RPG XML SUITE

connecting your iSeries to the world

Documentation for
RPG-XML Suite version 1.41

www.rpg-xml.com

QUESTIONS ABOUT THIS MANUAL CONTACT SUPPORT@KRENGELTECH.COM

TABLE OF CONTENTS

1 Installation

Here are the modified RPG-XML Suite installation instructions for training manual “XML Web Services for RPG Programmers”.

```
-------------------------------------------------------------------------------------------------------
                    RPG-XML Suite (RXS) v1.41
                         www.rpg-xml.com

                product of Krengel Technology Inc.
-------------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------------------------------
                        Installation
-------------------------------------------------------------------------------------------------------

!!!!NOTE!!!!
Read the license agreement titled "License Agreement.pdf" before installing
this software. By installing you declare that you agree to abide by the
license agreement.

1. Unzip the downloaded file to c:\temp (or the directory of your choice).

2. Issue the AS/400 command:
    CRTSAVF FILE(QGPL/RXS) AUT(*ALL)

3. FTP the file rxs.savf from your PC to the AS/400 in BINARY mode
    into the save file RXS in library QGPL.

        3.01 - open a DOS prompt (Start -> Run -> enter 'cmd' and hit enter)
        3.02 - type the following into the DOS prompt
        3.03 - ftp 192.168.0.11 (replace the IP address with that of your iSeries)
        3.04 - when prompted enter profile and password
        3.05 - binary
        3.06 - lcd c:\temp (where c:\temp is the location of the rxs.savf)
        3.07 - quote site namefmt 0
        3.08 - cd qgpl
        3.09 - put rxs.savf rxs.savf
        3.10 - quit


4. Issue the AS/400 commands.
    The value 'RXS' is used to denote where the base install of RPG-XML Suite should
    reside. Note that 81 is the default port your RXS runs under in Apache. Change it
    to meet your needs. The default of 81 should be fine 99% of the time.
    The value 'XML4RPG' will be the name of your Apache web server instance and will
    also create a library named XML4RPG for your development environment.

    CRTLIB LIB(RXS)

    RSTOBJ OBJ(*ALL) SAVLIB(RXS) DEV(*SAVF) SAVF(QGPL/RXS)

    CALL RXS/INSTALL PARM('RXS' '81' 'XML4RPG')

    ADDLIBLE XML4RPG



5. Registration
    A special site has been setup specifically for the "XML Web Services for RPG Programmers"
    training course.
    Please go to http://xml4rpg.com to get an evaluation license key emailed to you.

    Once you have recieved the registration key go ahead and enter the following
    on the command line:

    CALL RXS/REGRXSBASE PARM('<<insert key that was emailed to you>>')
```

6. Start the RPG-XML Suite Apache HTTP Server instance by typing in the below
   command. Note the XML4RPG instance is configured to run on port 81. If port 81 is already
   in use then the Apache config file will need to be changed using this command:
   EDTF '/www/xml4rpg/conf/httpd.conf'

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(XML4RPG)
```

7. Open your internet browser and enter the following (note, change the IP address
   to your iSeries IP address):

   http://192.168.0.11:81/xml4rpg/rxs1

   You should get the following result:

```
<output myAttr="static value">

I love home improvement plumbing. Especially when it leaks after you turn on the water!

</output>
```

8. Now it is time to install the example programs for the training course.
   Issue the AS/400 command:

```
CRTSAVF FILE(QGPL/XML4RPGSRC) AUT(*ALL)
```

9. FTP the file xml4rpgsrc.savf from your PC to the AS/400 in BINARY mode
   into the save file XML4RPGSRC in library QGPL.

```
    3.01 - open a DOS prompt (Start -> Run -> enter 'cmd' and hit enter)
    3.02 - type the following into the DOS prompt
    3.03 - ftp 192.168.0.11 (replace the IP address with that of your iSeries)
    3.04 - when prompted enter profile and password
    3.05 - binary
    3.06 - lcd c:\temp (where c:\temp is the location of the xml4rpgsrc.savf)
    3.07 - quote site namefmt 0
    3.08 - cd qgpl
    3.09 - put xml4rpgsrc.savf xml4rpgsrc.savf
    3.10 - quit
```

10. Restore the XML4RPGSRC library with the following command:

```
RSTLIB SAVLIB(XML4RPGSRC) DEV(*SAVF) SAVF(QGPL/XML4RPGSRC) RSTLIB(XML4RPGSRC)
```

11. Copy templates from source member to IFS stream file.

```
CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/CALCPRC.MBR')
          TOSTMF('/www/xml4rpg/templates/calcprc.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/CTADDPO.MBR')
          TOSTMF('/www/xml4rpg/templates/ctaddpo.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/CTSOAP.MBR')
          TOSTMF('/www/xml4rpg/templates/ctsoap.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/PING.MBR')
          TOSTMF('/www/xml4rpg/templates/ping.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/POREQ.MBR')
          TOSTMF('/www/xml4rpg/templates/purchord_req.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/PORSP.MBR')
          TOSTMF('/www/xml4rpg/templates/purchord_rsp.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/PORSP4.MBR')
          TOSTMF('/www/xml4rpg/templates/purchord4_rsp.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)

CPYTOSTMF FROMMBR('/QSYS.LIB/XML4RPGSRC.LIB/QTXT.FILE/UPSADRVLD.MBR')
          TOSTMF('/www/xml4rpg/templates/ups_adr_vld.tpl')
          STMFOPT(*REPLACE) STMFCODPAG(819)
```

12. Copy the source physical files QRPGLESRC and QDDSSRC from library XML4RPGSRC to
    library XML4RPG.

```
CRTDUPOBJ OBJ(QRPGLESRC)
          FROMLIB(XML4RPGSRC) OBJTYPE(*FILE)
          TOLIB(XML4RPG) NEWOBJ(QRPGLESRC) DATA(*YES)

CRTDUPOBJ OBJ(QDDSSRC)
          FROMLIB(XML4RPGSRC) OBJTYPE(*FILE)
          TOLIB(XML4RPG) NEWOBJ(QDDSSRC) DATA(*YES)
```

Requirements

- RPG-XML Suite was created on an iSeries running V5R3 of OS/400 but is compiled back to V5R1.

- SSL functionality requires cryptographic support installed on your iSeries.
-- 5722AC3   *BASE   Crypto Access Provider 128-bit

- Offering XML web services with RXS requires Apache which is installed on 99% of all machines
  we come in contact with.
-- 5722DG1   *BASE   IBM HTTP Server

Registration

To get a permanent license please go to www.rpg-xml.com and navigate to the Contact Us
page or email sales@krengeltech.com.

To get an evaluation license please go to www.xml4rpg.com.

2 Configuration

Portions of RPG-XML Suite are configurable and those values are stored in physical file RXSCFG which can be found in each RXS library on your system. When you first install RPG-XML Suite you will have two libraries – RXS and XML4RPG. RXS is the "pristine" library that shouldn't be altered, and XML4RPG is the library where you should do your development. Each of these libraries have their own copy of RXSCFG, but only XML4RPG has data in it.

Below is the definition of physical file RXSCFG.

```
A          R RXSCFGR                    TEXT('RXS Config File')
A            TPLDIR        256A         COLHDG('Template Dir')
A            TRANSDIR      256A         COLHDG('Transaction Dir')
A            DEBUG           1A         COLHDG('Debug')
A            MSNGDTA        30A         COLHDG('Missing Data')
A            SECBEG         20A         COLHDG('Section Begin')
A            SECEND         20A         COLHDG('Section End')
A            VARBEG         20A         COLHDG('Variable Begin')
A            VAREND         20A         COLHDG('Variable End')
A            ELEMBEG        10A         COLHDG('Element Begin')
A            ELEMCNT        10A         COLHDG('Element Content')
A            ELEMEND        10A         COLHDG('Element End')
A            ELEMATTR       10A         COLHDG('Element Attribute')
```

Field Descriptions:
TPLDIR – Specify the default IFS location for template files that are used to compose XML with the RPG-XML Suite Template Engine. At install time this is defaulted to /www/xml4rpg/templates/.

TRANSDIR – Specify the default IFS location for the XML transaction files. Transaction files are created when you choose to use IFS stream files for your medium of doing web services. For example, when composing an XML document you can specify RXS_STMF as the second parameter on the RXS_initTplEng API vs. RXS_STDOUT or RXS_VAR. XML files can also be created when using API RXS_readToFile when offering a web service, and lastly XML files can be created when you specify the response for API RXS_getUri should be an IFS file. At install time this is defaulted to /www/xml4rpg/trans/

DEBUG – Specify whether debugging should be turned on during RPG-XML Suite runtime processing. Valid values are T (true) or F (false). If T is specified then you will find information trickled into the job log to aid in debugging an issue.

MSNGDTA – When composing XML using the RPG-XML Suite Template Engine you will be updating variables in the .tpl file using RXS_updVar, and it is possible to miss a variable. The value in this field will be used to replace all variables within a section that didn't have a value specified for them. At install time this is defaulted to ***.

SECBEG – This is the Template Engine default for section begin delimiters. The default as of v1.4 is two colons (::). Before v1.4 it was /$ which caused CCSID conversion issues in some countries so it was defaulted to more "safe" characters. Two colons are also easier to type which is a solid benefit when having to repeat those keystrokes many times during template composition.

SECEND – This is the Template Engine default for section end delimiters. The default as of v1.4 is blanks.

VARBEG – This is the Template Engine default for variable begin delimiters. The default value as of v1.4 is period colon (.:). Before v1.4 it was /% which caused CCSID conversion issues in some countries so it was defaulted to more "safe" characters. A period colon are also easier to type which is a solid benefit when having to repeat those keystrokes many times during template composition.

VAREND - This is the Template Engine default for variable end delimiters. The default value as of v1.4 is colon period (:.). Before v1.4 it was %/ which caused CCSID conversion issues in some countries so it was defaulted to more "safe" characters. A colon period are also easier to type which is a solid benefit when having to repeat those keystrokes many times during template composition.

ELEMBEG – This is the element begin event default used during parsing of XML documents. At install time this value is defaulted to a greater than sign (>).

ELEMCNT – This is the element content event default used during parsing of XML documents. At install time this value is defaulted to a forward slash (/).

ELEMEND – This is the element end event default used during parsing of XML documents. At install time this value is defaulted to a forward slash and greater than sign (/>).

ELEMATTR – This is the element attribute event default used during parsing of XML documents. At install time this value is defaulted to an at sign (@).

3 Set Up My Own Web Service Environment

Different uses suggest the need for multiple separate environments. For instance, each developer might have a different "play area" so they can do business without worrying about getting in the way of other programmers. Perhaps each business unit or application used to provide a web service needs separation from the others. To facilitate creating new environments, the RPG-XML Suite provides a command to call. Issuing the following command sets up a new environment named XML4RPG2 running on port 82 based on library RXS:

```
CALL RXS/NEWENV PARM('RXS' '82' 'XML4RPG2')
```

This will create a new library named XML4RPG2 and a new Apache server instance named XML4RPG2. Note that the name is limited to 10 characters (to facilitate the library name limitation.) You will now have your own RXS and EXAMPLE source physical files along with the example programs that come with the base install. To test the new environments web service run the following command.

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(XML4RPG2)
```

Then point your browser to http://172.29.134.41:82/XML4RPG2/rxs1 making sure to place your iSeries IP address in the URL instead of 172.29.134.41.

4 Code Generators

RPG-XML Suite code generators are provided to facilitate writing code faster, because the code writing for composing and parsing XML can be laborious and often times similar from program to program. There are currently two different code generators included with RPG-XML Suite – BLDPRS (Build RPG Parse Subprocedure) and BLDTPL (Build XML Template). These code generators are callable from the command line and can be prompted for parameter assistance.

4.1 BLDPRS (Build RPG Parse Subprocedure)

The BLDPRS command is used to aid in writing RPG-XML Suite parsing code. In the examples included with RPG-XML Suite (i.e. XML4RPG/EXAMPLE,*) there are many "handler" or parsing subprocedures that take an XML file and parse it for the data contents. After writing a few of these parsing subprocedures you will find that much of keying is repetitive with just a few things changing from program to program. This brings about a perfect case for a code generator that will take care of much of the parsing code.

BLDPRS usage can best be explained with an example. The first thing needed is an XML document residing in the IFS. Type the following on the command line to create an XML file named /home/bldprs001.xml in the IFS.

```
QSH CMD('touch -C 819 /home/bldprs001.xml')
```

The file now exists but without content. To add content we will use the EDTF command and copy/paste the following XML.

```
EDTF '/home/bldprs001.xml'
```

```
<PostAdr residential="true">
  <name title="Mr.">
    <first>Aaron</first>
    <last>Bartell</last>
  </name>
  <street>123 Center Rd</street>
  <cty>Mankato</cty>
  <state>MN</state>
  <zip>56001</zip>
  <phone>123-123-1234</phone>
  <phone>321-321-4321</phone>
</PostAdr>
```

While in the Edit File editor select F2 to save the document changes. Now it is time to invoke the BLDPRS command to generate the parsing code. You will need to provide the library, source physical file, and source member of where you would like the generated code to be place and also where the XML document resides in the IFS that should be used. The last parameter, BASEENV, can be omitted but serves a code readability purpose. BASEENV allows you to specify the beginning portion of an XPath that is repeated for every element, or attribute, in the XML document. In the below example /PostAdr is specified for the BASEENV parameter. This value will be applied to an RPG variable named baseEnv and baseEnv will then be used in the WHEN clauses to make the statements shorter.

```
BLDPRS
  SRCLIB(XML4RPG)
  SRCPF(QSOURCE)
  SRCMBR(BLDPRS001)
  IFSXMLLOC('/home/bldprs001.xml')
```

```
BASEENV('/PostAdr')
```

The following is the contents of source member XML4RPG/EXAMPLE,BLDPRS001 after running the above command. Note that some of the code was omitted for brevity sake.

```
D allHandler      pr
D  pType                              value like(RXS_Type)
D  pXPath                             value like(RXS_XPath)
D  pData                              value like(RXS_XmlData)
D  pDataLen                           value like(RXS_Length)


 //-------------------------------------------------------------------------
 // @Author:
 // @Created:
 // @Desc:
 //-------------------------------------------------------------------------
P allHandler      b
D allHandler      pi
D  pType                              value like(RXS_Type)
D  pXPath                             value like(RXS_XPath)
D  pData                              value like(RXS_XmlData)
D  pDataLen                           value like(RXS_Length)

D chgMe           s                   like(RXS_XmlData)
D baseEnv         s             70a   varying
 /free

  baseEnv = '/PostAdr';

  select;

  when pXPath = baseEnv +'>';
    chgMe = pData;
  when pXPath = baseEnv +'@residential';
    chgMe = pData;
  when pXPath = baseEnv +'/';
    chgMe = pData;
  when pXPath = baseEnv +'/name>';
    chgMe = pData;
  when pXPath = baseEnv +'/name@title';
    chgMe = pData;
  when pXPath = baseEnv +'/name/';
    chgMe = pData;
  when pXPath = baseEnv +'/name/first>';
    chgMe = pData;
  when pXPath = baseEnv +'/name/first/';
    chgMe = pData;
  when pXPath = baseEnv +'/name/first/>';
    chgMe = pData;
  when pXPath = baseEnv +'/name/last>';
    chgMe = pData;
  when pXPath = baseEnv +'/name/last/';
    chgMe = pData;
  when pXPath = baseEnv +'/name/last/>';
    chgMe = pData;
  when pXPath = baseEnv +'/name/>';
    chgMe = pData;
  when pXPath = baseEnv +'/street>';
    chgMe = pData;
  when pXPath = baseEnv +'/street/';
    chgMe = pData;
  when pXPath = baseEnv +'/street/>';
    chgMe = pData;
   . . .
  when pXPath = baseEnv +'/>';
    chgMe = pData;
  endsl;

 /end-free
P                 e
```

4.2 BLDTPL (Build XML Template)

The BLDTPL command aids the composing of RPG-XML Suite template (*.tpl) files which are used in conjunction with the Template Engine to compose XML documents. Composing the necessary XML for the *.tpl files can become quite laborious and typing

errors can cause wasted time debugging trying to figure why your program is not working as expected. BLDTPL aims to alleviate those problems and make you more productive.

BLDTPL usage can best be explained with an example. The first thing needed is an XML document residing in the IFS. Type the following on the command line to create an XML file named /home/bldtpl001.xml in the IFS.

```
QSH CMD('touch -C 819 /home/bldtpl001.xml')
```

The file now exists but without content. To add content we will use the EDTF command and copy/paste the following XML into the EDTF editor.

```
EDTF '/home/bldtpl001.xml'
```

```
<PostAdr residential="true">
  <name title="Mr.">
    <first>Aaron</first>
    <last>Bartell</last>
  </name>
  <street>123 Center Rd</street>
  <cty>Mankato</cty>
  <state>MN</state>
  <zip>56001</zip>
  <phone>123-123-1234</phone>
  <phone>321-321-4321</phone>
</PostAdr>
```

While in the Edit File editor select F2 to save the document changes. Now it is time to invoke the BLDTPL command to generate a *.tpl file that will facilitate composing the PostAdr XML document.

```
BLDTPL
  IFSXMLLOC('/home/bldtpl001.xml')
  IFSTPLLOC('/www/xml4RPG/templates/bldtpl001.tpl')
  INDENT(2)
```

The IFSXMLLOC parameter is the location of the XML file that was created in the above steps. The IFSTPLLOC is where we want the new bldtpl001.tpl file to be located – place it in the location of all other RPG-XML Suite templates. The last parameter, INDENT, allows you to specify how the template's XML should be indented. Note that indenting is purely for ease of editing.

See below for the resulting template file located at /www/xml4rpg/templates/bldtpl001.tpl (to view . Note that all elements and attributes have had their values replaced with variable place holders (i.e. .:variable:.) respective to the element or attributes name. This file is now ready to be used by the RPG-XML Suite Template Engine. Note that you would reference the file bldtpl001.tpl on the RXS_loadTpl subprocedure (i.e. RXS_loadTpl('bldtpl001.tpl')).

```
<PostAdr residential=".:residential:.">
  <name title=".:title:.">
    <first>.:first:.</first>
    <last>.:last:.</last>
  </name>
  <street>.:street:.</street>
  <cty>.:cty:.</cty>
  <state>.:state:.</state>
  <zip>.:zip:.</zip>
  <phone>.:phone:.</phone>
</PostAdr>
```

One last thing to note is that in file bldtpl001.xml there were two <phone> elements. Whenever the BLDTPL command comes across the same XPath (i.e. '/PostAdr/phone' in this case) it will not process it a second time. It will instead assume the <phone> tag will most likely be in it's own section (maybe named ::phone). By putting the <phone> tag in its own section you have modularized the template file and now many <phone> tags can be easily written out in the transaction file by wrapping it with a DOW loop.

Once the BLDTPL process is complete you will need to add your sections. Below is an example of how bldtpl001.tpl has been modified with sections.

```
::PostAdr_beg
<PostAdr residential=".:residential:.">
  <name title=".:title:.">
    <first>.:first:.</first>
    <last>.:last:.</last>
  </name>
  <street>.:street:.</street>
  <cty>.:cty:.</cty>
  <state>.:state:.</state>
  <zip>.:zip:.</zip>
::phone
  <phone>.:phone:.</phone>
::PostAdr_end
</PostAdr>
```

5 Complete API Listing

5.1 Parsing APIs

5.1.1 RXS_parse

Description: Call this subprocedure to parse either an XML document residing in the IFS or the contents of an RPG variable that contains XML.

Prototype:

```
D RXS_parse       pr
D  pFilePathOrData...
D                              65535a   value varying
D  pType                          10a   value
D  pErrProcPtr                     *    procptr value
D  pElemBegVal                    10a   value options(*nopass)
D  pElemContentVal...
D                                 10a   value options(*nopass)
D  pElemEndVal                    10a   value options(*nopass)
D  pAttrVal                       10a   value options(*nopass)
```

Parameters:

pFilePathOrData – Used to specify a file in the IFS or actual XML residing in an RPG variable. If an IFS file is specified it must either be fully qualified (i.e. /home/aaron/mydoc.xml) or must reside in the default transaction directory (i.e. /www/rxs/trans.) If it resides in the default transaction directory then you can just specify it as 'mydoc.xml.'

pType - This parameter can either have RXS_STMF or RXS_VAR specified as it's value. Use RXS_STMF if the value passed in pFilePathOrData is a path to an IFS file. Use RXS_VAR if the value passed in pFilePathOrData contains XML.

pErrProcPtr - This parameter tells the XML parser which procedure in your program to call if the parser encounters an error. It is of type PROCPTR and can be obtained by using the %paddr() BIF (e.g. %paddr(myErrHandler)). You can name the local subprocedure for capturing errors anything you want, but it must specify these parameters in this order with these data types:

```
D errHandler      PI
D  pCurLine                     10i 0 value
D  pCurCol                      10i 0 value
D  pErrStr                    1024a   value varying
```

pElemBegVal – Override the default element begin value of '>' with the value of your choice. Recommend to leave at default unless you are experiencing codepage issues.

pElemContentVal – Override the default element content value of '/' with the value of your choice. Recommend to leave at default unless you are experiencing codepage issues.

pElemEndVal – Override the default element begin value of '/>' with the value of your choice. Recommend to leave at default unless you are experiencing codepage issues.

pAttrVal – Override the default element begin value of '@' with the value of your choice. Recommend to leave at default unless you are experiencing codepage issues.

Notes:

Ordinarily you will set up or "register" handler pointers using one or all of the following subprocedures before calling RXS_parse:

RXS_addHandler
RXS_allElemBeginHandler
RXS_allElemContentHandler
RXS_allElemEndHandler
RXS_allAttrHandler

The RXS_parse function will run even if you do not register any handlers. However, if you do not specify any handlers, the parser will only notify the program when it finds XML errors. It will not return notice of any other events.

Every locally defined subprocedure registered with the parser as an event handler must have the four parameters listed in the following example:

```
D Handler         pi
D  pType                        value like(RXS_Type)
D  pXPath                       value like(RXS_XPath)
D  pData                        value like(RXS_XmlData)
D  pDataLen                     value like(RXS_Length)
```

5.1.2 RXS_addHandler

Description: Call this subprocedure before RXS_parse to specify an event requiring notification. The event can be the beginning of an element, the content of an element, the end of an element or the attribute of an element.

Prototype:

```
D RXS_addHandler  pr
D  pXPath                       like(RXS_XPath) value
D  pHandler                  *  procPtr value
```

Parameters:

pXPath - The fully qualified path of the specified event.

Example attribute Xpath: "/PostAdr/name@title" - note the "@" before *title*.
Example element begin Xpath: "/PostAdr/name>" - note the ">" at the end.
Example element content Xpath: "/PostAdr/name/first/" - note the "/" at the end.
Example element end Xpath: "/PostAdr/name/>" – note the "/>" at the end.

pHandler - The address of the local subprocedure in your program for the parser to call when it encounters the Xpath. Use the %paddr RPG BIF to obtain the address of the local subprocedure (i.e. %paddr(myHandler)).

Tip:

Using RXS_addHandler for a specific event (i.e. /PostAdr/name@title) in a program will generate an event to the handler you specify. If you also specified RXS_allAttrHandler so your program were notified of all attributes, you would be better off removing the RXS_addHandler that was specific to /PostAdr/name@title as only one event will be generated and RXS_allAttrHandler would cover the /PostAdr/name@title event. It does not matter what the event type—RXS_ELEMBEGIN, RXS_ELEMCONTENT, RXS_ELEMEND —if the program registers an "all handler" for that event type, the parser will send the event to that handler

5.1.3 RXS_allElemContentHandler

Description: Call this subprocedure before RXS_parse to tell the parser to notify a specific subprocedure of your program every time it encounters the content for ***any*** element in the document (e.g. <element>***I am the content***</element>). When the parser encounters element content, it will send the content value to the handler specified in pHandler.

Prototype:

```
D RXS_allElemContentHandler...
D                 pr
D  pHandler                     *   value procptr
```

Parameters:

pHandler - The address of the local subprocedure in your program that the parser should call when it encounters ***any*** element's content. Use the %paddr RPG BIF to obtain the address of the local subprocedure (e.g. %paddr(myHandler)).

Tip:

Using this approach saves coding time when a program will retrieve a majority of the element content.

5.1.4 RXS_allElemEndHandler

Description: Call this subprocedure before RXS_parse to tell the parser to notify a specific subprocedure of your program every time it encounters the end of an element in a document (i.e. <element>I am the content***</element>***)

Prototype:

```
D RXS_allElemEndHandler...
D                 pr
D  pHandler                     *   value procptr
```

Parameters:

pHandler - The address of the local subprocedure in your program that should be called when ***any*** element's ending tag is encountered. Use the %paddr RPG BIF to obtain the address of the local subprocedure (e.g. %paddr(myHandler)).

Tip:

Using this approach saves coding time when a program will retrieve a majority of the end element events.

5.1.5 RXS_allAttrHandler

Description: Call this subprocedure before RXS_parse to tell the parser to notify a specific subprocedure of your program every time it encounters an attribute within an element (e.g. <element ***myAttr***="***Attr content***">I am the content</element>.) When the parser encounters the attribute, the specified handler will receive the value of the attribute (in this case "Attr content").

Prototype:

```
D RXS_allAttrHandler...
D                 pr
D  pHandler                     *   value procptr
```

Parameters:

pHandler - The address of the local subprocedure in your program for the parser to call when it encounters any attribute of **any** element. Use the %paddr RPG BIF to obtain the address of the local subprocedure (e.g. %paddr(myHandler)).

Tip:

Using this approach saves coding time when a program will retrieve a majority of the attributes values in a document. Also, the parser provides the data value of an attribute (i.e. attrname="I am attr data value") in the pData parameter of the handler when the attribute event takes place. This is different than the way the parser handles elements. Elements have separate events for their content (i.e. <element>element content</element> will generate three events – one for the beginning of the element, one for the element content, and one for the end of the element) .

5.1.6 RXS_allElemBeginHandler

Description: Call this subprocedure before RXS_parse to tell the parser to notify a specific subprocedure of your program every time it encounters a beginning element (i.e. **<element>**I am the content</element>.)

Prototype:

```
D RXS_allElemBeginHandler...
D                 pr
D  pHandler                     *   value procptr
```

Parameters:

pHandler - The address of the local subprocedure in your program for the parser to call when it encounters **any** element's begin tag. Use the %paddr RPG BIF to obtain the address of the local subprocedure (e.g. %paddr(myHandler).)

Tip:

Using this approach saves coding time when a program will retrieve a majority of the begin element events.

5.1.7 RXS_setParseEnc

Description: Call this subprocedure before RXS_parse to tell the parser what encoding to use while parsing the character data in elements and attributes. By default the parser will look at the first few characters of the file to attempt to determine what encoding should be used. By default, and more often than not, UTF-8 is used. That can cause problems if the data was actually obtained using CCSID 819 (i.e. encoding ISO88591). If you are having trouble parsing characters like Ü then your data is probably being returned as ISO88591 vs. UTF-8 (as many XML declarations incorrectly state).

Prototype:

```
D RXS_setParseEnc...
D                 pr
D  pPrsEnc                      25a   const
```

Parameters:
pPrsEnc – Specify the parse encoding that the parser should use. Valid values are RXS_UTF8, RXS_UTF16, RXS_ISO88591, RXS_USASCII. All these constants can be found in copy book RXSCP.

Tip:
An excellent reference for code pages can be found here: http://www.tachyonsoft.com/cpindex.htm

5.1.8 RXS_ignElemNamSpc

Description: Call this subprocedure before RXS_parse to tell the parser of an XML namespace to ignore while parsing. Ignore simply means it will not add the namespace to the XPath. This is especially useful when parsing SOAP envelopes as the namespace can vary depending on what language created the document. For example, .NET traditionally uses a SOAP envelope like the following: <soapenv:Envelope>. Java on the other hand often generates a SOAP envelope like the following: <SOAP-ENV:Envelope>. Both are entirely valid but cause problems when parsing because in your allHandler subprocedure it is looking for a specific XPath (i.e. /soapenv:Envelope/soapenv:Body/myelement). By telling the parser to ignore name spaces for the Envelope and Body SOAP tags the XPath will instead look like the following: /Envelope/Body/myelement. A code sample showing how to make use of

RXS_ignElemNamSpc can be found in the RPG-XML Suite installation library in member GETURI2 (i.e. XML4RPG/EXAMPLE,GETURI2)

Prototype:

```
D RXS_ignElemNamSpc...
D                 pr
D  pXPath                       const like(RXS_XPath)
```

Parameters:
pXPath – Specify the XPath to the element where you want name spaces ignored. When specifying the XPath to an element do NOT include anything at the end of the string (i.e. '>' or '/' or '/>' which are used to denote begin element event, content element event, and end element event in other parts of RPG-XML Suite). Instead just supply the path excluding the name space you are looking to ignore.

Example XML used to show correct and incorrect usages of the RXS_ignElemNamSpc API:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <FahrenheitToCelsius xmlns="http://tempuri.org/">
      <Fahrenheit>25</Fahrenheit>
    </FahrenheitToCelsius>
  </soap:Body>
</soap:Envelope>
```

Correct examples:
RXS_ignElemNamSpc('/Envelope')
RXS_ignElemNamSpc('/Envelope/Body')
RXS_ignElemNamSpc('/Envelope/Body/FahrenheitToCelsius/Fahrenheit/')

Incorrect examples:
RXS_ignElemNamSpc('/Envelope>')
RXS_ignElemNamSpc('/Envelope/')
RXS_ignElemNamSpc('/Envelope/>')

RXS_ignElemNamSpc('/soap:Envelope/soap:Body')
RXS_ignElemNamSpc('/Envelope/soap:Body')
RXS_ignElemNamSpc('/soap:Envelope/Body')

5.1.9 RXS_soapDecode

Description: Call this subprocedure to take XML from it's encoded form (i.e. <tagname>content</tagname>) to actual XML that can be parsed (i.e. <tagname>content</tagname>). The reason this subprocedure is named soapDecode speaks to why this API is needed in the first place. Many programmers in .NET and Java environments have their code generated which inherently means the encoding style usually defaults to RPC (Remote Procedure Call) vs. Document Literal. Note that RXS_soapDecode can also be used to *encode* values if your program is composing the XML to be sent to an RPC style web service. In the end RXS_soapDecode is simply an easy to use find/replace mechanism to use on files and variable data. See example(s) below.

Prototype:

```
D RXS_soapDecode  pr
D  pFileOrData                65535a   varying
D  pType                         10a   value
D  pFrom                         10a   dim(10) varying options(*nopass)
D  pTo                           10a   dim(10) varying options(*nopass)
```

Parameters:
pFilePathOrData – Used to specify a file in the IFS or actual XML residing in an RPG variable. If an IFS file is specified it must either be fully qualified (i.e. /home/aaron/mydoc.xml) or must reside in the default transaction directory (i.e. /www/xml4rpg/trans.) If it resides in the default transaction directory then you can just specify it as 'mydoc.xml.'

pType - This parameter can either have RXS_STMF or RXS_VAR specified as it's value. Use RXS_STMF if the value passed in pFilePathOrData is a path to an IFS file. Use RXS_VAR if the value passed in pFilePathOrData contains XML.

pFrom – Used to specify a list of *from* values in an array that correspond to the *to* values in pTo. Not a required parameter. If not specified the default set of decoding characters are used as specified in the notes below.

pTo - Used to specify a list of *to* values in an array that correspond to the *from* values in pFrom. Not a required parameter. If not specified the default set of decoding characters are used as specified in the notes below.

Notes:

The following are the default *from/to* array values if pFrom and pTo are not specifed on the call to RXS_soapDecode.

```
from(1) = '&lt;';
to(1) = '<';
from(2) = '&gt;';
to(2) = '>';
from(3) = '&quot;';
to(3) = '"';
from(4) = '&apos;';
to(4) = '"';
from(5) = '&amp;';
to(5) = '&';
```

The following is an example program showing how to translate the contents of a variable to an encoded form. Decoding would simply be taking the opposite approach, essentially switching the from values with the to values and vice versa.

```
H dftactgrp(*no) bnddir('RXSBND')

 /copy rxs,RXSCp

D from            s             10a   dim(10) varying
D to              s             10a   dim(10) varying
D str             s          65535a   varying
 /free

  str = 'ADAMS & SONS >< LTD';

  from(1) = '&';
  to(1) = '&amp;';
  from(2) = '<';
  to(2) = '&lt;';
  from(3) = '>';
  to(3) = '&gt;';
  RXS_soapDecode(str: RXS_VAR: from: to);

  *inlr = *on;
```

```
/end-free
```

5.2 Template Engine (XML Composition)

Description: Use the Template Engine subprocedures to compose XML that responds to a request made to your RPG Web Service or to compose a request stream file to be sent to a remote web service. A template is simply a text file residing in the IFS that has variable place holders. At transaction time a function needs to replace the place holders with live data. Ending the templates with a .tpl extension is the recommended convention.

Please refer to programs EXAMPLE/TPLENG1, EXAMPLE/TPLENG2, and EXAMPLE/TPLENG3 in library RXS for examples of Template Engine usage.

5.2.1 RXS_initTplEng

Description: Call this subprocedure to initialize the Template Engine environment. Its parameters allow programmers to override default options. To accept a default, pass the keyword **omit* for a particular parameter.

Prototype:

```
D RXS_initTplEng...
D                 pr
D  pOutType                     10i 0 value
D  pOutFile                           const options(*omit)
D                                     like(RXS_FilePath)
D  pCodePage                    10u 0 const options(*omit)
D  pTplDir                            like(RXS_FilePath)
D                                     const options(*omit)
D  pTransDir                          like(RXS_FilePath)
D                                     const options(*omit)
D  pDebug                       n     value
```

Parameters:

pOutType – Used to specify where to send output when the internal buffer is flushed. Passing RXS_STDOUT will route output to standard output (used when offering a web service from your System i5). Passing RXS_STMF will route output to an IFS stream file. Passing RXS_VAR will allow the content to be buffered until RXS_getBuffData is called to retrieve the data. Those are the only valid values.

pOutFile – If pOutType = RXS_STMF this parameter specifies the file name of the IFS file that will receive the output. Specify the full path (i.e. /myifs/folder/myfile.xml) only when the desired destination is not in the default transaction directory. If pOutType = RXS_STDOUT, specify *omit here.

pCodePage – If pOutType = RXS_STMF, this parameter can contain a code page to override the default. If pOutType = RXS_STMF and this parameter is omitted, RXS will use the job's default values. If pOutType = RXS_STDOUT, specify *omit here.

pTplDir – Specifying the template directory temporarily overrides the location the Template Engine should look for templates. For example the default template directory may be set to /www/xml4RPG/templates/ but your program may want to temporarily change it to /home/myname/templates/.

pTransDir – Specifying the transaction directory temporarily overrides the location where the Template Engine will place XML transaction files (i.e. XML documents.) For example, the default transaction directory may be set to /www/xml4rpg/trans/ but your program may want to temporarily change it to /home/myname/trans/

pDebug – Setting this to *on will enable writing informational messages to the job log for debugging purposes. Once an application goes into production, set this value to *off to keep job logs from filling up.

Notes:

Sending to standard out (using RXS_STDOUT for pOutType) when offering a web service generally executes faster because it does not need to go to an IFS stream file first. Using RXS_VAR for pOutType will also execute slightly faster when executing a web service on a remote machine because it does not need to go to an IFS stream file.

5.2.2 RXS_getTplDir

Description: Returns the value specified for the template directory that was either specified on the call to RXS_initTplEng or RXS_setTplDir or was obtained by default from the RXSCFG file.

Prototype:

```
D RXS_getTplDir...
D                 pr                  like(RXS_FilePath)
```

Parameters:

Return Parameter – The full path of the template directory will be returned (e.g. /www/rxs/template/)

5.2.3 RXS_setTplDir

Description: Use this subprocedure when your application needs to temporarily change the template directory. The temporary value can be reset within a job by calling RXS_initTplEng. Note that this function does not change the RXSCFG physical file. Instead it stores the override value within the job.

Prototype:

```
D RXS_setTplDir...
```

```
D                 pr
D  pDir                         value like(RXS_FilePath)
```

Parameters:

pDir – Specify the temporary directory to use for templates (e.g. /home/myname/template/)

5.2.4 RXS_getTransDir

Description: Returns the value specified for the transaction directory that was either specified on the call to RXS_initTplEng or RXS_setTplDir or was obtained by default from the RXSCFG file.

Prototype:

```
D RXS_getTransDir...
D                 pr                  like(RXS_FilePath)
```

Parameters:

Return Parameter – The full path of the transaction directory will be returned (e.g. /www/rxs/trans/)

5.2.5 RXS_setTransDir

Description: Use this subprocedure to temporarily change the transaction directory. The temporary value can be reset within a job by calling RXS_initTplEng. Note that this function does not change the RXSCFG physical file. Instead it stores the override value within the job.

Prototype:

```
D RXS_setTransDir...
D                 pr
D  pDir                         value like(RXS_FilePath)
```

Parameters:

pDir – Specify the temporary directory to use for transactions (e.g. /home/myname/template/)

5.2.6 RXS_getBuffData

Description: Use this subprocedure to retrieve the data that has buffered in the Template Engine. Note that RXS_VAR should have been specified for the pOutType parameter on the call to RXS_initTplEng API for this to work. This is most often used in conjunction with RXS_getURI and passed for the pReqData parameter.

Prototype:

```
D RXS_getBuffData...
D                 pr         65535a   varying
D  pFlushBuff                    n   const options(*nopass)
D  pOffset                     10u 0 const options(*nopass)
```

Parameters:

Return Parameter – A 65535 VARYING field will be returned that contains the data from the template engine buffer.

pFlushBuff – Determine whether or not to flush the buffer after the template data is returned. This value should most often be *ON.

pOffset – Used to tell the template engine where to start retrieving data in the template engine buffer. Unless necessity dictates it be used do not pass this parameter (note the *NOPASS). Use RXS_BuffLen to determine if there is more than 65535 bytes of data currently buffered in the template engine.

5.2.7 RXS_getBuffLen

Description: Use this subprocedure to retrieve the length of data that has buffered in the Template Engine. Note that RXS_VAR should have been specified for the pOutType parameter on the call to RXS_initTplEng API for this to work. This is most often used in conjunction with RXS_getURI where RXS_getBuffData is used to obtain the current data stored in the template engine.

Prototype:

```
D RXS_getBuffLen...
D                 pr            10u 0
```

Parameters:

Return Parameter – An unsigned integer will be returned declaring the length in bytes of data found in the buffer.

5.2.8 RXS_loadTpl

Description: Use this subprocedure to load a new template into the Template Engine. A template is essentially a file in the IFS that contains XML with named sections and variable data placeholders. Commonly programmers reuse templates across applications or web services. For instance, if many RPG web service applications use the same envelope tag, a separate file can store the header and footer of the envelope (e.g. envelope_header.tpl and envelope_footer.tpl) that encapsulates contents composed using a variety of content templates.

Prototype:

```
D RXS_loadTpl...
D                 pr
D  pFile                         value like(RXS_FilePath)
D  pSecStart                20a  const varying options(*nopass)
D  pSecEnd                  20a  const varying options(*nopass)
```

```
D  pVarStart                    20a   const varying options(*nopass)
D  pVarEnd                      20a   const varying options(*nopass)
```

Parameters:

pFile – Specify the template file to load into the Template Engine. You can pass a fully qualified path (e.g. '/www/rxs/template/mytemplate.tpl') or a relative path based on the default template directory (e.g. 'mytemplate.tpl'). The default template directory can be found in physical file RXSCFG.

pSecStart – Override the default section start delimiter found in RXSCFG with one of your own.

pSecEnd – Override the default section end delimiter found in RXSCFG with one of your own.

pVarStart – Override the default variable data placeholder start delimiter found in RXSCFG with one of your own.

pVarEnd – Override the default variable data placeholder end delimiter found in RXSCFG with one of your own.
Notes:

If your program uses the default section and variable data placeholder delimiters then you can simply use the pFile parameter and not pass the rest (i.e. RXS_loadTpl('myfile.tpl')).

5.2.9 RXS_updVar

Description: Use this subprocedure to update the contents of a template engine variable defined in a section (e.g. <phone>.:phone:.</phone>)

Prototype:

```
D RXS_updVar      pr
D  pName                      30a   const varying
D  pValue                   1024a   const varying options(*varsize)
D  pTrim                       n   value options(*nopass)
```

Parameters:

pName – The name of the variable to replace with the value specified in the second parameter. An example of what the variable looks like in the template file would be <phone>.:phone:.</phone> where .:phone:. is the template variable that will be replaced.

pValue – The value to replace the template variable's place holder. For example, a template variable of .:phone:. may be replaced by the value '123-123-1234'. To develop this concept further, please view the following template snippet:

<phone>.:phone:.</phone>

Doing an RXS_updVar('phone': '123-123-1234') results in <phone>123-123-1234</phone>. Note that options(*varsize) has been specified. This allows the program calling this procedure to pass more than 1024 bytes of data.

pTrim – Specify *on for this parameter in order to trim the data of blanks at the beginning and end of the string or *off to maintain the spaces as passed. If the pTrim parameter is not specified a default of *on will be used.

5.2.10 RXS_wrtSection

Description: Call this subprocedure to write out a section (e.g. ::envelope_begin.) The most common use will be to call RXS_wrtSection after one or many calls have been made to RXS_updVar. Make sure to specify the flush parameter before your program ends to ensure all data has been processed in the buffer and sent to either standard out or an IFS stream file.

Prototype:

```
D RXS_wrtSection...
D                 pr
D  pSections                  1024a   value varying
D  pFlush                         n   options(*nopass) value
```

Parameters:

pSections – Specify one or many sections to write. When specifying more than one section, separate them with a space.

pFlush – Flush everything that has been written to either standard out or an IFS File. The destination will depend on what was specified on the pOutType parameter for subprocedure RXS_tplEngInit. Note that if RXS_VAR was specified for pOutType on RXS_tplEngInit, the flush will have no effect here. Instead when RXS_getBuffData is called *ON should be specified to flush the buffer.

5.3 Transmit

Description: These APIs will facilitate the transmission of the XML document from your System i5 RPG program to the remote web service and appropriately receive the response back to store in the medium specified.

5.3.1 RXS_getUri

Description: Call this subprocedure to send an XML stream to an "end point". An "end point" describes a web service you call or "consume". GETURI3 provides an example of calling end point RXS3 which resides on your iSeries from the initial install of RPG-XML Suite. The RXS_getURI is VERY configurable concerning what can be sent to the remote server. When doing standard web services there are only a handful of values for the pInCfg data structure that need to be filled which is detailed by way of the examples in RXS/EXAMPLE.

Prototype:

```
D RXS_getUri      pr
D  pInCfg                       likeds(RXS_GetUriIn)
D  pReqData                     like(RXS_XMLData) options(*omit)
D  pRspData                     like(RXS_XMLOut) options(*omit)
D  pHttpHdr                     like(RXS_GetUriHead) options(*omit)
```

Parameters:

pInCfg – This parameter allows you to describe the communication that will happen with the server on the other end of the line. The data structure is broken out below by field with a definition.

pReqData – This parameter should be specified if you are not using an IFS file to hold your XML request but instead have it in an RPG variable. To get your XML into an RPG variable you would have specified RXS_VAR as the pOutType on RXS_initTplEng and then used RXS_getBuffData. Note that a max of 65535 bytes can be sent using this method. If more XML is to be sent then an IFS file should be used. If an IFS file is used you can use *OMIT to omit this parameter.

pRspData – This parameter should be specified if you are not using an IFS file to hold your XML response but instead wish to have it placed in an RPG variable. Note that RPG-XML Suite can parse XML that resides in either an RPG variable or an IFS file. Note that a max of 65535 bytes can be received using this method. If more XML is to be received then an IFS file should be used. If an IFS file is used you can use *OMIT to omit this parameter.

pHttpHdr – This parameter will receive back the raw HTTP response headers which can be used for debugging purposes. *OMIT can be specified if you do not wish to receive back the HTTP response headers.

Datastructure pInCfg details:
RXS_getUriIn.URI (Required)
Enter the URI to be used. Do not include any data (GET or POST) parameters or port in this field. Use pInCfg.port to specify a specific port. Example of what it should look like: www.myserver.com/apps/getcustomer.asp

RXS_getUriIn.RspType (Required)
Specify the output type to be used. When using the RXS_getUri to return a value to your application, set this value to RXS_VAR. To return the data to a physical file, use RXS_PHY_FILE. To return the data to a stream file, set the value to RXS_STMF. If you intend to parse the data using RXS_parse, use RXS_VAR or RXS_STMF. The parser does not parse XML stored in a physical file. RXS_VAR is the default.

RXS_getUriIn.UserAgent (Default: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98))
Enter the user agent that you wish to be used for the request. This parameter may or may not have an affect on the success of the request. If problems occur, try a common user agent value such as "Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)" (without the quotes).

RXS_getUriIn.ReqType

Specify contstant RXS_STMF to have the XML request pulled from a file in the IFS or RXS_VAR if you are passing the XML in via an RPG variable through RXS_getURI parameter pReqData. RXS_VAR is the default.

RXS_getUriIn.ReqStmf
If RXS_STMF was specified on the ReqType parameter then specify the fully qualified path of the stream file containing the XML. If it resides in the default transaction directory, just specify it as 'mydoc.xml'.

RXS_getUriIn.Proxy
Specify the IP address or DNS-resolvable name of a proxy server if required.

RXS_getUriIn.Port (Default: 80)
Enter the port number used to make the request. If using a proxy server, pass the proxy server port. In this case, if you make a URI request from a server on a port other than 80, that port must be specified in the URI request. For example, www.myserver.com:442.

RXS_getUriIn.ReqMeth
Specify the request method used for the web service request. The valid values are constants RXS_GET, and RXS_POST. The most common for web services is POST whick is also the default value if not specified.

RXS_getUriIn.Accept
Specify the type of data that you will accept. This value should be in a valid content-type form (such as "text/xml".) The most common value will be text/xml which is also the default.

RXS_getUriIn.Host (Default: localhost)
Specify the host name that will be sent with the request. The default will be set to the host on the domain used in the request.

RXS_getUriIn.Cookie
Specify any cookie data to be sent with this request. Further instructions for formatting this data is available at RFC2109 at http://www.w3.org/Protocols/rfc2109/rfc2109.

RXS_getUriIn.Referer
Specify the value to be used as the referrer for this request.

RXS_getUriIn.Connection
Specify a value to be sent with the Connection HTTP header. For example the value "keep-alive" may be used.

RXS_getUriIn.ContType
Specify the content type of the request. The most common value for this when doing web services is text/xml which is also the default.

RXS_getUriIn.Proto
Specify the protocol being used. HTTP is the most common and also the default.

RXS_getUriIn.HTTPVer
Specify the protocol version being used. Version 1.0 is the most command and also the default

RXS_getUriIn.NbrHdrs
Specify the number of user defined generic headers to be used.

RXS_getUriIn.UsrHdr
Specify the user defined generic headers to be used in this array.

RXS_getUriIn.UsrHdrDta
Specify the user defined generic header data to be used in this array. Each element should relate to a header in the RXS_getUriIn.UsrHdr array.

RXS_getUriIn.File
Specify the library and physical file to be used to store the data results if RXS_PHY_FILE was specified on the RXS_getUriIn.OutType parameter. The first ten characters should be the file name and the second ten characters should contain the library name. This should be used infrequently.

RXS_getUriIn.Mbr
Specify the name of the physical file member to store the data results if RXS_PHY_FILE was specified on the RXS_getUriIn.OutType parameter. This should be used infrequently.

RXS_getUriIn.RspStmf
Specify the fully qualified path of the stream file to store the data results if RXS_STMF was specified on the RXS_getUriIn.OutType parameter. If it resides in the default transaction directory, just specify it as 'mydoc.xml.'

RXS_getUriIn.UpFile
Specify RXS_YES on this parameter if you are simulating an HTML file upload web page. Note that this does not refer to the XML document being sent but a separate file. This should be used infrequently.

RXS_getUriIn.UpStmf
Specify the fully qualified location of the stream file that will be uploaded if RXS_STMF was specified for the Upload File parameter. Note that this does not refer to the XML document being sent but a separate file. This should be used infrequently.

RXS_getUriIn.UpName (v3.35 and up)
Specifies the name of the file as the server will see it. This value defaults to the file name specified on RXS_getUriIn.UpStmf. Note that this does not refer to the XML document being sent but a separate file. This should be used infrequently.

RXS_getUriIn.UpField
When specifying RXS_YES for the Upload File parameter, name the form field from the upload page that is being simulated. Note that this does not refer to the XML document being sent, but a separate file. This should be used infrequently.

RXS_getUriIn.UpCont
When specifying RXS_YES for the Upload File parameter, specify the content-type of the file being uploaded. Note that this does not refer to the XML document being sent but a separate file. This should be used infrequently.

RXS_getUriIn.BUser
Specify the user ID for this parameter if Basic Authentication is used on this request.

RXS_getUriIn.BPW
Specify the password for this parameter if Basic Authentication is used on this request.

RXS_getUriIn.PUser
If using a proxy on this request that requires a user id and password, specify the proxy user id on this parameter.

RXS_getUriIn.PPW
If using a proxy on this request that requires a user id and password, specify the proxy password on this parameter.

RXS_getUriIn.SSL
Specify RXS_YES on this parameter if a Secure Sockets Layer (SSL) request is being made.

RXS_getUriIn.SSLApp
Used to assign an application ID for RXS_getUri. Default is blank (none).

RXS_getUriIn.CStore
Specify the certificate store to use with the SSL request. This value defaults to RXS_SYSTEM which uses the system certificate store. If you wish to use another certificate store, specify the qualified location of the store.

RXS_getUriIn.CPW
Specify the certificate password, if applicable.

RXS_getUriIn.SSLTime
Specify a value in seconds to cause a timeout during an SSL handshake. A value of RXS_NONE will specify no timeout value.

RXS_getUriIn.Timeout
Specify a value in seconds for a timeout on the request. Default is 30 seconds.

RXS_getUriIn.CodPag
Specify a code page to be used when creating stream files. Default is 819.

RXS_getUriIn.Close
Specify whether or not to close the connection after the request has completed. Leaving the connection open may increase performance with multiple calls to the same server. Default is RXS_YES which closes the connection after the request has completed.

RXS_getUriIn.Debug
Specify RXS_YES to debug the request that was made. A file will be created in the IFS named /tmp/getUridebug.txt which includes the actual request that was made with the RXS_getUri call. You can override this value by specifying a different file in RXS_getUriIn.DebugFile.

RXS_getUriIn.DebugFile
Specify the location of a file to place debug information to override the default of /tmp/getUridebug.txt.

RXS_getUriIn.CCSID

Specify the CCSID to use for EBCDIC to ASCII and ASCII to EBCDIC translations. This parameter will override the tables if specified. If the tables are used instead of the CCSID for translations, specify 0 (zero) for this value.

RXS_getUriIn.EATable
Specify EBCDIC to ASCII translation table to be used. This is normally QTCPASC but can be other table depending on the code page you may be using. In some cases using code page 37 table Q037337850 works better. The table used must reside in library QUSRSYS.

RXS_getUriIn.AETable
Specify the ASCII to EBCDIC translation table to be used. This is normally QEBCDIC but can be a different table depending on the code page in use. The table used must reside in library QUSRSYS.

RXS_getUriIn.LocalIP
Specify an IP address to bind the request to a specific Local IP address.

RXS_getUriIn.LocalPort
Specify a port number to bind the request to a specific IP/Port combination. This value is only valid if a value is specified for RXS_getUriIn.LocalIP.

RXS_getUriIn.SprHead
Communicating via HTTP requires that certain "headers" be sent first on the communication line between the client and the server – your iSeries is the client in this case. When your program calls RXS_getUri, it sends HTTP headers to the remote web service based on the values specified in the data structure passed on the RXS_getUri API call. The web remote server will use those HTTP headers to know how to process the request. When that server responds to your request it will also send HTTP headers that detail the communication that took place and give information about the content passed back from the server (i.e. Content-type: text/xml). Since you will want to parse the response from the server 99% of the time we need to make sure the file or data we are parsing only contains valid XML. HTTP headers are not valid XML, so with this parameter we can tell the RXS_getUri sub procedure to separate the HTTP headers out into a separate file with a .hdr extension.

Specify RXS_YES to suppress and separate out the HTTP headers or RXS_NO to keep the HTTP headers in the response. If RXS_YES is specified and RXS_STMF is the output type, a file named "xxxxx.hdr" will be created (where "xxxxx" is the filename specified to contain the output) that will contain the response headers. Note that the headers will also be present in returned in the fourth parm of the call to RXS_getUri if it was not omitted. One final note – specify RXS_YES all the time unless you have requirements that dictate otherwise.

RXS_getUriIn.HTTPHead
Specifies if the HTTP headers should be sent with the request. Specifying anything other than RXS_YES, the request must be a POST. RXS_YES will tell RXS_getUri that all HTTP headers as well as the user defined headers are se the request. RXS_NO will tell GETURI that no HTTP headers are sent with the request and only the pReqData parameter is sent as-is. The value of RXS_USRHDR means that only the user defined headers are sent with the request.

5.4 CGI

Note that the term "standard out" refers to the default location of output. In the case of an RPG Web Service, *standard out* is sent to the program that sent the request. For instance, calling your web service from a browser would result in your RPG Web Service writing the response back to the browser.

Description: This is a "raw output" subprocedure that will allow you to push data to standard out in any fashion that suits your needs. An example usage is within RXS_stdOutError which simply composes a small XML stream to send to standard out without using the Template Engine.

Prototype:

```
D RXS_out         pr
D  pData                      65535a   Value varying
```

Parameters:

pData – Pass in the data to be send to standard out.

5.4.1 RXS_outFromFile

Description: This sends the entire contents of a file in the IFS to standard out. This is useful when using the Template Engine to compose files in the IFS. Once the XML file is created, use RXS_outFromFile to write the entire contents to standard out. Use this when writing RPG Web Services residing on your iSeries that other programs call and NOT when consuming web services on a remote system.

Prototype:

```
D RXS_outFromFile...
D                 pr
D  pFile                              value like(RXS_FilePath)
```

Parameters:

pFile – The qualified file name in the IFS (e.g. /myfolder/myfile.xml)

Notes:

5.4.2 RXS_writeXMLHdr

Description: Use this subprocedure to quickly write out the barebones http headers for a web service. It is basically outputting Content-type: text/xml followed by two carriage return line feeds. Note that this subprocedure could be used to write out the http headers instead of putting them in a template for the Template Engine to send them. Use this when writing RPG Web Services residing on your iSeries that other programs call and NOT when consuming web services on a remote system.

Prototype:

```
D RXS_writeXMLHdr...
D                 pr
```

Parameters:

There are no input or output parameters for this subprocedure.

Notes:

5.4.3 RXS_getEnvVar

Description: There are times when you may need to gain access to the Apache server's environment variables or http headers that are sent with the incoming request. This subprocedure will ease that retrieval.

Prototype:

```
D RXS_getEnvVar...
D                 pr         32767a   varying
D  pEnvVar                      30a   value
```

Parameters:

pEnvVar – The name of the environment variable wanted for retrieval.

Notes:

Some of the valid values to pass into RXS_getEnvVar include the following:

Environment variables

AUTH_TYPE
CGI_ASCII_CCSID
CGI_EBCDIC_CCSID
CONTENT_LENGTH
CONTENT_TYPE
GATEWAY_INTERFACE
HTTP_ACCEPT
HTTP_USER_AGENT
PATH_INFO
PATH_TRANSLATED
QUERY_STRING
REMOTE_ADDR
REMOTE_HOST
REMOTE_IDENT
REQUEST_METHOD
REMOTE_USER
SCRIPT_NAME
SERVER_NAME
SERVER_PORT
SERVER_PROTOCOL

SERVER_SOFTWARE

5.4.4 RXS_putEnvVar

Description: The QtmhPutEnv API allows the programmer to set or create a job-level environment variable.

Prototype:

```
D RXS_putEnvVar   pr
D  pEnvVar                    32767a   value varying
```

Parameters:

pEnvVar – The format of this is ”envVar=value”. ”envVar” is the name of the new or existing environment variable, and ”value” is the value you want to set the environment variable. **Note that they are both case sensitive.**

5.4.5 RXS_getUrlVar

Description: Some web services may require their consumers to pull variables off of the URL. This subprocedure will simplify that process. Here is an example URL with variables (say name value pair): http://mysite.com/rxs/mywebservice?customer=1&code=3

Prototype:

```
D RXS_getUrlVar   pr          32767a   varying
D  pVar                         300a   value
```

Parameters:

pVar – Specify the name of the variable you want the value for (e.g. customer from the above URL example)

Return Parameter – will contain the value of variable specified in pVar.

5.4.6 RXS_readStdIn

Description: This subprocedure will allow you to read the XML data that was sent to you via an HTTP POST. The contents of the HTTP POST will be returned to your program in the return parameter (i.e. xmlVar = RXS_readStdIn();).Once the XML is obtained it can be passed to the parser using RXS_parse().

Prototype:

```
D RXS_readStdIn...
D                 pr          65535a   varying
```

Parameters:

Return Parameter – will provide the post contents back to your program in the form of a VARYING string.

Notes:

The max returned will be 65535 bytes of data. If a web service has the potential need to receive more than that, use RXS_readToFile. It has the capability to read in up to 16MB of XML.

RXS_readStdIn is good for small web services and offers increased performance over RXS_readToFile.

5.4.7 RXS_readToFile

Description: This subprocedure allows the programmer to read the XML data into a file that was sent to your program via an HTTP POST. Once the XML is obtained it can be passed to the parser using RXS_parse().

Prototype:

```
D RXS_readToFile...
D                 pr
D  pFile                          value like(RXS_FilePath)
```

Parameters:

pFile – Specify the qualified name of the IFS file to read the results to (e.g. /myfolder/webservice_request.xml)

Tip:

This subprocedure allows the programmer to read up to 16MB of XML. If an application needs to transmit more than 16MB, other methods may better serve the requirement. XML is probably not the best method for data replication or for transferring thousands of records at a time and similar procedures. The extra overhead caused by passing tags causes problems. For example the following XML takes 36 characters to transmit 3 characters of data: <customerNumber>123</customerNumber>.

5.5 Miscellaneous

5.5.1 RXS_charToBln

Description: Use RXS_charToBln to easily convert a character Boolean value (i.e. true/false, t/f, on/off/, 1/0) to an RPG boolean. This will save the headache of programming for all of the different timestamp formats out there and doing a bunch of sub-stringing and concatenation.

If the value in pValFrmXml matches the value in pTrueVal then *on will be returned. If the value in pValFrmXml matches the value in pFalseVal then *off will be returned. If the value in pValFrmXml doesn't match either pTrueVal or pFalseVal then the value spefied in pDftVal will be returned.

```
myBlnVar = RXS_charToBln(strBln: 'true': 'false': *off);
```

Prototype:

```
D RXS_charToBln   pr              n
D  pValFrmXml                   10a   value
D  pTrueVal                     10a   value
D  pFalseVal                    10a   value
D  pDftVal                        n   value
```

Parameters:

pValFrmXml – Specify the string value parsed from an XML document. This may be a word (i.e. true/false) or a single character (i.e. t/f or 1/0). The next two parameters, pTrueVal and pFalseVal, will allow you to specify the values to expect for a true and false evaluation of the pValFrmXml's content.

pTrueVal – Specify the value representing a true Boolean condition (e.g. 'true' or '1' or 't')

pFalseVal – Specify the value representing a false Boolean condition (e.g. 'false' or '0' or 'f').

pDftVal – If the value in pValFrmXml doesn't match pTrueVal or pFalseVal then the value specified in pDftVal will be returned.

Notes:

5.5.2 RXS_timestampToChar

Description: Use RXS_timestampToChar to easily convert a timestamp coming from your database or program to an XML format that matches the requirements in the XSD (XML Schema Definition). This will save the headache of programming for all of the different timestamp formats out there and doing a bunch of sub-stringing and concatenation.

```
myTimestampString = RXS_timestampToChar('yyyy-MM-ddThh.mm.ss': %timestamp());
```

Prototype:

```
D                 pr            50a   varying
D  pFormat                      30a   value
D  pTimestamp                     z   value
```

Parameters:

pFormat – Specify the format of the timestamp string so the converter knows where to place each portion of the timestamp. The idea is to combine the below keywords (i.e. yyyy, MM, dd, hh, mm, ss, SSSSSS) into a string that represents the location of where the corresponding value should be placed in the string representation of the timestamp.

yyyy = Year
MM = Month
dd = Day
hh = Hour
mm = Minute
ss = Second
SS = Millisecond

pTimestamp – Specify the timestamp to be converted.

Notes:

5.5.3 RXS_charToTimestamp

Description: Use RXS_charToTimestamp to easily convert a timestamp coming from an XML document (i.e. string representation of a timestamp) to a valid iSeries/RPG timestamp format. This will save the headache of programming for all of the different timestamp formats out there and doing a bunch of sub-stringing and concatenation.

```
RXS_charToTimestamp(strTimestamp: 'yyyy-MM-dd-hh.mm.ss.SSSSSS': %timestamp());
```

Prototype:

```
D RXS_charToTimestamp...
D                 pr             z
D  pString                      30a   value
D  pFormat                      30a   value
D  pDft                           z   value
```

Parameters:

pString – Specify a variable name or character literal that came from the XML parsing process.

pFormat – Specify the format of the timestamp string so the converter knows where to pull each value. The idea is to combine the below keywords (i.e. yyyy, MM, dd, hh, mm, ss, SSSSSS) into a string that represents the location of where the corresponding value can be found in the string representation of the timestamp.

yyyy = Year
MM = Month
dd = Day
hh = Hour
mm = Minute
ss = Second

SS = Millisecond

pDft – If the string timestamp doesn't match the format specified then the process cannot complete successfully and this default field's value will be returned.

Notes:

5.5.4 RXS_cmpTransFile

Description: In the process of doing web services you will most likely run across the need to create many files in the IFS, and each of those files must be uniquely named. Instead of doing several concatenations of strings and dates and numbers to come up with a unique file name, use RXS_cmpTransFile. Basically, a programmer tells the function the extension (i.e. '.xml') the separator for each piece of information (up to 4) that will be passed to it (i.e. I like to use underscore, "_"). The following example produces a file named 99999_req.xml where 99999 is the next sequential number in data queue RXS/RXSUNQ:

```
gReqFile = RXS_cmpTransFile('_': '.xml': RXS_nextUnqChar(): 'req');
```

Prototype:

```
D RXS_cmpTransFile...
D                 pr                  like(RXS_FilePath)
D  pSep                            1a   value
D  pExt                           10a   value varying
D  pVal1                                value like(RXS_NameVal)
D                                       options(*nopass)
D  pVal2                                value like(RXS_NameVal)
D                                       options(*nopass)
D  pVal3                                value like(RXS_NameVal)
D                                       options(*nopass)
D  pVal4                                value like(RXS_NameVal)
D                                       options(*nopass)
D  pVal5                                value like(RXS_NameVal)
D                                       options(*nopass)
```

Parameters:

pSep – Specify the value to use for the separator between each pVal1-5 parameter. An underscore is a good choice.

pExt – Specify the extension of the file including the period(i.e. '.xml'.)

pVal1-5 – Use these parameters to specify the data for the subprocedure to use for qualifying the IFS file. RXS_cmpTransFile recognizes two special keywords – RXS_UnqNbr which retrieves a unique number from data area RXS/RSXUNQ, and RXS_Timestamp which returns a timestamp in character format. Using these keywords reduces the amount of %char() and %timestamp() invocations needed for passing parameters.

Notes:

5.5.5 RXS_getFileSize

Description: This subprocedure will allow the programmer to obtain the size of a file in the IFS in bytes.

Prototype:

```
D  RXS_getFileSize...
D                 pr            10i 0
D  pFile                              value like(RXS_FilePath)
```

Parameters:

pFile – Specify the qualified IFS file name of which the size is unknown (e.g. /myfolder/myfile.xml)

Notes:

5.5.6 RXS_deleteFile

Description: Allows the programmer to easily delete a file in the IFS.

Prototype:

```
D RXS_deleteFile  pr            10i 0
D  pFile                              value like(RXS_FilePath)
```

Parameters:

pFile – Specify the qualified name of the IFS file to be deleted (e.g. /myfolder/myfile.xml)

Notes:

5.5.7 RXS_log

Description: Use this subprocedure to easily write useful information to the job log. There are a variety of types of messages to send that are detailed below.

Prototype:

```
D RXS_log         pr
D  pType                              const like(RXS_MsgType)
D  pMsg                        32767a const options(*varsize)
```

Parameters:

pType – This can be any of the below valid values. The definitions for each type can be found at the following URL:
http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/apis/QMHSNDPM.htm

Valid Values for pType

```
D RXS_ESCAPE      s                   inz('*ESCAPE') like(RXS_MsgType)
D RXS_COMP        s                   inz('*COMP') like(RXS_MsgType)
D RXS_DIAG        s                   inz('*DIAG') like(RXS_MsgType)
D RXS_INFO        s                   inz('*INFO') like(RXS_MsgType)
D RXS_INQ         s                   inz('*INQ') like(RXS_MsgType)
D RXS_NOTIFY      s                   inz('*NOTIFY') like(RXS_MsgType)
D RXS_RQS         s                   inz('*RQS') like(RXS_MsgType)
D RXS_STATUS      s                   inz('*STATUS') like(RXS_MsgType)
```

pMsg – Specify the text that should appear in the job log.

Notes:

5.5.8 RXS_nextUnqNbr

Description: Use this subprocedure to easily obtain the next sequential number from data area RXS/RXSUNQ. Useful when creating IFS files that need to be unique.

Prototype:

```
D RXS_nextUnqNbr  pr            15  0
```

Parameters:

Return Parameter – returns the next sequential number from data area RXS/RXSUNQ.

Notes:

5.5.9 RXS_nextUnqChar

Description: Works the same as RXS_nextUnqNbr except it returns a 15 byte character. It helps when creating IFS files that need to be unique by eliminating the need for %char() in concatenation of variables.

Prototype:

```
D RXS_nextUnqChar...
D                 pr            15a
```

Parameters:

Return Parameter – returns the next sequential number from data area RXS/RXSUNQ in character form.

5.5.10 RXS_charToNbr

Description: XML data is passed as strings. If your system needs to store it as decimal or numeric data, you will need to convert it. Versions of the RPG ILE compiler newer than V5R1 allow programmers to convert strings to number using the %INT or %DEC

built-in functions. This subprocedure simplifies character to numeric conversion on a V5R1 system.

Prototype:

```
D RXS_charToNbr   pr             30P 9
D  pStr                          50A   varying const
```

Parameters:

pStr – Pass the string value to convert. If this function encounters a non-numeric character (i.e. only a decimal point and 0-9 are allowed), the conversion will stop at that point and return the value converted to that point.

Return Parameter – returns the numerical representation of the string passed in.

5.5.11 RXS_addLibLE

Description: More often than not your data files and business logic programs are not going to reside in the same library as your web service programs. RXS_addLibLE allows you to easily add additional libraries to your library list.

Prototype:

```
D RXS_addLibLE    pr
D  pLib                          10a   value
```

Parameters:

pLib – Specify the library you would like to add to this jobs library list. Note that it will be added to the beginning of the library list.

Notes: You can use RXS_libLEExists to see if a library list entry exists before adding the library to your library list.

5.5.12 RXS_libLEExists

Description: Used to check the existence of a library entry in the library list before using RXS_addLibLE to add one.

Prototype:

```
D RXS_libLEExists...
D                 pr             3  0
D  pLib                          10a   value
D  pLibType                      10a   value options(*nopass)
```

Parameters:

pLib – Specify the library you would like to ensure is in the library list.

pLibType – Specify the type of library to check for. This parameter can be omitted as you will be checking for *USR libraries 99% of the time.

Notes: This is usually used in conjunction with RXS_addLibLE to ensure a library doesn't exist in the library list before it is added.

5.5.13 RXS_rmvLibLE

Description: Used to remove a library entry from the library list.

Prototype:

```
D RXS_rmvLibLE    pr
D  pLib                         10a   value options(*nopass)
```

Parameters:

pLib – Specify the library you would like to remove from the library list.

Notes: This is usually used at the end of a program in conjunction with RXS_addLibLE to remove a library that was added at the beginning of the program.

5.5.14 RXS_handOff

Description: Depending on the chosen application architecture for your RPG Web Services, you may prefer to use separate programs or modules to do the XML parsing, composing and other tasks. With RXS_handOff, the programmer can dynamically invoke a subprocedure in a service program defined outside of the mainline program. RXS_handOff will take care of activating the specified service program and subprocedure and specify where the program should find the incoming XML file and where it should put the outgoing XML file.

The example program called DOORWAY demonstrates how to implement this architecture. This subprocedure works best when offering an RPG Web Service instead of when using a remote web service.

Prototype:

```
D RXS_handOff...
D                 pr                  like(RXS_Error)
D  pHandOff                           likeds(RXS_SubProc)
D  pInXml                             value like(RXS_FilePath)
D  pOutXml                            like(RXS_FilePath)
```

Parameters:

pHandOff (defined below) – Use this to specify the external service program and subprocedure to activate.

pInXml – Specify the qualified IFS file path of the request XML document.

pOutXml – Specify the qualified IFS file path in which the service program should place the response XML.

Notes:

```
D RXS_SubProc     ds                  qualified inz
D  srvPgm                       10a
D  subProc                     256a   varying
```

5.5.15 RXS_throwError

Description: RXS_throwError and RXS_catchError are used as an extension to the RPG compiler's MONITOR op-code. When IBM added the MONITOR op-code they only allowed for compiler predefined error messages to be "caught" on the corresponding ON-ERROR statement. That means you can't have an ON-ERROR statement watching for a programmer defined error code. With just a little bit of additional code we can utilize the MONITOR and ON-ERROR clauses to our benefit. By surrounding a piece of code that could be erroneous with the MONITOR op-code we can use the ON-ERROR clause to catch any errors that are thrown by programs further down the program call stack, and then using RXS_catchError to retrieve the most recent error off of the program call stack. For example, if PGMA wraps a MONITOR around a call to PGMB and PGMB uses RXS_throwError to generate an error, then the next line of code executed will be the ON-ERROR clause in PGMA because the RPG runtime recognizes that an error occurred and it looks for it's next stopping point (ON-ERROR is one such stopping point). RXS_catchError can then be used within the ON-ERROR clause to retrieve the last error off of the program call stack. At that point PGMA has access to the error code, severity, program name and error text. The decision of how to continue is up to PGMA as it caught the error and avoided an abnormal end. This is similar to using a *PSSR subroutine except that with *PSSR your program doesn't regain control after the *PSSR sub routine executes.

Prototype:

```
D RXS_throwError  pr                  extproc('ERROR_THROW')
D  pCode                              value Like(RXS_Error.code)
D  pSeverity                          value Like(RXS_Error.severity)
D  pPgm                               value Like(RXS_Error.pgm)
D  pText                              value Like(RXS_Error.text)
```

Parameters:

pCode – Specify a code that uniquely identifies this error. A common practice is to use codes similar to CPF errors (i.e. CPF0000). An example would be RXP0000001 which is an error code thrown from RXS_parse when the IFS file containing the XML to process cannot be found.

pSeverity – Specify the severity of the error. As a general practice this should be set to 100 (highest severity) as RXS_throwError should only be used in 'hard error' cases where processing cannot or shouldn't continue.

pPgm – Specify the program that is creating or throwing the error.

pText – Specify the error text to be sent with the error. This can be any value that helps describe the error that occurred.

Example Program (note you can also find this in XML4RPG/EXAMPLE,ERR1):

```
h dftactgrp(*no) bnddir('RXSBND')

 /copy rxs,RXSCp

d subproc1        pr

d gErr            ds                  likeds(RXS_Error) inz
 /free


  monitor;
    RXS_log(RXS_DIAG: 'Start of program...');

    subproc1();

    RXS_log(RXS_DIAG: '... program will never reach this statement.');
  on-error;
    gErr = RXS_catchError();  // Error retrieved from program call stack

    // Display the error contents to the current job log using RXS_log
    RXS_log(RXS_DIAG: 'code....:' + gErr.code);
    RXS_log(RXS_DIAG: 'severity:' + %char(gErr.severity));
    RXS_log(RXS_DIAG: 'pgm.....:' + gErr.pgm);
    RXS_log(RXS_DIAG: 'text....:' + gErr.text);
  endmon;

  *inlr = *on;

 /end-free

 //-------------------------------------------------------------------------
 // subproc1 will throw a generic error for the above to "catch"
 //-------------------------------------------------------------------------
p subproc1        b
d subproc1        pi
 /free

  RXS_throwError('ERR0001': 100: 'ERRPGMA': 'Misc error text');

 /end-free
p                 e
```

5.5.16 RXS_catchError

Description: RXS_throwError and RXS_catchError are used as an extension to the RPG compiler's MONITOR op-code. When IBM added the MONITOR op-code they only allowed for compiler predefined error messages to be "caught" on the corresponding ON-ERROR statement. That means you can't have an ON-ERROR statement watching for a programmer defined error code. With just a little bit of additional code we can utilize the MONITOR and ON-ERROR clauses to our benefit. By surrounding a piece of code that could be erroneous with the MONITOR op-code we can use the ON-ERROR clause to catch any errors that are thrown by programs further down the program call stack, and then using RXS_catchError to retrieve the most recent error off of the program call stack. For example, if PGMA wraps a MONITOR around a call to PGMB and PGMB uses RXS_throwError to generate an error, then the next line of code executed will be the ON-ERROR clause in PGMA because the RPG runtime recognizes that an error occurred and it looks for it's next stopping point (ON-ERROR is one such stopping point). RXS_catchError can then be used within the ON-ERROR clause to retrieve the last error off of the program call stack. At that point PGMA has access to the error code, severity, program name and error text. The decision of how to continue is up to PGMA as it caught the error and avoided an abnormal end. This is similar to using a *PSSR subroutine except that with *PSSR your program doesn't regain control after the *PSSR sub routine executes.

Prototype:

```
D RXS_catchError  pr                  likeds(RXS_Error)
D                                     extproc('ERROR_CATCH')
```

Parameters:

Return Parameter – A data structure like the following will be returned to the calling program.

```
D RXS_Error       ds                  qualified inz
D  code                         10a
D  severity                     10i 0
D  pgm                          30a   varying
D  text                      32000a   varying
```

Example Program (note you can also find this in XML4RPG/EXAMPLE,ERR1):

```
h dftactgrp(*no) bnddir('RXSBND')

 /copy rxs,RXSCp

d subproc1        pr

d gErr            ds                  likeds(RXS_Error) inz
 /free

  monitor;
    RXS_log(RXS_DIAG: 'Start of program...');

    subproc1();

    RXS_log(RXS_DIAG: '... program will never reach this statement.');
  on-error;
    gErr = RXS_catchError();  // Error retrieved from program call stack

    // Display the error contents to the current job log using RXS_log
    RXS_log(RXS_DIAG: 'code....:' + gErr.code);
    RXS_log(RXS_DIAG: 'severity:' + %char(gErr.severity));
    RXS_log(RXS_DIAG: 'pgm.....:' + gErr.pgm);
    RXS_log(RXS_DIAG: 'text....:' + gErr.text);
  endmon;

  *inlr = *on;

 /end-free

 //-------------------------------------------------------------------------
 // subproc1 will throw a generic error for the above to "catch"
 //-------------------------------------------------------------------------
p subproc1        b
d subproc1        pi
 /free

  RXS_throwError('ERR0001': 100: 'ERRPGMA': 'Misc error text');

 /end-free
p                 e
```

6 Version

To obtain the version of the base RPG-XML Suite installation, run the following from the command line.

```
CALL RXS/VERRXSBASE
```

7 Registration

To obtain the information necessary to register the RPG-XML Suite on your machine please run the following commands and copy/paste the info into an email and send to sales@krengeltech.com.

```
ADDLIBLE RXS
RXS/DSPMCHINF
```

8 Copyrights