z/VSE XML Parser Interface Description

Since VSE/ESA 2.7 there is a XML Parser implementation that enables you to parse and create XML documents (see http://www.w3.org/XML/) with your VSE programs. With APAR PQ78973 / PTF UQ81044 this function has been made available for VSE/ESA 2.6. Please make sure you have installed the latest service level, but at least PK18960 (VSE/ESA 2.6 and 2.7) or PK18932 (z/VSE 3.1) (see http://www.ibm.com/systems/z/os/zvse/support/connectors.html).

Overview

The z/VSE XML Parser can be used in a CICS application as well as in a batch application. Since it is implemented in LE/C, it needs to be called from an LE conform program, or the LE environment must be set up prior to calling the XML Parser. In CICS you do not need to take care about that, since CICS does this for you (EXEC CICS LINK).

There are two types of XML Parser interfaces:

- a SAX (Simple API for XML, see http://www.saxproject.org/) like interface that uses callback functions for each XML element. Due to the callback mechanism this interface is designed to work best with C programming language applications, but it can also be used by any LE enabled programming language.
- a DOM (Document Object Model, see http://www.w3.org/DOM/) like interface that builds a tree representation of the XML data in memory. When running in batch this interface can be used by a LE conform program, when running in CICS it can be used by any kind of application since it uses an EXEC CICS LINK interface.

The XML Parser can also be used to create an XML data stream from a given tree representation in memory (DOM).

The XML Parser interface is defined in the C-Header files IESXMLPH.H and IESXMLAH.H in PRD1.BASE. IESXMLPH.H defines the SAX like interface; IESXMLAH.H defines the DOM like interface.

The XML Parser supports XML version 1.0. XML namespaces are not interpreted, but are kept as attributes.

The XML Parser assumes that the XML data has already been translated into an EBCDIC codepage that is equivalent to UTF-8. The XML Parser only supports single byte character sets like UTF-8; UTF-16 is not supported.

To use the XML Parser one or more simple calls has to be made from the user application to the XML Parser. The XML Parser is given by the following programs:

- IESXMLCA.PHASE – CICS COMMAREA interface (DOM)
- IESXMLAP.PHASE – Batch COMMAREA interface (DOM)
- IESXMLPR.OBJ – Callback interface (SAX), linked to calling program
- IESXMLCT.PHASE – Callback interface (internal)

DOM like interface

When running in batch the phase IESXMLAP has to be fetched and a direct call to its entry point has to be performed (Dynamic call in COOL). When running in CICS a call is made by EXEC CICS LINK to program IESXMLCA. In both cases a parameter area (COMMAREA) is passed to the XML Parser containing all necessary parameters. In batch, a pointer to the parameter area is passed as argument, in CICS the parameter area is given by the COMMAREA. A return code is returned in R15 for batch and in RESP2 for CICS, and also in a field in the COMMAREA.

A typical code fragment for batch in C would look as follows:

```
#include “IESXMLAH.h”

typedef int (*XMLAPI)(void *ca);

{
  XMLAPI entrypoint;
  struct _b2tcommarea  b2t;

  // setup the commarea (not shown here)

  // Fetch the phase (for multiple calls, do that only the first time)
  entrypoint = (XMLAPI)fetch(“IESXMLAP”);

  // call the parser
  rc = entrypoint(&b2t);

  // release the phase (when you no longer need it)
  release((void(*)())entrypoint)

  if (rc!=0)
    ...
};
```

A typical code fragment for CICS in C would look as follows:

```
#include “IESXMLAH.h”

{
  struct _b2tcommarea  b2t;

  // setup the (not shown here)

  // call the parser
  EXEC CICS LINK PROGRAM("IESXMLCA")
                 COMMAREA(&b2t) LENGTH(sizeof(b2t))
                 RESP(rc) RESP2(rc2);
  if (rc2!=0)
    ...
}
```


The DOM like interface provides 3 functions:

- BUFFER2TREE Parse XML data and allocate a tree representation in memory
- TREE2BUFFER Create XML data from a tree representation in memory
- FREETREE Free a tree representation in memory (FREEVIS)

The FREETREE function must be used to free a tree that has been allocated by BUFFER2TREE. The XML Parser allocates the memory needed for the tree representations in larger junks. Since it is not known to the user how the parser spreads the data over the allocated memory, the FREETREE function must be used to free the whole tree representation.

For each of the 3 functions there is a COMMAREA format. Each of the 3 COMMAREAs starts with a field command, which specifies the function to be executed:

BUFFER2TREE:

```
_Packed struct _b2tcommarea
{
 unsigned int     command;          // (in) BUFFER2TREE function code (1)
 void *           xmlBuffer;        // (in) buffer with XML data
 unsigned int     xmlBufferLen;     // (in) buffer length
 void *           xmlTreeRoot;      // (out) result tree
 unsigned int     errorCode;        // (out) error code if any
 unsigned int     parserErrorCode;  // (out) error code of parser if any
 unsigned int     errorLineNumber;  // (out) line witch has an error
 unsigned int     lastCall;         // (in) 1 if last part of XML data
 unsigned int     options;          // (in) parser options
 unsigned int     pageSize;         // (in) page size of pages allocated
 void *           saveArea;         // (in/out) save area, used internally
 char              codepage[12];    // (in) EBCDIC codepage (optional)
};
```

The BUFFER2TREE function can be called multiple times if one XML document does not fit completely into a buffer or the application wants to process the XML data in smaller junks. The XML Parser saves the parsing state between the calls and therefore can continue to parse a XML documents with each call. Field saveArea is set by the parser to point to an internally used save area. It has to be passed to further calls when parsing XML data in smaller junks. For the first call to the parser, field saveArea can be set to 0.

The XML data to parse is supplied by a user allocated buffer in field xmlBuffer. Field xmlBufferLen specifies the length of the XML data to parse. If all data fits in the supplied buffer, field lastCall must be set to 1, if only a partial XML document fits into the buffer, lastCall must be set to 0 and the BUFFER2TREE functions has to be called several times, until the last part of the XML document has been reached (the application must then set lastCall to 1).

The field options allow specifying parser options:

- SKIP_WSDATA 0x01 skip data witch contain white spaces only
- SKIP_PI 0x02 skip processing instructions
- SKIP_DATA 0x04 skip all data
- SKIP_CDATA 0x08 skip all CDATA elements

- SKIP_COMMENT 0x10 skip comments

The field pageSize specifies the size of pages that are allocated when creating the tree representation. If running under CICS a minimum page size of 4KB is used. Specifying 0 causes each tree element to be allocated separately (not recommended). If large XML documents are being parsed, a larger page size decreases memory fragmentation and allocation overhead. Specifying a page size, that is too large, waists memory when small XML documents are parsed.

The field codepage specifies an EBCDIC codepage in LE ICONV format. It is used to handle XML entities and special characters.

On return, the fields errorCode, parserErrorCode and errorLineNumber are updated with return code information.

The errorCode field contains error codes for the certain function:

- NO_ERROR 0 no error
- UNKNOWN_COMMAND 1 no valid command found
- NO_COMMAREA 2 pointer to COMMAREA is NULL
- PARSER_ERROR 3 parser is in error state
- PARAM_ERROR 4 there is an error in parameters
- OUT_OF_MEMORY 98 no more memory available
- INTERNAL_ERROR 99 internal error

In case errorCode is set to PARSER_ERROR, field parserErrorCode contains one of the following return codes, and field errorLineNumber contains the line number that causes the error:

- PRSR_NO_ERROR 0 no error
- PRSR_NOT_WELL_FORMED 1 XML is not well formed
- PRSR_MISMATCHED_TAG 2 mismatching tag found
- PRSR_OUT_OF_MEMORY 3 no more memory available

In case errorCode is set to NO_ERROR (zero) and lastCall has been set to 1, the field xmlTreeRoot is set by the parser to point to the XML tree representation, starting with a tree root block (struct _troot).

TREE2BUFFER:

```
_Packed struct _t2bcommarea
{
 unsigned int     command;          // (in) TREE2BUFFER function code (2)
 void *           userBuffer;       // (in) buffer allocated by user
 unsigned int     userBufferLen;    // (in) length of buffer
 void *           xmlTreeRoot;      // (in) pointer to XML tree root
 unsigned int     userBufferEnd;    // (out) the length of filled buffer
 unsigned int     errorCode;        // (out) error code if any
 unsigned int     lastCall;         // (out) 1 if finished
 void *           saveArea;         // (in/out) save area, used internally
 char             codepage[12];     // (in) EBCDIC codepage (optional)
};
```


The TREE2BUFFER function can be called multiple times if the resulting XML document does not fit completely into a buffer or the application wants to process the XML data in smaller junks. The XML Generator saves the XML creation state between the calls and therefore can continue to generate a XML document with each call. Field saveArea is set by the generator to point to an internally used save area. It has to be passed to further calls when creating XML data in smaller junks. For the first call to the parser, field saveArea can be set to 0.

The XML tree representation that is to be used to create a XML data stream is supplied in field xmlTreeRoot. The tree must start with a tree root block (struct _troot). It can be allocated by the calling application, or it could have been created by the BUFFER2TREE function.

The resulting XML data stream is created into a user supplied buffer. The buffer is specified in field userBuffer and its length is specified in field userBufferLen. The XML generator fills this buffer up to the specified length. On return field userBufferEnd specifies the number of bytes used. In case the resulting XML data stream does not fit into the supplied buffer, field lastCall is set to 0 by the generator. The caller has to allocate a second buffer or process the buffer to free it, and call the XML Generator again. The generator will then continue to generate XML data and fill the buffer. The generation is completed if field lastCall is set to 1 on return.

The field codepage specifies an EBCDIC codepage in LE ICONV format. It is used to handle XML entities and special characters.

On return the field errorCode is updates with return code information.

• NO_ERROR	0	no error
• UNKNOWN_COMMAND	1	no valid command found
• NO_COMMAREA	2	pointer to COMMAREA is NULL
• PARSER_ERROR	3	parser is in error state
• PARAM_ERROR	4	there is an error in parameters
• OUT_OF_MEMORY	98	no more memory available
• INTERNAL_ERROR	99	internal error

FREETREE:

```
_Packed struct _ftcommarea
{
 unsigned int     command;          // (in) FREETREE function code (3)
 void *           xmlTreeRoot;      // (in) the xml tree to free
 char             errorCode;        // (out) error code
};
```

The FREETREE function must be used to free (FREEVIS) a tree that was allocated by the BUFFER2TREE function. Since the BUFFER2REE function allocates the storage needed for the tree representation in larger junks, only the FREETREE function knows how to free the storage.

Field xmlTreeRoot specifies the tree to free. The FREETREE function does not rely on the linkage between the XML tree blocks. This means the application can modify the tree by adding nodes or modifying the linkage between the nodes. However the FREETREE function

only frees these parts of the tree that has been allocated by the BUFFER2TREE function. If an application allocates additional nodes, the application must free them separately.

On return the field errorCode is updates with return code information.

• NO_ERROR	0	no error
• UNKNOWN_COMMAND	1	no valid command found
• NO_COMMAREA	2	pointer to COMMAREA is NULL
• PARAM_ERROR	4	there is an error in parameters
• OUT_OF_MEMORY	98	no more memory available
• INTERNAL_ERROR	99	internal error

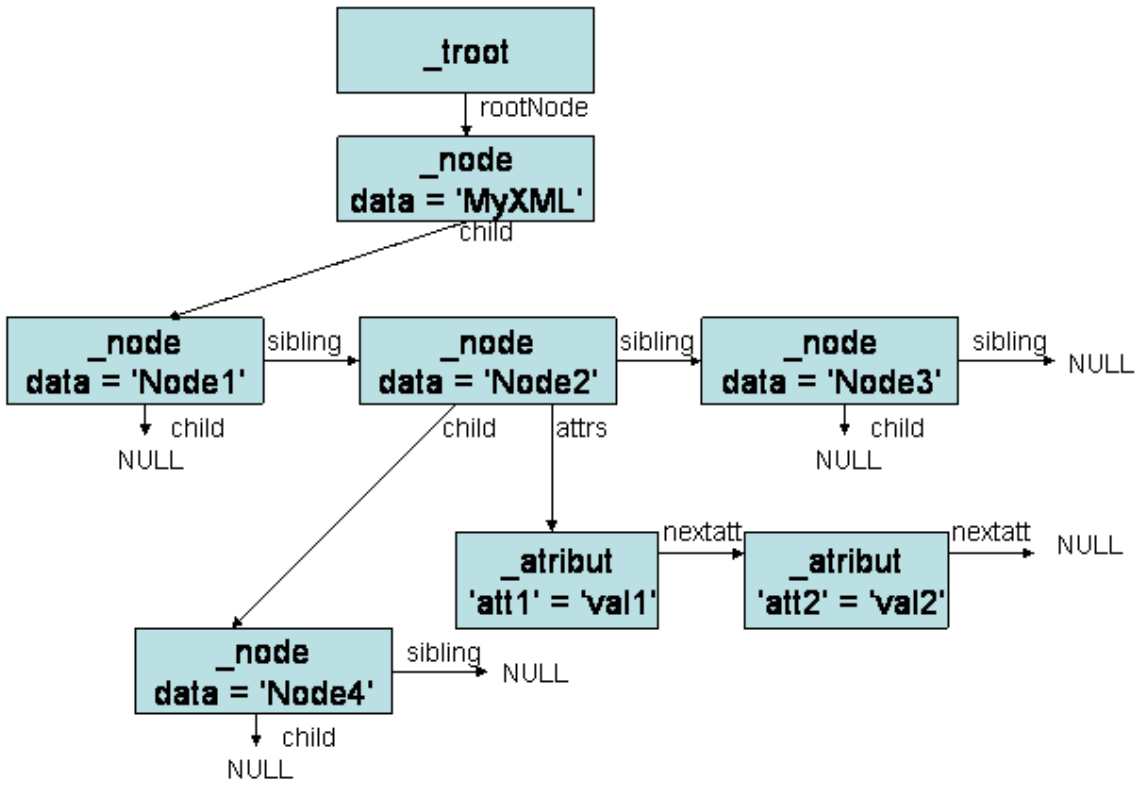
Structures used to build a XML tree in memory

The XML tree representation is built in memory using the following control blocks. All strings used in the tree are null terminated (C-string), but the length belonging to it specifies the number of bytes excluding the terminating NULL.

```
_Packed struct _troot               // tree root
{
 char *           doctype;          // the DOCTYPE of XML document
 unsigned int     doctypelen;       // length of DOCTYPE
 struct _node *   rootnode;         // points to the root node
 struct _page *   pageroot;         // internal area, do not modify
};

_Packed struct _node                // a XML node
{
 struct _node *   parent;           // the parent node of this node
 struct _node *   sibling;          // the sibling of this node
 struct _node *   child;            // the child of this node
 struct _attribut *atts;            // for START tags: the attributes
 char *           data;             // the tag name
 unsigned int     datalen;          // tag name length
 unsigned int     tagtype;          // type of tag
};

// node types used in struct _node
#define N_START     1               // start node
#define N_PI        2               // processing instruction
#define N_DATA      3               // DATA
#define N_CDATA     4               // CDATA
#define N_COMMENT   5               // a comment
#define N_ROOTNODE  6               // root node

_Packed struct _attribut            // a XML attribute
{
 char *           attname;          // attribute name
 unsigned int     anlen;            // length of attribute name
 char *           attvalue;         // attribute value
 unsigned int     avlen;            // length of attribute value
 void *           nextatt;          // next attribute or NULL
};
```


Sample how an XML tree is build in memory:

```
<MyXML>
  <Node1 />
  <Node2 att1=”val1” att2=”val2”>
    <Node4></Node4>
  </Node2>
  <Node3 />
</MyXML>
```



Sample how to parse an XML document in pseudo code:

Assume you have an XML document in a memory area named “buffer”. The length of the XML document is given by a variable called “bufferlen”:

```
/* Parse the XML document */
command           = 1    (= BUFFER2TREE)
xmlBuffer         = Addr(buffer)
xmlBufferLen      = bufferlen
xmlTreeRoot       = NULL
errorCode         = 0
parserErrorCode   = 0
errorLineNumber   = 0
lastCall          = 1
options           = ‘13’X (SKIP_WSDATA + SKIP_PI + SKIP_COMMENT)
pageSize          = 0  (use default page size)
saveArea          = NULL
codepage          = ‘IBM-1047’

Call ‘IESXMLAP’ (or ‘IESXMLCA’ in CICS)

If errorCode <> 0 then
  Print ‘errorCode         = ‘ + errorCode
  Print ‘parserErrorCode   = ‘ + parserErrorCode
  Print ‘errorLineNumber   = ‘ + errorLineNumber
End
Else
  /* Look at the XML nodes in the tree */
  TreeRoot     = xmlTreeRoot
  RootNode     = TreeRoot-> rootnode
  FirstChild   = RootNode->child
  SecondChild  = FirstChild->sibling

  Print ‘Root      = ‘ + RootNode->data(1:RootNode->datalen)
  Print ‘1st Child  = ‘ + FirstChild->data(1: FirstChild->datalen)
  Print ‘2nd Child  = ‘ + SecondChild->data(1: SecondChild->datalen)
End

/* Free the tree */
Command        = 3 (FREETREE)
xmlTreeRoot    = xmlTreeRoot  (from BUFFER2TREE commarea)
errorCode      = 0

Call ‘IESXMLAP’ (or ‘IESXMLCA’ in CICS)
```


SAX like interface

To use the SAX like interface the IBM provided object IESXMLPR has to be linked to the calling application. The IESXMLPR object provides entry points for the following functions. Please see C header file IESXMLPH.h in PRD1.BASE for details.

XML_ParserCreate	create a parser instance
XML_ParserFree	frees a parser instance
XML_SetElementHandler	sets start- and end-tag callback functions
XML_SetCharacterDataHandler	sets the character data callback function
XML_SetCommentHandler	sets the comment callback function
XML_SetDoctypeDeclHandler	sets the DOCTYPE declaration callback function
XML_SetCDataHandler	sets the CDATA callback function
XML_SetProcessingInstructionHandler	sets the processing instruction callback function
XML_Parse	parses a peace of XML data
XML_SetUserData	sets user data provided in each callback call
XML_ErrorString	gets an error description
XML_GetErrorCode	gets the error code
XML_GetCurrentLineNumber	gets the line number of an error

The SAX like parser calls user implemented callback functions for each kind of tag found in the XML data. The callback function may then process the tag or data.

XML_StartElementHandler	called when a start of a XML tag is found
XML_EndElementHandler	called when a end of a XML tag is found
XML_CharacterDataHandler	called when character data is found in a tag
XML_CDataHandler	called when CDATA is found in a tag
XML_CommentHandler	called when a comment is found
XML_DoctypeDeclHandler	called when a DOCTYPE declaration is found
XML_ProcessingInstructionHandler	called when a processing instructions is found

NOTE: Since APAR PQ97187 the callback function pointers provided to the XML_SetxxxHandler must be pointers returned by the C/LE function fetchep. This must be done because the pointers are provided to the IESXMLCT phase.

Appendix A – Parameter area definitions in Assembler

```
B2TCAREA DSECT          BUFFER TO TREE COMMAREA
B2TCMD   DS   F         (IN) BUFFER2TREE FUNCTION CODE
XMLB2T   EQU  1         BUFFER TO TREE FUNCTION
B2TBUF   DS   A         (IN) BUFFER WITH XML DATA
B2TBUFLN DS   F         (IN) BUFFER LENGTH
B2TTREER DS   A         (OUT) RESULT TREE
B2TERRCD DS   F         (OUT) ERROR CODE IF ANY
B2TPERR  DS   F         (OUT) ERROR CODE OF PARSER IF ANY
B2TELINE DS   F         (OUT) LINE WITCH HAS AN ERROR
B2TLASTC DS   F         (IN) 1 IF LAST PART OF XML DATA
B2TOPTS  DS   F         (IN) PARSER OPTIONS
SKPWSDTA EQU  X'01'     SKIP DATA WICH CONTAIN WHITESPACES ONLY
SKPPI    EQU  X'02'     SKIP PROCESSING INSTRUCTIONS
SKPDATA  EQU  X'04'     SKIP ALL DATA
SKPCDATA EQU  X'08'     SKIP ALL CDATA ELEMENTS
SKPCOMNT EQU  X'10'     SKIP COMMENTS
B2TPSIZE DS   F        (IN) PAGE SIZE OF PAGES ALLOCATED
B2TSAVEA DS   A        (IN/OUT) SAVE AREA, USED INTERNALLY
B2TCODEP DS   CL12     (IN) EBCDIC CODEPAGE

T2BCAREA DSECT          TREE TO BUFFER COMMAREA
T2BCMD   DS   F         (IN) TREE2BUFFER FUNCTION CODE
XMLT2B   EQU  2         TREE TO BUFFER FUNCTION
T2BBUF   DS   A         (IN) BUFFER ALLOCATED BY USER
T2BBUFLN DS   F         (IN) BUFFER LENGTH
T2BTREER DS   A         (IN) POINTER TO XML TREE ROOT
T2BBEND  DS   F         (OUT) THE LENGTH OF THE FILLED BUFFER
T2BERRCD DS   F         (OUT) ERROR CODE IF ANY
T2BLASTC DS   F         (OUT) 1 IF FINISHED
T2BSAVEA DS   A         (IN/OUT) SAVE AREA, USED INTERNALLY
T2BCODEP DS   CL12     (IN) EBCDIC CODEPAGE

FTRCAREA DSECT          FREE TREE COMMAREA
FTRCMD   DS   F         (IN) FREETREE FUNCTION CODE
XMLFTR   EQU  3         FREE TREE FUNCTION
FTRTREER DS   A         (IN) POINTER TO XML TREE ROOT
FTRERRCD DS   X         (OUT) ERROR CODE IF ANY

* ERROR CODES
XMLEOK    EQU  0         NO ERROR
XMLECMD   EQU  1         NO VALID COMMAND FOUND IN COMMAREA
XMLECA    EQU  2         POINTER TO COMMAREA IS NULL
XMLEPARSE EQU  3         PARSER IS IN ERROR STATE
XMLEPARM  EQU  4         THERE IS AN ERROR IN PARAMS
XMLEMEM   EQU  98        NO MORE MEMORY AVAILABLE
XMLEINTER EQU  99        INTERNAL ERROR

* PARSER ERROR CODES
PRSOK     EQU  0         NO ERROR
PRSNWELLF EQU  1         XML IS NOT WELL FORMED
PRSMSMTAG EQU  2         MISMATCHING TAG FOUND
PRSMEMORY EQU  3         OUT OF MEMORY
```



```
* TREE ROOT BLOCK
TROOT     DSECT          TREE ROOT DSECT
DOCTYPE   DS   A         THE DOCTYPE OF XML DOCUMENT
DOCTYPELN DS   F         LENGTH OF DOCTYPE
ROOTNODE  DS   A         POINTS TO THE ROOT NODE
PAGEROOT  DS   A         INTERNAL AREA, DO NOT MODIFY

* TREE NODE BLOCK
XMLNODE   DSECT          A XML NODE BLOCK
PARENT    DS   A         PTR TO THE PARENT NODE OF THIS NODE
SIBLING   DS   A         PTR TO THE SIBLING OF THIS NODE
CHILD     DS   A         PTR TO THE CHILD OF THIS NODE
ATTRS     DS   A         START TAGS: PTR TO THE ATTRIBUTES
DATA      DS   A         THE TAG NAME
DATALEN   DS   F         TAG NAME LENGTH
TAGTYPE   DS   F         TYPE OF TAG
N_START   EQU  1         START NODE
N_PI      EQU  2         PROCESSING INSTRUCTION
N_DATA    EQU  3         DATA
N_CDATA   EQU  4        CDATA
N_COMMENT EQU  5         A COMMENT
N_ROOTNDE EQU  6         ROOT NODE

* ATTRIBUTE BLOCK
ATTRIBUT  DSECT          A XML ATTRIBUTE BLOCK
ATTNAME   DS   A         ATTRIBUTE NAME
ANLEN     DS   F         LENGTH OF ATTRIBUTE NAME
ATTVALUE  DS   A         ATTRIBUTE VALUE
AVLEN     DS   F         LENGTH OF ATTRIBUTE VALUE
NEXTATT   DS   A         NEXT ATTRIBUTE OR NULL
```


Appendix B – Parameter area definitions in PL/I

```
DCL 1 B2TCOMMAREA BASED,
      2 COMMAND          FIXED BIN(31), /* (IN) FUNCTION CODE (1) */
      2 XMLBUFFER        POINTER,       /* (IN) BUFFER WITH XML DATA */
      2 XMLBUFFERLEN     FIXED BIN(31), /* (IN) BUFFER LENGTH */
      2 XMLTREEROOT      POINTER,       /* (OUT) RESULT TREE */
      2 ERRORCODE        FIXED BIN(31), /* (OUT) ERROR CODE IF ANY */
      2 PARSERERRORCODE FIXED BIN(31), /* (OUT) ERROR CODE OF PARSER */
      2 ERRORLINENUMBER FIXED BIN(31), /* (OUT) LINE WITCH HAS AN ERROR */
      2 LASTCALL         FIXED BIN(31), /* (IN) 1 = LAST PART OF XML DATA */
      2 OPTIONS          FIXED BIN(31), /* (IN) PARSER OPTIONS */
      2 PAGESIZE         FIXED BIN(31), /* (IN) SIZE OF PAGES ALLOCATED */
      2 SAVEAREA         POINTER,       /* (IN/OUT) SAVEAREA */
      2 CODEPAGE         CHAR(12);      /* (IN) EBCDIC CODEPAGE */

DCL 1 T2BCOMMAREA BASED,
      2 COMMAND          FIXED BIN(31), /* (IN) FUNCTION CODE (2) */
      2 USERBUFFER       POINTER,       /* (IN) BUFFER ALLOCATED BY USER */
      2 USERBUFFERLEN    FIXED BIN(31), /* (IN) LENGTH OF BUFFER */
      2 XMLTREEROOT      POINTER,       /* (IN) POINTER TO XML TREE ROOT */
      2 USERBUFFEREND    FIXED BIN(31), /* (OUT) LENGTH OF FILLED BUFFER */
      2 ERRORCODE        FIXED BIN(31), /* (OUT) ERROR CODE IF ANY */
      2 LASTCALL         FIXED BIN(31), /* (OUT) 1 IF FINISHED */
      2 SAVEAREA         POINTER,       /* (IN/OUT) SAVE AREA */
      2 CODEPAGE         CHAR(12);      /* (IN) EBCDIC CODEPAGE */

DCL 1 FTCOMMAREA BASED,
      2 COMMAND          FIXED BIN(31), /* (IN) FUNCTION CODE (3) */
      2 XMLTREEROOT      POINTER,       /* (IN) THE XML TREE TO FREE */
      2 ERRORCODE        FIXED BIN(8);  /* (OUT) ERROR CODE */

/* VALUES FOR XML COMMAND */
DCL BUFFER2TREE          FIXED BIN(31) STATIC INIT(1);
DCL TREE2BUFFER          FIXED BIN(31) STATIC INIT(2);
DCL FREETREE             FIXED BIN(31) STATIC INIT(3);

/* RETURN CODES */
DCL XMLERR_OK            FIXED BIN(31) STATIC INIT(0);
DCL XMLERR_UNKNOWN_CMD   FIXED BIN(31) STATIC INIT(1);
DCL XMLERR_NO_COMMAREA   FIXED BIN(31) STATIC INIT(2);
DCL XMLERR_PARSER_ERR    FIXED BIN(31) STATIC INIT(3);
DCL XMLERR_PARAM_ERR     FIXED BIN(31) STATIC INIT(4);
DCL XMLERR_OUTOFMEMORY   FIXED BIN(31) STATIC INIT(98);
DCL XMLERR_INTERNAL      FIXED BIN(31) STATIC INIT(99);

/* PARSER RETURN CODES */
DCL PERR_OK              FIXED BIN(31) STATIC INIT(0);
DCL PERR_NOTWELLFORMED   FIXED BIN(31) STATIC INIT(1);
DCL PERR_MISMATCHEDTAG   FIXED BIN(31) STATIC INIT(2);
DCL PERR_OUTOFMEMORY     FIXED BIN(31) STATIC INIT(3);
```



```
/* TREE ROOT BLOCK */
DCL 1 TROOT BASED,
      2 DOCTYPE         POINTER,       /* THE DOCTYPE OF XML DOCUMENT */
      2 DOCTYPELEN      FIXED BIN(31), /* LENGTH OF DOCTYPE */
      2 ROOTNODE        POINTER,       /* POINTS TO THE ROOT NODE */
      2 PAGEROOT        POINTER;       /* INTERNAL AREA, DO NOT MODIFY */

/* TREE NODE */
DCL 1 NODE BASED,
      2 PARENT          POINTER,       /* THE PARENT NODE OF THIS NODE */
      2 SIBLING         POINTER,       /* THE SIBLING OF THIS NODE */
      2 CHILD           POINTER,       /* THE CHILD OF THIS NODE */
      2 ATTRS           POINTER,       /* FOR START TAGS: THE ATTRIBUTES */
      2 DATA            POINTER,       /* THE TAG NAME */
      2 DATALEN         FIXED BIN(31), /* TAG NAME LENGTH */
      2 TAGTYPE         FIXED BIN(31); /* TYPE OF TAG */

/* VALUES FOR NODE TYPE */
DCL N_START     FIXED BIN(31) STATIC INIT(1); /* START NODE */
DCL N_PI        FIXED BIN(31) STATIC INIT(2); /* PROCESSING INSTRUCTION */
DCL N_DATA      FIXED BIN(31) STATIC INIT(3); /* DATA */
DCL N_CDATA     FIXED BIN(31) STATIC INIT(4); /* CDATA */
DCL N_COMMENT   FIXED BIN(31) STATIC INIT(5); /* A COMMENT */
DCL N_ROOTNODE  FIXED BIN(31) STATIC INIT(6); /* ROOT NODE */

/* XML ATTRIBUTE BLOCK */
DCL 1 ATTRIBUT BASED,
      2 ATTNAME         POINTER,       /* ATTRIBUTE NAME */
      2 ANLEN           FIXED BIN(31), /* LENGTH OF ATTRIBUTE NAME */
      2 ATTVALUE        POINTER,       /* ATTRIBUTE VALUE */
      2 AVLEN           FIXED BIN(31), /* LENGTH OF ATTRIBUTE VALUE */
      2 NEXTATT         POINTER;       /* NEXT ATTRIBUTE OR NULL */
```


Appendix C – Parameter area definitions in COBOL

```
******************************
* BUFFER2TREE PARAMETER AREA *
******************************
 01   B2TCOMMAREA.
      02 COMMAND            PIC 9(9) BINARY.
      02 XML-BUFFER         USAGE IS POINTER.
      02 XML-BUFFER-LEN     PIC 9(9) BINARY.
      02 XML-TREE-ROOT      USAGE IS POINTER.
      02 ERROR-CODE         PIC 9(9) BINARY.
      02 PARSER-ERROR-CODE PIC 9(9) BINARY.
      02 ERROR-LINE-NUMBER PIC 9(9) BINARY.
      02 LAST-CALL          PIC 9(9) BINARY.
      02 OPTIONS            PIC 9(9) BINARY.
      02 PAGE-SIZE          PIC 9(9) BINARY.
      02 SAVE-AREA          USAGE IS POINTER.
      02 CODE-PAGE          PIC X(12).

******************************
* TREE2BUFFER PARAMETER AREA *
******************************
 01   T2BCOMMAREA.
      02 COMMAND            PIC 9(9) BINARY.
      02 USER-BUFFER        USAGE IS POINTER.
      02 USER-BUFFER-LEN    PIC 9(9) BINARY.
      02 XML-TREE-ROOT      USAGE IS POINTER.
      02 USER-BUFFER-END    PIC 9(9) BINARY.
      02 ERROR-CODE         PIC 9(9) BINARY.
      02 LAST-CALL          PIC 9(9) BINARY.
      02 SAVE-AREA          USAGE IS POINTER.
      02 CODE-PAGE          PIC X(12).

******************************
* FREETREE PARAMETER AREA    *
******************************
 01   FTCOMMAREA.
      02 COMMAND            PIC 9(9) BINARY.
      02 XML-TREE-ROOT      USAGE IS POINTER.
      02 ERROR-CODE         PIC 9(2) BINARY.


******************************
* VALUES FOR XML COMMAND     *
******************************
 01   BUFFER2TREE           PIC 9(9) BINARY VALUE 1.
 01   TREE2BUFFER           PIC 9(9) BINARY VALUE 2.
 01   FREETREE              PIC 9(9) BINARY VALUE 3.

******************************
* RETURN CODES               *
******************************
 01 XMLERR-OK               PIC 9(9) BINARY VALUE 0.
 01 XMLERR-UNKNOWN-CMD      PIC 9(9) BINARY VALUE 1.
 01 XMLERR-NO-COMMAREA      PIC 9(9) BINARY VALUE 2.
 01 XMLERR-PARSER-ERR       PIC 9(9) BINARY VALUE 3.
 01 XMLERR-PARAM-ERR        PIC 9(9) BINARY VALUE 4.
 01 XMLERR-OUT-OF-MEMORY    PIC 9(9) BINARY VALUE 98.
 01 XMLERR-INTERNAL         PIC 9(9) BINARY VALUE 99.

******************************
```



```
* PASER RETURN CODES          *
*******************************
 01 PERR-OK                 PIC 9(9) BINARY VALUE 0.
 01 PERR-NOT-WELL-FORMED    PIC 9(9) BINARY VALUE 1.
 01 PERR-MISMATCHED-TAG     PIC 9(9) BINARY VALUE 2.
 01 PERR-OUT-OF-MEMORY      PIC 9(9) BINARY VALUE 3.
```



```
*****************************
* TREE ROOT AREA             *
*****************************
 01   TROOT.
      02 DOCTYPE           USAGE IS POINTER.
      02 DOCTYPE-LEN       PIC 9(9) BINARY.
      02 ROOT-NODE         USAGE IS POINTER.
      02 PAGE-ROOT         USAGE IS POINTER.

*****************************
* TREE NODE AREA             *
*****************************
 01   NODE.
      02 PARENT            USAGE IS POINTER.
      02 SIBLING           USAGE IS POINTER.
      02 CHILD             USAGE IS POINTER.
      02 ATTRS             USAGE IS POINTER.
      02 DATA              USAGE IS POINTER.
      02 DATA-LEN          PIC 9(9) BINARY.
      02 TAG-TYPE          PIC 9(9) BINARY.

*****************************
* NODE TYPES                 *
*****************************
 01 N_START                PIC 9(9) BINARY VALUE 1.
 01 N_PI                   PIC 9(9) BINARY VALUE 2
 01 N_DATA                 PIC 9(9) BINARY VALUE 3
 01 N_CDATA                PIC 9(9) BINARY VALUE 4
 01 N_COMMENT              PIC 9(9) BINARY VALUE 5
 01 N_ROOTNODE             PIC 9(9) BINARY VALUE 6

*****************************
* ATTRIBUTE AREA             *
*****************************
 01   ATTRIBUT.
      02 ATT-NAME          USAGE IS POINTER.
      02 ATT-NAME-LEN      PIC 9(9) BINARY.
      02 ATT-VALUE         USAGE IS POINTER.
      02 ATT-VALUE-LEN     PIC 9(9) BINARY.
      02 NEXT-ATT          USAGE IS POINTER.
```


Remarks

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

CICS IBM Language Environment VSE/ESA

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names, may be the trademarks or service marks of others.

Comments and Questions

Comments or questions on this documentation are welcome. Please send your comments to:

zvse@de.ibm.com

