



ERXXML

Parse an XML Document on z/OS using REXX

Version 1.5

October 12, 2011



Table of Contents

I	ERXXML – Parse XML Documents.....	4
1.1	Preamble.....	4
1.2	Overview.....	4
1.3	Invocation.....	5
1.4	REXX Control Variables	5
1.4.1	Output Variables.....	6
1.5	REXX Result Variables.....	6
1.6	Sample Illustration	7
1.7	Error Processing.....	8
1.7.1	Parsing Errors	8
1.8	Using ERXXML in Batch (IRXJCL).....	8
1.9	Using ERXXML in TSO (IKJEFT01)	9
1.10	Use of Filters	9
1.11	Examples.....	10
1.11.1	Example 1 - List Attributes.....	10
1.11.2	Example 2 - Parse Error Feedback File.....	11
1.11.3	Example 3 - Use of Filters and Prefixes to Simplify Processing	13
1.11.4	Example 4 - Process RDz Properties	14
2	ERXXML – Installation Instructions	15
2.1	Program Materials	15
2.2	Requirements	15
2.2.1	Machine Requirements	15
2.2.2	Software Requirements.....	15
2.3	Installation Procedure	15
2.3.1	Upload the Software	15
2.3.2	Receive the Software	16
2.3.1	Installation Verification	18



List of Figures

Figure 1: Sample XML.....	7
Figure 2: Parsing error example.....	8
Figure 3: Sample JCL for IRXJCL.....	8
Figure 4: Sample REXX with DDNAME.....	9
Figure 5: Sample JCL.....	9
Figure 6: Sample REXX with filter.....	9
Figure 7: Sample XML file with attributes.....	10
Figure 8: Sample REXX for attribute processing.....	10
Figure 9: Sample output for attribute processing.....	10
Figure 10: Sample error feedback file.....	11
Figure 11: Sample REXX for processing an error feedback file.....	12
Figure 12: Associated error feedback file output.....	13
Figure 13: Sample REXX showing the use of filters and prefixes.....	13
Figure 14: Sample REXX code to process RDz properties.....	14
Figure 15: JCL to allocate the installation XMIT file.....	16
Figure 16: FTP sample to upload the installation package.....	16
Figure 17: Receive the installation software.....	17
Figure 18: Receive the ERXXML PDS datasets.....	17

List of Tables

Table 1: Code pages.....	6
Table 2: ERXXML result variables.....	6
Table 3: Example of ERXXML result variables.....	7
Table 4: Sample REXX filtered variables.....	9



I ERXXML – Parse XML Documents

I.1 Preamble

The ERXXML function is a REXX interface to the high-performance XML System Service (XMLSS) integrated in z/OS (as of z/OS 1.8). This combines the ease-of-use of REXX with the performance provided by the XMLSS parser.

I.2 Overview

The ERXXML function is an XML parser (based on the high-performance XML System Service integrated in z/OS) that can be invoked directly from REXX. Because the content of the parsed document is returned as REXX compound variables with the associated tag name (an explicit prefix can be specified to ensure unique namespaces when ERXXML is invoked more than once), ERXXML greatly simplifies the processing of XML documents.

The following algorithm is used to create the REXX variables:

- A stem variable pair (tag name (`_TN.`) and tag data (`_TD.`), respectively) is created for each tag; the stem index corresponds to the order of the tag in the XML document
- A stem variable pair (attribute name (`_AN.`) and attribute value (`_AV.`), respectively) is created for each attribute of a tag (the primary stem index is the same as for the associated tag); the secondary stem index is the ordinal number of the tag attribute within the tag
- To allow direct navigation, a stem variable (`_EX.`) is created as index to each element; the element index is the ordinal number of the first associated tag
- As usual for REXX, the `.0` tail contains the number of associated stem variables

To reduce the number of generated variables, a tag name filter can be specified. Filters can also be used to simplify the processing.

Note:

Because of REXX language considerations, ERXXML tags are not case sensitive (all tag names are always folded to uppercase although the associated data remain unchanged).



I.3 Invocation

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

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

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

Return codes (<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 the <code>_XMLSIZE</code> variable can be used to increase the default size
40	REXX variable data too long
44	Hierarchy table overflow (maximum 32 levels are currently supported)
48	Storage for variable cannot be allocated
52	Too many variables (the <code>_MAXVN</code> variable can be used to increase the default size)
99	Internal error

Note:

A trial version (ERXXMLT) with restricted functionality (maximum number of variables: 100; maximum size of the XML file: 5000 bytes) is available for evaluation purposes). Except for the function name (ERXXMLT rather than ERXXML, and no support for the `_MAXVN` and `_XMLSIZE` variables), the invocation is identical.

I.4 REXX Control Variables

The use of the following control variables is optional

<code>_TRACE</code>	Program tracing: 1 = enable, default no trace.
<code>_MAXVN</code>	Maximum number of generated variables (default: 1000, 100 for the trial version).
<code>_XMLSIZE</code>	Maximum size of input XML file (default: 50,000 bytes, 5000 for the trial version).
<code>_CCSID</code>	Explicit document CCSID (by default the CCSID specified in the document is used, or, if omitted, 1047). If specified, the explicit <code>_CCSID</code> takes priority.
<code>_PFX</code>	Prefix suffixed to the generated variables; the use of a prefix allows the creation of explicit namespaces for variable names, for example, when ERXXML is invoked more than once within an exec. The prefix must conform to REXX naming conventions.

Note:

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

Table I lists the supported code pages and the associated CCSID values.



Code page	CCSID
IBM_037	37
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
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

Table 1: Code pages

1.4.1 Output Variables

In addition to the result variables described in Section 1.5, ERXXML also sets the following output variables:

- _MSG** The error message as written to joblog. **_MSG** is set only when ERXXML terminates with a non-zero return code.
- _VERSION** The ERXXML version identifier.

1.5 REXX Result Variables

ERXXML creates REXX stem variables for the parsed XML file. Table 2 summarises the created variables.

<code>_TN.0</code>	Number of tags
<code>_TN.i</code>	Tag name (fully qualified)
<code>_TD.i</code>	Data for tag <i>i</i>
<code>_AN.i.0</code>	Number of attributes for tag <i>i</i>
<code>_AN.i.j</code>	Name of attribute <i>j</i> for tag <i>i</i>
<code>_AV.i.j</code>	Value of attribute <i>j</i> for tag <i>i</i>
<code>_EX.k.0</code>	Number of tags in element <i>k</i>
<code>_EX.k</code>	Index to the first tag of element <i>k</i>

Table 2: ERXXML result variables



1.6 Sample Illustration

Table 3 uses a small (but comprehensive) XML document to illustrate the naming convention for the generated REXX variables.

Element index to tag	Element index	Tag number	Variable	Tag name
			_TN.0 = 8 (number of tags) _EX.0 = 3 (number of elements)	
EX.1 ↳ _TN.1	1	1	<TABLE> ↳ _TN.1	TABLE
EX.2 ↳ _TN.2	2	2	<HEADER> ↳ _TN.2	TABLE.HEADER
	2	3	<TITLE>Sample</TITLE> ↳ _TD.3 ↳ _TN.3	TABLE.HEADER.TITLE
			</HEADER>	
EX.3 ↳ _TN.4	3	4	<COLUMNS> ↳ _TN.4	TABLE.COLUMNS
	3	5	<NAME>PNO</NAME> ↳ _TD.5 ↳ _TN.5	TABLE.COLUMNS.NAME
	3	6	<VALUES>1234</VALUES> ↳ _TD.6 ↳ _TN.6	TABLE.COLUMNS.VALUES
	3	7	<VALUES ALPHA="1" BETA="23">56789</VALUES> ↳ _TD.7 ↳ _AN.7.0 = 2 ↳ _AV.7.2 ↳ _AN.7.2 ↳ _AV.7.1 ↳ _AN.7.1 ↳ _TN.7	TABLE.COLUMNS.VALUES
	3	8	<VALUES></VALUES> ↳ _TD.8 (null) ↳ _TN.8	TABLE.COLUMNS.VALUES
			</COLUMNS>	
			</TABLE>	

Table 3: Example of ERXXML result variables

Figure 1 shows the associated XML document; indentation is used only for clarity.

```
<?xml version="1.1" encoding="ibm-1047"?>
<TABLE>
  <HEADER>
    <TITLE>Sample</TITLE>
  </HEADER>
  <COLUMNS>
    <NAME>PNO</NAME>
    <VALUES>1234</VALUES>
    <VALUES ALPHA="1" BETA="23">56789</VALUES>
    <VALUES></VALUES>
  </COLUMNS>
</TABLE>
```

Figure 1: Sample XML



1.7 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 (JESMSGLOG). 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).

1.7.1 Parsing Errors

Parsing error example:

```
PRR RTC:12 RSC:3035
GXL1PRR RTC:12 RSC:3035
document offset:128
error text: <VALUES> ALPHA="1" BETA="23">56789</VALES></TABLE>
```

Figure 2: 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 **GXL1PRR** service sets return code 12 and reason code 3035. This reason code indicates an end-tag mismatch; <VALUES> and </VALES> do not match.

1.8 Using ERXXML in Batch (IRXJCL)

Sample implementation JCL:

```
//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 3: Sample JCL for IRXJCL

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 */
frc = ERXXML('DD:SYSXMLIN')
IF frc > 0 THEN EXIT frc /* parse error */
...
```

Figure 4: Sample REXX with DDNAME

1.9 Using ERXXML in TSO (IKJEFT01)

Sample implementation JCL with Batch TSO:

```
//S2          EXEC PGM=IKJEFT01, PARM='XMLEXEC'
//STEPLIB    DD DSN=SOFAR.LOAD, DISP=SHR
//SYSEXEC    DD DSN=SOFAR.EXEC, DISP=SHR
//SYSTSPRT   DD SYSOUT=*
//SYSUADS    DD DSN=SYS1.UADS, DISP=SHR
//SYSLBC     DD DSN=SYS1.BROADCAST, DISP=SHR
//SYSTSIN    DD DUMMY
//SYSPRINT   DD SYSOUT=*
//SYSXMLIN   DD DSN=SOFAR.XML(XMLDOC), DISP=SHR
```

Figure 5: Sample JCL

The same considerations as for ERXXML batch invocation apply.

1.10 Use of Filters

To reduce the number of generated variables, a filter can be specified as optional invocation parameter. The filter specifies the fully-qualified tag names for which variables will be created; multiple filters, each separated with a blank, may be specified. By default, no filtering is performed.

Figure 6 shows the specification of a filter.

```
/* REXX */
filter = "TABLE.COLUMNS.VALUES"
frc = ERXXML('DD:SYSXMLIN', filter)
```

Figure 6: Sample REXX with filter

In this case, only the REXX variables shown in Table 4 will be set (the XML file shown in Figure 1 is used).

_TN.0	3
_TN.1	TABLE.COLUMNS.VALUES
_TD.1	1234
_TN.2	TABLE.COLUMNS.VALUES
_TD.2	56789
_AN.2.0	2
_AN.2.1	ALPHA
_AV.2.1	1
_AN.2.2	BETA
_AV.2.2	23
_TN.3	TABLE.COLUMNS.VALUES
_TD.3	

Table 4: Sample REXX filtered variables



1.11 Examples

This section shows not only a number of examples of the use of ERXXML but also some techniques that can be used to simplify the processing.

1.11.1 Example 1 - List Attributes

This example navigates to the TABLE.COLUMNS element and then lists all the tag names and data (including attributes) for this element.

```
<?xml version="1.1" encoding="ibm-1047"?>
<TABLE>
  <HEADER>
    <TITLE>Sample</TITLE>
  </HEADER>
  <COLUMNS>
    <NAME>PNO</NAME>
    <VALUES>1234</VALUES>
    <VALUES ALPHA="1" BETA="23">56789</VALUES>
    <VALUES></VALUES>
  </COLUMNS>
</TABLE>
```

Figure 7: Sample XML file with attributes

```
/* REXX */
frc = ERXXML("'SOFAR.XML(XML)')")
IF frc > 4 THEN EXIT frc /* parse error */

/* Navigate to element */
DO eix = 1 TO _EX.0
  tix = _EX.eix /* tag index */
  IF _TN.tix = 'TABLE.COLUMNS' THEN CALL OK
END
EXIT

OK: /* element found */
DO j = 1 TO _EX.eix.0 /* loop over all element tags */
  SAY _TN.tix _TD.tix /* list tag name and data */
  /* test whether attributes for tag */
  IF DATATYPE(_AN.tix.0) = 'NUM' THEN DO k = 1 TO _AN.tix.0
    SAY _AN.tix.k _AV.tix.k /* list attribute name and value */
  END
  tix = tix+1
END
RETURN
```

Figure 8: Sample REXX for attribute processing

This exec produces the following output:

```
TABLE.COLUMNS
TABLE.COLUMNS.NAME PNO
TABLE.COLUMNS.VALUES 1234
TABLE.COLUMNS.VALUES 56789
ALPHA 1
BETA 23
TABLE.COLUMNS.VALUES
```

Figure 9: Sample output for attribute processing



1.11.2 Example 2 - Parse Error Feedback File

This more comprehensive example lists the content of the (two) MESSAGE elements in the sample error feedback file (Figure 10); the MESSAGE.MSGFILE tag specifies the file number. The associated file name for this file number is fetched from the FILEREFERENCETABLE element and displayed.

Figure 10 shows the sample error feedback file generated from a PLI Compile using the XML compile option.

```
<PACKAGE>
<FILEREFERENCETABLE>
<FILECOUNT>2</FILECOUNT>
<FILE>
<FILENUMBER>0</FILENUMBER>
<FILENAME>SCLM.DEV1.SOURCE (TESTMD2) </FILENAME>
</FILE>
<FILE>
<FILENUMBER>1</FILENUMBER>
<FILENAME>SCLM.TEST.SOURCE (TESTINC) </FILENAME>
</FILE>
</FILEREFERENCETABLE>
<MESSAGE>
<MSGNUMBER>IBM1316I E</MSGNUMBER>
<MSGLINE>17</MSGLINE>
<MSGFILE>0</MSGFILE>
<MSGTEXT>END label is not a label on any open group.</MSGTEXT>
</MESSAGE>
<MESSAGE>
<MSGNUMBER>IBM2619I W</MSGNUMBER>
<MSGLINE>10</MSGLINE>
<MSGFILE>0</MSGFILE>
<MSGTEXT>The include file SCLM.TEST.SOURCE (TESTINC) contains no cross-reference
</MSGTEXT>
</MESSAGE>
</PACKAGE>
```

Figure 10: Sample error feedback file

The associated processing exec is shown in Figure 11.



```

/* REXX */
filter = '' /* null filter */
frc = ERXXML("'SOFAR.XML(ERXXMLT2)'",filter)
IF frc > 4 THEN EXIT frc /* parse error */

/* Navigate (find PACKAGE.MESSAGE element) */
elemname = 'PACKAGE.MESSAGE'
DO tix = 1 TO _TN.0
  rc = FINDELEM(elemname,tix)
  IF rc = 0 THEN LEAVE /* last element found */
  PARSE VAR rc eix tix /* get <elem index> <next tag index> */
  CALL GETMSG
END
EXIT

FINDELEM: PROCEDURE EXPOSE _EX. _EX. _TN.
PARSE ARG elemname,tix
DO k = tix TO _EX.0
  eix = _EX.k /* node index */
  IF _TN.eix = elemname THEN RETURN eix k
END
RETURN 0

GETMSG: /* <tix> global */
DO j = 1 TO _EX.tix.0
  SAY _TN.eix _TD.eix /* display tag name, tag data */
  IF _TN.eix = 'PACKAGE.MESSAGE.MSGFILE' THEN DO
    fname = GETFILE(_TD.eix)
    IF fname = 0 THEN RETURN /* no filename */
    SAY 'FNAME:' fname /* display filename */
  END
  eix = eix+1
END
RETURN

GETFILE: PROCEDURE EXPOSE _EX. _EX. _TN. _TD.
PARSE ARG fno
elemname = 'PACKAGE.FILEREFERENCETABLE'
DO tix = 1 TO _TN.0
  rc = FINDELEM(elemname,tix)
  IF rc = 0 THEN LEAVE
  PARSE VAR rc eix tix
  fname = GETFNAME(fno)
  IF rc <> 0 THEN RETURN fname
END
RETURN 0 /* no filename found */

GETFNAME:
PARSE ARG fno
OK = 0 /* preset: no file number found */
DO j = 1 TO _EX.tix.0
  /* SAY eix _TN.eix _TD.eix */
  IF _TN.eix = 'PACKAGE.FILEREFERENCETABLE.FILE.FILENUMBER',
    & _TD.eix = fno THEN OK = 1 /* file no. found */
  IF _TN.eix = 'PACKAGE.FILEREFERENCETABLE.FILE.FILENAME',
    & OK = 1 THEN RETURN _TD.eix
  eix = eix+1
END
RETURN 0 /* no filename found */

```

Figure 11: Sample REXX for processing an error feedback file



Figure 12 shows the associated output.

```
PACKAGE.MESSAGE
PACKAGE.MESSAGE.MSGNUMBER IBM1316I E
PACKAGE.MESSAGE.MSGLINE 17
PACKAGE.MESSAGE.MSGFILE 0
FNAME: SCLM.DEV1.SOURCE(TESTMD2)
PACKAGE.MESSAGE.MSGTEXT END label is not a label on any open group.
PACKAGE.MESSAGE
PACKAGE.MESSAGE.MSGNUMBER IBM2619I W
PACKAGE.MESSAGE.MSGLINE 10
PACKAGE.MESSAGE.MSGFILE 0
FNAME: SCLM.DEV1.SOURCE(TESTMD2)
PACKAGE.MESSAGE.MSGTEXT The include file SCLM.TEST.SOURCE(FLM01IIN) contains no cross-reference
```

Figure 12: Associated error feedback file output

1.1.1.3 Example 3 - Use of Filters and Prefixes to Simplify Processing

This example performs the same processing as for Example 2 but uses filters and prefixes to simplify the processing. In this case, ERXXML is invoked twice (with PACKAGE.MESSAGE filter and the PACKAGE.FILEREFERENCETABLE filter, respectively, to fetch the message elements and file reference table element). To ensure unique namespaces for the generated variables, explicit prefixes (MSG. and FRT., respectively) are specified for each invocation.

Note: Although this technique can simplify the processing, because the XML file is reread, caution should be taken when a very large XML needs to be processed.

Figure 13 shows an example of an exec that uses filters and prefixes.

```
/* REXX */
dsn = 'SOFAR.XML(ERXXMLT2) '

filter = 'PACKAGE.MESSAGE'
_PFX = 'MSG.' /* set .MSG prefix */
frc = ERXXML("'"dsn"'",filter)
IF frc > 4 THEN EXIT frc /* parse error */

filter = 'PACKAGE.FILEREFERENCETABLE'
_PFX = 'FRT.'
frc = ERXXML("'"dsn"'",filter)
IF frc > 4 THEN EXIT frc /* parse error */

/* Display message elements */
DO tix1 = 1 TO MSG._TN.0
  SAY MSG._TN.tix1 MSG._TD.tix1 /* display tag name, tag data */
  IF MSG._TN.tix1 = 'PACKAGE.MESSAGE.MSGFILE' THEN SAY GETFILE()
END
EXIT

GETFILE: /* Scan file reference table elements for file number */
OK = 0
DO tix2 = 1 TO FRT._TN.0
  IF FRT._TN.tix2 = 'PACKAGE.FILEREFERENCETABLE.FILE.FILENAME',
  & OK = 1 THEN RETURN FRT._TD.tix2
  IF FRT._TN.tix2 = 'PACKAGE.FILEREFERENCETABLE.FILE.FILENUMBER',
  & FRT._TD.tix2 = MSG._TD.tix1 THEN OK = 1
END
RETURN 'no file'
```

Figure 13: Sample REXX showing the use of filters and prefixes



1.11.4 Example 4 - Process RDz Properties

To understand this sample you need some knowledge in IBM Rational Developer for System z (RDz).

This example shows how ERXXML can be used with native REXX functionality to process program options for RDz properties. This example lists the specified Link Options (the `USER_SPECIFIED_INSTRUCTIONS` tag in the `PROPERTY-GROUPS.PROPERTY-GROUP.CATEGORY-INSTANCE` element). For simplicity, it is assumed that the Property Value tag immediately follows the associated Property Name (`PROPERTY-GROUPS.PROPERTY-GROUP.CATEGORY-INSTANCE.PROPERTY.NAME = USER_SPECIFIED_INSTRUCTIONS`) tag (which normally will be the case). Figure 14 shows an example of a REXX exec to process RDz properties.

```

_CCSID = 1047 /* set explicit code page */
frc = ERXXML("'SOFAR.XML(ERXXML3P)')")
IF frc > 4 THEN DO
  SAY _MSG
  EXIT frc /* parse error */
END

/* Navigate (find element) */
tix = 0 /* first tag index */
tix1 = getCategory("LINK_OPTIONS")
IF tix1 = 0 THEN EXIT 8 /* category not found */
tix = tix1
tix2 = getCategory() /* get next category (= Link Options end) */
IF tix2 = 0 THEN tix2 = _TN.0 /* none = set end of document */
CALL getTagdata
EXIT 0

getCategory: PROCEDURE EXPOSE tix _TN. _TD.
PARSE ARG opt
tag = "PROPERTY-GROUPS.PROPERTY-GROUP.CATEGORY-INSTANCE"
sfx = ".CATEGORY"
ok = 0 /* default: no element found */
DO ix = (tix+1) TO _TN.0
  IF ok & tag||sfx = _TN.ix & opt = _TD.ix THEN RETURN ix

  IF _TN.ix = tag THEN DO /* element found */
    IF opt = '' THEN RETURN ix
    ok = 1 /* set element found */
  END
END
RETURN 0

getTagdata:
tag = "PROPERTY-GROUPS.PROPERTY-GROUP.CATEGORY-INSTANCE"
sfx = ".PROPERTY.NAME"
td = "USER_SPECIFIED_INSTRUCTIONS"
/* search for tag data */
DO i = tix1 TO tix2
  IF _TN.i <> tag||sfx THEN ITERATE
  IF _TD.i <> td THEN ITERATE
  /* property found, get associated value (next tag) */
  i = i+1 /* next tag index */
  temp = _TD.i /* tag data */
  eol = '0D'x /* end of line delimiter */
  DO WHILE temp <> ''
    PARSE VAR temp line (eol) temp
    SAY line
  END
  RETURN 0 /* OK */
END
EXIT 12 /* tag not found */

```

Figure 14: Sample REXX code to process RDz properties



2 ERXXML – Installation Instructions

2.1 Program Materials

ERXXML is distributed via Email.

To request a full license please contact SoforTe (see: <http://www.soforte.com/en/kontakt/index.php>).

You may download a free trial version of ERXXML from <http://www.soforte.com/en/downloads/index.php>.

You will receive the following components:

- Machine-readable material:
 - o ERXXML (load module or ERXXMLT when installing the trial version)
 - o REXXMLT (Sample REXX using ERXXML)
 - o Test XML File (To be used with REXXMLT)

- Publications:
 - o ERXXML (this document)

2.2 Requirements

Requirements to install and use ERXXML:

2.2.1 Machine Requirements

Any hardware environment can be used that supports the required software.

2.2.2 Software Requirements

z/OS V1.8 or later is required.

2.3 Installation Procedure

2.3.1 Upload the Software

After downloading the trial version or receiving the software via Email please unzip the file erxxml.zip. The zip file contains an xmit file erxxml\install.xmit.

Upload the erxxml\install.xmit file in binary format to a sequential dataset with LRECL 80 and BLKSIZE 3120.

First allocate the upload dataset on z/OS. You can do this by creating a dataset with the characteristics from the sample job below or by submitting the job below. If you choose to submit the following job you need to make the following updates:

- Add a job card and modify the parameters to meet your site's requirements before submitting.
- **hlq** will be the high level qualifier you choose to use for this dataset.
- (Optionally) add a valid VOLSER and/or change the UNIT parameter.



```
//ALLOC1 EXEC PGM=IEFBR14
//*
//FTPALLOC DD DSN=hlq.INSTALL.XMIT,
// DISP=NEW,CATLG,DELETE),
// DSORG=PS,
// RECFM=FB,
// LRECL=80,
// BLKSIZE=3120,
//* VOL=SER=vvvvvvv,
// UNIT=SYSALLDA,
// SPACE=(TRK,(45,15))
```

Figure 15: JCL to allocate the installation XMIT file

You can use FTP from a command prompt to upload the file. In the sample dialog shown below, commands or other information entered by the user are in bold, and the following values are assumed:

User enters	Description
mvsaddr	TCP/IP address or hostname of the z/OS system
tsouid	Your TSO user ID
tsopw	Your TSO password
hlq	High-level qualifier used for the data set you allocated in the job above

```
C:\>ftp mvsaddr
Connected to mvsaddr.
220-FTPD1 IBM FTP CS V1R10 at sofort.de, 12:03:32 on 2011-08-02.
220 Connection will close if idle for more than 5 minutes.
User (mvsaddr:(none)): tsouid
331 Send password please.
Password: tsopw
230 tsouid is logged on. Working directory is "tsouid.".
ftp> cd ..
250 " " is the working directory name prefix.
ftp> cd hlq
250 "hlq." is the working directory name prefix.
ftp> binary
200 Representation type is Image
ftp> put d:\erxml\install.xmit
200 Port request OK.
125 Storing data set hlq.INSTALL.XMIT
250 Transfer completed successfully.
FTP: nnnnnnnn bytes sent in nn,nn seconds nn,nn KB/sec
```

Figure 16: FTP sample to upload the installation package

2.3.2 Receive the Software

Using the TSO RECEIVE command will create the partitioned dataset from the compressed XMIT file. Further on we assume that tsouid.ERXML is the installation prefix.

```
Enter TSO or Workstation commands below:

===> receive indataset('hlq.INSTALL.XMIT')
```



When prompted enter:

```
INMR906A Enter restore parameters or 'DELETE' or 'END' +
restore dataset('tsouid.ERXXML.XMIT')
```

You should receive the following messages:

```
COPY INDD=((SYS00031,R)),OUTDD=SYS00030
IEB1013I COPYING FROM PDSU INDD=SYS00031 VOL=USR001 DSN=SYS11214.T134430.RA00
.SOFUP.R0100339
IEB1014I          TO PDS  OUTDD=SYS00030 VOL=USR002 DSN=SOUP.ERXXML.XMIT
IEB167I FOLLOWING MEMBER(S) LOADED FROM INPUT DATA SET REFERENCED BY SYS00031
IEB154I LOAD      HAS BEEN SUCCESSFULLY LOADED
IEB154I REXX      HAS BEEN SUCCESSFULLY LOADED
IEB154I XML       HAS BEEN SUCCESSFULLY LOADED
IEB1098I 3 OF 3 MEMBERS LOADED FROM INPUT DATA SET REFERENCED BY SYS00031
IEB144I THERE ARE 26 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY SYS00030
IEB149I THERE ARE 5 UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
IEB147I END OF JOB - 0 WAS HIGHEST SEVERITY CODE
INMR001I Restore successful to dataset 'SOUP.ERXXML.XMIT'
```

Figure 17: Receive the installation software

The result is a partitioned dataset with following members:

```
BROWSE          SOFUP.ERXXML.XMIT                               Volume: PROJ00
Command ===>                                           Scroll: CSR
      Name      Prompt      Size  Created      Changed      ID
-----
      LOAD
      REXX
      XML
      **End**
```

Each member is a compressed partitioned dataset:

Member	Description
LOAD	Contains the ERXXML load module
REXX	REXX sample
XML	Sample XML data

You may use the TSO RECEIVE command again for each member to create all partitioned datasets:

Member	Receive to PDS
LOAD	tsouid.ERXXML.LOADLIB
REXX	tsouid.ERXXML.REXX
XML	tsouid.ERXXML.XML

As an alternative you may use a batch job to receive all partitioned datasets:

```
//STEP1 EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
RECEIVE INDSNAME('tsouid.ERXXML.XMIT(LOAD)') NONAMES
DATASET('tsouid.ERXXML.LOADLIB')
RECEIVE INDSNAME('tsouid.ERXXML.XMIT(REXX)') NONAMES
DATASET('tsouid.ERXXML.REXX')
RECEIVE INDSNAME('tsouid.ERXXML.XMIT(XML)') NONAMES
DATASET('tsouid.ERXXML.XML')
/*
```

Figure 18: Receive the ERXXML PDS datasets



Verify that each RECEIVE has been successfully processed, e.g. the SYSTSPRT output from the batch job should show following messages ending with “Restore successful”:

```
RECEIVE INDSNAME('tsouid.ERXXML.XMIT(LOAD)') NONAMES
INMR901I Dataset SOFUP.DDNAME.INFILE from SOFUP on NODENAME
INMR154I The incoming data set is a 'PROGRAM LIBRARY'.
INMR906A Enter restore parameters or 'DELETE' or 'END' +
INMR908A The input file attributes are: DSORG=PARTITIONED, RECFM=U, BLKSIZE=4096
INMR909A You may enter DSNAME, SPACE, UNIT, VOL, OLD/NEW, or RESTORE/COPY/DELETE
INMR001I Restore successful to dataset 'tsouid.ERXXML.LOAD'
```

2.3.1 Installation Verification

Verify that you have successfully received all ERXXML components:

- The dataset tsouid.ERXXML.LOAD contains the member ERXXML (or ERXXMLT if you have installed the trial version).
- The dataset tsouid.ERXXML.REXX contains the REXX test procedure REXXMLT.
- The dataset tsouid.ERXXML.XML contains the XML test member ERXXML.

Execute the REXX test procedure REXXMLT e.g. from the ISPF command shell (mostly ISPF menu 6). Enter:

- EXEC 'tsouid.ERXXML.REXX(REXXMLT)'

If this REXX runs successfully you should receive the following messages on your terminal:

```
(c)SoforTe ERXXML.1.5: Oct 12 2011-19:23:55 - trial version
Default CCSID:1047 used
Rc from XML-Parse: 0
*****
PLI Compiler message:
Dataset: SCLM.DEV1.SOURCE
Member: TESTMD2
Line: 17
Msg No: IBM1316I E
Message: END label is not a label on any open group.
*****

*****
PLI Compiler message:
Dataset: SCLM.DEV1.SOURCE
Member: TESTMD2
Line: 10
Msg No: IBM2619I W
Message: The include file SCLM.TEST.SOURCE(TESTINC) contains no cross-reference
*****
```

If you have used other installation dataset names you have to pass the file names to this REXX procedure:

- EXEC 'tsouid.ERXXML.REXX(REXXMLT)' 'tsouid.ERXXML.XML(ERXXML) tsouid.ERXXML.LOAD'