software AG

webMethods EntireX

EntireX RPC-ACI Bridge

Version 10.5

October 2019

WEBMETHODS

This document applies to webMethods EntireX Version 10.5 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: EXX-ACI-105-20220422BRIDGE

Table of Contents

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Introduction to the EntireX RPC-ACI Bridge

Overview

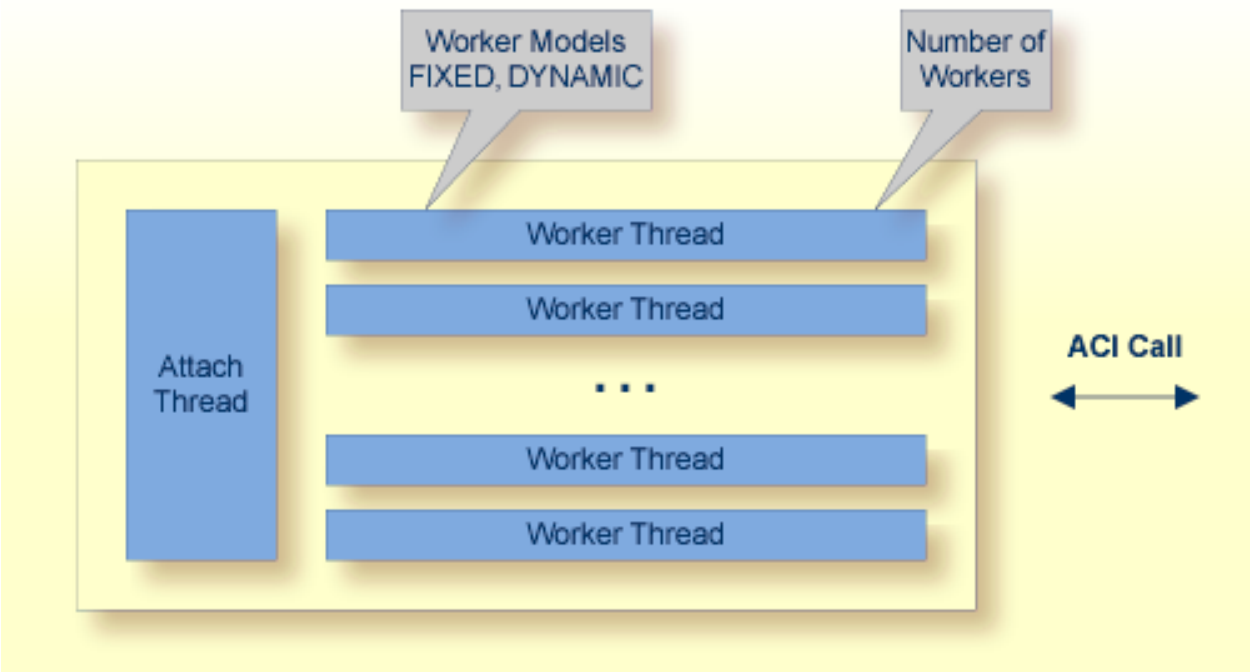
The RPC-ACI Bridge acts on one side as an RPC server and on the other side as an ACI client. In this documentation we distinguish between the Broker for RPC, which sends the RPCs from the client to the server side of the RPC-ACI Bridge and the Broker for ACI, which sends the messages to the ACI server. These two brokers can be the same instance. Use distinct `CLASS/SERVER/SERVICE` broker attributes for the RPC requests and ACI messages.

The RPC-ACI Bridge can connect to ACI servers written in programming languages where the ACI is supported, such as Natural | Assembler | C | COBOL | Java | PL/I. More detailed information is given on writing ACI servers for the RPC-ACI Bridge in **COBOL** and **Natural**.

For existing server programs, use an extractor to generate the *Software AG IDL File* in the IDL Editor documentation for the RPC clients.

- For COBOL, use the IDL Extractor for COBOL
- For PL/I, use the IDL Extractor for PL/I

Worker Models



RPC requests are worked off inside the RPC server in worker threads. Every RPC request occupies during its processing a worker thread. If you are using RPC conversations, each RPC conversation requires its own thread during the lifetime of the conversation. The RPC-ACI Bridge can adjust the number of worker threads to the number of parallel requests. The RPC-ACI Bridge provides two worker models:

- `FIXED`
  The *fixed* model creates a fixed number of worker threads. The number of worker threads does not increase or decrease during the lifetime of an RPC server instance.
- `DYNAMIC`
  The *dynamic* model creates worker threads depending on the incoming load of RPC requests.

For configuration and technical details, see property `entirex.server.fixedservers` under *Administering the RPC-ACI Bridge*.

3 Administering the RPC-ACI Bridge

The EntireX RPC-ACI Bridge allows standard RPC clients to communicate with an ACI server. The RPC-ACI Bridge transforms RPC requests from clients into ACI messages.

Customizing the RPC Server

The following elements are used to set up the RPC-ACI Bridge:

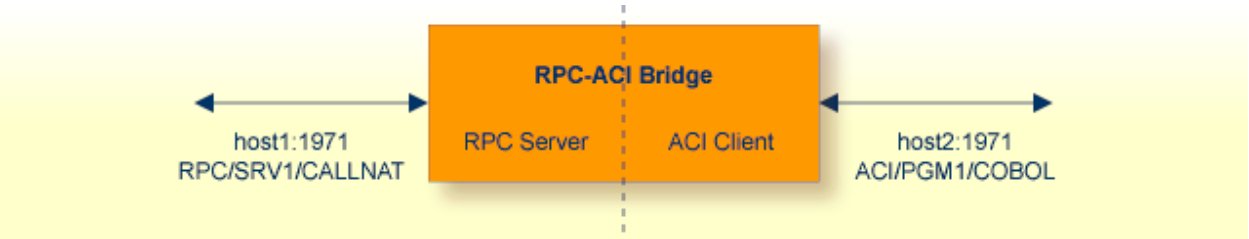
- Configuration File
- Start Script

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.



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 log for the Windows Service (property `entirex.server.serverlog`).

Start Script

The start script for the RPC-ACI Bridge is called *jrpcacibridge.bsh* (UNIX) or *jrpcacibridge.bat* (Windows) and is provided in the *bin* folder of the installation directory. You may customize this file. The RPC-ACI Bridge itself is contained in the file *entirex.jar*.

Configuring the RPC Server Side

The RPC-ACI Bridge uses the properties that start with "`entirex.server`" for configuring the RPC server side.

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

Property Name	Command-line Option	Default	Explanation
`entirex.server.brokerid`	`-broker`	`localhost : 1971`	Broker ID. See *URL-style Broker ID* in the EntireX Broker ACI Programming documentation.
`entirex.server.codepage`	`-codepage`		Specify the encoding which corresponds to the ACI server the RPC-ACI Bridge is talking to. The codepage is used for both sides of the communication: When communicating as an RPC server to receive incoming requests from RPC clients and as ACI client to transfer them to the ACI server. Enable character conversion in the broker by setting the service-specific attribute `CONVERSION` to "`SAGTRPC`". See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. More information can be found under *Internationalization with EntireX*.
`entirex.server.compresslevel`	`-compresslevel`	0 (no compression)	`BEST_COMPRESSION` 9 `BEST_SPEED` 1 `DEFAULT_COMPRESSION` -1 (*mapped to* 6) `DEFLATED` 8 `NO_COMPRESSION` 0 `N` 0 `Y` 8
`entirex.server.fixedservers`		`no`	`NO` The number of worker threads balances between what is specified in `entirex.server.minservers` and

Property Name	Command-line Option	Default	Explanation
			what is specified in `entirex.server.maxservers`. This is done by a so-called attach thread. At startup, the number of worker threads is the number specified in `entirex.server.minservers`. A new worker thread starts if the broker has more requests than there are worker threads waiting. If more than the number specified in `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`. See worker model `DYNAMIC`. `YES` The number of worker threads specified in `entirex.server.minservers` is started and the server can process this number of parallel requests. See worker model `FIXED`.
	`-help`		Display usage of the command-line parameters.
`entirex.server.logfile`	`-logfile`		Path and name of the trace output file.
`entirex.server.maxservers`		`32`	Maximum number of worker threads.
`entirex.server.minservers`		`1`	Minimum number of server threads.
`entirex.server.name`			The name of the server.
`entirex.server.password`	`-password`		The password for secured access to the broker. 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. To disable password encryption, set `entirex.server.passwordencrypt=no`. Default: `yes`.
`entirex.server.properties`	`-propertyfile`	`entirex.rpcacibridge.properties`	The file name of the property file.

Property Name	Command-line Option	Default	Explanation
entirex.server.restartcycles	-restartcycles	15	Number of restart attempts if the Broker is not available. This can be used to keep the RPC-ACI Bridge running while the Broker is down for a short time.
entirex.server.security	-security	no	no/yes/auto/Name of BrokerSecurity object.
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 for access to the broker.
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 Advanced 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. See *URL-style Broker ID*.

`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.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.
`entirex.rpcacibridge.password`	`-acipassword`		The password of the Broker for ACI. 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: `yes`.
`entirex.rpcacibridge.security`	`-acisecurity`	`auto`	no/yes/auto/Name of BrokerSecurity object.
`entirex.rpcacibridge.serveraddress`	`-aciserver`	`ACLASS/ASERVER/ASERVICE`	Server Address for the Broker for ACI.
`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`		`60S`	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 ***Start Script***.

  Or:

  Under Windows you can use the RPC-ACI Bridge as a Windows Service. See ***Running the RPC-ACI Bridge as a Windows Service***.

Stopping the RPC-ACI Bridge

To stop the RPC-ACI Bridge

- Use the command `stopService`. See *Stop Running Services* in Command Central's Command-line Interface.

  Or:

  Stop the service using Command Central's Graphical User Interface. See *Stopping a Service*.

  Or:

  Use the command-line utility `etbcmd`. See `ETBCMD` under *Broker Command-line Utilities* in the platform-specific Administration documentation.

  Or:

  Use `CTRL-C` in the session where you started the RPC server instance.

  Or:

  Under UNIX, enter command `kill -process-id`.

Using SSL/TLS with the RPC-ACI Bridge

To use SSL with RPC-ACI Bridge, you need to configure two sides, the RPC server side and the ACI client side.

- **For the ACI client side**
  ACI applications can use Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport medium. The term "SSL" in this section refers to both SSL and TLS. ACI-based clients or servers are always SSL clients. The SSL server can be either the EntireX Broker or the Broker SSL Agent. For an introduction see *SSL/TLS and Certificates with EntireX* in the Platform-independent Administration documentation.
- **For the RPC server side**
  The same is true for the RPC server side. Additionally, Direct RPC in webMethods Integration Server (IS inbound) can be used as the SSL server.

To set up SSL

1 To operate with SSL, certificates need to be provided and maintained. Depending on the platform, Software AG provides default certificates, but we strongly recommend that you create your own. See *SSL/TLS Sample Certificates Delivered with EntireX* in the EntireX Security documentation.

2 Set up the ACI side and RPC side for an SSL connection.

For both sides, use the *URL-style Broker ID* with protocol `ssl://` for the Broker ID. If no port number is specified, port 1958 is used as default. Example:

```
ssl://localhost:22101?trust_store=C:\SoftwareAG\EntireX\etc\ExxCACert.jks&verify_server=no
```

If the SSL client checks the validity of the SSL server only, this is known as *one-way SSL*. The mandatory `trust_store` parameter specifies the file name of a keystore that must contain the list of trusted certificate authorities for the certificate of the SSL server. By default a check is made that the certificate of the SSL server is issued for the hostname specified in the Broker ID. The common name of the subject entry in the server's certificate is checked against the hostname. If they do not match, the connection will be refused. You can disable this check with SSL parameter `verify_server=no`.

If the SSL server additionally checks the identity of the SSL client, this is known as *two-way SSL*. In this case the SSL server requests a client certificate (the parameter `verify_client=yes` is defined in the configuration of the SSL server). Two additional SSL parameters must be specified on the SSL client side: `key_store` and `key_passwd`. This keystore must contain the private key of the SSL client. The password that protects the private key is specified with `key_passwd`.

The ampersand (&) character cannot appear in the password.

SSL parameters are separated by ampersand (&). See also *SSL/TLS Parameters for SSL Clients*.

3 Make sure the SSL server to which the ACI side connects is prepared for SSL connections as well. The SSL server can be EntireX Broker or Broker SSL Agent. See:

- *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
- Broker SSL Agent in the UNIX and Windows Administration documentation

4 Make sure the SSL server to which the RPC side connects is prepared for SSL connections as well. The SSL server can be EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). See:

- *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
- Broker SSL Agent in the UNIX and Windows Administration documentation
- *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

Running the RPC-ACI Bridge as a Windows Service

For general information see *Running an EntireX RPC Server as a Windows Service*.

To run the RPC-ACI Bridge as a Windows Service

1 Customize the ***Start Script*** according to your system installation.

Note: The script must pass external parameters to the RPC server and use the reduced signaling of the JVM (option `-Xrs`):

```
java -Xrs com.softwareag.entirex.rpcacibridge.RPCACIBridge %*
```

If `-Xrs` is not used, the JVM stops and an entry 10164002 is written to the event log when the user logs off from Windows.

See also ***Starting the RPC Server***.

2 Test your RPC server to see whether it will start if you run your script file.

3 Use the *EntireX RPC Service Tool* and install the `RPCService` with some meaningful extension, for example `MyServer`. If your ***Start Script*** is *jrpcacibridge.bat*, the command will be

```
RPCService -install -ext MyServer ↩
-script install_path\EntireX\bin\jrpcacibridge.bat
```

The log file will be called *RPCservice_MyServer.log*.

4 In **Windows Services** menu (**Control Panel** > **Administrative Tools** > **Services**) select the service: `Software AG EntireX RPC Service [MyServer]` and change the property `Startup Type` from "Manual" to "Automatic".

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=10.5.0.0`

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

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

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 "cobol".

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, maximum 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.	
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*.	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.

Data Type	Description	Declaration and Marshalling
A*number*	Alphanumeric	Declaration for receive and send buffer: `PIC X(n)`
AV	Alphanumeric variable length	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	Kanji variable length	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*]	Unpacked decimal unsigned	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*]	Packed decimal unsigned	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)`

Data Type	Description	Declaration and Marshalling
T	Time	Declaration for receive and send buffer: `PIC X(15)`

5 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, maximum 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	Description	Declaration and Marshalling
A*number*	Alphanumeric	Declaration for receive and send buffer: (A*n*)
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: (A*n*)
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: (A*n*), 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	Description	Declaration and Marshalling
P*number1*[.*number2*]	Packed decimal	Declaration for receive and send buffer: (A*n*), 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)

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

The EntireX RPC-ACI Bridge allows standard RPC clients to communicate with an ACI server. The RPC-ACI Bridge transforms RPC requests from clients into ACI messages.

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 RPC Server for Java. 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* for additional information.

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

The EntireX RPC-ACI Bridge allows standard RPC clients to communicate with an ACI server. The RPC-ACI Bridge transforms RPC requests from clients into ACI messages.

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 RPC Server for XML/SOAP. 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*.