webMethods EntireX

RPC-ACI Bridge

Version 9.5 SP1

November 2013

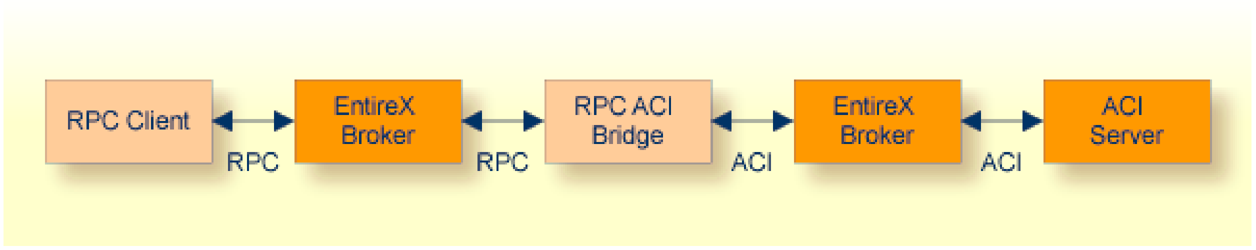
Document ID: EXX-ACI-95SP1-20140628BRIDGE

Table of Contents

EntireX RPC-ACI Bridge

1 Introduction to the EntireX RPC-ACI Bridge

The EntireX RPC-ACI Bridge allows standard RPC clients to communicate with an ACI server. The RPC-ACI Bridge transforms the RPCs from the clients into ACI messages. The RPC-ACI Bridge acts on one side as an RPC server and on the other side as an ACI client. In this documentation we distinguish between the Broker for RPC, which sends the RPCs from the client to the server side of the RPC-ACI Bridge and the Broker for ACI, which sends the messages to the ACI server. These two brokers can be the same instance. Use distinct services for the RPCs and ACI.



The RPC-ACI Bridge can connect to ACI servers in any language. We describe the use of Natural and COBOL ACI servers. For existing COBOL programs you can use the COBOL IDL Generator to generate the IDL file for the RPC clients.

The RPC-ACI Bridge supports RPC clients in different programming languages.

2 Administering EntireX RPC-ACI Bridge

The RPC-ACI Bridge enables RPC-based client applications to be used with ACI servers.

Customizing the RPC-ACI Bridge

For the setup of the RPC-ACI Bridge there are

- a configuration file and
- scripts to start the RPC-ACI Bridge.

Location of the RPC-ACI Bridge

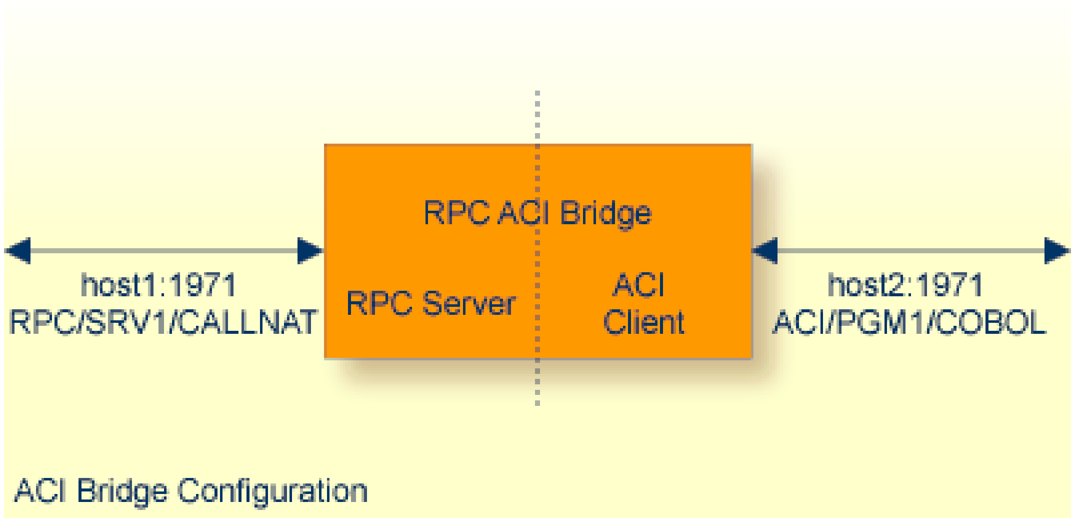
The RPC-ACI Bridge is contained in the file *entirex.jar*.

The Configuration File

The default name of the configuration file is *entirex.rpcacibridge.properties*. The RPC-ACI Bridge searches for this file in the current working directory.

You can set the name of the configuration file with `-Dentirex.server.properties=<your file name>` with "/" as file separator.

The configuration file contains the configuration for both parts of the RPC-ACI Bridge.



To set up the RPC-ACI Bridge

1. Use the RPC server agent of the System Management Hub.
2. Add the RPC-ACI Bridge as an RPC server.

See *Administering the EntireX RPC Servers using System Management Hub* in the UNIX and Windows administration documentation for details.

Or:

Use the scripts to start the RPC-ACI Bridge.

Under Windows use *jrpcacibridge.bat* in the folder *bin* to start the RPC-ACI Bridge. You may customize this file.

Under UNIX use *jrpcacibridge.bsh* in the folder *bin* to start the RPC-ACI Bridge. You may customize this file.

Both scripts use the configuration file *entirex.rpcacibridge.properties* in the folder *etc*.

Configuring more than one RPC-ACI Bridge

If you configure more than one RPC-ACI Bridge that connect to the same EntireX Broker, the following items must be distinct:

- The user for the ACI client side (property `entirex.rpcacibridge.userid`).
- The trace output file (property `entirex.server.logfile`).
- The monitor port for SMH (property `entirex.server.monitorport`).
- The log for the Windows Service (property `entirex.server.serverlog`).
- The trace output file of the SMH agent for RPC servers.

Configuring the RPC Server Side

The RPC server side of the RPC-ACI Bridge is configured like the Java RPC Server. The RPC-ACI Bridge uses the properties that start with “entirex.server”.

The RPC server side can adjust the number of worker threads to the number of parallel requests. Use the properties `entirex.server.fixedservers`, `entirex.server.maxservers`, `entirex.server.minservers` to configure this scalability. If `entirex.server.fixedservers=yes`, the number of `entirex.server.minservers` is started and the server can process this number of parallel requests. If `entirex.server.fixedservers=no`, the number of worker threads balances between `entirex.server.minservers` and `entirex.server.maxservers`. This is done by a so-called attach server thread. On startup, the number of worker threads is `entirex.server.minservers`. If more than `entirex.server.minservers` are waiting for requests, a worker thread stops if its receive call times out. The timeout period is configured with `entirex.server.waitserver`.

Alternatively to the properties, you can use the command-line option. The command-line options have a higher priority than the properties set as Java system properties and these have higher priority than the properties in the configuration file.

Name	Command-line Option	Default Value	Explanation
`entirex.server. brokerid`	`-broker`	`localhost : 1971`	Broker ID
`entirex.server. codepage`	`-codepage`		The codepage the server uses. Permitted values are the name of the codepages the JVM supports. Use the value LOCAL wh the default codepage of the JVM should used. See *Internationalization with EntireX* details.
`entirex.server. compresslevel`	`-compresslevel`	0 (no compression)	`BEST_COMPRESSION` 9 `BEST_SPEED` 1 `DEFAULT_COMPRESSION` `-1, mapped 6` `DEFLATED` 8 `NO_COMPRESSION` 0 `N` 0 `Y` 8
`entirex.server. encryptionlevel`	`-encryption`	`0`	Encryption level (if Broker is version 6.1. or higher. Valid values: 0,1,2).

Name	Command-line Option	Default Value	Explanation
entirex.server. environment			Can be used in a user-written tra exit of the broker. See BrokerService, setEnvironment(java.lang. (EntireX Java ACI) in the Javado documentation of the Java ACI.
entirex.server. fixedservers		no	If "no", use an attach server threa manage worker threads, otherwi minimum number of server thre properties entirex.server.ma entirex.server.minservers
entirex.server. logfile	-logfile		Path and name of the trace outpu
entirex.server. maxservers		32	Maximum number of worker thr
entirex.server. minservers		1	Minimum number of server thre
entirex.server. monitorport	-smhport	0	The port where the server listens commands from the System Ma Hub (SMH). If this port is 0, no p and the management by the SM disabled.
entirex.server.name			The name of the server.
entirex.server. password	-password		The password for secured access Broker. For Java 1.4 and above, the passw encrypted and written to the pro entirex.server.password.e To change the password, set the password in the properties file (*entirex.rpcacibridge.properties*). To disable password encryption entirex.server.passworden Default for this property is "yes" For Java 1.3 and below, the passw encryption is not available.
entirex.server. properties	-propertyfile	entirex.rpcacibridge.properties	The file name of the property fil
entirex.server. restartcycles	-restartcycles	15	Number of restart attempts if th not available. This can be used t Java RPC Server running while t is down for a short time.
entirex.server.	-security	no	no/yes/auto/Name of BrokerSecu

Name	Command-line Option	Default Value	Explanation
security			
entirex.server. serveraddress	-server	RPC/SRV1/CALLNAT	Server address
entirex.server. serverlog	-serverlog		Name of the file where start and stop of worker threads is logged. Used by the Windows RPC Service.
entirex.server. userid	-user	JavaServer	The user ID of the Broker for RPC. See entirex.server.password.
entirex.server. verbose	-verbose	no	Verbose output to standard output yes/n
entirex.server. waitattach		600S	Wait timeout for the attach server thread
entirex.server. waitserver		300S	Wait timeout for the worker threads.
entirex.timeout		20	TCP/IP transport timeout. See *Setting the Transport Timeout* under *Writing Advance Applications - EntireX Java ACI*.
entirex.trace	-trace	0	Trace level (1,2,3).

Configuring the ACI Client Side

These properties are used to configure the connection to the Broker for ACI.

Alternatively, you can use the command-line option. The command-line options have a higher priority than the properties set as Java system properties and these have higher priority than the properties in the configuration file

Name	Command-line Option	Default Value	Explanation
`entirex.rpcacibridge.brokerid`	`-acibroker`	`localhost`	Broker ID of the Broker for ACI.
`entirex.rpcacibridge.compresslevel`	`-acicompresslevel`	`0` (no compression)	Permitted values (you can enter the text or the numeric value): `BEST_COMPRESSION` 9 `BEST_SPEED` 1 `DEFAULT_COMPRESSION` `-1`, mapped to `6` `DEFLATED` 8 `NO_COMPRESSION` 0 `Y` 8
`entirex.rpcacibridge.encryptionlevel`	`-aciencryption`	`0`	Encryption level (if Broker is version 6.1.1 or higher. Valid values: 0,1,2).
`entirex.rpcacibridge.marshalling`	`-acimarshalling`		This is for arrays of groups. Set this property to "cobol" if the ACI server is a COBOL program. Set this property to "natural" if the ACI server is a Natural program. Default is " ", which lets the RPC client determine the marshalling.
`entirex.rpcacibridge.password`	`-acipassword`		The password of the Broker for ACI. For Java 1.4, the password is encrypted and written to the property entirex.server.password.e. To change the password, set the new password in the properties file (default is *entirex.rpcacibridge.properties*). To disable password encryption set `entirex.server.passwordencrypt=no`. Default for this property is "yes". For Java 1.3 and below, the password encryption is not available.
`entirex.rpcacibridge.`	`-acisecurity`	`no`	no/yes/auto/Name of BrokerSecurity object.

Name	Command-line Option	Default Value	Explanation
`security`			
`entirex.rpcacibridge.serveraddress`	`-aciserver`	`ACLASS/ASERVER/ASERVICE`	Server Address for the Broker for ACI.
`entirex.rpcacibridge.trace`	`-acitrace`	`No`	If set to "yes", additional trace output (exception stack-traces and request and reply buffers) is generated.
`entirex.rpcacibridge.userid`	`-aciuser`	Value of system property user.name .	The user ID of the Broker for ACI. Use different user IDs for different RPC-ACI Bridges on the same Broker.
`entirex.rpcacibridge.waittime`		`0S`	The wait time for receive requests. Permitted values are *`nS`*|*`nM`*|*`nH`*, where *`n`* is the number of seconds or minutes or hours.

Starting the RPC-ACI Bridge

To start the RPC-ACI Bridge

- Use the script *jrpcacibridge* in the folder *bin* to start the RPC-ACI Bridge. You may customize this file.

  Or:

  Use the RPC server agent in the System Management Hub to configure and start the RPC-ACI Bridge.

  See *Administering the EntireX RPC Servers using System Management Hub* in the UNIX and Windows administration documentation for details.

On Windows you can start the RPC-ACI Bridge as a Windows Service. The installation of the service is similar to the installation of the Java RPC Server. See *Running the Java RPC Server as a Windows Service* under under *Administering the EntireX Java RPC Server* in the Windows administration documentation in the Windows administration documentation.

Stopping the RPC-ACI Bridge

To stop the RPC-ACI Bridge

- Use the RPC server agent in the SMH to stop the RPC-ACI Bridge.

  Or:

  Use the agent for the Broker. Use `Deregister` on the service, specified with the property `entirex.server.serveraddress`.

Application Identification

The application identification is sent from the RPC-ACI Bridge to the Broker. It is visible with Broker Command and Info Services.

The identification consists of four parts: name, node, type, and version. These four parts are sent with each Broker call and are visible in the trace information.

For the RPC-ACI Bridge these values are:

Application name:	`ANAME=RPC-ACI Bridge`
Node name:	`ANODE=`*`<host name>`*
Application type:	`ATYPE=Java`
Version:	`AVERS=9.0.0.0`

3 Writing ACI Servers for the RPC-ACI Bridge in COBOL

The RPC-ACI Bridge is prepared for ACI servers written in COBOL.

Tasks

The RPC-ACI Bridge is prepared for ACI servers written in COBOL.

Writing an ACI server consists of two tasks:

- implement the Broker calls
- implement the processing of the received buffer and the response for the send buffer

Using Arrays of Groups

If your programs use arrays of groups, you have to adjust the marshalling.

To adjust the marshalling for arrays of groups

1. Use the property `entirex.rpcacibridge.marshalling` for the configuration.
2. Set the property to "cobol".

If your programs do not use arrays of groups, you do not need to set `entirex.rpcacibridge.marshalling`.

Data Types

Data Type	Format	Remarks
A`<number>` Alphanumeric	`<number>` bytes, encoding the characters	
AV[*number*] Alphanumeric variable length with maximum length	Bytes up to the end of the buffer, maximal length `<number>`	Only as last value
K`<number>` Kanji	Same as data type A	
KV[*number*] Kanji variable length with maximum length	Same as data type AV[*number*]	Only as last value
I1 Integer (small)	Sign (+, -) and 3 bytes (digits)	
I2 Integer (medium)	Sign (+, -) and 5 bytes (digits)	
I4 Integer (large)	Sign (+, -) and 10 bytes (digits)	
N`<number1>`[.*number2*] Unpacked decimal	Sign (+, -), `<number1>` bytes (digits) [*number2*] bytes (digits), no decimal point.	
NU`<number1>`[.*number2*] Unpacked decimal unsigned	`<number1>` bytes (digits) [*number2*] bytes (digits), no decimal point.	
P`<number1>`[.*number2*] Packed decimal	Sign (+, -), `<number1>` bytes (digits) [*number2*] bytes (digits), no decimal point.	
PU`<number1>`[.*number2*] Packed decimal unsigned	`<number1>` bytes (digits) [*number2*] bytes (digits), no decimal point.	
L Logical	1 Byte: X for true, all other false	
D Date	YYYYMMDD	YYYY year, MM month, DD day
T Time	YYYYMMDDhhmmssS	YYYY year, MM month, DD day, hh hour, mm minute, ss second, S tenth of a second.

Data Types not supported:

- Binary (B[n],BV, BV[n])
- Floating point (F4, F8)

Declaring the Variables for the Data Types

This section describes how to declare the variables for the data types.

Use these declarations to map the receive buffer and the send buffer to variables.

Data Type	Declaration and Marshalling
A<*number*> Alphanumeric	Declaration for receive and send buffer: PIC X(n)
AV[*number*] Alphanumeric variable length with maximum length	Declaration for receive and send buffer: PIC X(n)
K<*number*> Kanji	Declaration for receive and send buffer: PIC X(n)
KV[*number*] Kanji variable length with maximum length	Declaration for receive and send buffer: PIC X(n)
I1 Integer (small)	Declaration for receive and send buffer: PIC S9(3)
I2 Integer (medium)	Declaration for receive and send buffer: PIC S9(5)
I4 Integer (large)	Declaration for receive and send buffer: PIC S9(10)
N<*number1*>[.*number2*] Unpacked decimal	Declaration for receive and send buffer: PIC S9(*number1*)V(*number2*) SIGN LEADING SEPARATE
NU<*number1*>[.*number2*] Unsigned unpacked decimal	Declaration for receive and send buffer: PIC 9(*number1*)V(*number2*)
P<*number1*>[.*number2*] Packed decimal	Declaration for receive and send buffer: PIC S9(*number1*)V(*number2*) SIGN LEADING SEPARATE Declare local variable PIC S9(*number1*)V(*number2*) PACKED DECIMAL Move from receive buffer to local variable before computation and from local variable to send buffer afterwards.
PU<*number1*>[.*number2*] Unsigned packed decimal	Declaration for receive and send buffer: PIC 9(*number1*)V(*number2*)Declare local variable PIC 9(*number1*)V(*number2*) PACKED DECIMAL Move from receive buffer to local variable before computation and from local variable to send buffer afterwards.
L Logical	Declaration for receive and send buffer: PIC X(1)
D Date	Declaration for receive and send buffer: PIC X(8)
T Time	Declaration for receive and send buffer: PIC X(15)

4 Writing ACI Servers for the RPC-ACI Bridge in Natural

The RPC-ACI Bridge is prepared for ACI servers written in Natural.

Tasks

Writing an ACI server consists of two tasks:

- implement the Broker calls
- implement the processing of the received buffer and the response for the send buffer

Using Arrays of Groups

If your programs use arrays of groups, you have to adjust the marshalling.

To adjust the marshalling for arrays of groups

1. Use the property `entirex.rpcacibridge.marshalling` for the configuration.
2. Set the property to "natural".

If your programs do not use arrays of groups, you do not need to set `entirex.rpcacibridge.marshalling`.

Data Types

Data Type	Description	Format	Note
A*number*	Alphanumeric	*number* bytes, encoding the characters.	
AV	Alphanumeric variable length	Bytes up to the end of the buffer.	1
AV[*number*]	Alphanumeric variable length with maximum length	Bytes up to the end of the buffer, maximal length *number*.	1
K*number*	Kanji	Same as data type A.	
KV	Kanji variable length	Same as data type AV.	1
KV[*number*]	Kanji variable length with maximum length	Same as data type AV[*number*].	1
I1	Integer (small)	*sign* (+, -) and 3 bytes (digits).	
I2	Integer (medium)	*sign* (+, -) and 5 bytes (digits).	
I4	Integer (large)	*sign* (+, -) and 10 bytes (digits).	
N*number1*[.*number2*]	Unpacked decimal	*sign* (+, -), *number1* bytes (digits) [*number2*] bytes (digits), no decimal point.	
P*number1*[.*number2*]	Packed decimal	*sign* (+, -), *number1* bytes (digits) [*number2*] bytes (digits), no decimal point.	
L	Logical	1 byte: X for true, all other false.	
D	Date	*YYYYMMDD*.	2
T	Time	*YYYYMMDDhhmmssS*.	3

Notes:

1. Only as last value.
2. *YYYY* year, *MM* month, *DD* day.
3. *YYYY* year, *MM month*, *DD* day, *hh* hour, *mm* minute, *ss* second, *S* tenth of a second.

Data Types not supported:

- Binary (B[*n*],BV, BV[*n*])
- Floating point (F4, F8)

Declaring the Variables for the Data Types

This section describes how to declare the variables for the data types. Use these declarations to map the receive buffer and the send buffer to variables. For some data types, the values have to be moved to a local variable before computation.

Example:

```
* Declaration
DEFINE DATA LOCAL
1 PNUMERIC (A012)
1 #NUMERIC (N8.3)
1 REDEFINE #NUMERIC
2 #NUMERIC1 (N11)
* Computation
    MOVE EDITED RCVE-DATA.PNUMERIC TO #NUMERIC1 (EM=S9(11))
    #NUMERIC := #NUMERIC + 1
    MOVE EDITED #NUMERIC1 (EM=S9(11)) to SEND-DATA.PNUMERIC
```

Data Type	Declaration and Marshalling
A`<number>` Alphanumeric	Declaration for receive and send buffer: (An)
AV Alphanumeric variable length	Declaration for receive and send buffer: (A) DYNAMIC
AV[*number*] Alphanumeric variable length with maximum length	Declaration for receive and send buffer: (A) DYNAMIC
K`<number>` Kanji	Declaration for receive and send buffer: (An)
KV Kanji variable length	Declaration for receive and send buffer: (A) DYNAMIC
KV[*number*] Kanji variable length with maximum length	Declaration for receive and send buffer: (A) DYNAMIC
I1 Integer (small)	Declaration for receive and send buffer: (A4)MOVE EDITED to I1 variable with (EM=S9(3))
I2 Integer (medium)	Declaration for receive and send buffer: (A6)MOVE EDITED to I2 variable with (EM=S9(5))
I4 Integer (large)	Declaration for receive and send buffer: (A11)MOVE EDITED to I4 variable with (EM=S9(10))
N`<number1>`[.*number2*] Unpacked decimal	Declaration for receive and send buffer: (An), where n = *number1* + *number2* + 1 (one byte for the sign). Redefine N*number1*+*number2* variable as N*number1*.*number2* variable. MOVE EDITED to N*number1*+*number2* variable with (EM=S9(*number1* + *number2*))

Data Type	Declaration and Marshalling
P<*number1*>[.*number2*] Packed decimal	Declaration for receive and send buffer: (An), where n = *number1* + *number2* + 1 (one byte for the sign). Redefine P*number1*+*number2* variable as P*number1*.*number2* variable. MOVE EDITED to P*number1*+*number2* variable with (EM=S9(*number1* + *number2*))
L Logical	Declaration for receive and send buffer: (A1)
D Date	Declaration for receive and send buffer: (A8)MOVE EDITED to Date variable with (EM=YYYYMMDD)
T Time	Declaration for receive and send buffer: (A15)MOVE EDITED to Time variable with (EM=YYYYMMDDHHIISST)

5 Writing RPC Clients for the RPC-ACI Bridge with the C Wrapper

The RPC-ACI Bridge enables RPC-based client applications to be used with ACI servers.

To write a C client

- Follow the instructions under *Using the C Wrapper for the Client Side*.

The RPC-ACI Bridge reports errors from the RPC server side and the ACI side to the RPC clients. Errors from the ACI side include errors by the Broker for ACI.

The RPC-ACI Bridge reports the same error classes and error codes for the RPC server side as the Java RPC Server. The RPC-ACI Bridge reports errors of the ACI side in a client-specific way as error 10010007 (internal error of the RPC protocol). The detailed message of the error has the form `RPCACIBridge: < text >`, where `text` indicates the cause of the error. See *Message Class 1018 - EntireX RPC-ACI Bridge* under *Error Messages and Codes* for additional information.

6 Writing RPC Clients for the RPC-ACI Bridge in Java

The RPC-ACI Bridge enables RPC-based client applications to be used with ACI servers.

The EntireX RPC-ACI Bridge reports errors from the RPC server side and the ACI side to the RPC clients. Errors from the ACI side include errors by the Broker for ACI. The RPC-ACI Bridge reports the same error classes and error codes for the RPC server side as the XML/SOAP RPC Server. The RPC-ACI Bridge reports errors of the ACI side in a client-specific way as described below.

To write a Java client

1. Generate the Java RPC client stub from the IDL file as described in *Using the Java Wrapper*.
2. Implement the client with this stub.

All errors are reported as `BrokerExceptions`. Errors on the ACI side of the RPC-ACI Bridge are `BrokerExceptions` in class 1018. See *Message Class 1018 - EntireX RPC-ACI Bridge* under *Error Messages and Codes*.