RexxExpat Function Reference

by Dominik Stein

The RexxExpat library wraps the C-XML-parser 'expat', version 1.2, so it can be used from a Rexx environment. Since the wrapper functions mostly simply pass through the values from Expat to Rexx and vice versa, please have also a look at the following resources for further readings:

For documentation on the interface of 'expat' please refer to

- [Expat:] xmlparse.h, version 1.2, Thai Open Source Software Center
- [Cooper:] Clark Cooper, Using Expat, Sept. 1, 1999, XML.com, O'Reilly & Associates (http://www.xml.com/pub/1999/09/expat/reference.html)
- (to get the latest information and the newest versions please visit SourceForge, Expat XML Parser Project Info at http://sourceforge.net/projects/expat/)

For documentation on the Rexx SAA API please refer to

- [RexxSaa:] rexxsaa.h, version 1.5, Anders Christensen, The Regina Rexx Interpreter
- [Regina:] Anders Christensen, The Regina Rexx Interpreter, August 14, 2000, (**ftp://ftp.lightlink.com/pub/hessling/Regina/reginapdf22.zip**)
- [IBMPG:] IBM, Object REXX for Windows NT and Windows95 Programming Guide (Version 1.03), third edition, May 1999 (ftp://service.boulder.ibm.com/ps/products/ad/obj-xx/rexxpg.zip)
- (for the ongoing development of Rexx please visit the IBM Rexx Family pages at http://www-4.ibm.com/software/ad/rexx/ and http://www-4.ibm.com/software/ad/obj-rexx/ as well as Mark Hessling's Rexx Homepage at http://www.lightlink.com/hessling/ - to get the latest information about and the newest versions of Regina Rexx)

For matters specific to RexxExpat please read this document.

I tried to keep the headers of my RexxExpat functions as close as possible to the headers of the corresponding 'expat' functions. So for documentation purposes of my RexxExpat I'd like to simply adapt the article "Using Expat" by Clark Cooper (published on XML.com, Sept. 1, 1999), who did an excellent job in describing and explaining the function and functions of the Expat parser. Thank you very much, Clark, your article helped me a lot in understanding 'expat'!!

I used Regina Rexx for developing! Please use an Microsoft

Index

Parser Creation

[RexxExpat:] When creating a new parser, RexxExpat stores the information specific to this newly created parser in a special distinct data structure. This is in particular handler information, encoding information, the variable pool, and last not least the reference to the corresponding parser. In fact, the value returned by the parser creation functions is a reference to this data structure, from which the reference to the corresponding parser is then retrieved. This is to make RexxExpat thread-safe.

ExpatParserCreate

`numeric ExpatParserCreate(alphanumeric encoding)`
[Cooper:] Constructs a new parser. If **`encoding`** is non-null and not of zero length, it specifies a character encoding to use for the document. This overrides the document encoding declaration. There are four built-in encodings:

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

Any other value will invoke a call to the **`UnknownEncodingHandler`**.
[RexxExpat:] Returns 0 if out of memory. Otherwise returns a reference to the new parser.

ExpatParserCreateNS

`numeric ExpatParserCreateNS(alphanumeric encoding, alphanumeric sep)`
[Cooper:] Constructs a new parser that has namespace processing in effect. Namespace expanded element names and attribute names are returned as a concatenation of the namespace URI, **`sep`**, and the local part of the name. This means that you should pick a character for **`sep`** that can't be part of a legal URI.
[Expat:] While unprefixed attribute names are never expanded, unprefixed element type names are expanded only if there is a default namespace. When a namespace is not declared, the name and prefix will be passed through without expansion.
[RexxExpat:] Returns 0 if out of memory. Otherwise returns a reference to the new parser.

[RexxExpat:] In RexxExpat you can use both **`ExpatParserCreate`** and **`ExpatParserCreateNS`** to create either kind of parser. Which function is finally called depends on the number or arguments passed.

ExpatExternalEntityParserCreate

`numeric ExpatExternalEntityParserCreate(numeric parentparser, alphanumeric context, alphanumeric encoding)`
[Cooper:] Constructs a new parser for parsing an external general entity. Context is the **`context`** argument passed in a call to a ExternalEntityRefHandler. Other state information such as handlers, user data, namespace processing is inherited from the parser passed as the 1st argument, [RexxExpat:] which is the parser that encountered the entity reference and called (and is passed to) the ExternalEntityRefHandler. [Cooper:] So you shouldn't need to call any of the behaviour changing functions on this parser (unless you want it to act differently than the parent parser.)
[Expat:] The **`context`** string consists of a sequence of tokens separated by formfeeds; a token consisting of a name specifies that the general entity of the name is open; a token of the form prefix=uri specifies the namespace for a particular prefix; a token of the form =uri specifies the default namespace.

This can be called at any point after the first call to an ExternalEntityRefHandler so longer as the parser has not yet been freed. The new parser is completely independent and may safely be used in a separate thread.
[RexxExpat:] With the new parser a new data structure to store parser specific information is created as well. Handlers are initialized from the parser argument (so is NOT encoding information). The newly created parser uses the same variable pool as its parent parser.
Returns 0 if out of memory. Otherwise returns a reference to the new parser.

ExpatParserFree

```
none ExpatParserFree(numeric parser)
```

[Cooper:] Free memory used by the parser [RexxExpat:] 'expat' and by the wrapper 'RexxExpat'.

Parsing

[RexxExpat:] Parsing is done by RexxExpat by feeding a buffer provided by 'expat's **XML_GetBuffer** function and then calling 'expat's **XML_ParseBuffer** function. Not providing an own buffer but using 'expat's buffer turned out to be crucial when parse position functions were called (e.g. for error reporting). Since these functions are trying to access the buffer, it must remain in memory even as **`ExpatParse`** returns. So, **`ExpatParse`** should not de-allocate the buffer.
(Remark: Unfortunately it happens to be, that even though I let 'expat' take care of the buffer, parse position functions are sometimes still causing memory access violations… preferingly after fatal parsing errors are detected.)

RexxExpat itself does not wrap 'expat's **XML_ParseBuffer** and **XML_GetBuffer** functions, since **`ExpatParse`** will perfectly do ;-).

ExpatParse

```
numeric ExpatParse(numeric parser, alphanumeric buffer, numeric more2come)
```

[Cooper:] Parse some more of the document. The string **`buffer`** is a buffer containing part (or perhaps all) of the document. [RexxExpat:] In RexxExpat the third argument is used contrary to its original meaning in 'expat'. The **`more2come`** parameter informs the parser that there is still something left to be parsed. Pass 0, NULL, a zero length argument or no (third) argument to inform the parser that this is the last piece of the document. [Cooper:] Frequently, the last piece is empty (i.e. **`len`** is zero).
If a parse error occurred, it returns 0. Otherwise it returns a non-zero value.

Now, I'd like to adapt Clark Cooper's example (for use of **XML_ParseBuffer** and **XML_GetBuffer**) in his article 'Using Expat' to an Rexx environment. Note, that the current file offset is cached before calling **`ExpatParse`** and restored afterwards. This is necessary because (at least Regina) Rexx seems to forget the current offset when invoking a callback function (and thus instanciating a new Rexx interpreter).

```
/* parse file */
myfile = "filename.xml"
mypos  = 0

do while lines(myfile)<>0
   /* read next line from file */
   nextline = linein(myfile)

   /* remember file offset */
   mypos = STREAM(myfile, 'Command', 'Query Position Read');

   if ExpatParse(Parser, nextline, lines(myfile)) == 0 then
      /* handle parse error */
```

```
   /* restore file offset */
   call STREAM(myfile, 'Command', 'Position ='mypos Read);
end
```

Handler Setting

[RexxExpat:] To give maximum flexibility with using RexxExpat, there are three parameters to be set for each handler (**ProcName, EnvName, Instore**). The crucial parameter of these three is the last one (**Instore**). Depending on its value, the value in **ProcName** is considered to be either pure REXX code, a file name, or a macro. Since the parameters are directly passed to the RexxStart function, it's a good idea to refer to the IBM Programming Guide, "Calling the REXX Interpreter", p.102ff, to read about particular constraints of each parameter in either case. Nevertheless, I will try to quote the most important points here.

If **Instore** is left out, NULL, 0, or of zero length, **ProcName** is considered to be a file name of the REXX procedure. [IBMPG: p.102] You can also provide an extension, drive, and path. If you do not specify a file extension, the default is .CMD. If you do not specify the path and drive, the REXX interpreter uses the usual file search (current directory, then environment path).

If **Instore** contains "macro" or "MACRO", **ProcName** is considered to be a macro, which must already be loaded into the macrospace (for further information on the macrospace interface see IBM Programming Guide, "Macrospace Interface", p.150). [IBMPG: p. 150] Programs registered in the REXX macrospace are available to all processes.

If **Instore** is set to anything else than the values mentioned above, **ProcName** must contain the REXX procedure source. [IBMPG: p.105] The source must be an exact image of a REXX procedure disk file, complete with carriage returns, line feeds, and end-of-file characters.

EnvName is the initial ADDRESS environment name. If **EnvName** is null, the file extension is used as the initial ADDRESS environment. The environment name cannot be longer than 250 characters.

[Cooper:] Although handlers are typically set prior to parsing and left alone, an application may choose to set or change the handler for a parsing event while the parse is in progress. For instance, your application may choose to ignore all text not descended from a **para** element. One way it could do this is to set the character handler when a **para** start tag is seen, and unset it for the corresponding end tag.

[RexxExpat:] In RexxExpat a handler may be *unset* either by *not* providing a **ProcName** (leave it out) or by providing a **ProcName** which is NULL or of zero length to the appropriate handler setter. Remember to explicitly indicate left out parameters by kommata, if you intend to pass subsequent parameters.
If the handler setting functions return a value other than 0, an error occurred during memory allocation or de-allocation.

Please note the following remarks on character encoding – they are also valid for RexxExpat, since RexxExpat only passes through the returned characters without any alteration of character encoding.

The 'expat' parser returns [Cooper:] strings in arrays of type **ExpatChar**. This type is defined in xmlparse.h and is conditional upon the setting of either of the ExpatUNICODE macros. If neither of these is set, then **ExpatChar** contains characters encoding UTF-8. Otherwise you'll be receiving UTF-16 in the form of either **unsigned short** or **wchar_t** characters. […] You'll receive them in this form independent of the original encoding of the document […]. So character encoding has to be set on compilation time.

ExpatSetElementHandler

```
ExpatSetElementHandler(numeric parser,
                       alphanumeric StartElementHandler_ProcName,
                       alphanumeric StartElementHandler_EnvName,
                       alphanumeric StartElementHandler_Instore,
                       alphanumeric EndElementHandler_ProcName,
                       alphanumeric EndElementHandler_EnvName,
                       alphanumeric EndElementHandler_Instore);
```

Arguments passed to the StartElementHandler:

```
PARSE ARG parser, name, attribute1, value1, attribute2, value2, …, …, attributeN,
valueN
```

Arguments passed to the EndElementHandler:

```
PARSE ARG parser, name
```

[Cooper:] Set handlers for start and end tags. [RexxExpat:] The name of the element specified in the start tag is passed as second argument. [Cooper:] Each attribute seen in a start (or empty) tag occupies two consecutive places in the argument list: the attribute name followed by the attribute value.
[RexxExpat:] If either handler returns a valid value (not NULL and not of zero length), RexxExpat calls 'expat's **XML_DefaultCurrent** function an passes the corresponding markup to the default handler.

ExpatSetCharacterDataHandler

```
ExpatSetCharacterDataHandler(numeric parser,
                             alphanumeric CharacterDataHandler_ProcName,
                             alphanumeric CharacterDataHandler_EnvName,
                             alphanumeric CharacterDataHandler_Instore);
```

Arguments passed to the CharacterDataHandler:

```
PARSE ARG parser, text
```

[Cooper:] Set a text handler. A single block of contiguous text free of markup may still result in a sequence of calls to this handler. In other words, if you're searching for a pattern in the text, it may be split across calls to this handler.
[RexxExpat:] If the handler returns a valid value (not NULL and not of zero length), RexxExpat calls 'expat's **XML_DefaultCurrent** function an passes the corresponding markup to the default handler.

ExpatSetProcessingInstructionHandler

```
ExpatSetProcessingInstructionHandler(numeric parser,
                                     alphanumeric
ProcessingInstructionHandler_ProcName,
                                     alphanumeric
ProcessingInstructionHandler_EnvName,
```

```
                                        alphanumeric
ProcessingInstructionHandler_Instore);
```

Arguments passed to the ProcessingInstructionHandler:

```
PARSE ARG parser, target, data
```

[Cooper:] Set a handler for processing instructions. The target is the first word in the processing instruction. The data is the rest of the characters in it after skipping all whitespace after the initial word. [RexxExpat:] If the handler returns a valid value (not NULL and not of zero length), RexxExpat calls 'expat's **XML_DefaultCurrent** function an passes the corresponding markup to the default handler.

ExpatSetCommentHandler

```
ExpatSetCommentHandler(numeric parser,
                       alphanumeric CommentHandler_ProcName,
                       alphanumeric CommentHandler_EnvName,
                       alphanumeric CommentHandler_Instore);
```

Arguments passed to the CommentHandler:

```
PARSE ARG parser, data
```

[Cooper:] Set a handler for comments. The data is all text inside the comment delimiters.

ExpatSetCdataSectionHandler

```
ExpatSetCdataSectionHandler(numeric parser,
                            alphanumeric StartCdataSectionHandler_ProcName,
                            alphanumeric StartCdataSectionHandler_EnvName,
                            alphanumeric StartCdataSectionHandler_Instore,
                            alphanumeric EndCdataSectionHandler_ProcName,
                            alphanumeric EndCdataSectionHandler_EnvName,
                            alphanumeric EndCdataSectionHandler_Instore);
```

Arguments passed to the StartCdataSectionHandler:

```
PARSE ARG parser
```

Arguments passed to the EndCdataSectionHandler:

```
PARSE ARG parser
```

[Cooper:] Sets handlers that get called at the beginning and end of a CDATA section.

ExpatSetDefaultHandler

```
ExpatSetDefaultHandler(numeric parser,
                       alphanumeric DefaultHandler_ProcName,
```

```
                        alphanumeric DefaultHandler_EnvName,
                        alphanumeric DefaultHandler_Instore);
```

Arguments passed to the DefaultHandler:

```
PARSE ARG parser, data
```

[Cooper:] Sets a handler for any characters in the document which wouldn't otherwise be handled. This includes both data for which no handlers can be set (like some kinds of DTD declarations) and data which could be reported but which currently has no handler set. Note that a contiguous piece of data that is destined to be reported to the default handler may actually be reported over several calls to the handler. Setting the handler with this call has the side effect of turning off expansion of references to internally defined general entities. Instead these references are passed to the default handler. [Expat:] The characters are passed exactly as they were in the XML document except that they will be encoded in UTF-8 (RexxExpat does not change encodings). Line boundaries are not normalized. Note that a byte order mark character is not passed to the default handler.

ExpatSetDefaultHandlerExpand

```
ExpatSetDefaultHandlerExpand(numeric parser,
                             alphanumeric DefaultHandlerExpand_ProcName,
                             alphanumeric DefaultHandlerExpand_EnvName,
                             alphanumeric DefaultHandlerExpand_Instore);
```

Arguments passed to the DefaultHandlerExpand:

```
PARSE ARG parser, data
```

[Expat:] This sets the default handler but does not inhibit expansion of internal entities. The entity reference will not be passed to the default handler.

ExpatSetDoctypeDeclHandler

```
ExpatSetDoctypeDeclHandler(numeric parser,
                           alphanumeric StartDoctypeDeclHandler_ProcName,
                           alphanumeric StartDoctypeDeclHandler_EnvName,
                           alphanumeric StartDoctypeDeclHandler_Instore,
                           alphanumeric EndDoctypeDeclHandler_ProcName,
                           alphanumeric EndDoctypeDeclHandler_EnvName,
                           alphanumeric EndDoctypeDeclHandler_Instore);
```

Arguments passed to the StartDoctypeDeclHandler:

```
PARSE ARG parser, doctypeName
```

Arguments passed to the EndDoctypeDeclHandler:

```
PARSE ARG parser
```

[RexxExpat:] Sets handlers that get called at the beginning and end of a DOCTYPE declaration. The EndDoctypeDeclHandler is called after any external subsets are processed.

ExpatSetUnparsedEntityDeclHandler

```
ExpatSetUnparsedEntityDeclHandler(numeric parser,
                                  alphanumeric UnparsedEntityDeclHandler_ProcName,
                                  alphanumeric UnparsedEntityDeclHandler_EnvName,
                                  alphanumeric UnparsedEntityDeclHandler_Instore);
```

Arguments passed to the UnparsedEntityDeclHandler:

```
PARSE ARG parser, entityName, base, systemId, publicId, notationName
```

[Cooper:] Set a handler that receives declarations of unparsed entities. These are entity declarations that have a notation (NDATA) field:

```
<!ENTITY logo SYSTEM "images/logo.gif" NDATA gif>
```

[Cooper:] So for this example, the entityName would be "logo", the systemId would be "images/logo.gif" and notationName would be "gif". For this example the publicId parameter is null. The base parameter would be whatever has been set with **ExpatSetBase**. If not set, it would be null.

ExpatSetNotationDeclHandler

```
ExpatSetNotationDeclHandler(numeric parser,
                            alphanumeric NotationDeclHandler_ProcName,
                            alphanumeric NotationDeclHandler_EnvName,
                            alphanumeric NotationDeclHandler_Instore);
```

Arguments passed to the NotationDeclHandler:

```
PARSE ARG parser, notationName, base, systemId, publicId
```

[Cooper:] Set a handler that receives notation declarations. [Expat:] The base argument is whatever was set by **ExpatSetBase**. The notationName will never be null. The other arguments can be.

ExpatSetExternalParsedEntityDeclHandler

```
ExpatSetExternalParsedEntityDeclHandler(numeric parser,
                                        alphanumeric
ExternalParsedEntityDeclHandler_ProcName,
                                        alphanumeric
ExternalParsedEntityDeclHandler_EnvName,
                                        alphanumeric
ExternalParsedEntityDeclHandler_Instore);
```

Arguments passed to the ExternalParsedEntityDeclHandler:

```
PARSE ARG parser, entityName, base, systemId, publicId
```

[RexxExpat:] No description supplied by 'expat'. :-(

ExpatSetInternalParsedEntityDeclHandler

```
ExpatSetInternalParsedEntityDeclHandler(numeric parser,
                                        alphanumeric
InternalParsedEntityDeclHandler_ProcName,
                                        alphanumeric
InternalParsedEntityDeclHandler_EnvName,
                                        alphanumeric
InternalParsedEntityDeclHandler_Instore);
```

Arguments passed to the InternalParsedEntityDeclHandler:

```
PARSE ARG parser, entityName, replacementText
```

[RexxExpat:] No description supplied by 'expat'. :-(

ExpatSetNamespaceDeclHandler

```
ExpatSetNamespaceDeclHandler(numeric parser,
                             alphanumeric StartNamespaceDeclHandler_ProcName,
                             alphanumeric StartNamespaceDeclHandler_EnvName,
                             alphanumeric StartNamespaceDeclHandler_Instore,
                             alphanumeric EndNamespaceDeclHandler_ProcName,
                             alphanumeric EndNamespaceDeclHandler_EnvName,
                             alphanumeric EndNamespaceDeclHandler_Instore);
```

Arguments passed to the StartNamespaceDeclHandler:

```
PARSE ARG parser, prefix, uri
```

Arguments passed to the EndNamespaceDeclHandler:

```
PARSE ARG parser, prefix
```

[Cooper:] Set handlers for namespace declarations. Namespace declarations occur inside start tags. But the namespace declaration start handler is called before the start tag handler for each namespace declared in that start tag. The corresponding namespace end handler is called after the end tag for the element the namespace is associated with. [Expat:] For an xmlns attribute, prefix will be null. For an xmlns="" attribute, uri will be null.

ExpatSetNotStandaloneHandler

```
ExpatSetNotStandaloneHandler(numeric parser,
                             alphanumeric NotStandaloneHandler_ProcName,
                             alphanumeric NotStandaloneHandler_EnvName,
                             alphanumeric NotStandaloneHandler_Instore);
```

Arguments passed to the NotStandaloneHandler:

```
PARSE ARG parser
```

[Cooper:] Set a handler that is called if the document is not "standalone". This happens when there is an external subset or a reference to a parameter entity, but does not have standalone set to "yes" in an XML declaration.
If this handler returns 0, then the parser will throw an XML_ERROR_NOT_STANDALONE error.

ExpatSetExternalEntityRefHandler

```
ExpatSetExternalEntityRefHandler(numeric parser,
                                 alphanumeric ExternalEntityRefHandler_ProcName,
                                 alphanumeric ExternalEntityRefHandler_EnvName,
                                 alphanumeric ExternalEntityRefHandler_Instore);
```

Arguments passed to the ExternalEntityRefHandler:

```
PARSE ARG parser, context, base, systemId, publicId
```

[Cooper:] Set an external entity reference handler. This handler is also called for processing an external DTD subset if parameter entity parsing is in effect. (See **ExpatSetParamEntityParsing**) ([Expat:] The referenced entity is not automatically parsed. The application can parse it immediately or later using **ExpatExternalEntityParserCreate**.)

[Expat:] The parser argument is the parser parsing the entity containing the reference; it can be passed as the parser argument to **ExpatExternalEntityParserCreate**. The context argument specifies the parsing context in the format expected by the context argument to **ExpatExternalEntityParserCreate**; context is valid only until the handler returns, so if the referenced entity is to be parsed later, it must be copied. [Cooper:] The base parameter is the base to use for relative system identifiers. It is set by ExpatSetBase and may be null. The public id parameter is the public id given in the entity declaration and may be null [Expat:] if none was specified. The whitespace in the public identifier will have been normalized as required by the XML spec. [Cooper:] The system id is the system identifier specified in the entity declaration and is never null.

[Cooper:] This handler returns an integer. A non-zero value should be returned for successful handling of the external entity reference. Returning a zero indicates failure, and causes the calling parser to return an XML_ERROR_EXTERNAL_ENTITY_HANDLING error.

[RexxExpat:] This handler may be called recursively, when the body of an external entity is parsed recursively. So, be careful when using the variable pool functions, since all recursively created parsers will use the same variable pool!

ExpatSetUnknownEncodingHandler

```
ExpatSetUnknownEncodingHandler(numeric parser,
                               alphanumeric UnknownEncodingHandler_ProcName,
                               alphanumeric UnknownEncodingHandler_EnvName,
                               alphanumeric UnknownEncodingHandler_Instore);
```

Arguments passed to the UnknownEncodingHandler:

```
PARSE ARG parser, name
```

[Cooper:] Set a handler to deal with encodings other than the built in set. If the handler knows how to deal with an encoding with the given name, it should [RexxExpat:] return a string with 256 numbers separated by a non numeric character. Start out with a number. Terminate with a non numeric character. Left out numbers will be replaced by their index. The resulting string must describe a suitable encoding (see below) or the parser will return an XML_UNKNOWN_ENCODING error.
If the handler does not know how to deal with the encoding named in name, it should return a non-valid value (NULL, nothing or a zero length value).
The actual conversion is accomplished by a call to the function set by **`ExpatSetUnknownEncodingConverter`**.

Example: **`Return "//////////10//////////20//////////30//////-`**
`3////40//////////50//////////60//////////70//////////80//////////90//////////100/////`
`/////110//////////120//////////130/////";`

[RexxExpat:] The returned string must [Cooper:] contain information for every possible possible leading byte in a byte sequence. If the corresponding value is >= 0, then it's a single byte sequence and the byte encodes that Unicode value. If the value is -1, then that byte is invalid as the initial byte in a sequence. If the value is -n, where n is an integer > 1, then n is the number of bytes in the sequence.

Once again in words of 'expat':
[Expat:] The string returned by the **`ExpatUnknownEncodingHandler`** (returned_string) is to provide information to the parser about encodings that are unknown to the parser. The returned_string[b] member gives information about byte sequences whose first byte is b.

- If returned_string[b] is c where c is >= 0, then b by itself encodes the Unicode scalar value c.
- If returned_string[b] is -1, then the byte sequence is malformed.
- If returned_string[b] is -n, where n >= 2, then b is the first byte of an n-byte sequence that encodes a single Unicode scalar value.

IMPORTANT!

[Expat:] Expat places certain restrictions on the encodings that are supported using this mechanism.

1. Every ASCII character that can appear in a well-formed XML document, other than the characters $@\^`{}~ must be represented by a single byte, and that byte must be the same byte that represents that character in ASCII.
2. No character may require more than 4 bytes to encode.
3. All characters encoded must have Unicode scalar values <= 0xFFFF, (i.e. characters that would be encoded by surrogates in UTF-16 are not allowed). Note that this restriction doesn't apply to the built-in support for UTF-8 and UTF-16.
4. No Unicode character may be encoded by more than one distinct sequence of bytes.

ExpatSetUnknownEncodingConverter

```
ExpatSetUnknownEncodingConverter(numeric parser,
                                 alphanumeric UnknownEncodingConverter_ProcName,
                                 alphanumeric UnknownEncodingConverter_EnvName,
                                 alphanumeric UnknownEncodingConverter_Instore,
                                 alphanumeric UnknownEncodingRelease_ProcName,
                                 alphanumeric UnknownEncodingRelease_EnvName,
                                 alphanumeric UnknownEncodingRelease_Instore);
```

Arguments passed to the UnknownEncodingConverter:

```
PARSE ARG parser, string2convert
```

Arguments passed to the UnknownEncodingRelease:

```
PARSE ARG parser
```

[RexxExpat:] Set the converter (and release) functions to encode unknown encodings. The encoding information must be properly set in the UnknownEncodingHandler.
[Expat:] The convert function is used to convert the multibyte sequences [RexxExpat:] passed as second argument. [Expat:] The convert function must return the Unicode scalar value represented by this byte sequence or -1 if the byte sequence is malformed. The convert function may be unset if the encoding is a single-byte encoding, that is if returned_string[b] >= -1 for all bytes b. When the parser is finished with the encoding, then if the release function is set, it will call release passing it the parser reference; once release has been called, the convert function will not be called again.

Parse position and error reporting functions

[Cooper:] These are the functions you'll want to call when the parse functions return 0, although the position reporting functions are useful outside of errors. The position reported is either [Expat:] the location of the character at which the error was detected or that of the first of the sequence of characters that generated the current event.

ExpatGetErrorCode

```
numeric ExpatGetErrorCode(numeric parser)
```
[Cooper:] Return what type of error has occurred.

ExpatErrorString

```
alphanumeric ExpatErrorString(numeric code)
```
[Cooper:] Return a string describing the error corresponding to code. [RexxExpat:] The code should be retrieved calling **`ExpatGetErrorCode`**.

ExpatGetCurrentByteIndex

```
numeric ExpatGetCurrentByteIndex(numeric parser)
```
[Cooper:] Return the byte offset of the position.

ExpatGetCurrentByteCount

```
numeric ExpatGetCurrentByteCount(numeric parser)
```
[Expat:] Return the number of bytes in the current event. Returns 0 if the event is in an internal entity.

ExpatGetCurrentLineNumber

```
numeric ExpatGetCurrentLineNumber(numeric parser)
```
[Cooper:] Return the line number of the position. [RexxExpat:] Unfortunately, when fatal parsing errors

were detected, it happens that 'expat's **XML_GetCurrentLineNumber** function causes a memory access violation :-(.

ExpatGetCurrentColumnNumber

`numeric ExpatGetCurrentColumnNumber(numeric parser)`
[Cooper:] Return the offset, from the beginning of the current line, of the position. [RexxExpat:] Unfortunately, when fatal parsing errors were detected, it happens that 'expat's **XML_GetCurrentColumnNumber** function causes a memory access violation :-(.

Miscellaneous functions

[Cooper:] The functions in this section either obtain state information from the parser or can be used to dynamically set parser options.

[RexxExpat:] Internally, RexxExpat makes use of 'expat's **XML_SetUserData** and **XML_GetUserData** functions; but it won't make sense to expose them to a Rexx environment. To pass user specific data from the main program to the callback routines (and vice versa) use the variable pool functions instead.

RexxExpat does not wrap (and does not use) 'expat's **XML_UseParserAsHandlerArg** function either.

ExpatSetBase

`numeric ExpatSetBase(numeric parser, alphanumeric base)`
[Expat:] Sets the base to be used for resolving relative URIs in system identifiers in declarations. Resolving relative identifiers is left to the application: this value will be passed through as the base argument to the **`ExpatExternalEntityRefHandler`**, **`ExpatNotationDeclHandler`**, and **`ExpatUnparsedEntityDeclHandler`**. The base argument will be copied.
Returns 0 if out of memory, non-zero otherwise.

ExpatGetBase

`alphanumeric ExpatGetBase(numeric parser)`
[Cooper:] Return the base for resolving relative URIs.

ExpatGetSpecifiedAttributeCount

`numeric ExpatGetSpecifiedAttributeCount(numeric parser)`
[Cooper:] When attributes are reported to the start handler in the argument list, attributes that were explicitly set in the element occur before any attributes that receive their value from default information in an ATTLIST declaration. This function returns the number of attributes that were explicitly set, thus giving the offset of the first attribute set due to defaults. It supplies information for the last call to a start handler. If you're in a start handler, then that means the current call.

ExpatGetIdAttributeIndex

`numeric ExpatGetIdAttributeIndex(numeric parser)`
[Expat:] Returns the index of the ID attribute passed in the last call to ExpatStartElementHandler, or -1 if

there is no ID attribute. Each attribute/value pair counts as 2; thus this corresponds to an index into the argument list passed to the ExpatStartElementHandler.

ExpatSetEncoding

`numeric ExpatSetEncoding(numeric parser, alphanumeric encoding)`
[Cooper:] Set the encoding to be used by the parser. It is equivalent to passing a non-null encoding argument to the parser creation functions. It must not be called after **`ExpatParser`** have been called on the parser.

ExpatSetParamEntityParsing

`numeric ExpatSetParamEntityParsing(numeric parser, alphanumeric code)`
[Expat:] Controls parsing of parameter entities (including the external DTD subset). If parsing of parameter entities is enabled, then references to external parameter entities (including the external DTD subset) will be passed to the handler set with **`ExpatSetExternalEntityRefHandler`**. The context passed will be 0.
Unlike external general entities, external parameter entities can only be parsed synchronously. If the external parameter entity is to be parsed, it must be parsed during the call to the ExternalEntityRefHandler: the complete sequence of **`ExpatExternalEntityParserCreate`**, **`ExpatParse`** and **`ExpatParserFree`** calls must be made during this call. After ExpatExternalEntityParserCreate has been called to create the parser for the external parameter entity (context must be 0 for this call), it is illegal to make any calls on the old parser until ExpatParserFree has been called on the newly created parser.
If the library has been compiled without support for parameter entity parsing (i.e. without XML_DTD being defined), then **`ExpatSetParamEntityParsing`** will return 0 if parsing of parameter entities is requested; otherwise it will return non-zero.
Possible choices for code are: ([RexxExpat:] If your code does not match with one of the following (and only with them), code is interpreted as a numeric value. So, if 'expat's enumeration type XML_ParamEntityParsing should change one day, please refer to xmlparse.h to find the legal numeric values).

- "ExpatPARAM_ENTITY_PARSING_NEVER"
- "ExpatPARAM_ENTITY_PARSING_UNLESS_STANDALONE"
- "ExpatPARAM_ENTITY_PARSING_ALWAYS"

[RexxExpat:] The following functions **ExpatLoadFuncs** and **ExpatDropFuncs** are RexxExpat utility functions and do not wrap any 'expat' functions!

ExpatLoadFuncs

`ExpatLoadFuncs()`
[RexxExpat:] Loads all function from the RexxExpat library described here.

ExpatDropFuncs

`ExpatDropFuncs()`
[RexxExpat:] Unloads all function of the RexxExpat library described here.

Variable pool functions

[RexxExpat:] The variable pool functions are provided to exchange data between the main program and the callback routines. This came into need since the callback routines run on newly instanciated Rexx interpreters and thus are having a separate environment.

The following functions are RexxExpat utility functions and do not wrap any 'expat' functions!

ExpatSetVariablePool

`numeric ExpatSetVariablePool(numeric parser, alphanumeric varnames, ...)`
[RexxExpat:] Append variables to the variable pool. Appended are all variables named in the argument list following the parser reference. The variables are set to the value which they have when the function **`ExpatSetVariablePool`** is called. If you only supply the parser reference and no further variable name EVERY variable visible at invocation time of the function is appended to the variable pool. Use this function carefully so that you won't run into insufficient memory. Call **`ExpatFreeVariablePool`** to free memory occupied by the variable pool.
If the returned value is greater than 127 a fatal error occurred and the entire request failed. If the returned value is between 0 and 127 only some variables could not be set. The function returns 0 if no error was found.

ExpatGetVariablePool

`numeric ExpatGetVariablePool(numeric parser)`
[RexxExpat:] Fetch all variables from the variable pool. If the returned value is greater than 127 a fatal error occurred and the entire request failed. If the returned value is between 0 and 127 some variables could not be fetched. The function returns 0 if no error was found.

ExpatFreeVariablePool

`numeric ExpatFreeVariablePool(numeric parser, alphanumeric encoding)`
[RexxExpat:] Free the entire memory occupied by the variable pool. Returns 0 if successful, a non-zero value otherwise.

Helper functions

[RexxExpat:] The following is meant for the developers among you. It describes the internally used helper functions of the RexxExpat library. All function names of library function which are meant for internal use only, begin with an underscore (_).

I tried to keep the code as understandable as possible. So handler setting functions and the handler functions themselves should match a common pattern. Therefore I out-sourced all data management tasks (generating, initiating and freeing) to some helper functions. They are called by the handler setting routines to manage handler information, by the handler functions themselves to manage the arguments lists which are to be passed to the RexxStart function, and by the variable pool functions to free the variable pool.

_ExpatSetHandler

```
APIRET _ExpatSetHandler  (EXPATHANDLER *Handler,
                          RXSTRING ProcName,
```

```
                          RXSTRING EnvName,
                          RXSTRING Instore);
```

This function is used by the handler setting routines and will set handler information of a given **Handler** according to the passed values **ProcName**, **EnvName** and **Instore** (see paragraph Handler settings for their meanings).

_ExpatCopyHandler

```
APIRET _ExpatCopyHandler (EXPATHANDLER SourceHandler,
                          EXPATHANDLER *DestinationHandler);
```

This functions is used by the **ExpatExternalEntityParserCreate** function and copies the handler information from a given **SourceHandler** to a **DestinationHandler**. So handler information can be easily duplicated.

_ExpatFreeHandler

```
APIRET _ExpatFreeHandler (EXPATHANDLER *Handler);
```

This function is used by the handler setting routines and the **ExpatParserFree** function as well and frees the memory occupied by a given **Handler**.

Using these helper functions each handler setting routine is almost an exact match to the following pseudo-code:

```
APIRET APIENTRY ExpatSetHandlerFunction (APIINTERFACE)
{
   /* define local variables */
   RexxExpatData   userData;
   int             rc = 0;

   /* check on valid call - sufficient and valid arguments passed? */
   if (!(IsSufficient(ArgCount) && IsValid(ArgList)))
      return 40;                /* incorrect call – break, if not! */

   /* get reference to user data (first argument) */
   userData := ArgList[1];

   /* set handler – if ProcName (second argument) is defined! */
   if (IsValid(ArgList[2]))
      rc := _ExpatSetHandler (userData->Handler[HandlerIndex],
                              ArgList[2],
                              ArgList[3],
                              ArgList[4]);
   else   /* free handler */
      rc := _ExpatFreeHandler(userData->Handler[HandlerIndex]);

   /* register handler with 'expat' */
   XML_SetHandler(userData->Parser,
                  (userData->Handler[HandlerIndex] ? _ExpatHandlerFunction : NULL));

   /* set Rexx return code - must not be greater than 10^RXAUTOBUFLEN ;-) */
   ReturnString := Int2Str(rc);
```

```
   return 0;
}
```

So, to add a new handler setting procedure to RexxExpat there are only three things to adapt:
After copying an existent procedure, adjust the HandlerIndex of the Handler member of the RexxExpatData structure (be sure that your new HandlerIndex is listed in the enumeration EXPATHANDLERTYPES in rexxexpat.h as well!). Now modify the handler registration call to 'expat' and last not least change the name of the newly created RexxExpat procedure.

_ExpatMakeArgList

```
APIRET _ExpatMakeArgList    (ULONG ArgCount,
                             PRXSTRING *DestinationList,
                             char **SourceList);
```

This function is used by the handler functions and will generate an argument list of RXSTRINGS (as required by the RexxStart function; see [IBMPG:]) pointed to by **`DestinationList`** and initiates its elements according to the values in **`SourceList`** (which must be an array of NULL terminated strings). The initialisation halts when a NULL element is encountered in **`SourceList`**. The following arguments can then be set by calls to **`_ExpatAppend2ArgList`**.

_ExpatAppend2ArgList

```
APIRET _ExpatAppend2ArgList (PRXSTRING *DestinationList,
                             const char *ListEntry,
                             int EntryLength);
```

This function is used by the handler functions which are to pass non-NULL terminated values to the RexxStart functions: The first element of **`DestinationList`** which is NULL will be replaced by **`ListEntry`** of length **`EntryLength`**. For successful assignment **`DestinationList`** must provide such (distinct, NULL-) element.

_ExpatFreeArgList

```
APIRET _ExpatFreeArgList    (ULONG ArgCount,
                             PRXSTRING *ArgList);
```

This function is used by the handler setting routines and frees the memory occupied by the passed array of RXSTRINGs created by **`_ExpatMakeArgList`**.

Using these helper functions each handler function is almost an exact match to the following pseudo-code:

```
void _ExpatHandlerFunction (void *pUserData, const XML_Char *data)
{
   /* define local variables */
   RexxExpatData userData = (RexxExpatData) pUserData;
```

```
   char*         args      [3];
   const ULONG   ArgCount = 3;
   PRXSTRING     ArgList;
   short         result_int;
   RXSTRING      result_str;
   int           currentHandlerType = OnHandlerEvent;

   /* break - if handler's not set */
   if (!userData->Handler[currentHandlerType]) return;

   /* make ArgList */
   args[0] := userData;
   args[1] := data;
   args[2] := NULL;
   _ExpatMakeArgList (ArgCount, ArgList, args);

   /* call RexxStart */
   RexxStart (
      ArgCount,                                  /* number of arguments
*/
      ArgList,                                   /* pointer to argument array
*/
      userData->Handler[currentHandlerType]->ProcName, /* Rexx proc (code or file
name) */
      userData->Handler[currentHandlerType]->Instore, /* is ProcName code or file
name? */
      userData->Handler[currentHandlerType]->EnvName, /* initial ADDRESS environment
*/
      RXSUBROUTINE,                              /* call mode (here always
RXSUBROUTINE) */
      NULL,                                      /* RexxRegisterExitDll not
implemented! */
      &result_int,                               /* returned result as int
*/
      &result_str));                             /* returned result as RXSTRING
*/

   /* free memory */
   _ExpatFreeArgList (ArgCount, ArgList);

   /* wait for termination of all asynchronious Rexx procedures */
   REXXWAITFORTERMINATION;
}
```

Again, there are only few things to be done to add a new handler to RexxExpat. First, simply copy an existing handler. Second, provide enough space in the **`args`** array; set **`ArgCount`** accordingly. Third, adjust the **`currentHandlerType`** (again, be sure it's an index matching with the enumeration EXPATHANDLERTYPES in rexxexpat.h!). Then call **`_ExpatMakeArgList`** to create the **`ArgList`** and use **`_ExpatAppend2ArgList`** if you need to pass non-NULL-terminated strings. At last, add any handler specific code if needed (e.g. for return code processing as in _ExpatUnknownEncodingHandler).

_ExpatFreeRexxVariablePool

```
APIRET _ExpatFreeRexxVariablePool (PSHVBLOCK *FirstPointer);
```

This function is used by the variable pool functions to free the memory occupied by the variable pool. It frees the array of SHVBLOCKs pointed to by **`FirstPointer`**.