Rexx and the WebSphere Optimized Local Adapter APIs

Techdoc: WP102348

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number **WP102348** under the category of "TechDocs"

Version 1.1

Author: Edward McCarthy
IBM
edwardmc@au1.ibm.com

Last Updated: 24 September 2013, 02:06

Table of Contents

1 Rexx and the WOLA APIs

1.1 Introduction

IBM introduced WebSphere Optimized Local Adapters (WOLA) in WebSphere Application Server V7 for z/OS. WOLA provides cross-memory local communications between WebSphere Application Server for z/OS and external address spaces such as CICS, IMS, batch programs, Unix Systems Services (USS) programs, and the Airline Control System (ALCS). Extensive information about WOLA can be found at this techdoc link:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490

In a WebSphere Application Server, applications use the supplied WOLA APIs via standard Java programming methods.

In environments outside of a WebSphere server the common approach had been to use programming languages such as COBOL, Assembler or C to invoke the WOLA APIs.

Another language popular amongst z/OS developers is the Rexx programming language.

In this techdoc we supply a set of Rexx programs that invoke the WOLA APIs providing another option for developing applications that take advantage of WOLA.

1.2 Rexx background

Rexx was developed by Mike Cowlishaw of IBM in 1979. It has typically been used by z/OS system programmers and developers to developer various utilities and general purpose programs for many decades. Rexx code is interpretive, meaning no compilation is required to run Rexx code.

1.3 Why use Rexx

This techdoc in no way aims to advocate that Rexx is a better programming language then the other languages that can already be used to invoke the WOLA APIs.

That being said, coding a Rexx program can be quicker then other programming languages and is often used to develop relatively simple programs to perform non-complex tasks.

The aim of this techdoc is to provide a set of Rexx sample programs that function as de-facto wrapper APIs for the WOLA APIs.

These Rexx APIs can then be easily used by anyone who wants to use Rexx to develop applications that would make use of the WOLA functionality, and thus making it simpler to use Rexx and the WOLA APIs.

1.4 Why Rexx wrapper APIs?

The WOLA APIs are described at this link from the WebSphere Application Server V8.5 Infocenter:

> http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-zos&topic=cdat_olaapis

There are thirteen WOLA APIs. Each of these APIs requires various parameters some of which must be certain lengths, be blank or null filled, have values in binary format etc.

Using Rexx to invoke the WOLA APIs and ensuring that all the various parameters are set correctly, while not one of the great programming tasks of our time, is still none the less a non-trivial effort.

One particular aspect in particular of using Rexx to invoke the WOLA APIs is that some parameters require that they contain the address of an address in memory.

Rexx provides no support for obtaining the address of a Rexx variable, and thus there is no straightforward way to code Rexx to meet this requirement.

To simplify the coding of Rexx to invoke the WOLA APIs this techdoc delivers a Rexx wrapper program for each WOLA API.

Within each of these wrapper Rexx programs there is code to handle setting up the required parameters in the required way for each WOLA API. For those WOLA APIs that require parameters containing the address of an address in memory, we use a supplied assembler program to GETMAIN an area of memory and return that address back to the Rexx program.

Those who want to use Rexx to develop applications of any size using the WOLA APIs can simplify their programming effort by using the supplied Rexx WOLA wrapper APIs. Developers still need to understand what parameters to pass to the various Rexx WOLA wrapper APIs, but they need not be concerned with working out the more complex task of coding the parameter list to pass to the actual WOLA APIs.

Using this sort of approach means that if the WOLA APIs are changed e.g. new parameters or options added, then it is much simpler to modify the Rexx WOLA Wrapper APIs then potentially numerous Rexx programs.

1.5 Rexx Wrapper APIs

The following table lists the WOLA APIs, the supplied Rexx WOLA wrapper API and a brief description of what each API does.

WOLA API	**Rexx WOLA Wrapper API**	**Description**
BBOA1REG	RBOA1REG	Register to a WebSphere environment

BBOA1URG	RBOA1URG	Unregister from a WebSphere environment
BBOA1CNG	RBOA1CNG	Get a connection
BBOA1CNR	RBOA1CNR	Release a connection
BBOA1SRQ	RBOA1SRQ	Send a request
BBOA1SRP	RBOA1SRP	Send a response
BBOA1SRX	RBOA1SRX	Send a response exception
BBOA1RCA	RBOA1RCA	Receive any request
BBOA1RCS	RBOA1RCS	Receive a specific request
BBOA1RCL	RBOA1RCL	Receive the length of a response
BBOA1GET	RBOA1GET	Get message data
BBOA1INV	RBOA1INV	Invoke
BBOA1SRV	RBOA1SRV	Host a service

1.5.1 Why are they called RBO…

You may be wondering why the Rexx WOLA Wrapper APIs are not just called the same name as the actual WOLA APIs. The reason is that at runtime the Rexx programs must also be able to call the real WOLA APIs. If the Rexx WOLA Wrapper APIs had the same name as the actual WOLA APIs, then only the Rexx WOLA Wrapper API or the actual WOLA API load module is ever found and the whole process would not work. Thus they needed to be called something different, so the simplest idea was to just to replace the first character with R for Rexx.

1.5.2 Warranty and support

These Rexx programs are supplied as-is.

No warranty from IBM is supplied or implied in any way.

IBM will provide no official support for these programs. IBM will provide support for the underlying WOLA APIs.

You may however contact the program author at edwardmc@au1.ibm.com if you have queries or issues who will respond as time permits.

1.5.3 Feedback

If you have any comments on these Rexx programs please contact the program author at edwardmc@au1.ibm.com.

1.6 Only 31 bit APIs supported

Rexx only provides support for linking to 31 bit load modules, which means that only the 31 bit WOLA APIs can be invoked by these Rexx WOLA wrapper APIs.

The main limitation of this is on how much data could be transferred between the Rexx and WebSphere applications, as it would be limited to less then 2G.

1.7 How to use the Rexx WOLA wrapper APIs

The Rexx WOLA wrapper APIs require various parameters depending on which WOLA API is being invoked.

The method for passing parameters to the Rexx WOLA wrapper APIs and the way data is passed back to the caller is the same for all.

Parameters can be passed in any order.

1.7.1 Syntax

The syntax for calling a Rexx WOLA wrapper API and passing parameters is as follows:

```
                +-”,------------------------------+
                V                                 |
     RBOA1xxx–(-+”parameterName=’”parameterValue”’+”)
```

1.7.1.1 *Example – passing one parameter*

This is an example of how to pass just one parameter:

```
     RBOA1xxx ( “parameterName=’”parameterValue”’”)
```

1.7.1.2 *Example – passing multiple parameters*

This is an example of how to pass multiple parameters:

```
     RBOA1xxx ( “parameterName1=’”parameterValue1”’”, ||
                “parameterName2=’”parameterValue2”’”, ||
                “parameterName3=’”parameterValue3”’”)
```

1.7.2 Hexadecimal and parameters

There are many ways the Rexx Wrapper WOLA APIs could have been coded when it came to how to structure the way parameters are passed in and results returned.

The method used aims to find a balance between ease of coding for the parsing of parameters passed in and handling of the values of these parameters.

For example the purpose of the RBOA1CNG API is to obtain a connection handle. This handle then has to be passed to other APIs to do subsequent activity. This connection handle is a 12 byte value of seemingly random bytes. The potential problem is that one or more of these bytes could be a ‘ or a “ or a comma. These are the very same characters used to delineate the parameter name and parameter value pairs passed in. If this was to occur then the parsing process would break down.

Similarly data that is returned to the call could also contain various byte values that could result in processing errors in the calling code.

To prevent this parameters that could potentially cause problems in the parsing, must be converted to an expanded hexadecimal representation before being passed as a value. Similarly data that is to be returned to the caller that could potentially contain non-alpha numeric characters is converted to an expanded hexadecimal representation before being returned.

This approach prevents problems in parsing of received parameters and allows the calling program to handle data received back from the Rexx WOLA Wrapper APIs in the way it needs to.

Rexx provides the C2X and X2C APIs to easily facilitate the above and the supplied examples show how to use these.

1.7.3 Results from the Rexx WOLA Wrapper APIs

Some of the WOLA APIs return only a return code and reason code. Some however will also return some sort of data as well. This may be a connection handle or some data string.

All the Rexx WOLA Wrapper APIs use an exit statement of the form:

```
        Exit RC ReturnCode ReasonCode ResultData
```

These returned parameters are as follows:

RC – Return code from the actual call to the WOLA API module

ReturnCode – return code from the WOLA API

ReasonCode – reason code from the WOLA API

ResultData – Result data if any from the WOLA API call

Depending on the WOLA API there could be multiple ResultData entries returned.

Note as explained above the result data may be an expanded hexadecimal string.

1.8 Supplied files and setup

This section describes the files supplied with this techdoc and what you need to do to be able to use the supplied Rexx WOLA wrapper APIs on your z/OS environment.

1.8.1 Supplied files

On the URL where you downloaded this document from is a zip file that can also be downloaded. Un-zip this file to a directory on your PC. The zip file contains the following:

Files and sub-directories	**Description**
xmitfile-rexxapis	z/OS XMIT file that contains the REXX WOLA Wrapper APIs
xmitfile-rexxjcl	z/OS XMIT file that contains the sample JCL
xmitfile-rexxload	z/OS XMIT file that contains the REXXMMGR load module
rexx	Contains the Rexx programs
jcl	Sample JCL
assem	Contains the Rexxmmgr assembler program source code
ivp-job-logs	Contains the output from test runs of the supplied six IVPs

1.8.2 The REXXMMGR assembler program

A number of the Rexx wrapper WOLA APIs need to pass the address of an address in the parameter list to the actual WOLA API. The REXXMMGR assembler program is used by the Rexx programs to GETMAIN and FREEMAIN storage. This program must be available as a load module on the z/OS LPAR.

1.8.3 Contents of xmitfile-rexxapis

This file contains:

1. the 13 Rexx WOLA wrapper API programs

2. 6 Rexx programs to use to verify operation of the Rexx WOLA wrapper API programs

3. Rexx program RXAECONV which is used by the IVPs to convert ASCII to EBCDIC and EBCDIC to ASCII

It is suggested to receive this file to a dataset called yourUserid.REXX.WOLAAPIS.

1.8.4 Contents of xmitfile-rexxjcl

This file contains:

1. ASMRMMGR – assembles the REXXMMGR load module

2. RUNRWIVP – used to run the IVP’s

It is suggested to receive this file to a dataset called yourUserid.REXX.CNTL.

1.8.5 Contents of xmitfile-rexxload

This file contains:

1. REXXMMGR – used to GETMAIN and FREEMAIN storage

It is suggested to receive this file to a dataset called yourUserid.REXX.LOAD.

1.8.6 Using the xmit files

To get the contents of the supplied xmit files into your z/OS environment you first need to binary FTP the files to your z/OS LPAR.

The following shows the FTP of one of the supplied xmit files:

```
ftp> QUOTE SITE LRECL=80 RECFM=FB BLKSIZE=6160
200 SITE command was accepted
ftp> bin
200 Representation type is Image
ftp> put
Local file xmitfile-rexxapis
Remote file 'edmcar.rexxapis'
200 Port request OK.
125 Storing data set EDMCAR.REXXAPIS
250 Transfer completed successfully.
ftp: 280640 bytes sent in 28.69Seconds 9.78Kbytes/sec.
```

Note that you need to issue the QUOTE command shown above before doing the FTP of the file. The above process will automatically create the target dataset on the z/OS LPAR.

The next step is to use the receive command to extract the contents of the xmit file into a dataset. To do this issue a command similar to this:

```
receive indataset('edmcar.rexxapis')
```

You will then get a response similar to this:

```
INMR901I Dataset EDMCAR.REXX.WOLAAPIS from EDMCAR on WTSCPLX1
INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

Reply to this with the response:

```
dataset('edmcar.rexx.wolaapis')
```

using a high level prefix of your choosing after which you will receive this response:

```
INMR001I Restore successful to dataset 'EDMCAR.REXX.WOLAAPIS'
```

1.9 Verifying the Rexx WOLA Wrapper APIs

IBM ship in WebSphere Application Server sample applications to verify the operation of WOLA. Additionally the WP101490 techdoc on WOLA includes a pdf called ‘The WOLA Native APIs... a COBOL Primer’. That document describes how to perform various examples to show the use of the WOLA APIs.

That techdoc also supplies a sample application that can also be deployed in a WebSphere server on z/OS to further test the functionality of the WOLA APIs to go along with that pdf.

We supply sample Rexx examples that use the Rexx WOLA Wrapper APIs and the above sample code. The six supplied samples and the WOLA APIs they demonstrate the use of are as follows:

Rexx sample name	**Sequence of WOLA APIs used**
RXWIVP1	BBOA1REG, BBOA1INV, BBOA1URG
RXWIVP2	BBOA1REG, BBOA1CNG, BBOA1SRQ, BBOA1RCL, BBOA1GET, BBOA1CNR, BBOA1URG
RXWIVP3	BBOA1REG, BBOA1RCA, BBOA1GET, BBOA1SRP, BBOA1CNR, BBOA1URG
RXWIVP4	BBOA1REG, BBOA1CNG, BBOA1RCS, BBOA1GET, BBOA1SRP, BBOA1CNR, BBOA1URG
RXWIVP5	BBOA1REG, BBOA1SRV, BBOA1SRP, BBOA1CNR, BBOA1URG
RXWIVP6	BBOA1REG, BBOA1RCA, BBOA1GET, BBOA1SRX, BBOA1CNR, BBOA1URG

The following table also shows the verification programs and which WOLA APIs are invoked:

	RXWIVP1	**RXWIVP2**	**RXWIVP3**	**RXWIVP4**	**RXWIVP5**	**RXWIVP6**
BBOA1REG	X	X	X	X	X	X
BBOA1CNG		X		X		

BBOA1SRV					X	
BBOA1INV	X					
BBOA1SRQ		X				
BBOA1RCL		X				
BBOA1RCA			X			X
BBOA1RCS				X		
BBOA1GET		X	X	X		X
BBOA1SRX						X
BBOA1SRP			X	X	X	
BBOA1CNR		X	X	X	X	X
BBOA1URG	X	X	X	X	X	X

1.9.1 Sample IVP output

The zip file that is associated with this techdoc contains sample output from all six IVP jobs, which you can view to see what a successful run of the IVP jobs looks like.

1.9.2 Security

You may need to update your security definitions to allow batch jobs that use the Rexx WOLA wrapper APIs to connect to your WebSphere servers.

This link from the WebSphere Application Server V8.5 infocenter provides related advice on security:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-mp&topic=tdat_security_in

1.9.3 Required preparation

To run RXWIVP1 and RXWIVP2 deploy the supplied OLASample2.ear file into your target WebSphere server on z/OS. This link describes were you can find this ear file:

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-zos&topic=cdat_olasamples

To run RXWIVP3/4/5/6 you will need to download from the WP101490 techdoc the WP101490 Primer.zip file. Unzip this file then deploy the ATSSample1-new.ear into your target WebSphere server on z/OS.

1.9.4 Sample JCL

The following shows the JCL we used to run the supplied sample Rexx example:

```
//RUNRWIVP STEP EXEC PGM=IKJEFT01,REGION=0M
//STEPLIB  DD  DSN=EDMCAR.WMCELL.WOLALOAD,DISP=SHR
//         DD  DSN=EDMCAR.REXX.LOAD,DISP=SHR
//SYSEXEC  DD  DSN=EDMCAR.REXX.WOLAAPIS,DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSDUMP  DD  SYSOUT=*
```

```
//SYSABEND DD  SYSOUT=*
//SYSTSIN  DD  *
 RXWIVP1 -
        WMCELL -
        WMNODEA -
        WMSR01A -
        rwsubjob01
```

On your environment you would need to modify this JCL as follows:

Replace `EDMCAR.WMCELL.WOLALOAD` with name of your dataset that contains the WOLA load modules.

Replace `EDMCAR.REXX.LOAD` with name of your dataset that contains the assembler module used to obtain memory.

Replace `EDMCAR.REXX.WOLAPIS` with name of your dataset that contains the sample Rexx programs.

Replace `WMCELL, WMNODEA, WMSR01A` with values corresponding to the WebSphere cell on your z/OS system.

Optionally replace `rwsubjob01` if you want to use a different name to for registering the WOLA connection.

You change the RXWIVP1 value in the SYSTSIN DD card to specify the IVP Rexx you want to execute.

1.9.5 How to run RXWIVP1 and RXWIVP2

Modify the supplied sample JCL to specify whether to run RXWIVP1 or RXWIVP2 and submit the job.

1.9.6 How to run RXWIVP3/4/5/6

Modify the supplied sample JCL to specify whether to run RXWIVP3, RXWIVP4, RXWIVP5 or RXWIVP6 and submit the job.

These jobs once submitted will all eventually end up in a wait state waiting for a WOLA call from the WebSphere server.

This is where you use the ATSSample1-new.ear application.

In a browser enter a URL similar to this:

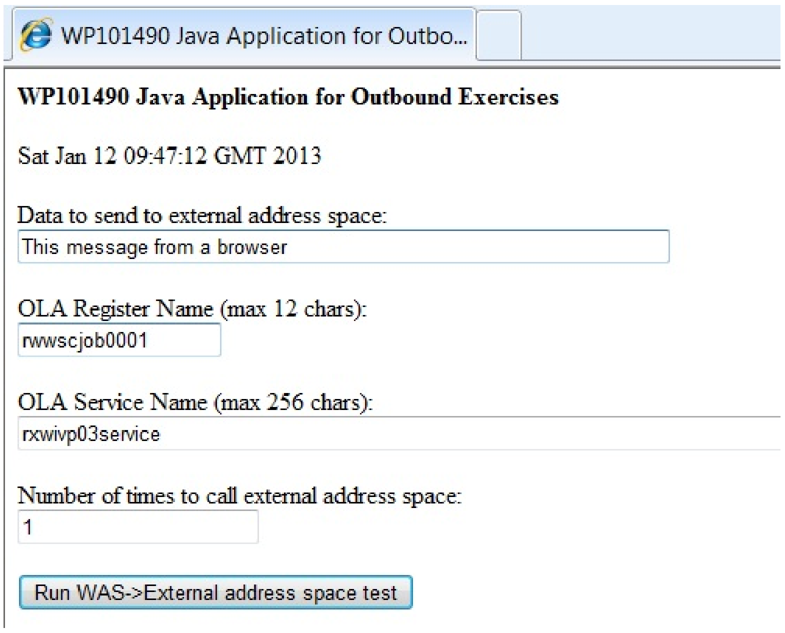
http://wtsc55.itso.ibm.com:17067/ATSSample1Web/

This will display a form in the browser, into which you will need to enter values to allow the application in WebSphere to send data to the waiting batch job.

For the RXWIVP3 example, we entered the following data into the three fields:

Form Field	Value	Description
Data to send to external address space	This message from a browser	Text to send to the batch job
OLA Register Name	rwwscjob001	Name the batch job has registered to the WebSphere
OLA Service Name	rxwivp03service	Name the batch job has used as the service

The following display shows how this looked during our testing:


WP101490 Java Application for Outbo...

WP101490 Java Application for Outbound Exercises

Sat Jan 12 09:47:12 GMT 2013

Data to send to external address space:
This message from a browser

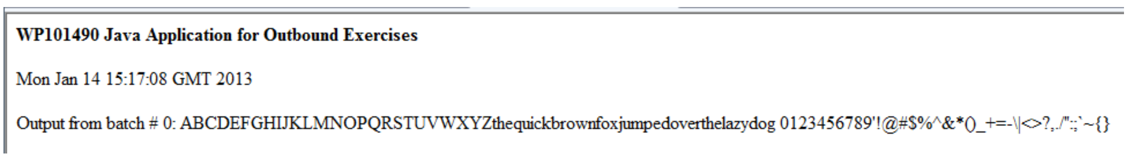
OLA Register Name (max 12 chars):
rwwscjob0001

OLA Service Name (max 256 chars):
rxwivp03service

Number of times to call external address space:
1

Run WAS->External address space test

Once the fields have been filled in click the 'Run WAS->External address space test' button. This should produce display similar to this:



WP101490 Java Application for Outbound Exercises

Mon Jan 14 15:17:08 GMT 2013

```
Output from batch # 0: ABCDEFGHIJKLMNOPQRSTUVWXYZthequickbrownfoxjumpedoverthelazydog 0123456789'!@#$%^&*()_+=-\|<>?,./";:`~{}
```

Note that when you run RXWIVP6 and click the button in the browser you will not get a successful completion. The RXWIVP6 batch job will still complete successfully though. The reason the browser does not get a successful completion message is because RXWIVP6 uses the BBOA1SRX WOLA API to send an exception response back to the application in the WebSphere server.

1.10 Parameters

This section describes the required and optional parameters that can be passed to the Rexx WOLA Wrapper APIs.

The parameter names used when calling the Rexx WOLA Wrapper API's are the same names as used in the WOLA API documentation. Descriptions of these parameters can be found at this link:

> http://www14.software.ibm.com/webapp/wsbroker/redirect?version=matt&product=was-nd-zos&topic=cdat_olaapis

In the following information the Description column will provide additional information if necessary.

1.10.1 RBOA1REG

Parameter	Required	Default value	Description
daemonGroupName	Y	WZCELL	Daemon group name
nodeName	Y	WZNODEA	
serverName	Y	WZSR01A	
registerName	Y	IBMREXX00001	
minConn	N	1	
maxConn	N	1	
registerFlags	N	0	
reg_flag_W2Cprop	N	0	
reg_flag_C2Wprop	N	0	
reg_flag_trcmod	N	0	
reg_flag_trcmore	N	0	
reg_flag_trcsome	N	0	

No additional parameters returned.

1.10.2 RBOA1CNG

Parameter	Required	Default value	Description
registerName	Y	IBMREXX00001	
connectionHandle	N	0	

The connectionHandle obtained is returned as a Hex string.

1.10.3 RBOA1SRV

Parameter	Required	Default value	Description
registerName	Y	IBMREXX00001	
requestServiceName	Y		
requestServiceNameL	N	0	
requestData	N		
requestDataLen	N	0	
connectionHandle	N	0	
waitTime	N	0	

Returns connectionHandle followed by rv followed by requestData.

The connectionHandle and requestData are returned as Hex strings.

1.10.4 RBOA1INV

Parameter	Required	Default value	Description
registerName	Y	IBMREXX00001	
requestType	N	1	
requestServiceName	Y		
requestServiceNameL	N	0	
requestData	N		

requestDataLen	N	0	
responseData	N		
responseDataLen	N	0	
waitTime	N	0	

The responseData obtained is returned as a Hex string.

1.10.5 RBOA1SRQ

Parameter	Required	Default value	Description
connectionHandle	Y	0	
requestType	N	1	
requestServiceName	Y		
requestServiceNameL	N	0	
requestData	N		
requestDataLen	N	0	
async	N	0	
responseDataLen	N	0	

The responseDataLen obtained is returned as a number.

1.10.6 RBOA1RCL

Parameter	Required	Default value	Description
connectionHandle	Y	0	
async	N	0	
responseDataLen	N	0	

The responseDataLen obtained is returned as a number.

1.10.7 RBOA1RCA

Parameter	Required	Default value	Description
registerName	Y		
requestServiceName	Y		
requestServiceNameL	N	0	
requestDataLen	N	0	
waitTime	N	0	

Returns requestDataLen followed by connectionHandle.

The connectionHandle obtained is returned as a Hex string.

1.10.8 RBOA1RCS

Parameter	Required	Default value	Description
connectionHandle	Y	0	Value must be supplied as Hex string
requestServiceName	Y		
requestServiceNameL	N	0	
requestDataLen	N	0	
async	N	0	

The requestDataLen obtained is returned as a number.

1.10.9 RBOA1GET

Parameter	Required	Default value	Description

connectionHandle	Y	0	Value must be supplied as Hex string
responseData	N		
msgDataLen	Y	0	

The responseData obtained is returned as a Hex string.

1.10.10 RBOA1SRX

Parameter	**Required**	**Default value**	**Description**
connectionHandle	Y	0	Value must be supplied as Hex string
excresponseData	Y		
excresponseDataLen	N	0	

No additional parameters returned.

1.10.11 RBOA1SRP

Parameter	**Required**	**Default value**	**Description**
connectionHandle	Y	0	Value must be supplied as Hex string
responseData	Y		
responseDataLen	N	0	

No additional parameters returned.

1.10.12 RBOA1CNR

Parameter	**Required**	**Default value**	**Description**

connectionHandle	Y	0	Value must be supplied as a Hex string

No additional parameters returned.

1.10.13 RBOA1URG

Parameter	Required	Default value	Description
registerName	Y	IBMREXX00001	
force	N	0	
unregFlags	N	0	

No additional parameters returned.

*** End of Document ***