



ERXXML

Parse an XML Document on z/OS using REXX

Version 1.0

July 15, 2011



Table of Contents

| | |
|--|----------|
| I ERXXML – Parse XML Document | 3 |
| I.1 Overview..... | 3 |
| I.2 Invocation..... | 3 |
| I.3 REXX Control Variables | 4 |
| I.4 REXX Result Variables..... | 5 |
| I.5 Sample Illustration..... | 5 |
| I.6 Error processing..... | 6 |
| I.6.1 Parsing Errors..... | 6 |
| I.7 Example 1: using ERXXML in Batch | 7 |
| I.8 Example 2: using Dataset Name | 9 |

List of Figures

| | |
|--|----------|
| <i>Figure 1: Parsing error example.....</i> | <i>6</i> |
| <i>Figure 2: Sample JCL.....</i> | <i>7</i> |
| <i>Figure 3: Sample REXX with DDNAME.....</i> | <i>7</i> |
| <i>Figure 4: Sample REXX with nodes</i> | <i>8</i> |
| <i>Figure 5: Sample XML file.....</i> | <i>9</i> |
| <i>Figure 6: Sample REXX with dataset specification.....</i> | <i>9</i> |

List of Tables

| | |
|---|----------|
| <i>Table 1: Code Pages.....</i> | <i>4</i> |
| <i>Table 2: Sample XML.....</i> | <i>5</i> |
| <i>Table 3: Generated REXX variables (for the above sample XML)</i> | <i>5</i> |
| <i>Table 4: _VN REXX variables.....</i> | <i>5</i> |
| <i>Table 5: Sample REXX node variables</i> | <i>8</i> |



I ERXXML – Parse XML Document

I.1 Overview

The ERXXML function is an XML parser (based on the XML System Service integrated in z/OS (as of z/OS 1.8) that can be invoked directly from REXX. The content of the parsed document is returned as REXX compound variables with the associated element name.

The following algorithm is used to create a compound variable name:

- The root element is the stem name
- Each start element is used as the next part of the compound variable
- The occurrence number of each unique compound variable is the final tail part (1, 2,...)
- Any attribute name is used as tail part

“Table 2: Sample XML” on page 5 illustrates this methodology.

I.2 Invocation

```
frc = ERXXML(dsname|DD:ddname, [node ]... )
```

dsname | DD:ddname Specification of the XML file, either as dataset name or ddname.

node (optional) A list of fully-qualified node names each separated with a blank. If specified, variables will be created only for these nodes (this can be used to reduce the number of generated REXX variables). Default: REXX variables are created for all nodes.

Return value (<frc> above):

| | |
|----|---|
| 0 | OK |
| 8 | Parsing error |
| 12 | File open error |
| 24 | Argument length error |
| 28 | Invalid argument |
| 32 | Argument missing |
| 36 | XML file buffer overflow |
| 40 | REXX variable name too long (> 250 characters; REXX limitation) |
| 44 | VNTAB overflow (too many variables) |
| 48 | Storage for variable cannot be allocated |
| 99 | Internal error |



I.3 REXX Control Variables

Using the following control variables is optional

| | |
|-----------------------|--|
| <code>_TRACE</code> | Program tracing (input): 1 = enable, 0 = disable (default). |
| <code>_MAXVN</code> | Maximum number name entries (default: 1000). |
| <code>_XMLSIZE</code> | Maximum size of input XML file (default: 50,000 bytes). |
| <code>_CCSID</code> | Explicit document CCSID (by default the document CCSID is used, or, if omitted, 1047). If specified, the explicit document CCSID takes priority. |

Note:

The set values for `_MAXVN` and `_XMLSIZE` affect the region. `_TRACE` is used only for debugging purposes.

The following table lists the supported code pages and the associated CCSID values:

| Code page | CCSID |
|-----------|-------|
| IBM_037 | 37 |
| IBM_1047 | 1047 |
| IBM_1140 | 1140 |
| IBM_1141 | 1141 |
| IBM_1142 | 1142 |
| IBM_1143 | 1143 |
| IBM_1144 | 1144 |
| IBM_1145 | 1145 |
| IBM_1146 | 1146 |
| IBM_1147 | 1147 |
| IBM_1148 | 1148 |
| IBM_1149 | 1149 |
| IBM_273 | 273 |
| IBM_277 | 277 |
| IBM_278 | 278 |
| IBM_280 | 280 |
| IBM_284 | 284 |
| IBM_285 | 285 |
| IBM_297 | 297 |
| IBM_500 | 500 |
| IBM_871 | 871 |
| UTF_8 | 1208 |
| UTF_16 | 1200 |

Table I: Code Pages



I.4 REXX Result Variables

| | |
|--|---|
| <code>_VN.0</code> | The number of nodes |
| <code>_VN.i</code> | Each node name |
| <code><node></code> | Element without data |
| <code><node>.<ix></code> <code><ix> = 1,...</code> | Each node (<code>ix = 0 = count</code>) |
| <code><node>.<ix>.<attribute name></code> | Element with attribute |

I.5 Sample Illustration

The following table uses a small (but comprehensive) XML document to illustrate the naming convention for the generated REXX variables.

| | Hier-archy | Generated variable name (= <code>_VN.n</code> data value) | Tail | n (tail of <code>_VN.n</code>) | Data value |
|---|------------|---|----------|---------------------------------|------------|
| <code><?xml version="1.1" encoding="ibm-1047"?></code> | | | | | |
| <code><TABLE></code> | 1 | TABLE | .1 | 1 | |
| <code><HEADER></code> | 2 | TABLE.HEADER | .1 | 2 | |
| <code><TITLE>Sample</TITLE></code> | 3 | TABLE.HEADER.TITLE | .1 | 3 | Sample |
| <code></HEADER></code> | | | | | |
| <code><COLUMNS>PNO</COLUMNS></code> | 2 | TABLE.COLUMNS | .1 | 4 | PNO |
| <code><VALUES>1234</VALUES></code> | 2 | TABLE.VALUES | .1 | 5 | 1234 |
| <code><VALUES ALPHA="1" BETA="23">56789</VALUES></code> | 2 | TABLE.VALUES | .2 | 6 | 56789 |
| | | TABLE.VALUES | .2.ALPHA | | 1 |
| | | TABLE.VALUES | .2.BETA | | 23 |
| <code></TABLE></code> | | | | | |

Table 2: Sample XML

| | |
|----------------------|--------|
| TABLE.VALUES.0 | 2 |
| TABLE.VALUES.1 | 1234 |
| TABLE.VALUES.2 | 56789 |
| TABLE.VALUES.2.ALPHA | 1 |
| TABLE.VALUES.2.BETA | 23 |
| TABLE.COLUMNS.0 | 1 |
| TABLE.COLUMNS.1 | PNO |
| TABLE.HEADER.TITLE.0 | 1 |
| TABLE.HEADER.TITLE.1 | Sample |

Table 3: Generated REXX variables (for the above sample XML)

As usual for REXX compound variables, the 0 tail value contains the number of associated compound variables (with tail value 1, 2, etc.)

To facilitate navigation where the XML element names are not known, an auxiliary compound variable (`_VN`) is created for each element (see Table 4).

| | |
|------|--------------------|
| VN.0 | 6 |
| VN.1 | TABLE |
| VN.2 | TABLE.HEADER |
| VN.3 | TABLE.HEADER.TITLE |
| VN.4 | TABLE.COLUMNS |
| VN.5 | TABLE.VALUES |
| VN.6 | TABLE.VALUES |

Table 4: `_VN` REXX variables



1.6 Error processing

Two types of error can occur while executing ERXXML:

- Logical errors
- Parsing errors (errors detected by the XMLSS Parser)

If an error is detected, ERXXML will terminate with the appropriate return code set (REXX variable RC) and a short explanatory message written to the joblog (JESMSG LG). In case of a parsing error, the XMLSS return code and reason code, the document offset and the document text at this offset are also written to the joblog (see also parsing error example, Figure 1).

1.6.1 Parsing Errors

Parsing error example:

```

PRS RTC:12 RSC:3035

GXL1PRS RTC:12 RSC:3035

document offset:128

error text: <VALUES> ALPHA="1" BETA="23">56789</VALES></TABLE>
    
```

Figure 1: Parsing error example

In an error situation, for example, caused by an invalid XML document, the XML System Services (XMLSS) Parser will issue an error code (return code > 4 with the appropriate hexadecimal reason code as documented in the XML System Services User's Guide and Reference).

To simplify troubleshooting, the ERXXML outputs three additional message lines (similar to those shown above):

- Line 1: The parser service, the return code (RTC keyword, decimal) and the reason code (RSC keyword, hexadecimal)
- Line 2: The offset in the document where the error was detected
- Line 3: A snippet of the document near this address

In this case, the **GXL1PRS** service sets return code 12 and reason code 3035. This reason code indicates an end-tag mismatch; <VALUES> and </VALES> do not match.



1.7 Example I: using ERXXML in Batch

Sample implementation JCL (via IRXJCL):

```
//S1 EXEC PGM=IRXJCL, PARM='XMLEXEC'
//STEPLIB DD DSN=SO FAR.LOAD, DISP=SHR
//SYSEXEC DD DSN=SO FAR.EXEC, DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSXMLIN DD DSN=SO FAR.XML (XMLDOC) , DISP=SHR
```

Figure 2: Sample JCL

Note:

- The STEPLIB must be changed to specify the library that contains the ERXXML load module.
- The SYSEXEC must be changed to specify the library that contains the REXX procedure.
- The SYSXMLIN must be changed to specify the library that contains the XML file.

Sample REXX XMLEXEC member of DD:SYSEXEC:

```
/* REXX */
_MAXVN = 50 /* set the maximum no. of variables */
_CC SID = 1047 /* set CCSID to 1047 */
frc = ERXXML('DD:SYSXMLIN')
/* List TABLE.VALUES elements */
IF frc = 0 THEN DO
  DO i = 1 TO TABLE.VALUES.0
    SAY TABLE.VALUES.i
  END
END
END
```

Figure 3: Sample REXX with DDNAME



This simple example that uses the same XML data as shown in Table I lists the elements for TABLE.VALUES, namely:

| |
|---------------|
| 1234 56789 |
|---------------|

To reduce the number of generated variables, the following REXX sample specifies two node names:

```
/* REXX */
_MAXVN = 50 /* set the maximum no. of variables */
_CCSID = 1047 /* set CCSID to 1047 */
nodes = "TABLE.VALUES TABLE.COLUMNS"
frc = ERXXML('DD:SYSXMLIN',nodes)
```

Figure 4: Sample REXX with nodes

In this case, only the following REXX variables are set:

| | |
|----------------------|-------|
| TABLE.VALUES.0 | 2 |
| TABLE.VALUES.1 | 1234 |
| TABLE.VALUES.2 | 56789 |
| TABLE.VALUES.2.ALPHA | 1 |
| TABLE.VALUES.2.BETA | 23 |
| TABLE.COLUMNS.0 | 1 |
| TABLE.COLUMNS.1 | PNO |

Table 5: Sample REXX node variables



1.8 Example 2: using Dataset Name

Let's have a look at the following XML file:

```
<?xml version="1.1" encoding="ibm-1047"?>
<PROGDEF>
  <COMP>
    <DSN>MY.SOURCE</DSN>
    <LANG>ASM</LANG>
    <SYSLIB>LIBRARY1</SYSLIB>
    <SYSLIB>LIBRARY2</SYSLIB>
  </COMP>
  <LINK>
    <OPTION LIST="YES" RENT="NO">TEST</OPTION>
    <DSN>MY.LOAD</DSN>
    <SYSLIB>LOAD.LIBRARY1</SYSLIB>
  </LINK>
</PROGDEF>
```

Figure 5: Sample XML file

The next REXX sample uses direct specification of the dataset name and nodes to get access to the <SYSLIB> values.

```
/* REXX */

nodes = 'PROGDEF.COMP'

frc = ERXXML('SOFAR.XML(XML0)',nodes)

/* navigate */

DO i = 1 TO PROGDEF.COMP.SYSLIB.0
  SAY PROGDEF.COMP.SYSLIB.i
END
```

Figure 6: Sample REXX with dataset specification

This example navigates to the

```
<PROGDEF>
  <COMP>
    <SYSLIB>
```

node and lists its elements:

```
LIBRARY1
LIBRARY2
```