XML Extensions Readme

Version 10

Contents

This document contains the following sections:

Introduction

Thank you for using XML Extensions. This readme contains important information that applies to the version 10 release for Microsoft Windows and UNIX platforms. Portions of this file also pertain to earlier versions of the product.

Documentation

The complete XML Extensions documentation, the *XML Extensions User's Guide, First Edition*, is provided in PDF format on the installation disk.

The *XML Extensions User's Guide, First Edition*, is the base document for the XML Extensions component in RM/COBOL version 10 and covers the XML Extensions features at the time of the version 9 release.

The *XML Extensions User's Guide* supplement document for version 10 is this readme file; the documentation changes since version 9 are described in "Version 10 Improvements" on page 2.

Useful Links

Liant Software Corporation
Link: www.liant.com (search for XML)

Microsoft Downloads
Link: www.microsoft.com/downloads/search.asp

Microsoft Developer Network
Link: msdn.microsoft.com/default.asp (search for: XML Core)
Link: msdn.microsoft.com/xml/default.asp

The XML C Library for Gnome
Link: xmlsoft.org

World Wide Web Consortium (W3C)
Link: www.w3.org.
Link: www.w3.org/XML

OASIS XML Information
Link: www.xml.org

O'Reilly XML Information
Link: www.xml.com
Link: www.xml.com/axml/testaxml.htm, (annotated XML information)

XML Software Site
Link: www.xmlsoftware.com

Version 10 Improvements

The improvements in XML Extensions version 10 are primarily bug fixes, and most of these improvements have been distributed with various releases of Xcentrisity Business Information Server (BIS). The following information details additional improvements and documentation revisions.

RM/COBOL Object Version 13

XML Extensions now supports object version 13 and all future object versions that do not change the object symbol table version.

Documentation Changes

The following documentation changes to the *XML Extensions User's Guide, First Edition* for version 10 are described below.

- **Page 41, Chapter 3: COBOL Considerations.** Add the following text to the Note in "Windows Character Encoding."

  In version 9 and later of the runtime system, the ANSI code page can be selected as the native character set, in which case, XML Extensions uses the ANSI code page in use for the conversion to and from Unicode when using the local character encoding.

- **Page 46, Chapter 3: COBOL Considerations.** Add the following new topic under "Copy Files."

  REPLACE Statement Considerations

  The copy file, lixmlall.cpy, contains a REPLACE statement to define the XML statements. A COBOL REPLACE statement overrides any lexically-preceding REPLACE statement. Thus, in cases where the user's program contains a REPLACE statement, it may not be possible to use the lixmlall.cpy file. For this reason, the lixmlrpl.cpy copy file, which is copied by the lixmlall.cpy file, is provided as part of XML Extensions. The lixmlrpl.cpy file contains the operands of the REPLACE statement needed to define the XML statements, but not the REPLACE statement itself. Accordingly, the user's REPLACE statement may be augmented by copying lixmlrpl.cpy into the REPLACE statement as follows:

```
REPLACE
*> include user's replacements here
COPY "lixmlrpl.cpy". *> define XML statements
. *> end of combined REPLACE statement

COPY "lixmldef.cpy". *> XML data definitions
```

Note If there are multiple REPLACE statements in your source program, each REPLACE statement that precedes any XML statements needs to copy the lixmlrpl.cpy file into the REPLACE statement to preserve the statements for replacement.

The InstantSQL product has a copy file, lisqlall.cpy, which contains a REPLACE statement to define the SQL statements. In cases where InstantSQL is used with XML Extensions, neither the lixmlall.cpy nor the lisqlall.cpy copy file should be used. Instead, create a copy file (for example, named isqlxml.cpy) with the following contents:

```
REPLACE
*> optionally include user's replacements
COPY "lisqlrpl.cpy". *> define SQL statements
COPY "lixmlrpl.cpy". *> define XML statements
. *> end of combined REPLACE statement

COPY "lisqldef.cpy". *> SQL data definitions
COPY "lixmldef.cpy". *> XML data definitions
```

Use this copy file in place of lixmlall.cpy and lisqlall.cpy.

- **Page 48, Chapter 3: COBOL Considerations.** Delete the following paragraph from "Data Items (Data Structures)."

  Data items that are passed to XML Extensions must be in memory that is local to the COBOL program. Therefore, EXTERNAL data items or data items in the Linkage Section may not be used for import or export operations.

- **Page 55, Chapter 4: XML Considerations.** Add the following new topic to this chapter following "XSLT Stylesheet Files."

  Handling Spaces and Whitespace in XML

  XML Extensions normally strips trailing spaces from COBOL data items when exporting data and restores trailing spaces to COBOL data items when importing data. Leading spaces are also removed and added for justified data items. This default behavior can be modified using the XML SET FLAGS statement, but the default behavior is generally best. The normal treatment of leading and trailing spaces does not apply to FILLER data items or edited data items.

  Once the data is in XML, further consideration must be given to XML treatment of whitespace, which includes spaces, carriage returns, and line feeds. XML provides a built-in attribute named xml:space, which takes a value of "preserve" or "default." The value "preserve" specifies that whitespace in an element should be preserved. The value "default" specifies that leading and trailing whitespace may be removed and embedded whitespace may be normalized to a single space wherever it occurs. The value "default" is the default treatment of whitespace in XML and is generally not changed unless one is trying to produce poetry or other special output. When using XSLT stylesheets, the xsl:strip-space and xsl:preserve-space elements indicate how whitespace should be handled while transforming a document. Preserving whitespace is the default, but tools that generate XSLT stylesheets might insert xsl:strip-space elements.

  Be aware that when documents are transformed to HTML for display by a browser, many browsers strip whitespace as they are allowed to do. Displaying data in tables is generally necessary to align data in columns rather than using whitespace as it is generally done in COBOL report output.

- **Page 85, Chapter 6: xmlif Library Reference.** Add the following text to the Description section of "XML SET ENCODING."

  On Windows, the local character encoding matches the native character set of the runtime; that is, it is specified by the Windows ANSI code page or Windows OEM code page depending on the

native character set of the runtime system; see the *RM/COBOL User's Guide* for how to select the native character set for the runtime system. On UNIX, the local character encoding is specified by the value of the RM_ENCODING environment variable, with a default of RM_LATIN_9 if the variable is not defined.

- **Page 122, Appendix A: XML Extensions Examples.**

  In the Program Structure section of "Example 5: Export Text and Import Text", the Document Pointer references in the XML calls are shown with quotation marks surrounding them. These quotes should be removed.

  Note that this problem also occurs in the similar section of "Example 8: Export Text, Test Well-Formed, and Validate Text" on page 142. The same solution applies.

- **Page 183, Appendix C: XML Extensions Error Messages.** Revise the description of message number 43 as follows.

  Error - wrong COBOL object symbol table version

  The cobtoxml utility has determined that the COBOL object symbol table version in the specified object file is newer than was available when this version of XML Extensions was released and, therefore, may contain features that are not supported by XML Extensions. Check with Liant Software for updates to XML Extensions.

Version 9 Improvements

The following improvements have been incorporated into the version 9 release of XML Extensions. These improvements are primarily bug fixes. Most of these improvements have been distributed with various releases of Xcentrisity Business Information Server (BIS).

RM/COBOL Object Version 12

XML Extensions now supports RM/COBOL object version 12, which was introduced with RM/COBOL version 9.

UNIX Diagnostics

Better diagnostic information is returned when an XML IMPORT FILE or XML IMPORT TEXT statement fails due to an XSLT transform error.

Windows Stylesheet processing

On Windows, Stylesheets that used a literal result element were incorrectly encoded in UTF-16.

Missing Windows MSXML Parser

On Windows systems, a more descriptive diagnostic is returned if Microsoft's MSXML 4.0 parser is not installed.

Buffer Overrun Problem

The XML IMPORT FILE and XML IMPORT TEXT statements failed to verify that then input data would fit in the selected data structure. This has been fixed.

URL Recognition

Previously, only filenames that began with "http://" were recognized as URLs. This has been expanded to include filenames that begin with "https://".

Filename Extensions

Normally, if a filename extension is not present, one is added. For URLs, this is not practical. Filename extension processing has been modified so that an extension is not added if the name is a URL.

RUNPATH Search

The RUNPATH search sequence has been modified to ignore directory names that use the Universal Naming Convention (UNC) notation (for example, "//system/directory"). UNC names are normally used in an application that uses RM/InfoExpress. XML Extensions cannot access files directly through RM/InfoExpress. By ignoring UNC directory names, unnecessary time delays are avoided when performing a RUNPATH search.

COBTOXML Banner

The cobtoxml utility has been modified to display the banner when necessary command line parameters are omitted.

XML EXPORT FILE / XML EXPORT TEXT Blank Suppression

In prior versions, the XML EXPORT FILE and XML EXPORT TEXT statements would strip leading spaces from all non-numeric data items. Leading spaces are now stripped only from data items that are defined with the JUSTIFIED phrase.

Stylesheets with DTD

An XSLT stylesheet loading error has been fixed on Windows so that a Document Type Definition (DTD) may be specified in the stylesheet. A DTD is required in order to define entity references, such as " " or "©", which are often inserted by stylesheet editors, such as Dreamweaver or FrontPage. Several entity references are commonly used in XHTML that are not defined by default in XML. Versions of xmlif.dll prior to 9.03 inadvertently loaded the stylesheet with validation set to true because of a Microsoft default setting. Since a valid DTD cannot be constructed for an XSLT stylesheet, the validation would fail and prevent the successful load of the stylesheet. If the stylesheet contains undefined entity references other than those defined by default in XML, the load would also fail. For version 9.03 and later, xmlif.dll loads stylesheets with validation explicitly set to false. Thus, a DTD that references the XHTML entity definitions may be specified in the stylesheet.

Improved Namespace Support for Schema Validation

The XML VALIDATE DOCUMENT statement has been fixed so that validation against a schema that specifies a targetNamespace will work correctly on Windows. Versions of the xmlif.dll file prior to 9.03 incorrectly used the empty namespace "" when adding a schema. Thus, schemas that specified a targetNamespace value would cause a schema load failure (XML Extensions error 29) when referenced by XML VALIDATE DOCUMENT. Version 9.03 and later of xmlif.dll extract the targetNamespace value from the schema and use that value instead of the empty namespace value, thus allowing the schema to load successfully. (Note that schema validation on UNIX is currently unimplemented.)

Version 2 Improvements

The following improvements have been incorporated into the version 2 release of XML Extensions.

UNIX Support

The XML Extensions is now available on both UNIX and Windows. The Windows implementation continues to use Microsoft's MSXML parser. The UNIX implementation is based on the XML C library for Gnome (libxml2 and libxslt).

While the Windows implementation continues to support the use of schemas, the UNIX implementation does not. Schema support in the underlying XML parser (libxml2) was still under development at the time of the XML Extensions version 9 release.

DTD Support

Exporting of documents has been enhanced to include the ability to specify a document type definition (DTD). A DTD can be used to define entity names that are referred to by the values of FILLER data items in the COBOL data structure being exported.

Anonymous COBOL Structures

The process of exporting and importing documents has been improved so that anonymous data structures may be used. An anonymous data structure is any data area that is the same size or larger than the data structure indicated by the template file. Anonymous data structure support means that exporting or importing can be done from or to LINKAGE SECTION data items that are based on either arguments passed to a called program or a pointer using the SET statement (for example, exporting from or importing to dynamically allocated memory). Anonymous data structure support also means that exporting or importing can be done from or to external data items, which are available to multiple programs, not just the program from which the data structure was captured using the **cobtoxml** utility. This change makes it easier to modularize a program that exports or imports XML data.

Relaxed Time Stamp Checking

It is no longer necessary for the compilation timestamp in the object program to match the **cobtoxml** timestamp in the template file. That is, the program may be recompiled without running the **cobtoxml** utility. It is necessary to run **cobtoxml** only when the relevant data structure(s) have changed.

UTF-8 Data Encoding

Support has been added to both the UNIX and Windows implementations of the XML Extensions to allow the in-memory representation of XML document element content to use UTF-8 encoding. UTF-8 is a format for representing Unicode. This may be useful for COBOL applications that wish to pass UTF-8 encoded data to other processes. XML documents are normally encoded using Unicode (UTF-8 is one of several representations of Unicode). The XML Extensions always generates XML documents with UTF-8 data encoding.

The XML SET ENCODING Statement

This statement has one parameter, the value for which must be either “local” or “utf-8”. If the value is “local”, then the character encoding used by the operating system is selected. If the value is “utf-8”, then the data is treated as UTF-8 encoded. The parameter value is case-insensitive. Hyphen and underscore characters are optional. For example, “LOCAL”, “Local”, and “local” are equivalent. “UTF-8”, “Utf_8”, and “utf8” are also equivalent.

This statement allows the developer to specify the character encoding of data within a COBOL data structure. The developer may use this statement to switch between the local character encoding and UTF-8. The XML SET ENCODING statement does not affect the encoding of the XML document; instead, it affects the encoding of the data in the COBOL program.

The XML SET ENCODING statement returns an error status value if the value of the “Encoding” parameter is not recognized. The default value is “local”. If XML SET ENCODING is never called, the default is used.

An example of this statement follows:

```
XML SET ENCODING "local".
IF NOT XML-OK GO TO EXIT-1.
```

The RM_ENCODING Environment Variable

If the value set by XML SET ENCODING is “local”, then the environment variable RM_ENCODING is used to determine an alternate “local” data encoding. The interpretation of this environment variable varies between UNIX and Windows as discussed in the following paragraphs.

Windows Character Encoding

For Windows, the RM/COBOL runtime uses OEM data encoding. The only “local” data encoding supported on the Windows implementation of XML Extensions is likewise OEM. The RM_ENCODING environment variable is ignored by the Windows implementation of XML Extensions.

UNIX Character Encoding

For UNIX, the RM/COBOL runtime is not concerned with the data encoding used by the underlying operating system. Liant believes that Latin-1 (ISO-8859-1) is important for the U.S. and that Latin-9 (ISO-8859-15) is important for Western Europe (because it contains the Euro currency symbol). The values RM_LATIN_1 and RM_LATIN_9 have been defined for use with the RM_ENCODING environment variable. These values are used to designate that either Latin-1 or Latin-9 is to be used as the local character encoding. Liant also provides internal translation functions that convert between either Latin-1 or Latin-9 (in COBOL memory) and UTF-8 (in the XML document). The value of the environment variable is case-insensitive with hyphen and underscore characters being optional. For example, “RM_LATIN_1”, “Rm-Latin-1”, and “rmlatin1” are equivalent.

Other conversions are possible using the iconv library. If the value of the RM_ENCODING environment variable is not RM_LATIN_1 or RM_LATIN_9, then the value is passed as is to the iconv library for conversion. In this case, the spelling may need to be exact (for example, the value may be case-sensitive and hyphens and underscores would be required). The exact spelling of the value of the RM_ENCODING environment variable is specific to the iconv library on the platform in use.

The RM_ICONV_NAME Environment Variable

If present, the value of the RM_ICONV_NAME environment variable is used to locate the iconv library (which must be a shared object) on the local system.

For example:

```
RM_ICONV_NAME=/usr/local/bin/iconv.so
```

If the RM_ICONV_NAME environment variables is not set, then the “PATH” environment variable is searched for either of the specific names “iconv.so” or “libiconv.so”.

Problems and Workarounds

Parser Errors on UNIX

On UNIX implementations of XML Extensions, the XML C Library for Gnome may display parser error information on standard error (stderr). This can cause problems with information displayed by the RM/COBOL runtime when parser errors are encountered.

The following is recommended as a work-around. The developer may modify the script that executes the RM/COBOL runtime so that standard error is piped to a file. The syntax for piping to standard error varies with the shell in use, but is similar to the following:

```
runcobol myapplication 2> stderr.txt
```

Development System

The XML Extensions development system works in conjunction with the RM/COBOL development system (version 8 or later).

Deployment System

The XMLIF library (xmlif.dll on Windows or xmlif.so on UNIX) works in conjunction with the RM/COBOL runtime system (version 8 or later).

Microsoft MSXML 4.0 Parser

The Microsoft MSXML 4.0 parser is required for development and deployment of applications using XML Extensions on Windows. Both the development and deployment system installation procedures install the Microsoft MSXML 4.0 Parser.

The XML C Library for Gnome

The XML C library for Gnome is required for development and deployment of applications using XML Extensions on UNIX. Both the development and deployment installation procedures install the XML C library for Gnome.