z/OS

XML System Services User's Guide and Reference

SA23-1350-03

z/OS

XML System Services User's Guide and Reference

SA23-1350-03

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 257.

Fourth Edition, September 2008

This edition applies to Version 1 Release 10 of z/OS (5694-A01) and to subsequent releases and modifications until otherwise indicated in new editions.

This is a major revision of SA23-1350-02.

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
MHVRCFS, Mail Station P181
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/systems/z/os/zos/webqs.html

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

Make sure to include the following in your comment or note:
- Title and order number of this document
- Page number or topic related to your comment

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

Contents

Tables

About this document

This document presents the information you need to use the z/OS XML System Services (z/OS XML) parser.

Who should use this document

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

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

Where to find more information

Where necessary, this document references information in other documents about the elements and features of z/OS™. For complete titles and order numbers for all z/OS documents, see *z/OS Information Roadmap*.

Direct your request for copies of any IBM publication to your IBM representative or to the IBM branch office serving your locality.

There is also a toll-free customer support number (1-800-879-2755) available Monday through Friday from 6:30 a.m. through 5:00 p.m. Mountain Time. You can use this number to:

- Order or inquire about IBM publications
- Resolve any software manufacturing or delivery concerns
- Activate the program reorder form to provide faster and more convenient ordering of software updates

Softcopy publications

The z/OS library is available on the *z/OS Collection Kit*, SK2T-6700. This softcopy collection contains a set of z/OS and related unlicensed product documents. The CD-ROM collection includes the IBM® Library Reader™, a program that enables customers to read the softcopy documents.

Softcopy z/OS publications are available for web-browsing and PDF versions of the z/OS publications for viewing or printing using Adobe Acrobat Reader. Visit the z/OS library at http://www.ibm.com/systems/z/os/zos/bkserv/.

IBM Systems Center publications

IBM Systems Centers produce Redbooks that can be helpful in setting up and using z/OS. You can order these publications through normal channels, or you can view them with a Web browser. See the IBM Redbooks site at http://www.ibm.com/redbooks.

These documents have not been subjected to any formal review nor have they been checked for technical accuracy, but they represent current product understanding (at the time of their publication) and provide valuable information on a wide range of z/OS topics. You must order them separately.

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from these locations to find IBM message explanations for z/OS elements and features, z/VM®, z/VSE™, and Clusters for AIX® and Linux™:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS systems to access IBM message explanations using LookAt from a TSO/E command line (for example: TSO/E prompt, ISPF, or z/OS UNIX® System Services).
- Your Microsoft® Windows® workstation. You can install LookAt directly from the *z/OS Collection* (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T-4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS > command line) version can still be used from the directory in which you install the Windows version of LookAt.
- Your wireless handheld device. You can use the LookAt Mobile Edition from www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookatm.html with a handheld device that has wireless access and an Internet browser (for example: Internet Explorer for Pocket PCs, Blazer or Eudora for Palm OS, or Opera for Linux handheld devices).

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from:

- A CD in the *z/OS Collection* (SK3T-4269).
- The *z/OS and Software Products DVD Collection* (SK3T-4271).
- The LookAt Web site (click **Download** and then select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

Using IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS component that installations can use to gather information about their system environment and system parameters to help identify potential configuration problems before they impact availability or cause outages. Individual products, z/OS components, or ISV software can provide checks that take advantage of the IBM Health Checker for z/OS framework. This book might refer to checks or messages associated with this component.

For additional information about checks and about IBM Health Checker for z/OS, see *IBM Health Checker for z/OS: User's Guide*.

SDSF also provides functions to simplify the management of checks. See *z/OS SDSF Operation and Customization* for additional information.

Summary of changes

Summary of Changes
for SA23-1350-03
z/OS Version 1 Release 10

This document contains information previously presented in *z/OS XML System Services User's Guide and Reference*, SA23-1350-01, which supports z/OS Release 9.

New Information

- New information in Chapter 4, "Parsing XML documents," on page 11
- "Aux info record" on page 19
- New information in Chapter 6, "z/OS XML parser API: C/C++," on page 35
- New information in Chapter 7, "z/OS XML parser API: Assembler," on page 79
- "XMLDATA IPCS subcommand" on page 111
- New reason codes in Appendix B, "Reason Codes Listed by Value," on page 119
- Appendix E, "xsdosrg command reference," on page 167
- New information in Appendix F, "C/C++ header files and assembler macros," on page 171
- Appendix G, "Java file," on page 177

Summary of Changes
for SA23-1350-02
z/OS Version 1 Release 9

This document is a refresh of *z/OS XML System Services User's Guide and Reference*, SA23-1350-01, which supports z/OS Release 9.

New Information

- Feature flag option added to "gxlpControl — perform a parser control function" on page 37 and "GXL1CTL (GXL4CTL) — perform a parser control function" on page 82
- Encodings added to on page 0, "gxlhxec.h (GXLYXEC) - constants definitions" on page 172, and on page 0
- Appendix L, "Supported encodings," on page 253

Summary of Changes
for SA23-1350-01
z/OS Version 1 Release 9

This document contains information previously presented in *z/OS XML System Services User's Guide and Reference*, SA23-1350-00, which supports z/OS Release 8.

New Information

- Feature flag option added to "GXL1CTL (GXL4CTL) — perform a parser control function" on page 82
- Chapter 6, "z/OS XML parser API: C/C++," on page 35
- "ARR recovery routine" on page 114
- Appendix F, "C/C++ header files and assembler macros," on page 171

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.

Chapter 1. Introduction

What is 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, by either a DTD or a document written in the XML Schema language, that can be used for validation. 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 these XML applications evolve and become more powerful, they 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, is still a developing technology.

An XML parser is a processor that reads an XML document and determines the structure and properties of the data. It breaks the data up into discrete units and provides them to other components. There are two basic types of XML parsers: non-validating and validating. A non-validating parser checks if a document is well-formed, but does not check a document against any DTDs or XML Schemas. A validating parser not only checks if a document is well-formed, but also verifies that it conforms to a specific DTD or XML Schema.

z/OS XML System Services

z/OS XML System Services (z/OS XML) is an XML processing component of the z/OS operating system. It contains an XML parser intended for use by system components, middleware, and applications that need a simple, efficient, XML parsing solution. z/OS XML can parse documents either with or without validation.

Note: The use of the term z/OS XML parser in this document refers specifically to the z/OS XML System Services parser.

The following are some distinct characteristics of z/OS XML:

- z/OS XML is an integrated component of z/OS. There is no need to download or install any additional packages to use it.
- z/OS XML provides a collection of programming interfaces for callers to use:
  - C/C++ and assembler interfaces to the z/OS XML parser itself.
  - C/C++, Java, and UNIX command line interfaces to utility functions that build binary artifacts required for validation during a parse.
  - Assembler interfaces for user exits that give callers control over how the z/OS XML parser interacts with the rest of z/OS.
  - C/C++ interfaces to a service similar to a user exit, called a StringID handler, that allows for shorthand communications between the z/OS XML parser and the caller.
- The z/OS XML parser utilizes a buffer-in, buffer-out processing model instead of the event driven model common to SAX parsers. Input to, and output from the parser may span multiple buffers, allowing the caller to request parses for documents that are arbitrarily long.
- z/OS XML has minimal linkage overhead in order to reduce CPU usage as much as possible.
- z/OS XML provides assistive aids to the user in debugging not-well-formed documents.
- z/OS XML supports a number of character encodings, among them UTF-8, UTF-16 (big endian), IBM-1046 and IBM-037. There is no need on the part of the caller to transcode documents to a canonical encoding before calling the z/OS XML parser. For a full list of these supported encodings, see Appendix L, “Supported encodings,” on page 253.

The z/OS XML parser is invoked as a callable service and can be used as such. The callable services stubs are shipped in CSSLIB.

Note about constant names: Some constant names begin with the string ″GXLH″. These constants are used solely by C callers. For assembler callers, remove the ″GXLH″ portion to get the appropriate constant name.

Chapter 2. Overview of z/OS XML System Services

This chapter provides an overview of the z/OS XML System Services; it briefly describes some of the XML features supported by the z/OS XML System Services and other technologies used by the z/OS XML parser. The following topics are discussed within this chapter:

- "z/OS XML System Services features" on page 3
- "z/OS XML System Services functions" on page 4
- "Document processing model" on page 5
- "Output buffer format" on page 7
- "Optimized Schema Representation" on page 7
- "String Identifiers" on page 7
- "Memory management" on page 7
- "Dynamic LPA support" on page 7

z/OS XML System Services features

The following is a list of features provided by z/OS XML System Services. References to additional information on the various features are provided where appropriate:

- An external C and C++ API, see Chapter 6, "z/OS XML parser API: C/C++," on page 35
- An external assembler API, see Chapter 7, "z/OS XML parser API: Assembler," on page 79
- Support for AMODE 31- and 64-bit callers with data above or below the bar.
- Support for UTF-8, UTF-16 (big endian only), IBM-1047, IBM-037, and several other encodings. See "Encoding support" on page 31 for more information.
- XML processing features
  - Parsing with schema validation (validation with DTD not supported)
  - Support in the parsed data stream for offsets back into the original source document
  - Optionally return fully qualified element names in end element records
  - Support for XML 1.0 and XML 1.1
  - Newline normalization, see "EBCDIC encoding considerations" on page 32
  - Attribute value normalization
  - Omit or return comments in the parsed data stream
  - Optionally return significant white space in unique white space records (instead of character data records)
  - Namespace support, see "Namespace declarations" on page 30
  - Entity resolution, see "Resolving entity references" on page 29
  - Partial DTD processing, see "Processing DTDs" on page 29
- User exits for system services, see Chapter 8, "z/OS XML System Services exit interface," on page 101
- Query service for determining document characteristics, see Chapter 3, "Querying XML documents," on page 9
- Diagnostic support (Chapter 9, "Diagnosis and problem determination," on page 111), including:
  - Diagnostic area, see "Diagnostic Area" on page 113

  - Slip trap support, see “SLIP trap for return codes from the z/OS XML parser” on page 114
  - ARR recovery routine, see “ARR recovery routine” on page 114
  - IPCS formatting, see “XMLDATA IPCS subcommand” on page 111
- Segmented input and output (the entire document does not have to reside in a single buffer), see “Spanning buffers” on page 28
- Mapping macro interfaces for parsed data
- “Enable offload to specialty engines” on page 8

z/OS XML System Services functions

z/OS XML System Services include the following three primary functions:

- A query service that allows callers to determine the encoding of the document and acquire information from the XML declaration.
- Parsing with schema validation
- Parsing without validation

These functions are provided in the form of callable services. A caller can access these services through the z/OS XML System Services APIs (for information on the APIs, see Chapter 6, “z/OS XML parser API: C/C++,” on page 35 and Chapter 7, “z/OS XML parser API: Assembler,” on page 79). The following two sections provide a summary of the functions, with pointers on where to go for more information.

Querying XML documents

XML documents have characteristics that affect the way they are parsed, and the kinds of information that the parser generates during the parse process. One such characteristic is the encoding scheme of the document, which the z/OS XML parser must know before parsing. Using the query service will allow the caller to acquire this information, after which it can then pass it to the z/OS XML parser. The z/OS XML parser will then be able to use the correct encoding scheme to parse the document. For more on this service, see Chapter 3, “Querying XML documents,” on page 9.

Parsing XML documents without validation

The non-validating parse process consists of three fundamental steps: initialize the parser, parse the document, and terminate the parser. Multiple documents may be parsed using either a single instance of the parser, or several distinct instances as the caller requires. For more information on this procedure and the individual services called, see Chapter 4, “Parsing XML documents,” on page 11.

Parsing XML documents with validation

Parsing with validation follows the same basic process as for a non-validating parse. The key difference is an additional step to load a pre-processed version of the schema used to validate the document during the parse. This binary schema, referred to as an Optimized Schema Representation (OSR) can be loaded once, and used to validate any document that conforms to it. For more information on OSRs, see “Optimized Schema Representation” on page 7.

Document processing model

There are three main components required for parsing XML documents: input buffer, z/OS XML parser, and output buffer. These three components and their interrelationships make up the processing model. There may be more than one input and output buffer, depending on the size of the document being parsed. If the document is sufficiently large, the caller may find it necessary to provide it to the parser in several pieces, using buffer spanning to maintain the document structure as it is being parsed. Similarly, the caller may need to provide multiple buffers to contain the data stream that the z/OS XML parser generates. For more information on how buffer spanning works, see “Spanning buffers” on page 28.

Document processing is the creation of the output buffers from the parsed input data. As the z/OS XML parser traverses through the input buffer, the output buffer is built. See “Parsed data model” on page 15 for more information on this format.

The following is a diagram of the processing model using buffer spanning. It shows both the input and output buffers, where buffers 2-5 represent the additional buffers created to support a large document.

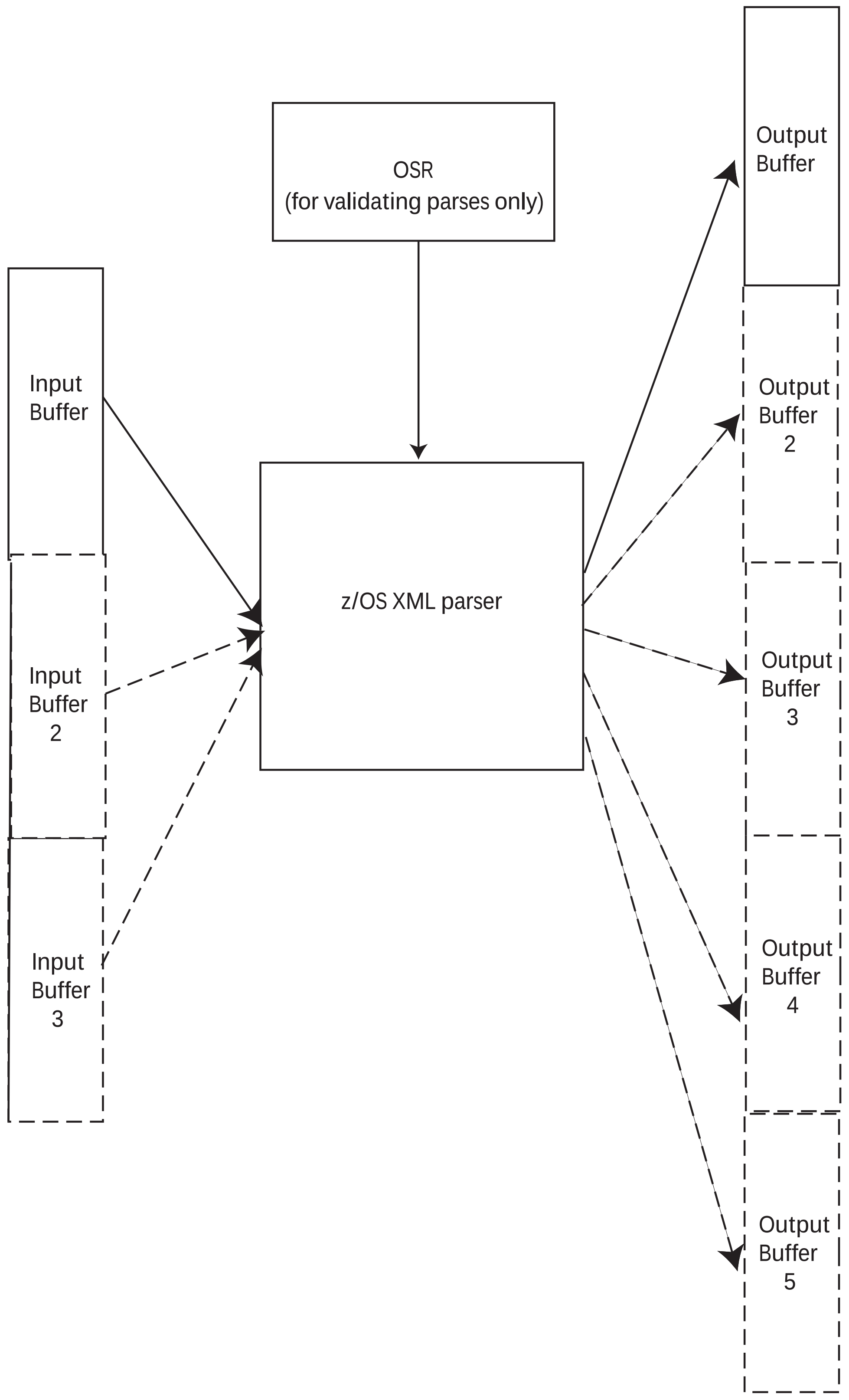

Figure 1. Processing model

For more on how to parse XML documents using the z/OS XML parser, see Chapter 4, "Parsing XML documents," on page 11.

Output buffer format

The output buffer contains the parsed data stream that results from the parse process. This data stream will contain the parsed XML document contents, along with header and any error information that was produced during the parse. For more information on the format of the output data stream, see “Parsed data model” on page 15.

Optimized Schema Representation

Optimized Schema Representations (OSRs) are pre-processed versions of schemas. They are more easily and more efficiently handled than schemas in text form. When parsing with validation, this form of schema is utilized. An OSR API is provided to assist in the generation, loading and manipulation of these specialized schemas. For information on how to use OSRs, see “Using Optimized Schema Representations” on page 13. For more information on the OSR API, see “OSR generator API” on page 52. For information on performing a validating parse, see Chapter 4, “Parsing XML documents,” on page 11.

String Identifiers

String Identifiers (StringIDs) are a special type of output data used to represent some of the strings that the z/OS XML parser encounters during a parse. A StringID is a 4 byte numeric value used to represent a complete string of text, thereby substantially reducing the size of the parsed data stream for documents containing frequently recurring strings, like namespace references. StringIDs can only be used if the optional StringID exit service is activated. For more information on StringIDs, see “String Identifiers” on page 23.

Memory management

The z/OS XML parser provides a memory allocation/deallocation exit allowing callers to provide a pair of allocation/deallocation services. For callers that do not provide a memory allocation exit, the z/OS XML parser provides an option at initialization time allowing the caller to specify how the z/OS XML parser’s default routine allocates memory. For more information on these services and the special initialization time feature, see “Managing memory resources” on page 32.

Dynamic LPA support

The non-validating z/OS XML parser resides in the link pack area (LPA), and as such, the non-validating parser can take full advantage of dynamic LPA. The validating z/OS XML parser is initially located in SIEALNKE, but can be moved to LPA at post-installation. Both z/OS XML parsers are always accessible, and any service item fixes can easily be applied to either of the parsers since an IPL is not required. For more information on dynamic LPA support and servicing of the non-validating and validating z/OS XML parsers, see Chapter 10, “System Admin: Servicing the z/OS XML parser,” on page 115. For additional information on dynamic LPA, see z/OS MVS Initialization and Tuning Reference.

Enable offload to specialty engines

z/OS XML System Services provides the ability for parsing operations to be run on specialty processors: a zAAP (System z Application Assist Processor) or a zIIP (IBM System z10 Integrated Information Processor). The z/OS XML parser, when executing in TCB mode, is eligible to run on a zAAP, in environments in which one or more zAAPs are configured. The z/OS XML parser, when executing in enclave SRB mode, is eligible to run on a zIIP processor, in environments where one or more zIIPs are configured. Ancillary z/OS XML System Services, such as the query service and the control service, as well as the StringID exit and memory management exits, are not eligible to run on specialty processors. Execution of z/OS XML System Services parsing operations on a specialty processor occurs transparently to the calling application.

Chapter 3. Querying XML documents

An XML document contains declarations that may need special handling during a parse. For instance, if the encoding of the document to parse is unknown, the query service provided by z/OS XML parser can be used to help determine the encoding in order to provide the correct Coded Character Set IDentifier (CCSID) to the parser, when the actual parse is performed.

In order for the caller to query an XML document, all the caller needs to do is use the query service (gxlpQuery for C/C++ callers, GXL1QXD (GXL4QXD) for assembler callers). This service allows the caller to obtain all the XML characteristics of the document. These characteristics can be either the default values or those explicitly contained in an XML declaration. Once these characteristics are obtained, the caller can then determine the encoding scheme needed to parse the document, along with any additional steps that may be needed.

For example, if the document in question uses an encoding scheme of UTF-16, it will require that the z/OS XML parser also uses the UTF-16 encoding scheme when parsing this document. The caller would use the query service to ascertain the encoding type of the document being parsed. Once this information is acquired, the z/OS XML parser can be initialized using the initialization service (gxlpInit for C/C++ callers, GXL1INI (GXL4INI) for assember callers, see Chapter 4, “Parsing XML documents,” on page 11) passing the encoding scheme to parse the UTF-16 encoded document.

Note: The query service is the only service that provides support for both UTF-16 (little endian) and UTF-16 (big endian), whereas the other services only support UTF-16 (big endian).

The CCSID value returned by the query service can be used to invoke Unicode Services in order to convert the input document into one of the encodings supported by the z/OS XML parser.

For more information on the query service, see “gxlpQuery — query an XML document” on page 49 for C/C++ callers, and “GXL1QXD (GXL4QXD) — query an XML document” on page 93 for assembler callers. For more information on document encoding support, see “Encoding support” on page 31.

Header files and data macros

This section provides information on the various header files and data macros associated with the z/OS XML parser query service. The names and purposes of these files are listed below:

| **Note:** For header file names, replace the ″*″ with the letter ″H″, convert all letters
| of the name to lowercase, and append ″.h″ to the end of the filename. For
| data macro names, replace the ″*″ with the letter ″Y″.

GXL*XEC
Contains assorted constant values that are used in the parsed data stream, values used for assorted fields of the API, and minimum sizes for data areas passed to the z/OS XML parser.

GXL*QXD
Maps the data area returned from the query service.

GXL*XR
: Contains mnemonic values that describe the return and reason codes generated by the z/OS XML parser.

For information on these header files and data macros, see Appendix F, “C/C++ header files and assembler macros,” on page 171.

Chapter 4. Parsing XML documents

Before the z/OS XML parser can perform a parse on an XML document, it must first establish a context in which it can operate. This is accomplished when the caller invokes the initialization routine and passes in a piece of memory where the z/OS XML parser establishes a Parse Instance Memory Area (PIMA). This is the area where the z/OS XML parser creates a base for the internal data structures it uses to complete the parse process.

Rule: A particular PIMA must only be used during the parse of a single XML document at a time. Only after the parse is complete and the parse instance is reset can a PIMA be reused for the parse of another document.

In addition to control information, the PIMA is used as a memory area to store temporary data required during the parse. When the z/OS XML parser needs more storage than was provided in the PIMA, additional storage is allocated. Because allocating additional storage is an expensive operation, the PIMA should be initially allocated with sufficient storage to handle the expected document size, in order to optimize memory allocation requests.

Rule: For the non-validating z/OS XML parser, the minimum size for the PIMA is 128 kilobytes. For the validating z/OS XML parser, the minimum size for the PIMA is 768 kilobytes.

Everything that the z/OS XML parser needs to complete the parse of a document is kept in the PIMA, along with any associated memory extensions that the parser may allocate during the parse process. The caller also must provide input and output buffers on each call to the parse service (gxlpParse for C/C++ callers, GXL1PRS (GXL4PRS) for assember callers). In the event that either the text in the input buffer is consumed or the parsed data stream fills the output buffer, the z/OS XML parser will return XRC_WARNING, along with a reason code indicating which buffer (possibly both) needs the caller's attention. It also indicates the current location and number of bytes remaining in each buffer by updating the *buffer_addr* and *buffer_bytes_left* parameters passed in on the parse request (for C/C++ callers, see the description of "gxlpParse — parse a buffer of XML text" on page 46; for assembler callers, see the description of "GXL1PRS (GXL4PRS) — parse a buffer of XML text" on page 90). This process is referred to as buffer spanning. For more information, see "Spanning buffers" on page 28.

If the entire document has been processed when the z/OS XML parser returns to the caller, the parse is complete and the caller proceeds accordingly. If the caller requires another document to be parsed, it has the option of terminating the current parse instance by calling the termination service (gxlpTerminate for C/C++ callers, GXL1TRM(GXL4TRM) for assembler callers). This will free up any resources that the z/OS XML parser may have acquired and resets the data structures in the PIMA. If the caller needs to parse another document, it will have to call the initialization service again to either completely re-initialize an existing PIMA that has been terminated or initialize a new PIMA from scratch.

Another option is to use the finish/reset function of the z/OS XML parser control service (gxlpControl for C/C++ callers, GXL1CTL (GXL4CTL) for assembler callers) to reset the PIMA so that it can be reused. This is a lighter-weight operation that preserves certain information that can be reused across parsing operations for multiple documents. This potentially improves the performance for subsequent parses, since this information can be reused instead of rebuilt from scratch.

Reusing the PIMA in this way is particularly beneficial to callers that need to handle multiple documents that use the same symbols (for example, namespaces and local names for elements and attributes). The PIMA can only be reused in this way when the XML documents are in the same encoding.

Restriction: The following restrictions apply when conducting a validating parse:

- When parsing in non-Unicode encodings, irrepresentable character entities are replaced with the "-" character prior to validation.
- There is a maximum of 64 KB non-wildcard attributes for a single element, and 64 KB elements in an `All` group.

Steps for parsing XML documents without validation

The following steps summarize the process of parsing XML documents using the z/OS XML parser:

1. Call the initialization service. This establishes the PIMA, which is then used to create and store the initial data structures required to begin the parse process.
2. Call the parse service to parse the document.

   Note: During the parse process and before the end of the document is reached, if the input buffer is empty or the output buffer is full, a warning is issued and the parse service is stopped. Otherwise, the parse service will continue until the document is fully processed.
3. The application processes the output buffer.
4. Determine if there are additional documents to be processed. If so, call the termination service to terminate the existing parse process, and repeat Steps 1-3.

   Tip: For increased performance, the caller can use the control service in place of the termination and initialization services. The control service enables the PIMA to be reused, avoiding the need to free resources and initialize a new PIMA. However, the PIMA can only be reused in this way when the XML documents are in the same encoding. See "gxlpControl — perform a parser control function" on page 37 and "GXL1CTL (GXL4CTL) — perform a parser control function" on page 82 for more information on the control service.

There are several data types that can be returned in the output buffer. Therefore, the caller must know what type of data is being returned to effectively process it. The following topics discuss the various data types:

- "Parsed data model" on page 15 - overview of the structures that make up the data stream produced by the parser.
- "Length/Value pairs" on page 23 - the default representation of strings that have been parsed from the original XML document.
- "String Identifiers" on page 23 - a unique numeric value returned by the z/OS XML parser that represents a given text string (a StringID exit service must be provided by the caller to generate these IDs).
- "Metadata records" on page 18 - data records that contain metadata about the parse stream or error information

Steps for parsing XML documents with validation

The following steps summarize the process of parsing XML documents using the z/OS XML parser with validation:

1. Call the parser load service. This will load the parser into storage.
2. Call the OSR initialization utility. This establishes the OSR generator Instance Memory Area (OIMA), which is then used as the work area for the OSR generator.
3. Call the OSR generator utility. This utility creates an OSR from one or more text-based schemas passed to the OSR generator instance, using the load schema utility.

   Note: An OSR can be saved and then used for parsing future documents that share the same schema(s) from which the OSR was generated. As a result, steps 2 and 3 may not be required each time an XML document is parsed using validation.
4. Call the parser initialization service. This establishes the PIMA, which is then used to create and store the initial data structures required to begin the parse process.
5. Call the control service. This will load the generated OSR into the z/OS XML parser.
6. Call the parse service to parse the document.

   Note: During the parse process and before the end of the document is reached, if the input buffer is empty or the output buffer is full, a warning is issued and the parse service is stopped. Otherwise, the parse service will continue until the document is fully processed.
7. The application processes the output buffer.
8. Determine if additional schemas need to be processed. If so, repeat steps 3, 5 and 6. If you want to reuse an existing OSR, use the OSR load utility.
9. Determine if there are additional documents to be processed. If so, call the termination service to terminate the existing parse process, and repeat steps 1 -7.

   Tip: For increased performance, the caller can use the control service in place of the termination and initialization services. The control service enables the PIMA to be reused, avoiding the need to free resources and re-initiate a new PIMA. However, the PIMA can only be reused in this way when the XML documents are in the same encoding. See “gxlpControl — perform a parser control function” on page 37 and “GXL1CTL (GXL4CTL) — perform a parser control function” on page 82 for more information on the control service.

Using Optimized Schema Representations

Optimized Schema Representations (OSRs) are specialized forms of schemas used during the validating parse process. They can be created from utilities provided by the OSR generator API. For more information about the OSR generator API, see “OSR generator API” on page 52.

Setting up the environment

Before the caller can begin generating OSRs, some environment variables must be set. The following lists the environment variables that must be set along with their appropriate values.

Note: The caller should use the proper 31/64-bit versions of the binaries listed below. Mixing versions of different binaries will result in unpredictable results.

LIBPATH
must include paths to the following:
- For C API callers only (gxlcosr1.dll for 31-bit, gxlcosr4.dll for 64-bit) - /usr/lib
- For 31-bit callers - /usr/lib/java_runtime
- For 64-bit callers - /usr/lib/java_runtime64
- Java binaries and JVM
  - For 31-bit callers -
    - /usr/lpp/java/J5.0/bin
    - /usr/lpp/java/J5.0/bin/j9vm
  - For 64-bit callers -
    - /usr/lpp/java/J5.0_64/bin
    - /usr/lpp/java/J5.0_64/bin/j9vm

CLASSPATH
must include paths to the following:
- The Java API callers only (gxljapi.jar) - /usr/include/java_classes

  Note: Do not include gxljosrgimpl.jar. It will be loaded from /usr/include/java_classes

  Note: Callers of the Java API must choose the 31- or 64-bit version of Java that they intend to use. They may either specify the explicit path to the required executable (/usr/lpp/java/J5.0/bin/java for 31-bit, /usr/lpp/java/J5.0_64/bin/java for 64-bit), or include the path to the required Java version in their PATH variable. Users of the C API and command interfaces do not need to be concerned with this.

Usage tips

Tips are provided below to facilitate the usage of OSRs:
- An OSR is not a schema library. In other words, you should not throw all necessary schemas into a single OSR and use it similar to a library.
- Schemas should reference one another via the `<xsl:import ...>` construct. That is, OSRs are meant to contain hierarchies of schemas, where one or more schemas reference others to handle increasingly more specific structures in the source XML document being transformed.
- You should consider creating one schema OSR to validate entire classes of documents.

The OSR used for validation becomes part of the parse instance, and remains in use for all validating parse requests until a different one is specified through the control service. Callers who use buffer spanning to pass documents to and from the parser in pieces should know that schemas cannot be changed in the middle of the parse process. A control request to specify a different schema will cause a reset of the parse instance so that the next parse request must be for a new XML document.

Note: For callers using schemas written in XML 1.1 format, use IBM Java Technology Edition V6.

Header files and data macros

This section provides information on the header files and data macros associated with the z/OS XML parser. The names and purposes of these files are listed below:

Note: For header file names, replace the ″*″ with the letter ″H″, convert all letters of the name to lowercase, and append ″.h″ to the end of the filename. For data macro names, replace the ″*″ with the letter ″Y″.

GXL*XEH
Describes all of the structures that the z/OS XML parser generates in the parsed data stream. This includes both the records that represent the individual markup and content parsed from the document, as well as metadata about the data stream itself.

GXL*XEC
Contains assorted constant values that are used in the parsed data stream, values used for assorted fields of the API, and minimum sizes for data areas passed to the z/OS XML parser.

GXL*XD
Maps the z/OS XML parser extended diagnostic area.

GXL*XR
Contains mnemonic values that describe the return and reason codes generated by the z/OS XML parser.

GXL*XSV
Maps the system service vector that the caller uses to describe the exits that it provides to the z/OS XML parser.

GXL*XFT
Maps the input and output area used by the control feature flag of the GXL1CTL (GXL4CTL) service.

GXL*XOSR
Maps the input and output area used by the optimized schema representation.

For information on these header files and corresponding data macros, see Appendix F, “C/C++ header files and assembler macros,” on page 171.

Parsed data model

This section provides information on the data model used to represent the contents of the output buffer. The caller needs to understand this data model so that it can effectively process the parsed data stream that has been created in the output buffer.

The z/OS XML parser produces a structured data stream resulting from the parse process. It is a feature that distinguishes the z/OS XML parser from most other XML parsers. The parsed data stream consists of a set of self-describing records representing the output of the parser. These records provide a structure to the data stream that allows a consumer to navigate the data stream as needed. Some of the records represent the actual semantic content of the parsed document, while others provide metadata about the parse itself. There may be more than one group of these records (or record groups) in a single output buffer. This can occur if the input buffer spans multiple times before the output buffer is filled.

Common record header

Each record in the parsed data stream consists of a common header, followed by information that is specific to a given record type. The common header has the following structure:

Table 1. Common record header

+0	record type (2 bytes)	flags (1 byte)	reserved (1 byte)
+4	record length		

The record type determines the form of the data that immediately follows the header and which makes up the body of the record. The record flags provide information about the specific record to which they belong. Each bit of the flags byte has the following meaning:

Table 2. Record flag bits

Bit position	**Name**	**Purpose**
0	XEH_Continued	This record is continued in the next output buffer.
1	XEH_No_Escapes	There are no characters that need to be escaped in this record.
2	*	reserved

The XEH_No_Escapes flag is provided as an aid to callers that need to re-serialize the parsed data stream back to an XML document in text form. It is relevant only for records that represent character data or attribute values (its meaning is undefined for all other records). It indicates that there are no special characters present that need to be escaped in the text of the record during re-serialization. The set of these special characters is made up of ″<″, ″>″, ″&″, and the single and double quotes. The caller must substitute either one of the well known strings (″&lt;″, ″&gt;″, ″amp;″, ″&apos;″, ″&quot;″) or a numeric character reference in the serialized text in order to create a well formed XML document.

When this flag is on, the caller can safely avoid scanning the text associated with the record to look for characters that must be escaped during re-serialization. When the flag is off, one of the special characters may be present, and such a scan is required. Note that there are certain instances involving buffer spanning when it is not possible for the parser to determine that this bit should be set. As a result, for character data and attribute value records that span multiple output buffers, the XEH_No_Escapes bit may be off, even when there are actually no characters that need to be replaced during serialization. If the bit is on though, it will always be safe to avoid scanning for characters that need escaping.

The flags field is followed by 1 reserved byte and the record length. The record length contains the total length of the record - including the header. Navigating from one record to the next is done by moving a pointer, by the specified record length, from the first byte of the current record header.

Record (token) types

Record types are values used to identify the purpose of each record parsed from the input document. The record type, along with the data stream options in the buffer info record (see "Buffer info record" on page 18), indicates the form of the record. Record forms are a means of indicating the number of values that make up the record itself, and are described in a separate section below. Here are the record types returned by the z/OS XML parser (their definitions are provided in gxlhxec.h for C and C++ callers, and GXLYXEC for assembler callers):

Table 3. Record types

Token name	Meaning
GXLHXEC_TOK_BUFFER_INFO	information about the buffer containing the parsed data stream
GXLHXEC_TOK_ERROR	error information
GXLHXEC_TOK_XML_DECL	an XML declaration
GXLHXEC_TOK_START_ELEM	start of an element
GXLHXEC_TOK_END_ELEM	end of an element
GXLHXEC_TOK_ATTR_NAME	name of an attribute
GXLHXEC_TOK_ATTR_VALUE	value of an attribute
GXLHXEC_TOK_NS_DECL	a namespace declaration
GXLHXEC_TOK_CHAR_DATA	character data
GXLHXEC_TOK_START_CDATA	start of a CDATA section
GXLHXEC_TOK_END_CDATA	end of a CDATA section
GXLHXEC_TOK_WHITESPACE	a string of white space characters
GXLHXEC_TOK_PI	processing instruction
GXLHXEC_TOK_COMMENT	a comment
GXLHXEC_TOK_DTD_DATA	DOCTYPE declaration information
GXLHXEC_TOK_UNRESOLVED_REF	an entity reference that cannot be resolved
GXLHXEC_TOK_AUX_INFO	auxiliary information about individual items in the parsed data stream

The above token names are for the C/C++ callers. Assembler callers use token names without the prefix ″GXLH″.

Most of the record types listed above fall into one of four classes, based on the number of values they contain from the document being parsed. Two of these record types - the buffer info and error records - are different (see "Buffer info record" on page 18 and "Error info record" on page 19) because they contain metadata about the information in one of the buffers (input or output), rather than data parsed from the input stream. The form of the data they contain is unique to the purpose of the record.

The data structures that describe this data stream can be found in the data model header file gxlhxeh.h for C/C++ callers, and the mapping macro GXLYXEH for assembler callers. Data is not aligned on any kind of boundary, and there are no alignment requirements for the input or output buffers provided by the caller.

Metadata records

Some records contain metadata related to the parsing process. These records are discussed below.

Buffer info record

Because the data stream that the z/OS XML parser generates in the output buffer consists of one or more groups of records, each group always begins with the buffer info record - a record containing metadata about the parsed data stream contained in the current output buffer. This record includes the length of the buffer used by the record group and flags indicating the characteristics of the data stream.

The following is the structure for the buffer info record, including the record header:

Table 4. Buffer info record structure

+0	record type	flags	reserved
+4	record length		
+8	datastream options		
+C	parse status	reserved	
+10	buffer length used		
+14			
+18	offset to error record		
+20			

This record is not allowed to span output buffers, so the continuation flag in the record flags field of the buffer header will always be zero. The datastream options contain a flag indicating whether or not StringIDs are in use, plus some of the flags from the feature flags parameter on the z/OS XML parser init call. These flags indicate some characteristic of the data in the parsed data stream. The full list of flags indicate:

- StringIDs are in effect
- Comments are stripped (GXLHXEC_FEAT_STRIP_COMMENTS)
- White space is being tokenized (GXLHXEC_FEAT_TOKENIZE_WHITESPACE)
- Returning CDATA as CHARDATA (GXLHXEC_FEAT_CDATA_AS_CHARDATA)
- Validating parser is enabled (GXLHXEC_FEAT_VALIDATE)
- Source offsets are enabled (GXLHXEC_FEAT_SOURCE_OFFSETS)
- Full end tag feature is enabled (GXLHXEC_FEAT_FULL_END)

Notes:

1. The GXLHXEC_FEAT* flags in above parentheses are defined in gxlhxec.h for C/C++ callers and GXLYXEC for assembler callers. For assembler callers, remove the ″GXLH″ prefix from the constant names.
2. The buffer info record is mapped out in gxlhxeh.h for C/C++ callers and GXLYXEH for assembler callers.

The ″parse status″ field is another set of flags that indicate whether unresolved external references are present in this particular buffer.

The ″buffer length used″ field indicates the portion of the output buffer consumed by the group of records represented by this buffer info record. If no buffers are spanned during the parse process, there will be only one buffer info record present in the output buffer, representing a single group of records. If buffers are spanned, there may be several record groups, each with corresponding buffer info records present in the output buffer. The number of record groups and buffer info records depends on how the caller manages the buffers that are passed to the parser. See “Spanning buffers” on page 28 for more information.

The ″error record offset″ field indicates the offset from the beginning of the buffer info record to the beginning of the error info record. If this offset is zero, there is no error record present in the group of records represented by the buffer info record.

Error info record

The error info record is placed in the parsed data stream whenever a parsing error is detected. The offset to the error from the start of the document, along with the return and reason code generated by the z/OS XML parser when the error was encountered, are kept in a field of the error info record. Here is the structure of the record, including the record header:

Table 5. Error info record structure

+0	record type	flags	reserved
+4	record length		
+8	return code		
+C	reason code		
+10	offset of the error from the start of the document		
+14			

Note: The error info record is mapped out in gxlhxeh.h for C/C++ callers, and GXLYXEH for assembler callers.

For information on error codes and how to use them, see “Using return and reason codes” on page 33.

Aux info record

When the source document offsets feature (GXLHXEC_FEAT_SOURCE_OFFSETS) is selected, a new information record is inserted in the output buffer. This record has the following structure:

Table 6. Aux info record

+0	record type	flags	reserved
+4	record length		
+8	aux flags	information type	
+C	-varied information-		

The record values are defined as follows:

Record header
This is the standard record header of all records in the data model. The record type is XEC_TOK_AUX_INFO.

Aux flags
These flags provide information about the form of the data in the rest of the record:

- XEH_AUX_LONG_VALUE - This flag is only used in records which contain values which can vary in size. This bit will be OFF if the record contains integer values that are 4 bytes in length. The bit will be ON if the record contains values that are 8 bytes in length. Any value or record length fields in the record such as the record length in the header will always be 4 bytes no matter what the value of this bit is.

All offset values which are under 4GB-1 in magnitude will be represented as a 4 byte value in the data stream and the XEH_AUX_LONG_VALUE flag will be OFF. When an offset is encountered which exceeds 4 GB, then all offset records from that point on will be represented as 8 byte values in the data stream and the XEH_AUX_LONG_VALUE will be set ON.

- AUX_ENTITY - This is set for information records that are generated from entities.

Information type
This value identifies what information is contained in the record. See on page 20 for details on the different types.

-varied information-
The contents of the additional information will depend on the information type and flags. For offset records, this will contain either a 4 or an 8 byte value which represents the offset of the particular structure from the beginning of the document. It will be 4 bytes if the XEH_AUX_LONG_VALUE bit flag bit is OFF in the header. It will be 8 bytes if the XEH_AUX_LONG_VALUE bit flag is ON in the header.

Information types:

XEC_OFFSET_START_STARTTAG
This is the offset of the ‘<’ at the beginning of an XML start tag. This record occurs in the datastream immediately preceding the XEC_TOK_START_ELEM .

XEC_OFFSET_END_STARTTAG
This is the offset of the ‘>’ at the end of an XML start tag. This record occurs in the datastream immediately after the last XEC_OFFSET_END_ATTRVALUE, if there are attributes, or the XEC_OFFSET_END_STARTTAGNAME record if there are no attributes.

XEC_OFFSET_END_STARTTAGNAME
This is the offset to the end of the XML start name qname. This record occurs in the datastream immediately following the XEC_TOK_START_ELEM.

XEC_OFFSET_START_ATTRVALUE
This is the offset of the beginning quote of the attribute value. This record occurs in the datastream immediately preceding the XEC_TOK_ATTR_VALUE.

XEC_OFFSET_END_ATTRVALUE
This is the offset of the ending quote of the attribute value. This record occurs in the datastream immediately after the XEC_TOK_ATTR_VALUE.

XEC_OFFSET_START_COMMENT
This is the offset of the ‘<’ at the beginning of an XML comment. This record occurs in the datastream immediately preceding the XEC_TOK_COMMENT.

XEC_OFFSET_END_COMMENT
This is the offset of the ‘>’ at the end of an XML comment. This record occurs in the datastream immediately following the XEC_TOK_COMMENT.

XEC_OFFSET_START_CDATA
This is the offset of the '<' at the beginning of an XML CDATA. This record occurs in the datastream immediately preceding the XEC_TOK_START_CDATA.

XEC_OFFSET_END_CDATA
This is the offset of the '>' at the end of an XML CDATA. This record occurs in the datastream immediately following the XEC_TOK_END_CDATA.

XEC_OFFSET_START_PI
This is the offset of the '<' at the beginning of an XML PI. This record occurs in the datastream immediately preceding the XEC_TOK_PI.

XEC_OFFSET_END_PI
This is the offset of the '>' at the end of an XML PI. This record occurs in the datastream immediately following the XEC_TOK_PI.

XEC_OFFSET_START_XMLDECL
This is the offset of the '<' at the beginning of an XML Declaration. This record occurs in the datastream immediately preceding the XEC_TOK_XML_DECL.

XEC_OFFSET_END_XMLDECL
This is the offset of the '>' at the end of an XML Declaration. This record occurs in the datastream immediately following the XEC_TOK_XML_DECL.

XEC_OFFSET_START_ENDTAG
This is the offset of the '<' at the beginning of an XML end tag. This record occurs in the datastream immediately preceding the XEC_TOK_END_ELEM.

XEC_OFFSET_END_ENDTAG
This is the offset of the '>' at the end of an XML end tag. This record occurs in the datastream immediately following the XEC_TOK_END_ELEM.

XEC_OFFSET_START_DTD
This is the offset of the '<' at the beginning of an XML DOCTYPE declaration. This record occurs in the datastream immediately preceding the XEC_TOK_DTD_DATA.

XEC_OFFSET_END_DTD
This is the offset of the '>' at the end of an XML DOCTYPE declaration. This record occurs in the datastream immediately following the XEC_TOK_DTD_DATA.

XEC_OFFSET_START_NSVALUE
This is the offset of the quote at the beginning of an XML namespace declaration value. This record occurs in the datastream immediately preceding the XEC_TOK_NS_DECL.

XEC_OFFSET_END_NSVALUE
This is the offset of the quote at the end of an XML namespace declaration value. This record occurs in the datastream immediately following the XEC_TOK_NS_DECL.

Entities and default XML structures

If the records are inserted in the output stream via XML entity replacement or default generation, then offset information records will be generated, and the varied information field will represent the offset of the ';' character of the entity reference in the main document or the '>' character of the element which contains the default attribute. Also, all information records generated from entities will have the entity flag bit set ON.

Default XML structures include any of the following:

Attributes
These can be generated from DTDs or schemas.

Namespace declarations
These can be generated from DTDs or schemas.

Start tags and end tags
These can be generated from schemas only.

Content
These can be generated from schemas only and only within default start and end tags.

Interactions with other features

The source offsets feature can interact with other features. The following is a list of those features, along with an explanation of the interaction:

Strip comments (GXLHXEC_FEAT_STRIP_COMMENTS)
When source offsets are enabled, comment records will continue to be stripped. However, the source offset information records for comment markup will continue to be inserted into the output.

CDATA as char data (GXLHXEC_FEAT_CDATA_AS_CHARDATA)
When source offsets are enabled, CDATA will continue to be outputted as character records. However, the source offset information records for CDATA markup will continue to be inserted into the output. In this case, the order of the information records will not be in document order in relation to the data in the character records.

Validation
Information records will be created when using the validating parser as well as the non-validating parsing.

Extended end element record

If the XEC_FEAT_FULL_END feature is enabled, then the XEC_TOK_END_ELEM record will be generated as a Record Form 3 instead of Record Form 0. Here are the contents of the record when StringIDs are disabled:

Table 7. Extended end element record (no StringID)

+0	record type	flags	reserved
+4	record length		
+8	length of Lname		
+C	value of Lname		
+10			
+14	length of URI		
+18	value of URI		
+1C			
+20	length of prefix		
+24	value of prefix		
+28			

Here are the contents of the record when StringIDs are enabled:

Table 8. Extended end element record (StringID)

+0	record type	flags	reserved
+4	record length		
+8	StringID of Lname		
+C	StringID of URI		
+10	StringID of prefix		

Default attribute flag

When an Attribute Name record is generated from a definition in the DTD or schema, a flag bit will be set in the record header flags field. This bit will indicate that the attribute was generated from the DTD or schema.

31- and 64-bit compatibility

The length and offset fields outlined in the metadata records above are all 64-bit values, with associated 31-bit versions to provide 31- and 64-bit compatibility. Assembler callers in 64-bit mode can pass in buffer lengths greater than 2 GB to GXL4PRS. As a result, the z/OS XML parser may have values in length and offset fields that are much greater than 2 GB. 31-bit assembler callers are limited to 2 GB, and should reference the XEH_*31 fields in order to use the proper value. The XEH_*31 fields are in GXLYXEH . These fields can also be found in gxlhxeh.h for C/C++ callers.

Note: The offset of the error from the start of the document, when the input document is segmented and the sum of the segment sizes is greater than 2 GB, may be a 64-bit value even though the caller may only be 31-bit.

Length/Value pairs

Strings that have been parsed from the original XML document (qualified name components, character data, comment text, etc.) are, by default, represented by length/value pairs. This length indicates the actual length of the text represented by the pair. There are no string terminators, such as a NULL character used to indicate the end of a piece of text. Length fields may be zero, indicating that a particular string is not present (for example, the namespace string length for an element that is not namespace qualified will be zero), and the value length will also be zero. In the absence of a String Identifier exit (see “String Identifiers”), all strings in the parsed data stream are represented by a length/value pair.

String Identifiers

This section provides information on the String Identifiers (StringIDs) that can be passed back to the caller by the z/OS XML parser.

Note: The StringID exit is an optional service that the caller may supply. If there is no StringID exit available, the z/OS XML parser will simply return the actual length/value pairs for the strings representing localnames, URIs, and prefixes in the data stream it returns to the caller. See “Length/Value pairs” for more discussion on this topic.

StringIDs are 4 byte numeric values that are used to represent a given string that is returned from the z/OS XML parser to the caller. StringIDs can be used to represent the localname (lname), namespace prefix, and namespace URI for the following items:

- element names
- attribute names
- namespace declarations

These are the strings in the parsed data stream that are most likely to be repeated. StringIDs are provided by a caller-supplied service exit that the z/OS XML parser invokes any time it encounters certain strings that it hasn't seen before. See the description of the symbol service exit ("GXLSYM31 (GXLSYM64) — StringID service" on page 109, "GXLPSYM31 (GXLPSYM64) — StringID handler" on page 76) for more details.

Once the z/OS XML parser receives a StringID for a given string, it will record the ID, and return it in place of the actual lname, namespace prefix, or namespace URI string in the parsed data stream that is returned to the caller. The use of StringIDs reduces the size of the parsed data stream especially for documents with namespace references. URIs that would normally be returned for every element and attribute name can be represented in 4 bytes instead of their text that is generally much longer.

Record forms

The general form of a record created in the parsed data stream contains a fixed header section, followed by zero or more values. These values may consist of either a length and value pair, or a single StringID value, depending on the type of data being represented, and the data stream options that are in use. StringIDs are used to represent attribute and element name components - the lname, namespace URI, and namespace prefix for start element and attribute name records, and the namespace prefix and URI for namespace declarations. When StringIDs are not in use, these name components are represented by length and value pairs, just like other types of data returned in the records that make up the parsed data stream.

Each record begins with a fixed section that contains the record type, a set of flags, and the length of the entire record. This is followed by the values relevant to the specific type of information represented by the record. In most cases, these values represent an individual item parsed from the XML document. The exceptions are the metadata records (the buffer info and error records), which contain information describing the input and output streams, but which are not directly related to a specific item from the XML document.

The record length field is the value that must be used to navigate from one record to the next in the parsed data stream. Although the lengths and types of the individual fields of a record are explained below, the caller must not use these to calculate the location of subsequent records.

The data stream options contained in the buffer info record of each output buffer, and the token types of each record within those buffers uniquely identify the type of information contained in each record. This type information is reflected in the record form used for each record. These structures are defined in the header file "gxlhxeh.h (GXLYXEH) - mapping of the output buffer record" on page 171. For assembler callers, they are defined in GXLYXEH. Also, see Table 15 on page 26 for a description of the various record types.

Record form 0

This is a simple record that is used to describe items in the output stream that have no associated value. It consists of only a record header.

Table 9. Record form 0

record type	flags	reserved
record length		

Record form 1

These records describe items in the output stream that have one associated value - most often a character string.

Table 10. Record form 1

record type	flags	reserved
record length		
value 1 length		
bytes 1 to n of value 1		

These records are used to return things like character data to the caller. StringIDs are never used in these records.

Record form 2

These records describe items in the output stream that contain two values. There are two variations of this record form, depending on whether or not StringIDs are being used. Namespace declaration records are examples of these. In the case where StringIDs are provided by the caller through the GXLSYM31 (GXLSYM64) StringID service exit, the record form looks like the following:

Table 11. Record form 2 (with StringID)

record type	flags	reserved
record length		
StringID for value 1		
StringID for value 2		

When StringIDs are not in use, values one and two are represented as conventional length and value pairs:

Table 12. Record form 2 (without StringID)

record type	flags	reserved
record length		
value 1 length		
bytes 1 to n of value 1		
value 2 length		
bytes 1 to n of value 2		

There are other form 2 records that will always use length and value pairs, regardless of whether or not StringIDs are available. Processing instructions are an example of this kind of record, since the target and value of a processing instruction are always returned as strings represented by length and value pairs.

Record form 3

Records of this form are for parsed data that is described by 3 separate values. These records include those for element and attribute names, which can contain either StringIDs or length and value pairs, as well as XML declarations, which are always represented by the length and value pair version of this record form. Here is what the StringID based version of this form looks like:

Table 13. Record form 3 (with StringID)

record type	flags	reserved
record length		
StringID for value 1		
StringID for value 2		
StringID for value 3		

The following is the length and value pair version of the record form:

Table 14. Record form 3 (without StringID)

record type	flags	reserved
record length		
value 1 length		
bytes 1 to n of value 1		
value 2 length		
bytes 1 to n of value 2		
value 3 length		
bytes 1 to n of value 3		

Field values by record type

The following is a complete listing of the descriptions of values for each record type. The actual type of certain values will differ, depending on the use of StringID.

Table 15. Field values by record type

Record type	Record form	Contains StringIDs	Value number	Value description
GXLHXEC_TOK_BUFFER_INFO	Not applicable	Not applicable	-	See “Buffer info record” on page 18
GXLHXEC_TOK_ERROR	Not applicable	Not applicable	-	See “Error info record” on page 19
GXLHXEC_TOK_XML_DECL	3	-	1	length and value for version
			2	length and value for encoding
			3	length and value for standalone

Table 15. Field values by record type (continued)

Record type	Record form	Contains StringIDs	Value number	Value description
GXLHXEC_TOK_START_ELEM	3	No	1	length and value of Lname
			2	length and value of namespace URI
			3	length and value of namespace prefix
GXLHXEC_TOK_START_ELEM	3	Yes	1	StringID of Lname
			2	StringID of namespace URI
			3	StringID of namespace prefix
GXLHXEC_TOK_END_ELEM	0	-	-	none
GXLHXEC_TOK_END_ELEM (only used when XEC_FEAT_FULL_END feature is enabled)	3	No	1	length and value of Lname
			2	length and value of namespace URI
			3	length and value of namespace prefix
GXLHXEC_TOK_END_ELEM (only used when XEC_FEAT_FULL_END feature is enabled)	3	Yes	1	StringID of Lname
			2	StringID of namespace URI
			3	StringID of namespace prefix
GXLHXEC_TOK_ATTR_NAME	3	No	1	length and value of Lname
			2	length and value of namespace URI
			3	length and value of namespace prefix
| GXLHXEC_TOK_ATTR_NAME	3	Yes	1	StringID of Lname
			2	StringID of namespace URI
			3	StringID of namespace prefix
GXLHXEC_TOK_ATTR_VALUE	1	-	1	length and value of attribute value
GXLHXEC_TOK_NS_DECL	2	No	1	length and value of namespace prefix
			2	length and value of namespace URI
GXLHXEC_TOK_NS_DECL	2	Yes	1	StringID of namespace prefix
			2	StringID of namespace URI
GXLHXEC_TOK_CHAR_DATA	1	-	1	length and value of character data

Table 15. Field values by record type (continued)

Record type	Record form	Contains StringIDs	Value number	Value description
GXLHXEC_TOK_START_CDATA	0	-	-	none
GXLHXEC_TOK_END_CDATA	0	-	-	none
GXLHXEC_TOK_WHITESPACE	1	-	1	length and value of a white space string
GXLHXEC_TOK_PI	2	-	1	length and value of PI target
			2	length and value of PI text
GXLHXEC_TOK_COMMENT	1	-	1	length and value of comment
GXLHXEC_TOK_DTD_DATA	3	-	1	length and value of root element name
			2	length and value of public identifier
			3	length and value of system identifier
GXLHXEC_TOK_UNRESOLVED_REF	1	No	1	length and value of entity name
GXLHXEC_TOK_AUX_INFO				

The above token names are for the C/C++ callers. Assembler callers use token names without the ″GXLH″ prefix.

Spanning buffers

The z/OS XML parser is built to handle documents that may be larger than any single buffer the caller can pass to the z/OS XML parser. When buffers need to be spanned (because either the text in the input buffer is consumed, or the parsed data stream fills the output buffer), the z/OS XML parser returns a conditional success return code (XRC_WARNING), and a reason code that indicates which buffer caused the spanning condition. The caller then should handle the spanning buffer, and can optionally manage the other buffer as well.

For example, if the z/OS XML parser indicates that the output buffer is full on a return to the caller after saving and refreshing the output buffer pointers, the caller may choose to refill the input buffer with more text to parse before calling the parse service again to continue the parse process. This will require either moving the unparsed text to the front of the current input buffer, or to a new input buffer, and filling in the remainder with more unparsed text. In this way, the caller potentially reduces the number of times the z/OS XML parser has to return to the caller because of a spanned buffer during the parse of a document.

The z/OS XML parser will advance the input and output pointers to the byte after the last byte that the parser processed in each buffer. Similarly, it will update the *bytes_left* parameters to indicate the number of unprocessed or unused bytes in each buffer. The caller must use the reason code returned from the z/OS XML parser to tell which buffer must be handled and which buffer may optionally be handled. The caller cannot rely on either the *address* values or the *bytes_left* values to tell which buffer has spanned.

Splitting records

When building the parsed data stream in the output buffer, the z/OS XML parser will always ensure that all records are fully formed. Since some records represent items from the document that may be very long (eg. CDATA, white space, or comments), certain types of records are deemed to be splittable. In these cases, the z/OS XML parser will always ensure that the header for the split record is complete, but the value(s) in the record will only contain a part of the item being parsed. A flag in the record header will be set to indicate that the record is continued. Split records may span several output buffers if they are very long, or if the output buffers are relatively short.

Records that represent items of fixed length or that contain multiple values are mostly deemed to be non-splittable. If there is no room in the current output buffer to hold them, the entire record will be placed in the next output buffer. These records represent things like start element tags, attribute names, namespace declarations, or end element tags.

Note: The one exception to this rule are processing instructions (PIs). Because the text associated with PIs can be arbitrarily long, they are permitted to split.

If the z/OS XML parser determines that an output buffer is spanned, and requests another buffer to continue processing, the caller needs to return a new buffer large enough to contain a minimum set of complete data. If the item that needs to be placed at the beginning of this new buffer is a non-splittable record that doesn't fit, the z/OS XML parser will return with a return code of XRC_FAILURE, and a reason code of XRSN_BUFFER_OUTBUF_SMALL.

Splitting multibyte characters

When a caller segments an input stream for passing to the z/OS XML parser in several parts, the possibility exists that the end of an input buffer falls in the middle of a multibyte character. When this happens, the z/OS XML parser will detect the partial character, and buffer up any bytes for that character from the current buffer before returning to the caller for more input. When the next buffer of input arrives, the z/OS XML parser will virtually prefix the saved bytes of the split character to the beginning of the new buffer, and continue processing. This relieves the caller from having to ensure that multibyte characters at the end of a buffer are complete before calling the z/OS XML parser.

Processing DTDs

| z/OS XML System Services will handle internal DTDs for the purpose of processing
| entity declarations and default attribute value definitions. It only processes entity
| declarations and default attribute values from the internal DTD. Processing
| instructions that fall within the internal DTD will be returned to the caller, but no
| other text from the DTD will be processed. The z/OS XML parser will return a DTD
| record in the parsed data stream that contains the name of the root element, plus
| the system and public literals that make up the identifier of any external subset. The
| content of the internal subset is not returned to the caller.

Resolving entity references

Entities declared in the internal DTD will have all references to them in the root element resolved. These references will have the text from the entity declaration substituted for the reference, and there will be no other indications made in the parsed data stream that an entity reference was present in the parsed document.

Unresolved entities are references to entity names that have no declaration in the internal DTD. Unresolved entities in the root element are tolerated if there is an external subset (standalone=″no″ in the XML declaration). In this case, a record of type XEC_TOK_UNRESOLVED_REF is generated in the parsed data stream, with the associated value being the name of the entity. When the document only has an internal subset (standalone=″yes″), all unresolved entities are flagged as errors.

Namespace declarations

Namespace declaration records are placed in the parsed data stream between the start and end element records for the elements that contain them. This is different than in SAX-like environments where the namespace declaration events precede the start element event for a given element.

Only the namespaces that have been declared within an element, including the default namespace, will have entries in the parsed data stream for that element. The caller may construct the complete namespace context for an element by keeping a stack of namespace declarations as they are encountered in the parsed data stream. Default namespaces will have URI values, but no associated prefix. When a default namespace is unset, it is represented in the parsed data stream as a namespace declaration record with no URI or prefix.

Note: The z/OS XML parser is an XML compliant namespace parser only, and not an XML non-namespace parser. Because of this, if the z/OS XML parser parses a document compliant with the XML non-namespace standard, it can attribute namespace characteristics to an element that is not intended to contain namespaces. This is because non-namespace documents can have a ″:″ in an element structure that does not actually indicate a namespace. Thus, if non-namespace documents are being parsed, the resulting parsed data stream may not match the expected parsed data stream or the parser may flag the document as erroneous.

Using the z/OS XML parser in a multithreaded environment

The z/OS XML parser can be called from multiple work units (threads/tasks or SRBs) to parse multiple documents at the same time, provided that each parse utilizes a unique Parse Instance Memory Area (PIMA). Multiple work units must not utilize the same PIMA simultaneously, or the z/OS XML parser will behave unpredictably. As long as the invoker has a separate PIMA that has been initialized by the z/OS XML parser for each document being processed, multiple documents can be handled simultaneously. A caller may choose to preallocate a pool of PIMAs to be used for parse requests. It is the responsibility of the caller to allocate the PIMA in a subpool that will not be cleaned up while the PIMA is in use. Subpools tied to the job step task are recommended.

Chapter 5. Additional usage considerations

This chapter provides additional usage information for the z/OS XML parser. The following topics are discussed:

- “Recovery considerations”
- “Encoding support”
- “Managing memory resources” on page 32
- “Using return and reason codes” on page 33

Recovery considerations

z/OS XML provides an ARR recovery routine. This recovery routine can be turned on through an initialization option when invoked through the assembler API. For callers of the C/C++ parse API (gxlpParse), when running in Language Environment, the ARR recovery routine is provided by default in most cases. For more information on the ARR recovery routine, see “ARR recovery routine” on page 114.

Recovery can also be supplied by the caller. Callers who want to clean up z/OS XML parser resources should invoke GXL1TRM (GXL4TRM), the parser termination service, either when the parse completes or if an unexpected error occurs during the parse. The termination service will cause all secondary storage to be freed. It is up to the caller to free the PIMA storage (see “Managing memory resources” on page 32 for more information).

Encoding support

z/OS XML System Services supports several code pages. The caller must supply the CCSID of the encoding for the document at the time the z/OS XML parser is initialized. For a complete listing of the supported code pages, see Appendix L, “Supported encodings,” on page 253. The following table lists more commonly used code pages with their associated CCSID values, along with the equates provided for the caller.

Table 16. Code page CCSID values

Code page	CCSID	Equate Names
UTF-8	1208	GXLHXEC_ENC_UTF_8
UTF-16 (big endian)	1200	GXLHXEC_ENC_UTF_16
| EBCDIC/IBM-037	37	GXLHXEC_ENC_IBM_037
EBCDIC/IBM-1047	1047	GXLHXEC_ENC_IBM_1047

Assembler callers use equate names without the ″GXLH″ prefix.

The query service provides a query of a document’s XML declaration so that a caller can determine if the document has to first be converted to one of the supported encodings before parsing begins. This function will return a parsed record for the XML declaration that contains, among other things, a Coded Character Set IDentifier (CCSID) which can be passed to an encoding conversion service, such as Unicode Services, to put the document in a form that the z/OS XML parser can process. See the description for “gxlpQuery — query an XML document” on page 49 or “GXL1QXD (GXL4QXD) — query an XML document” on page 93 for more information.

EBCDIC encoding considerations

There are a couple of EBCDIC encoding considerations to deal with when trying to parse an XML file on z/OS. The first involves the character set differences between EBCDIC and Unicode. Because only a small number of Unicode characters can be represented in EBCDIC, when an EBCDIC encoded XML document is parsed, any Unicode character entity in the parsed document that does not have an EBCDIC value is converted into a dash.

Secondly, if the EBCDIC XML document has been created or modified on a z/OS system, then the line ending character is typically a NL (x'15') character. This is commonly associated with the Unicode NEL character (x'85'). For EBCDIC code page documents, the z/OS XML parser will accept XML 1.0 documents that have a NL as a line termination character, and will normalize all line-endings to EBCDIC NL (NEL). 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: For XML 1.1 documents, NL is legitimate and the z/OS XML parser is compliant in processing it as such.

Managing memory resources

The z/OS XML parser processes a document using memory resources that are provided by the caller. This storage is passed from caller to z/OS XML parser in the form of a Parse Instance Memory Area (PIMA). This required data area is used by the z/OS XML parser to suballocate a call stack, control blocks, and the tables and trees that are used to hold assorted document-specific information for the document being parsed. The environment created by the z/OS XML parser in this memory area completely describes the context of a given document parse.

A memory allocation exit is supported by the z/OS XML parser so that the caller can provide a pair of allocation/deallocation services. The allocation service will be called by the z/OS XML parser in the event that a given document causes the z/OS XML parser to exhaust the PIMA. For performance reasons, it is best if the PIMA provided by the invoker is large enough that this exit is not used. However, the exit gives the z/OS XML parser a means to complete processing of a document in the event that the memory area provided at initialization time is too small. This exit is only used to extend the PIMA, and is not used in any way to manage input or output buffers.

The deallocation service will be called by the z/OS XML parser to free the memory extension created by the allocation service. The deallocation service will never free the original PIMA storage.

For callers that do not provide a memory allocation exit, the z/OS XML parser provides default routines to allocate and free memory. The z/OS XML parser also provides an option at initialization time allowing the caller to specify how the z/OS XML parser's default routine allocates memory. This feature should be specified when PIMAs are used on multiple tasks, in order to prevent task termination from causing storage extents to be freed before the z/OS XML parser is done using them. Normally, z/OS XML parser will allocate memory at the task level. However, when the feature is specified, the z/OS XML parser will allocate memory at the Job Step Task (JST) level instead.

In both cases, the caller is assuming the responsibility to call GXL1TRM (GXL4TRM) in the event the z/OS XML parser abends and the caller's recovery gets control.

When no memory allocation exit is provided, the subpool used will be as follows:

- If running in SRB or cross memory mode, subpool 129 will be used. This is JST related and cannot be freed by unauthorized callers. The key will be the same as the key at the time the z/OS XML parser is invoked.
- If running in task mode (PSATOLD not zero), with PRIMARY=SECONDARY=HOME, then the subpool chosen will depend on the authorization state of the caller and on the specification of the XEC_FEAT_JST_OWNS_STORAGE feature on the GXL1INI (GXL4INI) call. If the caller is running in key 0-7 or supervisor state, they will be considered authorized.
  - Authorized and JST requested — subpool 129
  - Authorized and JST not requested — subpool 229
  - Unauthorized and JST requested — subpool 131
  - Unauthorized and JST not requested — subpool 0

  Note: If running on a subtask which is sharing subpool 0, then this storage will be owned by the task that owns subpool 0.

These choices of subpool will eliminate the possibility of the z/OS XML parser running in an authorized state while using problem key storage which could be freed and reallocated.

Using return and reason codes

The z/OS XML parser API services provide a return and reason code to indicate the success or failure of the parse process. The return code is a fullword value that indicates the class of the return status, and takes on one of the following values:

- Success (XRC_SUCCESS)
- Warning (XRC_WARNING) - parsing is successful, but incomplete. This is most often caused by the z/OS XML parser reaching the end of either the input or the output buffer.
- Failure (XRC_FAILURE) - a terminating failure has occurred. The return information passed back in the parameters, such as the numbers of bytes left in the input and output buffers, are valid. The extended diagnostic information may also contain additional problem determination information that is of use.
- Not-well-formed (XRC_NOT_WELL_FORMED) - a terminating failure has occurred because the input document is not well formed. As with the failure case above, all return information passed back through the parameters and extended diagnostic area is valid.
- Fatal (XRC_FATAL) - a terminating error has occurred. None of the return information is valid.
- Not valid (XRC_NOT_VALID) - The document is not valid according to the specified schema.

In addition to the return code describing the class of error, the reason code provides more detail. The reason code is only valid when the return code is not XRC_SUCCESS. When a service of the z/OS XML parser API returns XRC_SUCCESS, the reason code may have any random value.

The reason code itself is a fullword value, but is made up of two halfwords. The upper halfword is reserved for a module identifier that is used by IBM Service to isolate the source of the problem, and the lower halfword indicates the reason why the parse process was paused or terminated. When checking the value of the reason code, the caller must be sure to AND the reason code with the reason code mask (XRSN_REASON_MASK) before testing the value. The declaration of XRSN_REASON_MASK and all of the defined reason code values are contained in the GXLYXR macro. A list of the reason codes and their descriptions can be found in Appendix B, "Reason Codes Listed by Value," on page 119.

Chapter 6. z/OS XML parser API: C/C++

This chapter lists the C/C++ callable services interface used for the z/OS XML parser.

Setting XPLINK compiler option

If the calling application is compiled without XPLINK and wants to use z/OS XML System Services, the calling application must set the following option:

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

If this option is not set, an error will occur once the application is run.

For more information on the XPLINK compiler option, see z/OS XL C/C++ User's Guide.

Support for the Metal C compiler option

Support is provided for callers who wish to use the Metal C compiler option. The same APIs available to the standard C and C++ callers are also available to Metal C users, with the following restrictions:

- All parameters must be variables.
- The functions do not return values.

Note: Return codes and reason codes are still returned through the parameter lists.

For more information on how to use the Metal C compiler option, see *Metal C Run-time Library Guide and Reference*.

Where to find the header files, DLLs and side decks

Header files for non-Metal C can be found in the z/OS UNIX directory `/usr/include`. Header files for Metal C can be found in the z/OS UNIX directory `/usr/include/metal`. If you are not using z/OS UNIX, then the non-Metal C header files can be found in the PDSE SYS1.SIEAHDRV.H . There are no Metal C header files for the batch environment.

DLLs for non-Metal C can be found in the z/OS UNIX directory `/usr/lib`. If you are not using z/OS UNIX, then the DLLs can be found in SYS1.SIEALNKE . There are no DLLs for Metal C.

Side decks for non-Metal C can be found in `/usr/lib`. If you are not using z/OS UNIX, then the side decks can be found in SYS1.SIEALNKE . There are no side decks for Metal C.

Using the recovery routine

z/OS XML provides an ARR recovery routine to assist with problem determination and diagnostics. In the C/C++ environment, the recovery routine is provided as the default setting in most cases and will recover the code and collect dumps for most abends that occur during a parse. For unauthorized C/C++ callers, an IEATDUMP will be taken in data set *userid*.GXLSCXML.DYYMMDD.THHMMSS.DUMP, where the *userid* is extracted from the task level ACEE if present or the address space

ACEE, and where DYYMMDD is the date and THHMMSS is the time the dump was taken. For authorized C/C++ callers, an SDUMPX will be taken into a system dump data set. See “ARR recovery routine” on page 114 for more information.

In order to effectively use the recovery routine, you must set the following runtime option: TRAP(ON,NOSPIE). If this runtime option is not set, unpredictable behavior may result with regard to recovery.

z/OS XML XL C/C++ API

gxlpControl — perform a parser control function

Description

This is a general purpose service which provides control functions for interacting with the z/OS XML parser. The function performed is selected by setting the *ctl_option* parameter using the constants defined in gxlhxec.h . These functions include:

GXLHXEC_CTL_FIN
The caller has finished parsing the document. Reset the necessary structures so that the PIMA can be reused on a subsequent parse, and return any useful information about the current parse.

GXLHXEC_CTL_FEAT
The caller wants to change the feature flags. A XEC_CTL_FIN function will be done implicitly.

Note: XEC_FEAT_JST_OWNS_STORAGE is not supported on gxlpControl. Make sure that the feature flag is turned to the ″off″ state before calling gxlpControl to set the feature flag, otherwise the gxlpControl request will fail.

GXLHXEC_CTL_LOAD_OSR
The caller wants to load and use an Optimized Schema Representation (OSR) for a validating parse.

Note: finish-and-reset processing is performed by all operations available through this control service.

Performance Implications

The finish-and-reset function allows the caller to re-initialize the PIMA to make it ready to handle a new XML document. This re-initialization path enables the z/OS XML parser to preserve its existing symbol table, and avoid other initialization pathlength that's performed by calling the initialization service. The reset features function also allows the caller to re-initialize the z/OS XML parser as above and allows the feature flags to be reset as well.

Syntax

```
int gxlpControl (void * PIMA,
                int ctl_operation,
                void * ctl_data_p,
                int * rc_p,
                int * rsn_p);
```

Parameters

PIMA
Supplied parameter

Type:	void *

The name of the Parse Instance Memory Area (PIMA which has been previously initialized with a call to the initialization service.

ctl_operation
Supplied parameter

Type:	int

The name of the parameter containing an integer value representing one of the following operations:

GXLHXEC_CTL_FIN
This indicates that the caller wishes to end the current parse at the current position in the XML document. The PIMA is re-initialized to allow it to be used on a new parse request. To free up all resources associated with the parse instance, the caller should use the termination service.

GXLHXEC_CTL_FEAT
This indicates that the caller wishes to re-initialize the z/OS XML parser, as with the reset-and-finish function as above and in addition, to reset some of the feature flags used during the parse.

Note: The following feature flags are not supported by this service:
- GXLHXEC_FEAT_JST_OWNS_STORAGE
- GXLHXEC_FEAT_RECOVERY
- GXLHXEC_FEAT_VALIDATE

Make sure that these feature flags are turned to the OFF state before calling this service to set the feature flags. If these features need to be changed (for example, if switching between validating and non-validating parses), the parse instance must be terminated and re-initialized with the required feature settings.

GXLHXEC_CTL_LOAD_OSR
This indicates that the caller wants to load and use a given Optimized Schema Representation (OSR) during a validating parse. This operation will also cause the parser to perform reset-and-finish processing.

ctl_data_p
Supplied and returned parameter

Type: void *

The name of the parameter that contains the address of an area as defined by ctl_operation:

GXLHXEC_CTL_FIN
This parameter must contain a pointer to where the service will store the address of the diagnostic area, which is mapped by header file gxlhxd.h . This provides additional information that can be used to debug problems in data passed to the z/OS XML parser. The diagnostic area resides within the PIMA, and will be overlaid on the next call to the z/OS XML parser. If the caller does not wish receive diagnostic information, the NULL value is used in place of the address of the diagnostic area.

GXLHXEC_CTL_FEAT
This parameter must contain the address of a fullword (doubleword), which is mapped by header file gxlhxft.h. See “gxlhxft.h (GXLYXFT) - mapping of the control feature input output area” on page 174 for more information on this header file.

The GXLHXFT_FEAT_FLAGS parameter is an input parameter to the API and contains the value of feature flags to be used in the subsequent parse. It is defined as follows:

GXLHXEC_FEAT_STRIP_COMMENTS
This effectively strips comments from the document by not returning any comments in the parsed data stream. Default: off.

GXLHXEC_FEAT_TOKENIZE_WHITESPACE
This sets the default token value for white space preceding markup in the root element to an explicit white space value. Default: off – white space is returned as character data.

GXLHXEC_FEAT_CDATA_AS_CHARDATA
This returns CDATA in records with a CHARDATA token type. The content of these records may contain text that would normally have to be escaped to avoid being handled as markup. Default: off.

GXLHXEC_FEAT_SOURCE_OFFSETS
This feature is used to include records in the parsed data stream which contain offsets to the corresponding structures in the input document. Default: off.

GXLHXEC_FEAT_FULL_END
This feature is used to expand the end tags to include the local name, prefix and URI corresponding to the qname on the end tag. Default: off.

If none of the features are required, pass the name of a fullword field containing zero. Do not construct a parameter list with a zero pointer in it.

The GXLHXFT_XD_PTR must contain the address of a fullword (doubleword) where the service will store the address of the diagnostic area, which is mapped by header file gxlhxd.h. This provides additional information that can be used to debug problems in data passed to the z/OS XML parser. The diagnostic area resides within the PIMA, and will be overlaid on the next call to the z/OS XML parser.

GXLHXEC_CTL_LOAD_OSR

This indicates that the caller wants to load and use a given Optimized Schema Representation (OSR) during a validating parse. Once an OSR has been loaded, it remains in use for all validating parse requests until a different OSR is provided by calling this service again.

This parameter must contain the address of an area containing information about the OSR to load. This area is mapped by gxlhxosr.h. See “gxlhxosr.h (GXLYXOSR) - mapping of the OSR control area” on page 174 for more information on the structures in this header. Also, see the usage notes below for details about how the information in this area is used.

rc_p
Returned parameter

Type:	int *

The name of the area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

The name of the area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code is 0 (XRC_SUCCESS). Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173), and are dependent on the control function specified by the caller. For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
void * PIMA;
int ctl_operation = GXLHXEC_CTL_FIN;
void * diag_addr;
int rc_p, rsn_p;
gxlpControl(PIMA, ctl_operation, &diag_addr, &rc_p, &rsn_p);

void * PIMA;
int ctl_operation = GXLHXEC_CTL_FEAT;
GXLHXFT xft;
xft.XFT_FEAT_FLAGS = GXLHXEC_FEAT_STRIP_COMMENTS;
void * ctlData = (void*)&xft;
int rc_p, rsn_p;
gxlpControl(PIMA, ctl_operation, (void*)&ctlData, &rc_p, &rsn_p);
```

Usage notes

This callable service is mapped to GXL1CTL (GXL4CTL). Refer to“Usage notes” on page 85 of GXL1CTL (GXL4CTL) for usage information.

gxlpInit — initialize the z/OS XML parser

Description

The gxlpInit callable service initializes the PIMA and records the addresses of the caller's system service routines (if any). The PIMA storage is divided into the areas that will be used by the z/OS XML parser to process the input buffer and produce the parsed data stream.

Performance Implications

The initialization of structures used by the z/OS XML parser in the PIMA is only done once per parse and is therefore unlikely to affect performance. The caller may choose to reuse the PIMA after each parse to eliminate the overhead of storage allocation and the page faults that occur when referencing new storage. In this case, a control operation is required to reset the necessary fields in the PIMA before parsing can continue. For more information on the control operation, see "gxlpControl — perform a parser control function" on page 37.

Syntax

```
int gxlpInit (void * PIMA,
              long PIMA_LEN,
              int ccsid,
              int feature_flags,
              GXLHXSV sys_svc_vector,
              void * sys_svc_parm,
              int * rc_p,
              int * rsn_p);
```

Parameters

PIMA

Pointer to Parse Instance Memory Area (PIMA).

Type:	void *

PIMA_Len

Length of PIMA

Type:	long

| The name of an area containing the length of the Parse Instance Memory Area. This service validates the length of this area against a minimum length value. The minimum length of the PIMA depends on whether or not validation will be performed during the parse:

- GXLHXEC_NVPARSE_MIN_PIMA_SIZE (non-validating)
- GXLHXEC_VPARSE_MIN_PIMA_SIZE (validating)

ccsid

Supplied parameter

Type:	Integer

The Coded Character Set IDentifier (CCSID) that identifies the document's character set. The CCSID value in this parameter will override any character set or encoding information contained in the XML declaration of the document. A set of CCSID constants for supported encodings has been declared in GXLYXEC. See Appendix L, "Supported encodings," on page 253 for a full list of supported encodings.

feature_flags

Supplied parameter

Type: Integer

The name of the area that contains an integer value representing one or more of the following z/OS XML parser features. OR these flags together as needed to enable features. Choose any of the following:

- **GXLHXEC_FEAT_CDATA_AS_CHARDATA** - return CDATA in records with a CHARDATA token type. The content of these records may contain text that would normally have to be escaped to avoid being handled as markup.
- **GXLHXEC_FEAT_FULL_END** - expand the end tags to include the local name, prefix and URI corresponding to the qname on the end tag.
- **GXLHXEC_FEAT_JST_OWNS_STORAGE** - allocate storage as Job Step Task (JST) related instead of task related. See the "Usage notes" on page 88 below for more information.
- **GXLHXEC_FEAT_RECOVERY** - this option is used to turn on the recovery routine.

  Note: This option is only valid when using the Metal C compiler option.
- **GXLHXEC_FEAT_SOURCE_OFFSETS** - include records in the parsed data stream which contain offsets to the corresponding structures in the input document.
- **GXLHXEC_FEAT_STRIP_COMMENTS** - effectively strip comments from the document by not returning any comments in the parsed data stream.
- **GXLHXEC_FEAT_TOKENIZE_WHITESPACE** - set the default token value for white space preceeding markup within the context of the root element to an explicit white space value. Use this value in conjunction with the special xml:space attribute to determine how such white space gets classified.
- **GXLHXEC_FEAT_VALIDATE** - perform validation while parsing. See "Usage notes" on page 88 for details of parsing with validation.

Note: By using the values of off (zero), W3C XML compliant output is generated. Turning on options GXLHXEC_FEAT_STRIP_COMMENTS and GXLHXEC_FEAT_CDATA_AS_CHARDATA will cause the output to vary from standard compliance.

If none of the features are required, pass the name of a fullword field containing zero. Do not construct a parameter list with a zero pointer in it.

sys_svc_vector
Supplied parameter

Type: GXLHXSV

The name of a structure containing a count of entries that follow and then a list of 31 (64) bit pointers to system service routines. The GXLHXSV member XSV_COUNT must have a value of 0 if no services are provided. For more details on usage, see "Usage notes" on page 43. For more information on exit routines, see the Chapter 8, "z/OS XML System Services exit interface" chapter.

sys_svc_parm
Supplied parameter

Type: void *

The name of the area which is passed to all system service exits. This provides for communication between the z/OS XML parser caller and its exit routines. Specify the name of a location containing 0 if no parameter is required for communication.

rc_p
Returned parameter

Type:	int *

The name of the area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

The name of the area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <gxlhxec.h>

void *  pima_p;
long    pima_l;
GXLHXSV sysServiceVec;
int     rc, rsn;

if (pima_p = malloc(GXLHXEC_MIN_PIMA_SIZE))
  { /* pima malloc succeeded */
   pima_l = GXLHXEC_MIN_PIMA_SIZE;
   sysServiceVec.XSV_COUNT = 0;

gxlpInit(pima_p, pima_l,
      GXLHXEC_ENC_UTF_8,
      GXLHXEC_FEAT_STRIP_COMMENTS,
      sysServiceVec,
      NULL,
      &rc, &rsn);

  } /* pima malloc succeeded */
```

Usage notes

System service exit routines cannot get control in the C/C++ environment. Instead, they must be coded to the assembler interface.

Addresses passed in the *system_service_vec* parameter must point to the entry point of the exit being supplied. To obtain the entry point address of a function in 31-bit NOXPLINK DLL compiled module, refer to the FDCB structure in *z/OS Language Environment Vendor Interfaces, SA22-7568*. Otherwise, taking the address of the function will return the entry point address.

This callable service is a direct map to the callable service GXL1INI (GXL4INI). Refer to “Usage notes” on page 88 of GXL1INI (GXL4INI) for additional usage information.

gxlpLoad — load a z/OS XML function

Description

Load a module that implements a z/OS XML function into storage.

Performance Implications

There are no performance implications.

Syntax

```
int gxlpLoad (int function_code,
                  void * function_data,
                  int * rc_p,
                  int * rsn_p)
```

Parameters

PIMA
Supplied parameter

Type:	int

This parameter identifies the z/OS XML function to load. It is the name of an integer value representing the following function:

XEC_LOD_VPARSE
The validating parse function

See gxlhxec.h for the list of function code constants.

function_data
Returned parameter

Type:	void *

Specify a word of zeroes for this parameter.

rc_p
Returned parameter

Type:	int *

The name of the area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

The name of the area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

None.

Usage notes

This load step is not required for performing non-validating parsing. This operation is only required when using the validating parser. The caller does have the option of loading the load module for the specified function without using this service - either through the z/OS LOAD macro (assembler interface), or by putting it in LPA or the extended LPA. Both the LOAD macro and calls to this service are not allowed when running in an SRB. The use of either interface must be performed in the task before entering SRB mode.

If the required z/OS XML function is made available, either by LOADing the executable load module for it or putting the load module in LPA, this service is not required. Documentation on the LOAD macro can be found in z/OS MVS Programming: Assembler Services Reference, Volume 2, and information on how to load modules into LPA can be found in z/OS Initialization and Tuning Guide.

The load module associated with the function is as follows:

Table 17. Load module for C/C++ parser

Function code	**Function performed**	**Load module name**
XEC_LOD_VPARSE	Validating parser function	GXLIMODV

gxlpParse — parse a buffer of XML text

Description

The gxlpParse callable service parses a buffer of XML text and places the result in an output buffer.

Performance Implications

Ideal performance will be obtained when the PIMA is sufficiently large to contain all the needed data structures, and the input and output buffers are large enough to process the entire XML document. During the parsing process, the z/OS XML parser constructs persistent information in the PIMA that can be reused within a parse instance. If the caller is going to process multiple documents that contain similar sets of symbols (namespaces and local element and attribute names in particular), then reusing the PIMA will improve performance during the processing of subsequent documents. If this behavior is not required, the PIMA should be cleaned up by calling the termination service and reinitialized by calling the initialization service before using the PIMA for another parse request.

Syntax

```
int gxlpParse(void * PIMA,
              int * option_flags,
              void ** input_buffer_addr,
              long * input_buffer_bytes_left,
              void ** output_buffer_addr,
              long * output_buffer_bytes_left,
              int * rc_p,
              int * rsn_p);
```

Parameters

PIMA
Supplied parameter

Type:	void *

The name of the Parse Instance Memory Area (PIMA which has been previously initialized with a call to the initialization service.

option_flags
Supplied parameter

Type:	int *

This parameter must point to a word with the value 0.

input_buffer_addr
Supplied and returned parameter

Type:	void **

The name of the area that contains the address of the buffer with the XML text to parse. The z/OS XML parser updates this parameter to provide important return information when control returns to the caller. See the “Usage notes” on page 92 for details.

input_buffer_bytes_left
Supplied and returned parameter

Type:	long *

The name of the area that contains the number of bytes in the input buffer that have not yet been processed. The z/OS XML parser updates this parameter to provide important return information when control returns to the caller. See the “Usage notes” on page 92 for details.

output_buffer_addr
Supplied and returned parameter

Type:	void **

The name of the area that contains the address of the buffer where the z/OS XML parser should place the parsed data stream. The z/OS XML parser updates this parameter to provide important return information when control returns to the caller. See the “Usage notes” on page 92 for details.

output_buffer_bytes_left
Supplied and returned parameter

Type:	long *

The name of the area that contains the number of available bytes in the output buffer. When the z/OS XML parser returns control to the caller, this parameter will be updated to indicate the number of unused bytes in the output buffer. This buffer must always contain at least a minimum number of bytes as defined by the **GXLHXEC_MIN_OUTBUF_SIZE** constant, declared in header file gxlhxec.h. This service will validate the length of this area against this minimum length value.

rc_p
Returned parameter

Type:	int *

The name of the area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

The name of the area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
void * PIMA;

int * option_flags;

void * input_buffer_addr; int input_buffer_bytes_left;

void * output_buffer_addr; int output_buffer_bytes_left;
```

```
int rc, rsn;

gxlpParse(PIMA,
          option_flags,
          &input_buffer_addr, &input_buffer_bytes_left,
          &output_buffer_addr, &output_buffer_bytes_left,
          &rc, &rsn);
```

Usage notes

This callable service is a direct map to GXL1PRS (GXL4PRS). Refer to "Usage notes" on page 92 of GXL1PRS (GXL4PRS) for usage information.

gxlpQuery — query an XML document

Description

This service allows a caller to obtain the XML characteristics of a document. The XML characteristics are either the default values, the values contained in an XML declaration or a combination of both.

Performance Implications

There are no performance implications.

Syntax

```
int gxlpQuery (void * work_area,
               long work_area_length,
               void * input_buffer,
               long input_buffer_length,
               GXLHQXD ** return_data,
               int * rc_p,
               int * rsn_p);
```

Parameters

work_area

Supplied parameter

Type:	void *

The name of a work area. The work area must be aligned on a doubleword boundary. If not on a doubleword boundary, results are unpredictable. See the “Usage notes” on page 94 for additional details on the use of this area.

work_area_length

Supplied parameter

Type:	long

The name of an area containing the length of the work area. The minimum length of this area is declared as a constant **GXLHXEC_MIN_QXDWORK_SIZE** in header file gxlhxec.h . This service validates the length of this area against this minimum length value.

input_buffer

Supplied parameter

Type:	void *

The name of an input buffer containing the beginning of the XML document to process. See the “Usage notes” on page 94 for details.

input_buffer_length

Supplied parameter

Type:	long

The name of an area containing the length of the input buffer.

return_data

Returned parameter

Type:	GXLHQXD **

The pointer to where the service will return the address of the data which describes the XML document characteristics. This return information will contain values that are either extracted from the XML declaration or defaulted according to the XML standard. This return area is mapped by the header file gxlhqxd.h (see “gxlhqxd.h (GXLYQXD) - mapping of the output from the query XML declaration service” on page 172), and is located within the work area specified by the work_area parameter. The caller must not free the work_area until it is done referencing the data returned from this service.

rc_p
Returned parameter

Type:	int *

The name of the area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

The name of the area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
void * work_area;
long work_area_length = XEC_MEM_QIMA_SIZE;
void * input_buffer;
long input_buffer_length;
GXLHQXD * return_data;
int rc, rsn;
gxlpQuery(work_area, work_area_length, input_buffer, input_buffer_length, &return_data, &rc, &rsn );
```

Usage notes

This callable service is a direct map to GXL1QXD (GXL4QXD). Refer to “Usage notes” on page 94 of GXL1QXD (GXL4QXD) for usage information.

gxlpTerminate — terminate a parse instance

Description

The gxlpTerminate callable service releases all resources obtained (including storage) by the z/OS XML parser and resets the PIMA so that it can be re-initialized or freed.

Performance Implications

There are no performance implications.

Syntax

```
int gxlpTerminate (void * PIMA,
                   int * rc,
                   int * rsn);
```

Parameters

PIMA
Supplied parameter

Type:	void *

The name of the Parse Instance Memory Area (PIMA) which has been previously initialized with a call to the initialization service.

rc Returned parameter

Type:	int *

The name of the area where the service stores the return code.

rsn
Returned parameter

Type:	int *

The name of the area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
void * PIMA;

int rc, rsn;

gxlpTerminate (PIMA, &rc, &rsn);
```

Usage notes

This callable service is a direct map to GXL1TRM (GXL4TRM). Refer to “Usage notes” on page 97 of GXL1TRM (GXL4TRM) for usage information.

OSR generator API

gxluInitOSRG — initialize an OSR generator instance

Description

Initialize an OSR generator instance. This establishes a context within which the OSR generator performs operations on schemas, Optimized Schema Representations (OSRs), and StringID tables. This context is defined by the OSR generator Instance Memory Area (OIMA).

Performance Implications

The OIMA must be initialized before any OSR generation operations are performed. If operations are to be performed on different OSRs, the caller may enhance performance by resetting the OIMA through a control operation (see gxluControlOSRG), rather than terminating the generator instance and re-initializing. There are implications for memory consumption that must be considered when multiple OSRs are created from the same generator instance. See the usage notes below.

Syntax

```
int gxluInitOSRG (void * oima_p,
                  unsigned long oima_l,
                  int feature_flags,
                  void * sys_svc_parm_p,
                  int * rc_p,
                  int * rsn_p)
```

Parameters

oima_p
Supplied and returned parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA). This area must be at least GXLHXEC_MIN_OIMA_SIZE bytes long. It is used as the work area for the OSR generator.

oima_l
Supplied parameter

Type:	unsigned long

The length of the OSR generator Instance Memory Area (OIMA) pointed to by the oima_p parameter.

feature_flags
Supplied parameter

Type:	int

Specify a value of zero for this parameter. This parameter is not used.

sys_svc_parm_p
Supplied parameter

Type:	void *

A pointer to an area which is passed to all system service exits, handlers, and resolvers. This provides for communication between the caller of the z/OS XML OSR generator and its exit routines. Specify the NULL pointer if no parameter is required for communication.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this utility is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see "gxlhxr.h (GXLYXR) - defines the return codes and reason codes" on page 173). For reason code descriptions, also see Appendix B, "Reason Codes Listed by Value," on page 119.

Example

```
#include <stdlib.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
  {       /* oima malloc succeeded */
    oima_l = GXLHXEC_MIN_OIMA_SIZE;

     gxluInitOSRG(oima_p, oima_l,
           0,
                 (void *)handler_parms,
                 &rc, &rsn);

   /* Use the OSR generation instance to perform schema operations ... */
  }       /* oima malloc succeeded */
```

Usage notes

When creating multiple OSRs, the best practice will usually be to initialize one generator instance, and use it for all of the generation operations, with control requests to reset the generator between OSRs. This will consume fewer CPU cycles, and provide better overall performance than initializing and terminating a generator instance for each OSR being created or operated upon. However, all generated OSRs will remain in memory for the duration of the generator instance. If memory constraints are a concern, or you plan to generate OSRs for either a large number of schemas, or for schemas that are very large, you may need to terminate and re-initialize the OSR generator.

gxluControlOSRG — perform an OSR generator control operation

Description

This is a general purpose utility which provides operations for controlling the z/OS XML OSR generator. The operation performed is selected by setting the ctl_option parameter using the constants defined in gxlhxoc.h and gxlhxec.h. These functions include:

GXLHXEC_OSR_CTL_FIN
: The caller has finished working with a particular OSR. Reset the necessary structures so that the OIMA can be reused for subsequent generator operations on a different OSR. Receive extended diagnostic information about the current context of the OSR generator.

GXLHXEC_OSR_CTL_DIAG
: The caller has finished working with a particular OSR. Receive extended diagnostic information about the current context of the OSR generator.

Performance Implications

The finish-and-reset function allows the caller to re-initialize the OIMA to make it ready to handle a new OSR. This re-initialization path enables the z/OS XML OSR generator to avoid one-time initialization pathlength that's performed by the initialization service.

Syntax

```
int gxluControlOSRG(void * oima_p,
                int ctl_operation,
                void * ctl_data_p,
                int *  rc_p,
                int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

ctl_operation
Supplied parameter

Type:	int

The name of the parameter containing an integer value representing one of the following operations:

GXLHXEC_OSR_CTL_FIN
: This indicates that the caller wants to end processing on the current OSR. The OIMA is re-initialized to allow it to be used to process a new, different OSR. This operation will also return the extended diagnostic information area that is mapped by the gxlhosrd.h header. This includes problem determination information relevant to the current context of the OSR generator.

GXLHXEC_OSR_CTL_DIAG
: This indicates that the caller wants to end processing on the current OSR. This operation will return the extended diagnostic information

area that is mapped by the gxlhosrd.h header. This includes problem determination information relevant to the current context of the OSR generator.

ctl_data_p
Supplied and returned parameter

Type:	void *

A pointer to an area that will be used for a purpose that depends on the control operation being performed:

GXLHXEC_OSR_CTL_FIN
A pointer to an area that will receive the address of the extended diagnostic area mapped by gxlhosrd.h. If NULL is specified for this parameter, no extended diagnostic information will be returned. See the usage notes for more about how to use this area.

GXLHXEC_OSR_CTL_DIAG
A pointer to an area that will receive the address of the extended diagnostic area mapped by gxlhosrd.h. If NULL is specified for this parameter, no extended diagnostic information will be returned. See the usage notes for more about how to use this area.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the service stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this service is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see "gxlhxr.h (GXLYXR) - defines the return codes and reason codes" on page 173). For reason code descriptions, also see Appendix B, "Reason Codes Listed by Value," on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
```

```
GXLHOSRD *    XDArea_p;
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
   { /* oima malloc succeeded   */
     oima_l = GXLHXEC_MIN_OIMA_SIZE;

     gxluInitOSRG(oima_p, oima_l,
                  0,
                  (void *)handler_parms,
                  &rc, &rsn);
   } /* oima malloc succeeded   */

/* Now perform operations using the generator instance. */

if ((oima_p > 0) && (rc == XRC_SUCCESS))

  { /* generator ininitialized */

/* generate or load an OSR, generate a StringID table, etc */

   gxluControlOSRG(oima_p,
             GXLHXEC_OSR_CTL_FIN,
             (void *)&XDArea_p,
             &rc, &rsn);

if (rc == XRC_SUCCESS)
    { /* reset succeeded    */

if (XDArea_p->strID_RC != 0)
     {  /* StringID exit failure   */
      fprintf(stderr,"StringID exit failure: %08x\n",
              XDArea_p->strID_RC);
      ...
      } /* StringID exit failure   */

     } /* reset succeeded         */

    ...

  } /* generator ininitialized */
```

Usage notes

The purpose of the finish-and-reset operation of this service is to reset the necessary structures and fields within the OIMA to prepare the generator instance for reuse without the overhead of full initialization. This reset operation uses fewer CPU cycles than terminating and re-initializing from scratch. However, all schemas that are loaded, and all OSRs and StringID tables that are generated, remain in memory for the duration of the OSR generation instance. If you have a large number of schemas to process, or if the schemas are very large in size, memory constraints may become an issue. In this case, it will be necessary to terminate and re-initialize the OSR generator instance.

The extended diagnostic area returned by the GXLHXEC_OSR_CTL_FIN and GXLHXEC_OSR_CTL_DIAG operations are mapped by gxlhosrd.h. The structure in this header contains assorted diagnostic information about the particular phase of OSR generation that may have failed. The fields of this structure may be used for the duration of the OSR generator instance, but must not be referenced after the instance is terminated. Doing so may result in unpredictable results.

gxluTermOSRG — terminate an OSR generator instance

Description

The gxlpTermOSRG utility releases all resources obtained by the z/OS XML OSR generator. It also sets the eyecatcher in the OIMA to prevent it from being reused by other OSR API functions, with the exception of re-initialilization by gxluInitOSRG.

Performance Implications

There are no performance implications.

Syntax

```
int gxluTermOSRG(void * oima_p,
                 int *  rc_p,
                 int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type: void *

A pointer to an OSR generator Instance Memory Area (OIMA).

rc_p
Returned parameter

Type: int *

A pointer to an area where the service stores the return code.

rsn_p
Returned parameter

Type: int *

A pointer to an area where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this service is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>
```

```
void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
        oima_l = GXLHXEC_MIN_OIMA_SIZE;

        gxluInitOSRG(oima_p, oima_l,
                     0,
                     (void *)handler_parms,
                     &rc, &rsn);
    }  /* oima malloc succeeded   */

/* Now perform operations using the generator instance. */
if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator ininitialized */

    /* generate or load an OSR, generate a StringID table, etc */

    gxluTermOSRG(oima_p,
                 &rc, &rsn);
  /* Do not use any resources that the OSR generator   */
  /* has allocated from here on.                       */

  }  /* generator ininitialized */
```

Usage notes

This utility does not free the OSR Generator Instance Memory Area (OIMA). It is up to the caller to free the OIMA after termination completes. gxluTermOSRG will, however, free any binary OSR buffers, StringID tables, and extended diagnostic areas that may have been allocated during the OSR generator instance. Once termination has completed, you must not reference any of these areas, or any extended diagnostic areas that may have been created during the generator instance. It is the caller's responsibility to create persistent copies of these structures as needed while the generator instance is active.

gxluLoadSchema — load a schema into the OSR generator

Description

This utility is used to load text schemas into the OSR generator. It is called once for each schema that will be processed to create an Optimized Schema Representation.

Performance Implications

There are no performance implications.

Syntax

```
int gxluLoadSchema(void * oima_p,
                   char * schema_resource_p,
                   int *  rc_p,
                   int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

schema_resource_p
Supplied parameter

Type:	char *

A pointer to the schema resource to process. This parameter must contain a NULL terminated, IBM-1047 text string representing one of the following:

- The pathname of a file in the z/OS UNIX file system containing the schema in text form.
- URI specifying the location of the schema text to load. URIs are indicated by a scheme name, followed by a colon, followed by a relative URI reference. See RFC 3986 (http://tools.ietf.org/html/rfc3986) for a complete description of URIs.

Whether the resource passed is a URI or a pathname to a file, the name must represent an absolute path. Relative paths cannot be processed.

Note: Although the URI or pathname to the schema being loaded must be an IBM-1047 string, the schema itself must be encoded as a UTF-8 document.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this service is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
char   schema_uri[URI_LEN] = "file:///u/user01/myschema.xsd";
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
         oima_l = GXLHXEC_MIN_OIMA_SIZE;

        gxluInitOSRG(oima_p, oima_l,
                      0,
                     (void *)handler_parms,
                      &rc, &rsn);
    }  /* oima malloc succeeded   */

/* Now perform operations using the generator instance. */
if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator initialized */

       gxluLoadSchema(oima_p,
                      schema_uri,
                      &rc, &rsn);

        if (rc == XRC_SUCCESS)
        {/* schema load succeeded   */

 /*generate an OSR from the loaded schema*/


    ...
    }  /* schema load succeeded   */

   ...
   } /* generator initialized */
```

Usage notes

Call this service iteratively to load one or more schemas that will be processed to create an OSR. Once a schema has been loaded, the schema text buffer specified by the schema_resource_p parameter may be re-used for other purposes.

Make sure to load only UTF-8 schema documents in the OSR generator instance. UTF-8 is the canonical form that the OSR generator requires, and schemas in other encodings will not be processed.

gxluSetStrIDHandler — specify the StringID handler for OSR generation

Description

This utility allows the caller to specify a StringID handler service to the OSR generator. The StringID handler utility allows the caller to avoid making StringID calls at parse time for a number of symbols. This handler must be written in C.

Performance Implications

There are no performance implications.

Syntax

```
int gxluSetStrIDHandler(void * oima_p,
                   char * dll_name_p,
                   char * func_name_p,
                   int *  rc_p,
                   int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

dll_name_p
Supplied parameter

Type:	char *

A pointer to the NULL terminated name of the DLL containing the StringID handler executable. This string must be in the IBM-1047 code page. A NULL string indicates that the current StringID handler should be unset, and StringIDs no longer used during the creation of OSRs.

func_name_p
Supplied parameter

Type:	char *

A pointer to the NULL terminated name of the StringID handler within the DLL. This string must be in the IBM-1047 code page. If the dll_name_p parameter above is NULL, this function name is ignored.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this service is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
char   dll_name[SIZE]  = "dllpath/dllname.so";
char   func_name[SIZE] = "strIDHandler";
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
      oima_l = GXLHXEC_MIN_OIMA_SIZE;

 gxluInitOSRG(oima_p, oima_l,
         0,
         (void *)handler_parms,
         &rc, &rsn);
    }  /* oima malloc succeeded   */

/* Now set a StringID handler that will be used to  */
/* create StringIDs when OSRs are generated.        */

if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator initialized */
    gxluSetStrIDHandler (oima_p,
               dll_name, func_name,
               &rc, &rsn);

    if (rc == XRC_SUCCESS)
       { /* set handler succeeded   */

       <continue processing using the StringID handler>

         ...
       }  /* set handler succeeded   */

      ...
      } /* generator initialized */
```

Usage notes

This handler differs from the other handlers and resolvers provided to the OSR generator in that it must be written in C. Both the validating z/OS XML parser and the OSR generator allow the caller to specify a StringID handler, and by implementing this handler as a C DLL, the same source may be used in both environments. A key difference is that this handler must be compiled and linked with conventional C and Language Environment capabilities for the OSR generator environment, while it must be built using Metal C for the parser.

The DLL containing the StringID handler will be loaded in order to obtain a function pointer to it. The function pointer will be kept within the OIMA until a StringID is needed during OSR generation. The DLL path must reside in one of the paths specified in the LIBPATH environment variable.

This routine may be called more than once during an OSR generation instance to change the StringID handler that the generator uses.

gxluSetEntityResolver — specify the entity resolver for OSR generation

Description

This utility allows the caller to specify an entity resolver to the OSR generator. This resolver must be written in Java.

Performance Implications

There are no performance implications.

Syntax

```
int gxluSetEntityResolver(void * oima_p,
                   char * class_name_p,
                   int *  rc_p,
                   int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

class_name_p
Supplied parameter

Type:	char *

A pointer to the NULL terminated name of a Java class that implements the XMLEntityResolver interface of the XML4J parser (see the usage notes below). This string must be in the IBM-1047 code page. A NULL string indicates that the current entity resolver should be unset, and the default resolver used during the creation of the OSR.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this service is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
char          class_name[SIZE]  = "xml/appl/handlers/EntityResolver";
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
      oima_l = GXLHXEC_MIN_OIMA_SIZE;

 gxluInitOSRG(oima_p, oima_l,
         0,
         (void *)handler_parms,
         &rc, &rsn);
    }  /* oima malloc succeeded   */

/* Now set an entity resolver that will be used during  */
/* OSR generation.                                      */

if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator initialized */
    gxluSetEntityResolver(oima_p,
                 class_name,
                 &rc, &rsn);

    if (rc == XRC_SUCCESS)
       { /* set resolver succeeded   */

       <continue processing using the entity resolver>

         ...
       }  /* set resolver succeeded   */

      ...
      } /* generator initialized */
```

Usage notes

Although this is a C interface, the entity resolver must be implemented in Java. This resolver will be provided to the XML4J parser, which is used during the OSR generation process. The resolver must implement the XMLEntityResolver interface of the Xerces Native Interface (XNI), including the return of an XMLInputSource object. See the XMLEntityResolver documentation at http://xerces.apache.org/xerces2-j/javadocs/xni/index.html.

This routine may be called more than once during an OSR generation instance to change the entity resolver that the generator uses.

gxluLoadOSR — load an OSR into the OSR generator

Description

This utility is used to load an Optimized Schema Representation into the OSR generator. Once loaded, the OSR may be processed using one of the OSR generator operations.

Performance Implications

There are no performance implications.

Syntax

```
int gxluLoadOSR(void * oima_p,
                void * osr_p,
                int osr_l,
                int *  rc_p,
                int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

osr_p
Supplied parameter

Type:	void *

A pointer to a buffer containing an OSR.

osr_l
Supplied parameter

Type:	int

The length of a buffer containing an OSR.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this service is return code (see below).

Return and Reason Codes

On return from a call to this utility, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
char          osrbuf[OSR_BUFFER_LEN];
int           osrbuf_l;
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
      oima_l = GXLHXEC_MIN_OIMA_SIZE;

 gxluInitOSRG(oima_p, oima_l,
         0,
         (void *)handler_parms,
         &rc, &rsn);
    }  /* oima malloc succeeded   */

<acquire the OSR from a persistent location like a file ...>
/* Load an OSR to be processed.    */

if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator initialized */
    gxluLoadOSR(oima_p,
               (void *)osrbuf
                osrbuf_l,
                &rc, &rsn);

    if (rc == XRC_SUCCESS)
       { /* OSR load succeeded   */

       <process the loaded OSR>

         ...
       }  /* OSR load succeeded   */

      ...
      } /* generator initialized */
```

Usage notes

Use this utility when you need to query an OSR that has already been created from one or more human-readable schemas. This is useful, for instance, when a caller needs access to a StringID table from an existing OSR. This allows the StringID table to be used by the validating parser at parse time.

gxluGenOSR — generate an Optimized Schema Representation (OSR)

Description

This utility generates an optimized representation of one or more XML schemas.

Performance Implications

There are no performance implications.

Syntax

```
unsigned int gxluGenOSR(void * oima_p,
                void ** schema_osr_p_p,
                int * rc_p,
                int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

schema_osr_p_p
Returned parameter

Type:	void **

A pointer to an area to receive the address of the optimized schema representation generated by this utility. See the usage notes below for important details about this parameter.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this utility is the length of the OSR buffer returned to the caller through the schema_osr_p parameter. If there is a problem during the generation of the OSR, the value returned will be zero. See the usage notes below for more information about this value and the OSR buffer returned.

Return and Reason Codes

Register 15 will contain the return value of this utility. The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
char          schema_uri[URI_LEN] = "file:///u/user01/myschema.xsd";
void *        osr_p;
int           osr_l;
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
      oima_l = GXLHXEC_MIN_OIMA_SIZE;

 gxluInitOSRG(oima_p, oima_l,
         0,
         (void *)handler_parms,
         &rc, &rsn);
    }  /* oima malloc succeeded   */


/* Load a schema and create an OSR from it. */

if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator initialized */
    gxluLoadSchema(oima_p,
               schema_uri,
               &rc, &rsn);

    if (rc == XRC_SUCCESS)
       { /* schema load succeeded   */

       /* Generate the OSR */
       osr_l = gxluGenOSR(oima_p,
                          &osr_p,
                          &rc, &rsn);

       if (osr_l > 0) then
        {  /* OSR generate succeeded  */

   <write the OSR out to a persistent repository>
   <like a file or a database so that it can be>
   <used later for parsing a document>

           ...
           }      /* OSR generate succeeded  */
       ...
       }  /* schema load succeeded   */

      ...
      } /* generator initialized */
```

Usage notes

This utility generates Optimized Schema Representations in a manner similar to the xsdosrg command (see Appendix E, “xsdosrg command reference,” on page 167). It provides additional flexibility and control by allowing the caller to use the following handlers to augment the default generator behavior:

StringID handler
: This handler generates and/or returns an integer identifier that serves as a handle for a given string. These strings are most often the components of qualified names that are encountered in the schema text during processing. This must be implemented as a C routine, and built for the C Language Environment. If no StringID handler is specified, then StringIDs will not be used during the generation of the OSR. All qualified names and other strings for which IDs could be used will instead be present in the OSR in their text form. The same handler may be used by the validating parser when built for the Metal C environment.

entity resolver
: A Java routine that receives control when a reference to an external entity is made from one schema to another through an include, import, or redefine XML Schema construct. It acquires the external schema from an appropriate source, and returns it to the OSR generator for further processing. If no entity resolver is specified, the default entity resolver from the XML4J parser is used.

One or both of these routines may be specified to the OSR generator through the gxluSetStringID ("gxluSetStrIDHandler — specify the StringID handler for OSR generation" on page 63) and gxluSetEntityResolver ("gxluSetEntityResolver — specify the entity resolver for OSR generation" on page 66) utilities. Once set, the generator will make use of them until they are changed to a different value.

This utility will allocate the buffer used to receive the generated OSR, and will return the length of the buffer as its return value. The maximum length of an OSR that will be returned is 2 GB. The buffer remains allocated for the duration of the OSR generator instance, and gets freed when the instance is terminated. The caller may use or copy the OSR to another location as long as the instance is active. Referencing the OSR buffer after the generator instance has been terminated may result in unpredictable results. This buffer may also be written to a permanent location, such as a z/OS UNIX file or an MVS data set, so that it can be used again at some point in the future.

gxluGenStrIDTbl — generate StringID table from an OSR

Description

This utility will extract generate and return the StringID table associated with the current OSR for this generator instance. See the usage notes for a description of how to make an OSR current.

Performance Implications

There are no performance implications.

Syntax

```
int gxluGenStrIDTbl(void * oima_p,
                GXLHXSTR ** strid_tbl_p_p,
                int * rc_p,
                int * rsn_p)
```

Parameters

oima_p
Supplied parameter

Type:	void *

A pointer to an OSR generator Instance Memory Area (OIMA).

strid_tbl_p_p
Supplied and returned parameter

Type:	GXLHXSTR **

A pointer to an area that will receive the address of a table of containing the StringIDs that are generated from the current OSR. See the usage notes below for more details.

rc_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the return code.

rsn_p
Returned parameter

Type:	int *

A pointer to an area where the utility stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return Value

The value returned by this utility is the length of the StringID table returned to the caller through the strid_tbl_p_p parameter. If StringIDs were not in use when the current OSR was originally generated, the return value will be zero, and the pointer specified by strid_tbl_p will remain unchanged. If there is a problem during the generation of the StringID table, the value returned will be -1. See the usage notes below for more information about this value, and the StringID table returned.

Return and Reason Codes

Register 15 will contain the return value of this utility (see above). The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in the header file gxlhxr.h (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <gxlhosrg.h>
#include <gxlhxec.h>

void *        oima_p;
unsigned long oima_l;
char          handler_parms[128];
char          osrbuf[OSR_BUFFER_LEN];
int           osrbuf_l;
GXLHXSTR *    strIDTbl_p;
int           strIDTbl_l;
int           osr_l;
int           rc, rsn;

if (oima_p = malloc(GXLHXEC_MIN_OIMA_SIZE))
      { /* oima malloc succeeded   */
      oima_l = GXLHXEC_MIN_OIMA_SIZE;

 gxluInitOSRG(oima_p, oima_l,
         0,
         (void *)handler_parms,
         &rc, &rsn);
    }  /* oima malloc succeeded   */

<acquire the OSR from a persistent location like a file>

/* Load the OSR to operate on. */

if ((oima_p > 0) && (rc == XRC_SUCCESS))
    {  /* generator initialized */
    gxluLoadOSR(oima_p,
               (void *)osrbuf,
                osrbuf_l,,
                &rc, &rsn);

    if (rc == XRC_SUCCESS)
       { /* OSR load succeeded   */

       /* Generate the OSR */
       strIDTbl_l = gxluGenSTRIDTbl(oima_p,
                          &strIDTbl_p,
                          &rc, &rsn);

       if (strIDTbl_l > 0) then
        { /* strID table generated   */

   <write the StringID table out to a persistent>
   <repository like a file or a database so that>
   <it can be used later when parsing a document>

           ...
           }      /* strID table generated  */
       ...
       }  /* OSR load succeeded      */

      ...
      } /* generator initialized */
```

Usage notes

The StringID table is generated from the OSR that has been made current through either a gxluGenOSR or a gxluLoadOSR request. The actual length of the StringID table is calculated during table generation, and cannot be known ahead of time. For this reason, the gxluGenStrIDTbl service will return the address and length of the generated table on success. The table remains allocated for the duration of the OSR generator instance, and gets freed when the instance is terminated. The caller may use or copy the StringID table to another location as long as the instance is active. Referencing the StringID table after the generator instance has been terminated may result in unpredictable results.

StringID tables may be generated from OSRs that were created either with or without StringIDs. If no StringIDs were used when the OSR was originally generated, this service will assign the StringID values to return in the table. Callers who wish to control the values of StringIDs must use the StringID handler interface at OSR generation time.

The format of the StringID table that the OSR generator creates is defined by the gxlhxstr.h header file. See the definition of this header file below for more details.

GXLPSYM31 (GXLPSYM64) — StringID handler

Description

This handler accepts an input string and performs a lookup for its corresponding symbol, which is identical to the string itself. If the symbol has been located, the exit returns the StringID associated with the symbol. If the string does not have a defined symbol, a symbol is created for the string and a StringID is assigned to it.

Performance Implications

There are no performance implications.

Syntax

```
int gxlpSym31(void ** sys_svc_p,
              char * string_p,
              int string_l,
              unsigned int * string_id_p,
              int ccsid,
              int * handler_diag_p,
              int * rc_p)
```

Parameters

sys_svc_p

Supplied parameter

Type:	void **

A pointer to the system service parameter that was passed to the z/OS XML OSR generator at initialization time.

string_p

Supplied parameter

Type:	char *

The string to return an ID for. The length of the string is variable, and is specified by the *string_l* parameter.

string_l

Supplied parameter

Type:	int

An integer containing the length of the string pointed to by the string parameter.

string_id_p

Returned parameter

Type:	unsigned int *

A pointer to an integer where the handler stores the numeric identifier for the string. The range of valid values is 1 to 2 GB - 1.

ccsid

Supplied parameter

Type:	int

The Coded Character Set IDentifier (CCSID) that identifies the character set of the string. The z/OS XML parser will provide the same CCSID in this parameter that the caller of the parser specified at parser initialization time.

handler_diag_p

Returned parameter

Type: int *

A pointer to an integer where the handler can store any diagnostic information (usually a reason code). This will be stored in the diagnostic area and made available on the gxluControlOSRG call.

rc_p
Returned parameter

Type: int *

A pointer to an integer where the handler can store a return code. A return code value of zero means success; any nonzero return code indicates failure.

Return Codes

The z/OS XML OSR generator uses the convention that the handler will provide a return code value of zero when successful. Any nonzero value indicates failure. If a nonzero return code is provided by the exit, the z/OS XML OSR generator saves it in the extended diagnostic area so that the caller of the parser has access to it by calling gxluControlOSRG.

Example

None.

Default Implementation

There is no default implementation. If this handler is not specified by the caller, StringIDs are not used by the z/OS XML OSR Generator.

Chapter 7. z/OS XML parser API: Assembler

How to invoke the z/OS XML System Services assembler API

This section provides information on how to invoke the z/OS XML System Services assembler API.

Callers written in assembler can invoke the z/OS XML System Services assembler API by binding the z/OS XML parser's callable service stubs to their module. The callable service stubs can be found in SYS1.CSSLIB. Alternatively, the addresses of the APIs can be obtained from system control blocks. The following is a list of offsets for the callable services first and second tables (all offsets are in hex):

1. +10 — Pointer to CVT (field FLCCVT in IHAPSA)
2. +220 — Pointer to the callable services first table (field CVTCSRT in CVT)
3. +48 — Pointer to the z/OS XML parser callable services second table (entry 19)

   Note: Prior to z/OS V1R7, this field will point to the address of an undefined callable service. In z/OS V1R7 and later releases, this field is zero until the z/OS XML parser initialization routine fills it in. To avoid calling z/OS XML System Services when it is not present, the caller first needs to verify that it is running on V1R7 or later, and that this field in the callable services first table is non-zero.
4. +nn — The offset for each callable service in hex is listed below.

The following stubs are provided for 31- and 64-bit mode callers:

Table 18. Caller stubs and associated offsets

Stub	Second Table offset (hex)
GXL1INI — 31-bit parser initialization	10
GXL1PRS — 31-bit parse	14
GXL1TRM — 31-bit parser termination	18
GXL1CTL — 31-bit parser control operation	1C
GXL1QXD — 31-bit query XML document	20
GXL1LOD — 31-bit load a function	
GXL4INI — 64-bit parser initialization	28
GXL4PRS — 64-bit parse	30
GXL4TRM — 64-bit parser termination	38
GXL4CTL — 64-bit parser control operation	40
GXL4QXD — 64-bit query XML document	48
GXL4LOD — 64-bit load a function	

Note: The 64-bit stubs are defined with 8 byte pointers.

The following assembler code is an example of how to call a z/OS XML parser service. The example assumes the caller uses the CVT field names instead of hard coding those offsets.

```
LLGT 15,CVTPTR         R15L -> CVT, R15H = 0
L    15,CVTCSRT-CVT(15) Get the CSRTABLE
L    15,72(15)         Get CSR slot 19 (zero based) for XML parser
L    15,16(15)         Get address of GXL1INI from XML second table.
BALR 14,15             Branch to XML service.
```

z/OS XML parser Assembler API

This section lists the assembler callable services interface used for the z/OS XML parser. The following rules apply to some or all of the callable services listed below:

- The 31- and 64-bit versions of the services were designed to work independently of one another. For example, the following sequence of calls would not work: GXL1INI (31-bit service) followed by GXL4PRS (64-bit service).
- The 31- and 64-bit versions of the services are documented together with any differences for 64- bit shown in parenthesis, after its corresponding 31-bit description.
- In AMODE 31, all address and length parameters of the z/OS XML parser API are 4 bytes long. In AMODE 64, these fields are 8 bytes long.
- In AMODE 31, the parsed data stream produced by the z/OS XML parser contains length fields that are all 31 bits (4 bytes) long. In AMODE 64, the field in the buffer header representing the length of the output buffer used is 64-bits (8 bytes) long, while all record length fields in the data stream are 31-bit (4 byte) values.

API entry points

The z/OS XML parser API contains 5 entry points for each addressing mode (AMODE) type (31- or 64-bit):

- GXL1CTL (GXL4CTL) — perform a parser control operation
- GXL1INI (GXL4INI) — initialize a parse instance
- GXL1PRS (GXL4PRS) — parse an input stream
- GXL1QXD (GXL4QXD) — query an XML document
- GXL1TRM (GXL4TRM) — terminate a parse instance
- GXL1LOD (GXL4LOD) — load a function

Common register conventions

The following sections describe common register conventions used for all of the z/OS XML parser's callable services.

Input registers

When a caller invokes the z/OS XML parser, these registers have the following meaning:

Table 19. Input register conventions

Register	Contents
1	Address of a standard parameter list containing 31 (64) bit addresses.
14	Return address.

Output registers

When the z/OS XML parser returns to the caller, these registers have the following meaning:

Table 20. Output register conventions

Register	Contents
0-1	Unpredictable
2-13	Unchanged

Table 20. Output register conventions (continued)

Register	Contents
14	Unpredictable
15	Return code (return code is also a parameter)

Table 21. Output access register conventions

Access Register	Contents
0-1	Unpredictable
2-13	Unchanged
14-15	Unpredictable

Environmental requirements

The following are environmental requirements for the caller of any z/OS XML parser service:

Minimum authorization
any state and any PSW key

Dispatchable unit mode
Task or SRB

Cross memory mode
PASN=HASN=SASN or PASN^=HASN^=SASN

AMODE
31-bit (64-bit)

ASC mode
primary

Interrupt status
enabled for I/O and external interrupts

Locks no locks held

Control parameters
Control parameters and all data areas the parameter list points to must be addressable from the current primary address space.

Using the recovery routine

z/OS XML provides an ARR recovery routine to assist with problem determination and diagnostics. This is an optional routine and can be turned on and off as desired. See“ARR recovery routine” on page 114 for more information.

Restriction: When running in either SRB mode or under an existing FRR routine, the ARR recovery routine cannot be used.

GXL1CTL (GXL4CTL) — perform a parser control function

Description

This is a general purpose service which provides control functions for interacting with the z/OS XML parser. The function performed is selected by setting the *ctl_option* parameter using the constants defined in GXLYXEC. These functions include:

XEC_CTL_FIN
The caller has finished parsing the document. Reset the necessary structures so that the PIMA can be reused on a subsequent parse, and return any useful information about the current parse.

XEC_CTL_FEAT
The caller wants to change the feature flags. A XEC_CTL_FIN function will be done implicitly.

Note: Two feature flags, JST_OWNS_STORAGE and RECOVERY, are not supported on GXL1CTL (GXL4CTL). Make sure that these feature flags are turned to the ″off″ state before calling GXL1CTL (GXL4CTL) to set the feature flags.

XEC_CTL_LOAD_OSR
The caller wants to load and use an Optimized Schema Representation (OSR) for a validating parse.

Note: finish and reset processing is performed by all operations available through this control service.

Performance Implications

The finish/reset function allows the caller to re-initialize the PIMA to make it ready to handle a new XML document. This re-initialization path enables the z/OS XML parser to preserve its existing symbol table, and avoid other initialization pathlength that's performed by calling GXL1INI (GXL4INI).

Syntax

```
call gxl1ctl,(PIMA,
            ctl_option,
            ctl_data,
            return_code,
            reason_code)
```

Parameters

PIMA
Supplied parameter

Type:	Character string
Length:	Variable

The name of the Parse Instance Memory Area (PIMA which has been previously initialized with a call to GXL1INI (GXL4INI)).

ctl_option
Supplied parameter

Type:	Integer
Length:	Fullword

The name of a fullword that contains an integer value representing one of the following functions:

XEC_CTL_FIN
This indicates that the caller wants to end the current parse at the current position in the XML document. The PIMA is re-initialized to allow it to be used on a new parse request. To free up all resources associated with the parse instance, the caller should use GXL1TRM (GXL4TRM).

XEC_CTL_FEAT
This indicates that the caller wishes to re-initialize the z/OS XML parser, as with the reset / finish function as above and in addition, reset the feature flags used during the parse.

Note: The following feature flags are not supported on GXL1CTL (GXL4CTL):

- XEC_FEAT_JST_OWNS_STORAGE
- XEC_FEAT_RECOVERY
- XEC_FEAT_VALIDATE

Make sure that these feature flags are turned to the OFF state before calling GXL1CTL (GXL4CTL) to set the feature flags. If these features need to be changed (for example, if switching between validating and non-validing parses), the parse instance must be terminated and re-initialized with the required feature settings.

XEC_CTL_LOAD_OSR
This indicates that the caller wants to load and use a given Optimized Schema Representation (OSR) during a validating parse. When this operation in chosen, the ctl_data parameter must also be utilized to specify the required schema. This operation will also cause the z/OS XML parser to perform reset and finish processing.

ctl_data
Supplied and returned parameter

Type:	Address
Length:	Fullword (Doubleword)

The name of a fullword (doubleword) that contains the address of an area as defined by ctl_option:

XEC_CTL_FIN
This parameter must contain the address of a fullword (doubleword) where the service will store the address of the diagnostic area, which is mapped by macro GXLYXD. This provides additional information that can be used to debug problems in data passed to the z/OS XML parser. The diagnostic area resides within the PIMA, and will be overlaid on the next call to the z/OS XML parser.

XEC_CTL_FEAT
This parameter must contain the address of a fullword (doubleword), which is mapped by macro GXLYXFT. See "gxlhxft.h (GXLYXFT) - mapping of the control feature input output area" on page 174 for more information on this macro. A simplified form of the macro is provided below:

```
XFT_FEAT_FLAG
XFT1_XD_PTR
```

The XFT_FEAT_FLAGS parameter is an input parameter to the API and contains the value of feature flags to be used in the subsequent parse. It is defined as follows:

XEC_FEAT_STRIP_COMMENTS
This effectively strips comments from the document by not returning any comments in the parsed data stream. Default: off.

XEC_FEAT_TOKENIZE_WHITESPACE
This sets the default token value for white space preceding markup in the root element to an explicit white space value. Default: off – white space is returned as character data.

XEC_FEAT_CDATA_AS_CHARDATA
This returns CDATA in records with a CHARDATA token type. The content of these records may contain text that would normally have to be escaped to avoid being handled as markup. Default: off.

XEC_FEAT_SOURCE_OFFSETS
This feature is used to include records in the parsed data stream which contain offsets to the corresponding structures in the input document. Default: off.

XEC_FEAT_FULL_END
This feature is used to expand the end tags to include the local name, prefix and URI corresponding to the qname on the end tag. Default: off.

If none of the features are required, passing a fullword field containing zero will turn them all off. Do not construct a parameter list with a zero pointer in it.

The XFT1_XD_PTR must contain the address of a fullword (doubleword) where the service will store the address of the diagnostic area, which is mapped by macro GXLYXD. This provides additional information that can be used to debug problems in data passed to the z/OS XML parser. The diagnostic area resides within the PIMA, and will be overlaid on the next call to the z/OS XML parser.

XEC_CTL_LOAD_OSR

The schema specified must be in the Optimized Schema Representation (OSR) form created through one of the interfaces to the OSR generation utility. Once a schema has been loaded, it remains in use for all validating parse requests until a different schema is provided by calling this service again with a different schema OSR.

This parameter must contain the address of an area containing information about the OSR to load. This area is mapped by macro GXLYXOSR. See "gxlhxosr.h (GXLYXOSR) - mapping of the OSR control area" on page 174 for more information on this macro. Also, see the usage notes below for details about how the information in this area is used.

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the return code.

reason_code
Returned parameter

Type:	Integer

Length: Fullword

The name of a fullword where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code is 0 (XRC_SUCCESS). Return and reason codes are defined in macro GXLYXR, and are dependent on the control function specified by the caller. For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

For an AMODE 31 example using this callable service, see “GXL1CTL example” on page 182. For an AMODE 64 example using this callable service, see “GXL4CTL example” on page 188.

Usage notes

The purpose of the finish/reset function of the GXL1CTL (GXL4CTL) service is to perform the following:

- Reset the necessary structures and fields within the PIMA to effect a re-initialization so that it can be reused without the overhead of full initialization.
- Allow the z/OS XML parser to return extended diagnostic information to the caller in the event of a failure. This allows the caller to identify certain problems that can be corrected.
- The ″finish and reset″ operation can be thought of as the most basic control operation that is a functional subset of all control operations. It resets the state of the parser to the original state immediately after the parse instance was first initialized. This state includes the feature flags. If the caller initializes a parse instance, then changes the feature settings with a feature control operation, and still later performs a ″finish and reset″ control operation, the feature flags will revert back to those settings at the time the parse instance was originally initialized. If the caller wishes to retain the current feature settings during a parser reset, they should simply perform another feature control operation with the current feature set.
- The OSR load operation allows the caller to specify an OSR for the parser to use, and to bind a handle to associate with to it. The GXLYXOSR macro provides the interface for passing information to the parser about the OSR. See Appendix F, “C/C++ header files and assembler macros,” on page 171 for more details about how it is used. As mentioned above, ″finish and reset″ processing will occur as a part of this load operation. However, the reset will occur through a feature control operation, using the current feature set. In this way, the current feature flags for the parse instance are not altered by the OSR load control operation.

GXL1INI (GXL4INI) — initialize a parse instance

Description

The GXL1INI (GXL4INI) callable service initializes the PIMA and records the addresses of the caller's system service routines (if any). The PIMA storage is divided into the areas that will be used by the z/OS XML parser to process the input buffer and produce the parsed data stream.

Performance Implications

The initialization of structures used by the z/OS XML parser in the PIMA is only done once per parse and is therefore unlikely to affect performance. The caller may choose to reuse the PIMA after each parse to eliminate the overhead of storage allocation and the page faults that occur when referencing new storage. In this case, a control operation is required to reset the necessary fields in the PIMA before parsing can continue.

Syntax

```
call gxl1ini,(PIMA,
            PIMA_len,
            ccsid,
            feature_flags,
            sys_svc_vector,
            sys_svc_parm,
            return_code,
            reason_code)
```

Parameters

PIMA

Supplied parameter

Type:	Character string
Length:	determined by the PIMA_len parameter

The name of the Parse Instance Memory Area (PIMA). The PIMA must be aligned on a doubleword boundary, otherwise, results are unpredictable. See the "Usage notes" on page 88 below for additional details on the use of this area.

PIMA_len

Supplied parameter

Type:	Integer
Length:	Fullword (Doubleword)

| The name of an area containing the length of the Parse Instance Memory Area. This service validates the length of this area against a minimum length value. The minimum length of the PIMA depends on whether or not validation will be performed during the parse. This minimum length value can be found in:

- XEC_NVPARSE_MIN_PIMA_SIZE (non-validating parse)
- XEC_VPARSE_MIN_PIMA_SIZE (validating parse)

ccsid

Supplied parameter

Type:	Integer
Length:	Fullword

The Coded Character Set IDentifier (CCSID) that identifies the document's character set. The CCSID value in this parameter will override any character set or encoding information contained in the XML declaration of the document. A set of CCSID constants for supported encodings has been declared in GXLYXEC. See Appendix L, "Supported encodings," on page 253 for a full list of supported encodings.

feature_flags
Supplied parameter

Type:	Integer
Length:	Fullword

The name of a fullword that contains an integer value representing one or more of the following z/OS XML parser features. OR these flags together as needed to enable features. Choose any of the following:

- **XEC_FEAT_STRIP_COMMENTS** - effectively strip comments from the document by not returning any comments in the parsed data stream.
- **XEC_FEAT_TOKENIZE_WHITESPACE** - set the default token value for white space preceeding markup within the context of the root element to an explicit white space value. Use this value in conjunction with the special xml:space attribute to determine how such white space gets classified.
- **XEC_FEAT_CDATA_AS_CHARDATA** - return CDATA in records with a CHARDATA token type. The content of these records may contain text that would normally have to be escaped to avoid being handled as markup.
- **XEC_FEAT_JST_OWNS_STORAGE** - allocate storage as Job Step Task (JST) related instead of task related. See the "Usage notes" on page 88 below for more information.
- **XEC_FEAT_RECOVERY** - this turns on the recovery routine.

  Notes: The following only applies when the feature flag is ON:
  - If running in SRB mode, an error message will be returned to the caller.
  - If a parse request is made in SRB mode, the parse will fail.
  - If there is an FRR, an error message will be returned to the caller during the parse step.
- **XEC_FEAT_SOURCE_OFFSETS** - this includes records in the parsed data stream which contain offsets to the corresponding structures in the input document.
- **XEC_FEAT_FULL_END** - this expands the end tags to include the local name, prefix and URI corresponding to the qname on the end tag.
- **XEC_FEAT_VALIDATE** - this initializes a parse instance that allows for validation during parsing. See the usage notes below for details on validation.

Note: By using the values of off (zero), W3C XML compliant output is generated. Turning on options XEC_FEAT_STRIP_COMMENTS, XEC_FEAT_TOKENIZE_WHITESPACE and XEC_FEAT_CDATA_AS_CHARDATA will cause the output to vary from standard compliance.

If none of the features are required, pass the name of a fullword field containing zero. Do not construct a parameter list with a zero pointer in it.

sys_svc_vector
Supplied parameter

Type:	Structure

Length:	Variable

The name of a structure containing a count of entries that follow and then a list of 31 (64) bit pointers to system service routines. Specify the name of a word containing 0 if no services are provided. See the Chapter 8, “z/OS XML System Services exit interface” chapter for more details.

sys_svc_parm
Supplied parameter

Type:	Address
Length:	Fullword (Doubleword)

The name of a parameter which is passed to all system service exits. This provides for communication between the z/OS XML parser caller and its exit routines. Specify the name of a location containing 0 if no parameter is required for communication.

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the return code.

reason_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in macro GXLYXR. For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

For an AMODE 31 example using this callable service, see “GXL1INI example” on page 183. For an AMODE 64 example using this callable service, see “GXL4INI example” on page 189.

Usage notes

- The z/OS XML parser creates a variety of control blocks, tables, stacks, and other structures in the Parse Instance Memory Area. The caller must provide an area that is at least as large as constant **XEC_MIN_PIMA_SIZE**. In the event that this area is not large enough to parse the input document, the z/OS XML parser will allocate additional memory using either the default memory allocation mechanism or the memory allocation exit that the caller has provided.
- When the PIMA is reused for subsequent parses, the same features, ccsid and service exits will apply. If any of these values need to change, you should

terminate the parse instance (call GXL1TRM (GXL4TRM)) and call GXL1INI (GXL4INI) again with the options you require.

- When the **XEC_FEAT_TOKENIZE_WHITESPACE** feature is set, the default classification for white space that precedes markup within the context of the root element will be **XEC_TOK_WHITESPACE**. This token type is returned if either the white space being parsed does not have an xml:space context, or if the xml:space setting is 'default'. When the tokenize white space feature is not enabled, or if the white space does not precede markup, this white space will be returned in the parsed data stream containing character data with a token type of **XEC_TOK_CHAR_DATA**.
- The **XEC_FEAT_JST_OWNS_STORAGE** feature only applies to callers running in non-cross memory task mode who take the option of allowing the z/OS XML parser to allocate additional storage as needed. This feature should be specified when PIMAs are used on multiple tasks in order to prevent task termination from causing storage extents to be freed before the z/OS XML parser is done using them.
- Before requesting the initialization of a validating parse instance, the validation function must be loaded – either through one of the methods that the system provides, or by the z/OS XML load service. Failure to do so will result in an error indicating that the function is not available. See the description of "GXL1LOD (GXL4LOD) — load a z/OS XML function" on page 98 for more information.
- Be sure that the size of the PIMA provided is large enough for the XML processing function, either validating or non-validating parse, that will be performed. Also, make sure that there is an appropriate minimum PIMA size constant defined for each in GXLYXEC.
- The performance of a validating parse will be best when the parsed document is in the UTF-8 encoding. The other encodings supported by z/OS XML System Services are also supported during a validating parse, but there is significant additional overhead that will impact performance.

GXL1PRS (GXL4PRS) — parse a buffer of XML text

Description

The GXL1PRS callable service parses a buffer of XML text and places the result in an output buffer.

Performance Implications

Ideal performance will be obtained when the PIMA is sufficiently large to contain all the needed data structures, and the input and output buffers are large enough to process the entire XML document. During the parsing process, the z/OS XML parser constructs persistent information in the PIMA that can be reused within a parse instance. If the caller is going to process multiple documents that contain similar sets of symbols (namespaces and local element and attribute names in particular), then reusing the PIMA will improve performance during the processing of subsequent documents. If this behavior is not required, the PIMA should be cleaned up by calling GXL1TRM (GXL4TRM) and reinitialized by calling GXL1INI (GXL4INI) before using the PIMA for another parse request.

Syntax

```
call gxl1prs,(PIMA,
             option_flags,
             input_buffer_addr,
             input_buffer_bytes_left,
             output_buffer_addr,
             output_buffer_bytes_left,
             return_code,
             reason_code)
```

Parameters

PIMA

Supplied parameter

Type:	Character string
Length:	Variable

The name of the Parse Instance Memory Area (PIMA which has been previously initialized with a call to GXL1INI (GXL4INI)).

option_flags

Supplied parameter

Type:	Integer
Length:	Fullword

Specify a word of zeroes for this parameter. In the future, this field will allow options to be compatibly added to the service.

input_buffer_addr

Supplied and returned parameter

Type:	Address
Length:	Fullword (Doubleword)

The name of a fullword (doubleword) that contains the address of the buffer with the XML text to parse. The z/OS XML parser updates this parameter to provide important return information when control returns to the caller. See the "Usage notes" on page 92 below for details.

input_buffer_bytes_left
Supplied and returned parameter

Type:	Integer
Length:	Fullword (Doubleword)

The name of a fullword (doubleword) that contains the number of bytes in the input buffer that have not yet been processed. The z/OS XML parser updates this parameter to provide important return information when control returns to the caller. See the “Usage notes” on page 92 for details.

output_buffer_addr
Supplied and returned parameter

Type:	Address
Length:	Fullword (Doubleword)

The name of a fullword (doubleword) that contains the address of the buffer where the z/OS XML parser should place the parsed data stream. The z/OS XML parser updates this parameter to provide important return information when control returns to the caller. See the “Usage notes” on page 92 for details.

output_buffer_bytes_left
Supplied and returned parameter

Type:	Integer
Length:	Fullword (Doubleword)

The name of a fullword (doubleword) that contains the number of available bytes in the output buffer. When the z/OS XML parser returns control to the caller, this parameter will be updated to indicate the number of unused bytes in the output buffer. This buffer must always contain at least a minimum number of bytes as defined by the **XEC_MIN_OUTBUF_SIZE** constant, declared in macro GXLYXEC. This service will validate the length of this area against this minimum length value.

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the return code.

reason_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in macro GXLYXR. For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

For an AMODE 31 example using this callable service, see “GXL1PRS example” on page 185. For an AMODE 64 example using this callable service, see “GXL4PRS example” on page 191.

Usage notes

- When the z/OS XML parser returns successfully to the caller, the input and output buffer addresses will be updated to point to the byte after the last byte successfully processed. The *input_buffer_bytes_left* and *output_buffer_bytes_left* parameters will also be updated to indicate the number of bytes remaining in their respective buffers. In the event of an error caused by a problem with the document being parsed, the input buffer address will point to the byte of the input stream where the problem was detected, and the associated bytesleft value will indicate the same position in the buffer. An error record will be written to the parsed data stream indicating the nature of the problem, and the output buffer address and bytesleft fields will point to the next available byte, as in the success case. See Chapter 4, “Parsing XML documents,” on page 11 for more information about how input and output buffers are managed between the caller and z/OS XML parser.
- In cases where parsing terminates because of an error, the z/OS XML parser will often have partially processed an item from the input document before returning to the caller. The caller has the option of retrieving the address of the diagnostic area using the GXL1CTL (GXL4CTL) service. The **XD_LastRC/XD_LastRsn** return/reason code combination will contain an indication of the item being parsed. Retrieving the reason code in this manner is an example of the indirect method for obtaining a specific reason code.
- The z/OS XML parser will always check that the output buffer length passed to it is greater than the required minimum (**XEC_MIN_OUTBUF_SIZE**). If this minimum length requirement is not met, the z/OS XML parser will return with a return/reason code of **XRC_FAILURE/XRSN_BUFFER_OUTBUF_SMALL**. Output buffer spanning will only occur if the caller meets the minimum output buffer length requirement when the z/OS XML parser is invoked. Once parsing begins, and the buffer info record has been written to the output buffer, buffer spanning is enabled. The caller will then receive an end-of-output-buffer indication when the end of the output buffer is reached. In addition, many non-splittable records will be larger than the minimum output buffer size. If there is not enough space in the output buffer for the first record, then XRC_FAILURE/XRSN_BUFFER_OUTBUF_SMALL will be returned. Therefore, it’s recommended that the output buffer sizes should be large enough to fit the largest record that is expected to be encountered.

GXL1QXD (GXL4QXD) — query an XML document

Description

This service allows a caller to obtain the XML characteristics of a document. The XML characteristics are either the default values, the values contained in an XML declaration or a combination of both.

Performance Implications

There are no performance implications.

Syntax

```
call gxl1qxd,(work_area,
             work_area_length,
             input_buffer,
             input_buffer_length,
             return_data,
             return_code,
             reason_code)
```

Parameters

work_area
Supplied parameter

Type:	Character string
Length:	Variable

The name of a work area. The work area must be aligned on a doubleword boundary. If not on a doubleword boundary, results are unpredictable. See the "Usage notes" on page 94 below for additional details on the use of this area.

work_area_len
Supplied parameter

Type:	Integer
Length:	Fullword (Doubleword)

The name of an area containing the length of the work area. The minimum length of this area is declared as a constant **XEC_MIN_QXDWORK_SIZE** in macro GXLYXEC. This service validates the length of this area against this minimum length value.

input_buffer
Supplied parameter

Type:	Character string
Length:	Variable

The name of an input buffer containing the beginning of the XML document to process. See the "Usage notes" on page 94 below for details.

input_buffer_length
Supplied parameter

Type:	Integer
Length:	Fullword (Doubleword)

The name of an area containing the length of the input buffer.

return_data
Returned parameter

Type:	Address
Length:	Fullword (Doubleword)

The name of a fullword (doubleword) where the service will return the address of the data which describes the XML document characteristics. This return information will contain values that are either extracted from the XML declaration or defaulted according to the XML standard. This return area is mapped by macro GXLYQXD (see “gxlhqxd.h (GXLYQXD) - mapping of the output from the query XML declaration service” on page 172), and is located within the work area specified by the work_area parameter. The caller must not free the work_area until it is done referencing the data returned from this service.

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the return code.

reason_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in macro GXLYXR (see “gxlhxr.h (GXLYXR) - defines the return codes and reason codes” on page 173). For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

Usage notes

- The input buffer passed to this service must contain the beginning of the XML document to process. It will look for any XML declaration that is present and extract the version, encoding, and standalone value that are present. In the event that the document does not contain an XML declaration, or a given value is missing from the declaration, this service will return an appropriate default, as specified by the XML standard. On success, the return data address for this service will contain a pointer into the work area where the return data has been collected.
- Unlike the GXL1PRS (GXL4PRS) or GXL1CTL (GXL4CTL) services that must be performed within a parse instance, this service does not require any of the internal resources that the z/OS XML parser creates in the PIMA during

initialization. It does not advance the input pointer or modify the state of the parse in any way. It is a simple standalone service that allows a caller to query important information about the document before establishing a parse instance and performing the parse.

- Buffer spanning is not supported by this service, as it is by GXL1PRS (GXL4PRS). If either the input buffer or the work area are too small, this service will terminate with an appropriate return/reason code.
- This service is useful for checking to see if a conversion to one of the supported encodings is required before parsing the document.
- Encoding names supported include the IANA recommended names which have corresponding IBM CCSID values.

GXL1TRM (GXL4TRM) — terminate a parse instance

Description

The GXL1TRM callable service releases all resources obtained (including storage) by the z/OS XML parser and resets the PIMA so that it can be re-initialized or freed.

Performance Implications

There are no performance implications.

Syntax

```
call gxl1trm,(PIMA,
            return_code,
            reason_code)
```

Parameters

PIMA

Supplied parameter

Type:	Character string
Length:	Variable

The name of the Parse Instance Memory Area (PIMA which has been previously initialized with a call to GXL1INI (GXL4INI)).

return_code

Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the return code.

reason_code

Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both also set as output parameters. The value of the reason code is undefined when the return code has no associated reasons. Return and reason codes are defined in macro GXLYXR. For reason code descriptions, also see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

For an AMODE 31 example using this callable service, see “GXL1TRM example” on page 186. For an AMODE 64 example using this callable service, see “GXL4TRM example” on page 192.

Usage notes

Termination can be requested any time the caller gets control back from the z/OS XML parser. This service does not free the Parse Instance Memory Area (PIMA) as a part of termination. If the caller's recovery gets control while a parse is still in progress, the caller should invoke this termination service to clean up resources.

GXL1LOD (GXL4LOD) — load a z/OS XML function

Description

Load a module that implements a z/OS XML function into storage.

Performance Implications

None.

Syntax

```
call gxl1lod(function_code,
             function_data,
             return_code,
             reason_code)
```

Parameters

function_code

Supplied parameter

Type:	Integer
Length:	Fullword

This parameter identifies the z/OS XML function to load. It is the name of a fullword that contains an integer value representing one of the following functions:

XEC_LOD_VPARSE
The validating parse function.

See the GXLYXEC macro for the list of function code constants.

function_data

Supplied parameter

Type:	Address
Length:	Fullword (doubleword)

Specify a word of zeroes for this parameter.

return_code

Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the return code.

reason_code

Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the service stores the reason code. The reason code is only relevant if the return code is not **XRC_SUCCESS**.

All parameters in the parameter list are required.

Return and Reason Codes

On return from a call to this service, register 15 will contain the return code. The return and reason code are both set as output parameters. The value of the reason code is undefined when the return code is 0 (XRC_SUCCESS). Return and reason codes are defined in macro GXLYXR, and are dependent on the control function specified by the caller. For reason code descriptions, also see Appendix B, "Reason Codes Listed by Value," on page 119.

Usage notes

This load step is not required when performing a non-validating parse. This operation is only required when using the validating parser. The caller does have the option of loading the load module for the specified function without using this service - either through the z/OS LOAD macro, or by putting it in LPA or the extended LPA. Both the LOAD macro and calls to this service are not allowed when running in an SRB. The use of either interface must be performed in the task before entering SRB mode.

If the required z/OS XML function is made available, either by LOADing the executable load module for it or putting the load module in LPA, this service is not required. Documentation on the LOAD macro can be found in z/OS MVS Programming: Assembler Services Reference, Volume 2, and information on how to load modules into LPA can be found in z/OS MVS Initialization and Tuning Guide.

The load modules associated with each function are as follows:

Table 22. Load modules

Function code	Function performed	Load module name
XEC_LOD_VPARSE	validating parser function	GXLIMODV

Chapter 8. z/OS XML System Services exit interface

The system services exit interface defines a series of exits that give the original caller of the GXL1PRS (GXL4PRS) service control over the way the z/OS XML parser acquires/releases resources, and over certain parser operations. The interface is implemented as a vector of addresses to routines that perform these operations. The first word in the vector is a count of the number of addresses which follow — both NULL addresses indicating that a specific exit is not present, and non–NULL addresses. If this count is zero, then the z/OS XML parser will use default services. Similarly, an entry in the system service vector may be left NULL, and the default service that corresponds to that entry will be used. For the storage allocation and deallocation exits, either both or neither exit must be specified. The addresses of the routines are 4 bytes when in AMODE 31 and 8 bytes when in AMODE 64. The mapping macro GXLYXSV (see “gxlhxsv.h (GXLYXSV) - mapping of the system service vector” on page 174) is available to help set up this structure.

Exit functions

The system services exit interface contains exits to perform the following functions:

- Allocate memory
- Free memory
- String identifier service — this is used to create a unique 4 byte numerical value (StringID) that corresponds to a string parsed from the document. This exit allows the caller to control the individual StringID values that the z/OS XML parser uses and serves as an efficient mechanism to communicate these values between caller and parser. If no StringID service is specified, StringIDs are not exploited by the z/OS XML parser and the parsed data stream will contain only length/value pairs for all parsed strings.

These exits are all passed the address of a system service work area. This work area is storage that was obtained by the caller and can be used to store any information which may make communication between the caller and the exits easier.

Common register conventions

The following are common register conventions for all of the system service interface exits:

Input registers

When the z/OS XML parser invokes an exit, these registers have the following meaning:

Table 23. System services input register conventions

Register	Contents
1	Address of a standard parameter list containing 31 (64) bit addresses.
13	Address of a 72 (144) byte save area.
14	Return address

Table 24. System services input access register conventions

Access Register	Contents
0-15	Unpredictable

Output registers

When an exit returns to the z/OS XML parser, these registers have the following meaning:

Table 25. System services output register conventions

Register	Contents
0-1	Unpredictable
2-13	Unpredictable
14	Return address
15	Unpredictable

Table 26. System services output access register conventions

Register	Contents
0-15	Unpredictable

The z/OS XML parser saves all general purpose and access registers prior to calling the user exit. The user exit must simply return to the address in register 14. The save area provided can be used for any needs of the exit.

Environmental requirements

The system services exit interface exits are called in the same environment in which the z/OS XML parser was invoked. This means the following:

Minimum authorization
any state and any PSW key

Dispatchable unit mode
Task or SRB

Cross memory mode
Any PASN, any HASN, any SASN

AMODE
31-bit (64-bit)

ASC mode
primary

Interrupt status
enabled for I/O and external interrupts

Locks no locks held

Control parameters
Control parameters and all data areas the parameter list points to are addressable from the current primary address space.

Restrictions

These exit routines must not call any of the services provided in the z/OS XML parser API, either directly or indirectly.

These exit routines are required to use linkage OS. As a result, they will need to be written in assembler and not C or C++.

The two storage exits, “GXLGST31 (GXLGST64) — get memory” on page 104 and “GXLFST31 (GXLFST64) — free memory” on page 107, must be called together. They cannot be called independently of one another.

Although the actual name of the entry points to each of these exit services may be anything the caller wishes, the z/OS XML parser will call these services as if they had the interfaces listed below.

GXLGST31 (GXLGST64) — get memory

Description

This service allocates an area of memory of the size requested by the z/OS XML parser. The z/OS XML parser requests memory in large quantities and manages sub-allocations of this memory within the parser.

Performance Implications

There are no performance implications.

Syntax

```
call gxlgst31,(sys_svc_parm,
              memory_addr,
              memory_len,
              exit_diag_code,
              return_code,
              reason_code)
```

Parameters

sys_svc_parm
Supplied parameter

Type:	Address
Length:	Fullword (Doubleword)

The address of the system service parameter (or zero) that was passed to the z/OS XML parser at initialization time.

memory_addr
Returned parameter

Type:	Address
Length:	Fullword (Doubleword)

The address of a fullword (doubleword) where the memory allocation exit should store the address of the allocated memory. If the caller wants to terminate the parse, then it should set a nonzero return code.

memory_len
Supplied and Returned parameter

Type:	Integer
Length:	Fullword (Doubleword)

A fullword that contains the length of the memory area requested by the z/OS XML parser. The exit is allowed to return an area of greater size and set this parameter to the length returned.

exit_diag_code
Returned parameter

Type:	Integer
Length:	Fullword (Fullword)

The name of a fullword where the exit stores any diagnostic information (usually a reason code). This is stored in the diagnostic area and made available on the GXL1CTL (GXL4CTL) call.

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the exit service stores the return code.

reason_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the exit service stores the reason code.

Return and Reason Codes

The z/OS XML parser uses the convention that the exit will provide a return code value of zero when successful. Any nonzero value indicates failure. If a nonzero return code is provided by the exit, the z/OS XML parser does not look at the reason code. Instead, the z/OS XML parser saves the reason code, along with the return code and the diagnostic code, in the extended diagnostic area so that the caller of the z/OS XML parser has access to it by calling GXL1CTL (GXL4CTL). The z/OS XML parser will provide return and reason codes to the caller in the event of a failure by the exit, or if the parser detects a problem with the storage returned from the exit.

For reason code descriptions, see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

For an AMODE 31 example using this exit service, see “GXLE1GTM (GXLGST31 example)” on page 198. For an AMODE 64 example using this exit service, see “GXLE4GTM (GXLGST64 example)” on page 228. These examples are located in SYS1.SAMPLIB .

Default Implementation

If the exit is not provided, then the subpool used will be as follows:

- If running in SRB or cross memory mode, subpool 129 will be used. This is JST related and cannot be freed by unauthorized callers. The key will be the same as the key at the time the z/OS XML parser is invoked.
- If running in task mode (PSATOLD not zero), with PRIMARY=SECONDARY=HOME, then the subpool chosen will depend on the authorization state of the caller and on the specification of the XEC_FEAT_JST_OWNS_STORAGE feature on the GXL1INI (GXL4INI) call. If the caller is running in key 0-7 or supervisor state, they will be considered authorized.
  – Authorized and JST requested — subpool 129
  – Authorized and JST not requested — subpool 229
  – Unauthorized and JST requested — subpool 131

– Unauthorized and JST not requested — subpool 0

Note: If running on a subtask which is sharing subpool 0, then this storage will be owned by the task that owns subpool 0.

These choices of subpool will eliminate the possibility of the z/OS XML parser running in an authorized state while using problem key storage which could be freed and reallocated.

The CONTROL setting will be AUTH for authorized callers. This prevents the storage from being unallocated by an unauthorized caller in the same address space. The storage will be allocated in the caller's key.

For 64-bit callers, the ownership of the storage will be controlled by the setting of the tcbtoken on the TTOKEN parameter:

- When running in SRB mode, the storage will be associated with the cross memory owning task in the primary address space (generally the Job Step Task (JST)).
- When running in Task mode with PRIMARY=SECONDARY=HOME, ownership of the storage will be determined by the specification of the XEC_FEAT_JST_OWNS_STORAGE feature on the GXL1INI (GXL4INI) call. If JST ownership is requested, the TTOKEN of the JST will be used. Otherwise, the TTOKEN of the caller's task will be used.
- When running in Task mode in cross memory mode, the storage will be associated with the cross memory owning task (generally the JST) in the primary address space.

GXLFST31 (GXLFST64) — free memory

Description

This service frees an area of memory previously obtained by the GXLGST31 (GXLGST64) service.

Performance Implications

There are no performance implications.

Syntax

```
call gxlfst31,(sys_svc_parm,
              memory_addr,
              memory_len,
              exit_diag_code,
              return_code
              reason_code)
```

Parameters

sys_svc_parm
Supplied parameter

Type:	Address
Length:	Fullword (Doubleword)

The address of the system service parameter that was passed to the z/OS XML parser at initialization time.

memory_addr
Supplied parameter

Type:	Address
Length:	Fullword (Doubleword)

The address of a fullword (doubleword) that contains the address of the memory to be freed.

memory_len
Supplied parameter

Type:	Integer
Length:	Fullword (Doubleword)

A fullword (doubleword) that contains the length of the memory to be freed. Memory will always be freed in the same quantities under which it was allocated.

exit_diag_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the exit can store any diagnostic information (usually a reason code).

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the exit service stores the return code.

reason_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the exit service stores the reason code.

Return and Reason Codes

The z/OS XML parser uses the convention that the exit will provide a return code value of zero when successful. Any nonzero value indicates failure.

For reason code descriptions, see Appendix B, “Reason Codes Listed by Value,” on page 119.

Example

For an AMODE 31 example using this exit service, see “GXLE1FRM (GXLFST31 example)” on page 194. For an AMODE 64 example using this exit service, see “GXLE4FRM (GXLFST64 example)” on page 224. These examples are located in SYS1.SAMPLIB .

Default Implementation

The z/OS XML parser will free all memory obtained. Memory is freed in the same quantities under which it was allocated. See the MVS assembler services reference (SA22-7606) for more details on the STORAGE macro.

GXLSYM31 (GXLSYM64) — StringID service

Description

This service accepts an input string and performs a lookup for its corresponding symbol, which is identical to the string itself. If the symbol has been located, the exit returns the StringID associated with the symbol. If the string does not have a defined symbol, a symbol is created for the string and a StringID is assigned to it. The StringID is then returned to the z/OS XML parser.

Performance Implications

There are no performance implications.

Syntax

```
call gxlsym31,(sys_svc_parm,
              string,
              string_len,
              string_id,
              ccsid,
              exit_diag_code,
              return_code
              reason_code)
```

Parameters

sys_svc_parm
Supplied parameter

Type:	Address
Length:	Fullword (Doubleword)

The address of the system service parameter that was passed to the z/OS XML parser at initialization time.

string
Supplied parameter

Type:	Character string
Length:	determined by the string_len parameter

The string to return an ID for. The length of the string is variable, and is specified by the string_len parameter.

string_len
Supplied parameter

Type:	Integer
Length:	Fullword

A fullword that contains the length of the string pointed to by the string parameter.

string_id
Returned parameter

Type:	Unsigned integer
Length:	Fullword

The numeric identifier for the string. The range of valid values is 1 to 2 GB - 1. The value zero is reserved for use by the z/OS XML parser.

ccsid
Supplied parameter

Type:	Integer
Length:	Fullword

The Coded Character Set IDentifier (CCSID) that identifies the character set of the string. The z/OS XML parser will provide the same CCSID in this parameter that the caller of the parser specified at parser initialization time.

exit_diag_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword where the exit can store any diagnostic information (usually a reason code). This will be stored in the diagnostic area and made available on the GXL1CTL (GXL4CTL) call.

return_code
Returned parameter

Type:	Integer
Length:	Fullword

The name of a fullword containing the return code. A return code value of zero means success; any nonzero return code indicates failure.

Return Codes

The z/OS XML parser uses the convention that the exit will provide a return code value of zero when successful. Any nonzero value indicates failure. If a nonzero return code is provided by the exit, the z/OS XML parser saves it in the extended diagnostic area so that the caller of the parser has access to it by calling GXL1CTL (GXL4CTL).

Example

For an AMODE 31 example of using this exit service, see “GXLSYM31 example” on page 203. For an AMODE 64 example of using this exit service, see “GXLSYM64 example” on page 234. These examples are located in SYS1.SAMPLIB .

Default Implementation

There is no default implementation. If this exit is not specified by the caller, StringIDs are not used by the z/OS XML parser. Length/value pairs representing all strings from the XML text are passed through to the parsed data stream for return to the caller. See “String Identifiers” on page 23 for more details about length/value pairs and StringIDs in the parsed data stream.

Chapter 9. Diagnosis and problem determination

The diagnostic facilities of this z/OS XML parser can be used to debug both the operation of the z/OS XML parser itself and the input XML document. Since well-formedness checking is an integral part of the parsing process, and since the complexity of XML documents can be very high, the opportunity for encountering a flaw in the input stream that is difficult to diagnose is significant. To assist in diagnosis, the z/OS XML parser provides the following support:

- XMLDATA IPCS subcommand
- Diagnostic Area
- SLIP trap for reason codes from z/OS XML parser
- ARR recovery routine

XMLDATA IPCS subcommand

To make it easier to analyze z/OS XML System Services dumps, the **XMLDATA** subcommand is provided for use with the IPCS formatter. To use the subcommand, input the following under IPCS option 6:

```
COMMAND: XMLDATA address option
```

The *address* parameter is the address of the z/OS XML parser's Parser Anchor Block (PAB); this is a required parameter. The *address* parameter accepts both 31- and 64-bit addresses. If you do not know the value for the *address* parameter, you can place a '0' in the *address* field, and XMLDATA will try to locate the value for you, for example: `XMLDATA 0 TRACE`. Although this method is not guaranteed to work, it is still an available option.

The *option* parameter allows you to select what information you want to review within the provided dump (see Table 27 for a list of options and their descriptions). If nothing is provided for the *option* parameter, **XMLDATA** will use the default option BASIC. The following table lists the options available for **XMLDATA**:

Table 27. XMLDATA options

Option	Description
BASIC	Displays to the screen widely used dump information. Such information includes the following: the PSW and any general information during the abend; the value of the registers; an API trace; a user input parameter list; feature flags, return code and reason codes; and the last 64 bytes of the input and output buffers.
PARAM	Displays the parameter list values for the GXL1PRS or GXL4PRS entry points.

Table 27. XMLDATA options (continued)

Option	Description
BUFFER (*inlen*, *outlen*, *fraglen*)	Displays the last *inlen* bytes of the input buffer ending at where the parser abends, displays the last *outlen* bytes of the output buffer and displays the first *fraglen* bytes of the fragment buffer. The fragment buffer option is only available for a non-validating dump. For a validating dump, the input buffer option will not display the most current bytes of data at where the z/OS XML parser abends, but instead the input buffer option will display from the beginning of the input buffer for *inlen* bytes that has been loaded for parsing within the validating z/OS XML parser. If the length value of zero is provided for a specific buffer type, that specific buffer information will be skipped. The *inlen*, *outlen*, and *fraglen* parameters are all optional. For any that are not specified, the default is 128 bytes.
EXTENT	Displays all available free and external extents' information.
MISC	Displays the status of each feature flag, input document encoding, exit services, return code and reason codes.
TRACE (*option*)	Displays the trace of the API calls. The *option* parameter is optional. Providing 'ADV' in the *option* parameter displays a more advanced API trace. Otherwise, a simple API trace will be displayed. (Default is to display a simple API trace).
PAB	Displays all the defined fields in the PAB.
STRUCT (*option*, *address*)	Displays the formatted control blocks including the z/OS XML parser diagnostic area, element stack, default attribute record, local name tree, prefix tree, namespace tree and data buffer. The data buffer option is only available if the dump is taken with the validation feature flag turned on. The *option* parameter is required, otherwise no control block will be displayed. The options include the following: XD, XELE, XATT, LN, PFX, URI, DBUF, respectively. The *address* parameter is optional and is only available for local name, prefix, namespace tree and data buffer option. For local name, prefix, and namespace trees, if you do not want to display the tree from the root node, then provide a child node address for the tree to use as the root node. For data buffer, if you want to display the details of a specific internal input buffer, then provide a data buffer address. (For the address parameter, the default for the trees is the tree root node address.) If no options or addresses are selected, a menu of all available options will be displayed.
MARKED	Displays data that was parsed by the z/OS XML parser, but has not yet been placed in the output buffer, due to the interruption of an abend. This option is only available if the dump is taken with the validation feature flag turned off.
PMM	Displays the formatted Module Map: PMM, Secondary Table: PST, and System Control: PSC.
HELP	Displays all available options and their descriptions.

The following is an example of the **XMLDATA** subcommand:

```
XMLDATA 00002940121498028 PAB
```

Diagnostic Area

On the GXL1CTL (GXL4CTL) call, there is a diagnostic area where the z/OS XML parser places information that can be useful when debugging a failure or incorrect behavior in the parser. This area is mapped by macro GXLYXD. The diagnostic area contains the following fields:

XD_Eye
: Eyecatcher GXLYXD

XD_Version
: The z/OS XML parser version number.

XD_PAB
: Address of Parser Anchor Block for this parse instance.

XD_InBuff
: Address of current input buffer.

XD_InBuffOffset
: Offset into input buffer where the z/OS XML parser stopped.

XD_OutBuff
: Address of current output buffer.

XD_OutBuffOffset
: Offset into output buffer where the last valid entry can be found.

XD_LastRC
: Return code from the last call to GXP1PRS (GXP4PRS) or GXL1CTL (GXL4CTL).

XD_LastRSN
: Reason code from the last call to GXP1PRS (GXP4PRS) or GXL1CTL (GXL4CTL).

XD_StorageRC
: Return code from call to STORAGE.

XD_StorageRsn
: Reason code from call to STORAGE.

XD_Iarv64Rc
: Return code from call to IARV64.

XD_Iarv64Rsn
: Reason code from call to IARV64.

XD_StorExitRc
: Return code from storage exit.

XD_StorExitRsn
: Reason code from storage exit.

XD_StorExitDiag
: The diag code from the storage exit.

XD_SymbolLength
: Length of the symbol which was rejected by the user symbol exit routine.

XD_IFA_RC
: The return code from the request to run on a ZAAP.

SLIP trap for return codes from the z/OS XML parser

To obtain a dump on a specific reason code from any of the z/OS XML parser callable services, use the release appropriate SLIP example in the following table:

Table 28. SLIP examples by release

z/OS release	SLIP example

V1.9 or lower	`SLIP SET,IF,A=SYNCSVCD,RANGE=(10?+220?+48?+8?+E),`
	`DATA=(13G!!+b0,EQ,xxxxxxxx),`
	`SDATA=(CSA,LPA,TRT,SQA,RGN,SUM),j=jobname,END`

V1.10	`SLIP SET,IF,A=SYNCSVCD,RANGE=(10?+220?+48?+8?+E),`
	`DATA=(4G!+E8!+b0,EQ,xxxxxxxx),`
	`SDATA=(CSA,LPA,TRT,SQA,RGN,SUM),j=jobname,END`

where `xxxxxxxx` is the 8 digit (4 byte) reason code that is to be trapped and `j=jobname` is the optional jobname that is expected to issue the error (for example, j=IBMUSER).

ARR recovery routine

z/OS XML provides an ARR recovery routine to assist with problem determination and diagnostics. This recovery routine can be turned on through an initialization option when invoked through the assembler API. For callers of the C/C++ parser API (gxlpParse), when running in Language Environment, the ARR recovery routine is provided by default in most cases. For C or C++ callers who are running in either SRB mode or under an existing FRR routine, the z/OS XML ARR will not be provided, as it would not work properly in those environments.

If the z/OS XML parser abends, the z/OS XML ARR routine will get control and will collect dumps and return to the caller with a XRC_FATAL return code. For unauthorized callers, an IEATDUMP will be taken in data set *userid*.GXLSCXML.DYYMMDD.THHMMSS.DUMP, where DYYMMDD is the date and THHMMSS is the time the dump was taken. The task level ACEE is used to obtain the *userid*. If there is no task level ACEE, the address space level ACEE is used. If there is no address space level ACEE, a dump is not taken. For authorized callers, an SDUMPX will be taken into a system dump data set.

If the user would like to continue parsing, he must terminate and re-initialize a PIMA following any abend in the z/OS XML parser.

Chapter 10. System Admin: Servicing the z/OS XML parser

This chapter describes how to install fixes for both the non-validating and validating z/OS XML parsers. Before you begin, you should know that the non-validating z/OS XML parser resides in the LPA. However, the default location for the validating z/OS XML parser is in SIEALNKE.

Note: The option is available to place the validating z/OS XML parser into LPA. However, the instructions below assume it is located in SIEALNKE.

Also, before updating the code, it is recommended that you quiesce any applications that might be using the z/OS XML parser. Otherwise, these applications may receive parsing errors. Here are the steps to install fixes for the z/OS XML parser:

1. Install the fix to the z/OS XML parser in either one of the two locations:
   - SYS1.LPALIB (for the non-validating parser)
   - SYS1.SIEALNKE (for the validating parser)

   You can also install the fix in a temporary library.

   Note: The next step assumes the fix is installed in one of the above locations. If this is not the case, replace the above location in the instruction with the name of the library where the fix is installed.
2. Issue operator command. For the non-validating parser:
   ```
   SETPROG LPA,ADD,MODNAME=GXLINLPA,DSNAME=SYS1.LPALIB
   ```
   For the validating parser:
   ```
   SETPROG LPA,ADD,MODNAME=GXLIMODV,DSNAME=SYS1.SIEALNKE
   ```
   Note: If a parse is in progress at the time of a z/OS XML parser code update, and that parse requires that input or output buffers be spanned, a failure will occur on the first call back to the parser when the caller has a new input or output buffer. The caller has to terminate the parse instance in this case, and begin the parse again within the context of a new parse instance.
3. If you do a dynamic add of GXLINLPA or GXLIMODV, and are unsatisfied with the fix, you can remove the new version with one of the following commands:
   - For GXLINPA:
     ```
     SETPROG LPA,DELETE,MODNAME=GXLINLPA,FORCE=YES,CURRENT
     ```
   - For GXLIMODV:
     ```
     SETPROG LPA,DELETE,MODNAME=GXLIMODV,FORCE=YES,CURRENT
     ```

Servicing the dynamic LPA exit

The z/OS XML parser's dynamic LPA exit is an LPA routine itself. If this routine requires service, do the following:

1. Install the fix to SYS1.LPALIB or a temporary directory.

   Note: The next step assumes the fix is installed in SYS1.LPALIB. If this is not the case, replace SYS1.LPALIB in the instruction with the name of the library of where the fix is installed.
2. Issue the following operator commands :

```
SETPROG EXIT DELETE EXITNAME(CSVDYLPA) MODULE(GXLINDLX)

SETPROG EXIT ADD EXITNAME(CSVDYLPA) MODULE(GXLINDLX) DSNAME(SYS1.LPALIB)
```

Appendix A. Return Codes Listed by Value

This section lists return codes by value and describes them.

Hex Value	Return Code	Description
0000	XRC_SUCCESS	The z/OS XML parser service was successful.
0004	XRC_WARNING	The z/OS XML parser service has partial success.
0008	XRC_FAILURE	Processing failed. Returned data areas and parms valid.
000C	XRC_NOT_WELL_FORMED	The document is not-well-formed.
0010	XRC_FATAL	Processing failed. Returned data areas or output parameters cannot be relied on to contain valid data.
0014	XRC_LOAD_FAILED	The load of the specified service failed. The return code from the LOAD macro is returned in the reason code field.
0018	XRC_NOT_VALID	The document is not valid according to the specified schema.

Appendix B. Reason Codes Listed by Value

This section describes reason codes, listing them by hexadecimal value and describing actions to correct the error.

0000	**XRSN_SUCCESS** The z/OS XML parser service was successful. *Action:* None
1000	**XRSN_PIMA_NOT_INITIALIZED** The PIMA passed to a z/OS XML parser service is unusable. *Action:* The PIMA passed has not been initialized with a call to the z/OS XML parser initialization service GXL1INI or GXL4INI or the PIMA address is incorrect.
1001	**XRSN_PIMA_SMALL** The length of the PIMA is too small. *Action:* The PIMA passed on GXL1INI or GXL4INI must be at least as big as the constant XEC_MIN_PIMA_SIZE equal to 0x20000 defined in macro GXLYXEC.
1002	**XRSN_PIMA_RESIDUAL_DATA** Initialization has already been done on this PIMA. *Action:* The GXL1INI or GXL4INI service has been called to initialize the PIMA, but the PIMA storage has already been initialized. You must call GXL1TRM or GXL4TRM before the PIMA can be reinitialized to guarantee that all resources have been cleaned up.
1004	**XRSN_PIMA_INCONSISTENT_STATE** The z/OS XML parser exited without cleaning up. *Action:* Attempt to collect a dump of the problem. The joblog for the address space should contain a symptom dump which identifies the abend code. If running from a user address space, allocate a SYSMDUMP DD and recreate the problem. If running in some system address space, use SLIP to get a dump of the abend. Contact your system administrator for help in getting the dump and possibly contacting IBM.
1005	**XRSN_CTL_DATA_PARM_INVALID** The CTL_DATA parm is invalid. *Action:* It is null, but is a required input parmameter for this feature flag. Call the ctl function again, passing in the required parameter.
1006	**XRSN_IMODV_NOT_LOADED** The validating parser has not been loaded. *Action:* Invoke the GXL1LOD or GXL4LOD to load the validating parser. Call initialization again, after a successful load.

1100	**XRSN_STORAGE_31_GET_ERROR** Unable to allocate memory. *Action:* If your application does not already call GXL1CTL after the parse, add a call to GXL1CTL. The address returned by GXL1CTL points to an area mapped by GXLYXD. Extract the return and reason code from the XD area, pertaining to storage access failures that occurred using the STORAGE macro. Contact your system administrator for help in interpreting these values.
1101	**XRSN_STORAGE_64_GET_ERROR** Unable to allocate memory. *Action:* If your application does not already call GXL1CTL after the parse, add a call to GXL1CTL. The address returned by GXL1CTL points to an area mapped by GXLYXD. Extract the return and reason code from the XD area, pertaining to storage access failures using the IARV64 service. Contact your system administrator for help in interpreting these values.
1140	**XRSN_STORAGE_GET_EXIT_TOO_SMALL** The storage returned from get storage exit is too small. *Action:* If your application does not already call GXL1CTL after the parse, add a call to GXL1CTL. The address returned by GXL1CTL points to an area mapped by GXLYXD. Extract the return and reason code from the XD area, pertaining to storage exit failure. Contact your system administrator for help in interpreting these values.
1143	**XRSN_STORAGE_31_SFREE_ERROR** Single failure when attempting to free storage. *Action:* Contact your system administrator.
1144	**XRSN_STORAGE_31_MFREE_ERROR** Multiple failures when attempting to free storage. *Action:* Contact your system administrator.
1145	**XRSN_STORAGE_64_SFREE_ERROR** Single failure when attempting to free storage. *Action:* Contact your system administrator.
1146	**XRSN_STORAGE_64_MFREE_ERROR** Multiple failures when attempting to free storage. *Action:* Contact your system administrator.
1147	**XRSN_STORAGE_CORRUPTED_ERROR** Storage header has been corrupted. *Action:* Contact your system administrator.

1148	**XRSN_INPUT_BUFFER_ACCESS_ERROR** The user abended when trying to access the input buffer. *Action:* Check the input buffer parameter and length passed into the parser to be sure they are correct. If the input parameters are correct, Contact your system administrator. .
1149	**XRSN_INPUT_BUFFER_ACCESS_ERROR_ND** The user abended when trying to access the input buffer. No dump was taken. *Action:* Check the input buffer parameter and length passed into the parser to be sure they are correct. If the input parameters are correct, Contact your system administrator. .
1150	**XRSN_OUTPUT_BUFFER_ACCESS_ERROR** The user abended when trying to access the output buffer. *Action:* Check the output buffer parameter and length passed into the parser to be sure they are correct. If the output parameters are correct, Contact your system administrator. .
1151	**XRSN_OUTPUT_BUFFER_ACCESS_ERROR_ND** The user abended when trying to access the output buffer. No dump was taken. *Action:* Check the output buffer parameter and length passed into the parser to be sure they are correct. If the output parameters are correct, Contact your system administrator. .
1152	**XRSN_PIMA_ACCESS_ERROR** The user abended when trying to access the PIMA. *Action:* Check the PIMA parameter and length passed into the parser to be sure they are correct. If the PIMA parameters are correct, Contact your system administrator. .
1153	**XRSN_PIMA_ACCESS_ERROR_ND** The user abended when trying to access the PIMA. No dump was taken. *Action:* Check the PIMA parameter and length passed into the parser to be sure they are correct. If the PIMA parameters are correct, Contact your system administrator. .
1154	**XRSN_UNKNOWN_ERROR** An unknown abend occurred. *Action:* Contact your system administrator.
1155	**XRSN_UNKNOWN_ERROR_ND** Unknown abend occurred and no dump was taken. *Action:* Contact your system administrator.
1156	**XRSN_STORAGE_OBTAIN_FAILED** A storage obtain request failed *Action:* Contact your system administrator.

1157	**XRSN_STORAGE_OBTAIN_FAILED_ND** A storage obtain request failed, no dump taken *Action:* Contact your system administrator.
1201	**XRSN_PARM_ENCODING_SPEC_INVALID** The ccsid passed is not supported. *Action:* The CCSID parameter on the call to GXL1INI or GXL4INI is not one of the supported character encodings. Pass only permitted CCSID parameters. See the documentation of the GXL1INI service for supported ccsid constants.
1202	**XRSN_PARM_FEATURE_FLAG_INVALID** Undefined feature flag is set *Action:* The feature flag parameter passed to GXL1INI or GXL4INI or GXL1CTL or GXL4CTL has an undefined bit set or a bit that is invalid for this api set. You can only set features that are defined or supported on the api.
1203	**XRSN_PARM_UNSUPPORT_ENCODING** XML encoding string is not supported. *Action:* The encoding string in the XML declaration is not supported. Use only the supported encoding names.
1300	**XRSN_BUFFER_INBUF_SMALL** The input buffer size is too small. *Action:* The query service was not able to parse a complete XML declaration. The caller needs to pass more of the document to the service.
1301	**XRSN_BUFFER_INBUF_END** The end of the input buffer has been reached. *Action:* This is a normal reason code for spanning buffers.
1302	**XRSN_BUFFER_OUTBUF_SMALL** The output buffer was too small to contain the next item. *Action:* The caller must reset the parser, then parse the document again from the beginning, passing in a larger output buffer.
1303	**XRSN_BUFFER_OUTBUF_END** The end of the output buffer has been reached *Action:* This is a normal reason code for spanning buffers.
1304	**XRSN_BUFFER_INOUTBUF_END** The end of both buffers have been reached *Action:* This is a normal reason code for spanning buffers.

1305 **XRSN_STORAGE_GET_EXIT_ERROR**

Application storage exit unable to allocate memory.

Action: If your application does not already call GXL1CTL after the parse, add a call to GXL1CTL. The address returned by GXL1CTL points to an area mapped by GXLYXD. Extract the return and reason code from the XD area, pertaining to storage access failures. Contact your system administrator for help in interpreting these values.

1307 **XRSN_STORAGE_SFREE_EXIT_ERROR**

User free storage exit has one failure.

Action: Contact your system administrator.

1308 **XRSN_STORAGE_MFREE_EXIT_ERROR**

User free storage exit has multiple failures.

Action: Contact your system administrator.

1309 **XRSN_DYNAMIC_CODE_CHANGE**

z/OS XML parser was re-installed.

Action: Caller needs to terminate the parser and restart with parser initialization.

1310 **XRSN_SYM_EXIT_ERROR**

The symbol exit returned an error.

Action: Contact the owner of the symbol exit and have them debug the problem.

1400 **XRSN_DEALLOC_EXIT_MISSING**

Allocation exit specified with deallocation exit

Action: The service exit specification on a call to GXL1INI or GXL4INI contains an exit to allocate storage, but no exit to deallocate storage. Either both or neither is required.

1401 **XRSN_ALLOC_EXIT_MISSING**

Deallocation exit specified with allocation exit

Action: The service exit specification on a call to GXL1INI or GXL4INI contains an exit to deallocate storage, but no exit to allocate storage. Either both or neither is required.

1403 **XRSN_OPTN_UNKNOWN**

Unsupported value set on the options parameter.

Action: Refer to the API documentation for the correct values to pass to this service.

1404 **XRSN_QXDWORK_AREA_SMALL**

Query service work area length is too small.

Action: Pass a bigger area.

1405	**XRSN_INTERNAL_ERROR** Internal error in the z/OS XML parser. *Action:* Contact your system administrator.
1407	**XRSN_FEATURE_FLAG_INVALID_IN_ENV** The recovery feature flag is on, but the program either has an existing FRR or is in SRB mode. This feature is not valid in these environments. *Action:* Reinitialize the parse with the recovery feature flag turned off.
1408	**XRSN_INVALID_OPTION** The operation being performed is not valid for this service. *Action:* Refer to the API documentation to determine which parsing services this option is valid for.
1500	**XRSN_SVC_UNKNOWN** The code specified for the svc_code parameter is invalid. *Action:* Refer to the API documentation for the correct values for the svc_code parameter.
1501	**XRSN_NO_OSR_SPECIFIED** No OSR has been loaded via a CTL call. *Action:* Perform a CTL_LOAD_OSR operation via CTL with a nonzero XSCH_OSR_PTR.
1502	**XRSN_NO_SCHEMAS_SPECIFIED** Either the schema vector parameter passed was NULL, or the number of schemas specified in the vector was zero. *Action:* Pass in a valid schema vector that contains one or more text. schemas to process.
1503	**XRSN_NO_OSR_BUFFER_SPECIFIED** No OSR buffer was for generation. *Action:* Pass in the address of a buffer to receive a generated OSR.
1504	**XRSN_OSR_INVALID** The data within the OSR is invalid. *Action:* Ensure that the correct address of the OSR is being passed.
2000	**XRSN_COMMENT_INCOMPLETE** The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within comment markup. *Action:* Change the document to correct the error and retry.
2001	**XRSN_CDATA_INCOMPLETE** The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within CDATA markup. *Action:* Change the document to correct the error and retry.

2002 **XRSN_PI_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within processing instruction markup.

Action: Change the document to correct the error and retry.

2003 **XRSN_ATTR_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within attribute markup.

Action: Change the document to correct the error and retry.

2004 **XRSN_ENDTAG_NOT_REACHED**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended without reaching the document element end tag.

Action: Change the document to correct the error and retry.

2006 **XRSN_TAG_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within an element start tag.

Action: Change the document to correct the error and retry.

2007 **XRSN_NS_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within namespace declaration markup.

Action: Change the document to correct the error and retry.

2008 **XRSN_XML_DECL_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within the XML declaration.

Action: Change the document to correct the error and retry.

2009 **XRSN_DTD_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within doctype declaration markup.

Action: Change the document to correct the error and retry.

2010 **XRSN_SUBSET_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within internal subset markup.

Action: Change the document to correct the error and retry.

2011 **XRSN_SUBSET_ELEM_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within an element declaration.

Action: Change the document to correct the error and retry.

2012 **XRSN_SUBSET_NOTATION_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within a notation declaration.

Action: Change the document to correct the error and retry.

2013 **XRSN_SUBSET_COMMENT_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within comment markup.

Action: Change the document to correct the error and retry.

2015 **XRSN_SUBSET_PEREF_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within a parameter entity reference.

Action: Change the document to correct the error and retry.

2016 **XRSN_SUBSET_ENTITY_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within an entity declaration.

Action: Change the document to correct the error and retry.

2017 **XRSN_SUBSET_ATTL_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within an attribute list declaration.

Action: Change the document to correct the error and retry.

2018 **XRSN_MARKUP_INCOMPLETE**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended within markup.

Action: Change the document to correct the error and retry.

2019 **XRSN_DOC_ELEM_NOT_FOUND**

The GXL1CTL (GXL4CTL) API was called with the finish option and the input document was not complete. The document ended without finding the document element.

Action: Change the document to correct the error and retry.

3000 **XRSN_ATTR_DUPLICATE**

Duplicate attributes were found.

Action: Change the document to correct the error and retry.

3001	**XRSN_NS_DUPLICATE** Duplicate namespace declaration found. *Action:* Change the document to correct the error and retry.
3002	**XRSN_NS_ATTR_PREFIX_NOT_DECL** Namespace prefix on attribute not declared. *Action:* Change the document to correct the error and retry.
3003	**XRSN_NS_ELEM_PREFIX_NOT_DECL** Namespace prefix on element tag not declared. *Action:* Change the document to correct the error and retry.
3004	**XRSN_ENC_DETECTED_INVALID** Encoding detected during query is unsupported. *Action:* During the query service, an unsupported byte sequence is found at the beginning of the document.
3006	**XRSN_CHAR_ERROR** Incorrectly encoded character found in the input stream. *Action:* Contact your system administrator.
3007	**XRSN_COMMENT_DASH_MISSING** Comment without starting dash found. *Action:* Check the document for a comment markup missing a dash in the beginning and correct the document.
3008	**XRSN_COMMENT_CHAR_INVALID** Comment markup contains incorrect character. *Action:* Change the document to correct the error and retry.
3009	**XRSN_COMMENT_RIGHT_ANGLE_MISSING** Comment is missing the ending angle bracket at the end of the markup. *Action:* Change the document to correct the error and retry.
3010	**XRSN_CDATA_KEYWORD_INVALID** CDATA keyword expected but not found. *Action:* Change the document to correct the error and retry.
3011	**XRSN_CDATA_LEFT_BRACKET_MISSING** Left square bracket expected in CDATA markup. *Action:* Change the document to correct the error and retry.
3013	**XRSN_CDATA_CHAR_INVALID** A character was found that is not allowed within a CDATA section. *Action:* Change the document to correct the error and retry.

3017 **XRSN_PI_CHAR_INVALID**

A character was found that is not allowed within a Processing Instruction.

Action: Change the document to correct the error and retry.

3018 **XRSN_ATTR_NAME_CHAR_INVALID**

A character was found that is not allowed within an attribute name.

Action: Change the document to correct the error and retry.

3019 **XRSN_ATTR_LNAME_CHAR_INVALID**

A character was found that is not allowed within an attribute local name.

Action: Change the document to correct the error and retry.

3020 **XRSN_ATTR_EQUAL_MISSING**

An incorrect character was found after the attribute name, and the only character allowed is ″=″.

Action: Change the document to correct the error and retry.

3021 **XRSN_ATTR_QUOTE_MISSING**

An incorrect character was found after the attribute ″=″ character, and the only characters allowed here is either white space, or a single or double quote.

Action: Change the document to correct the error and retry.

3022 **XRSN_ATTR_VALUE_CHAR_INVALID**

An incorrect character was found in an attribute value.

Action: Change the document to correct the error and retry.

3023 **XRSN_ATTR_REF_CHAR_INVALID**

An incorrect character was found in entity reference in an attribute value.

Action: Change the document to correct the error and retry.

3024 **XRSN_ATTR_REF_NAME_CHAR_INVALID**

An incorrect character was found in entity reference in an attribute value.

Action: Change the document to correct the error and retry.

3025 **XRSN_ATTR_REF_VALUE_INVALID**

Incorrect character found in character entity reference in an attribute value.

Action: Change the document to correct the error and retry.

3026 **XRSN_CONTNT_REF_CHAR_INVALID**

An incorrect character was found in entity reference in element content.

Action: Change the document to correct the error and retry.

3027 **XRSN_CONTNT_REF_NAME_INVALID**

An incorrect character was found in entity reference in element content.

Action: Change the document to correct the error and retry.

3028 **XRSN_CONTNT_REF_VALUE_INVALID**

An incorrect character was found in character entity reference in element content.

Action: Change the document to correct the error and retry.

3029 **XRSN_MARKUP_INVALID**

An incorrect character is found within markup.

Action: Change the document to correct the error and retry.

3030 **XRSN_CONTNT_CHAR_INVALID**

An incorrect character is found in element content

Action: Change the document to correct the error and retry.

3031 **XRSN_TAG_ELEMNAME_INVALID**

An incorrect character is found in an element tag name

Action: Change the document to correct the error and retry.

3032 **XRSN_TAG_LNAME_INVALID**

An incorrect character is found in an element tag name.

Action: Change the document to correct the error and retry.

3033 **XRSN_TAG_CHAR_INVALID**

An incorrect character is found in an element start tag.

Action: Change the document to correct the error and retry.

3034 **XRSN_TAG_EMPTY_INVALID**

An incorrect character is found after the ″/″ character to end the element tag. The only character allowed is a greater than symbol to end the empty element tag.

Action: Change the document to correct the error and retry.

3035 **XRSN_ENDTAG_NAME_MISMATCH**

At the element end tag, a mis-match element name is found compared to the name of the start element

Action: Change the document to correct the error and retry.

3036 **XRSN_ENDTAG_EMPTY_TAG_INVALID**

An incorrect character is found in the element end tag after the element name. The only characters allowed after the name is white space or the greater than symbol.

Action: Change the document to correct the error and retry.

3038 **XRSN_NS_CHAR_INVALID**

Incorrect character found in namespace URI.

Action: Change the document to correct the error and retry.

3039 **XRSN_NS_WHITESPACE_CHAR_INVALID**

Incorrect character in namespace declaration. Expecting either white space or ″=″.

Action: Change the document to correct the error and retry.

3040 **XRSN_NS_PFX_NAME_INVALID**

An incorrect character is found in the prefix name portion of a namespace declaration.

Action: Change the document to correct the error and retry.

3041 **XRSN_NS_QUOTE_MISSING**

Incorrect character in namespace declaration after the ″=″ character. Expected a single or double quote or a white space character.

Action: Change the document to correct the error and retry.

3042 **XRSN_NS_REF_CHAR_INVALID**

An incorrect character was found in entity reference in a namespace declaration.

Action: Change the document to correct the error and retry.

3043 **XRSN_NS_REF_NAME_CHAR_INVALID**

An incorrect character was found in entity reference in a namespace declaration.

Action: Change the document to correct the error and retry.

3044 **XRSN_NS_REF_VALUE_INVALID**

Incorrect character found in character entity reference in a namespace declaration.

Action: Change the document to correct the error and retry.

3045 **XRSN_DTD_DOCTYPE_INVALID**

Incorrect character found while parsing DOCTYPE keyword.

Action: Change the document to correct the error and retry.

3046 **XRSN_XML_VER_VALUE_INVALID**

An incorrect XML version number was specified. The only allowed values are ″1.0″ or ″1.1″.

Action: Change the document to correct the error and retry.

3047 **XRSN_XML_VER_KEYWORD_INVALID**

The characters do not match the word ″version″

Action: Change the document to correct the error and retry.

3048 **XRSN_XML_VER_EQUAL_MISSING**

Expected white space or ″=″ character after ″version″.

Action: Change the document to correct the error and retry.

3049 **XRSN_XML_VER_QUOTE_MISSING**

An incorrect character is detected after the ″=″ where it is expected to be a single quote, double quote or a white space character.

Action: Change the document to correct the error and retry.

3050 **XRSN_XML_CHAR_INVALID**

In the XML Declaration after the close of the version value, an incorrect character is detected.

Action: Change the document to correct the error and retry.

3051 **XRSN_XML_NAME_CHAR_INVALID**

Incorrect character in XML Declaration. Expected either ″s″ for standalone, ″e″ for encoding, white space or ″?″.

Action: Change the document to correct the error and retry.

3052 **XRSN_XML_ENC_KEYWORD_INVALID**

The characters do not match the word ″encoding″.

Action: Change the document to correct the error and retry.

3053 **XRSN_XML_ENC_EQUAL_MISSING**

Expected white space or ″=″ character after ″encoding″.

Action: Change the document to correct the error and retry.

3054 **XRSN_XML_ENC_QUOTE_MISSING**

An incorrect character is detected after the ″=″ where it is expected to be a single quote, double quote or a white space character.

Action: Change the document to correct the error and retry.

3055 **XRSN_XML_ENC_CHAR_INVALID**

An incorrect character is detected in the XML Declaration encoding value.

Action: Change the document to correct the error and retry.

3056 **XRSN_XML_STD_KEYWORD_INVALID**

The characters do not match the word ″standalone″

Action: Change the document to correct the error and retry.

3057 **XRSN_XML_STD_VALUE_INVALID**

An incorrect value for standalone was specified. The only allowed values are ″yes″ or ″no″.

Action: Change the document to correct the error and retry.

3058	**XRSN_XML_STD_EQUAL_MISSING** Expected white space or ″=″ character after ″standalone″. *Action:* Change the document to correct the error and retry.
3059	**XRSN_XML_STD_QUOTE_MISSING** An incorrect character is detected after the ″=″ where it is expected to be a single quote, double quote or a white space character. *Action:* Change the document to correct the error and retry.
3060	**XRSN_XML_END_CHAR_INVALID** An incorrect character is detected at the end of the XML declaration, where ″?>″ is expected. *Action:* Change the document to correct the error and retry.
3061	**XRSN_ENTITY_NOT_DEFINED** Entity not defined or not defined correctly. *Action:* Change the document to correct the error and retry.
3062	**XRSN_CHAR_INVALID** An incorrect character was detected in the document. Either white space or ″<″ was expected. *Action:* Change the document to correct the error and retry.
3063	**XRSN_PROLOGUE_CHAR_INVALID** The initial character in the document was incorrect. Either white space or ″<″ was expected. Possibly the document encoding does not match the parser encoding specified during initialization. *Action:* Change the document to correct the error and retry.
3064	**XRSN_XML_DECL_NOT_ALLOWED** Any Characters other than BOM are not allowed before the XML declaration in the XML document. *Action:* Change the document to correct the error and retry.
3065	**XRSN_MULTIPLE_DOC_ELEMENTS** Multiple elements were found at the document level. Only one is allowed. *Action:* Change the document to correct the error and retry.
3066	**XRSN_ENTITY_LOOP_REF** An entity refers directly, or indirectly to itself. Recursion is not allowed. *Action:* Change the document to correct the error and retry.
3067	**XRSN_NS_URI_EMPTY** A non-default namespace declaration contains a URI value of zero length and the XML version is 1.0. *Action:* Change the document to correct the error and retry.

3068	**XRSN_INVALID_CHAR_SEQ**	An invalid character sequence found in the content portion of the document.	*Action:* Change the document to correct the error and retry.
3069	**XRSN_ENTITY_MARKUP_INCOMPLETE**	Incomplete markup in entity.	*Action:* Change the document to correct the error and retry.
5000	**XRSN_DTD_NAME_CHAR_INVALID**	An incorrect character is detected after the root element name of the document type declaration where only ″SYSTEM″, ″PUBLIC″, square bracket, or greater than than characters are allowed.	*Action:* Change the document to correct the error and retry.
5001	**XRSN_DTD_CHAR_INVALID**	Incorrect character found in document type declaration.	*Action:* Change the document to correct the error and retry.
5002	**XRSN_DTD_EXTERNALID_INVALID**	The external ID keyword does not match the word ″SYSTEM″ or ″PUBLIC″.	*Action:* Change the document to correct the error and retry.
5003	**XRSN_DTD_QUOTE_MISSING**	Incorrect quotation delimiter after external identifier. It is expected to be a single quote, double quote or a white space character.	*Action:* Change the document to correct the error and retry.
5004	**XRSN_DTD_FILENAME_INVALID**	Incorrect character in external identifier filename.	*Action:* Change the document to correct the error and retry.
5005	**XRSN_SUBSET_CHAR_INVALID**	Incorrect character in internal subset of the DTD.	*Action:* Change the document to correct the error and retry.
5006	**XRSN_SUBSET_MARKUP_INVALID**	An incorrect character is detected within the markup keyword in the internal subset of the doctype declaration.	*Action:* Change the document to correct the error and retry.
5007	**XRSN_ELEM_CONTNT_CHAR_INVALID**	An incorrect character is found in the element content portion of the element type declaration located in the internal subset of the doctype declaration.	*Action:* Change the document to correct the error and retry.

5008 **XRSN_ELEM_CHAR_INVALID**

Incorrect character in element declaration in DTD.

Action: Change the document to correct the error and retry.

5009 **XRSN_ELEM_LNAME_INVALID**

An incorrect character is found in the element name portion of an element declaration.

Action: Change the document to correct the error and retry.

5010 **XRSN_ELEM_ELEMNAME_INVALID**

An incorrect character is found in the element name portion of an element declaration.

Action: Change the document to correct the error and retry.

5011 **XRSN_NTTN_CHAR_INVALID**

Incorrect character in notation declaration in DTD.

Action: Change the document to correct the error and retry.

5012 **XRSN_NTTN_NAME_INVALID**

An incorrect character is found in the notation declaration name.

Action: Change the document to correct the error and retry.

5013 **XRSN_NTTN_ID_INVALID**

The external or public identifier string in the notation declaration does not match with the word ″SYSTEM″ or ″PUBLIC″.

Action: Change the document to correct the error and retry.

5014 **XRSN_NTTN_QUOTE_MISSING**

Incorrect quotation delimiter after external identifier. It is expected to be a single quote, double quote or a white space character.

Action: Change the document to correct the error and retry.

5015 **XRSN_NTTN_FILENAME_INVALID**

Incorrect character in notation identifier literal.

Action: Change the document to correct the error and retry.

5020 **XRSN_PEREF_NAME_CHAR_INVALID**

Incorrect character in parameter entity reference in DTD.

Action: Change the document to correct the error and retry.

5021 **XRSN_ENTY_NAME_CHAR_INVALID**

Incorrect character in entity declaration name in DTD.

Action: Change the document to correct the error and retry.

5022 **XRSN_ENTY_CHAR_INVALID**

Incorrect character in entity declaration in DTD.

Action: Change the document to correct the error and retry.

5023	**XRSN_ENTY_VALUE_INVALID** Incorrect character in entity declaration value in DTD. *Action:* Change the document to correct the error and retry.
5024	**XRSN_ENTY_REF_CHAR_INVALID** An incorrect character was found in entity reference in an entity declaration. *Action:* Change the document to correct the error and retry.
5025	**XRSN_ENTY_REF_NAME_INVALID** Incorrect character was found in entity reference in an entity declaration. *Action:* Change the document to correct the error and retry.
5026	**XRSN_ENTY_REF_VALUE_INVALID** Incorrect character found in character entity reference in an entity declaration. *Action:* Change the document to correct the error and retry.
5027	**XRSN_ENTY_QUOTE_MISSING** Incorrect quotation delimiter in entity declaration in DTD. It is expected to be a single quote, double quote or a white space character. *Action:* Change the document to correct the error and retry.
5028	**XRSN_ENTY_EXTERNALID_INVALID** The external or public identifier string in the entity declaration does not match with the word ″SYSTEM″ or ″PUBLIC″. *Action:* Change the document to correct the error and retry.
5029	**XRSN_ENTY_FILENAME_INVALID** Incorrect character in entity identifier value. *Action:* Change the document to correct the error and retry.
5030	**XRSN_ENTY_NDATA_INVALID** Incorrect character in entity NDATA declaration in DTD. *Action:* Change the document to correct the error and retry.
5031	**XRSN_ENTY_NDATA_NAME_INVALID** An incorrect character is found in the entity NDATA declaration name. *Action:* Change the document to correct the error and retry.
5040	**XRSN_ATTL_ELEMNAME_INVALID** An incorrect character is found in the attribute list declaration element name in the DTD. *Action:* Change the document to correct the error and retry.

5041	**XRSN_ATTL_CHAR_INVALID** An incorrect character is found in the attribute list declaration in the DTD. *Action:* Change the document to correct the error and retry.
5042	**XRSN_ATTL_NAME_CHAR_INVALID** An incorrect character is found in the attribute list declaration attribute name in the DTD. *Action:* Change the document to correct the error and retry.
5043	**XRSN_ATTL_LNAME_CHAR_INVALID** An incorrect character is found in the attribute list declaration attribute name in the DTD. *Action:* Change the document to correct the error and retry.
5044	**XRSN_ATTL_TYPE_INVALID** Incorrect character in attribute list declaration type. The type must match one of these strings: ″ID″,″IDREF″,″IDREFS″,″ENTITY″,″ENTITIES″, ″CDATA″,″NMTOKEN″,″NMTOKENS″ or ″NOTATION″. *Action:* Change the document to correct the error and retry.
5045	**XRSN_ATTL_ENUMLIST_CHAR_INVALID** Incorrect character is found in the attribute list declaration enumerated list. *Action:* Change the document to correct the error and retry.
5046	**XRSN_ATTL_DEFVALUE_CHAR_INVALID** Incorrect character is found in attribute list declaration default. Expected white space, ″#″, or a single or double quote *Action:* Change the document to correct the error and retry.
5047	**XRSN_ATTL_DEF_VALUE_INVALID** Incorrect character is found in attribute list declaration default value. Expected ″REQUIRED″, ″IMPLIED″, or ″FIXED″. *Action:* Change the document to correct the error and retry.
5048	**XRSN_ATTL_QUOTE_MISSING** Incorrect character is found in attribute list declaration default value. Expected single quote, double quote or white space. *Action:* Change the document to correct the error and retry.
5049	**XRSN_ATTL_REF_CHAR_INVALID** An incorrect character was found in entity reference in an attribute list declaration. *Action:* Change the document to correct the error and retry.

5050	**XRSN_ATTL_REF_NAME_INVALID** An incorrect character was found in entity reference in an attribute list declaration. *Action:* Change the document to correct the error and retry.
5051	**XRSN_ATTL_REF_VALUE_INVALID** Incorrect character found in character entity reference in an attribute list declaration. *Action:* Change the document to correct the error and retry.

Appendix D. Reason Codes Listed by Value

This section describes reason codes, listing them by hexadecimal value and describing actions to correct the error.

7001	**XRSN_OIMA_NOT_INITIALIZED** The OIMA provided is unusable. *Action:* Change the schema and retry.
7002	**XRSN_OIMA_NOT_USABLE** The OIMA provided is unusable because a previous reset failed. *Action:* Change the schema and retry.
7003	**XRSN_OIMA_SMALL** The OIMA provided is too small. *Action:* Change the schema and retry.
7005	**XRSN_OIMA_RESIDUAL_DATA** The OIMA is already initialized. *Action:* Change the schema and retry.
7007	**XRSN_JVM_START_FAILED** The Java Virtual Machine failed to start. *Action:* Change the schema and retry.
7008	**XRSN_JVM_STOP_FAILED** The Java Virtual Machine failed to stop. *Action:* Change the schema and retry.
7009	**XRSN_CTLOPTN_UNSUPPORTED** The operation specified for the control parameter is unsupported. *Action:* Change the schema and retry.
7011	**XRSN_JAVACLASS_NOT_FOUND** Java class not found by the ClassLoader. *Action:* Change the schema and retry.
7019	**XRSN_FUNC_NAME_NULL** The specified function name is null. *Action:* Change the schema and retry.
7021	**XRSN_DLL_OPEN_FAILED** Open for the specified DLL failed. *Action:* Change the schema and retry.

7023	**XRSN_FUNC_RETRIEVE_FAILED** Retrieve for the specified DLL function failed. *Action:* Change the schema and retry.
7027	**XRSN_JAVA_METHOD_NOT_FOUND** The Java method cannot be found in the class. See the diagnostic area for the method name. *Action:* Change the schema and retry.
7029	**XRSN_JAVA_METHOD_CALL_FAILED** A Java method call failed. *Action:* Change the schema and retry.
7031	**XRSN_DLL_CLOSE_FAILED** Close for the specified DLL failed. *Action:* Change the schema and retry.
7033	**XRSN_JNI_METHOD_FAILED** A JNI method returned with an exception. *Action:* Change the schema and retry.
7035	**XRSN_OBJECT_NOT_CREATED** Failed to create a new Java object. *Action:* Change the schema and retry.
7037	**XRSN_SCHEMA_NOT_LOADED** No schemas have been loaded into the OSR generator. *Action:* Change the schema and retry.
7039	**XRSN_OIMAPTR_NOT_PROVIDED** No OIMA pointer has been specified. *Action:* Change the schema and retry.
7043	**XRSN_GEN_OSR_ASM_FAILED** OSR generation failed in the assemble phase. *Action:* Change the schema and retry.
7045	**XRSN_GEN_OSR_COMP_FAILED** OSR generation failed in the compile phase. *Action:* Change the schema and retry.
7046	**XRSN_GEN_OSR_FAILED** OSR generation failed. *Action:* Change the schema and retry.

7049	**XRSN_OSR_NOT_VALID** The OSR to load is not valid. *Action:* Change the schema and retry.
7050	**XRSN_OSR_MALLOC_FAILED** The OSR generator could not allocate memory. *Action:* Change the schema and retry.
7051	**XRSN_OSR_MFREE_FAILED** The OSR generator could not free memory. *Action:* Change the schema and retry.
7055	**XRSN_JAVAEXCEPTION_DIAG_FAILED** Could not save the Java exception in the diagnostic area. *Action:* Change the schema and retry.
7057	**XRSN_JAVAEXCEPTION_INCOMPLETE** The Java exception saved in the diagnostic area is incomplete. *Action:* Change the schema and retry.
7059	**XRSN_JAVARSNCODE_NOT_FOUND** Unable to obtain the reason code set by the Java exception. *Action:* Change the schema and retry.
7061	**XRSN_INCORRECT_SCHEMA_URI** The URI specified is incorrect. *Action:* Change the schema and retry.
7063	**XRSN_JAVARSNCODE_UNKNOWN** No specific reason code was set by Java. *Action:* Change the schema and retry.
7065	**XRSN_SCHEMA_URI_NOT_FOUND** The schema identified by the specified URI is not found. *Action:* Change the schema and retry.
7067	**XRSN_SCHEMA_LOAD_FAILED** Unable to load the specified schema. *Action:* Change the schema and retry.
7069	**XRSN_OSR_URI_NOT_FOUND** The OSR identified by the specified URI is not found. *Action:* Change the schema and retry.
7071	**XRSN_STRINGID_SYSSVC_NULL** The system service parameter specified is null. *Action:* Change the schema and retry.

7079	**XRSN_JAVAERRORMESSAGE_INCOMPLETE** The Java error information saved in the diagnostic area is incomplete. *Action:* Change the schema and retry.
7081	**XRSN_SCHEMA_INCORRECT** The specified schema contains an error that caused an exception. *Action:* Change the schema and retry.
7082	**XRSN_SCHEMA_WARNING** The specified schema contains an error that caused a warning. *Action:* Change the schema and retry.
7083	**XRSN_JAVAERRORMESSAGE_DIAG_FAILED** The Java error information saved in the diagnostic area is not valid. *Action:* Change the schema and retry.
7087	**XRSN_OSR_UNSUPPORTED_FEATURE** An unsupported feature flag was specified. *Action:* Change the schema and retry.
7089	**XRSN_OSR_PARM_NOT_SPECIFIED** No OSR parameter was specified. *Action:* Change the schema and retry.
7091	**XRSN_SCHEMA_PARM_NOT_SPECIFIED** No schema parameter was specified. *Action:* Change the schema and retry.
7093	**XRSN_STRIDTBL_PARM_NOT_SPECIFIED** No stringID table parameter was specified. *Action:* Change the schema and retry.
7095	**XRSN_JAVAPROPERTY_MALFORMED_URL** A well-formed URL could not be constructed for the specified class. *Action:* Change the schema and retry.
7097	**XRSN_JAVAPROPERTY_CLASS_NOTFOUND** The OSR generator classes could not be found. *Action:* Change the schema and retry.
7099	**XRSN_CLSLOADER_ACCESS_FAILED** The OSR generator classes could not be loaded. *Action:* Change the schema and retry.
7101	**XRSN_CLSLOADER_INSTANTIATION_FAILED** The OSR generator classes could not be instantiated. *Action:* Change the schema and retry.

7103 **XRSN_OSR_NOT_LOADED**

No OSRs have been loaded into the OSR generator.

Action: Change the schema and retry.

7107 **XRSN_JVM_OUT_OF_MEMORY**

The Java Virtual Machine is out of memory.

Action: Change the schema and retry.

7109 **XRSN_JVM_STACK_OVERFLOW**

The Java Virtual Machine stack overflow occurrs.

Action: Change the schema and retry.

7111 **XRSN_JVM_INTERNAL_ERROR**

Internal error has occurred in the Java Virtual Machine.

Action: Change the schema and retry.

7113 **XRSN_JVM_UNKNOWN_ERROR**

An unknown and seirous exception has occurred in the JVM.

Action: Change the schema and retry.

Appendix D. Reason Codes Listed by Value

This section describes reason codes, listing them by hexadecimal value and describing actions to correct the error.

8000	**XRSN_XML_QUOTEREQUIREDINENTITYVALUE** An entity value must begin with a single or double quote. *Action:* Change the document or schema to correct and retry.
8001	**XRSN_XML_INVCHARINENTITYVALUE** An invalid XML character was found in the literal entity value. *Action:* Change the document or schema to correct and retry.
8002	**XRSN_XML_INVCHARINSYSTEMID** An invalid XML character was found in a system identifier. *Action:* Change the document or schema to correct and retry.
8003	**XRSN_XML_INVCHARINPUBLICID** An invalid XML character was found in a public identifier. *Action:* Change the document or schema to correct and retry.
8004	**XRSN_XML_INVCHARINDOCTYPEDECL** An invalid XML character was found in a document declaration. *Action:* Change the document or schema to correct and retry.
8005	**XRSN_XML_INVCHARININTERNALSUBSET** An invalid XML character found in the internal subset of the DTD. *Action:* Change the document or schema to correct and retry.
8006	**XRSN_XML_INVCHARINEXTERNALSUBSET** An invalid XML character found in the external subset of the DTD. *Action:* Change the document or schema to correct and retry.
8007	**XRSN_XML_INVCHARINIGNORESECT** An invalid XML character was found in the excluded conditional section. *Action:* Change the document or schema to correct and retry.
8008	**XRSN_XML_QUOTEREQUIREDINSYSTEMID** A system identifier must begin with either a single or double quote. *Action:* Change the document or schema to correct and retry.
8009	**XRSN_XML_SYSTEMIDUNTERMINATED** A system identifier must end with a matching quote. *Action:* Change the document or schema to correct and retry.

8010 **XRSN_XML_QUOTEREQUIREDINPUBLICID**

A public identifier must begin with a single or double quote.

Action: Change the document or schema to correct and retry.

8011 **XRSN_XML_PUBLICIDUNTERMINATED**

A public identifier must end with a matching quote.

Action: Change the document or schema to correct and retry.

8012 **XRSN_XML_PUBIDCHARILLEGAL**

A public identifier character is not permitted.

Action: Change the document or schema to correct and retry.

8013 **XRSN_XML_ENTITYVALUEUNTERMINATED**

The literal value for the entity must end with a matching quote.

Action: Change the document or schema to correct and retry.

8014 **XRSN_XML_SPACEREQDINDECL**

White space is required after DOCTYPE in the document type declaration.

Action: Change the document or schema to correct and retry.

8015 **XRSN_XML_ROOTELEMENTTYPEREQUIRED**

A root element type must appear after DOCTYPE in the document type declaration.

Action: Change the document or schema to correct and retry.

8016 **XRSN_XML_DOCTYPEDECLUNTERMINATED**

A document type declaration for the root element type must end with a ″>″.

Action: Change the document or schema to correct and retry.

8017 **XRSN_XML_PEREFERENCEWITHINMARKUP**

A parameter entity reference cannot occur within markup in the internal subset of the DTD.

Action: Change the document or schema to correct and retry.

8018 **XRSN_XML_PEREFINCOMPLETEMARKUP**

A parameter entity reference cannot occur within the internal subset of the DTD.

Action: Change the document or schema to correct and retry.

8019 **XRSN_XML_MARKUPNORECOGNIZEDINDTD**

The markup declarations contained or pointed to by the document type declaration must be well-formed.

Action: Change the document or schema to correct and retry.

8020	**XRSN_XML_XMLSPACEDECLARATIONILLEGAL** The attribute declaration for xml:space must be given an enumerated type whose only possible values are default and preserve. *Action:* Change the document or schema to correct and retry.
8021	**XRSN_XML_SPACEREQDETYPEINEDECL** A space is required before an element type. *Action:* Change the document or schema to correct and retry.
8022	**XRSN_XML_ETYPEREQDINELEMENTDECL** An element type is required in an element declaration. *Action:* Change the document or schema to correct and retry.
8023	**XRSN_XML_SPACEREQDINELEMENTDEC** White space is required after the element type in the element type declaration. *Action:* Change the document or schema to correct and retry.
8024	**XRSN_XML_CONTENTSPECREQDINEDECL** A constraint is required after the element type in the element type declaration. *Action:* Change the document or schema to correct and retry.
8025	**XRSN_XML_ELEMENTDECLUNTERMINATED** The declaration for an element must end with ″>″. *Action:* Change the document or schema to correct and retry.
8026	**XRSN_XML_OPENPARENORELEREQDINCHIL** A ″(″ or an element type is required in the declaration of an element. *Action:* Change the document or schema to correct and retry.
8027	**XRSN_XML_CLOSEDPARENREQDINCHIL** A ″)″ is required in the declaration. *Action:* Change the document or schema to correct and retry.
8028	**XRSN_XML_ELEMTYPEREQDINMIXEDCON** An element type is required in mixed content. *Action:* Change the document or schema to correct and retry.
8029	**XRSN_XML_CLOSEPARENTREQDINMIXEDCON** A ″)″ is required in the declaration of an element. *Action:* Change the document or schema to correct and retry.
8030	**XRSN_XML_MIXEDCONTENTUNTERMINATED** The mixed content model must end with ″)*″ when the types of child elements are constrained. *Action:* Change the document or schema to correct and retry.

8031	**XRSN_XML_SPACEREQDINATTLISTDECL** White space is required after !ATTLIST in an attribute list declaration. *Action:* Change the document or schema to correct and retry.
8032	**XRSN_XML_ELEMTYPEREQDINATTLISTDECL** An element type is required in an attribute list declaration. *Action:* Change the document or schema to correct and retry.
8033	**XRSN_XML_SPACEREQDINATTDEF** White space is required after !ATTLIST in an attribute list declaration. *Action:* Change the document or schema to correct and retry.
8034	**XRSN_XML_ATTRNAMEREQDINATTDEF** The attribute name must be specified in the attribute list declaration for the element. *Action:* Change the document or schema to correct and retry.
8035	**XRSN_XML_SPACEREQDBATINATTDEF** White space is required before an attribute type in an attribute list declaration. *Action:* Change the document or schema to correct and retry.
8036	**XRSN_XML_ATTTYPEREQDINATTDEF** The attribute type is required in the declaration of the attribute for the element. *Action:* Change the document or schema to correct and retry.
8037	**XRSN_XML_SPACEREQDBDDINATTDEF** White space is required before the default declaration in an attribute list declaration. *Action:* Change the document or schema to correct and retry.
8038	**XRSN_XML_DEFDECLREQDINATTDEF** The attribute default is required in the declaration in an attribute list declaration. *Action:* Change the document or schema to correct and retry.
8039	**XRSN_XML_SPACEREQDANOTINNOTTYPE** White space must follow NOTATION in the attribute declaration. *Action:* Change the document or schema to correct and retry.
8040	**XRSN_XML_OPENPARENREQDINNOTTYPE** The ″(″ character must follow NOTATION in the attribute declaration. *Action:* Change the document or schema to correct and retry.

8041	**XRSN_XML_NAMEREQDINNOTTYPE** The notation name is required in the notation type list for the attribute declaration. *Action:* Change the document or schema to correct and retry.
8042	**XRSN_XML_NOTTYPEUNTERMINATED** The notation type list must end with a ″)″ in the attribute declaration. *Action:* Change the document or schema to correct and retry.
8043	**XRSN_XML_NMTOKREQDINENUM** The name token is required in the enumerated type list for the attribute declaration. *Action:* Change the document or schema to correct and retry.
8044	**XRSN_XML_ENUMUNTERMINATED** The enumerated type list must end with ″)″ in the attribute declaration. *Action:* Change the document or schema to correct and retry.
8045	**XRSN_XML_SPACEREQDINDEFDECL** White space must appear after FIXED in the attribute declaration. *Action:* Change the document or schema to correct and retry.
8046	**XRSN_XML_INCLUDESECTUNTERMINATED** The included conditional section must end with ″″. *Action:* Change the document or schema to correct and retry.
8047	**XRSN_XML_IGNORESECTUNTERMINATED** The excluded conditional section must end with ″″. *Action:* Change the document or schema to correct and retry.
8048	**XRSN_XML_NAMEREQDINPEREF** The entity name must immediately follow the ″%″ in the parameter entity reference. *Action:* Change the document or schema to correct and retry.
8049	**XRSN_XML_SEMICOLONREQDINPEREF** The parameter entity reference must end with the semicolon delimiter. *Action:* Change the document or schema to correct and retry.
8050	**XRSN_XML_SPACEREQDBENINENTITYDECL** White space is required after ″ *Action:* Change the document or schema to correct and retry.
8051	**XRSN_XML_SPACEREQDBPINPEDECL** White space is required between the ″ *Action:* Change the document or schema to correct and retry.

8052	**XRSN_XML_SPACEREQDBEINPEDECL** White space is required between the ″%″ and the entity name in the parameter entity declaration. *Action:* Change the document or schema to correct and retry.
8053	**XRSN_XML_ENTITYNAMEREQINEDECL** The name of the entity is required in the entity declaration. *Action:* Change the document or schema to correct and retry.
8054	**XRSN_XML_SPACEREQDAENAMEINEDECL** White space is required between the entity name and the definition in the entity declaration. *Action:* Change the document or schema to correct and retry.
8055	**XRSN_XML_SPACEREQDBNDATAINUEDECL** White space is required before NDATA in the declaration for the entity. *Action:* Change the document or schema to correct and retry.
8056	**XRSN_XML_SPACEREQDBNNAMEINUEDECL** White space is required between ″NDATA″ and the notation name in the declaration for the entity. *Action:* Change the document or schema to correct and retry.
8057	**XRSN_XML_NOTATIONNAMEREQDINUEDECL** The notation name is required after NDATA in the declaration for the entity. *Action:* Change the document or schema to correct and retry.
8058	**XRSN_XML_ENTITYDECLUNTERMINATED** The declaration for the entity must end with ″>″. *Action:* Change the document or schema to correct and retry.
8059	**XRSN_XML_EXTERNALIDREQD** The external entity declaration must begin with either SYSTEM or PUBLIC. *Action:* Change the document or schema to correct and retry.
8060	**XRSN_XML_SPACEREQDBPLINEXTERNALID** White space is required between PUBLIC and the public identifier. *Action:* Change the document or schema to correct and retry.
8061	**XRSN_XML_SPACEREQDAPLINEXTERNALID** White space is required between the public identifier and the system identifier. *Action:* Change the document or schema to correct and retry.

8062	**XRSN_XML_SPACEREQDBSLINEXTERNALID** White space is required between SYSTEM and the system identifier. *Action:* Change the document or schema to correct and retry.
8063	**XRSN_XML_URIFRAGINSYSTEMID** The fragment identifier should not be specified as part of the system identifier. *Action:* Change the document or schema to correct and retry.
8064	**XRSN_XML_SPACEREQDBNNINNOTATIONDECL** White space is required after *Action:* Change the document or schema to correct and retry.
8065	**XRSN_XML_NOTATIONNAMEREQDINNOTDECL** The name of the notation is required in the notation declaration. *Action:* Change the document or schema to correct and retry.
8066	**XRSN_XML_SPACEREQDANNINNOTATIONDECL** White space is required after the notation name in the notation declaration. *Action:* Change the document or schema to correct and retry.
8067	**XRSN_XML_NOTATIONDECLUNTERMINATED** The declaration for the notation must end with a ″>″. *Action:* Change the document or schema to correct and retry.
8068	**XRSN_XML_UNDECLAREDELEMINCONTSPEC** The content model of the element refers to the undeclared element. *Action:* Change the document or schema to correct and retry.
8069	**XRSN_XML_DUPLICATEATTDEF** There is a duplicate attribute definition found. *Action:* Change the document or schema to correct and retry.
8070	**XRSN_XML_ROOTELEMTMUSTMATCHDOCTDECL** The root element type must match the document type declaration. *Action:* Change the document or schema to correct and retry.
8071	**XRSN_XML_IMPROPERDECLNESTING** The replacement text of a parameter entity must include properly nested declarations. *Action:* Change the document or schema to correct and retry.
8072	**XRSN_XML_WSINELEMCONTENTWHENSA** White space must not occur between elements declared in an external parsed entity with element content in a standalone document. *Action:* Change the document or schema to correct and retry.

8073	**XRSN_XML_REFTOEXTDECLAREDENTWHENSA** The reference to an entity declared in an external parsed entity is not permitted in a standalone document. *Action:* Change the document or schema to correct and retry.
8074	**XRSN_XML_EXTENTITYNOTPERMITED** The reference to an external entity is not permitted in a standalone document. *Action:* Change the document or schema to correct and retry.
8075	**XRSN_XML_ATTVALCHANGEDDURNORMWHENSA** The value of an attribute must not be changed by normalization in a standalone document. *Action:* Change the document or schema to correct and retry.
8076	**XRSN_XML_DEFATTNOTSPECIFIED** An attribute has a default value and must be specified in a standalone document. *Action:* Change the document or schema to correct and retry.
8077	**XRSN_XML_CONTENTINCOMPLETE** The content of an element type is incomplete. *Action:* Change the document or schema to correct and retry.
8078	**XRSN_XML_CONTENTINVALID** The content is invalid. *Action:* Change the document or schema to correct and retry.
8079	**XRSN_XML_ELEMENTNOTDECLARED** An element must be declared. *Action:* Change the document or schema to correct and retry.
8080	**XRSN_XML_ATTRIBUTENOTDECLARED** An attribute must be declared. *Action:* Change the document or schema to correct and retry.
8081	**XRSN_XML_ELEMENTALREADYDECLARED** An element type must not be declared more than once. *Action:* Change the document or schema to correct and retry.
8082	**XRSN_XML_IMPROPERGROUPNESTING** The replacement text of a parameter entity must include properly nested pairs of parentheses. *Action:* Change the document or schema to correct and retry.
8083	**XRSN_XML_DUPTYPEINMIXEDCONTENT** A duplicate type found in a mixed content declaration. *Action:* Change the document or schema to correct and retry.

8084	**XRSN_XML_NOTATIONONEMPTYELEMENT** For compatibility, an attribute of type NOTATION must not be declared on an element declared EMPTY. *Action:* Change the document or schema to correct and retry.
8085	**XRSN_XML_ENTITIESINVALID** Attribute value of type ENTITIES must be the name of one or more unparsed entities. *Action:* Change the document or schema to correct and retry.
8086	**XRSN_XML_ENTITYINVALID** An attribute value of type ENTITY must be the name of an unparsed entity. *Action:* Change the document or schema to correct and retry.
8087	**XRSN_XML_IDDEFTYPEINVALID** An ID attribute must have a declared default of #IMPLIED or #REQUIRED. *Action:* Change the document or schema to correct and retry.
8088	**XRSN_XML_IDINVALID** An attribute value of type ID must be a name. *Action:* Change the document or schema to correct and retry.
8089	**XRSN_XML_IDNOTUNIQUE** An attribute value of type ID must be unique within the document. *Action:* Change the document or schema to correct and retry.
8090	**XRSN_XML_IDREFINVALID** An attribute value of type IDREF must be a name. *Action:* Change the document or schema to correct and retry.
8091	**XRSN_XML_IDREFSINVALID** An attribute value of type IDREFS must be one or more names. *Action:* Change the document or schema to correct and retry.
8092	**XRSN_XML_ATTVALUENOTINLIST** An attribute value is not in the list. *Action:* Change the document or schema to correct and retry.
8093	**XRSN_XML_NMTOKENINVALID** An attribute value of type NMTOKENS must be a name token. *Action:* Change the document or schema to correct and retry.
8094	**XRSN_XML_NMTOKENSINVALID** An attribute value for type NMTOKENS must be one or more name tokens. *Action:* Change the document or schema to correct and retry.

8095	**XRSN_XML_ELEMWITHIDREQD** An element with an ID is required. *Action:* Change the document or schema to correct and retry.
8096	**XRSN_XML_MORETHANONEIDATTR** A second attribute of type ID is not permitted. *Action:* Change the document or schema to correct and retry.
8097	**XRSN_XML_MORETHANONENOTATTR** A second attribute of type NOTATION is not permitted. *Action:* Change the document or schema to correct and retry.
8098	**XRSN_XML_DUPTOKENINLIST** The enumerated type list must not contain duplicate tokens. *Action:* Change the document or schema to correct and retry.
8099	**XRSN_XML_FIXATTVALUEINVALID** A FIXED attribute value is invalid. *Action:* Change the document or schema to correct and retry.
8100	**XRSN_XML_REQDATTNOTSPECIFIED** An attribute is required and must be specific for the element type. *Action:* Change the document or schema to correct and retry.
8101	**XRSN_XML_ATTDEFINVALID** The default value must meet the lexical type constraints declared for the attribute. *Action:* Change the document or schema to correct and retry.
8102	**XRSN_XML_IMPROPERCONDSECTNESTING** The replacement text of the parameter entity must include properly nested conditional sections. *Action:* Change the document or schema to correct and retry.
8103	**XRSN_XML_NOTATIONNOTDECLFORNOTTATT** The notation must be declared when referenced in the notation type list for the attribute. *Action:* Change the document or schema to correct and retry.
8104	**XRSN_XML_NOTATIONNOTDECLFORUPEDECL** The notation must be declared when referenced in the unparsed entity declaration. *Action:* Change the document or schema to correct and retry.
8105	**XRSN_XML_UNIQUENOTNAME** Only one notation declaration can declare a given name. *Action:* Change the document or schema to correct and retry.

8106	**XRSN_XML_REFTOEXTENTITY** The external entity reference is not permitted in an attribute value. *Action:* Change the document or schema to correct and retry.
8107	**XRSN_XML_PENOTDECLARED** The parameter entity was referenced but not declared. *Action:* Change the document or schema to correct and retry.
8108	**XRSN_XML_REFTOUNPENTITY** The unparsed reference is not permitted. *Action:* Change the document or schema to correct and retry.
8109	**XRSN_XML_RECURSIVEREFERENCE** A recursive reference was found. *Action:* Change the document or schema to correct and retry.
8110	**XRSN_XML_RECURSIVEPEREFERENCE** A recursive PE reference was found. *Action:* Change the document or schema to correct and retry.
8111	**XRSN_XML_ENCODINGNOTSUPPORTED** The encoding is not supported in the entity. *Action:* Change the document or schema to correct and retry.
8112	**XRSN_XML_ENCODINGREQD** A parsed entity not encoded in either UTF-8 or UTF-16 must contain an encoding declaration. *Action:* Change the document or schema to correct and retry.
8200	**XRSN_IMP_UNABLETOCONVERTCHAR** Unable to convert an out of range unicode character. *Action:* Change the document or schema to correct and retry.
8201	**XRSN_IMP_INSUFFINPUTTODECCHAR** There is insufficient input to decode the character. *Action:* Change the document or schema to correct and retry.
8202	**XRSN_IMP_MISSING2NDHALFOFPAIR** A surrogate pair is missing its second half for a unicode character. *Action:* Change the document or schema to correct and retry.
8203	**XRSN_IMP_INVAL2NDHALFOFPAIR** An invalid second half of a surrogate pair for a unicode character was found. *Action:* Change the document or schema to correct and retry.

8204 **XRSN_IMP_INVAL1STHALFOFPAIR**

An invalid first half of a surrogate pair for a unicode character was found.

Action: Change the document or schema to correct and retry.

8205 **XRSN_IMP_BOMREQD**

A byte order mark is required.

Action: Change the document or schema to correct and retry.

8206 **XRSN_IMP_INVUTF8SURENCODING**

An invalid UTF-8 surrogate encoding found.

Action: Change the document or schema to correct and retry.

8207 **XRSN_IMP_PARTIALMPCHARSEQ**

A partial multipart character sequence found.

Action: Change the document or schema to correct and retry.

8208 **XRSN_IMP_INCONSISTENTENC**

An encoding name and byte stream contents are inconsistent.

Action: Change the document or schema to correct and retry.

8209 **XRSN_IMP_INVUTF8CHARENC**

An invalid UTF-8 character encoding was found.

Action: Change the document or schema to correct and retry.

8210 **XRSN_IMP_RUNTIMEIOERROR**

A runtime IO error has occurred.

Action: Change the document or schema to correct and retry.

8400 **XRSN_DEM_ROOTELEMENTREQD**

The root element is required in a well-formed document.

Action: Change the document or schema to correct and retry.

8401 **XRSN_DEM_INVCHARINCDSECT**

An invalid XML character was found in the CDATA section of the document.

Action: Change the document or schema to correct and retry.

8402 **XRSN_DEM_INVCHARINCONTENT**

An invalid XML character was found in the element content of the document.

Action: Change the document or schema to correct and retry.

8403 **XRSN_DEM_INVCHARINMISC**

An invalid XML character was found in the markup after the end of the element content.

Action: Change the document or schema to correct and retry.

8404	**XRSN_DEM_INVCHARINPROLOG** An invalid XML character was found in the prolog of a document. *Action:* Change the document or schema to correct and retry.
8405	**XRSN_DEM_CDENDINCONTENT** The character sequence must not appear in content unless used to mark the end of a CDATA section. *Action:* Change the document or schema to correct and retry.
8406	**XRSN_DEM_CDSECTUNTERMINATED** The CDATA section must end with. *Action:* Change the document or schema to correct and retry.
8407	**XRSN_DEM_EQREQDINXMLDECL** The equal character must follow the keyword in the XML declaration. *Action:* Change the document or schema to correct and retry.
8408	**XRSN_DEM_QUOTEREQDINXMLDECL** This value in the XML declaration must be a quoted string. *Action:* Change the document or schema to correct and retry.
8409	**XRSN_DEM_XMLDECLUNTERMINATED** The XML declaration must end with ?>. *Action:* Change the document or schema to correct and retry.
8410	**XRSN_DEM_VERSIONINFOREQD** The version is required in the XML declaration. *Action:* Change the document or schema to correct and retry.
8411	**XRSN_DEM_MARKUPNOTRECINPROLOG** The markup in the document preceding the root element must be well-formed. *Action:* Change the document or schema to correct and retry.
8412	**XRSN_DEM_MARKUPNORECINMISC** The markup in the document following the root element must be well-formed. *Action:* Change the document or schema to correct and retry.
8413	**XRSN_DEM_SDDECLINVALID** The standalone document declaration must be yes or no. *Action:* Change the document or schema to correct and retry.
8414	**XRSN_DEM_ETAGREQD** End-tag is required. *Action:* Change the document or schema to correct and retry.

8415	**XRSN_DEM_ELEMUNTERMINATED** The element must be followed by either attribute specifications, > or />. *Action:* Change the document or schema to correct and retry.
8416	**XRSN_DEM_EQREQDINATTR** The attribute name must be followed by the = character. *Action:* Change the document or schema to correct and retry.
8417	**XRSN_DEM_ATTRNOTUNQ** The attribute was already specified for the element. *Action:* Change the document or schema to correct and retry.
8418	**XRSN_DEM_ETAGUNTERM** The end-tag for the element must end with a > delimiter. *Action:* Change the document or schema to correct and retry.
8419	**XRSN_DEM_MARKUPNORECINCONT** The content of elements must consist of well-formed character data or markup. *Action:* Change the document or schema to correct and retry.
8420	**XRSN_DEM_ELEMENTMISMATCH** The element must start and end within the same entity. *Action:* Change the document or schema to correct and retry.
8421	**XRSN_DEM_INVALCHARINATTRVAL** An invalid XML character was found in the attribute value. *Action:* Change the document or schema to correct and retry.
8422	**XRSN_DEM_INVALCHARINCOMM** An invalid XML character was found in the comment. *Action:* Change the document or schema to correct and retry.
8423	**XRSN_DEM_INVALCHARINPI** An invalid XML character was found in the processing instruction. *Action:* Change the document or schema to correct and retry.
8424	**XRSN_DEM_QUOTEREQDINATTRVAL** The value of an attribute must begin with either a single or double quote character. *Action:* Change the document or schema to correct and retry.
8425	**XRSN_DEM_LESSTHANINATTRVAL** The value of the attribute must not contain the < character. *Action:* Change the document or schema to correct and retry.

8426 **XRSN_DEM_ATTRVALUNTERM**

The attribute value must end with the matching quote character.

Action: Change the document or schema to correct and retry.

8427 **XRSN_DEM_INVALCOMMSTART**

The comment must begin with .

Action: Change the document or schema to correct and retry.

8430 **XRSN_DEM_PITARGETREQD**

The processing instruction must begin with the name of the target.

Action: Change the document or schema to correct and retry.

8431 **XRSN_DEM_SPACEREQDINPI**

A white space character is required between the processing instruction target and the data.

Action: Change the document or schema to correct and retry.

8432 **XRSN_DEM_PIUNTERMINATED**

The processing instruction must end with ?>.

Action: Change the document or schema to correct and retry.

8433 **XRSN_DEM_RESERVEDPITARGET**

The processing instruction target matching [xX][mM][lL] is not allowed.

Action: Change the document or schema to correct and retry.

8434 **XRSN_DEM_VERNOTSUPPORTED**

The XML version specified is not supported.

Action: Change the document or schema to correct and retry.

8435 **XRSN_DEM_DIGREQDINCHARREF**

A decimal representation must immediately follow the &# in the character reference.

Action: Change the document or schema to correct and retry.

8436 **XRSN_DEM_HEXREQDINCHARREF**

A hexadecimal representation must immediately follow the in the character reference.

Action: Change the document or schema to correct and retry.

8437 **XRSN_DEM_SEMICOLONREQDINCHARREF**

The character reference must end with a semicolon delimiter.

Action: Change the document or schema to correct and retry.

8438 **XRSN_DEM_INVCHARREF**

The character reference contains an invalid character.

Action: Change the document or schema to correct and retry.

8439 **XRSN_DEM_NAMEREQDINREF**

The entity name must immediately follow the & in the entity reference.

Action: Change the document or schema to correct and retry.

8440 **XRSN_DEM_SEMICOLONREQDINREF**

The reference to the entity must end with a semicolon delimiter.

Action: Change the document or schema to correct and retry.

8441 **XRSN_DEM_EQREQDINTDECL**

The = character is required in the text declaration.

Action: Change the document or schema to correct and retry.

8442 **XRSN_DEM_QUOTEREQDINTDECL**

The value in the text declaration must be a quoted string.

Action: Change the document or schema to correct and retry.

8443 **XRSN_DEM_SPACEREQDINTDECL**

White space is required between the version and the encoding declaration.

Action: Change the document or schema to correct and retry.

8444 **XRSN_DEM_TEXTDECLUNTERM**

The text declaration must end with ?>.

Action: Change the document or schema to correct and retry.

8445 **XRSN_DEM_ENCDECLREQD**

The encoding is required in the text declaration.

Action: Change the document or schema to correct and retry.

8446 **XRSN_DEM_ENCDECLINV**

The encoding name is invalid.

Action: Change the document or schema to correct and retry.

8447 **XRSN_DEM_ENTNOTDECL**

A general entity was referenced but not declared.

Action: Change the document or schema to correct and retry.

8448 **XRSN_DEM_COLONINNAME**

Namespaces disallow a colon character except in element types or attribute names.

Action: Change the document or schema to correct and retry.

8449 **XRSN_DEM_TWOCOLONSQN**

Namespaces allows only one colon character in element types or attribute names.

Action: Change the document or schema to correct and retry.

8450	**XRSN_DEM_PREFDECL** The namespace prefix was not declared. *Action:* Change the document or schema to correct and retry.
8451	**XRSN_DEM_PREFLEGAL** The namespace name for prefix xml is not bound to a legal namespace name. *Action:* Change the document or schema to correct and retry.
8452	**XRSN_DEM_NSNAMEEMPTY** The namespace name declared for the prefix may not be empty. *Action:* Change the document or schema to correct and retry.
8453	**XRSN_DEM_NSRSRD** The namespace prefix is bound to the reserved namespace name. *Action:* Change the document or schema to correct and retry.
8454	**XRSN_DEM_NSPREFRSRD** The namespace prefix ″xmlns″ must not be declared. *Action:* Change the document or schema to correct and retry.
8600	**XRSN_VME_INVATTVALUE** The attribute value is not valid with respect to its type. *Action:* Change the document or schema to correct and retry.
8601	**XRSN_VME_INVATTVALUEFORFIXED** The attribute value is not valid with respect to its fixed value constraint. *Action:* Change the document or schema to correct and retry.
8602	**XRSN_VME_CONTENTFOREMPTYELEM** The element may not contain any character data or child elements because the element type is EMPTY. *Action:* Change the document or schema to correct and retry.
8603	**XRSN_VME_NONWSCHARINELEMONLYCONT** The element cannot have non-white space character data because the type’s content type is element-only. *Action:* Change the document or schema to correct and retry.
8604	**XRSN_VME_EXPELEMNOMATCH** An expected element match was not found. *Action:* Change the document or schema to correct and retry.
8605	**XRSN_VME_REQDELEMMISSING** The required element or one of its substitutions is required. *Action:* Change the document or schema to correct and retry.

8606	**XRSN_VME_STRICTWCREQTDECL** The matching wildcard is strict, but no declaration can be found for the element. *Action:* Change the document or schema to correct and retry.
8607	**XRSN_VME_EXPECTENDTAG** An end tag is expected. Invalid content is found. No child element is expected at this point. *Action:* Change the document or schema to correct and retry.
8608	**XRSN_VME_ELEMNOTINCHOICE** An unexpected element was found. The element was not one of the choices. *Action:* Change the document or schema to correct and retry.
8609	**XRSN_VME_ELEMDUP** A duplicate element or one of its substitutions was found. *Action:* Change the document or schema to correct and retry.
8610	**XRSN_VME_EMPTYTABINCOMPCONT** An empty element tag is not expected. The content of the element is not complete. *Action:* Change the document or schema to correct and retry.
8611	**XRSN_VME_UNEXPECTEDENDELEM** An unexpected end element event is found. The content of the element is incomplete. *Action:* Change the document or schema to correct and retry.
8612	**XRSN_VME_UNDECLATT** The attribute found is not allowed to appear in the element. *Action:* Change the document or schema to correct and retry.
8613	**XRSN_VME_REQDATTMISSING** The attribute must appear on the element. *Action:* Change the document or schema to correct and retry.
8614	**XRSN_VME_MULTIWILDIDS** ID values must be unique. *Action:* Change the document or schema to correct and retry.
8615	**XRSN_VME_WILDIDFORBID** The attribute is a wildcard ID. But there is already an attribute derived from the ID among the attribute uses. *Action:* Change the document or schema to correct and retry.

8616	**XRSN_VME_NONNILLELEM** Attribute ″xsi:nil″ must not appear on the element, because the nillable property is false. *Action:* Change the document or schema to correct and retry.
8617	**XRSN_VME_NILFORBIDWFIXEDVC** There must be no fixed value constraint for the element because ″xsi:nil″ is specified. *Action:* Change the document or schema to correct and retry.
8618	**XRSN_VME_XSITVALINV** The attribute value ″xsi:type″ of the element is not a valid QName. *Action:* Change the document or schema to correct and retry.
8619	**XRSN_VME_XSITVALDOESNOTEXIST** The value cannot be resolved to a type definition for the element. *Action:* Change the document or schema to correct and retry.
8620	**XRSN_VME_XSITYPEVALNOTALLOWED** The type is not validly derived from the type definition of the element. *Action:* Change the document or schema to correct and retry.
8621	**XRSN_VME_VCINVFORCURTYPE** The value constraint of the element is not a valid default value for the type. *Action:* Change the document or schema to correct and retry.
8622	**XRSN_VME_FIXEDVCFAILURE** The value does not match the fixed value constraint value for the element. *Action:* Change the document or schema to correct and retry.
8623	**XRSN_VME_IDREFMISSINGID** There is no ID/IDREF binding for IDREF. *Action:* Change the document or schema to correct and retry.
8624	**XRSN_VME_ELEMHASABSTYPE** The type definition cannot be abstract for the element. *Action:* Change the document or schema to correct and retry.
8625	**XRSN_VME_INVSIMPLECONT** Invalid value of element. *Action:* Change the document or schema to correct and retry.
8626	**XRSN_VME_DUPKEY** A duplicate key value was declared for an identity constraint. *Action:* Change the document or schema to correct and retry.

8627	**XRSN_VME_DUPUNIQUE** A duplicate unique value was declared for an identity constraint. *Action:* Change the document or schema to correct and retry.
8628	**XRSN_VME_FIELDMULTMATCH** A field matches more than one value within the scope of its selector. The fields must match unique values. *Action:* Change the document or schema to correct and retry.
8629	**XRSN_VME_KEYNOTENOUGHVALS** Not enough values were specified for a key identity constraint. *Action:* Change the document or schema to correct and retry.
8630	**XRSN_VME_IDCKEYREFMISSINGKEY** A keyref is missing a corresponding key. *Action:* Change the document or schema to correct and retry.
8631	**XRSN_VME_ABSELEMERROR** The abstract element cannot be used to validate the element content. *Action:* Change the document or schema to correct and retry.
8632	**XRSN_VME_UNEXPECTEDROOT** The root element is not defined in the schema. *Action:* Change the document or schema to correct and retry.
8800	**XRSN_DVE_SIMPLETYPEINVVAL** Simple type is invalid. *Action:* Change the document or schema to correct and retry.
8801	**XRSN_DVE_IDMULTVAL** There are multiple occurrences of the ID value. *Action:* Change the document or schema to correct and retry.
8802	**XRSN_DVE_FACETLENVAL** The value is not facet-valid with respect to the length for this type. *Action:* Change the document or schema to correct and retry.
8803	**XRSN_DVE_FACETMAXEXCVAL** The value is not facet-valid with respect to maxExclusive for this type. *Action:* Change the document or schema to correct and retry.
8804	**XRSN_DVE_FACETMAXINCVAL** The value is not facet-valid with respect to maxInclusive for this type. *Action:* Change the document or schema to correct and retry.

8805	**XRSN_DVE_FACETMAXLENVAL** The value is not facet-valid with respect to maxLength for this type. *Action:* Change the document or schema to correct and retry.
8806	**XRSN_DVE_FACETMINEXCVAL** The value is not facet-valid with respect to minExclusive for this type. *Action:* Change the document or schema to correct and retry.
8807	**XRSN_DVE_FACETMININCVAL** The value is not facet-valid with respect to minInclusive for this type. *Action:* Change the document or schema to correct and retry.
8808	**XRSN_DVE_FACETMINLENVAL** The value is not facet-valid with respect to minLength for this type. *Action:* Change the document or schema to correct and retry.
8809	**XRSN_DVE_FACETPATTERNVAL** The value is not facet-valid with respect to the pattern for this type. *Action:* Change the document or schema to correct and retry.
8810	**XRSN_DVE_FACETTOTDIGVAL** The value has a mismatch in total number of digits for the type. *Action:* Change the document or schema to correct and retry.
8811	**XRSN_DVE_FACETFRACTDIGVAL** The value has a mismatch in fraction digits for this type. *Action:* Change the document or schema to correct and retry.
8812	**XRSN_DVE_FACETENUMVAL** The value is not facet-valid with respect to the enumeration for this type. It must be a value from the enumeration. *Action:* Change the document or schema to correct and retry.

Appendix E. xsdosrg command reference

Name

xsdosrg - generate an optimized schema representation format

Synopsis

xsdosrg [-v] [-o output_file] [-l list_file] | (input_file [input_file ...])

Note: The l option signifies a lower case L, not an upper case I. The option signifies lower case O, not zero.

Description

A z/OS UNIX shell command that creates an optimized representation from one or more schemas which can be used by the z/OS XML System Services validating parser.

Options

xsdosrg accepts the following command line switches:

-v This option produces verbose output during the generation of the OSR. This is for problem determination purposes only.

-o This option identifies the name of the output file that will contain the generated OSR.

-l This option identifies the list of file names containing the text schemas to process.

Operands

xsdosrg contains the following operands:

input_file
The name of the file containing the text version of an XML schema. At least one input file must be specified, either with this operand, or through the file list operand.

list_file
A list of schema names in text form that will be used to create the optimized schema representation. The text in this file must be in the current local codepage so that the command can open each file in the list.

output_file
The output_file operand is the name of the file that will contain the optimized schema representation. This file name defaults to out.osr if no name is specified.

Example

```
xsdosrg -o myschema.osr myschema.xsd
```

Environment variables

See “Setting up the environment” on page 14 for information on setting and using environment variables.

Localization

xsdosrg uses the following localization environment variables:

- LANG
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MESSAGES
- LC_SYNTAX
- NLSPATH

See Appendix F of the z/OS UNIX System Services Command Reference for descriptions of these environment variables.

Files

None.

Usage notes

One or more schemas may be processed by the **xsdosrg** command into a single optimized schema representation. Multiple schema names may be specified either directly on the command line or using the file list operand with the -l option. Use either the input file operand or the list option to specify a list of schemas to process. Do not use both methods on the same command invocation.

This command provides a simplified interface to the OSR generation utility. See “gxluGenOSR — generate an Optimized Schema Representation (OSR)” on page 70, which allows greater control over the behavior of the generation process and the characteristics of the generated OSR.

The codepage of the text contained in the list file for the -l option is managed in the same way as any other z/OS UNIX System Services command (for example, **cp**). The localization variables above and file tags may be used to set the proper code page so that file names can be handled properly.

Exit values

The following list contains the exit values generated by this command:

0	success
4	no schema specified
16	OSR creation failed

Portability

Not applicable.

Related information

gxluGenOSR is a C routine that also invokes the OSR generator. It provides greater control over the behavior of the generation process and the characteristics of the generated OSR. See “gxluGenOSR — generate an Optimized Schema Representation (OSR)” on page 70 for more information.

Appendix F. C/C++ header files and assembler macros

The z/OS XML System Services API includes several sets of structures, variables and constants that the caller uses to provide input to and receive output from the assorted processing services of the API. These definitions are contained in parallel sets of C/C++ header files and assembler macros. The header files are named gxlh*.h, and are found in the `/usr/include` directory. The assembler macros are named GXLY* and are installed in SYS1.MACLIB.

The names of the C/C++ and assembler macros are similar. For example, the output buffer record mapping is contained in `/usr/include/gxlhxeh.h`, while the assembler version of the same mapping is in SYS1.MACLIB(GXLYXEH). In addition to the parallel nature of these headers and macros, the C/C+ headers come in regular Language Environment run-time and Metal C versions. Both versions have the same file names, but the Language Environment run-time versions are in `/usr/include`, while the Metal C versions are in `/usr/include/metal`. See Chapter 6, "z/OS XML parser API: C/C++," on page 35 for more details about these differences.

All of the core parser services have C/C++ interfaces (both Language Environment C and Metal C) and assembler interfaces. In addition, there are a set of utility services to generate Optimized Schema Representations (OSRs) from text schemas. These utility services are implemented in Language Environment C/C++ and Java. As a result, there are Language Environment C/C++ headers that have no corresponding assembler macro or Metal C version.

These are the header file and assembler macros of the z/OS XML processing API. The header file names are listed first, followed by the assembler macro names in parentheses (if there is a corresponding macro).

gxlhxml.h - main z/OS XML header file

This is the main z/OS XML C/C++ header file that a caller should include in order to use the z/OS XML C/C++ API. It contains prototypes for all of the API entry points, as well as include statements for all of the other header files that are required for the API. The Metal C version of this header also includes logic to call either the 31 or 64 bit version of the requested API, depending on the addressing mode of the caller.

There is no corresponding assembler version of this header file.

gxlhxeh.h (GXLYXEH) - mapping of the output buffer record

This mapping describes the form of the parsed data stream returned from the z/OS XML parser. It contains the following:

- A structure describing the fixed portion of a record in the data stream. This includes the record type and assorted flags describing the characteristics of the record.
- A structure to map the length value pairs (if there are any) that make up the variable portion of the record.
- A structure describing the format of string identifiers (StringIDs) used to represent the strings associated with a record when the StringIDs feature is enabled.

- Structures to map the special records that represent buffer information (data stream metadata), error information, and auxiliary information.

The items defined in this mapping provide a complete interface for the caller to make use of the parsed data stream returned from a parse request. See Chapter 4, “Parsing XML documents,” on page 11for more a more detailed explanation of the z/OS XML parsed data stream.

gxlhxec.h (GXLYXEC) - constants definitions

This header and assembler macro contain constant values that are a key part of the z/OS XML API. They include the following:

- Record/token types. These identify the semantic meaning of a record in the parsed data stream.
- Feature flags. These are the z/OS XML parser features that the caller enables when making an initialization or control request.
- Minimum work area sizes for the z/OS XML parser and query XML declaration services. There are unique minimum work area sizes for the z/OS XML parser, depending on whether or not validation is required.
- The minimum output buffer size.
- The allowable option flag values for the control function service.
- Assorted OSR generator constants.
- CCSID constants for all of the encodings that z/OS XML supports.
- Type identifiers for the data contained in source offset information records.

This is the header (macro) that contains all of the well known and required values for the z/OS XML API.

gxlhqxd.h (GXLYQXD) - mapping of the output from the query XML declaration service

This header (macro) contains the structure that describes the information returned from the Query XML Declaration (QXD) service. It also contains constants that enumerate the allowable values for certain fields of the structure. The types of data returned in this area include the following:

- The type of encoding that the service was able to auto-detect. This is not a CCSID, but an indication as to whether the document is in UCS, UTF, or EBCDIC form. It also gives an indication of whether the document is big-endian or little-endian for certain encoding types.
- The CCSID of the document that the service was able to auto-detect. This value is suitable to pass to the z/OS XML parser initialization service to let the z/OS XML parser know the encoding of the document.

  Note: The QXD service is capable of detecting CCSIDs that are not supported by the z/OS XML parser.
- The version and release number from the ″version″ keyword value in the XML declaration.
- The CCSID from the ″encoding″ keyword value in the XML declaration. It may be the case that the detected encoding does not match the CCSID from the XML declaration. This could happen if the document has been transcoded from the original encoding to the detected encoding. If this is the case, the auto-detected value is the CCSID that should be used when initializing the parser.

- Flags indicating which keyword values in the XML declaration were actually present.
- A flag to indicate how the auto-detected encoding value was determined. In certain cases, it's not possible to actually detect the encoding based on the bytes examined. In this case, the XML spec requires a parser to treat the document as if it were UTF-8 encoded, and this is what the QXD service will provide in the auto-detect value. A flag will be set in the flags field to indicate that the encoding was actually undetected, and that the encoding returned is the default UTF-8 value.
- The overall length of the XML declaration.

See "gxlpQuery — query an XML document" on page 49 or "GXL1QXD (GXL4QXD) — query an XML document" on page 93 for more details about how to acquire and use this data area.

gxlhxd.h (GXLYXD) - mapping of extended diagnostic area

This header (macro) contains the structure describing the extended diagnostic area that is returned when there is a failure in the z/OS XML parser. It is returned whenever the caller requests a control operation through the gxlpControl (GXL1CTL/GXL4CTL) service. The particular area that it is used to map depends on the control operation performed:

- *XEC_CTL_FIN (finish, and reset the parser) – this header (macro) maps the area pointed to directly by the ctl_data_p parameter of the gxlpControl (GXL1CTL/GXL4CTL) service.
- *XEC_CTL_FEAT (reset the parser with different features) – this header (macro) maps the area pointed to by the XFT_XD_PTR field of the GXLHXFT (GXLYXFT) structure.
- *XEC_CTL_LOAD_OSR (reset the parser and load an OSR for validation) – this header (macro) maps the area pointed to by the XOSR_XD_PTR field of the GXLHXOSR (GXLYXOSR) structure.

This mapping contains several types of key information that are of use for problem determination. Some of the more useful fields include the following:

- The address of the main parser anchor block. This is not generally useful for a caller, but is important for IBM service purposes.
- The input and output buffer addresses, and the current offsets into each. This shows which data the z/OS XML parser was processing at the time of the error.
- The size of the last memory allocation request made by the z/OS XML parser.
- Return and reason codes from the last memory allocation request made by the z/OS XML parser.
- Return and reason codes from system service exits (if exits are provided by the caller).
- Return code from the last request to switch to a specialty engine.

gxlhxr.h (GXLYXR) - defines the return codes and reason codes

This contains all of the return and reason codes returned by z/OS XML. Each return and reason code has a descriptive comment. Also included is a reason code mask - *XRSN_REASON_MASK that is used to facilitate access to the low order 2 bytes of the reason code full word.

gxlhxsv.h (GXLYXSV) - mapping of the system service vector

Maps the area used to make assorted exit routines available to the z/OS XML parser. A complete description of the exits that can be specified and how to provide them can be found in Chapter 8, “z/OS XML System Services exit interface,” on page 101.

gxlhxft.h (GXLYXFT) - mapping of the control feature input output area

This structure describes the area that is passed in to and back from the gxlpControl (GXL1CTL/GXL4CTL) service through the ctl_data_p (ctl_data) parameter. It is used to map this area when the caller is changing the parser feature settings by specifying the *XEC_CTL_FEAT value for the ctl_operation (ctl_option) parameter.

This structure includes an integer (fullword) value that contains the required features to reset. There are some features that cannot be reset, and which require that the parse instance to be terminated and re-initialized. This structure also contains the address of a fullword area in which the z/OS XML parser will place a pointer to the extended diagnostic area. This is the area that is mapped by gxlhxd.h (GXLYXD).

See the description of the ctl_data_p parameter in “gxlpControl — perform a parser control function” on page 37, or the ctl_data parameter in “GXL1CTL (GXL4CTL) — perform a parser control function” on page 82 for more details about the use of this structure.

gxlhxosr.h (GXLYXOSR) - mapping of the OSR control area

This structure describes the area that is passed in to and back from the gxlpControl (GXL1CTL/GXL4CTL) service through the ctl_data_p parameter. It should be used to map this area when the caller is loading an OSR for a validating parse by specifying the *XEC_CTL_LOAD_OSR value for the ctl_operation (ctl_option) parameter.

This structure holds the address of a buffer that contains the OSR, plus an optional name string that will be associated with the OSR. This name is currently optional, but it is recommended that every different OSR loaded be given a unique name. This can be useful for problem determination purposes in the event of an error. This structure also contains the address of a fullword area in which the parser will place a pointer to the extended diagnostic area. This is the area that is mapped by gxlhxd.h (GXLYXD).

See the description of the ctl_data_p parameter in “gxlpControl — perform a parser control function” on page 37, or the ctl_data parameter in “GXL1CTL (GXL4CTL) — perform a parser control function” on page 82 for more details about the use of this structure.

gxlhxosrg.h - OSR generator prototypes

This header contains includes for all of the OSR generator utility services, as well as the prototypes for those services. There are no Metal C or assembler macro versions of this header file.

gxlhxosrd.h - mapping of the OSR generator diagnostic area

This header contains the structure that maps the extended diagnostic area returned from the OSR generator utility – similar to the way that gxlhxd.h (GXLYXD) describes the extended diagnostic area returned by the z/OS XML parser. Some of the more useful fields include the following:

- The address of the OSR generator Instance Memory Area (OIMA).
- The last return and reason code issued by the OSR generator.
- The last return and reason code issued by the StringID exit.
- An area containing a Java exception that may have been the cause of the failure. Some of the OSR generator is implemented in Java, so this area will contain the exception information when an error occurs in the Java code.

There are no Metal C or assembler macro versions of this header file.

gxlhxstr.h - StringID table

StringIDs are numeric values that are substituted for certain character strings that are encountered during the parse process. They can save space in the parsed data stream, and possibly improve performance if there are large numbers of repeated strings in the XML document being parsed. This can be the case with documents that make heavy use of namespaces with long URIs.

A caller may specify a StringID exit for the OSR generator to use, such that when a string is encountered, it will call the exit to either generate a new ID, if the string hasn't been seen before, or return an existing ID for strings which have been previously encountered. As the generator acquires these StringIDs, it saves them away in a table, and substitutes them for the strings that they represent within the OSR. The z/OS XML parser implements a similar behavior when it parses an XML document using StringIDs.

It will often be the case that the caller needs to use the same set of StringIDs at OSR generation time, and when a validating parse is performed with that OSR. The OSR generator API contains the gxluGenStrIDTbl service that allows the caller to extract the StringID table from the OSR so that the table can be imported by the StringID exit used during the parse process. See “gxluGenStrIDTbl — generate StringID table from an OSR” on page 73 for more details about how this service works.

This header file contains the structure definitions that describe the format of the StringID table that is exported from the OSR generator. The table is broken down into a fixed portion that contains information about the table, and a variable length portion containing the individual entries of the table. These are the structures that the StringID exit service can use to import the StringID table in preparation for a validating parse.

There are no Metal C or assembler macro versions of this header file.

Appendix G. Java file

The following Java file is provided with z/OS XML System Services.

gxljxr.java - return and reason code declarations

This Java file contains return and reason code declarations for the OSR generator.

```
/***************************************************************
*                                                              *
* Name: GXLJXR.JAVA                                            *
*                                                              *
* Description: OSR generator reason codes                      *
*                                                              *
* LICENSED MATERIALS - PROPERTY OF IBM                         *
*                                                              *
* 5694-A01                                                     *
*                                                              *
*  EXTERNAL CLASSIFICATION: PI                                 *
*  END OF EXTERNAL CLASSIFICATION:                             *
*                                                              *
*  COPYRIGHT IBM CORP. 2007                                    *
*                                                              *
* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                 *
* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP              *
* SCHEDULE CONTRACT WITH IBM CORP.                             *
*                                                              *
* STATUS = HBB7740                                             *
*                                                              *
* Change Activity:                                             *
*                                                              *
*   $LM=VALIDATE,HBB7740,070721,PDGL: OSR Generator Retcode    *
*                                                              *
**************************************************************/
package com.ibm.zos.xml;

/** OSR generator reason codes                              */
public class GXLJXR
{
/*******************************************************************/
/** OSR generator reason codes                                    */
/*******************************************************************/

/** The OIMA provided is unusable. */
public static final int GXLJXRSN_OIMA_NOT_INITIALIZED = 0x7001;
/** The OIMA provided is unusable because a previous
    reset failed. */
public static final int GXLJXRSN_OIMA_NOT_USABLE = 0x7002;
/** The OIMA provided is too small. */
public static final int GXLJXRSN_OIMA_SMALL = 0x7003;
/** The OIMA is already initialized. */
public static final int GXLJXRSN_OIMA_RESIDUAL_DATA = 0x7005;
/** The Java Virtual Machine failed to start. */
public static final int GXLJXRSN_JVM_START_FAILED = 0x7007;
/** The Java Virtual Machine failed to stop. */
public static final int GXLJXRSN_JVM_STOP_FAILED = 0x7008;
/** The operation specified for the control parameter
    is unsupported. */
public static final int GXLJXRSN_CTLOPTN_UNSUPPORTED = 0x7009;
/** Java class not found by the ClassLoader. */
public static final int GXLJXRSN_JAVACLASS_NOT_FOUND = 0x7011;
/** The specified function name is null. */
public static final int GXLJXRSN_FUNC_NAME_NULL = 0x7019;
/** Open for the specified DLL failed. */
public static final int GXLJXRSN_DLL_OPEN_FAILED = 0x7021;
```

```
/** Retrieve for the specified DLL function failed. */
public static final int GXLJXRSN_FUNC_RETRIEVE_FAILED = 0x7023;
/** The Java method cannot be found in the class.
    See the diagnostic area for the method name. */
public static final int GXLJXRSN_JAVA_METHOD_NOT_FOUND = 0x7027;
/** A Java method call failed. */
public static final int GXLJXRSN_JAVA_METHOD_CALL_FAILED = 0x7029;
/** Close for the specified DLL failed. */
public static final int GXLJXRSN_DLL_CLOSE_FAILED = 0x7031;
/** A JNI method returned with an exception. */
public static final int GXLJXRSN_JNI_METHOD_FAILED = 0x7033;
/** Failed to create a new Java object. */
public static final int GXLJXRSN_OBJECT_NOT_CREATED = 0x7035;
/** No schemas have been loaded into the OSR generator. */
public static final int GXLJXRSN_SCHEMA_NOT_LOADED = 0x7037;
/** No OIMA pointer has been specified. */
public static final int GXLJXRSN_OIMAPTR_NOT_PROVIDED = 0x7039;
/** OSR generation failed in the assemble phase. */
public static final int GXLJXRSN_GEN_OSR_ASM_FAILED = 0x7043;
/** OSR generation failed in the compile phase. */
public static final int GXLJXRSN_GEN_OSR_COMP_FAILED = 0x7045;
/** OSR generation failed. */
public static final int GXLJXRSN_GEN_OSR_FAILED = 0x7046;
/** The OSR to load is not valid. */
public static final int GXLJXRSN_OSR_NOT_VALID = 0x7049;
/** The OSR generator could not allocate memory. */
public static final int GXLJXRSN_OSR_MALLOC_FAILED = 0x7050;
/** The OSR generator could not free memory. */
public static final int GXLJXRSN_OSR_MFREE_FAILED = 0x7051;
/** Could not save the Java exception in the diagnostic area. */
public static final int GXLJXRSN_JAVAEXCEPTION_DIAG_FAILED = 0x7055;
/** The Java exception saved in the diagnostic area
    is incomplete. */
public static final int GXLJXRSN_JAVAEXCEPTION_INCOMPLETE = 0x7057;
/** Unable to obtain the reason code set by the Java exception. */
public static final int GXLJXRSN_JAVARSNCODE_NOT_FOUND = 0x7059;
/** The URI specified is incorrect. */
public static final int GXLJXRSN_INCORRECT_SCHEMA_URI = 0x7061;
/** No specific reason code was set by Java. */
public static final int GXLJXRSN_JAVARSNCODE_UNKNOWN = 0x7063;
/** The schema identified by the specified
    URI is not found. */
public static final int GXLJXRSN_SCHEMA_URI_NOT_FOUND = 0x7065;
/** Unable to load the specified schema. */
public static final int GXLJXRSN_SCHEMA_LOAD_FAILED = 0x7067;
/** The OSR identified by the specified
    URI is not found. */
public static final int GXLJXRSN_OSR_URI_NOT_FOUND = 0x7069;
/** The system service parameter specified is null. */
public static final int GXLJXRSN_STRINGID_SYSSVC_NULL = 0x7071;
/** The Java error information saved in the
    diagnostic area is incomplete. */
public static final int GXLJXRSN_JAVAERRORMESSAGE_INCOMPLETE = 0x7079;
/** The specified schema contains an error that caused
    an exception. */
public static final int GXLJXRSN_SCHEMA_INCORRECT = 0x7081;
/** The specified schema contains an error that caused
    a warning. */
public static final int GXLJXRSN_SCHEMA_WARNING = 0x7082;
/** The Java error information saved in the diagnostic
    area is not valid. */
public static final int GXLJXRSN_JAVAERRORMESSAGE_DIAG_FAILED = 0x7083;
/** An unsupported feature flag was specified. */
public static final int GXLJXRSN_OSR_UNSUPPORTED_FEATURE = 0x7087;
/** No OSR parameter was specified. */
public static final int GXLJXRSN_OSR_PARM_NOT_SPECIFIED = 0x7089;
/** No schema parameter was specified. */
```

```
public static final int GXLJXRSN_SCHEMA_PARM_NOT_SPECIFIED = 0x7091;
/** No stringID table parameter was specified. */
public static final int GXLJXRSN_STRIDTBL_PARM_NOT_SPECIFIED = 0x7093;
/** A well-formed URL could not be constructed for
    the specified class. */
public static final int GXLJXRSN_JAVAPROPERTY_MALFORMED_URL = 0x7095;
/** The OSR generator classes could not be found. */
public static final int GXLJXRSN_JAVAPROPERTY_CLASS_NOTFOUND = 0x7097;
/** The OSR generator classes could not be loaded. */
public static final int GXLJXRSN_CLSLOADER_ACCESS_FAILED = 0x7099;
/** The OSR generator classes could not be instantiated. */
public static final int GXLJXRSN_CLSLOADER_INSTANTIATION_FAILED = 0x7101;
/** No OSRs have been loaded into the OSR generator. */
public static final int GXLJXRSN_OSR_NOT_LOADED = 0x7103;
/** The Java Virtual Machine is out of memory. */
public static final int GXLJXRSN_JVM_OUT_OF_MEMORY = 0x7107;
/** The Java Virtual Machine stack overflow occurrs. */
public static final int GXLJXRSN_JVM_STACK_OVERFLOW = 0x7109;
/** Internal error has occurred in the Java Virtual Machine. */
public static final int GXLJXRSN_JVM_INTERNAL_ERROR = 0x7111;
/** An unknown and seirous exception has occurred in the JVM. */
public static final int GXLJXRSN_JVM_UNKNOWN_ERROR = 0x7113;

}
```

Appendix H. Callable services examples - AMODE 31

GXL1CTL example

The following code calls the GXL1CTL service to change the feature bits for the z/OS XML parser. For the callable service, see “GXL1CTL (GXL4CTL) — perform a parser control function” on page 82. AMODE 64 callers use “GXL4CTL example” on page 188.

```
**********************************************************
* Setup parameter list to call GXL1CTL.                  *
*   Then call GXL1CTL.                                   *
**********************************************************
* Call GXL1CTL(PIMA,        (00)
*              CTL_Option,  (04)
*              CTL_Data,    (08)
*              Return_Code, (12)
*              Reason_Code) (16)
*
       LA    R9,SAMPLE_PIMA_PTR
       L     R9,0(R9)
       ST    R9,Parser_Parm
       SLR   R4,R4
       LA    R10,SAMPLE_CTL_OPTION
       ST    R10,Parser_Parm+4
       LA    R10,SAMPLE_CTL_DATA
       ST    R10,Parser_Parm+8
       LA    R10,SAMPLE_CTL_RC
       ST    R10,Parser_Parm+12
       LA    R10,SAMPLE_CTL_RSN
       ST    R10,Parser_Parm+16
*********************************************************
       LLGT  R15,CVTPTR
       L     R15,CVTCSRT-CVT(R15)
       L     R15,72(R15)
       L     R15,28(R15)
       LA    R1,Parser_Parm
       BALR  R14,R15
                    :
*******************************************************************
* Description of the SAMPLE Structure:
* *******************************************************************
SAMPLE                DSECT           Memory storage area
SAMPLE_HEADER         DS   0D
SAMPLE_EYE_CATCHER    DS   CL8        eye-catcher string
SAMPLE_RETCODE        DS   1F
SAMPLE_RSNCODE        DS   1F
SAMPLE_PIMA_PTR       DS   1F
SAMPLE_PIMA_LEN       DS   1F
SAMPLE_INIT_FEAT      DS   1F
SAMPLE_INIT_RC        DS   1F
SAMPLE_INIT_RSN       DS   1F
SAMPLE_CTL_OPTION     DS   1F
SAMPLE_CTL_DATA       DS   1F
SAMPLE_CTL_RC         DS   1F
SAMPLE_CTL_RSN        DS   1F
SAMPLE_TERM_RC        DS   1F
SAMPLE_TERM_RSN       DS   1F
SAMPLE_FLAGS1         DS   1F
SAMPLE_FLAGS2         DS   1F
SAMPLE_END            DS   0X
*******************************************************************
NULL_Value            DC   1D'0'
CCSID                 DS   1F
PARSER_PARM           DS   8A
```

GXL1INI example

The following code initializes the PIMA and records the addresses of the caller's system service routines (if any). For the callable service, see "GXL1INI (GXL4INI) — initialize a parse instance" on page 86. AMODE 64 callers use "GXL4INI example" on page 189.

```
***********************************************************
* Setup parameter list to call GXL1INI.                   *
*   Then call GXL1INI.                                    *
***********************************************************
* Call GXL1INI(PIMA,           (00)
*              PIMA_LEN,       (04)
*              CCSID,          (08)
*              Feature_Flags,  (12)
*              Sys_SVC_Vector, (16) Will be set to NULL
*              Sys_SVC_parm,   (20) Will be set to NULL
*              Return_Code,    (24)
*              Reason_Code)    (28)
*
       LA    R9,SAMPLE_PIMA_PTR
       L     R9,0(R9)
       ST    R9,Parser_Parm
       LA    R10,SAMPLE_PIMA_LEN
       ST    R10,Parser_Parm+4
       SLR   R4,R4
       LA    R10,XEC_ENC_IBM_037(R4)
       ST    R10,CCSID
       LA    R10,CCSID
       ST    R10,Parser_Parm+8
       LA    R10,SAMPLE_INIT_FEAT
       ST    R10,Parser_Parm+12
       LA    R10,NULL_Value
       ST    R10,Parser_Parm+16
       ST    R10,Parser_Parm+20
       LA    R10,SAMPLE_INIT_RC
       ST    R10,Parser_Parm+24
       LA    R10,SAMPLE_INIT_RSN
       ST    R10,Parser_Parm+28
**********************************************************
       LLGT  R15,CVTPTR
       L     R15,CVTCSRT-CVT(R15)
       L     R15,72(R15)
       L     R15,16(R15)
       LA    R1,Parser_Parm
       BALR  R14,R15
                    :
*******************************************************************
* Description of the SAMPLE Structure:
* *******************************************************************
SAMPLE               DSECT           Memory storage area
SAMPLE_HEADER        DS   0D
SAMPLE_EYE_CATCHER   DS   CL8        eye-catcher string
SAMPLE_RETCODE       DS   1F
SAMPLE_RSNCODE       DS   1F
SAMPLE_PIMA_PTR      DS   1F
SAMPLE_PIMA_LEN      DS   1F
SAMPLE_INIT_FEAT     DS   1F
SAMPLE_INIT_RC       DS   1F
SAMPLE_INIT_RSN      DS   1F
SAMPLE_CTL_OPTION    DS   1F
SAMPLE_CTL_DATA      DS   1F
SAMPLE_CTL_RC        DS   1F
SAMPLE_CTL_RSN       DS   1F
SAMPLE_TERM_RC       DS   1F
SAMPLE_TERM_RSN      DS   1F
SAMPLE_FLAGS1        DS   1F
```

```
SAMPLE_FLAGS2       DS   1F
SAMPLE_END          DS   0X
********************************************************************
NULL_Value          DC   1D'0'
CCSID               DS   1F
PARSER_PARM         DS   8A
```

GXL1PRS example

The following code parses a buffer of XML text and places the result in an output buffer. For the callable service, see “GXL1PRS (GXL4PRS) — parse a buffer of XML text” on page 90. AMODE 64 callers use “GXL4PRS example” on page 191.

```
*/*****************************************************************************************
*/   PARSE
*/*****************************************************************************************
*   CALL GXL1PRS(PIMA,OPTION_FLAGS,INBUF_PTR,INBUF_LEN,OUTBUF_PTR,
*       OUTBUF_LEN,RC,RSN);
         L     @02,PARM_PTR(,@03_PARM_PTR_PTR)
         L     @10,PIMA_PTR(,@02)
         ST    @10,@AL00001
         LA    @10,OPTION_FLAGS(,@02)
         ST    @10,@AL00001+4
         LA    @10,INBUF_PTR(,@02)
         ST    @10,@AL00001+8
         LA    @10,INBUF_LEN(,@02)
         ST    @10,@AL00001+12
         LA    @10,OUTBUF_PTR(,@02)
         ST    @10,@AL00001+16
         LA    @02,OUTBUF_LEN(,@02)
         ST    @02,@AL00001+20
         LA    @10,RC
         ST    @10,@AL00001+24
         LA    @02,RSN
         ST    @02,@AL00001+28
         OI    @AL00001+28,X'80'
         L     @10,CS$CVT
         L     @02,CS$CSRT+544(,@10)
         L     @10,CS$CSRFT+72(,@02)
         L     @15,GXLST31+20(,@10)
         LA    @01,@AL00001
         BALR  @14,@15
*   PARSE_RC = RC;
         L     @02,PARM_PTR(,@03_PARM_PTR_PTR)
         L     @10,RC
         ST    @10,PARSE_RC(,@02)
*   PARSE_RSN = RSN;
         L     @10,RSN
         ST    @10,PARSE_RSN(,@02)
*   END DO_PARSE;
*
```

GXL1TRM example

The following code releases all resources obtained (including storage) by the z/OS XML parser and resets the PIMA so that it can be re-initialized. For the callable service, see “GXL1TRM (GXL4TRM) — terminate a parse instance” on page 96. AMODE 64 callers use “GXL4TRM example” on page 192.

```
**********************************************************
* Setup parameter list to call GXL1TRM.                 *
*    Then call GXL1TRM.                                  *
**********************************************************
* Call GXL1TRM(PIMA,        (00)
*              Return_Code, (04)
*              Reason_Code) (08)
*
       LA    R10,SAMPLE_PIMA_PTR
       L     R10,0(R10)
       ST    R10,Parser_Parm
       LA    R10,SAMPLE_TERM_RC
       ST    R10,Parser_Parm+4
       LA    R10,SAMPLE_TERM_RSN
       ST    R10,Parser_Parm+8
*********************************************************
       LLGT  R15,CVTPTR
       L     R15,CVTCSRT-CVT(R15)
       L     R15,72(R15)
       L     R15,24(R15)
       LA    R1,Parser_Parm
       BALR  R14,R15
                   :
*******************************************************************
* Description of the SAMPLE Structure:
* *******************************************************************
SAMPLE               DSECT           Memory storage area
SAMPLE_HEADER        DS   0D
SAMPLE_EYE_CATCHER   DS   CL8        eye-catcher string
SAMPLE_RETCODE       DS   1F
SAMPLE_RSNCODE       DS   1F
SAMPLE_PIMA_PTR      DS   1F
SAMPLE_PIMA_LEN      DS   1F
SAMPLE_INIT_FEAT     DS   1F
SAMPLE_INIT_RC       DS   1F
SAMPLE_INIT_RSN      DS   1F
SAMPLE_CTL_OPTION    DS   1F
SAMPLE_CTL_DATA      DS   1F
SAMPLE_CTL_RC        DS   1F
SAMPLE_CTL_RSN       DS   1F
SAMPLE_TERM_RC       DS   1F
SAMPLE_TERM_RSN      DS   1F
SAMPLE_FLAGS1        DS   1F
SAMPLE_FLAGS2        DS   1F
SAMPLE_END           DS   0X
*******************************************************************
NULL_Value           DC   1D'0'
CCSID                DS   1F
PARSER_PARM          DS   8A
```

Appendix I. Callable services examples - AMODE 64

GXL4CTL example

The following code calls the GXL4CTL service to change the feature bits for the z/OS XML parser. For the callable service, see “GXL1CTL (GXL4CTL) — perform a parser control function” on page 82. AMODE 31 callers use “GXL1CTL example” on page 182.

```
**********************************************************
* Setup parameter list to call GXL4CTL.                  *
*    Then call GXL4CTL.                                  *
**********************************************************
* Call GXL4CTL(PIMA,           (00)
*              CTL_Option,     (08)
*              CTL_Data,       (16)
*              Return_Code,    (24)
*              Reason_Code)    (32)
*
       LA    R9,SAMPLE_PIMA_PTR
       LG    R9,0(R9)
       STG   R9,Parser_Parm
       SLGR  R4,R4
       LA    R10,SAMPLE_CTL_OPTION
       STG   R10,Parser_Parm+8
       LA    R10,SAMPLE_CTL_DATA
       STG   R10,Parser_Parm+16
       LA    R10,SAMPLE_CTL_RC
       STG   R10,Parser_Parm+24
       LA    R10,SAMPLE_CTL_RSN
       STG   R10,Parser_Parm+32
*********************************************************
       LLGT  R15,CVTPTR
       L     R15,CVTCSRT-CVT(R15)
       L     R15,72(R15)
       LG    R15,64(R15)
       LA    R1,Parser_Parm
       BALR  R14,R15
                    :
********************************************************************
* Description of the SAMPLE Structure:
* ********************************************************************
SAMPLE                DSECT           Memory storage area
SAMPLE_HEADER         DS   0D
SAMPLE_EYE_CATCHER    DS   CL8        eye-catcher string
SAMPLE_RETCODE        DS   1F
SAMPLE_RSNCODE        DS   1F
SAMPLE_PIMA_PTR       DS   1D
SAMPLE_PIMA_LEN       DS   1F
SAMPLE_INIT_FEAT      DS   1F
SAMPLE_INIT_RC        DS   1F
SAMPLE_INIT_RSN       DS   1F
SAMPLE_CTL_OPTION     DS   1F
SAMPLE_CTL_DATA       DS   1F
SAMPLE_CTL_RC         DS   1F
SAMPLE_CTL_RSN        DS   1F
SAMPLE_TERM_RC        DS   1F
SAMPLE_TERM_RSN       DS   1F
SAMPLE_FLAGS1         DS   1F
SAMPLE_FLAGS2         DS   1F
SAMPLE_END            DS   0X
********************************************************************
NULL_Value            DC   1D'0'
CCSID                 DS   1F
PARSER_PARM           DS   16A
```

GXL4INI example

The following code initializes the PIMA and records the addresses of the caller's system service routines (if any). For the callable service, see "GXL1INI (GXL4INI) — initialize a parse instance" on page 86. AMODE 31 callers use "GXL1INI example" on page 183.

```
**********************************************************
* Setup parameter list to call GXL4INI.                  *
*    Then call GXL4INI.                                  *
**********************************************************
* Call GXL4INI(PIMA,          (00)
*             PIMA_LEN,       (08)
*             CCSID,          (16)
*             Feature_Flags,  (24)
*             Sys_SVC_Vector, (32) Will be set to NULL
*             Sys_SVC_parm,   (40) Will be set to NULL
*             Return_Code,    (48)
*             Reason_Code)    (56)
*
       LA    R9,SAMPLE_PIMA_PTR
       LG    R9,0(R9)
       STG   R9,Parser_Parm
       LA    R10,SAMPLE_PIMA_LEN
       STG   R10,Parser_Parm+8
       SLGR  R4,R4
       LA    R10,XEC_ENC_IBM_037(R4)
       ST    R10,CCSID
       LA    R10,CCSID
       STG   R10,Parser_Parm+16
       LA    R10,SAMPLE_INIT_FEAT
       STG   R10,Parser_Parm+24
       LA    R10,NULL_Value
       STG   R10,Parser_Parm+32
       STG   R10,Parser_Parm+40
       LA    R10,SAMPLE_INIT_RC
       STG   R10,Parser_Parm+48
       LA    R10,SAMPLE_INIT_RSN
       STG   R10,Parser_Parm+56
*********************************************************
       LLGT  R15,CVTPTR
       L     R15,CVTCSRT-CVT(R15)
       L     R15,72(R15)
       LG    R15,40(R15)
       LA    R1,Parser_Parm
       BALR  R14,R15
                    :
*******************************************************************
* Description of the SAMPLE Structure:
* *******************************************************************
SAMPLE               DSECT           Memory storage area
SAMPLE_HEADER        DS   0D
SAMPLE_EYE_CATCHER   DS   CL8        eye-catcher string
SAMPLE_RETCODE       DS   1F
SAMPLE_RSNCODE       DS   1F
SAMPLE_PIMA_PTR      DS   1D
SAMPLE_PIMA_LEN      DS   1F
SAMPLE_INIT_FEAT     DS   1F
SAMPLE_INIT_RC       DS   1F
SAMPLE_INIT_RSN      DS   1F
SAMPLE_CTL_OPTION    DS   1F
SAMPLE_CTL_DATA      DS   1F
SAMPLE_CTL_RC        DS   1F
SAMPLE_CTL_RSN       DS   1F
SAMPLE_TERM_RC       DS   1F
SAMPLE_TERM_RSN      DS   1F
SAMPLE_FLAGS1        DS   1F
```

```
SAMPLE_FLAGS2         DS   1F
SAMPLE_END            DS   0X
*********************************************************************
NULL_Value            DC   1D'0'
CCSID                 DS   1F
PARSER_PARM           DS   16A
```

GXL4PRS example

The following code parses a buffer of XML text and places the result in an output buffer. For the callable service, see “GXL1PRS (GXL4PRS) — parse a buffer of XML text” on page 90. AMODE 31 callers use “GXL1PRS example” on page 185.

```
*/*****************************************************            */
*/*  DO_PARSE                                                       */
*/*****************************************************            */
*
*DO_PARSE:
*   PROCEDURE;
DO_PARSE STM   @14,@12,@SA00004
         STMH  @14,@12,@SH00004
*   CALL GXL4PRS(PIMA,OPTION_FLAGS,INBUF_PTR,INBUF_LEN,OUTBUF_PTR,
*       OUTBUF_LEN,RC,RSN);
         LG    @02,PARM_PTR(,@03_PARM_PTR_PTR)
         LG    @10,PIMA_PTR(,@02)
         STG   @10,@AX00001
         LA    @10,OPTION_FLAGS(,@02)
         STG   @10,@AX00001+8
         LA    @10,INBUF_PTR(,@02)
         STG   @10,@AX00001+16
         LA    @10,INBUF_LEN(,@02)
         STG   @10,@AX00001+24
         LA    @10,OUTBUF_PTR(,@02)
         STG   @10,@AX00001+32
         LA    @02,OUTBUF_LEN(,@02)
         STG   @02,@AX00001+40
         LA    @10,RC
         STG   @10,@AX00001+48
         LA    @02,RSN
         STG   @02,@AX00001+56
         L     @10,CS$CVT
         LLGTR @10,@10
         L     @02,CS$CSRT+544(,@10)
         LLGTR @02,@02
         L     @10,CS$CSRFT+72(,@02)
         LLGTR @10,@10
         LG    @15,GXLST64+48(,@10)
         LA    @01,@AX00001
         BASR  @14,@15
*   PARSE_RC = RC;
         LG    @02,PARM_PTR(,@03_PARM_PTR_PTR)
         L     @10,RC
         ST    @10,PARSE_RC(,@02)
*   PARSE_RSN = RSN;
         L     @10,RSN
         ST    @10,PARSE_RSN(,@02)
*   END DO_PARSE;
*
@EL00004 DS    0H
@EF00004 DS    0H
@ER00004 LMH   @14,@12,@SH00004
         LM    @14,@12,@SA00004
         BR    @14
```

GXL4TRM example

The following code releases all resources obtained (including storage) by the z/OS XML parser and resets the PIMA so that it can be re-initialized. For the callable service, see “GXL1TRM (GXL4TRM) — terminate a parse instance” on page 96. AMODE 31 callers use “GXL1TRM example” on page 186.

```
**********************************************************
* Setup paramter list to call GXL4TRM.                   *
*    Then call GXL4TRM.                                  *
**********************************************************
* Call GXL4TRM(PIMA,        (00)
*              Return_Code, (08)
*              Reason_Code) (16)
*
       LA    R10,SAMPLE_PIMA_PTR
       LG    R10,0(R10)
       STG   R10,Parser_Parm
       LA    R10,SAMPLE_TERM_RC
       STG   R10,Parser_Parm+8
       LA    R10,SAMPLE_TERM_RSN
       STG   R10,Parser_Parm+16
*********************************************************
       LLGT  R15,CVTPTR
       L     R15,CVTCSRT-CVT(R15)
       L     R15,72(R15)
       LG    R15,56(R15)
       LA    R1,Parser_Parm
       BALR  R14,R15
*********************************************************
                   :
********************************************************************
* Description of the SAMPLE Structure:
* ********************************************************************
SAMPLE               DSECT           Memory storage area
SAMPLE_HEADER        DS   0D
SAMPLE_EYE_CATCHER   DS   CL8        eye-catcher string
SAMPLE_RETCODE       DS   1F
SAMPLE_RSNCODE       DS   1F
SAMPLE_PIMA_PTR      DS   1D
SAMPLE_PIMA_LEN      DS   1F
SAMPLE_INIT_FEAT     DS   1F
SAMPLE_INIT_RC       DS   1F
SAMPLE_INIT_RSN      DS   1F
SAMPLE_CTL_OPTION    DS   1F
SAMPLE_CTL_DATA      DS   1F
SAMPLE_CTL_RC        DS   1F
SAMPLE_CTL_RSN       DS   1F
SAMPLE_TERM_RC       DS   1F
SAMPLE_TERM_RSN      DS   1F
SAMPLE_FLAGS1        DS   1F
SAMPLE_FLAGS2        DS   1F
SAMPLE_END           DS   0X
********************************************************************
NULL_Value           DC   1D'0'
CCSID                DS   1F
PARSER_PARM          DS   16A
```

Appendix J. Exit examples - AMODE 31

GXLE1FRM (GXLFST31 example)

Restrictions: The following restrictions apply to this example:

- This sample was designed to be a basic example of a memory service exit, and was not designed with other system considerations in mind, such as the z/OS XML parser running in cross memory mode, SRB mode, or in a different key, for instance.
- This sample is not designed to work with any other service exits. The exit workarea is assumed to be used by this memory service exit only. (Note that both GXLGST31 and GXLFST31 services are considered as one service exit). As a result, this memory service exit can only work independently, with no other service exits running.

The following code frees an area of memory passed by the z/OS XML parser. For the exit service, see “GXLFST31 (GXLFST64) — free memory” on page 107. AMODE 64 callers use “GXLE4FRM (GXLFST64 example)” on page 224.

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example:

SYS_SVC_PARM
: Address of storage area that the caller of the z/OS XML parser wants to pass on to the exit.

MEMORY_LEN
: Contain the length of the memory area requested to be free.

The following output variables are used in the example:

MEMORY_ADDR
: The address of the memory to be freed.

EXIT_DIAG_CODE
: Contains diagnostic information.

  XSM_DC_INVALID_EYECATCHER_STR
  : Eye catcher is incorrect.

  XSM_DC_FAIL_FREE_MEM31
  : Fail to release storage memory.

RETCODE

  XSM_RC_FAILURE
  : Unable to free memory

  XSM_RC_SUCCESS
  : The storage macro released the allocated memory successfully (greater than zero if deallocation failed).

EXIT_DIAG_CODE

  XSM_DC_INVALID_EYECATCHER_STR
  : Eye catcher is incorrect.

  XSM_DC_FAIL_FREE_MEM31
  : Fail to release storage memory.

```
* SYSSTATE is used to initialize some variables used by storage macro
         SYSSTATE ASCENV=P,AMODE64=,ARCHLVL=,OSREL=
         SYSSTATE ASCENV=,AMODE64=,ARCHLVL=2,OSREL=
GXLE1FRM CSECT                      Set up control section
GXLE1FRM AMODE 31                   Address mode of module
GXLE1FRM RMODE ANY                  Residence of module
         USING DATA+0,R12           Relative Branching
         ENTRY GXLE1FRM             external entry point
         STM   R14,R12,12(R13)      Save regs
         LARL  R12,DATA             Load DATA Address

         J     MSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE1FRM'
MSTART   DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R1
* Establish addressability for the XSM DSECT to the SYS_SVC_PARM
         USING XSM,R2

*********************************************************
* Check if the XSM eye catcher string matches correctly *
*********************************************************
         L     R2,SYS_SVC_PARM_PTR
         L     R2,SYS_SVC_PARM(,R2)

         CLC   XSM_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSM structure, so we not *
* sure if the data in the XSM structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         L     R8,RETCODE_PTR
         LHI   R9,XSM_RC_FAILURE
         ST    R9,RETCODE(,R8)
         L     R10,EXITDIAGCODE_PTR
         LHI   R9,XSM_DC_INVALID_EYECATCHER_STR
         ST    R9,EXIT_DIAG_CODE(,R10)
         J     EXIT

**********************************************************
* The XSM structure is intact and eye catcher is correct *
* Set the dynamic area pointer to register 11            *
**********************************************************
PROCEED  DS    0H

*********************************************************
* Setting up the parameter list passed in by the caller *
*********************************************************
         XR    R7,R7
         ST    R7,RETCODE(,R8)
         ST    R7,XSM_DIAG_CODE

*****************************************************
* Call the Storage Release macro to free the memory *
*****************************************************
         L     R7,MEMORY_LEN_PTR
         L     R7,MEMORY_LEN(,R7)
         L     R5,MEMORY_ADDR_PTR
         L     R5,MEMORY_ADDR(,R5)

         DROP  R1
         LR    R8,R1  Backup Reg1 Parameter address
```

```
* After storage macro finished, register 15 will contain the
* return code of the macro.  Zero if deallocation is successful,
* or greater than zero if failed to deallocate the memory
         STORAGE RELEASE,COND=YES,LENGTH=(R7),ADDR=(R5)

************************************************
* Make sure the storage release is successful *
************************************************
         LR    R1,R8  Restore Reg1 Parameter address

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R1

         L     R7,RETCODE_PTR
         ST    R15,RETCODE(,R7)   reg15 is the macro return code
         XR    R9,R9
         CR    R15,R9
         JE    EXIT

******************************
* Memory deallocation failed *
******************************
         LHI   R10,XSM_DC_FAIL_FREE_MEM31
         L     R9,EXITDIAGCODE_PTR
         ST    R10,EXIT_DIAG_CODE(,R9)

*************************************************
* EXITING THE FREE MEMORY SERVICE EXIT ROUTINE *
*************************************************
QUITMEM  DS    0H
         ST    R10,XSM_DIAG_CODE

EXIT     DS    0H
         DROP  R1,R2
         LM    R14,R12,12(R13)
         BSM   0,R14

DATA     DS    0F
         DC    A(GXLE1FRM)
EYECHAR  DC    CL8'XSMEYECA'

*******************************************************
* The end of the Free Memory Service Exit subroutine *
*******************************************************

*********************************************************************
* Description of the XSM Structure:                                 *
*   The XSM structure is including the XSM Header, eye catcher     *
*   string, and the dynamic area pointer that points to the empty  *
*   storage area space for the dynamic storage area                *
*      o XSM Header: Contain all sub-variables within the XSM      *
*                    structure before the dynamic area             *
*      o Eye Catcher: A string used to identify if this chunk of   *
*                     memory is allocated and referenced correctly *
*      o Dynamic Area:  managed by the dynamic area pointer and is *
*                       used for storing local variables and uses  *
*********************************************************************
XSM                 DSECT           Memory storage area
XSM_HEADER          DS    0D
XSM_EYE_CATCHER     DS    CL8       eye-catcher string
XSM_DIAG_CODE       DS    1F        exit diagnostic code
XSM_TOTAL_SIZE      DS    1F        sys_svc_parm total size
XSM_NULL            DS    1F
XSM_DYN_AREA_PTR31  DS    1F        31bit address of dynamic area
XSM_HEADER_END      DS    0X

*********************************************************************
```

```
* Description of the Parameters:                                    *
*     Input: SYS_SVC_PARM - Address that was passed to the parser  *
*                           at initialization time                  *
*            MEMORY_LEN - Contain the length of the memory area    *
*                         requested to be free                      *
*     Output: MEMORY_ADDR - The Address of the  memory to be freed *
*             EXIT_DIAG_CODE - Contain diagnostic information      *
*             RETCODE - XSM_RC_FAILURE if unable to free memory    *
*             EXIT_DIAG_CODE - XSM_DC_INVALID_EYECATCHER_STR       *
*                              XSM_DC_FAIL_FREE_MEM31              *
********************************************************************
PARMLIST           DSECT
SYS_SVC_PARM_PTR   DS    1F
MEMORY_ADDR_PTR    DS    1F
MEMORY_LEN_PTR     DS    1F
EXITDIAGCODE_PTR   DS    1F
RETCODE_PTR        DS    1F
RSNCODE_PTR        DS    1F

SYS_SVC_PARM       EQU   0
MEMORY_ADDR        EQU   0
MEMORY_LEN         EQU   0
EXIT_DIAG_CODE     EQU   0
RETCODE            EQU   0
RSNCODE            EQU   0

R0                 EQU   0
R1                 EQU   1
R2                 EQU   2
R3                 EQU   3
R4                 EQU   4
R5                 EQU   5
R6                 EQU   6
R7                 EQU   7
R8                 EQU   8
R9                 EQU   9
R10                EQU   10
R11                EQU   11
R12                EQU   12
R13                EQU   13
R14                EQU   14
R15                EQU   15

********************************************************************
* EXIT DIAGNOSTIC CODE                                             *
********************************************************************
* Eye Catcher string is incorrect
XSM_DC_INVALID_EYECATCHER_STR   EQU   1
* Memory length passed into subroutine is out of bound
XSM_DC_FAIL_FREE_MEM31          EQU   5

********************************************************************
* RETURN CODE                                                      *
********************************************************************
XSM_RC_FAILURE     EQU   1
                   END
```

GXLE1GTM (GXLGST31 example)

Restrictions: The following restrictions apply to this example:

- This sample was designed to be a basic example of a memory service exit, and was not designed with other system considerations in mind, such as the z/OS XML parser running in cross memory mode, SRB mode, or in a different key, for instance.
- This sample is not designed to work with any other service exits. The exit workarea is assumed to be used by this memory service exit only. (Note that both GXLGST31 and GXLFST31 services are considered as one service exit). As a result, this memory service exit can only work independently, with no other service exits running.

The following code allocates an area of memory of the size requested by the z/OS XML parser. For the exit service, see “GXLGST31 (GXLGST64) — get memory” on page 104. AMODE 64 callers use “GXLE4GTM (GXLGST64 example)” on page 228.

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example:

SYS_SVC_PARM
: Address that was passed to the z/OS XML parser at initialization time.

MEMORY_LEN
: Contains the length of the memory area requested by the z/OS XML parser.

The following output variables are used in the example:

MEMORY_ADDR
: The address of the allocated memory.

EXIT_DIAG_CODE
: Contains diagnostic information.

  XSM_DC_INVALID_EYECATCHER_STR
  : Eye catcher is incorrect.

  XSM_DC_INVALID_GET_MEM_LEN
  : Memory length is out of bound.

  XSM_DC_FAIL_ALLOCATE_MEM31
  : Storage memory allocation failed.

RETCODE

  XSM_RC_FAILURE
  : Unable to allocate memory.

  XSM_RC_SUCCESS
  : The storage macro allocated the memory successfully (greater than zero if allocation failed).

```
* SYSSTATE is used to initialize some variables used by storage macro
         SYSSTATE ASCENV=P,AMODE64=,ARCHLVL=,OSREL=
         SYSSTATE ASCENV=,AMODE64=,ARCHLVL=1,OSREL=
GXLE1GTM CSECT                      Set up control section
GXLE1GTM AMODE 31                   Address mode of module
```

```
GXLE1GTM RMODE ANY                  Residence of module
         USING DATA+0,R12           Relative Branching
         ENTRY GXLE1GTM             external entry point
         STM   R14,R12,12(R13)      Save regs
         LARL  R12,DATA             Load DATA Address

         J     MSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE1GTM'
MSTART   DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R1
* Establish addressability for the XSM DSECT to the SYS_SVC_PARM
         USING XSM,R2

*********************************************************
* Check if the XSM eye catcher string matches correctly *
*********************************************************
         L     R2,SYS_SVC_PARM_PTR
         L     R2,SYS_SVC_PARM(,R2)

         CLC   XSM_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSM structure, so we not *
* sure if the data in the XSM structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         L     R8,RETCODE_PTR
         LHI   R9,XSM_RC_FAILURE
         ST    R9,RETCODE(,R8)
         L     R10,EXITDIAGCODE_PTR
         LHI   R9,XSM_DC_INVALID_EYECATCHER_STR
         ST    R9,EXIT_DIAG_CODE(,R10)
         J     EXIT

**********************************************************
* The XSM structure is intact and eye catcher is correct *
**********************************************************
PROCEED  DS    0H

*********************************************************
* Setting up the parameter list passed in by the caller *
*********************************************************
         XR    R3,R3
         L     R4,RETCODE_PTR
         ST    R3,RETCODE(,R4)
         ST    R3,XSM_DIAG_CODE

*****************************************************
* Check to see if a valid memory length is provided *
*****************************************************
         L     R3,MEMORY_LEN_PTR
         ICM   R7,15,MEMORY_LEN(R3)
         JP    ALLOCMEM


*********************************************************
* Incorrect memory length is found, exiting subroutine *
*********************************************************
         LHI   R8,XSM_RC_FAILURE
         ST    R8,RETCODE(,R4)    reg4 is RETCODE_PTR
         L     R9,EXITDIAGCODE_PTR
         LHI   R8,XSM_DC_INVALID_GET_MEM_LEN
         ST    R8,EXIT_DIAG_CODE(,R9)
         J     QUITMEM

**************************************************************
* Start allocating memory and check if the allocation is OK *
**************************************************************
ALLOCMEM DS    0H
         DROP  R1
         LR    R8,R1  Backup Reg1 Parameter address

* After storage macro finished, register 15 will contain the
* return code of the macro.  Zero if allocation is successful,
* or greater than zero if failed to allocate memory
         STORAGE OBTAIN,COND=YES,LENGTH=(R7),ADDR=(R10)

***************************************************************
* Check if allocated memory returns a successful return code *
***************************************************************
         LR    R1,R8  Restore Reg1 Parameter address

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R1

         L     R9,MEMORY_ADDR_PTR
         ST    R10,MEMORY_ADDR(,R9)
         L     R7,RETCODE_PTR
         ST    R15,RETCODE(,R7)   reg15 is the macro return code
         XR    R4,R4
         CR    R15,R4
         JE    EXIT

************************************************
* Memory allocation failed, exiting subroutine *
************************************************
         LHI   R8,XSM_DC_FAIL_ALLOCATE_MEM31
         L     R9,EXITDIAGCODE_PTR
         ST    R8,EXIT_DIAG_CODE(,R9)

***********************************************
* EXITING THE GET MEMORY SERVICE EXIT ROUTINE *
***********************************************
QUITMEM  DS    0H
         ST    R8,XSM_DIAG_CODE

EXIT     DS    0H
         DROP  R1,R2
         LM    R14,R12,12(R13)
         BSM   0,R14

DATA     DS    0F
         DC    A(GXLE1GTM)
EYECHAR  DC    CL8'XSMEYECA'

*****************************************************
* The end of the Get Memory Service Exit subroutine *
*****************************************************


*******************************************************************
* Description of the XSM Structure:                               *
*   The XSM structure is including the XSM Header, eye catcher     *
*   string, and the dynamic area pointer that points to the empty *
*   storage area space for the dynamic storage area               *
*      o XSM Header: Contain all sub-variables within the XSM      *
*                    structure before the dynamic area            *
*      o Eye Catcher: A string used to identify if this chunk of   *
*                     memory is allocated and referenced correctly *
*      o Dynamic Area:  managed by the dynamic area pointer and is *
*                       used for storing local variables and uses  *
********************************************************************
XSM                 DSECT          Memory storage area
XSM_HEADER          DS    0D
XSM_EYE_CATCHER     DS    CL8      eye-catcher string
XSM_DIAG_CODE       DS    1F       exit diagnostic code
XSM_TOTAL_SIZE      DS    1F       sys_svc_parm total size
XSM_NULL            DS    1F
XSM_DYN_AREA_PTR31  DS    1F       31bit address of dynamic area
XSM_HEADER_END      DS    0X

*********************************************************************
* Description of the Parameters:                                    *
*     Input: SYS_SVC_PARM - Address of storage area that the caller *
*                       of the parser wants to pass on to the exit *
*            MEMORY_LEN - Contain the length of the memory area     *
*                        requested by the parser                    *
*     Output: MEMORY_ADDR - The Address of the allocated memory     *
*             EXIT_DIAG_CODE - Contain diagnostic information       *
*                               XSM_DC_INVALID_EYECATCHER_STR       *
*                               XSM_DC_INVALID_GET_MEM_LEN          *
*                               XSM_DC_FAIL_ALLOCATE_MEM31          *
*             RETCODE - XSM_RC_FAILURE if failed to allocate memory *
*********************************************************************
PARMLIST            DSECT
SYS_SVC_PARM_PTR    DS    1F
MEMORY_ADDR_PTR     DS    1F
MEMORY_LEN_PTR      DS    1F
EXITDIAGCODE_PTR    DS    1F
RETCODE_PTR         DS    1F
RSNCODE_PTR         DS    1F

SYS_SVC_PARM        EQU   0
MEMORY_ADDR         EQU   0
MEMORY_LEN          EQU   0
EXIT_DIAG_CODE      EQU   0
RETCODE             EQU   0

R0                  EQU   0
R1                  EQU   1
R2                  EQU   2
R3                  EQU   3
R4                  EQU   4
R5                  EQU   5
R6                  EQU   6
R7                  EQU   7
R8                  EQU   8
R9                  EQU   9
R10                 EQU   10
R11                 EQU   11
R12                 EQU   12
R13                 EQU   13
R14                 EQU   14
R15                 EQU   15

********************************************************************
* EXIT DIAGNOSTIC CODE                                             *
********************************************************************
* Eye Catcher string is incorrect
XSM_DC_INVALID_EYECATCHER_STR   EQU   1
* Memory length passed into subroutine is out of bound
XSM_DC_INVALID_GET_MEM_LEN      EQU   2
* Memory allocation failed to allocate storage in gxle1gtm
XSM_DC_FAIL_ALLOCATE_MEM31      EQU   3

********************************************************************
```

```
* RETURN CODE                                                      *
********************************************************************
XSM_RC_FAILURE      EQU   1
                    END
```

GXLSYM31 example

Restrictions: The following restrictions apply to this example:

- This example was designed to be a basic example of a StringID service exit. It was not designed with other system considerations in mind, such as the z/OS XML parser running in cross memory mode, SRB mode, or in a different key, for instance.
- This example is not designed to work with any other service exits. The exit workarea is assumed to be used by this StringID service exit only. As a result, this StringID service exit can only work independently, with no other service exits running.

Note: This exit example is divided into the following 3 modules:

- “GXLE1INI” on page 204
- “GXLE1IDI (GXLSYM31 example module)” on page 209
- “GXLE1IDR” on page 217

For the exit service, see “GXLSYM31 (GXLSYM64) — StringID service” on page 109. AMODE 64 callers use “GXLSYM64 example” on page 234.

GXLE1INI

This example module does the following:

- Validates the caller specification and determines whether to use user defined or default values for storage size.
- Initializes all variables in XSI. (XSI is the data structure for the StringID sample exit).

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example module assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example:

STRID_AREA_ADDR
Address of the XSI storage area.

STRID_AREA_LEN
Total length of the XSI Storage area.

STRID_MAX_NUM
The maximum number of StringIDs allowed.

SYM_MAX_SIZE
The maximum string length for each symbol.

The following output variables are used in this example module:

RETCODE

XSI_RC_FAILURE
If the storage area failed to initialize.

XSI_RC_SUCCESS
If the storage area successfully initialized.

DIAG_CODE
Contains diagnostic information.

XSI_DC_SYMBOL_STORAGE_TOO_SMALL
Storage size is too small.

```
GXLE1INI CSECT                  Set up control section
GXLE1INI AMODE 31               Address mode of module
GXLE1INI RMODE ANY              Residence of module
         USING DATA+0,R12       Relative Branching
         ENTRY GXLE1INI         external entry point
         BAKR  R14,0            Save register/psw status
         LARL  R12,DATA         Load DATA Address

         J     IDSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE1INI'
IDSTART  DS    0H

*********************************************************************
*** MAINLINE CODE STARTS HERE                                     ***
*********************************************************************
*** GXLE1INI.PLX - INITIALIZE XSI MACROS' DSECTS                  ***
*********************************************************************

* below will calculate the minimum storage area size allowed
* inorder for StringID service exit to function correctly
* XSI_DYN_AREA_SIZE + length(XSI_HEADER) + length(XSI_NODE_HEADER)
* Accumulating the minimum required size for storage area to
* initialize into register 9
         LHI   R9,XSI_DYN_AREA_SIZE
         AHI   R9,XSI_HEADER_SIZE
         AHI   R9,XSI_NODE_HEADER_SIZE
         AHI   R9,1  Minimum of one byte of string to be stored

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R1

* Check if the allocation size is enough or not
         L     R7,STRID_AREA_LEN_PTR
         L     R4,STRID_AREA_LEN(,R7)
* Check if the user given storage area size has enough space
         CLR   R4,R9   reg9 is the accumulated value for minimum size
         JNL   MAXVALUE

* The storage size is too small for the StringID initialization
         LHI   R6,XSI_RC_FAILURE
         L     R5,RET_CODE_PTR
         ST    R6,RET_CODE(,R5)
         LHI   R6,XSI_DC_SYMBOL_STORAGE_TOO_SMALL
         L     R5,DIAG_CODE_PTR
         ST    R6,DIAG_CODE(,R5)
         J     EXITING

MAXVALUE DS    0H

* see if caller provided a maximum number of StringID allowed in
* this StringID service exit samples, if not provided, a default
* value will be used instead
         L     R5,STRID_MAX_NUM_PTR
         ICM   R2,15,STRID_MAX_NUM(R5)
         JZ    ID_DEF

* A valid StringID maximum number is provided
         LR    R5,R2   reg2 is user defined max num of strid allowed
         J     SYM_MAX

* StringID maximum number is not provided, so default value is used
ID_DEF   DS    0H
         LHI   R5,XSI_DEFAULT_MAX_ID#

* see if caller provided a maximum string size allowed in this
* StringID service exit samples, if not provided, a default
* string size will be used instead
SYM_MAX  DS    0H
         L     R8,SYM_MAX_SIZE_PTR
         ICM   R6,15,SYM_MAX_SIZE(R8)
         JZ    SYM_DEF

* A valid maximum string size is provided
         LR    R2,R6   reg6 is user defined maximum string length
         J     INITIAL

* The default value of the maximum string size is used
SYM_DEF  DS    0H
         LHI   R2,XSI_DEFAULT_SYM_MAX_SIZE

* Begin to initialize the StringID storage area
INITIAL  DS    0H

* Establish addressability for the XSI DSECT to the SYS_SVC_PARM
         USING XSI,R3
         L     R9,STRID_AREA_ADDR_PTR
         L     R3,STRID_AREA_ADDR(,R9)

* Initialize all StringID Storage area header values
         LR    R7,R5     reg5 is maximum number of StringID allowed
         MVC   XSI_EYE_CATCHER,EYECHAR
         ST    R2,XSI_SYM_MAX_SIZE   reg2 is max length of each string
         SLR   R8,R8
         ST    R8,XSI_DIAG_CODE

* StringID value starts at number one index
         LHI   R9,1
         ST    R9,XSI_NEXT_ID#
         ST    R7,XSI_MAX_ID#   reg7 is max number of StringID allowed
         ST    R4,XSI_TOTAL_SIZE   reg6 is StringID storage area size

* Setting up the dynamic area pointer
         LR    R2,R7   reg7 is maximum number of StringID allowed
         MHI   R2,XSI_LENGTH_ID_LIST    multiply by length(XSI_ID_LIST)
         AHI   R2,XSI_HEADER_SIZE

* add 7 then bitwise and with '7FFFFFF8' to ensure the dynamic
* area is starting at double word boundary
         LHI   R5,7
         ALR   R5,R3  reg3 is the StringID storage area address
         ALR   R5,R2  reg2 is size of XSI_header + max # of StringID
         N     R5,DW_BDRY

         ST    R5,XSI_DYN_AREA31

* current free pointer is after all XSI header, StringID
* array list, and dynamic area
         LHI   R7,XSI_DYN_AREA_SIZE
         AL    R7,XSI_DYN_AREA31
         ST    R7,XSI_CURR_FREE@31

* Calculate the amount of free space that is left in the storage area
         LR    R2,R3   reg3 is the StringID storage area address
         LCR   R2,R2
         AL    R2,XSI_CURR_FREE@31
         SLR   R4,R2   reg6 is the total StringID storage area size
         ST    R4,XSI_FREE_SPACE

* Set the XSI_TREE_HEAD@ to NULL, which means the tree is empty
         ST    R8,XSI_TREE_HEAD@31   reg8 is set to zero/NULL

         LHI   R8,XSI_RC_SUCCESS
         L     R5,RET_CODE_PTR
         ST    R8,RET_CODE(,R5)
         J     EXITING

***************************************************************
*** EXITING THE StringID INITIALIZATION ROUTINE             ***
***************************************************************
EXITING  DS    0H
         DROP  R1,R3
         PR

DATA     DS    0F
         DC    A(GXLE1INI)
DW_BDRY  DC    X'7FFFFFF8'
EYECHAR  DC    CL8'XSIEYECA'


R0                 EQU   0
R1                 EQU   1
R2                 EQU   2
```

```
R3                  EQU   3
R4                  EQU   4
R5                  EQU   5
R6                  EQU   6
R7                  EQU   7
R8                  EQU   8
R9                  EQU   9
R10                 EQU   10
R11                 EQU   11
R12                 EQU   12
R13                 EQU   13
R14                 EQU   14
R15                 EQU   15

*********************************************************************
* Description of the XSI Structure:                                 *
*   The XSI structure is including the XSI Header, StringID array  *
*   list, the dynamic area, and end with empty storage area space  *
*   for the StringID tree                                           *
*      o XSI Header: Contain information for the XSI Structure to  *
*                    operate correctly (space management)           *
*      o StringID list: List of pointers that indexed by StringID  *
*                       and each ptr point to the corresponding    *
*                       leaf node within the StringID tree         *
*      o Dynamic Area: Used by ASAENTRY where all local variables, *
*                      stacks, and storage usage will take up the  *
*                      storage space within the dynamic area       *
*      o Empty Storage: managed by the XSI header and is used for  *
*                       storing the tree structure for StringID    *
*********************************************************************
XSI                 DSECT         StringID storage area
XSI_HEADER          DS    0D
XSI_EYE_CATCHER     DS    CL8     eye-catcher string
XSI_SYM_MAX_SIZE    DS    1F      max string len for symbol
XSI_DIAG_CODE       DS    1F      exit diagnostic code
XSI_NEXT_ID#        DS    1F      next usable StringID num
XSI_MAX_ID#         DS    1F      max num of StringID nodes
XSI_TOTAL_SIZE      DS    1F      sys_svc_parm total size
XSI_FREE_SPACE      DS    1F      space left for data
XSI_CURR_FREE       DS    1F
XSI_CURR_FREE@31    DS    1F      31bit current @ of free space
XSI_TREE_NULL       DS    1F
XSI_TREE_HEAD@31    DS    1F      31bitlocation of tree head
XSI_DYN_AREA        DS    1F
XSI_DYN_AREA31      DS    1F      31bitaddress of dynamic area
XSI_HEADER_END      DS    0X
XSI_ID_LIST_NULL    DS    1F
XSI_ID_LIST31       DS    1F      31bit ID list for quick search

XSI_HEADER_SIZE     EQU   XSI_HEADER_END-XSI_HEADER

*********************************************************************
* Description of the Parameters:                                    *
*   Input: STRID_AREA_PTR - Pointer to the XSI storage area         *
*          STRID_AREA_LEN - total length of the XSI Storage area    *
*          STRID_MAX_NUM - the maximum number of StringID allowed   *
*          SYM_MAX_SIZE - maximum string length for each symbol     *
*   Output: DIAG_CODE - XSI_DC_XXX provide detail to retcode        *
*           RETCODE - XSI_RC_FAILURE if string cannot be inserted   *
*                     XSI_RC_SUCCESS if string inserted ok          *
*********************************************************************
PARMLIST             DSECT
STRID_AREA_ADDR_PTR  DS    1F
STRID_AREA_LEN_PTR   DS    1F
STRID_MAX_NUM_PTR    DS    1F
SYM_MAX_SIZE_PTR     DS    1F
RET_CODE_PTR         DS    1F
```

```
DIAG_CODE_PTR        DS    1F

*******************************************************************
* EXIT DIAGNOSTIC CODE                                            *
*******************************************************************
* the storage size is too small for initialization
XSI_DC_SYMBOL_STORAGE_TOO_SMALL        EQU    17

* Default Maximum Constant values
XSI_DYN_AREA_SIZE            EQU   4096
XSI_DEFAULT_MAX_ID#          EQU   800
XSI_DEFAULT_SYM_MAX_SIZE     EQU   256

*******************************************************************
* RETURN CODE                                                     *
*******************************************************************
XSI_RC_SUCCESS       EQU   0
XSI_RC_FAILURE       EQU   1

STRID_AREA_ADDR      EQU   0
STRID_AREA_LEN       EQU   0
STRID_MAX_NUM        EQU   0
SYM_MAX_SIZE         EQU   0
RET_CODE             EQU   0
DIAG_CODE            EQU   0

XSI_NODE_HEADER_SIZE EQU   24
XSI_LENGTH_ID_LIST   EQU   8
                     END
```

GXLE1IDI (GXLSYM31 example module)

This example module does the following:

- Search for an identical string in the tree.
- Inserts a string into a tree and returns a unique StringID. This is done as follows:
  1. Check first to make sure the length of the string is within the maximum symbol buffer size.
  2. Inserts the string into the root if the tree is empty or searches down the tree to find the appropriate empty leaf node.
  3. When the insert node location is found, it's address will be passed to the INSERT_STRING subroutine. The subroutine will create a new leaf node and then insert the string.
  4. Return the StringID if the string inserted successfully.

Note: This is the actual exit pointed to in the SYS_SVC_VECTOR table.

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example module assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example module:

SYS_SVC_PARM
Address of storage area that the caller of the z/OS XML parser wants to pass to the exit. It also contains the XSI structure information.

STR The string that will be inserted into the tree.

STRLEN
Length of the current string needed to be inserted. Length is derived from the number of bytes of the characters in the string.

CCSID
Identifier for the string's character set.

The following output variables are used in the example module:

STRID The index of the inserted or found string.

EXIT_DIAG_CODE
Contains diagnostic information.

XSI_DC_INCORRECT_PARM_STRLEN
String length is out of bound.

XSI_DC_OUT_OF_STORAGE_SPACE
Allocated storage is full.

XSI_DC_INCORRECT_EYE_CATCHER
Eye catcher is incorrect.

XSI_DC_MAX_OUT_ID_LIST_ENTRIES
StringID list is full.

RETCODE

XRC_FAILURE
Failed to insert or search for *STR*.

XRC_SUCCESS
String was inserted or found.

```
GXLE1IDI CSECT                   Set up control section
GXLE1IDI AMODE 31                Address mode of module
GXLE1IDI RMODE ANY               Residence of module
         USING DATA+0,R12        Relative Branching
         ENTRY GXLE1IDI          external entry point
         STM   R14,R12,12(R13)   Save regs
         LARL  R12,DATA          Load DATA Address

         J     IDSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE1IDI'
IDSTART  DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R7
* Establish addressability for the XSI DSECT to the SYS_SVC_PARM
         USING XSI,R6

*********************************************************
* Check if the XSI eye catcher string matches correctly *
*********************************************************
         LR    R7,R1
         L     R3,SYS_SVC_PARM_PTR
         L     R6,SYS_SVC_PARM(,R3)
         CLC   XSI_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSI structure, so we not *
* sure if the data in the XSI structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         LHI   R5,XSI_RC_FAILURE
         L     R9,RETCODE_PTR
         ST    R5,RETCODE(,R9)
         LHI   R5,XSI_DC_INCORRECT_EYE_CATCHER
         L     R9,DIAGCODE_PTR
         ST    R5,EXIT_DIAG_CODE(,R9)
         J     EXITIDI

*********************************************************************
*** MAINLINE CODE STARTS HERE:                                    ***
*********************************************************************
*** GXLE1IDI.PLX - Insert/Search String & return StringID         ***
*********************************************************************

**********************************************************
* The XSI structure is intact and eye catcher is correct *
* Set the dynamic area pointer to register 11            *
**********************************************************
PROCEED  DS    0H
         L     R11,XSI_DYN_AREA31
         USING DATD,R11

*********************************************************************
* - If string length and string contain the valid information      *
*   then gxle1idi will proceed and continue to insert the string   *
*   into the tree and return the appropriate StringID value       *
*********************************************************************
         L     R3,STRLEN_PTR
         ICM   R8,15,STRLEN(R3)   Check if string length is zero
         JNP   LEN_INV
* Check if strlen is greater then the maximum allowed string length
         CL    R8,XSI_SYM_MAX_SIZE
         JH    LEN_INV

* If the StringID list is not empty, search before insert
         ICM   R2,15,XSI_TREE_HEAD@31
         JNZ   SEARCH_TREE

* If StringID list is empty, insert string
         LA    R9,XSI_TREE_HEAD@31
         ST    R9,INSERTION_ADDR_PTR
         J     INSERT_STRING

* Value of the string length is invalid, exiting the subroutine
LEN_INV  DS    0H
         LHI   R5,XSI_RC_FAILURE
         L     R3,RETCODE_PTR
         ST    R5,RETCODE(,R3)
         LHI   R5,XSI_DC_INCORRECT_PARM_STRLEN
         L     R3,DIAGCODE_PTR
         ST    R5,EXIT_DIAG_CODE(,R3)
         J     EXITING

*********************************************************************
*                                                                   *
* INSERT_STRING - Insert string into tree and return the StringID  *
*                                                                   *
*         The string will be pushed into the tree and the index    *
*         of the list will be saved as an unique StringID,         *
*         immediately upon the procedure checked the valid string  *
*         and string Length data.                                  *
*                                                                   *
* Input: STR_PTR - The string that will be inserted into the tree  *
*                                                                   *
*        STR_LEN - The length of the string that is loaded into R8 *
*                                                                   *
*        INSERTION_ADDR_PTR - address of a pointer to the location *
*                             where the new node will go           *
*                                                                   *
*        XSI - user control area header variables                  *
*                                                                   *
* Output: STRID - the index of the inserted string                 *
*                                                                   *
*         RETCODE - XRC_FAILURE if string cannot be inserted       *
*                   XRC_SUCCESS if string inserted ok              *
*                                                                   *
*********************************************************************
INSERT_STRING DS    0H
* plus 1 is for adding a NULL character at the end of each
* string so user can know when the end of string reaches
         LR    R9,R8     reg8 is the string length
         LHI   R10,25    24 is length of XSI_NODE_HEADER, 1 is NULL
         ALR   R9,R10
         ST    R9,NEW_NODE_SIZE

* check if the StringID reaches it's maximum allowed value
         L     R3,XSI_NEXT_ID#
         CL    R3,XSI_MAX_ID#
         JNH   ID_OK

* The next available StringID has reached the maximum ID limit
         LHI   R5,XSI_RC_FAILURE
         L     R3,RETCODE_PTR
         ST    R5,RETCODE(,R3)
         LHI   R5,XSI_DC_MAX_OUT_ID_LIST_ENTRIES
         L     R3,DIAGCODE_PTR
         ST    R5,EXIT_DIAG_CODE(,R3)
         J     EXITING
```

```
ID_OK    DS    0H
* check if the storage area has enough space for the new node
         L     R5,XSI_FREE_SPACE
         C     R5,NEW_NODE_SIZE
         JNL   SPACE_OK

* Amount of free space is not enough to store the new node
         LHI   R5,XSI_RC_FAILURE
         L     R3,RETCODE_PTR
         ST    R5,RETCODE(,R3)
         LHI   R5,XSI_DC_OUT_OF_STORAGE_SPACE
         L     R3,DIAGCODE_PTR
         ST    R5,EXIT_DIAG_CODE(,R3)
         J     EXITING

* If XSI Storage contain enough free space and StringID hasn't
* reach the limit yet, then start inserting the string into the
* empty leaf node
SPACE_OK DS    0H
* Establish addressability for the XSI_NODE DSECT to the StringID Tree
         USING XSI_NODE,R9

         L     R9,XSI_CURR_FREE@31

* Set the left/right node to NULL
         XR    R4,R4
         ST    R4,XSI_NODE_LEFT31
         ST    R4,XSI_NODE_RIGHT31

* Assign the next available StringID to the node
         LR    R4,R3       reg3 contains the StringID
         ST    R4,XSI_NODE_StringID
         ST    R8,XSI_NODE_STRLEN    Insert the string length

* Move the string from the parameter into the tree node
         L     R14,STRID_PTR
         ST    R4,STRID(,R14)    reg4 is next available StringID #
         LA    R10,XSI_NODE_STRING
         L     R2,STR_PTR
         LR    R4,R8             reg8 is the string length

* check the string length to see if it's greater than 256 characters
         CHI   R4,256
         JNH   STR_256

* MVC the string in loops for each 256 characters
MVC_STR  DS    0H
         MVC   0(256,R10),0(R2)  copy 256 chars at a time
         LHI   R14,256
         ALR   R10,R14       move Reg10 256chars to right
         ALR   R2,R14        move Reg2 256chars to right
         SLR   R4,R14        Subtract string length by 256
         CHI   R4,256        check if string length still > 256
         JH    MVC_STR

* Copy string into XSI tree node only if the string length or what
* is left to copy from the string is less than 256 characters
STR_256  DS    0H
         LR    R15,R4
         BCTR  R15,R0
         EX    R15,STRCOPY

* add null at the end of the string for higher level programming uses
         XR    R4,R4
         STC   R4,XSI_NODE_STRING(R8)   reg8 is original string length

* Setting the address of the new tree node to the StringID list
```

```
         LR    R4,R3    reg3 is the current insertion StringID #
         SLL   R4,3
         ST    R9,XSI_ID_LIST31-8(R4)

* re-calculate the total amount of the XSI free space
         L     R10,NEW_NODE_SIZE
         SLR   R5,R10    reg5 is the old free space
         ST    R5,XSI_FREE_SPACE

* Increment the XSI_NEXT_ID# value by one
         LHI   R15,1
         ALR   R3,R15
         ST    R3,XSI_NEXT_ID#

* assign parent node pointer to point to this new node
         L     R3,INSERTION_ADDR_PTR
         ST    R9,INSERTION_PTR(,R3)
         ALR   R9,R10
         ST    R9,XSI_CURR_FREE@31

* Insertion of a new string to the tree is completed
         LHI   R5,XSI_RC_SUCCESS
         L     R3,RETCODE_PTR
         ST    R5,RETCODE(,R3)
         L     R3,DIAGCODE_PTR
         L     R5,EXIT_DIAG_CODE(,R3)
         J     EXITING

*******************************************************************
*                                                                 *
* SEARCH_TREE - search the tree for input string and return the   *
*               StringID if found or call INSERT_STRING to create *
*               a new node for the new string                     *
*                                                                 *
*         Iteratively going down the tree base on the string to   *
*         determine going left or right leaf. If no identical     *
*         string is found, then INSERT_STRING will be called or   *
*         the identified string will be returned with its StringID *
*                                                                 *
* Input: The input parameters passed to the program.              *
*                                                                 *
* Output: STRID - the index of the inserted or found string       *
*                                                                 *
*         RETCODE - XRC_SUCCESS if string is found within tree    *
*                                                                 *
*******************************************************************
SEARCH_TREE DS    0H
         LR    R9,R2   reg2 is the current searching node address

* Establish addressability for the XSI_NODE DSECT to the StringID Tree
         USING XSI_NODE,R9

         L     R0,STR_PTR
         LG    R1,STRLEN_PTR
LOOP_IN  DS    0H
* string compare and decide to branch toward left or right leaf
         LA    R14,XSI_NODE_STRING
         L     R15,XSI_NODE_STRLEN

* CLCL requires to use the registers in pair, so R0 contains the
* string characters while R1 requires to contain the string length.
* So CLCL will knows how many characters in the string to be compared
         CLCL  R0,R14
         JH    RIGHT_TREE
         JL    LEFT_TREE

* String is found in the StringID tree and its corresponding
* StringID will be saved and returned
         L     R4,STRID_PTR
         L     R10,XSI_NODE_StringID
         ST    R10,STRID(,R4)
         LHI   R5,XSI_RC_SUCCESS
         L     R3,RETCODE_PTR
         ST    R5,RETCODE(,R3)
         L     R3,DIAGCODE_PTR
         L     R5,EXIT_DIAG_CODE(,R3)
         J     EXITING

LEFT_TREE DS    0H
* Check if the left leaf is empty, if empty then go insert string,
* if not empty, then branch to the left leaf and continue searching
         ICM   R2,15,XSI_NODE_LEFT31
         JNZ   GO_LEFT

* Left leaf node is empty, so insert the new string into it
         LA    R10,XSI_NODE_LEFT31
         ST    R10,INSERTION_ADDR_PTR
         J     INSERT_STRING

GO_LEFT  DS    0H
         LR    R9,R2
         J     LOOP_IN

RIGHT_TREE DS    0H
* Check if the right leaf is empty, if empty then go insert string,
* if not empty, then branch to the right leaf and continue searching
         ICM   R2,15,XSI_NODE_RIGHT31
         JNZ   GO_RIGHT

* Right leaf node is empty, so insert the new string into it
         LA    R10,XSI_NODE_RIGHT31
         ST    R10,INSERTION_ADDR_PTR
         J     INSERT_STRING

GO_RIGHT DS    0H
         LR    R9,R2
         J     LOOP_IN

****************************************************************
*** EXITING THE StringID SERVICE EXIT ROUTINE                ***
****************************************************************
EXITING  DS    0H
         ST    R2,XSI_DIAG_CODE

EXITIDI  DS    0H
         DROP  R6,R7,R9
         LM    R14,R12,12(R13)
         BSM   0,R14

DATA     DS    0F
         DC    A(GXLE1IDI)
STRCOPY  MVC   0(0,R10),0(R2)
EYECHAR  DC    CL8'XSIEYECA'



DATD     DSECT
         DS    36F
         DS    0D
* Size of new node being inserted: node header + var string length
NEW_NODE_SIZE DS F
* insertion pointer is needed when new node is being inserted and
* their parent node's ptr address will be saved in this insertion
* pointer. When the new node is created and initialized with data,
* the address in insertion_ptr will be assigned to point to the
* new node, completing the insertion process of new node into tree
INSERTION_ADDR_PTR DS A

R0                 EQU   0
R1                 EQU   1
R2                 EQU   2
R3                 EQU   3
R4                 EQU   4
R5                 EQU   5
R6                 EQU   6
R7                 EQU   7
R8                 EQU   8
R9                 EQU   9
R10                EQU   10
R11                EQU   11
R12                EQU   12
R13                EQU   13
R14                EQU   14
R15                EQU   15

*********************************************************************
* Description of the XSI Structure:                                 *
*   The XSI structure is including the XSI Header, StringID array  *
*   list, the dynamic area, and end with empty storage area space  *
*   for the StringID tree                                           *
*      o XSI Header: Contain information for the XSI Structure to  *
*                    operate correctly (space management)           *
*      o StringID list: List of pointers that indexed by StringID  *
*                       and each ptr point to the corresponding    *
*                       leaf node within the StringID tree          *
*      o Dynamic Area: Used by ASAENTRY where all local variables, *
*                      stacks, and storage usage will take up the  *
*                      storage space within the dynamic area       *
*      o Empty Storage: managed by the XSI header and is used for  *
*                       storing the tree structure for StringID    *
*********************************************************************
XSI                DSECT         StringID storage area
XSI_HEADER         DS    0D
XSI_EYE_CATCHER    DS    CL8     eye-catcher string
XSI_SYM_MAX_SIZE   DS    1F      max string len for symbol
XSI_DIAG_CODE      DS    1F      exit diagnostic code
XSI_NEXT_ID#       DS    1F      next usable StringID num
XSI_MAX_ID#        DS    1F      max num of StringID nodes
XSI_TOTAL_SIZE     DS    1F      sys_svc_parm total size
XSI_FREE_SPACE     DS    1F      space left for data
XSI_CURR_NULL      DS    1F
XSI_CURR_FREE@31   DS    1F      31bit current @ of free space
XSI_TREE_NULL      DS    1F
XSI_TREE_HEAD@31   DS    1F      31bitlocation of tree head
XSI_DYN_NULL       DS    1F
XSI_DYN_AREA31     DS    1F      31bitaddress of dynamic area
XSI_HEADER_END     DS    0X
XSI_ID_LIST_NULL   DS    1F
XSI_ID_LIST31      DS    1F      31bit ID list for quick search


*********************************************************************
* StringID Tree Node Header                                         *
*********************************************************************
XSI_NODE           DSECT
XSI_NODE_HEADER    DS    0D
XSI_NODE_LEFT_NULL DS    1F
XSI_NODE_LEFT31    DS    1F      31bit LESS THAN PARENT NODE
XSI_NODE_RIGHT_NULL DS   1F
XSI_NODE_RIGHT31   DS    1F      31bit GREATER THAN PARENT NODE
XSI_NODE_StringID  DS    1F      STRING ID VALUE
```

```
XSI_NODE_STRLEN     DS    1F      LENGTH OF THE STRING
XSI_NODE_HEADER_END DS    0X
XSI_NODE_STRING     DS    C       THE ACTUAL STRING

**********************************************************************
* Description of the Parameters:                                     *
*   Input: SYS_SVC_PARM - Contain the XSI structure information      *
*          STR - The string that will be inserted into the tree      *
*          STRLEN - Length of the current string needed to insert    *
*                   Length is in number of bytes of the characters   *
*          CCSID - Identifier that identify character set of string  *
*   Output: STRID - the index of the inserted string                 *
*           EXIT_DIAG_CODE - XSI_DC_XXX provide detail to retcode    *
*           RETCODE - XRC_FAILURE if failed to insert/search for str *
*                     XRC_SUCCESS if string inserted/found           *
**********************************************************************
PARMLIST             DSECT
SYS_SVC_PARM_PTR     DS    1F
STR_PTR              DS    1F
STRLEN_PTR           DS    1F
STRID_PTR            DS    1F
CCSID_PTR            DS    1F
DIAGCODE_PTR         DS    1F
RETCODE_PTR          DS    1F

********************************************************************
* EXIT DIAGNOSTIC CODE                                             *
********************************************************************
* String length passed into gxle1idi is out of bound
XSI_DC_INCORRECT_PARM_STRLEN           EQU   11
* Tries to insert new string, but allocated storage is full
XSI_DC_OUT_OF_STORAGE_SPACE            EQU   12
* Eye Catcher string is incorrect
XSI_DC_INCORRECT_EYE_CATCHER           EQU   15
* StringID list is max out with entries,requires larger list size
XSI_DC_MAX_OUT_ID_LIST_ENTRIES         EQU   16

********************************************************************
* RETURN CODE                                                      *
********************************************************************
XSI_RC_SUCCESS      EQU  0
XSI_RC_FAILURE      EQU  1

SYS_SVC_PARM        EQU   0
STR                 EQU   0
STRLEN              EQU   0
STRID               EQU   0
CCSID               EQU   0
EXIT_DIAG_CODE      EQU   0
RETCODE             EQU   0

INSERTION_PTR       EQU   0
                    END
```

GXLE1IDR

This example module uses the input StringID to access a table and returns the address and length of the string associated with the StringID. The string is saved in the storage pointed to by SYS_SVC_PARM during the initialization of the parser (GXL1INI) and in StringID processing (GXLE1INI).

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example module assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example module:

SYS_SVC_PARM
: Address of storage area that the caller of the z/OS XML parser wants to pass to the exit. It also contains the XSI structure information.

STRID StringID used for indexing the list.

The following output variables are used in the example module:

STR_ADDR
: Address of string from requested StringID.

STRLEN
: The length of the string found by StringID.

DIAG_CODE
: Contains diagnostic information.

  XSI_DC_INCORRECT_StringID_OUTOFBOUND
  : *STRID* length is out of bound.

  XSI_DC_INCORRECT_ID_LOCATION_ERROR
  : StringID does not match.

  XSI_DC_INCORRECT_EYE_CATCHER
  : Eye catcher is incorrect.

RETCODE

  XSI_RC_FAILURE
  : The string cannot be retrieved.

  XSI_RC_SUCCESS
  : The string was retrieved successfully.

```
GXLE1IDR CSECT                  Set up control section
GXLE1IDR AMODE 31               Address mode of module
GXLE1IDR RMODE ANY              Residence of module
         USING DATA+0,R12       Relative Branching
         ENTRY GXLE1IDR         external entry point
         STM   R14,R12,12(R13)  Save regs
         LARL  R12,DATA         Load DATA Address

         J     IDSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE1IDR'
IDSTART  DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,R1
* Establish addressability for the XSI DSECT to the SYS_SVC_PARM
```

```
         USING XSI,R6

*********************************************************
* Check if the XSI eye catcher string matches correctly *
*********************************************************
         L     R5,SYS_SVC_PARM_PTR
         L     R6,SYS_SVC_PARM(,R5)
         CLC   XSI_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSI structure, so we not *
* sure if the data in the XSI structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         LHI   R5,XSI_RC_FAILURE
         L     R2,RETCODE_PTR
         ST    R5,RETCODE(,R2)
         LHI   R5,XSI_DC_INCORRECT_EYE_CATCHER
         L     R2,DIAGCODE_PTR
         ST    R5,DIAG_CODE(,R2)
         J     EXITIDR

********************************************************************
*** MAINLINE CODE STARTS HERE:                                   ***
********************************************************************
*** GXLE1IDR.PLX - Search & Retrieve String from StringID        ***
********************************************************************

**********************************************************
* The XSI structure is intact and eye catcher is correct *
* Set the dynamic area pointer to register 11            *
**********************************************************
PROCEED  DS    0H

********************************************************************
* If string ID contain a valid index, gxlp1idr will proceed and    *
* retrieve the corresponding string from the tree                  *
********************************************************************
         L     R2,STRID_PTR
         ICM   R4,15,STRID(R2)
         JZ    BAD_ID
         CL    R4,XSI_NEXT_ID#
         JL    RETRIEVE

* The StringID value is out of bound
BAD_ID   DS    0H
         LHI   R7,XSI_RC_FAILURE
         L     R5,RETCODE_PTR
         ST    R7,RETCODE(,R5)
         LHI   R2,XSI_DC_INCORRECT_StringID_OUTOFBOUND
         L     R5,DIAGCODE_PTR
         ST    R2,DIAG_CODE(,R5)
         J     EXITING

RETRIEVE DS    0H
         LR    R5,R4   reg4 contains the next available StringID #
         SLL   R5,3    multiply the number of StringID by 8

* Establish addressability for the XSI_NODE DSECT to the StringID Tree
         USING XSI_NODE,R3

         L     R3,XSI_ID_LIST31-8(R5)
         CL    R4,XSI_NODE_StringID  check if the StringID is correct
         JE    DONE

* The value of the StringID in the tree is not consistent with
* the value in the StringID list, exiting subroutine
         LHI   R4,XSI_RC_FAILURE
         L     R2,RETCODE_PTR
         ST    R4,RETCODE(,R2)
         LHI   R2,XSI_DC_INCORRECT_ID_LOCATION_ERROR
         L     R4,DIAGCODE_PTR
         ST    R2,DIAG_CODE(,R4)
         J     EXITING

* Successful in finding the StringID
DONE     DS    0H
         L     R5,STR_ADDR_PTR
         LA    R8,XSI_NODE_STRING
         ST    R8,STR_ADDR(,R5)     Store the string pointer
         L     R5,STRLEN_PTR
         L     R9,XSI_NODE_STRLEN
         ST    R9,STRLEN(,R5)       Store the string length

* Set the return code to successful and return back to caller
         LHI   R2,XSI_RC_SUCCESS
         L     R7,RETCODE_PTR
         ST    R2,RETCODE(,R7)
         L     R4,DIAGCODE_PTR
         ST    R2,DIAG_CODE(,R4)
         J     EXITING

****************************************************************
*** EXITING THE StringID SERVICE EXIT ROUTINE               ***
****************************************************************
EXITING  DS    0H
         ST    R2,XSI_DIAG_CODE

EXITIDR  DS    0H
         DROP  R1,R3,R6
         LM    R14,R12,12(R13)
         BSM   0,R14

DATA     DS    0F
         DC    A(GXLE1IDR)
EYECHAR  DC    CL8'XSIEYECA'


R0                 EQU   0
R1                 EQU   1
R2                 EQU   2
R3                 EQU   3
R4                 EQU   4
R5                 EQU   5
R6                 EQU   6
R7                 EQU   7
R8                 EQU   8
R9                 EQU   9
R10                EQU   10
R11                EQU   11
R12                EQU   12
R13                EQU   13
R14                EQU   14
R15                EQU   15

*********************************************************************
* Description of the XSI Structure:                                 *
*   The XSI structure is including the XSI Header, StringID array  *
*   list, the dynamic area, and end with empty storage area space  *
*   for the StringID tree                                          *
*      o XSI Header: Contain information for the XSI Structure to  *
*                    operate correctly (space management)          *
*      o StringID list: List of pointers that indexed by StringID  *
*                        and each ptr point to the corresponding    *
*                        leaf node within the StringID tree         *
*      o Dynamic Area: Used by ASAENTRY where all local variables, *
*                       stacks, and storage usage will take up the  *
*                       storage space within the dynamic area       *
*      o Empty Storage: managed by the XSI header and is used for  *
*                        storing the tree structure for StringID    *
*********************************************************************
XSI                 DSECT         StringID storage area
XSI_HEADER          DS    0D
XSI_EYE_CATCHER     DS    CL8     eye-catcher string
XSI_SYM_MAX_SIZE    DS    1F      max string len for symbol
XSI_DIAG_CODE       DS    1F      exit diagnostic code
XSI_NEXT_ID#        DS    1F      next usable StringID num
XSI_MAX_ID#         DS    1F      max num of StringID nodes
XSI_TOTAL_SIZE      DS    1F      sys_svc_parm total size
XSI_FREE_SPACE      DS    1F      space left for data
XSI_CURR_NULL       DS    1F
XSI_CURR_FREE@31    DS    1F      31bit current @ of free space
XSI_TREE_NULL       DS    1F
XSI_TREE_HEAD@31    DS    1F      31bitlocation of tree head
XSI_DYN_NULL        DS    1F
XSI_DYN_AREA31      DS    1F      31bitaddress of dynamic area
XSI_HEADER_END      DS    0X
XSI_ID_LIST_NULL    DS    1F
XSI_ID_LIST31       DS    1F      31bit ID list for quick search


*********************************************************************
* StringID Tree Node Header                                         *
*********************************************************************
XSI_NODE            DSECT
XSI_NODE_HEADER     DS    0D
XSI_NODE_LEFT_NULL  DS    1F
XSI_NODE_LEFT31     DS    1F      31bit LESS THAN PARENT NODE
XSI_NODE_RIGHT_NULL DS    1F
XSI_NODE_RIGHT31    DS    1F      31bit GREATER THAN PARENT NODE
XSI_NODE_StringID   DS    1F      STRING ID VALUE
XSI_NODE_STRLEN     DS    1F      LENGTH OF THE STRING
XSI_NODE_HEADER_END DS    0X
XSI_NODE_STRING     DS    C       THE ACTUAL STRING

*********************************************************************
* Description of the Parameters:                                    *
*    Input: SYS_SVC_PARM - Contain the XSI structure information    *
*           STRID - String ID used for indexing the list            *
*    Output: STR_ADDR - address of string from requested StringID  *
*            STRLEN - the length of the string found by StringID    *
*            DIAG_CODE - XSI_DC_XXX details to return code          *
*                        (Set only if error found)                  *
*            RETCODE - XSI_RC_FAILURE if string cannot be retrieved*
*                      XSI_RC_SUCCESS if string retrieved ok        *
*********************************************************************
PARMLIST             DSECT
SYS_SVC_PARM_PTR     DS    1F
STR_ADDR_PTR         DS    1F
STRLEN_PTR           DS    1F
STRID_PTR            DS    1F
DIAGCODE_PTR         DS    1F
RETCODE_PTR          DS    1F

*********************************************************************
* EXIT DIAGNOSTIC CODE                                              *
*********************************************************************
* Caller tries to retrieve string, but StringID is out of bound
XSI_DC_INCORRECT_StringID_OUTOFBOUND   EQU   13
* After gxle1idr obtained the pointer to the tree node, the string
* ID is not matching with the StringID passed in from parameter
XSI_DC_INCORRECT_ID_LOCATION_ERROR     EQU   14
* Eye Catcher string is incorrect
XSI_DC_INCORRECT_EYE_CATCHER           EQU   15

*******************************************************************
* RETURN CODE                                                     *
*******************************************************************
XSI_RC_SUCCESS      EQU  0
XSI_RC_FAILURE      EQU  1

SYS_SVC_PARM         EQU   0
STR_ADDR             EQU   0
STRLEN               EQU   0
STRID                EQU   0
DIAG_CODE            EQU   0
RETCODE              EQU   0
                     END
```

Appendix K. Exit examples - AMODE 64

GXLE4FRM (GXLFST64 example)

Restrictions: The following restrictions apply to this example:

- This sample was designed to be a basic example of a memory service exit, and was not designed with other system considerations in mind, such as the z/OS XML parser running in cross memory mode, SRB mode, or in a different key, for instance.
- This sample is not designed to work with any other service exits. The exit workarea is assumed to be used by this memory service exit only. (Note that both GXLGST64 and GXLFST64 services are considered as one service exit). As a result, this memory service exit can only work independently, with no other service exits running.

The following code frees an area of memory passed by the z/OS XML parser. For the exit service, see “GXLFST31 (GXLFST64) — free memory” on page 107. AMODE 31 callers use “GXLE1FRM (GXLFST31 example)” on page 194.

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example:

SYS_SVC_PARM
Address that was passed to the z/OS XML parser at initialization time.

MEMORY_LEN
Contains the length of the memory area requested to be free.

The following output variables are used in the example:

MEMORY_ADDR
The address of the memory to be freed.

EXIT_DIAG_CODE
Contains diagnostic information.

XSM_DC_INVALID_EYECATCHER_STR
Eye catcher is incorrect.

XSM_DC_FAIL_FREE_MEM64
Fail to release storage memory.

RETCODE

XSM_RC_FAILURE
Unable to free memory.

XSM_RC_SUCCESS
The iarv64 macro released the allocated memory successfully (greater than zero if deallocation failed).

RSNCODE
Contains the reason code generated by the IARV64 macro.

```
* SYSSTATE is used to initialize some variables used by iarv64 macro
         SYSSTATE ASCENV=P,AMODE64=,ARCHLVL=,OSREL=
         SYSSTATE ASCENV=,AMODE64=YES,ARCHLVL=,OSREL=
         SYSSTATE ASCENV=,AMODE64=,ARCHLVL=2,OSREL=
GXLE4FRM CSECT                         Set up control section
GXLE4FRM AMODE 64                      Address mode of module
```

```
GXLE4FRM RMODE ANY                  Residence of module
         USING DATA+0,G64R12        Relative Branching
         ENTRY GXLE4FRM             external entry point
         STMG  G64R14,G64R12,8(G64R13)
         LARL  G64R12,DATA          Load DATA Address

         J     MSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE4FRM'
MSTART   DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R1
* Establish addressability for the XSM DSECT to the SYS_SVC_PARM
         USING XSM,G64R2

*********************************************************
* Check if the XSM eye catcher string matches correctly *
*********************************************************
         LG    G64R2,SYS_SVC_PARM_PTR
         LG    G64R2,SYS_SVC_PARM(,G64R2)

         CLC   XSM_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSM structure, so we not *
* sure if the data in the XSM structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         LG    G64R7,RETCODE_PTR
         LGHI  G64R10,XSM_RC_FAILURE
         ST    G64R10,RETCODE(,G64R7)
         LG    G64R7,EXITDIAGCODE_PTR
         LGHI  G64R10,XSM_DC_INVALID_EYECATCHER_STR
         ST    G64R10,EXIT_DIAG_CODE(,G64R7)
         J     EXIT

**********************************************************
* The XSM structure is intact and eye catcher is correct *
* Set the dynamic area pointer to register 10            *
**********************************************************
PROCEED  DS    0H
         LG    G64R10,XSM_DYN_AREA_PTR
         USING DATD,G64R10

*********************************************************
* Setting up the parameter list passed in by the caller *
*********************************************************
         XGR   G64R3,G64R3
         LG    G64R6,RETCODE_PTR
         ST    G64R3,RETCODE(,G64R6)
         ST    G64R3,XSM_DIAG_CODE

****************************************************
* Call the Storage Detach macro to free the memory *
****************************************************
         LG    G64R3,MEMORY_ADDR_PTR
         LG    G64R3,MEMORY_ADDR(,G64R3)
         STG   G64R3,MEMPTR
         DROP  G64R1
         LGR   G64R4,G64R1  Backup Reg1 Parameter address

* After iarv64 macro finished, register 15 will contain the
* return code of the macro.  Zero if deallocation is successful,
* or greater than zero if failed to deallocate the memory
         IARV64 REQUEST=DETACH,COND=YES,RSNCODE=V64RSN,              +
               MEMOBJSTART=MEMPTR,MF=(E,IARV64L,COMPLETE)

************************************************
* Make sure the storage release is successful *
************************************************
         LGR   G64R1,G64R4  Restore Reg1 Parameter Address

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R1

         XGR   G64R8,G64R8
         LG    G64R5,RETCODE_PTR
         ST    G64R15,RETCODE(,G64R5)   r15 is macro retcode
         CGR   G64R15,G64R8
         JE    EXIT

******************************
* Memory deallocation failed *
******************************
         LG    G64R5,RSNCODE_PTR
         MVC   RSNCODE(,G64R5),V64RSN
         LGHI  G64R4,XSM_DC_FAIL_FREE_MEM64
         LG    G64R3,EXITDIAGCODE_PTR
         ST    G64R4,EXIT_DIAG_CODE(,G64R3)

*************************************************
* EXITING THE FREE MEMORY SERVICE EXIT ROUTINE *
*************************************************
EXITMEM  DS    0H
         ST    G64R4,XSM_DIAG_CODE

EXIT     DS    0H
         DROP  G64R1,G64R2
         LMG   G64R14,G64R12,8(G64R13)
         BR    G64R14

DATA     DS    0F
         DC    A(GXLE4FRM)
EYECHAR  DC    CL8'XSMEYECA'

*******************************************************
* The end of the Free Memory Service Exit subroutine *
*******************************************************


DATD     DSECT
         DS    0D
MEMPTR   DS    D
V64RSN   DS    F
         IARV64 MF=(L,IARV64L)

*********************************************************************
* Description of the XSM Structure:                                 *
*   The XSM structure is including the XSM Header, eye catcher      *
*   string, and the dynamic area pointer that points to the empty  *
*   storage area space for the dynamic storage area                 *
*      o XSM Header: Contain all sub-variables within the XSM       *
*                    structure before the dynamic area              *
*      o Eye Catcher: A string used to identify if this chunk of    *
*                     memory is allocated and referenced correctly *
*      o Dynamic Area:  managed by the dynamic area pointer and is *
*                       used for storing local variables and uses  *
*********************************************************************
XSM               DSECT           Memory storage area
XSM_HEADER        DS    0D
XSM_EYE_CATCHER   DS    CL8       eye-catcher string
XSM_DIAG_CODE     DS    1F        exit diagnostic code
```

```
XSM_TOTAL_SIZE    DS    1F        sys_svc_parm total size
XSM_DYN_AREA_PTR  DS    1D        64bit address of dynamic area
XSM_HEADER_END    DS    0X

*********************************************************************
* Description of the Parameters:                                    *
*     Input: SYS_SVC_PARM - Address that was passed to the parser  *
*                          at initialization time                   *
*            MEMORY_LEN - Contain the length of the memory area    *
*                         requested to be free                      *
*     Output: MEMORY_ADDR - The Address of the  memory to be freed *
*             EXIT_DIAG_CODE - Contain diagnostic information      *
*             RETCODE - XSM_RC_FAILURE if unable to free memory    *
*             EXIT_DIAG_CODE - XSM_DC_INVALID_EYECATCHER_STR       *
*                              XSM_DC_FAIL_FREE_MEM64               *
*             RSNCODE - Reason code generated by the IARV64 macro  *
*********************************************************************
PARMLIST          DSECT
SYS_SVC_PARM_PTR  DS    1D
MEMORY_ADDR_PTR   DS    1D
MEMORY_LEN_PTR    DS    1D
EXITDIAGCODE_PTR  DS    1D
RETCODE_PTR       DS    1D
RSNCODE_PTR       DS    1D

SYS_SVC_PARM      EQU   0
MEMORY_ADDR       EQU   0
MEMORY_LEN        EQU   0
EXIT_DIAG_CODE    EQU   0
RETCODE           EQU   0
RSNCODE           EQU   0

G64R0             EQU   0
G64R1             EQU   1
G64R2             EQU   2
G64R3             EQU   3
G64R4             EQU   4
G64R5             EQU   5
G64R6             EQU   6
G64R7             EQU   7
G64R8             EQU   8
G64R9             EQU   9
G64R10            EQU   10
G64R11            EQU   11
G64R12            EQU   12
G64R13            EQU   13
G64R14            EQU   14
G64R15            EQU   15

*********************************************************************
* EXIT DIAGNOSTIC CODE                                              *
*********************************************************************
* Eye Catcher string is incorrect
XSM_DC_INVALID_EYECATCHER_STR   EQU   1
* Memory de-allocation failed to free storage in gxle1frm
XSM_DC_FAIL_FREE_MEM64          EQU   6

*********************************************************************
* RETURN CODE                                                       *
*********************************************************************
XSM_RC_FAILURE    EQU   1
                  END
```

GXLE4GTM (GXLGST64 example)

Restrictions: The following restrictions apply to this example:

- This sample was designed to be a basic example of a memory service exit, and was not designed with other system considerations in mind, such as the z/OS XML parser running in cross memory mode, SRB mode, or in a different key, for instance.
- This sample is not designed to work with any other service exits. The exit workarea is assumed to be used by this memory service exit only. (Note that both GXLGST64 and GXLFST64 services are considered as one service exit). As a result, this memory service exit can only work independently, with no other service exits running.

The following code allocates an area of memory for the size requested by the z/OS XML parser. For the exit service, see “GXLGST31 (GXLGST64) — get memory” on page 104. AMODE 31 callers use “GXLE1GTM (GXLGST31 example)” on page 198.

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example:

SYS_SVC_PARM
: Address that was passed to the z/OS XML parser at initialization time.

MEMORY_LEN
: Contains the length of the memory area requested by the z/OS XML parser.

The following output variables are used in the example:

MEMORY_ADDR
: The address of the allocated memory.

EXIT_DIAG_CODE
: Contains diagnostic information.

  XSM_DC_INVALID_EYECATCHER_STR
  : Eye catcher is incorrect.

  XSM_DC_INVALID_GET_MEM_LEN
  : Memory length is out of bound.

  XSM_DC_FAIL_ALLOCATE_MEM64
  : Storage memory allocation failed.

RETCODE

  XSM_RC_FAILURE
  : Unable to allocate memory.

  XSM_RC_SUCCESS
  : The iarv64 macro allocated the memory successfully (greater than zero if allocation failed).

RSNCODE
: Contains the reason code generated by the IARV64 macro.

```
* SYSSTATE is used to initialize some variables used by iarv64 macro
         SYSSTATE ASCENV=P,AMODE64=,ARCHLVL=,OSREL=
         SYSSTATE ASCENV=,AMODE64=YES,ARCHLVL=,OSREL=
         SYSSTATE ASCENV=,AMODE64=,ARCHLVL=2,OSREL=
GXLE4GTM CSECT                      Set up control section
GXLE4GTM AMODE 64                   Address mode of module
GXLE4GTM RMODE ANY                  Residence of module
         USING DATA+0,G64R12        Relative Branching
         ENTRY GXLE4GTM             external entry point
         STMG  G64R14,G64R12,8(G64R13)
         LARL  G64R12,DATA          Load DATA Address

         J     MSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE4GTM'
MSTART   DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R1
* Establish addressability for the XSM DSECT to the SYS_SVC_PARM
         USING XSM,G64R2

*********************************************************
* Check if the XSM eye catcher string matches correctly *
*********************************************************
         LG    G64R2,SYS_SVC_PARM_PTR
         LG    G64R2,SYS_SVC_PARM(,G64R2)

         CLC   XSM_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSM structure, so we not *
* sure if the data in the XSM structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         LG    G64R7,RETCODE_PTR
         LGHI  G64R10,XSM_RC_FAILURE
         ST    G64R10,RETCODE(,G64R7)
         LG    G64R7,EXITDIAGCODE_PTR
         LGHI  G64R10,XSM_DC_INVALID_EYECATCHER_STR
         ST    G64R10,EXIT_DIAG_CODE(,G64R7)
         J     EXIT

**********************************************************
* The XSM structure is intact and eye catcher is correct *
* Set the dynamic area pointer to register 10            *
**********************************************************
PROCEED  DS    0H
         LG    G64R10,XSM_DYN_AREA_PTR
         USING DATD,G64R10

*********************************************************
* Setting up the parameter list passed in by the caller *
*********************************************************
         XGR   G64R6,G64R6
         LG    G64R3,RETCODE_PTR
         ST    G64R6,RETCODE(,G64R3)
         ST    G64R6,XSM_DIAG_CODE

*****************************************************
* Check to see if a valid memory length is provided *
*****************************************************
         LG    G64R7,MEMORY_LEN_PTR
         LG    G64R5,MEMORY_LEN(,G64R7)
         LTGR  G64R5,G64R5
         JNZ   ALLOCMEM

********************************************************
* Incorrect memory length is found, exiting subroutine *
********************************************************
         LGHI  G64R4,XSM_RC_FAILURE
         ST    G64R4,RETCODE(,G64R3)   reg3 is RETCODE_PTR
         LGHI  G64R4,XSM_DC_INVALID_GET_MEM_LEN
         LG    G64R3,EXITDIAGCODE_PTR
         ST    G64R4,EXIT_DIAG_CODE(,G64R3)
         J     EXITMEM

*************************************************************
* Start allocating memory and check if the allocation is OK *
*************************************************************
ALLOCMEM DS    0H
* Change allocation size into megabytes' segment
         LGR   G64R8,G64R5     reg5 is the memory length
         ALG   G64R8,SIZENUM
         SRLG  G64R8,G64R8,20
         STG   G64R8,NUMSEGS

* Calling epar to obtain the ASVTENTY index for the ASCB addr
         EPAR  G64R8     Obtain the ASCB index into reg8
         LLGFR G64R8,G64R8
         SLAG  G64R8,G64R8,2  Multiply r8 by 4bytes for fixed(32)

* Setting up TTOKEN for the IARV64 Get Storage Marco
         USING PSA,0
         LLGT  G64R7,FLCCVT          Obtain CVT Addr from PSA
         USING CVT,G64R7
         LLGT  G64R7,CVTASVT         Load ASVT Address
         USING ASVT,G64R7
         LLGT  G64R7,ASVTENTY(G64R8) Load ASCB Address
         USING ASCB,G64R7
         LLGT  G64R7,ASCBXTCB        Load TCB Address
         USING TCB,G64R7
         LLGT  G64R7,TCBSTCB         Load STCB Address
         USING STCB,G64R7
         MVC   TOKN(16),STCBTTKN     Copy TTOKEN to local TOKN

         DROP  G64R1
         LGR   G64R4,G64R1  Backup Reg1 Parameter Address

* After iarv64 macro finished, register 15 will contain the
* return code of the macro.  Zero if allocation is successful,
* or greater than zero if failed to allocate memory
         IARV64 REQUEST=GETSTOR,COND=YES,SEGMENTS=NUMSEGS,
               RSNCODE=V64RSN,ORIGIN=MEMPTR,MF=(E,IARV64L,COMPLETE)


         LGR   G64R1,G64R4  Restore Reg1 Parameter Address

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R1

**************************************************************
* Check if allocated memory returns a successful return code *
**************************************************************
         XGR   G64R7,G64R7
         LG    G64R5,RETCODE_PTR
         ST    G64R15,RETCODE(,G64R5)  r15 is macro retcode
         CGR   G64R15,G64R7
         JE    FINISH

************************************************
* Memory allocation failed, exiting subroutine *
```

```
************************************************
         LG    G64R5,RSNCODE_PTR
         MVC   RSNCODE(,G64R5),V64RSN
         LGHI  G64R4,XSM_DC_FAIL_ALLOCATE_MEM64
         LG    G64R9,EXITDIAGCODE_PTR
         ST    G64R4,EXIT_DIAG_CODE(,G64R9)
         J     EXITMEM

************************************
* Memory allocation is successful! *
************************************
FINISH   DS    0H
         LG    G64R3,MEMORY_ADDR_PTR
         MVC   MEMORY_ADDR(,G64R3),MEMPTR

         LG    G64R5,MEMORY_LEN_PTR
         SLLG  G64R9,G64R8,20
         STG   G64R9,MEMORY_LEN(,G64R5)
         J     EXIT

***********************************************
* EXITING THE GET MEMORY SERVICE EXIT ROUTINE *
***********************************************
EXITMEM  DS    0H
         ST    G64R4,XSM_DIAG_CODE

EXIT     DS    0H
         DROP  G64R1,G64R2
         LMG   G64R14,G64R12,8(G64R13)
         BR    G64R14

DATA     DS    0F
         DC    A(GXLE4GTM)
SIZENUM  DC    FD'1048575'
EYECHAR  DC    CL8'XSMEYECA'

*****************************************************
* The end of the Get Memory Service Exit subroutine *
*****************************************************

DATD     DSECT
         DS    0D
NUMSEGS  DS    D
MEMPTR   DS    D
TOKN     DS    CL16
V64RSN   DS    F
         IARV64 MF=(L,IARV64L)

*********************************************************************
* Description of the XSM Structure:                                 *
*   The XSM structure is including the XSM Header, eye catcher      *
*   string, and the dynamic area pointer that points to the empty   *
*   storage area space for the dynamic storage area                 *
*      o XSM Header: Contain all sub-variables within the XSM       *
*                    structure before the dynamic area              *
*      o Eye Catcher: A string used to identify if this chunk of    *
*                     memory is allocated and referenced correctly  *
*      o Dynamic Area:  managed by the dynamic area pointer and is  *
*                       used for storing local variables and uses   *
*********************************************************************
XSM               DSECT           Memory storage area
XSM_HEADER        DS    0D
XSM_EYE_CATCHER   DS    CL8       eye-catcher string
XSM_DIAG_CODE     DS    1F        exit diagnostic code
XSM_TOTAL_SIZE    DS    1F        sys_svc_parm total size
XSM_DYN_AREA_PTR  DS    1D        64bit address of dynamic area
XSM_HEADER_END    DS    0X
```

```
*********************************************************************
* Description of the Parameters:                                    *
*     Input: SYS_SVC_PARM - Address that was passed to the parser  *
*                         at initialization time                    *
*            MEMORY_LEN - Contain the length of the memory area    *
*                         requested by the parser                   *
*     Output: MEMORY_ADDR - The Address of the allocated memory    *
*             EXIT_DIAG_CODE - Contain diagnostic information      *
*             RETCODE - XSM_RC_FAILURE if failed to allocate memory*
*             EXIT_DIAG_CODE - XSM_DC_INVALID_EYECATCHER_STR       *
*                              XSM_DC_INVALID_GET_MEM_LEN          *
*                              XSM_DC_FAIL_ALLOCATE_MEM64          *
*             RSNCODE - Reason code generated by the IARV64 macro  *
*********************************************************************
PARMLIST          DSECT
SYS_SVC_PARM_PTR  DS    1D
MEMORY_ADDR_PTR   DS    1D
MEMORY_LEN_PTR    DS    1D
EXITDIAGCODE_PTR  DS    1D
RETCODE_PTR       DS    1D
RSNCODE_PTR       DS    1D

SYS_SVC_PARM      EQU   0
MEMORY_ADDR       EQU   0,8,C'A'
MEMORY_LEN        EQU   0
EXIT_DIAG_CODE    EQU   0
RETCODE           EQU   0
RSNCODE           EQU   0

G64R0             EQU   0
G64R1             EQU   1
G64R2             EQU   2
G64R3             EQU   3
G64R4             EQU   4
G64R5             EQU   5
G64R6             EQU   6
G64R7             EQU   7
G64R8             EQU   8
G64R9             EQU   9
G64R10            EQU   10
G64R11            EQU   11
G64R12            EQU   12
G64R13            EQU   13
G64R14            EQU   14
G64R15            EQU   15

*********************************************************************
* EXIT DIAGNOSTIC CODE                                              *
*********************************************************************
* Eye Catcher string is incorrect
XSM_DC_INVALID_EYECATCHER_STR   EQU   1
* Memory length passed into subroutine is out of bound
XSM_DC_INVALID_GET_MEM_LEN      EQU   2
* Memory allocation failed to allocate storage in gxle4gtm
XSM_DC_FAIL_ALLOCATE_MEM64      EQU   4
* Memory de-allocation failed to free storage in gxle1frm

*********************************************************************
* RETURN CODE                                                       *
*********************************************************************
XSM_RC_FAILURE    EQU   1

*********************************************************************
* MACROS INCLUDED FOR OBTAINING TTOKEN USED BY IARV64               *
*********************************************************************
                  CVT DSECT=YES
```

```
|                                            IHAASVT
|                                            IKJTCB
|                                            IHASTCB
|                                            IHAASCB
|                                            IHAPSA
|                                            END
```

GXLSYM64 example

Restrictions: The following restrictions apply to this example:

- This example was designed to be a basic example of a StringID service exit. It was not designed with other system considerations in mind, such as the z/OS XML parser running in cross memory mode, SRB mode, or in a different key, for instance.
- This example is not designed to work with any other service exits. The exit workarea is assumed to be used by this StringID service exit only. As a result, this StringID service exit can only work independently, with no other service exits running.

Note: This exit example is divided into the following 3 modules:

- “GXLE4INI” on page 235
- “GXLE4IDI (GXLSYM64 example module)” on page 240
- “GXLE4IDR” on page 248

For the exit service, see “GXLSYM31 (GXLSYM64) — StringID service” on page 109. AMODE 31 callers use “GXLSYM31 example” on page 203.

GXLE4INI

This example module does the following:

- Validates the caller specification and determines whether to use user defined or default values for storage size.
- Initializes all variables in XSI. (XSI is the data structure for the StringID sample exit).

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example module assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example module:

STRID_AREA_ADDR
  Address of the XSI storage area.

STRID_AREA_LEN
  Total length of the XSI Storage area.

STRID_MAX_NUM
  The maximum number of StringIDs allowed.

SYM_MAX_SIZE
  The maximum string length for each symbol.

The following output variables are used in the example module:

RETCODE

  XSI_RC_FAILURE
    The storage area failed to initialize.

  XSI_RC_SUCCESS
    The storage area successfully initialized.

DIAG_CODE
  Contains diagnostic information.

  XSI_DC_SYMBOL_STORAGE_TOO_SMALL
    Storage size is too small.

```
GXLE4INI CSECT                  Set up control section
GXLE4INI AMODE 64               Address mode of module
GXLE4INI RMODE ANY              Residence of module
         USING DATA+0,G64R12    Relative Branching
         ENTRY GXLE4INI         external entry point
         BSM   G64R14,0         Set G64R14 for later PR
         BAKR  G64R14,0         Save register/psw status
         LARL  G64R12,DATA      Load DATA Address

         J     IDSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE4INI'
IDSTART  DS    0H

*********************************************************************
*** MAINLINE CODE STARTS HERE                                     ***
*********************************************************************
*** GXLE1INI.PLX - INITIALIZE XSI MACROS' DSECTS                  ***
*********************************************************************
```

```
* below will calculate the minimum storage area size allowed
* inorder for StringID service exit to function correctly

* XSI_DYN_AREA_SIZE + length(XSI_HEADER) + length(XSI_NODE_HEADER)
* Accumulating the minimum required size for storage area to
* initialize into register 3
         LGHI  G64R3,XSI_DYN_AREA_SIZE
         AGHI  G64R3,XSI_HEADER_SIZE
         AGHI  G64R3,XSI_NODE_HEADER_SIZE
         AGHI  G64R3,1  Minimum of one byte of string to be stored

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R1

* Check if the allocation size is enough or not
         LG    G64R10,STRID_AREA_LEN_PTR
         L     G64R8,STRID_AREA_LEN(,G64R10)
* Check if the user given storage area size has enough space
         CLGR  G64R8,G64R3   reg3 is accumulated value for minimum size
         JNL   MAXVALUE

* The storage size is too small for the StringID initialization
         LGHI  G64R6,XSI_RC_FAILURE
         LG    G64R5,RET_CODE_PTR
         ST    G64R6,RET_CODE(,G64R5)
         LGHI  G64R6,XSI_DC_SYMBOL_STORAGE_TOO_SMALL
         LG    G64R5,DIAG_CODE_PTR
         ST    G64R6,DIAG_CODE(,G64R5)
         J     EXITING

MAXVALUE DS    0H

* see if caller provided a maximum number of StringID allowed in
* this StringID service exit samples, if not provided, a default
* value will be used instead
         LG    G64R5,STRID_MAX_NUM_PTR
         ICM   G64R2,15,STRID_MAX_NUM(G64R5)
         JZ    ID_DEF

* A StringID maximum number is provided
         LLGFR G64R7,G64R2  reg2 is user defined max # of strid allowed
         J     SYM_MAX

* StringID maximum number is not provided, so default value is used
ID_DEF   DS    0H
         LGHI  G64R7,XSI_DEFAULT_MAX_ID#

* see if caller provided a maximum string size allowed in this
* StringID service exit samples, if not provided, a default
* string size will be used instead
SYM_MAX  DS    0H
         LG    G64R5,SYM_MAX_SIZE_PTR
         ICM   G64R2,15,SYM_MAX_SIZE(G64R5)
         JZ    SYM_DEF

* A valid maximum string size is provided
         LLGFR G64R6,G64R2  reg2 is user defined maximum string length
         J     INITIAL

* The default value of the maximum string size is used
SYM_DEF  DS    0H
         LGHI  G64R6,XSI_DEFAULT_SYM_MAX_SIZE

* Begin to initialize the StringID storage area
INITIAL  DS    0H

* Establish addressability for the XSI DSECT to the SYS_SVC_PARM
```

```
         USING XSI,G64R3

         LG    G64R3,STRID_AREA_ADDR_PTR
         LG    G64R3,STRID_AREA_ADDR(,G64R3)

* Initialize all StringID Storage area header values
         MVC   XSI_EYE_CATCHER,EYECHAR
         ST    G64R6,XSI_SYM_MAX_SIZE reg6 is max length of each string
         XGR   G64R6,G64R6
         ST    G64R6,XSI_DIAG_CODE

* StringID value starts at number one index
         LHI   G64R9,1
         ST    G64R9,XSI_NEXT_ID#
         ST    G64R7,XSI_MAX_ID#   reg7 is max num of StringID allowed
         ST    G64R8,XSI_TOTAL_SIZE  reg8 is StringID storage area size

* Setting up the dynamic area pointer
         MGHI  G64R7,XSI_LENGTH_ID_LIST   reg7 is XSI_MAX_ID#
         LLGFR G64R2,G64R7  reg7 is max num of StringID allowed times 8
         LGHI  G64R9,XSI_HEADER_SIZE
         ALGR  G64R2,G64R9

* add 7 then bitwise and with 'FFFFFFFFFFFFFFF8' to ensure the
* dynamic area is starting at double word boundary
         LGHI  G64R5,7
         ALGR  G64R5,G64R2  reg2 is size of XSI_header + max # of strid
         ALGR  G64R5,G64R3  reg3 is the StringID storage area address
         NG    G64R5,DW_BDRY

         STG   G64R5,XSI_DYN_AREA

* current free pointer is after all XSI header, StringID
* array list, and dynamic area
         LGHI  G64R9,XSI_DYN_AREA_SIZE
         LGR   G64R2,G64R5
         ALGR  G64R2,G64R9
         STG   G64R2,XSI_CURR_FREE@

* Calculate the amount of free space that is left in the storage area
         LLGFR G64R5,G64R8  reg8 is total StringID storage area size
         LGR   G64R7,G64R2
         SLGR  G64R7,G64R3  reg3 is the StringID storage area address
         SLGR  G64R5,G64R7
         ST    G64R5,XSI_FREE_SPACE

* Set the XSI_TREE_HEAD@ to NULL, which means the tree is empty
         XGR   G64R2,G64R2
         STG   G64R2,XSI_TREE_HEAD@  reg2 is set to zero/NULL

         LGHI  G64R6,XSI_RC_SUCCESS
         LG    G64R5,RET_CODE_PTR
         ST    G64R6,RET_CODE(,G64R5)
         J     EXITING

***************************************************************
*** EXITING THE StringID INITIALIZATION ROUTINE             ***
***************************************************************
EXITING  DS    0H
         DROP  G64R1,G64R3
         PR

DATA     DS    0F
         DC    A(GXLE4INI)
* One megabyte in size substract by one (1048576 minus 1)
SIZE     DC    FD'1048575'
DW_BDRY  DC    X'FFFFFFFFFFFFFFF8'
```

```
EYECHAR  DC    CL8'XSIEYECA'


G64R0                EQU   0
G64R1                EQU   1
G64R2                EQU   2
G64R3                EQU   3
G64R4                EQU   4
G64R5                EQU   5
G64R6                EQU   6
G64R7                EQU   7
G64R8                EQU   8
G64R9                EQU   9
G64R10               EQU   10
G64R11               EQU   11
G64R12               EQU   12
G64R13               EQU   13
G64R14               EQU   14
G64R15               EQU   15

*********************************************************************
* Description of the XSI Structure:                                 *
*   The XSI structure is including the XSI Header, StringID array  *
*   list, the dynamic area, and end with empty storage area space  *
*   for the StringID tree                                           *
*      o XSI Header: Contain information for the XSI Structure to  *
*                    operate correctly (space management)           *
*      o StringID list: List of pointers that indexed by StringID  *
*                       and each ptr point to the corresponding    *
*                       leaf node within the StringID tree         *
*      o Dynamic Area: Used by ASAENTRY where all local variables, *
*                      stacks, and storage usage will take up the  *
*                      storage space within the dynamic area       *
*      o Empty Storage: managed by the XSI header and is used for  *
*                       storing the tree structure for StringID    *
*********************************************************************
XSI                 DSECT          StringID storage area
XSI_HEADER          DS    0D
XSI_EYE_CATCHER     DS    CL8      eye-catcher string
XSI_SYM_MAX_SIZE    DS    1F       max string len for symbol
XSI_DIAG_CODE       DS    1F       exit diagnostic code
XSI_NEXT_ID#        DS    1F       next usable StringID num
XSI_MAX_ID#         DS    1F       max num of StringID nodes
XSI_TOTAL_SIZE      DS    1F       sys_svc_parm total size
XSI_FREE_SPACE      DS    1F       space left for data
XSI_CURR_FREE@      DS    1D       64bit current @ of free space
XSI_TREE_HEAD@      DS    1D       64bit location of tree head
XSI_DYN_AREA        DS    1D       64bit address of dynamic area
XSI_HEADER_END      DS    0X
XSI_ID_LIST         DS    1D       64bit ID list for quick search

XSI_HEADER_SIZE     EQU   XSI_HEADER_END-XSI_HEADER

*********************************************************************
* Description of the Parameters:                                    *
*   Input: STRID_AREA_PTR - Pointer to the XSI storage area         *
*          STRID_AREA_LEN - total length of the XSI Storage area    *
*          STRID_MAX_NUM - the maximum number of StringID allowed  *
*          SYM_MAX_SIZE - maximum string length for each symbol     *
*   Output: DIAG_CODE - XSI_DC_XXX provide detail to retcode        *
*           RETCODE - XSI_RC_FAILURE if string cannot be inserted  *
*                     XSI_RC_SUCCESS if string inserted ok          *
*********************************************************************
PARMLIST             DSECT
STRID_AREA_ADDR_PTR  DS    1D
STRID_AREA_LEN_PTR   DS    1D
STRID_MAX_NUM_PTR    DS    1D
```

```
SYM_MAX_SIZE_PTR     DS    1D
RET_CODE_PTR         DS    1D
DIAG_CODE_PTR        DS    1D

*********************************************************************
* EXIT DIAGNOSTIC CODE                                              *
*********************************************************************
* the storage size is too small for initialization
XSI_DC_SYMBOL_STORAGE_TOO_SMALL        EQU    17

* Default Maximum Constant values
XSI_DYN_AREA_SIZE            EQU   4096
XSI_DEFAULT_MAX_ID#          EQU   800
XSI_DEFAULT_SYM_MAX_SIZE     EQU   256

*********************************************************************
* RETURN CODE                                                       *
*********************************************************************
XSI_RC_SUCCESS       EQU   0
XSI_RC_FAILURE       EQU   1

STRID_AREA_ADDR      EQU   0
STRID_AREA_LEN       EQU   0
STRID_MAX_NUM        EQU   0
SYM_MAX_SIZE         EQU   0
RET_CODE             EQU   0
DIAG_CODE            EQU   0

XSI_NODE_HEADER_SIZE EQU   24
XSI_LENGTH_ID_LIST   EQU   8
                     END
```

GXLE4IDI (GXLSYM64 example module)

This example module does the following:

- Search for an identical string in the tree.
- Inserts a string into a tree and returns a unique StringID. This is accomplished as follows:
  1. Check first to make sure the length of the string is within the maximum symbol buffer size.
  2. Inserts the string into the root if the tree is empty or searches down the tree to find the appropriate empty leaf node.
  3. When the insert node location is found, it's address will be passed to the INSERT_STRING subroutine. The subroutine will create a new leaf node and then insert the string.
  4. Return the StringID if the string inserted successfully.

Note: This is the actual exit pointed to in the SYS_SVC_VECTOR table.

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example module assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example module:

SYS_SVC_PARM
Address of storage area that the caller of the z/OS XML parser wants to pass to the exit. It also contains the XSI structure information.

STR The string that will be inserted into the tree.

STRLEN
Length of the current string needed to be inserted. Length is derived from the number of bytes of the characters in the string.

CCSID
Identifier for the string's character set.

The following output variables are used in the example module:

STRID The index of the inserted or found string.

EXIT_DIAG_CODE
Contains diagnostic information.

XSI_DC_INCORRECT_PARM_STRLEN
String length is out of bound.

XSI_DC_OUT_OF_STORAGE_SPACE
Allocated storage is full.

XSI_DC_INCORRECT_EYE_CATCHER
Eye catcher is incorrect.

XSI_DC_MAX_OUT_ID_LIST_ENTRIES
StringID list is full.

RETCODE

XRC_FAILURE
Failed to insert or search for *STR*.

XRC_SUCCESS
STR was inserted or found.

```
GXLE4IDI CSECT                      Set up control section
GXLE4IDI AMODE 64                   Address mode of module
GXLE4IDI RMODE ANY                  Residence of module
         USING DATA+0,G64R12        Relative Branching
         ENTRY GXLE4IDI             external entry point
         STMG  G64R14,G64R12,8(G64R13) Save regs
         LARL  G64R12,DATA          Load DATA Address

         J     IDSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE4IDI'
IDSTART  DS    0H

* Establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R7
* Establish addressability for the XSI DSECT to the SYS_SVC_PARM
         USING XSI,G64R6

*********************************************************
* Check if the XSI eye catcher string matches correctly *
*********************************************************
         LGR   G64R7,G64R1
         LG    G64R6,SYS_SVC_PARM_PTR
         LG    G64R6,SYS_SVC_PARM(,G64R6)
         CLC   XSI_EYE_CATCHER,EYECHAR
         JE    PROCEED

**********************************************************
* Eye catcher is invalid in the XSI structure, so we not *
* sure if the data in the XSI structure is still valid,  *
* exiting subroutine, and return to the caller           *
**********************************************************
         LGHI  G64R9,XSI_RC_FAILURE
         LG    G64R8,RETCODE_PTR
         ST    G64R9,RETCODE(,G64R8)
         LGHI  G64R11,XSI_DC_INCORRECT_EYE_CATCHER
         LG    G64R10,DIAGCODE_PTR
         ST    G64R11,EXIT_DIAG_CODE(,G64R10)
         J     EXITIDI

*********************************************************************
*** MAINLINE CODE STARTS HERE:                                    ***
*********************************************************************
*** GXLE1IDI.PLX - Insert/Search String & return StringID         ***
*********************************************************************

**********************************************************
* The XSI structure is intact and eye catcher is correct *
* Set the dynamic area pointer to register 11            *
**********************************************************
PROCEED  DS    0H
         LG    G64R11,XSI_DYN_AREA
         USING DATD,G64R11

*********************************************************************
* - If string length and string contain the valid information      *
*   then gxle1idi will proceed and continue to insert the string   *
*   into the tree and return the appropreiate StringID value       *
*********************************************************************
         LG    G64R5,STRLEN_PTR
         ICM   G64R8,15,STRLEN(G64R5)   Check if string length is zero
         JZ    LEN_INV
* Check if strlen is greater then the maximum allowed string length
         CL    G64R8,XSI_SYM_MAX_SIZE
         JH    LEN_INV

* If the StringID list is not empty, search before insert
         LG    G64R2,XSI_TREE_HEAD@
         LTGR  G64R2,G64R2
         JNZ   SEARCH_TREE

* If StringID list is empty, insert string
         LA    G64R9,XSI_TREE_HEAD@
         STG   G64R9,INSERTION_ADDR_PTR
         J     INSERT_STRING

* Value of the string length is invalid, exiting the subroutine
LEN_INV  DS    0H
         LGHI  G64R5,XSI_RC_FAILURE
         LG    G64R3,RETCODE_PTR
         ST    G64R5,RETCODE(,G64R3)
         LGHI  G64R2,XSI_DC_INCORRECT_PARM_STRLEN
         LG    G64R3,DIAGCODE_PTR
         ST    G64R2,EXIT_DIAG_CODE(,G64R3)
         J     EXITING

*********************************************************************
*                                                                   *
* INSERT_STRING - Insert string into tree and return the StringID  *
*                                                                   *
*         The string will be pushed into the tree and the index    *
*         of the list will be saved as an unique StringID,         *
*         immediately upon the procedure checked the valid string  *
*         and string Length data.                                  *
*                                                                   *
* Input: STR_PTR - The string that will be inserted into the tree  *
*                                                                   *
*        STR_LEN - The length of the string that is loaded into R8 *
*                                                                   *
*        INSERTION_ADDR_PTR - address of a pointer to the location *
*                             where the new node will go           *
*        XSI - user control area header variables                  *
*                                                                   *
* Output: STRID - the index of the inserted string                 *
*                                                                   *
*         RETCODE - XRC_FAILURE if string cannot be inserted       *
*                   XRC_SUCCESS if string inserted ok              *
*                                                                   *
*********************************************************************
INSERT_STRING DS    0H
* plus 1 is for adding a NULL character at the end of each
* string so user can know when the end of string reaches
         LGHI  G64R5,25   24 is length of XSI_NODE_HEADER, 1 is NULL
         ALR   G64R5,G64R8    reg8 is the string length
         LLGFR G64R5,G64R5
         STG   G64R5,NEW_NODE_SIZE

* check if the StringID reaches it's maximum allowed value
         L     G64R4,XSI_NEXT_ID#
         CL    G64R4,XSI_MAX_ID#
         JNH   ID_OK

* The next available StringID has reached the maximum ID limit
         LGHI  G64R3,XSI_RC_FAILURE
         LG    G64R5,RETCODE_PTR
         ST    G64R3,RETCODE(,G64R5)
         LGHI  G64R2,XSI_DC_MAX_OUT_ID_LIST_ENTRIES
         LG    G64R3,DIAGCODE_PTR
         ST    G64R2,EXIT_DIAG_CODE(,G64R3)
         J     EXITING
```

```
ID_OK    DS    0H
* check if the storage area has enough space for the new node
         L     G64R5,XSI_FREE_SPACE
         LGFR  G64R9,G64R5
         CLG   G64R9,NEW_NODE_SIZE
         JNL   SPACE_OK

* Amount of free space is not enough to store the new node
         LGHI  G64R5,XSI_RC_FAILURE
         LG    G64R3,RETCODE_PTR
         ST    G64R5,RETCODE(,G64R3)
         LGHI  G64R2,XSI_DC_OUT_OF_STORAGE_SPACE
         LG    G64R3,DIAGCODE_PTR
         ST    G64R2,EXIT_DIAG_CODE(,G64R3)
         J     EXITING

* If XSI Storage contain enough free space and StringID hasn't
* reach the limit yet, then start inserting the string into the
* empty leaf node
SPACE_OK DS    0H
* Establish addressability for the XSI_NODE DSECT to the StringID Tree
         USING XSI_NODE,G64R9

         LG    G64R9,XSI_CURR_FREE@

* Set the left/right node to NULL
         XGR   G64R15,G64R15
         STG   G64R15,XSI_NODE_LEFT
         STG   G64R15,XSI_NODE_RIGHT

* Assign the next available StringID to the node
         LR    G64R2,G64R4       reg4 contains the StringID
         ST    G64R2,XSI_NODE_StringID
         ST    G64R8,XSI_NODE_STRLEN     Insert the string length

* Move the string from the parameter into the tree node
         LG    G64R14,STRID_PTR
         LA    G64R10,XSI_NODE_STRING
         ST    G64R2,STRID(,G64R14)  reg4 is next available StringID #
         LLGFR G64R2,G64R8          reg8 is the string length
         LG    G64R3,STR_PTR

* check the string length to see if it's greater than 256 characters
         CGHI  G64R2,256
         JNH   STR_256

* MVC the string in loops for each 256 characters
MVC_STR  DS    0H
         MVC   0(256,G64R10),0(G64R3)  copy 256 chars at a time
         LGHI  G64R14,256
         ALGR  G64R10,G64R14     move Reg10 256chars to right
         ALGR  G64R3,G64R14      move Reg3 256chars to right
         SLGR  G64R2,G64R14      Subtract string length by 256
         CGHI  G64R2,256         check if string length still > 256
         JH    MVC_STR

* Copy string into XSI tree node only if the string length or what
* is left to copy from the string is less than 256 characters
STR_256  DS    0H
         LGR   G64R15,G64R2
         BCTGR G64R15,0
         EX    G64R15,STRCOPY

* add null at the end of the string for higher level programming uses
         LLGFR G64R2,G64R8     reg8 is original string length
         XGR   G64R3,G64R3
         STC   G64R3,XSI_NODE_STRING(G64R2)
```

```
* Setting the address of the new tree node to the StringID list
         LLGFR G64R10,G64R4   reg3 is the current insertion StringID #
         SLAG  G64R10,G64R10,3
         STG   G64R9,XSI_ID_LIST-8(G64R10)

* re-calculate the total amount of the XSI free space
         LG    G64R2,NEW_NODE_SIZE
         LGFR  G64R10,G64R5    reg5 is the old free space
         SLGR  G64R10,G64R2
         LR    G64R5,G64R10
         ST    G64R5,XSI_FREE_SPACE

* Increment the XSI_NEXT_ID# value by one
         LGHI  G64R10,1
         ALR   G64R4,G64R10
         ST    G64R4,XSI_NEXT_ID#

* assign parent node pointer to point to this new node
         LG    G64R10,INSERTION_ADDR_PTR
         STG   G64R9,INSERTION_PTR(,G64R10)
         ALGR  G64R9,G64R2
         STG   G64R9,XSI_CURR_FREE@

* Insertion of a new string to the tree is completed
         LGHI  G64R3,XSI_RC_SUCCESS
         LG    G64R5,RETCODE_PTR
         ST    G64R3,RETCODE(,G64R5)
         LG    G64R5,DIAGCODE_PTR
         L     G64R2,EXIT_DIAG_CODE(,G64R5)
         J     EXITING

*********************************************************************
*                                                                   *
* SEARCH_TREE - search the tree for input string and return the    *
*              StringID if found or call INSERT_STRING to create  *
*              a new node for the new string                      *
*                                                                   *
*        Iteratively going down the tree base on the string to    *
*        determine going left or right leaf. If no identical      *
*        string is found, then INSERT_STRING will be called or    *
*        the identified string will be returned with its StringID *
*                                                                   *
* Input: The input parameters passed to the program.               *
*                                                                   *
* Output: STRID - the index of the inserted or found string        *
*                                                                   *
*        RETCODE - XRC_SUCCESS if string is found within tree      *
*                                                                   *
*********************************************************************
SEARCH_TREE DS    0H
         LGR   G64R9,G64R2  reg2 is the current searching node address

* Establish addressability for the XSI_NODE DSECT to the StringID Tree
         USING XSI_NODE,G64R9

         LG    G64R0,STR_PTR
         LG    G64R1,STRLEN_PTR
LOOP_IN  DS    0H
* string compare and decide to branch toward left or right leaf
         LA    G64R14,XSI_NODE_STRING
         LLGF  G64R15,XSI_NODE_STRLEN

* CLCL requires to use the registers in pair, so R0 contains the
* string characters while R1 requires to contain the string length.
* So CLCL will knows how many characters in the string to be compared
```

```
         CLCL  G64R0,G64R14
         JH    RIGHT_TREE
         JL    LEFT_TREE

* String is found in the StringID tree and its corresponding
* StringID will be saved and returned
         LG    G64R5,STRID_PTR
         L     G64R10,XSI_NODE_StringID
         ST    G64R10,STRID(,G64R5)
         LGHI  G64R4,XSI_RC_SUCCESS
         LG    G64R5,RETCODE_PTR
         ST    G64R4,RETCODE(,G64R5)
         LG    G64R5,DIAGCODE_PTR
         L     G64R2,EXIT_DIAG_CODE(,G64R5)
         J     EXITING

LEFT_TREE DS    0H
* Check if the left leaf is empty, if empty then go insert string,
* if not empty, then branch to the left leaf and continue searching
         LG    G64R2,XSI_NODE_LEFT
         LTGR  G64R2,G64R2
         JNZ   GO_LEFT

* Left leaf node is empty, so insert the new string into it
         LA    G64R10,XSI_NODE_LEFT
         STG   G64R10,INSERTION_ADDR_PTR
         J     INSERT_STRING

GO_LEFT  DS    0H
         LGR   G64R9,G64R2
         J     LOOP_IN

RIGHT_TREE DS    0H
* Check if the right leaf is empty, if empty then go insert string,
* if not empty, then branch to the right leaf and continue searching
         LG    G64R2,XSI_NODE_RIGHT
         LTGR  G64R2,G64R2
         JNZ   GO_RIGHT

* Right leaf node is empty, so insert the new string into it
         LA    G64R10,XSI_NODE_RIGHT
         STG   G64R10,INSERTION_ADDR_PTR
         J     INSERT_STRING

GO_RIGHT  DS    0H
         LGR   G64R9,G64R2
         J     LOOP_IN

****************************************************************
*** EXITING THE StringID SERVICE EXIT ROUTINE                ***
****************************************************************
EXITING  DS    0H
         ST    G64R2,XSI_DIAG_CODE

EXITIDI  DS    0H
         DROP  G64R6,G64R7,G64R9
         LMG   G64R14,G64R12,8(G64R13)
         BR    G64R14

DATA     DS    0F
         DC    A(GXLE4IDI)
STRCOPY  MVC   0(0,G64R10),0(G64R3)
EYECHAR  DC    CL8'XSIEYECA'



DATD     DSECT
```

```
         DS    0D
* Size of new node being inserted: node header + var string length
NEW_NODE_SIZE DS FD
* insertion pointer is needed when new node is being inserted and
* their parent node's ptr address will be saved in this insertion
* pointer. When the new node is created and initialized with data,
* the address in insertion_ptr will be assigned to point to the
* new node, completing the insertion process of new node into tree
INSERTION_ADDR_PTR DS AD

G64R0                EQU   0
G64R1                EQU   1
G64R2                EQU   2
G64R3                EQU   3
G64R4                EQU   4
G64R5                EQU   5
G64R6                EQU   6
G64R7                EQU   7
G64R8                EQU   8
G64R9                EQU   9
G64R10               EQU   10
G64R11               EQU   11
G64R12               EQU   12
G64R13               EQU   13
G64R14               EQU   14
G64R15               EQU   15

*********************************************************************
* Description of the XSI Structure:                                 *
*   The XSI structure is including the XSI Header, StringID array  *
*   list, the dynamic area, and end with empty storage area space  *
*   for the StringID tree                                          *
*      o XSI Header: Contain information for the XSI Structure to  *
*                    operate correctly (space management)          *
*      o StringID list: List of pointers that indexed by StringID  *
*                       and each ptr point to the corresponding    *
*                       leaf node within the StringID tree         *
*      o Dynamic Area: Used by ASAENTRY where all local variables, *
*                      stacks, and storage usage will take up the  *
*                      storage space within the dynamic area       *
*      o Empty Storage: managed by the XSI header and is used for  *
*                       storing the tree structure for StringID    *
*********************************************************************
XSI                DSECT         StringID storage area
XSI_HEADER         DS    0D
XSI_EYE_CATCHER    DS    CL8     eye-catcher string
XSI_SYM_MAX_SIZE   DS    1F      max string len for symbol
XSI_DIAG_CODE      DS    1F      exit diagnostic code
XSI_NEXT_ID#       DS    1F      next usable StringID num
XSI_MAX_ID#        DS    1F      max num of StringID nodes
XSI_TOTAL_SIZE     DS    1F      sys_svc_parm total size
XSI_FREE_SPACE     DS    1F      space left for data
XSI_CURR_FREE@     DS    1D      64bit current @ of free space
XSI_TREE_HEAD@     DS    1D      64bit location of tree head
XSI_DYN_AREA       DS    1D      64bit address of dynamic area
XSI_HEADER_END     DS    0X
XSI_ID_LIST        DS    1D      64bit ID list for quick search


*********************************************************************
* StringID Tree Node Header                                         *
*********************************************************************
XSI_NODE           DSECT
XSI_NODE_HEADER    DS    0D
XSI_NODE_LEFT      DS    1D      64bit LESS THAN PARENT NODE
XSI_NODE_RIGHT     DS    1D      64bit GREATER THAN PARENT NODE
XSI_NODE_StringID  DS    1F      STRING ID VALUE
```

```
XSI_NODE_STRLEN     DS    1F      LENGTH OF THE STRING
XSI_NODE_HEADER_END DS    0X
XSI_NODE_STRING     DS    C       THE ACTUAL STRING

**********************************************************************
* Description of the Parameters:                                     *
*   Input: SYS_SVC_PARM - Contain the XSI structure information      *
*          STR - The string that will be inserted into the tree      *
*          STRLEN - Length of the current string needed to insert    *
*                   Length is in number of bytes of the characters   *
*          CCSID - Identifier that identify character set of string  *
*   Output: STRID - the index of the inserted string                 *
*           EXIT_DIAG_CODE - XSI_DC_XXX provide detail to retcode    *
*           RETCODE - XRC_FAILURE if failed to insert/search for str *
*                     XRC_SUCCESS if string inserted/found           *
**********************************************************************
PARMLIST             DSECT
SYS_SVC_PARM_PTR     DS    1D
STR_PTR              DS    1D
STRLEN_PTR           DS    1D
STRID_PTR            DS    1D
CCSID_PTR            DS    1D
DIAGCODE_PTR         DS    1D
RETCODE_PTR          DS    1D

********************************************************************
* EXIT DIAGNOSTIC CODE                                             *
********************************************************************
* String length passed into gxle1idi is out of bound
XSI_DC_INCORRECT_PARM_STRLEN           EQU   11
* Tries to insert new string, but allocated storage is full
XSI_DC_OUT_OF_STORAGE_SPACE            EQU   12
* Eye Catcher string is incorrect
XSI_DC_INCORRECT_EYE_CATCHER           EQU   15
* StringID list is max out with entries,requires larger list size
XSI_DC_MAX_OUT_ID_LIST_ENTRIES         EQU   16

********************************************************************
* RETURN CODE                                                      *
********************************************************************
XSI_RC_SUCCESS        EQU  0
XSI_RC_FAILURE        EQU  1

SYS_SVC_PARM          EQU   0
STR                   EQU   0
STRLEN                EQU   0
STRID                 EQU   0
CCSID                 EQU   0
EXIT_DIAG_CODE        EQU   0
RETCODE               EQU   0

INSERTION_PTR         EQU   0
                      END
```

GXLE4IDR

This example module uses the input StringID to access a table and returns the address & length of the string associated with the StringID. The string is saved in the storage pointed to by SYS_SVC_PARM during the initialization of the parser (GXL1INI) and in StringID processing (GXLE1INI).

Register 14 is used to store the return address, which must be kept intact in order to exit this subroutine correctly.

This example module assumes register one, which is passed in from the caller, contains the address of the parameter list. The following input variables are used in the example module:

SYS_SVC_PARM
: Address of storage area that the caller of the z/OS XML parser wants to pass to the exit. It also contains the XSI structure information.

STRID StringID used for indexing the list.

The following output variables are used in the example module:

STR_ADDR
: Address of string from requested StringID.

STRLEN
: The length of the string found by StringID.

DIAG_CODE
: Contains diagnostic information.

XSI_DC_INCORRECT_StringID_OUTOFBOUND
: *STRID* length is out of bound.

XSI_DC_INCORRECT_ID_LOCATION_ERROR
: StringID does not match.

XSI_DC_INCORRECT_EYE_CATCHER
: Eye catcher is incorrect.

RETCODE

XSI_RC_FAILURE
: The string cannot be retrieved.

XSI_RC_SUCCESS
: The string was retrieved successfully.

```
GXLE4IDR CSECT                        Set up control section
GXLE4IDR AMODE 64                     Address mode of module
GXLE4IDR RMODE ANY                    Residence of module
         USING DATA+0,G64R12          Relative Branching
         ENTRY GXLE4IDR               external entry point
         STMG  G64R14,G64R12,8(G64R13) Save regs
         LARL  G64R12,DATA            Load DATA Address

         J     IDSTART  Jump to mainline of code
EYECATCH DS    0H
         DC    CL8'GXLE4IDR'
IDSTART  DS    0H

* establish addressability for the PARMLIST DSECT to the PARMLIST
         USING PARMLIST,G64R1
* establish addressability for the XSI DSECT to the SYS_SVC_PARM
```

```
         USING XSI,G64R6

**********************************************************
* Check if the XSI eye catcher string matches correctly *
**********************************************************
         LG    G64R5,SYS_SVC_PARM_PTR
         LG    G64R6,SYS_SVC_PARM(,G64R5)
         CLC   XSI_EYE_CATCHER,EYECHAR
         JE    PROCEED

***********************************************************
* Eye catcher is invalid in the XSI structure, so we not *
* sure if the data in the XSI structure is still valid,  *
* exiting subroutine, and return to the caller           *
***********************************************************
         LGHI  G64R9,XSI_RC_FAILURE
         LG    G64R8,RETCODE_PTR
         ST    G64R9,RETCODE(,G64R8)
         LGHI  G64R11,XSI_DC_INCORRECT_EYE_CATCHER
         LG    G64R10,DIAGCODE_PTR
         ST    G64R11,EXIT_DIAG_CODE(,G64R10)
         J     EXITIDR

*********************************************************************
*** MAINLINE CODE STARTS HERE:                                    ***
*********************************************************************
*** GXLE4IDR.PLX - Search & Retrieve String from StringID         ***
*********************************************************************

***********************************************************
* The XSI structure is intact and eye catcher is correct *
***********************************************************
PROCEED  DS    0H

*********************************************************************
* If string ID contain a valid index, gxlp1idr will proceed and    *
* retrieve the corresponding string from the tree                  *
*********************************************************************
         LG    G64R3,STRID_PTR
         ICM   G64R5,15,STRID(G64R3)
         JZ    BAD_ID
         CL    G64R5,XSI_NEXT_ID#
         JL    RETRIEVE

* The StringID value is out of bound
BAD_ID   DS    0H
         LGHI  G64R5,XSI_RC_FAILURE
         LG    G64R3,RETCODE_PTR
         ST    G64R5,RETCODE(,G64R3)
         LGHI  G64R2,XSI_DC_INCORRECT_StringID_OUTOFBOUND
         LG    G64R3,DIAGCODE_PTR
         ST    G64R2,EXIT_DIAG_CODE(,G64R3)
         J     EXITING

RETRIEVE DS    0H
         LLGFR G64R3,G64R5 reg5 contains the next available StringID #
         SLAG  G64R3,G64R3,3   multiply the number of StringID by 8

* establish addressability for the XSI_NODE DSECT to the StringID Tree
         USING XSI_NODE,G64R4

         LG    G64R4,XSI_ID_LIST-8(G64R3)
         CL    G64R5,XSI_NODE_StringID check if the StringID is correct
         JE    DONE

* The value of the StringID in the tree is not consistent with
* the value in the StringID list, exiting subroutine
```

```
         LGHI  G64R3,XSI_RC_FAILURE
         LG    G64R5,RETCODE_PTR
         ST    G64R3,RETCODE(,G64R5)
         LGHI  G64R2,XSI_DC_INCORRECT_ID_LOCATION_ERROR
         LG    G64R3,DIAGCODE_PTR
         ST    G64R2,EXIT_DIAG_CODE(,G64R3)
         J     EXITING

* Successful in finding the StringID
DONE     DS    0H
         LG    G64R5,STR_ADDR_PTR
         LA    G64R7,XSI_NODE_STRING
         STG   G64R7,STR_ADDR(,G64R5)     Store the string pointer
         LG    G64R2,STRLEN_PTR
         L     G64R7,XSI_NODE_STRLEN
         ST    G64R7,STRLEN(,G64R2)       Store the string length

* Set the return code to successful and return back to caller
         LGHI  G64R7,XSI_RC_SUCCESS
         LG    G64R5,RETCODE_PTR
         ST    G64R7,RETCODE(,G64R5)
         LG    G64R5,DIAGCODE_PTR
         L     G64R2,EXIT_DIAG_CODE(,G64R5)
         J     EXITING

***************************************************************
*** EXITING THE StringId SERVICE EXIT ROUTINE              ***
***************************************************************
EXITING  DS    0H
         ST    G64R2,XSI_DIAG_CODE

EXITIDR  DS    0H
         DROP  G64R1,G64R4,G64R6
         LMG   G64R14,G64R12,8(G64R13)
         BR    G64R14

DATA     DS    0F
         DC    A(GXLE4IDR)
EYECHAR  DC    CL8'XSIEYECA'


G64R0                EQU   0
G64R1                EQU   1
G64R2                EQU   2
G64R3                EQU   3
G64R4                EQU   4
G64R5                EQU   5
G64R6                EQU   6
G64R7                EQU   7
G64R8                EQU   8
G64R9                EQU   9
G64R10               EQU   10
G64R11               EQU   11
G64R12               EQU   12
G64R13               EQU   13
G64R14               EQU   14
G64R15               EQU   15

********************************************************************
* Description of the XSI Structure:                                *
*   The XSI structure is including the XSI Header, StringID array  *
*   list, the dynamic area, and end with empty storage area space  *
*   for the StringID tree                                          *
*      o XSI Header: Contain information for the XSI Structure to  *
*                    operate correctly (space management)          *
*      o StringID list: List of pointers that indexed by StringID  *
*                       and each ptr point to the corresponding    *
*                       leaf node within the StringID tree         *
*      o Dynamic Area: Used by ASAENTRY where all local variables, *
*                      stacks, and storage usage will take up the  *
*                      storage space within the dynamic area       *
*      o Empty Storage: managed by the XSI header and is used for  *
*                       storing the tree structure for StringID    *
********************************************************************
XSI                 DSECT         StringID storage area
XSI_HEADER          DS    0D
XSI_EYE_CATCHER     DS    CL8     eye-catcher string
XSI_SYM_MAX_SIZE    DS    1F      max string len for symbol
XSI_DIAG_CODE       DS    1F      exit diagnostic code
XSI_NEXT_ID#        DS    1F      next usable StringID num
XSI_MAX_ID#         DS    1F      max num of StringID nodes
XSI_TOTAL_SIZE      DS    1F      sys_svc_parm total size
XSI_FREE_SPACE      DS    1F      space left for data
XSI_CURR_FREE@      DS    1D      64bit current @ of free space
XSI_TREE_HEAD@      DS    1D      64bit location of tree head
XSI_DYN_AREA        DS    1D      64bit address of dynamic area
XSI_HEADER_END      DS    0X
XSI_ID_LIST         DS    1D      64bit ID list for quick search


********************************************************************
* StringID Tree Node Header                                        *
********************************************************************
XSI_NODE            DSECT
XSI_NODE_HEADER     DS    0D
XSI_NODE_LEFT       DS    1D      64bit LESS THAN PARENT NODE
XSI_NODE_RIGHT      DS    1D      64bit GREATER THAN PARENT NODE
XSI_NODE_StringID   DS    1F      STRING ID VALUE
XSI_NODE_STRLEN     DS    1F      LENGTH OF THE STRING
XSI_NODE_HEADER_END DS    0X
XSI_NODE_STRING     DS    C       THE ACTUAL STRING

********************************************************************
* Description of the Parameters:                                   *
*    Input: SYS_SVC_PARM - Contain the XSI structure information   *
*           STRID - String ID used for indexing the list           *
*    Output: STR_ADDR - address of string from requested StringID  *
*            STRLEN - the length of the string found by StringID   *
*            EXIT DIAG_CODE - XSI_DC_XXX details to return code    *
*                            (Set only if error found)             *
*            RETCODE - XSI_RC_FAILURE if string cannot be retrieved*
*                      XSI_RC_SUCCESS if string retrieved ok       *
********************************************************************
PARMLIST             DSECT
SYS_SVC_PARM_PTR     DS    1D
STR_ADDR_PTR         DS    1D
STRLEN_PTR           DS    1D
STRID_PTR            DS    1D
DIAGCODE_PTR         DS    1D
RETCODE_PTR          DS    1D

********************************************************************
* EXIT DIAGNOSTIC CODE                                             *
********************************************************************
* Caller tries to retrieve string, but StringID is out of bound
XSI_DC_INCORRECT_StringID_OUTOFBOUND   EQU   13
* After gxle1idr obtained the pointer to the tree node, the string
* ID is not matching with the StringID passed in from parameter
XSI_DC_INCORRECT_ID_LOCATION_ERROR     EQU   14
* Eye Catcher string is incorrect
XSI_DC_INCORRECT_EYE_CATCHER           EQU   15

********************************************************************
* RETURN CODE                                                      *
```

```
*********************************************************************
XSI_RC_SUCCESS      EQU  0
XSI_RC_FAILURE      EQU  1

SYS_SVC_PARM          EQU   0
STR_ADDR              EQU   0
STRLEN                EQU   0
STRID                 EQU   0
EXIT_DIAG_CODE        EQU   0
RETCODE               EQU   0
                      END
```

Appendix L. Supported encodings

The following table displays the encodings supported by z/OS XML System Services. Displayed in the table are the code page names, associated CCSID and equate names. Assembler callers use equate names without the ″GXLH″ prefix.

Rule: If you require a different encoding, you must first convert to one of the below before invoking the z/OS XML parser.

Table 29. Code page CCSID values

Code page	CCSID	Equate Names
UTF-8	1208	GXLHXEC_ENC_UTF_8
UTF-16 (big endian)	1200	GXLHXEC_ENC_UTF_16
EBCDIC/IBM-037	37	GXLHXEC_ENC_IBM_037
EBCDIC/IBM-273	273	GXLHXEC_ENC_IBM_273
EBCDIC/IBM-277	277	GXLHXEC_ENC_IBM_277
EBCDIC/IBM-278	278	GXLHXEC_ENC_IBM_278
EBCDIC/IBM-280	280	GXLHXEC_ENC_IBM_280
EBCDIC/IBM-284	284	GXLHXEC_ENC_IBM_284
EBCDIC/IBM-285	285	GXLHXEC_ENC_IBM_285
EBCDIC/IBM-297	297	GXLHXEC_ENC_IBM_297
EBCDIC/IBM-500	500	GXLHXEC_ENC_IBM_500
EBCDIC/IBM-871	871	GXLHXEC_ENC_IBM_871
EBCDIC/IBM-1047	1047	GXLHXEC_ENC_IBM_1047
EBCDIC/IBM-1140	1140	GXLHXEC_ENC_IBM_1140
EBCDIC/IBM-1141	1141	GXLHXEC_ENC_IBM_1141
EBCDIC/IBM-1142	1142	GXLHXEC_ENC_IBM_1142
EBCDIC/IBM-1143	1143	GXLHXEC_ENC_IBM_1143
EBCDIC/IBM-1144	1144	GXLHXEC_ENC_IBM_1144
EBCDIC/IBM-1145	1145	GXLHXEC_ENC_IBM_1145
EBCDIC/IBM-1146	1146	GXLHXEC_ENC_IBM_1146
EBCDIC/IBM-1147	1147	GXLHXEC_ENC_IBM_1147
EBCDIC/IBM-1148	1148	GXLHXEC_ENC_IBM_1148
EBCDIC/IBM-1149	1149	GXLHXEC_ENC_IBM_1149

Appendix M. 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/systems/z/os/zos/bkserv/`

Notices

Programming Interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS XML System Services.

Trademarks

Acknowledgments

Contact your system administrator.

Index

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

z/OS
XML System Services User's Guide and Reference

Publication No. SA23-1350-03

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: mhvrcfs@us.ibm.com

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

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

Readers' Comments — We'd Like to Hear from You
SA23-1350-03

Cut or Fold
Along Line

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

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
MHVRCFS, Mail Station P181
2455 South Road
Poughkeepsie, NY
12601-5400

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

SA23-1350-03

Cut or Fold
Along Line

Program Number: 5655-G52

Printed in USA

SA23-1350-03