throw SAXNotSupportedException
http://apache.org/xml/features/validation/schema/skip-dtd-validation	Only TRUE is supported, FALSE will throw SAXNotSupportedException
http://apache.org/xml/features/validation/ignoreCachedDTD	Only TRUE is supported, FALSE will throw SAXNotSupportedException

The following table lists the SAX2 Properties and any unique behavior for the z/OS specific parser classes. The SAX2XMLReader::setProperty(const XMLCh* const, void*) method can be used to set a property. However in the current implementation none of the properties below are valid. The current setting of a property would normally be queried with the the void* SAX2XMLReader::getProperty(const XMLCh*

const) method.

Table 12. SAX2 properties and behaviors in z/OS

SAX2 properties	Unique class behavior in z/OS environment
http://apache.org/xml/properties/schema/external-schemaLocation	Not supported, will throw SAXNotSupportedException
http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation	Not supported, will throw SAXNotSupportedException
http://apache.org/xml/properties/scannerName	Not supported, will throw SAXNotSupportedException
http://apache.org/xml/properties/security-manager	Not supported, will throw SAXNotSupportedException

Constructing a zXercesDOMParser

The zXercesDOMParser class provides the ability to use the z/OS XML System Services to parse an XML document. Unlike the SAX2XMLReader class which may be used as a common base class for accessing both the embedded open source parser or the new z/OS specific parser classes, the zXercesDOMParser and zAbstractDOMParser classes are both entirely new classes. The methods they support are the same as XercesDOMParser and AbstractDOMParser, however a new pointer type (zXercesDOMParser*) is required to use the z/OS specific parser classes. Using this new pointer type will result in z/OS XML System Services being used to parse the XML document instead of the embedded open source parser.

The example below shows the code you need in order to create an instance of the zXercesDOMParser class that will use the new z/OS specific parser classes. Changes from the existing XercesDOMParser implementation are in bold.

```
#include <xercesc/zparsers/zXercesDOMParser.hpp>
#include <xercesc/dom/DOM.hpp>
#include <xercesc/sax/HandlerBase.hpp>
#include <xercesc/util/XMLString.hpp>
#include <xercesc/util/PlatformUtils.hpp>

#ifdef XERCES_NEW_IOSTREAMS
#include <iostream>
#else
#include <iostream.h>
#endif

XERCES_CPP_NAMESPACE_USE

int main (int argc, char* args[]) {

    try {
        XMLPlatformUtils::Initialize();
    }
    catch (const XMLException& toCatch) {
        char* message = XMLString::transcode(toCatch.getMessage());
        cout << "Error during initialization! :\n"
             << message << "\n";
        XMLString::release(&message);
        return 1;
    }

    zXercesDOMParser* parser = new zXercesDOMParser();
    parser->setValidationScheme(zXercesDOMParser::Val_Never);
    parser->setDoNamespaces(true);
}
```

```

ErrorHandler* errHandler = (ErrorHandler*) new HandlerBase();
parser->setErrorHandler(errHandler);

char* xmlFile = "x1.xml";

try {
    parser->parse(xmlFile);
}
catch (const XMLException& toCatch) {
    char* message = XMLString::transcode(toCatch.getMessage());
    cout << "Exception message is: \n"
         << message << "\n";
    XMLString::release(&message);
    return -1;
}
catch (const DOMException& toCatch) {
    char* message = XMLString::transcode(toCatch.msg);
    cout << "Exception message is: \n"
         << message << "\n";
    XMLString::release(&message);
    return -1;
}
catch (...) {
    cout << "Unexpected Exception \n" ;
    return -1;
}

delete parser;
delete errHandler;
return 0;
}

```

The `zAbstractDOMParser` and `zXercesDOMParser` APIs are listed below along with any unique behaviors for the z/OS implementations. These APIs contain “setter” (for example, `void setDoNamespaces (const bool newState)`) and “getter” (for example, `bool getDoNamespaces () const`) methods for setting and querying the features and properties that control the behavior of the parser.

Table 13. zAbstractDOMParser APIs and behaviors in z/OS

zAbstractDOMParser APIs	Unique class behavior in z/OS environment
<code>void reset()</code>	None
<code>DOMDocument * adoptDocument()</code>	None
<code>DOMDocument * getDocument()</code>	None
<code>const XMLValidator & getValidator () const</code>	always returns a NULL XMLValidator
<code>ValSchemes getValidationScheme () const</code>	always returns <code>Val_Never</code>
<code>bool getDoSchema () const</code>	always returns <code>FALSE</code>
<code>bool getValidationSchemaFullChecking () const</code>	always returns <code>FALSE</code>
<code>bool getIdentityConstraintChecking () const</code>	always returns <code>FALSE</code>
<code>int getErrorCount () const</code>	always returns either 0 or 1
<code>bool getDoNamespaces () const</code>	always returns <code>TRUE</code>
<code>bool getExitOnFirstFatalError () const</code>	always returns <code>TRUE</code>
<code>bool getValidationConstraintFatal () const</code>	always returns <code>TRUE</code>
<code>bool getCreateEntityReferenceNodes () const</code>	always returns <code>FALSE</code>

Table 13. *zAbstractDOMParser APIs and behaviors in z/OS (continued)*

zAbstractDOMParser APIs	Unique class behavior in z/OS environment
bool getIncludeIgnorableWhitespace () const	always returns FALSE
XMLCh * getExternalSchemaLocation () const	always returns a NULL schema location
XMLCh * getExternalNoNamespaceSchemaLocation () const	always returns a NULL schema location
SecurityManager* getSecurityManager() const	always returns a NULL SecurityManager
bool getLoadExternalDTD () const	always returns FALSE
bool getCreateCommentNodes () const	always returns TRUE
bool getCalculateSrcOfs () const	always returns FALSE
bool getStandardUriConformant () const	always returns FALSE
PSVIHandler * getPSVIHandler ()	always returns a NULL PSVIHandler
const PSVIHandler * getPSVIHandler () const	always returns a NULL PSVIHandler
bool getCreateSchemaInfo () const	always returns FALSE
bool getGenerateSyntheticAnnotations () const	always returns FALSE
bool getValidateAnnotations () const	always returns FALSE
bool getIgnoreAnnotations () const	always returns TRUE
bool getDisableDefaultEntityResolution () const	always returns FALSE
bool getSkipDTDValidation () const	always returns TRUE
void setGenerateSyntheticAnnotations (const bool newValue)	only FALSE is supported, TRUE will throw DOMException
void setValidateAnnotations (const bool newValue)	only FALSE is supported, TRUE will throw DOMException
void setDoNamespaces (const bool newState)	Only TRUE is supported, FALSE will throw DOMException
void setExitOnFirstFatalError (const bool newState)	Only TRUE is supported, FALSE will throw DOMException
void setValidationConstraintFatal (const bool newState)	Only TRUE is supported, FALSE will throw DOMException
void setCreateEntityReferenceNodes (const bool create)	Only FALSE is supported, TRUE will throw DOMException
void setIncludeIgnorableWhitespace (const bool include)	Only FALSE is supported, TRUE will throw DOMException
void setValidationScheme (const ValSchemes newScheme)	Only Val_Never is supported, any other value will throw DOMException

Table 13. *zAbstractDOMParser APIs and behaviors in z/OS (continued)*

zAbstractDOMParser APIs	Unique class behavior in z/OS environment
void setDoSchema (const bool newState)	Only FALSE is supported, TRUE will throw DOMException
void setValidationSchemaFullChecking (const bool schemaFullChecking)	Only FALSE is supported, TRUE will throw DOMException
void setIdentityConstraintChecking (const bool newState)	Only FALSE is supported, TRUE will throw DOMException
void setExternalSchemaLocation (const XMLCh *const schemaLocation)	Not supported, throws a DOMException error
void setExternalSchemaLocation (const char *const schemaLocation)	Not supported, throws a DOMException error
void setExternalNoNamespaceSchemaLocation (const XMLCh *const noNamespaceSchemaLocation)	Not supported, throws a DOMException error
void setExternalNoNamespaceSchemaLocation (const char *const noNamespaceSchemaLocation)	Not supported, throws a DOMException error
void setSecurityManager (SecurityManager *const securityManager)	only a NULL securityManager is supported, otherwise throws DOMException
void setLoadExternalDTD (const bool newState)	only FALSE is supported, TRUE will throw DOMException
void setCreateCommentNodes (const bool create)	None
void setCalculateSrcOfs (const bool newState)	only FALSE is supported, TRUE will throw DOMException
void setStandardUriConformant (const bool newState)	only FALSE is supported, TRUE will throw DOMException
void useScanner (const XMLCh *const scannerName)	Not supported, throws a DOMException error
void useImplementation (const XMLCh *const implementationFeatures)	Not supported, throws a DOMException error
virtual void setPSVIHandler (PSVIHandler *const handler)	Not supported, throws a DOMException error
void setCreateSchemaInfo (const bool newState)	only FALSE is supported, TRUE will throw DOMException
void setIgnoreAnnotations (const bool newValue)	only TRUE is supported, FALSE will throw DOMException
void setDisableDefaultEntityResolution (const bool newValue)	only FALSE is supported, TRUE will throw DOMException

Table 13. *zAbstractDOMParser APIs and behaviors in z/OS (continued)*

zAbstractDOMParser APIs	Unique class behavior in z/OS environment
void setSkipDTDValidation (const bool newValue)	only TRUE is supported, FALSE will throw DOMException
void parse (const InputSource &source)	None
void parse (const XMLCh *const systemId)	None
void parse (const char *const systemId)	None
bool parseFirst (const XMLCh *const systemId, XMLPScanToken &toFill)	None
bool parseFirst (const char *const systemId, XMLPScanToken &toFill)	None
bool parseFirst (const InputSource &source, XMLPScanToken &toFill)	None
bool parseNext (XMLPScanToken &token)	None
void parseReset (XMLPScanToken &token)	None
virtual void handleElementPSVI (const XMLCh *const localName, const XMLCh *const uri, PSVIElement *elementInfo)	Not supported
virtual void handlePartialElementPSVI (const XMLCh *const localName, const XMLCh *const uri, PSVIElement *elementInfo)	Not supported
virtual void handleAttributesPSVI (const XMLCh *const localName, const XMLCh *const uri, PSVIAttributeList *psviAttributes)	Not supported
bool getDoValidation () const	Always returns FALSE
void setDoValidation (const bool newState)	only FALSE is supported, TRUE will throw DOMException
bool getExpandEntityReferences () const	Always returns TRUE
void setExpandEntityReferences (const bool expand)	only TRUE is supported, FALSE will throw DOMException
DOMNode * getCurrentNode()	None
XMLScanner * getScanner() const	None
GrammarResolver * getGrammarResolver() const	Always returns a NULL GrammarResolver
bool getParseInProgress() const	None
void setCurrentNode (DOMNode *toSet)	None
virtual DOMELEMENT * createElementNSNode (const XMLCh *fNamespaceURI, const XMLCh *qualifiedName)	None
void resetPool()	None
bool isDocumentAdopted () const	None

Table 14. zXercesDOMParser APIs and behaviors in z/OS

zXercesDOMParser APIs	Unique class behavior in z/OS environment
ErrorHandler * getErrorHandler()	None
const ErrorHandler * getErrorHandler() const	None
EntityResolver * getEntityResolver()	Always returns a NULL EntityResolver
const EntityResolver * getEntityResolver()	Always returns a NULL EntityResolver
XMLEntityResolver * getXMLEntityResolver ()	Always returns a NULL XMLEntityResolver
const XMLEntityResolver * getXMLEntityResolver () const	Always returns a NULL XMLEntityResolver
bool isCachingGrammarFromParse() const	Always returns FALSE
bool isUsingCachedGrammarInParse () const	Always returns FALSE
Grammar * getGrammar (const XMLCh *const nameSpaceKey)	Not supported, throws SAXNotSupportedException
Grammar * getRootGrammar ()	Not supported, throws SAXNotSupportedException
const XMLCh * getURIText (unsigned int urild) const	None
unsigned int getSrcOffset () const	Not supported, throws SAXNotSupportedException
bool getIgnoreCachedDTD () const	Always returns TRUE
void setErrorHandler (ErrorHandler *const handler)	None
void setEntityResolver (EntityResolver *const handler)	Only a NULL EntityResolver is supported
void setXMLEntityResolver (XMLEntityResolver *const handler)	Only a NULL XMLEntityResolver is supported
void cacheGrammarFromParse (const bool newState)	Only FALSE is supported, TRUE throws SAXNotSupportedException
void useCachedGrammarInParse (const bool newState)	Only FALSE is supported, TRUE throws SAXNotSupportedException
void setIgnoreCachedDTD (const bool newValue)	Only TRUE is supported, FALSE, throws SAXNotSupportedException
void resetDocumentPool ()	None
virtual void error (const unsigned int errCode, const XMLCh *const msgDomain, const XMLErrorReporter::ErrTypes errType, const XMLCh *const errorText, const XMLCh *const systemId, const XMLCh *const publicId, const XMLSSize_t lineNum, const XMLSSize_t colNum)	None
virtual void resetErrors ()	None
virtual void endInputSource (const InputSource &inputSource)	Not supported
virtual bool expandSystemId (const XMLCh *const systemId, XMLBuffer &toFill)	Not supported

Table 14. zXercesDOMParser APIs and behaviors in z/OS (continued)

zXercesDOMParser APIs	Unique class behavior in z/OS environment
virtual void resetEntities ()	Not supported
virtual InputSource * resolveEntity (const XMLCh *const publicId, const XMLCh *const systemId, const XMLCh *const baseURI=0)	Not supported, throws SAXNotSupportedException
virtual InputSource * resolveEntity (XMLResourceIdentifier *resourceIdentifier)	Not supported, throws SAXNotSupportedException
virtual void startInputSource (const InputSource&inputsource)	Not supported
Grammar * loadGrammar (const InputSource &source, const short grammarType, const bool toCache=false)	Not supported, throws SAXNotSupportedException
Grammar * loadGrammar (const XMLCh *const systemId, const short grammarType, const bool toCache=false)	Not supported, throws SAXNotSupportedException
Grammar * loadGrammar (const char *const systemId, const short grammarType, const bool toCache=false)	Not supported, throws SAXNotSupportedException
void resetCachedGrammarPool ()	Not supported, throws SAXNotSupportedException

Constructing a DOMBuilder

DOMBuilder is an interface introduced by the W3C DOM Level 3.0 Abstract Schemas and Load and Save Specification. DOMBuilder provides the "Load" interface for parsing XML documents and building the corresponding DOM document tree from various input sources.

A DOMBuilder instance is obtained from the DOMImplementationLS interface by invoking its createDOMBuilder method. There are no new z/OS specific classes externalized here that you need to be aware of. The only change needed to access the z/OS specific parser classes is the string that is passed in on the DOMImplementationRegistry::getDOMImplementation(const XMLCh *features) method needs to be the Unicode string "zScanner LS" instead of "LS". This will result in your code using the z/OS specific parser classes which will utilize z/OS XML System Services to parse the XML document instead of the embedded open source parser.

The example below shows the code you need in order to create an instance of DOMBuilder that will use the new z/OS specific parser classes. Changes from the existing DOMBuilder implementation are in bold.

```
#include <xercesc/dom/DOM.hpp>
#include <xercesc/util/XMLString.hpp>
#include <xercesc/util/PlatformUtils.hpp>

#ifdef XERCES_NEW_IOSTREAMS
#include <iostream>
#else
#include <iostream.h>
#endif

XERCES_CPP_NAMESPACE_USE

int main (int argc, char* args[]) {
    try {
        XMLPlatformUtils::Initialize();
```

```

    }
    catch (const XMLException& toCatch) {
        char* message = XMLString::transcode(toCatch.getMessage());
        cout << "Error during initialization! :\n"
             << message << "\n";
        XMLString::release(&message);
        return 1;
    }
}

        static const XMLCh gLS[] = {chLatin_z, chLatin_S, chLatin_c, chLatin_a, chLatin_n,
chLatin_n,
chLatin_L, chLatin_S,  chNull};

        DOMImplementation *impl = DOMImplementationRegistry::getDOMImplementation(gLS);
        DOMBuilder* parser = ((DOMImplementationLS*)impl)->createDOMBuilder(DOMImplementationLS::MODE_SYNCHRONOUS, 0);

        // optionally you can set some features on this builder
        if (parser->canSetFeature(XMLUni::fgDOMValidation, false))

parser->setFeature(XMLUni::fgDOMValidation, false);
        if (parser->canSetFeature(XMLUni::fgDOMNamespaces, true))
            parser->setFeature(XMLUni::fgDOMNamespaces, true);
        // optionally you can implement your DOMErrorHandler (e.g. MyDOMErrorHandler)
        // and set it to the builder
        MyDOMErrorHandler* errHandler = new myDOMErrorHandler();
        parser->setErrorHandler(errHandler);

        char* xmlFile = "x1.xml";
        DOMDocument *doc = 0;

        try {
            doc = parser->parseURI(xmlFile);
        }
        catch (const XMLException& toCatch) {
            char* message = XMLString::transcode(toCatch.getMessage());
            cout << "Exception message is: \n"
                 << message << "\n";
            XMLString::release(&message);
            return -1;
        }
        catch (const DOMException& toCatch) {
            char* message = XMLString::transcode(toCatch.msg);
            cout << "Exception message is: \n"
                 << message << "\n";
            XMLString::release(&message);
            return -1;
        }
        catch (...) {
            cout << "Unexpected Exception \n" ;
            return -1;
        }

        parser->release();
        delete errHandler;
        return 0;
    }
}

```

The next section lists the DOMBuilder APIs and any unique behaviors for the z/OS specific parser classes.

Table 15. DOMBuilder APIs and behaviors in z/OS

DOMBuilder APIs	Unique class behaviors in z/OS environment
virtual DOMErrorHandler * getErrorHandler ()=0	None
virtual const DOMErrorHandler * getErrorHandler () const =0	None
virtual DOMEntityResolver * getEntityResolver ()=0	Always returns a NULL DOMEntityResolver
virtual const DOMEntityResolver * getEntityResolver () const =0	Always returns a NULL DOMEntityResolver
virtual XMLEntityResolver* getXMLEntityResolver()	Always returns a NULL XMLEntityResolver
virtual const XMLEntityResolver* getXMLEntityResolver() const	Always returns a NULL XMLEntityResolver
virtual DOMBuilderFilter * getFilter ()=0	Always returns a NULL filter
virtual const DOMBuilderFilter * getFilter () const =0	Always returns a NULL filter
virtual void setErrorHandler (DOMErrorHandler *const handler)=0	None
virtual void setEntityResolver (DOMEntityResolver *const handler)=0	Only a NULL DOMEntityResolver is supported. Otherwise throws DOMException.
virtual void setXMLEntityResolver (XMLMEntityResolver *const handler)=0	Only a NULL XMLEntityResolver is supported. Otherwise throws DOMException.
virtual void setFilter (DOMBuilderFilter *const filter)=0	Not supported. Throws DOMException.
virtual void setFeature (const XMLCh *const name, const bool state)=0	None
virtual bool getFeature (const XMLCh *const name) const =0	None
virtual bool canSetFeature (const XMLCh *const name, const bool state) const = 0	None
virtual DOMDocument * parse (const DOMInputSource &source)=0	None
virtual DOMDocument * parseURI (const XMLCh *const systemId)=0	None
virtual DOMDocument * parseURI (const char *const systemId)=0	None
virtual void parseWithContext (const DOMInputSource &source, DOMNode *const contextNode, const short action)=0	None
virtual void * getProperty (const XMLCh *const name) const =0	None
virtual void setProperty (const XMLCh *const name, void *value)=0	None
virtual void release ()=0	None
virtual void resetDocumentPool ()=0	None

Table 15. DOMBuilder APIs and behaviors in z/OS (continued)

DOMBuilder APIs	Unique class behaviors in z/OS environment
virtual Grammar * loadGrammar (const DOMInputSource &source, const short grammarType, const bool toCache=false)=0	Not supported. Throws DOMException.
virtual Grammar * loadGrammar (const XMLCh *const systemId, const short grammarType, const bool toCache=false)=0	Not supported. Throws DOMException.
virtual Grammar * loadGrammar (const char *const systemId, const short grammarType, const bool toCache=false)=0	Not supported. Throws DOMException.
virtual Grammar * getGrammar (const XMLCh *const nameSpaceKey) const =0	Not supported. Throws DOMException.
virtual Grammar * getRootGrammar () const = 0	Not supported. Throws DOMException.
virtual const XMLCh * getURIText (unsigned int urild) const =0	Not supported. Throws DOMException.
virtual void resetCachedGrammarPool ()=0	Not supported. Throws DOMException.
virtual unsigned int getSrcOffset () const =0	Not supported. Throws DOMException.

Table 16. DOMImplementation APIs and behaviors in z/OS

DOMImplementation APIs	Unique class behaviors in z/OS environment
virtual DOMDocument * createDocument (MemoryManager *const manager=XMLPlatformUtils::fgMemoryManager)=0	None
static zzDOMImplementationImpl * getImplementation ()	None
static bool loadDOMExceptionMsg (const DOMException::ExceptionCode msgToLoad, XMLCh *const toFill, const unsigned int maxChars)	None
static bool loadDOMExceptionMsg (const DOMRangeException::RangeExceptionCode msgToLoad, XMLCh *const toFill, const unsigned int maxChars)	None
virtual bool hasFeature (const XMLCh *feature, const XMLCh *version) const =0	None
virtual DOMDocumentType * createDocumentType (const XMLCh *qualifiedName, const XMLCh *publicId, const XMLCh *systemId)=0	None
virtual DOMDocument * createDocument (const XMLCh *namespaceURI, const XMLCh *qualifiedName, DOMDocumentType *doctype, MemoryManager *const manager=XMLPlatformUtils::fgMemoryManager)=0	None
virtual zzDOMImplementationImpl * getInterface (const XMLCh *feature)=0	None
virtual DOMBuilder * createDOMBuilder (const short mode, const XMLCh *const schemaType, MemoryManager *const manager=XMLPlatformUtils::fgMemoryManager, XMLGrammarPool *const gramPool=0)=0	None
virtual DOMInputSource * createDOMInputSource ()=0	None

Table 16. DOMImplementation APIs and behaviors in z/OS (continued)

DOMImplementation APIs	Unique class behaviors in z/OS environment
virtual DOMImplementation* zDOMImplementationImpl::getInterface(const XMLCh*)	Not supported. Throws DOMException.

The following DOMBuilder and Xerces features can be set using the DOMBuilder::setFeature(const XMLCh* const, const bool) method. The current setting can be queried using the DOMBuilder::getFeature(const XMLCh* const) method. The DOMBuilder::canSetFeature(const XMLCh* const, const bool) method can be called to determine whether setting a feature to a specific value is supported.

Table 17. DOMBuilder features and behaviors in z/OS

DOMBuilder features	Unique class behaviors in z/OS environment
cdata-sections	Only TRUE is supported, FALSE throws SAXNotSupportedException
comments	
charset-overrides-xml-encoding	
datatype-normalization	
entities	
canonical-form	
infoset	Only FALSE is supported, TRUE throws SAXNotSupportedException
namespaces	Only TRUE is supported, FALSE throws SAXNotSupportedException
namespace-declarations	Only TRUE is supported, FALSE throws SAXNotSupportedException
supported-mediatypes-only	
validate-if-schema	
validation	
whitespace-in-element-content	

Table 18. Xerces features and behaviors in z/OS

Xerces features	Unique class behaviors in z/OS environment
http://apache.org/xml/features/validation/schema	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/validation/schema-full-checking	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/nonvalidating/load-external-dtd	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/continue-after-fatal-error	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/validation-error-as-fatal	

Table 18. Xerces features and behaviors in z/OS (continued)

Xerces features	Unique class behaviors in z/OS environment
http://apache.org/xml/features/validation/cache-grammarFromParse	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/standard-uri-conformant	None
http://apache.org/xml/features/calculate-src-ofs	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/validation/identity-constraint-checking	Only FALSE is supported, TRUE throws SAXNotSupportedException
http://apache.org/xml/features/generate-synthetic-annotations	None
http://apache.org/xml/features/validate-annotations	None
http://apache.org/xml/features/schema/ignore-annotations	None
http://apache.org/xml/features/disable-default-entity-resolution	None
http://apache.org/xml/features/validation/schema/skip-dtd-validation	None
http://apache.org/xml/features/validation/ignoreCachedDTD	None
http://apache.org/xml/features/dom-has-psvi-info	None
http://apache.org/xml/features/dom/user-adopts-DOMDocument	None

The following Xerces properties would normally be set using the `DOMBuilder::setProperty(const XMLCh* const, const bool)` method. The current setting would be queried using the `DOMBuilder::getProperty(const XMLCh* const)` method. However, in the current implementation, none of these properties are supported.

Table 19. Xerces properties and behaviors in z/OS

Xerces properties	Unique class behaviors in z/OS environment
http://apache.org/xml/properties/schema/external-schemaLocation	Not supported, throws SAXNotSupportedException
http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation	Not supported, throws SAXNotSupportedException
http://apache.org/xml/properties/scannerName	Not supported, throws SAXNotSupportedException
http://apache.org/xml/properties/parser-use-DOMDocument-from-Implementation	Not supported, throws SAXNotSupportedException
http://apache.org/xml/properties/security-manager	Only a NULL security manager is supported, otherwise throws SAXNotSupportedException

Using samples for the z/OS specific parser classes

Samples have been provided below to enable use of the z/OS specific parser classes. The following samples are located in the `zsamples` directory:

zSAX2Count

This program creates a SAX2XMLReader object that will use z/OS XML System Services to parse the XML document(s). Upon completion of the parse, it prints the number of elements, attributes, spaces and characters found in each XML file.

zSAX2Print

This program creates a SAX2XMLReader object that will use z/OS XML System Services to parse an XML document. It then prints the data returned by the various SAX2 handlers for the specified XML file.

zDOMCount

This program invokes DOMBuilder to create a parser object that uses z/OS XML System Services to parse the input document(s). It then builds the DOM tree, and then prints the number of elements found in each XML file.

zDOMPParse

This program demonstrates the progressive parse capabilities of the parser using z/OS XML System Services to parse an XML document. It does a scanFirst() call followed by a loop which calls scanNext(). Upon completion of the scanNext() loop, it reports the number of elements encountered during the parse.

zDOMPrint

This program invokes the z/OS XML System Services parser to parse the input document and build the DOM tree. It then utilizes DOMWriter to serialize the DOM tree.

zMemParse

This program uses the z/OS XML System Services parser to parse a memory buffer containing XML statements. It reports the number of elements, attributes, spaces and characters encountered during the parse.

zPParse

This program demonstrates the progressive parse capabilities of the z/OS XML System Services parser. It does a scanFirst() call followed by a loop which calls scanNext(). Upon completion of the scanNext() loop, it reports the number of elements, attribute, spaces, and characters encountered during the parse.

Rule: These samples are only examples of how to use the specific z/OS parser classes. 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. The process for building and running these samples is the same as for the open source parser classes in XML Parser, C++ Edition. See Chapter 5, "How to use the XML Parser, C++ Edition," on page 21 for more information on this.

Note: When following the instructions in Chapter 5, "How to use the XML Parser, C++ Edition," on page 21, make sure to use the zsamples directory in place of the samples directory. Also, if you do not build the zsamples using XPLINK, then the following environment variable needs to be set in order to prevent a failure:

```
export _CEE_RUNOPTS="XPLINK(ON)"
```

Chapter 7. 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 52 section for instructions on how to use these pre-built versions.

Rule: The prebuilt samples shipped with the Toolkit are the non-XPLINK versions. If you want to use the XPLINK versions of these samples, you must build your own copy of them.

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 53 and “Building sample applications for the MVS Environment” on page 57. The procedures for using your built samples are outlined in sections “Using your sample applications on the z/OS UNIX Environment” on page 56 and “Using your sample applications on the MVS Environment” on page 60

The XSLT Processor, C++ Edition component is installed in `/usr/lpp/ixm/IBM/xs1t4c-1_10` 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 XALANCR00T=/usr/lpp/ixm/IBM/xs1t4c-1_10
```

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 XERCECR00T=/usr/lpp/ixm/IBM/xm14c-5_6
```

Next, type in the following command statements:

```
export LIBPATH=$XALANCR00T/lib:$XERCESSCR00T/lib:$LIBPATH
export PATH=$XALANCR00T/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 $XALANCR00T/samples/SimpleTransform/foo.* .
```

You are now set to run the sample applications. For example, to run the SimpleTransform application, in the \$XALANCR00T/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 for running non-XPLINK samples

In order to run the non-XPLINK 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 this 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.

Rule for running XPLINK samples

In order to run the XPLINK samples, the XSLT Processor, C++ Edition requires the run-time libraries provided by Language Environment, SCEERUN and SCEERUN2, to be made available in the program search order. The best way to do this is by adding the SCEERUN and SCEERUN2 data sets into the LNKLST. If you do not wish to add SCEERUN and SCEERUN2 to the LNKLST, access SCEERUN and SCEERUN2 data sets 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 20. Product Files Required to Build Sample XML Applications for z/OS UNIX Environments

Product file name	Product file description
non-XPLINK product files	
libxslt4c.1_10_0.x	the definition side-deck that describes the XSLT Processor, C++ Edition functions and the variables
libxml4c5_6_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.
libxml4c-depdom5_6_0.x	The definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs.
XPLINK product files	
libxslt4c.1_10_0.xplink.x	The definition side-deck that describes the XSLT Processor, C++ Edition functions and the variables. This is required in order to use XPLINK to bind application code.
libxml4c5_6_0.xplink.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 use XPLINK to bind application code.
libxml4c-depdom5_6_0.xplink.x	The definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs and XPLINK.

Rules for invoking the XSLT Processor, C++ Edition in z/OS UNIX

Any application that is to invoke the XSLT Processor, C++ Edition processor under the z/OS UNIX System Services environment must include libxslt4c.1_10_0.x and libxml4c5_6_0.x, (or libxslt4c.1_10_0.xplink.x and libxml4c5_6_0.xplink.x if using XPLINK), when they bind. The binder uses the definition side-deck to resolve references to functions and variables defined in libxslt4c.1_10_0.dll and libxml4c5_6_0.dll (libxslt4c.1_10_0.xplink.dll and libxml4c5_6_0.xplink.dll if using XPLINK).

If you are using the deprecated DOM APIs, you need to include the libxml4c-depdom5_6_0.x sidedeck, or for XPLINK applications, the libxml4c-depdom5_6_0.xplink.x sidedeck.

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

```
export XALANCR00T=/usr/lpp/ixm/IBM/xslt4c-1_10
```

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 XERCECROOT=/usr/lpp/ixm/IBM/xml4c-5_6
```

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_10/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.

The next statement is only required for building XPLINK samples:

```
export OS390_XPLINK=1
```

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

Note: The below `export` statement is not required if using XPLINK; the contents of the `_CXX_PSYSIX` variable are loaded by default when using XPLINK. Also, if `_CXX_PSYSIX` was previously set and you are now building an XPLINK version, unset the variable.

For z/OS 1.9 or higher releases:

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

For z/OS 1.8 and earlier releases:

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

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

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

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 $XALANCOU/samples
gmake
```

After issuing the `gmake` command, the build process is completed. The samples are built into the `$XALANCOU/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 `$XALANCR00T/lib` and `$XERCESCRO0T/lib` directories. The file names and their descriptions are displayed in the following table:

Table 21. Library Files Required to Run Sample XML Applications on z/OS UNIX

Library File Name	Library File Description
non-XPLINK library files	
libxslt4c.1_10_0.dll, libxslt4cMessages.1_10_0.dll	XSLT Processor, C++ Edition library files
libxml4c5_6_0.dll	XML Parser, C++ Edition library file
libxml4c-depdom5_6_0.dll	library file for the previously deprecated DOM API
libicudata33.0.dll, libicudata_stub33.0.dll, libicuuc33.0.dll, libicui18n33.0.dll	ICU library files
XPLINK library files	
libxslt4c.1_10_0.xplink.dll, libxslt4cMessages.1_10_0.xplink.dll	XSLT Processor, C++ Edition library files
libxml4c5_6_0.xplink.dll	XML Parser, C++ Edition library file
libxml4c-depdom5_6_0.xplink.dll	library file for the previously deprecated DOM API
libicudata33.0.xplink.dll, libicudata_stub33.0.xplink.dll, libicuuc33.0.xplink.dll, libicui18n33.0.xplink.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 XALANCR00T=/usr/lpp/ixm/IBM/xslt4c-1_10
```

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 XERCECROOT=/usr/lpp/ixm/IBM/xml4c-5_6
```

Then type in the following command statements:

```
export LIBPATH=$XALANCR00T/lib:$XERCECROOT/lib:$LIBPATH
export ICU_DATA=$XERCECROOT/lib
```

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

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

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

```
cd $XALANCOU/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/bpxalty1.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 22. 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.	
non-XPLINK product files		
IXMLC20X	Definition side-deck that describes the XSLT Processor, C++ Edition functions and variables.	hlq.SIXMEXP
IXM4C56X	Definition side-deck that describes the XML Parser, C++ Edition functions and the variables.	hlq.SIXMEXP
IXMDD56X	Definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs.	hlq.SIXMEXP
XPLINK product files		
IXMLX20X	Definition side-deck that describes the XSLT Processor, C++ Edition functions and variables. This is required in order to use XPLINK to bind application code.	hlq.SIXMEXP
IXM4C6XX	XPLINK definition side-deck that describes the XML Parser, C++ Edition functions and the variables. This is required in order to use XPLINK to bind application code.	hlq.SIXMEXP
IXMDD6XX	Definition side-deck that describes the previously deprecated DOM APIs; it is only required if you are using these APIs and XPLINK.	hlq.SIXMEXP

Rule: Any non-XPLINK application that is to invoke the XSLT Processor, C++ Edition parser under the native MVS environment must include the IXMLC20X and IXM4C56X definition side-decks when they bind. The binder uses the definition side-decks to resolve references to functions and variables defined in the IXMLC20 and IXM4C56. Any XPLINK application that is to invoke the XSLT Processor, C++ Edition parser under the native MVS environment must include the IXMLX20X and IXM4C6XX definition side-decks when they bind. The binder uses the definition side-decks to resolve references to functions and variables defined in the IXMLX20 and IXM4C6X.

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, 400 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.V190** .LOAD for samples from Toolkit V1.9.0 and **SAMPLES.V180** .LOAD for samples from Toolkit V1.8.0.

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 XALANCR00T=/usr/lpp/ixm/IBM/xslt4c-1_10
```

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_6
```

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 XALANCOU=$HOME/mysamples
```

3. Copy the samples directory to your directory:

```
cp -r /usr/lpp/ixm/IBM/xslt4c-1_10/samples $XALANCOU
```

Since the XALANCOU 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.

The next statement is only required if you are building XPLINK samples:

```
export OS390_XPLINK=1
```

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

Note: The below `export` statement is not required if using XPLINK; the contents of the `_CXX_PSYSIX` variable are loaded by default when using XPLINK. Also, if `_CXX_PSYSIX` was previously set and you are now building an XPLINK version, unset the variable.

For z/OS 1.9 or higher releases:

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

For z/OS 1.8 and earlier releases:

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

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

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

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 23. Library Files Required to Run Sample XML Applications on MVS

Library file name	Library file description	Library data set name
non-XPLINK library files		
IXMLC20	XSLT Processor, C++ Edition library file	hlq.SIXMLOD1
IXMMSG20	XSLT Processor, C++ Edition message handling	hlq.SIXMLOD1
IXM4C56	XML Parser, C++ Edition library file	hlq.SIXMLOD1

Table 23. Library Files Required to Run Sample XML Applications on MVS (continued)

Library file name	Library file description	Library data set name
non-XPLINK library files		
IXMDD56	previously deprecated DOM API library file	hlq.SIXMLOD1
IXMI33UC	ICU library file	hlq.SIXMLOD1
IXMI33DA	ICU library file	hlq.SIXMLOD1
IXMI33D1	ICU library file	hlq.SIXMLOD1
IXMI33IN	ICU library file	hlq.SIXMLOD1
XPLINK library files		
IXMLX20	XSLT Processor, C++Edition library file	hlq.SIXMLOD1
IXMMXG20	XSLT Processor, C++ Edition message handling	hlq.SIXMLOD1
IXM4C6X	XML Parser, C++ Edition library file	hlq.SIXMLOD1
IXMDD6X	previously deprecated DOM API library file	hlq.SIXMLOD1
IXMI33XC	ICU library file	hlq.SIXMLOD1
IXMI33XA	ICU library file	hlq.SIXMLOD1
IXMI33X1	ICU library file	hlq.SIXMLOD1
IXMI33XN	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 8. 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 *XALANCR00T* to be `/usr/lpp/ixm/IBM/xslt4c-1_10`
2. Set *XERCESCRO0T* to be `/usr/lpp/ixm/IBM/xml4c-5_6`
3. Set the *PATH* to include `$XALANCR00T/bin`
4. Set the *LIBPATH* to include `$XALANCR00T/lib:$XERCESCRO0T/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
      $XALANCR00T/samples/SimpleTransform/foo.xml
      $XALANCR00T/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 (IXMXAL20):

```
//XALAN1 JOB REGION=0M,NOTIFY=&SYSUID
//STEP1 EXEC PGM=IXMXAL20
// PARM='/-e ibm-1047-s390 -o DD:OUTFILE DD:INXML DD:INXSL'
//STEPLIB DD DSN=h1q.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 24. 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 24. 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 *XALANCRROOT* to be /usr/lpp/ixm/IBM/xslt4c-1_10
2. Set *XERCESSROOT* to be /usr/lpp/ixm/IBM/xml4c-5_6
3. Set the *PATH* to include \$XALANCRROOT/bin
4. Set the *LIBPATH* to include \$XALANCRROOT/lib:\$XERCESSROOT/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 $XALANCRROOT/samples/SimpleTransform/foo.xml
          -XSL $XALANCRROOT/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 (IXMTST20):

```
//TSTXSLT1 JOB REGION=0M,NOTIFY=&SYSUID
//STEP1 EXEC PGM=IXMTST20
// PARM='/-IN FILE:////FOO.XML -XSL FILE:////FOO.XSL -OUT FOO.OUT'
//STEPLIB DD DSN=h1q.SIXMLOD1,DISP=SHR,
//*
```

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

Table 25. 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 25. 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 9. 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-oriented architecture (SOA) and Web Services. This document can be accessed from the following link:
<http://www.redbooks.ibm.com/redbooks/pdfs/sg246826.pdf>

Appendix A. Building samples for native MVS using JCL

Building XML Parser, C++ Edition samples for native MVS using JCL

The samples for the XML Parser, C++ Edition, which are included in the XML Toolkit for z/OS (XML Toolkit), reside in the product HFS. They may be compiled and linked into either an HFS or into a PDSE data set depending upon your preference. However, the instructions provided only deal with building the samples from the HFS using the gmake utility under z/OS UNIX System Services (z/OS UNIX). Alternatively, it is possible to copy the sample C++ source code and header files to PDSE data sets, and compile and link-edit them from TSO using JCL instead of using gmake from z/OS UNIX. All of the header files needed from the XML Toolkit will still have to be picked up out of the product HFS because of their long names and hierarchical structure.

In order to illustrate an application whose source code resides in data sets, we will use the source and header files for the SAXCount sample which reside in the /usr/lpp/ixm/IBM/xml4c-5_6/samples/SAXCount directory and copy them to data sets.

You first need to allocate the following PDSE data sets:

```
[userid].BATC.H.CPP -- recfm=FB, lrecl=240, blksize=12960
[userid].BATC.H.HPP -- recfm=FB, lrecl=240, blksize=12960
[userid].BATC.H.JCL -- recfm=FB, lrecl=80, blksize=12960
[userid].BATC.H.OBJ -- recfm=FB, lrecl=80, blksize=12960
[userid].BATC.H.LOAD -- recfm=U, lrecl=0, blksize=32760
```

Then you need to copy the SAXCount.cpp and SAXCountHandlers.cpp files to the [userid].BATC.H.CPP PDSE. Since the member names in a PDSE may not exceed 8 characters, you will need to rename the SAXCountHandlers.cpp part to SAXCONTH. For SAXCount.cpp, you can use a member name of SAXCOUNT. Since files in the HFS can have more than 80 byte records in them, a logical record length of 240 is used here to avoid truncating any code.

The corresponding header files (SAXCount.hpp and SAXCountHandlers.hpp) also need to be copied to the [userid].BATC.H.HPP PDSE. Use SAXCONTH for the member name for the SAXCountHandlers.hpp member here too.

Once you have copied all of these files, you need to edit the SAXCOUNT member of the [userid].BATC.H.HPP PDSE and change the following line:

```
#include "SAXCountHandlers.hpp"
```

to

```
#include "SAXConth.hpp"
```

This now refers to the 8 character name we copied the header file to in the PDSE. The next step is to copy the following JCL to the [userid].BATC.H.JCL PDSE. This JCL will compile the SAXCOUNT and SAXCONTH members and store the object files in [userid].BATC.H.OBJ.

```
//SAXCOMP JOB MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID.
//JOBLIB DD DSN=SYS1.CEE.SCEERUN,DISP=SHR
// DD DSN=SYS1.CEE.SCEERUN2,DISP=SHR
// DD DSN=SYS1.CBC.SCCNCMP,DISP=SHR
//STEP1 EXEC PGM=CCNDVR,PARM='/CXX OPTFILE(DD:OPTS),OBJ,LIST'
//OPTS DD *
LANGLVL(EXTENDED)
```

```

NOSEARCH SEARCH(
    /usr/lpp/ixm/IBM/xml4c-5_6/include/,
    //'[userid].BATCH.+ ',
    //'SYS1.CEE.SCEEH.+ ',
    //'SYS1.CBC.SCLBH.+ ')
DEFINE(OS390=1)
DEFINE(_OPEN_THREADS=1)
DEFINE(_XOPEN_SOURCE_EXTENDED=1)
/*
//SYSLIN DD DSNAME=&SYSUID..BATCH.OBJ(SAXCOUNT),DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN DD DSNAME=&SYSUID..BATCH.CPP(SAXCOUNT),DISP=SHR
//SYSUT1 DD DUMMY
/*
//STEP2 EXEC PGM=CCNDVR,PARM='/CXX OPTFILE(DD:OPTS),OBJ,LIST'
//OPTS DD *
LANGLVL(EXTENDED)
NOSEARCH SEARCH(
    /usr/lpp/ixm/IBM/xml4c-5_6/include/,
    //'[userid].BATCH.+ ',
    //'SYS1.CEE.SCEEH.+ ',
    //'SYS1.CBC.SCLBH.+ ')
DEFINE(OS390=1)
DEFINE(_OPEN_THREADS=1)
DEFINE(_XOPEN_SOURCE_EXTENDED=1)
/*
//SYSLIN DD DSNAME=&SYSUID..BATCH.OBJ(SAXCONTH),DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN DD DSNAME=&SYSUID..BATCH.CPP(SAXCONTH),DISP=SHR
//SYSUT1 DD DUMMY
/*

```

In this JCL, if you allocated the data sets with your TSO userid and run from that ID, you can leave "&SYSUID" as the high-level qualifier. In the compiler options (under the OPTS DD statement), you need to change [userid] to the high-level qualifier of the [userid].BATCH.HPP data set. If your system does not use the "SYS1" prefix on the CEE.SCEEH and CBC.SCLBH data sets, you need to remove that qualifier as well. What this SEARCH option does is instruct the compiler to first look in /usr/lpp/ixm/IBM/xml4c-5_6/include/ for header files, then in the [userid].BATCH.HPP data set, and so on and so forth.

The options could actually be stored in a data set and that data set name used on the OPTS DD statement, but the options are shown here to make the example complete. You can submit this job and it will create the SAXCOUNT and SAXCONTH object files in the [userid].BATCH.OBJ PDSE.

The next step is to link-edit (bind) these object files into an executable file. You can use the following JCL to accomplish this:

```

//SAXBIND JOB MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID.
//BIND1 EXEC PGM=IEWL,PARM='OPTIONS=OPTS'
//OPTS DD *
    AMODE=31,RMODE=ANY
    DYNAM=DLL,ALIASES=NO,UPCASE=NO,
    LIST=NO,MAP=NO,XREF=NO,MSGLEVEL=4,
    REUS=RENT,EDIT=YES,AC=0,CALL=YES,CASE=MIXED
/*
//SYSLIB DD DISP=SHR,DSN=SYS1.CEE.SCEELKEX
// DD DISP=SHR,DSN=SYS1.CEE.SCEELKED
// DD DISP=SHR,DSN=SYS1.CEE.SCEECP
// DD DISP=SHR,DSN=SYS1.CBC.SCLBSID
//SYSLIB1 DD DISP=SHR,DSN=SYS1.SIXMEXP
//SYSLIB2 DD DISP=SHR,DSN=&SYSUID..BATCH.OBJ
//SYSLMOD DD DISP=SHR,DSN=&SYSUID..BATCH.LOAD

```

```

//SYSDEFSD DD DUMMY
//SYSPRINT DD SYSOUT=A
//SYSLIN DD *
    INCLUDE SYSLIB(IOSTREAM)
    INCLUDE SYSLIB(COMPLEX)
    INCLUDE SYSLIB1(IXM4C56X)
    INCLUDE SYSLIB2(SAXCOUNT)
    INCLUDE SYSLIB2(SAXCONTH)
    ENTRY CEESTART
    NAME SAXCOUNT(R) RC=0
/*

```

In this JCL, you can also leave “&SYSUID” there as long as you are running this from your TSO ID and it matches the high-level qualifier you allocated these data sets under. On the SYSLIB DD statements, if “SYS1” is not the high-level qualifier for these data sets, you will need to remove or replace that. The SYSLIB1 DD statement assumes the side-decks for the XML Toolkit were installed as recommended and they are in SYS1.SIXMEXP. You can submit this JCL, and it should link-edit the SAXCOUNT and SAXCONTH object files into a single executable file called SAXCOUNT in [userid].BATCH.LOAD.

If you want to execute the SAXCOUNT executable, you can use the following JCL:

```

//SAXCOUNT JOB MSGLEVEL=1,REGION=0M,NOTIFY=&SYSUID.
//JOB LIB DD DSN=SYS1.SIXMLOD1,DISP=SHR
// DD DSN=&SYSUID..BATCH.LOAD,DISP=SHR
//TEST1 EXEC PGM=SAXCOUNT,
// PARM='//usr/lpp/ixm/IBM/xml4c-5_6/samples/data/personal.xml'
/*

```

This JCL assumes that the XML Toolkit DLLs were installed as recommended to the SYS1.SIXMLOD1 data set.

Building XSLT Processor, C++ Edition samples for native MVS using JCL

The samples for the XSLT Processor, C++ Edition, may also be built using JCL. This is very similar to the process for building the XML Parser, C++ Edition samples. You should review that section first. You will need to allocate the same MVS data sets. For the XSLT Processor we will use the SimpleTransform sample as an example. This resides in the /usr/lpp/ixm/IBM/xml4c-1_10/samples/SimpleTransform directory.

The first thing you need to do is copy the SimpleTransform.cpp file to the [userid].BATCH.CPP PDSE. Since the member names in a PDSE may not exceed 8 characters, you will need to rename the SimpleTransform.cpp part to SMPLTRNS.

The corresponding header file (XalanMemoryManagerImpl.hpp) also needs to be copied to the [userid].BATCH.HPP PDSE. Use XALANMMI for the 8 character member name.

Once you have copied all of these files you need to edit the SMPLTRNS member of the [userid].BATCH.CPP PDSE and change the following line:

```
#include "XalanMemoryManagerImpl.hpp"
```

to

```
#include "XalanMMI.hpp"
```

This is so that this refers to the 8 character name we copied this header file to in the PDSE. The next step is to copy the following JCL to the [userid].BATCH.JCL PDSE. This JCL will compile the SMPLTRNS member and store the object file in [userid].BATCH.OBJ.

```
//SMPTCOMP JOB MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID.
//JOBLIB DD DSN=SYS1.CEE.SCEERUN,DISP=SHR
// DD DSN=SYS1.CEE.SCEERUN2,DISP=SHR
// DD DSN=SYS1.CBC.SCCNCMP,DISP=SHR
//STEP1 EXEC PGM=CCNDRVR,PARM='/CXX OPTFILE(DD:OPTS),OBJ,LIST'
//OPTS DD *
LANGLVL(EXTENDED)
NOSEARCH SEARCH(./,
                /usr/lpp/ixm/IBM/xml4c-5_6/include/,
                /usr/lpp/ixm/IBM/xslt4c-1_10/include/,
                /usr/lpp/ixm/IBM/xslt4c-1_10/include/xalanc/Include/,
                /usr/lpp/ixm/IBM/xslt4c-1_10/include/xalanc/XSLT,
                //'[userid].BATCH.+ ',
                //'SYS1.CEE.SCEEH.+ ',
                //'SYS1.CBC.SCLBH.+ ')
DEFINE(OS390=1)
DEFINE(_OPEN_THREADS=1)
DEFINE(_XOPEN_SOURCE_EXTENDED=1)
/*
//SYSLIN DD DSN=&SYSUID..BATCH.OBJ(SMPLTRNS),DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN DD DSN=&SYSUID..BATCH.CPP(SMPLTRNS),DISP=SHR
//SYSUT1 DD DUMMY
/*
```

The next step is to link-edit (bind) this object file into an executable file. You can use the following JCL to accomplish this:

```
//SMPTBIND JOB MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID.
//BIND1 EXEC PGM=IEWL,PARM='OPTIONS=OPTS'
//OPTS DD *
        AMODE=31,RMODE=ANY
        DYNAM=DLL,ALIASES=NO,UPCASE=NO,
        LIST=NO,MAP=NO,XREF=NO,MSGLEVEL=4,
        REUS=RENT,EDIT=YES,AC=0,CALL=YES,CASE=MIXED
/*
//SYSLIB DD DISP=SHR,DSN=SYS1.CEE.SCEELKEX
// DD DISP=SHR,DSN=SYS1.CEE.SCEELKED
// DD DISP=SHR,DSN=SYS1.CEE.SCEECPP
// DD DISP=SHR,DSN=SYS1.CEE.SCEELIB
// DD DISP=SHR,DSN=SYS1.CBC.SCLBSID
//SYSLIB1 DD DISP=SHR,DSN=SYS1.SIXMEXP
//SYSLIB2 DD DISP=SHR,DSN=&SYSUID..BATCH.OBJ
//SYSLMOD DD DISP=SHR,DSN=&SYSUID..BATCH.LOAD
//SYSDEFSD DD DUMMY
//SYSPRINT DD SYSOUT=A
//SYSLIN DD *
        INCLUDE SYSLIB(IOSTREAM)
        INCLUDE SYSLIB(COMPLEX)
        INCLUDE SYSLIB(C128N)
        INCLUDE SYSLIB1(IXM4C56X)
        INCLUDE SYSLIB1(IXMLC20X)
        INCLUDE SYSLIB2(SMPLTRNS)
        ENTRY CEESTART
/*
```

In this JCL, you can also leave “&SYSUID.” there as long as you are running this from your TSO ID and it matches the high-level qualifier you allocated these data sets under. On the SYSLIB DD statements, if “SYS1” is not the high-level qualifier for these data sets, you will need to remove or replace that. The SYSLIB1 DD

statement assumes the side-decks for the XML Toolkit were installed as recommended and they are in SYS1.SIXMEXP. You can submit this JCL, and it should link-edit the SMPLTRNS object file into an executable file called SMPLTRNS in [userid].BATCH.LOAD.

If you want to execute the SAXCOUNT executable, you can use the following JCL:

```
//SMPLTRNS JOB  MSGLEVEL=(1,1),CLASS=5,REGION=0M,NOTIFY=&SYSUID.  
//STEP1   EXEC  PGM=SMPLTRNS  
//STEPLIB DD   DSN=&SYSUID..BATCH.LOAD,DISP=SHR  
//        DD   DSN=SYS1.SIXMLOD1,DISP=SHR  
//*
```

This JCL assumes that the XML Toolkit DLLs were installed as recommended to the SYS1.SIXMLOD1 data set.

Appendix B. Calling XML Parser, C++ Edition from COBOL

Source code samples

This appendix shows an example of invoking the XML Toolkit from a COBOL application. Enterprise COBOL provides its own built-in XML processing capability, but this support lacks certain functions that you may be interested in. For example, Enterprise COBOL lacks ability to validate an XML document based on a Document Type Definition (DTD) or an XML schema, and also does not provide XSLT transformation support. However, by utilizing XML Toolkit components that do provide such capability from your COBOL application, you may be able to achieve your desired goals. The samples provided in the XML Toolkit only illustrate how to invoke the XML Parser, C++ Edition, from a strictly C++ environment.

In the following example, the COBOL program “SAXParseFrontEnd” calls the “parse_validate” method in the SAXParse C++ sample code provided here, which in turn invokes the XML Parser, C++ Edition to do the actual validation of an XML file.

The COBOL code is illustrated in “SAXParseFrontEnd (SPFE) COBOL program” on page 75.

SAXParseFrontEnd (SPFE) COBOL program

```
Process pgmname(longmixed),dll,noexportall,outdd(sysprint)
  Identification division.
    Program-id 'SAXParseFrontEnd'.
  Data division.
    Working-storage section.
    * 1 fileName pic x(80) value z'/usr/lpp/ixm/IBM/xml4c-5_6/sample
    -   's/data/personal-schema.xml'.
    1 invalidFile pic x(12) value z'DD:INVALID'.
    1 rc comp-5 pic s9(9).
    1 fnlen comp pic 99.
  Procedure division.
    * parse_validate does a single validating parse of the XML file
    *
    * First, try a valid file from the samples/data directory
    Call 'parse_validate'
      using by value address of fileName
      returning rc
    Move 0 to tally
    Inspect fileName
      tallying tally for characters before x'00'
    If rc = 0
      Display ''' fileName(1:tally) '' is valid.'
    Else
      Display 'Error! '
        ''' fileName(1:tally) '' should have been valid.'
    Move rc to return-code
    End-if
    *
    * Next, try a well-formed but invalid file, accessed by DDNAME INVALID
    Call 'parse_validate'
      using by value address of invalidFile
      returning rc
    Move 0 to tally
    Inspect invalidFile
      tallying tally for characters before x'00'
    If rc = 0
      Display 'Error! The file in ''
```

```

        invalidFile(1:tally) '" should have been invalid.'
    Move 8 to return-code
Else
    Display 'The file in "'
        invalidFile(1:tally) '" is invalid.'
End-if
Goback.
End program 'SAXParseFrontEnd'

```

The COBOL code in “SAXParseFrontEnd (SPFE) COBOL program” on page 75, first passes in by value, a null terminated string containing the fully qualified file path of an HFS file. “parse_validate” will return an integer indicating whether the XML file was valid or not valid.

The COBOL code will make a second call to “parse_validate”, also by value, which illustrates using a DDNAME instead of a file path in the HFS. In this case, the XML file passed in contains errors so validation will fail.

SAXParse.cpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes
// -----
#include <xercesc/parsers/SAXParser.hpp>
#include xercesc/util/OutOfMemoryException.hpp>
#include "SAXParse.hpp"
#include "stdio.h"

// -----
// Function prototypes
// -----
extern "C"
{
int parse_validate(char * xmlFilePath);
}

// -----
// "parse_validate" will initialize the parser environment and do a single
// parse of the XML document specified in the parameter list. It terminates
// and cleans up the environment before exiting.
// -----

int parse_validate(char * xmlFilePath)
{

```

```

bool error_occurred = false;
bool warning_occurred = false;
unsigned long duration;

SAXParser * parser;
SAXParseHandlers * handler;

try
{
    // Initialize the XML4C system

    XMLPlatformUtils::Initialize();

}
catch (const OutOfMemoryException&)
{
    XERCES_STD_QUALIFIER cerr << "OutOfMemoryException during initialization!" << XERCES_STD_QUALIFIER endl;
    return 8;
}
catch (const XMLException& toCatch)
{
    XERCES_STD_QUALIFIER cerr << "Error during initialization! Message:\n"
    << StrX(toCatch.getMessage()) << XERCES_STD_QUALIFIER endl;
    return 8;
}
catch (...)
{
    XERCES_STD_QUALIFIER cerr << "Error during initialization!"
    << XERCES_STD_QUALIFIER endl;
    return 8;
}

// Create the parser instance and set the parser options

parser = new SAXParser;
parser->setDoNamespaces(true);
parser->setDoSchema(true);
parser->setValidationSchemaFullChecking(true);
parser->setValidationScheme(SAXParser::Val_Auto);
parser->cacheGrammarFromParse(false);
handler = new SAXParseHandlers();
parser->setErrorHandler(handler);

    //
    // Kick off the parse and catch any exceptions.
    //

try
{
    const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
    parser->parse(xmlFilePath);
    const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
    duration = endMillis - startMillis;
}
catch (const OutOfMemoryException&)
{
    XERCES_STD_QUALIFIER cerr << "OutOfMemoryException during parsing!"
    << XERCES_STD_QUALIFIER endl;
    error_occurred = true;
}
catch (const XMLException& e)
{
    XERCES_STD_QUALIFIER cerr << "\nError during parsing: \n"
    << StrX(e.getMessage())
    << XERCES_STD_QUALIFIER endl;
    error_occurred = true;
}

```

```

catch (...)
{
  XERCES_STD_QUALIFIER cerr << "Error during parsing!"
  << XERCES_STD_QUALIFIER endl;
  error_occurred = true;
}

if (handler->getSawErrors()) {
  error_occurred = true;
}

if (handler->getSawWarning()) {
  warning_occurred = true;
}

// Print out the filename and time taken
if (!error_occurred) {
  XERCES_STD_QUALIFIER cout << xmlFilePath << ": " << duration << " MS " << XERCES_STD_QUALIFIER endl;
}

delete handler;
delete parser;

XMLPlatformUtils::Terminate();

if (error_occurred)
{
  return 8;
}
else if (warning_occurred)
{
  return 4;
}
else
{
  return 0;
}

// return 0;
}

```

The C++ code that the COBOL code will invoke is illustrated in “SAXParse.cpp” on page 76. The “parse_validate” method is declared as “extern C” even though it is being invoked from COBOL. This is much easier than using “extern COBOL”. The only parameter is the null terminated string which contains the file name. The “parse_validate” routine will initialize the XML parser environment and then create a parser instance and error handler. If this is successful it will then set some parser features and then attempt to parse the file name passed in. If successful the caller will receive a zero in the return code parameter. If an error occurs a non-zero return code will be returned and an appropriate error message displayed.

The parser instance and error handler are deleted and the XML parser environment is terminated before returning to the caller.

“SAXParse.hpp” on page 79 contains the header file that corresponds to the SAXParse.cpp file.

SAXParse.hpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes for all the program files to see
// -----
#include <string.h>
#include <stdlib.h>
#ifdef(XERCES_NEW_IOSTREAMS)
#include <iostream>
#else
#include <iostream.h>
#endif
#include <xercesc>
#include <xercesc>
#include "SAXParHD.hpp"

// -----
// This is a simple class that lets us do easy (though not terribly efficient)
// transcoding of XMLCh data to local code page for display.
// -----
class StrX
{
public :
// -----
// Constructors and Destructor
// -----
StrX(const XMLCh* const toTranscode)
{
// Call the private transcoding method
fLocalForm = XMLString::transcode(toTranscode);
}

~StrX()
{
XMLString::release(&fLocalForm);
}

// -----
// Getter methods
// -----
const char* localForm() const
{
return fLocalForm;
}

private :
// -----
// Private data members
//
// fLocalForm
// This is the local code page form of the string.
// -----
char* fLocalForm;

```

```
};

inline XERCES_STD_QUALIFIER ostream& operator<<(XERCES_STD_QUALIFIER ostream& target, const StrX& toDump)
{
    target << toDump.localForm();
    return target;
}
```

“SAXParseHandlers.cpp” contains the C++ code for the error handler (SAXParseHandlers.cpp).

SAXParseHandlers.cpp

```
/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes
// -----
#include <xercesc/sax/SAXParseException.hpp>
#include <xercesc/util/XMLString.hpp>
#include <xercesc/util/XMLUniDefs.hpp>
#include "SAXParse.hpp"

// -----
// SAXParseHandlers: Constructors and Destructor
// -----
SAXParseHandlers::SAXParseHandlers() :
    fSawErrors(false)
    ,fSawWarning(false)
{
}

SAXParseHandlers::~SAXParseHandlers()
{
}

// -----
// SAXParseHandlers: Overrides of the SAX ErrorHandler interface
// -----
void SAXParseHandlers::error(const SAXParseException& e)
{
    fSawErrors = true;
    XERCES_STD_QUALIFIER cerr << "\nError at (file " << StrX(e.getSystemId())
    << ", line " << e.getLineNumber()
    << ", char " << e.getColumnNumber()
    << "): " << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
}

```

```

void SAXParseHandlers::fatalError(const SAXParseException& e)
{
    fSawErrors = true;
    XERCES_STD_QUALIFIER cerr << "\nFatal Error at (file " << StrX(e.getSystemId())
    << ", line " << e.getLineNumber()
    << ", char " << e.getColumnNumber()
    << "): " << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
}

void SAXParseHandlers::warning(const SAXParseException& e)
{
    fSawWarning = true;
    XERCES_STD_QUALIFIER cerr << "\nWarning at (file " << StrX(e.getSystemId())
    << ", line " << e.getLineNumber()
    << ", char " << e.getColumnNumber()
    << "): " << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
}

void SAXParseHandlers::resetDocument()
{
    fSawWarning = false;
    fSawErrors = false;
}

```

SAXParseHandlers.cpp is similar to the existing SAXCount and SAXPrint samples in the Toolkit. In this example, only the error handler methods are implemented. Since this only illustrates the validation of an XML file, the default document handler callback methods are inherited, which will ignore the data returned.

“SAXParseHandlers.hpp” on page 81 contains the header file that corresponds to the SAXParseHandlers.cpp file.

SAXParseHandlers.hpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes
// -----
#include <xercesc/sax/HandlerBase.hpp>

XERCES_CPP_NAMESPACE_USE

class SAXParseHandlers : public HandlerBase
{
public:
// -----

```

```

// Constructors and Destructor
// -----
SAXParseHandlers();
~SAXParseHandlers();

// -----
// Handlers for the SAX ErrorHandler interface
// -----
void warning(const SAXParseException& exception);
void error(const SAXParseException& exception);
void fatalError(const SAXParseException& exception);

bool getSawErrors() const
{
    return fSawErrors;
}

bool getSawWarning() const
{
    return fSawWarning;
}

void resetDocument();

private:
bool    fSawErrors;
bool    fSawWarning;
};

```

```

<?xml version="1.0" encoding="ibm-1140"?>
<personnel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation=
    '/usr/lpp/ixm/IBM/xml4c-5_6/samples/data/personal.xsd'>

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

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

  <person id="two.worker">
    <name><family>Worker</family> <given>Two</given></name>
    <email>two@foo.com</email>
    <link manager="Big.Boss"/>
    <phone>+1.123.555.1234</phone>
  </person>

</personnel>

```

Figure 5. Non-valid XML file to be processed via DDNAME

The XML in Figure 5 should be copied to an MVS dataset so that it can be accessed by a DDNAME. The data set name, etc. are provided later in “Setup instructions” on page 84.

Compilation instructions

The JCL in Figure 6 can be used to compile, bind, and execute the COBOL and C++ code from the above section.

```
//CBLXMLVL JOB MSGLEVEL=(1,1),REGION=0M,NOTIFY=&SYSUID,MSGCLASS=H
//ORDER JCLLIB ORDER=(SYS1.ADCOB.V3R3M0.SIGYPROC,SYS1.CBC.SCCNPRC)
//*STDORD JCLLIB ORDER=(CBC.SCCNPRC,IGY.V3R4M0.SIGYPROC) !!!
//*
//*-- Compile C++ program parse_validate -----
//CPPSP EXEC CBCC,INFILE=[userid].BATCH.CPP(SAXPARSE),
// PARM.COMPILE='/CXX OPTFILE(DD:OPTIONS)'
//COMPILE.OPTIONS DD *
DEFINE(OS390=1) DEFINE(_OPEN_THREADS=1)
DEFINE(_XOPEN_SOURCE_EXTENDED=1) EXPORTALL LANG(EXTENDED)
SEARCH(
    /usr/lpp/ixm/IBM/xml4c-5_6/include/,
    //'[userid].BATCH.+ ',
    //'SYS1.CEE.SCEEH.+ ',
    //'SYS1.CBC.SCLBH.+ ')
/*
//*
//*-- Compile C++ class SAXParseHandlers and bind with parse_validate --
//CPPSPH EXEC CBCCB,INFILE=[userid].BATCH.CPP(SAXPARHD),
// PARM.COMPILE='/CXX OPTFILE(DD:OPTIONS)',
// BPARM='NOLIST,NOMAP,RMODE=ANY'
//COMPILE.OPTIONS DD *
DEFINE(OS390=1) DEFINE(_OPEN_THREADS=1)
DEFINE(_XOPEN_SOURCE_EXTENDED=1) LANG(EXTENDED)
SEARCH(
    /usr/lpp/ixm/IBM/xml4c-5_6/include/,
    //'[userid].BATCH.+ ',
    //'SYS1.CEE.SCEEH.+ ',
    //'SYS1.CBC.SCLBH.+ ')
/*
//BIND.XML4C DD DISP=SHR,DSN=SYS1.SIXMEXP
//BIND.SYSIN DD *
INCLUDE XML4C(IXM4C56X)
/*
//BIND.SYSLMOD DD DSN=&&GOSET(SAXPARSE)
//BIND.SYSDEFSD DD DSN=&&IMPXSET,UNIT=&TUNIT.,
// DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*
//*-- Compile, link and execute COBOL program SAXParseFrontEnd -----
//CBLFE EXEC IGYWCLG,
// PARM.LKED='CASE=MIXED,DYNAM=DLL,RENT,NOLIST',GOPGM=SPFE,
//* IGYWCLG not properly customized; we shouldn't need the following !!!
// LNGPRFX=SYS1.ADCOB.V3R3M0,LIBPRFX=SYS1.CEE
//COBOL.SYSIN DD DISP=SHR,DSN=[userid].BATCH.COBOL(SPFE)
//LKED.SYSLIN DD
// DD DSN=&&IMPXSET,DISP=(OLD,DELETE)
//LKED.SYSLMOD DD DSN=&&GOSET(&GOPGM)
//GO.STEPLIB DD
// DD DSN=&&GOSET,DISP=(OLD,DELETE)
// DD DISP=SHR,DSN=SYS1.SIXMLOD1
//GO.SYSOUT DD SYSOUT=*
//GO.INVALID DD DISP=SHR,DSN=[userid].BATCH.XML(INVALID)
//
```

Figure 6. JCL to compile, bind and run the sample code

This JCL uses catalogued JCL procedures to invoke the C++ Compiler, the COBOL compiler and the z/OS Binder.

The CBCC procedure will compile the SAXParse C++ code that contains “parse_validate”.

The CBCCB procedure will compile the SAXParseHandlers C++ code and then bind it into a DLL. This DLL will be linked with the XML Parser, C++ Edition's main DLL.

The IGYWCLG procedure will compile, link, and execute the COBOL SAXParseFrontEnd code.

Setup instructions

Several data sets will need to be allocated to copy the sample code into. You should substitute your actual USERID for [userid].

```
[userid].BATCH.COBOL -- recfm=FB, lrecl=240, blksize=12960 (contains COBOL code)
[userid].BATCH.CPP   -- recfm=FB, lrecl=240, blksize=12960 (contains C++ code)
[userid].BATCH.HPP   -- recfm=FB, lrecl=240, blksize=12960 (contains C++ header files)
[userid].BATCH.CNTL -- recfm=FB, lrecl=80,  blksize=12960 (contains JCL)
[userid].BATCH.OBJ   -- recfm=FB, lrecl=80,  blksize=12960 (contains object code)
[userid].BATCH.LOAD  -- recfm=U,  lrecl=0,   blksize=32760 (contains executable code)
[userid].BATCH.XML   -- recfm=FB, lrecl=80,  blksize=12960 (contains actual XML data)
```

In order to compile, link, and run this sample you will need to copy the code, header files, JCL and XML file to the above MVS data sets.

- You will need to copy the SAXParseFrontEnd COBOL code to the SPFE member in [userid].BATCH.COBOL.
- The SAXParse C++ code should be copied to the SAXPARSE member in [userid].BATCH.CPP.
- The SAXParseHandlers C++ code should be copied to the SAXPARHD member in [userid].BATCH.CPP.
- The SAXParse header file should be copied to the SAXPARSE member in [userid].BATCH.HPP.
- The SAXParseHandlers header should be copied to the SAXPARHD member in [userid].BATCH.HPP.
- The invalid XML data from Figure 6 should be copied to the INVALID member in [userid].BATCH.XML.

Once all of this is done, then the JCL in [userid].BATCH.CNTL(SPFE) can be submitted to compile, link, and run this sample code.

Note: : This example is only intended to demonstrate the usage to the XML Toolkit from COBOL. It is not intended to have high performance characteristics. For more information on how to enhance the performance of this example, refer to the Performance section of the XML Toolkit Web site or the Persistent Parser example in Appendix C, "Parser environment and instance reuse," on page 85.

Appendix C. Parser environment and instance reuse

This appendix shows you how a parser environment can be initialized and a parser instance created and saved for subsequent calls to parse documents. The overhead of initializing the environment and the creation of a parser instance (as well as the later termination overhead) can be spread across all subsequent parses instead of doing an initialize/parse/terminate sequence for each individual XML document.

In this example, the grammar may also be cached to enhance performance.

The “PersistParseFrontEnd.cpp” on page 85 code contains the main routine for this example. It accepts several command line parameters from the PersistParse command that control the settings used to create the parser instance as well as the file name to parse.

This code is referred to as a “Front End” because it differs from what the other samples in the Toolkit do in their main routine. Other samples normally process the command line arguments and then initialize the parser environment, create a parser instance, process the document and then terminate all in a single routine.

The purpose of this “Front End” module is to separate the initialization, parse, and terminate functions from the main routine and move them into separate methods in the PersistParse.cpp module. This way, the “Front End” code can process the command line arguments and then request the parser environment/instance be created by another method. When control returns to it, a pointer now exists that can be used subsequently to parse an indefinite number of documents using this existing environment and parser instance. When this environment is no longer needed, a single termination request can be performed.

The PersistParse command has a -i option that allows you to specify the number of times the document is to be parsed. This is intended to illustrate how the parser environment is initialized and terminated a single time and used across multiple parses. In more practical situations, you would use this feature to parse many different documents.

One thing to keep in mind when creating an application that needs to parse multiple documents is that since the parser instance is only created once, the options specified at the time of creation cannot be changed without terminating and re-initializing. For example, you cannot create a parser instance that does grammar caching and then process a few documents and then stop caching grammars.

“PersistParseFrontEnd.hpp” on page 89 contains the header file for the main routine “PersistParseFrontEnd.cpp” on page 85. “PersistParse.cpp” on page 90 contains the methods that initialize the environment and parser instance, parse the document, and terminate. “PersistParse.hpp” on page 93 contains the header file for the code in “PersistParse.cpp” on page 90. “PersistParseHandlers.cpp” on page 94 contains the error handler methods needed. “PersistParseHandlers.hpp” on page 95 contains the header file the code in “PersistParseHandlers.cpp” on page 94.

PersistParseFrontEnd.cpp

```
/*  
 * Copyright 1999-2006 The Apache Software Foundation.
```

```

*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

// -----
// Includes
// -----
#if defined(XERCES_NEW_IOSTREAMS)
#include "fstream"
#else
#include "fstream.h"
#endif
#include "PersistParseFrontEnd.hpp"
#include "stdio.h"

// -----
// Function prototypes
// -----
int initialize_env(parmList *);
int validate_file(parmList *);
int terminate_env(parmList *);

// -----
// Local helper methods
// -----
void usage()
{
    XERCES_STD_QUALIFIER cout << "\nUsage:\n"
        "    PersistParse [options] <XML file>\n\n"
        "This program demonstrates how to initialize the parser\n"
        "environment and parser instance a single time and then\n"
        "reuse the saved environment on later parse requests. In this\n"
        "sample the -i option allows the file specified to be parsed\n"
        "multiple times.\n\n"
        "Options:\n"
        "    -v=xxx    Validation scheme [always | never | auto*].\n"
        "    -n        Enable namespace processing. Defaults to off.\n"
        "    -s        Enable schema processing. Defaults to off.\n"
        "    -f        Enable full schema constraint checking.\n"
        "              Defaults to off.\n"
        "    -gc       Cache Grammar from parse. Defaults to off.\n"
        "    -i nnn    Parse file in loop nnn times. Default is 1 time.\n"
        "    -?        Show this help.\n"
        "    * = Default if not provided explicitly.\n"
    << XERCES_STD_QUALIFIER endl;
}

// -----
// Program entry point
// -----
int main(int argC, char* argV[])
{
    parmList SAXParms;
    int xercesc_rc = 0;

    // Initialize defaults in parameter block

```

```

SAXParms.valScheme      = SAXParser::Val_Auto;
SAXParms.doNamespaces  = false;
SAXParms.doSchema      = false;
SAXParms.schemaFullChecking = false;
SAXParms.doGrammarCaching = false;
SAXParms.repeatParse    = false;
SAXParms.numParses     = 1;
SAXParms.errorOccurred  = false;
SAXParms.warningOccurred = false;
SAXParms.xmlFile        = 0;
SAXParms.parser         = 0;
SAXParms.handler        = 0;

// If enough parameters not specified display help information
if (argC < 2)
{
    usage();
    return 4;
}

// Grab user parameter(s)

int parmInd;

for (parmInd = 1; parmInd < argC; parmInd++)
{
    // Break out on first parameter not starting with a dash
    if (argV[parmInd][0] != '-')
    {
        break;
    }

    // Watch for special case help request
    if (!strcmp(argV[parmInd], "-?"))
    {
        usage();
        return 4;
    }
    else if (!strncmp(argV[parmInd], "-v=", 3)
        || !strncmp(argV[parmInd], "-V=", 3))
    {
        const char* const parm = &argV[parmInd][3];

        if (!strcmp(parm, "never"))
            SAXParms.valScheme = SAXParser::Val_Never;
        else if (!strcmp(parm, "auto"))
            SAXParms.valScheme = SAXParser::Val_Auto;
        else if (!strcmp(parm, "always"))
            SAXParms.valScheme = SAXParser::Val_Always;
        else
        {
            XERCES_STD_QUALIFIER cerr << "Unknown -v= value: " << parm << XERCES_STD_QUALIFIER endl;
            return 8;
        }
    }
    else if (!strcmp(argV[parmInd], "-n")
        || !strcmp(argV[parmInd], "-N"))
    {
        SAXParms.doNamespaces = true;
    }
    else if (!strcmp(argV[parmInd], "-s")
        || !strcmp(argV[parmInd], "-S"))
    {
        SAXParms.doSchema = true;
    }
}

```

```

else if (!strcmp(argv[parmInd], "-f")
        || !strcmp(argv[parmInd], "-F"))
{
    SAXParms.schemaFullChecking = true;
}
        else if (!strcmp(argv[parmInd], "-gc")
                || !strcmp(argv[parmInd], "-GC"))
{
    SAXParms.doGrammarCaching = true;
}
else if (!strcmp(argv[parmInd], "-i"))
{
    ++parmInd;
    if (parmInd >= argC)
    {
        XERCES_STD_QUALIFIER cerr << "Invalid -i option (missing # of iterations)" << XERCES_STD_QUALIFIER endl;
        return 8;
    }
    SAXParms.repeatParse = true;
    SAXParms.numParses = atoi(argv[parmInd]);
    if (SAXParms.numParses < 0)
    {
        XERCES_STD_QUALIFIER cerr << "Invalid -i option (negative # of iterations)" << XERCES_STD_QUALIFIER endl;
        return 8;
    }
}
else
{
    XERCES_STD_QUALIFIER cerr << "Unknown option '" << argv[parmInd]
        << "', ignoring it\n" << XERCES_STD_QUALIFIER endl;
}
} // end - for ...

//
// There should at least one parameter left, and that
// should be the file name.

SAXParms.xmlFile = argv[parmInd];

// "initialize_env" call will do the XMLPlatformUtils::Initialize() and
// create a parser instance and error handler and store their addresses
// in the parameter block for later use.

xercesc_rc = initialize_env(&SAXParms);
if (xercesc_rc != 0)
{
    return xercesc_rc;
}

// "validate_file" call will parse the input file either once or
// a multiple number of times based on the "-i" parameter.

int i = SAXParms.numParses;
if (SAXParms.repeatParse)
{
    XERCES_STD_QUALIFIER cout << "-i parameter specified, doing parse " << SAXParms.numParses << " times" <<XERCES_STD_QUALIFIER
}

    for(; i > 0 ; i--)
    {
        xercesc_rc = validate_file(&SAXParms);
        if (xercesc_rc != 0)
        {
            break;
        }
    }
}

```

```

// "terminate_env" call will delete the error handler and parser
// instance and call XMLPlatformUtils::Terminate().

terminate_env(&SAXParms);

if (SAXParms.errorOccurred)
    return 8;
else
    return xercesc_rc;
}

```

PersistParseFrontEnd.hpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes for all the program files to see
// -----
#include <string.h>
#include <stdlib.h>
#ifdef XERCES_NEW_IOSTREAMS
#include <iostream>
#else
#include <iostream.h>
#endif
#include <xercesc/util/PlatformUtils.hpp>
#include <xercesc/parsers/SAXParser.hpp>
#include <xercesc/sax/HandlerBase.hpp>
#include "PersistParse.hpp"

// -----
// The parmList structure is used to pass parameters to the methods
// and to preserve the parser and handler pointers so that
// the parser instance can be preserved across calls to avoid the
// initialization costs.
// -----

struct parmList {
    SAXParser::ValSchemes valScheme;
    bool doNamespaces;
    bool doSchema;
    bool schemaFullChecking;
    bool doGrammarCaching;
    bool repeatParse;
    bool errorOccurred;
    bool warningOccurred;
    int numParses;
};

```

```

char * xmlFile;
SAXParser * parser;
PersistParseHandlers * handler;
};

```

PersistParse.cpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes
// -----
#include <xercesc/internal/XMLGrammarPoolImpl.hpp>
#include <xercesc/util/OutOfMemoryException.hpp>
#include "PersistParseFrontEnd.hpp"
#include "stdio.h"

// -----
// The "initialize_env" method initializes the XML environment and creates a
// parser instance and error handler and stores their addresses in the
// parameter block so that subsequent parse ("validate_file") calls can be made
// without the overhead of creating the XML environment, parser instance, etc.
// -----

int initialize_env(parmList * reqBlock)
{
    XERCES_STD_QUALIFIER cout << "Processing initialize_env request" << XERCES_STD_QUALIFIER endl;

    try
    {
        // Initialize the XML4C system
        XMLPlatformUtils::Initialize();

        // Create the parser instance that can be used later on
        // Use options from parameter block to customize the parser.

        reqBlock->parser = new SAXParser;

        if (reqBlock->doNamespaces)
        {
            (reqBlock->parser)->setDoNamespaces(reqBlock->doNamespaces);
        }

        if (reqBlock->doSchema)
        {
            (reqBlock->parser)->setDoSchema(reqBlock->doSchema);
        }

        if (reqBlock->schemaFullChecking)

```

```

    {
        (reqBlock->parser)->setValidationSchemaFullChecking(reqBlock->schemaFullChecking);
    }

    if (reqBlock->doGrammarCaching)
    {
        (reqBlock->parser)->cacheGrammarFromParse(reqBlock->doGrammarCaching);
    }

    reqBlock->handler = new PersistParseHandlers();
    (reqBlock->parser)->setErrorHandler(reqBlock->handler);
}
catch (const OutOfMemoryException&)
{
    XERCES_STD_QUALIFIER cerr << "OutOfMemoryException during initialization!"
        << XERCES_STD_QUALIFIER endl;
    return 8;
}
catch (const XMLException& toCatch)
{
    XERCES_STD_QUALIFIER cerr << "Error during initialization! Message:\n"
        << StrX(toCatch.getMessage()) << XERCES_STD_QUALIFIER endl;
    return 1;
}
catch (...)
{
    XERCES_STD_QUALIFIER cerr << "Error during initialization!"
        << XERCES_STD_QUALIFIER endl;
    return 8;
}
return 0;
} // end -- "initialize_env"

// -----
// The "validate_file" routine uses the previously established XML environment,
// parser instance, and error handler to do the actual parse request.
// -----

int validate_file(parmList * reqBlock)
{
    int rc;
    XERCES_STD_QUALIFIER cout << "Processing validate_file request" << XERCES_STD_QUALIFIER endl;

    // Make sure we have valid values
    if (reqBlock->parser == 0 || reqBlock->handler == 0)
    {
        rc = initialize_env(reqBlock);
        if (rc != 0)
        {
            reqBlock->errorOccurred = true;
            return rc;
        }
    }

    unsigned long duration;

    reqBlock->errorOccurred = false;
    reqBlock->warningOccurred = false;
    (reqBlock->handler)->resetDocument();

    // Kick off the parse and catch any exceptions.

    try
    {
        const unsigned long startMillis = XMLPlatformUtils::getCurrentMillis();
        (reqBlock->parser)->parse(reqBlock->xmlFile);
    }
}

```

```

    const unsigned long endMillis = XMLPlatformUtils::getCurrentMillis();
    duration = endMillis - startMillis;
}
catch (const OutOfMemoryException&)
{
    XERCES_STD_QUALIFIER cerr << "OutOfMemoryException during parsing!"
    << XERCES_STD_QUALIFIER endl;
    reqBlock->errorOccurred = true;
    return 8;
}
catch (const XMLException& e)
{
    XERCES_STD_QUALIFIER cerr << "\nError during parsing: \n"
    << StrX(e.getMessage())
    << XERCES_STD_QUALIFIER endl;
    reqBlock->errorOccurred = true;
    return 8;
}
catch (...)
{
    XERCES_STD_QUALIFIER cerr << "Error during parsing!"
    << XERCES_STD_QUALIFIER endl;
    reqBlock->errorOccurred = true;
    return 8;
}

if ((reqBlock->handler)->getSawErrors())
{
    reqBlock->errorOccurred = true;
    return 8;
}

if ((reqBlock->handler)->getSawWarning())
{
    reqBlock->warningOccurred = true;
    return 4;
}

// Print out the filename and time taken
if (!reqBlock->errorOccurred)
{
    XERCES_STD_QUALIFIER cout << reqBlock->xmlFile << ": " << duration << " ms " << XERCES_STD_QUALIFIER endl;
}

    return 0;
} // end -- "validate_file"

// -----
// The "terminate_env" routine will delete the previously established error
// handler and parser instance and then terminate the XML environment.
// -----
int terminate_env(parmList * reqBlock)
{
    XERCES_STD_QUALIFIER cout << "Processing terminate_env request" << XERCES_STD_QUALIFIER endl;

    delete reqBlock->handler;
    delete reqBlock->parser;

    XMLPlatformUtils::Terminate();

    return 0;
} // end -- "terminate_env"

```

PersistParseFrontEnd.hpp

```
/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes for all the program files to see
// -----
#include <string.h>
#include <stdlib.h>
#if defined(XERCES_NEW_IOSTREAMS)
#include <iostream>
#else
#include <iostream.h>
#endif
#include <xercesc/util/PlatformUtils.hpp>
#include <xercesc/parsers/SAXParser.hpp>
#include "PersistParseHandlers.hpp"

// -----
// This is a simple class that lets us do easy (though not terribly efficient)
// transcoding of XMLCh data to local code page for display.
// -----
class StrX
{
public :
// -----
// Constructors and Destructor
// -----
StrX(const XMLCh* const toTranscode)
{
// Call the private transcoding method
fLocalForm = XMLString::transcode(toTranscode);
}

~StrX()
{
XMLString::release(&fLocalForm);
}

// -----
// Getter methods
// -----
const char* localForm() const
{
return fLocalForm;
}

private :
// -----
// Private data members
//
// fLocalForm
// This is the local code page form of the string.
// -----

```

```

char * fLocalForm;
};

inline XERCES_STD_QUALIFIER ostream& operator<<(XERCES_STD_QUALIFIER ostream& target, const StrX& toDump)
{
    target << toDump.localForm();
    return target;
}

```

PersistParseHandlers.cpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes
// -----
#include <xercesc/sax/SAXParseException.hpp>
#include <xercesc/util/XMLString.hpp>
#include <xercesc/util/XMLUniDefs.hpp>
#include "PersistParse.hpp"

// -----
// PersistParseHandlers: Constructors and Destructor
// -----
PersistParseHandlers::PersistParseHandlers() :
    fSawErrors(false), fSawWarning(false)
{
}

PersistParseHandlers::~PersistParseHandlers()
{
}

// -----
// PersistParseHandlers: Overrides of the SAX ErrorHandler interface
// -----
void PersistParseHandlers::error(const SAXParseException& e)
{
    fSawErrors = true;
    XERCES_STD_QUALIFIER cerr << "\nError at (file " << StrX(e.getSystemId())
    << ", line " << e.getLineNumber()
    << ", char " << e.getColumnNumber()
    << "): " << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
}

void PersistParseHandlers::fatalError(const SAXParseException& e)
{
    fSawErrors = true;
    XERCES_STD_QUALIFIER cerr << "\nFatal Error at (file " << StrX(e.getSystemId())

```

```

    << ", line " << e.getLineNumber()
    << ", char " << e.getColumnNumber()
    << "): " << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
}

void PersistParseHandlers::warning(const SAXParseException& e)
{
    fSawWarning = true;
    XERCES_STD_QUALIFIER cerr << "\nWarning at (file " << StrX(e.getSystemId())
    << ", line " << e.getLineNumber()
    << ", char " << e.getColumnNumber()
    << "): " << StrX(e.getMessage()) << XERCES_STD_QUALIFIER endl;
}

void PersistParseHandlers::resetDocument()
{
    fSawWarning = false;
    fSawErrors = false;
}

```

PersistParseHandlers.hpp

```

/*
 * Copyright 1999-2006 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

// -----
// Includes
// -----
#include <xercesc/sax/HandlerBase.hpp>

XERCES_CPP_NAMESPACE_USE

class PersistParseHandlers : public HandlerBase
{
public:
// -----
// Constructors and Destructor
// -----
PersistParseHandlers();
~PersistParseHandlers();

// -----
// Handlers for the SAX ErrorHandler interface
// -----
void warning(const SAXParseException& exception);
void error(const SAXParseException& exception);
void fatalError(const SAXParseException& exception);

bool getSawErrors() const
{
    return fSawErrors;
}

```

```
}  
  
bool getSawWarning() const  
{  
    return fSawWarning;  
}  
  
void resetDocument();  
  
private:  
bool fSawErrors;  
bool fSawWarning;  
};
```

Appendix D. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Vol I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at:

<http://www.ibm.com/servers/eserver/zseries/zos/bkserv/>

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

Adobe, Acrobat, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

IBM, the IBM logo, ibm.com and DB2 are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

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

A

- accessibility 97
- accessing data sets
 - how to 13
- accessing XML data
 - how to 13
- Apache project, Xerces 7
- Apache Software Foundation 7
- ASCII, encoding 16
- avoiding conversion
 - DRDA 17
 - FTP 17
 - MQSeries 17

B

- B2B 1
- business-to-business 1

C

- characteristics of
 - DOM API 5
 - SAX API 5
- conversion, avoiding
 - DRDA 17
 - MQSeries 17

D

- deprecated DOM support 11
- disability 97
- Document Object Model 1
- Document Type Definition 1
- DOM 1
- DTD 1, 14
- DTD, accessing 14

E

- EBCDIC, encoding 16
- encoding, general 15
- encoding, XML 15
- event-based interface 3

F

- FTP
 - DRDA 17
 - MQSeries 17

H

- HTML 1

I

- iconv() 16
- interface, event-based 3

K

- keyboard 97

M

- MVS environment, Toolkit support 12

N

- namespaces 1
- native MVS
 - building samples using JCL
 - XML Parser, C++ Edition 69
 - XSLT Processor, C++ Edition 71

P

- packaging strategy, Toolkit 11
- parser, XML4C 7
- parsing documents
 - using DOM 5
 - using SAX 5
- processor, XSLT C++ 7

S

- SAX 3
- Schema, accessing 14
- Schema, XML 1
- schematic of the DOM parsing model 2
- schematic of the SAX API 4
- shortcut keys 97
- Simple API for XML 3
- specifying data sets using absolute URIs 14
- specifying data sets using relative URIs 13

T

- Toolkit 7
- Toolkit packaging strategy 11
- Toolkit parser, C++
 - multi-threading considerations 31
 - sample applications 21
 - using MVS multi-tasking 32
 - using UNIX pthreads 32
 - z/OS 21
 - building sample applications 28, 57
 - running sample applications 30, 60
 - z/OS UNIX 21
 - building sample applications 24, 53
- Toolkit parser, interfaces and specifications chart 7

- Toolkit processor, interfaces and specifications chart 7
- Toolkit support
 - MVS 12
 - z/OS UNIX System Services 12

U

- Unicode, encoding 16
- using the DOM API 2

V

- validating XML documents
 - results 6
- validation results 6

W

- W3C 1
- World Wide Web Consortium 1
- writing applications using the SAX specification 3

X

- Xerces Apache project 7
- XML 1, 7
- XML data, accessing 13
- XML documents, validation 6
- XML encoding 15
- XML Parser, C++ Edition 7
 - native MVS
 - building samples using JCL 69
- XML Path Language 7
- XML Schema 1
- XML Toolkit for z/OS 7
- XML4C parser 7
- XPath 6, 7
- XPLINK application, building 19
- XPLINK application, running 20
- XPLINK support 19
- XPLINK support, using 19
- XSL Transformations (XSLT) Version 1.0 7
- XSLT Processor, C++ Edition 7
 - native MVS
 - building samples using JCL 71
- XSLT ProcessorS, C++ Edition 7

Z

- z/OS 7
- z/OS parser classes
 - how to use 33, 51
 - sample applications 49, 51
 - using 49
 - z/OS 51
- z/OS specific parser classes 8
- z/OS UNIX System Services, Toolkit support 12
- z/OS XML 8
- z./OS XML System Services 8

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

**XML Toolkit for z/OS
User's Guide**

Publication No. SA22-7932-06

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product 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.

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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send your comments via e-mail to: mhvrfs@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



Fold and Tape

Please do not staple

Fold and Tape



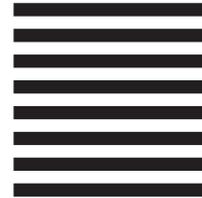
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

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



Program Number: 5655-J51

Printed in USA

SA22-7932-06

