# webMethods EntireX

## EntireX RPC Programming

Version 9.5 SP1

November 2013

This document applies to webMethods EntireX Version 9.5 SP1.

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



**Document ID: EXX-RPC-95SP1-20140628**

# Table of Contents

# 1 Introduction to RPC Programming

# RPC Technology

A Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. (A procedure call is also sometimes known as a function call or a subroutine call.)

RPC uses the client/server model. The requesting program is a client and the service-providing program is the server. Like a regular or local procedure call, an RPC is a synchronous operation requiring the requesting program to be suspended until the results of the remote procedure are returned. However, the use of lightweight processes or threads that share the same address space allows multiple RPCs to be performed concurrently.

When program statements that use RPC are compiled into an executable program, an interface object is included in the compiled code that acts as the representative of the remote procedure code. When the program is run and the procedure call is issued, the interface object receives the request and forwards it to a client runtime program in the local computer. The client runtime program has the knowledge of how to address the remote computer and server application and sends the message across the network that requests the remote procedure. Similarly, the server includes a runtime program and interface object that interface with the remote procedure itself. Results are returned the same way.

Some examples of RPC technology are Software AG's EntireX/Natural RPC, Microsoft RPC, and DCE RPC.

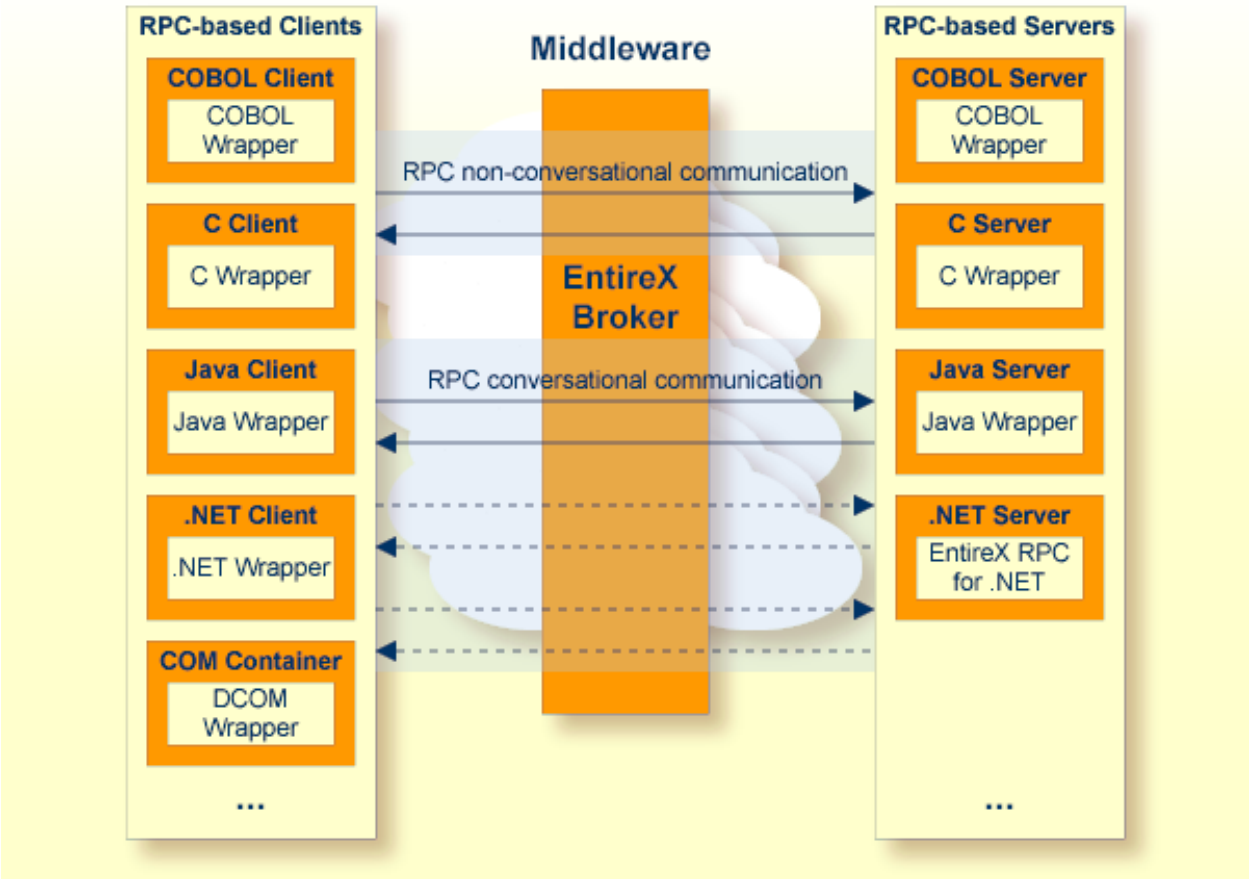
# RPC-based Components



## Introduction

The production of RPC-based components is called “wrapping” (Java Wrapper, XML/SOAP Wrapper, DCOM Wrapper, .NET Wrapper etc.). The wrapped components are perfectly embedded in their environments, for example:

- in C as functions and procedures
- in Java and .NET as classes and methods
- in COBOL and Natural as subprograms
- in COM container-enabled applications as COM/DCOM objects




### Advantages of RPC-based Components

- The programmer can work with familiar data types without having to worry about their representation on different hardware platforms, including conversion of codepages, etc.
- RPC-based components use the *EntireX Interface Definition Language* (*IDL*) to create programming-language-independent interfaces between client and server components. See *Software AG IDL File* in the IDL Editor documentation.
- The Software AG IDL file can be automatically created from Natural subprograms, COBOL and PL/I. See *IDL Extractor for Natural*, *IDL Extractor for COBOL*, *IDL Extractor for PL/I*.
- There are generation tools for the RPC-based client and server components (stubs, skeletons, etc.). See the *EntireX Workbench* and the *IDL Compiler* in the IDL Editor documentation.
- RPC-based components support non-conversational and conversational RPC communication.
- EntireX RPC-based components are compatible with Natural RPC. For Natural RPC servers, see *Setting Up a Natural RPC Environment* in your Natural documentation.

### Connectivity Matrix

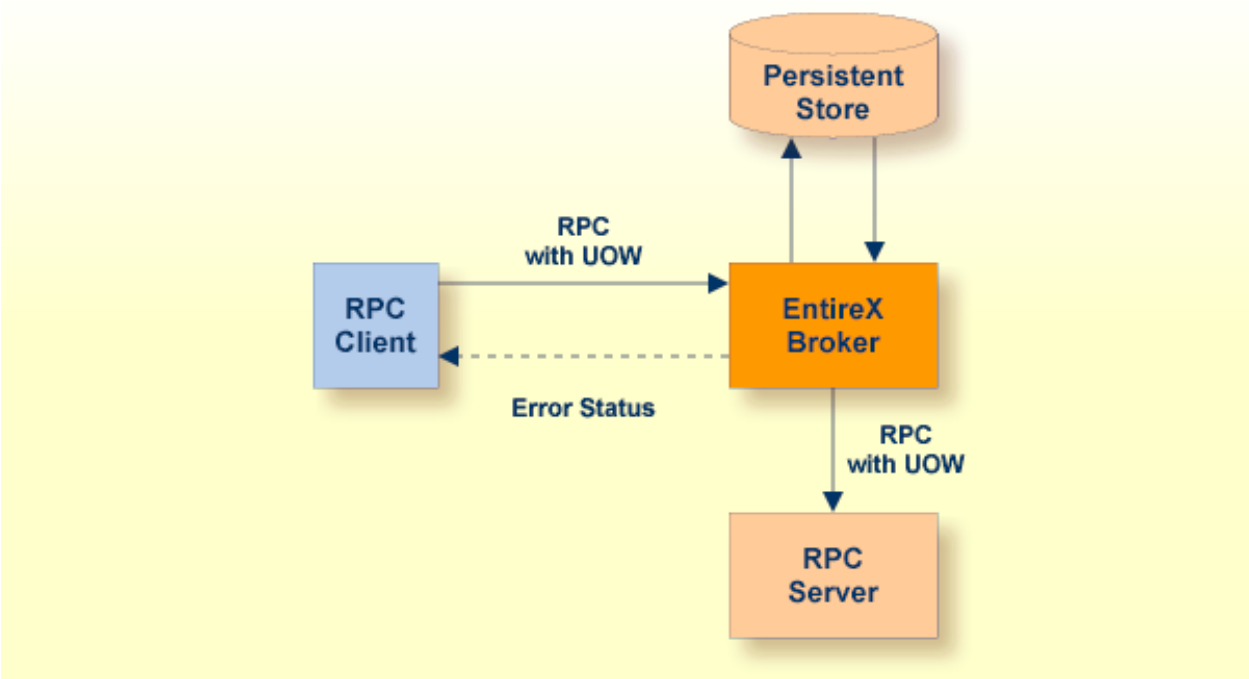
| Connection | Feature | | | | | |
|---|---|---|---|---|---|---|
| | Client Extraction | Client Generation | Server Extraction | Server Generation | Client Connectivity | Server Connectivity |
| Web Service | x | x | x | x | x | x |
| XML Service | x | x | x | x | x | x |
| Java | | x | | x | x | x |
| .NET | | x | | x | x | x |
| PHP, Perl, Ruby | via WSDL/XSD | | | | x | |
| COBOL | | x | x | x | x | x |
| PL/I | | x | x | x | x | x |
| Natural | x | x | x | x | x | x |
| RPC | | | | | | x |
| CL | | | | | | x |
| C | | x | | x | x | x |
| Assembler | | | | | | |
| WebSphere MQ | | x | | x | x | x |
| webMethods Integration Server | | x | x | x | x | x |

## Reliable RPC

In the architecture of modern e-business applications (such as SOA), loosely coupled systems are becoming more and more important. Reliable messaging is one important technology for this type of system.

Reliable RPC is the EntireX implementation of a reliable messaging system. It combines EntireX RPC technology and persistence, which is implemented with units of work (UOWs).

- Reliable RPC allows asynchronous calls (“fire and forget”)
- Reliable RPC is supported by most EntireX wrappers
- Reliable RPC messages are stored in the Broker's persistent store until a server is available
- Reliable RPC clients are able to request the status of the messages they have sent


Persistent Store
RPC with UOW
RPC Client
EntireX Broker
Error Status
RPC with UOW
RPC Server

# 2 Using the Broker ID in Applications

The Broker ID describes the connection from a client or server to a Broker instance. It indicates the protocol or transport method to be used and where the Broker is located. We distinguish two styles of Broker IDs: the URL-style Broker ID and the transport-method-style Broker ID.

The URL-style Broker ID is the recommended style. Simple forms of this style are identical with the transport method style. For both styles, the syntax, values, defaults, examples, and restrictions are listed.

## URL-style Broker ID

The URL syntax is described in RFC1738 and related RFCs.

```
<protocol><host><port><parameter>
```

| Element | Description | Permitted Values | Default | Note |
|---|---|---|---|---|
| <protocol> | The transport protocol. | tcpip://, ssl://, http://, https://, or none; | tcpip:// | Not case-sensitive. |
| <host> | The host where the Broker operates. | A valid host name. This may be a numerical IP address or a domain name. | localhost | For the syntax of the domain name, see RFC1034 (Domain Names - Concepts and Facilities). |
| <port> | The port where the Broker listens. | a valid port number in the form ": *n* ", where *n* is an integer. | Non-Java-based components:<br>The default port is resolved by the domain name service (DNS) for all components. If the DNS cannot resolve the port, 1971 is used for TCP/IP and 1958 is used for SSL.<br><br>Java-based-components:<br>The default depends on the protocol:<br>tcpip:// 1971<br>ssl:// 1958<br>http:// 80<br>https:// 443 | |

| Element | Description | Permitted Values | Default | Note |
|---|---|---|---|---|
| <parameter> | Parameters in the form<br>`?<parm1>&<parm2>&...` | The keys and the permitted values depend on the protocol. | none | See *Examples of Parameters*. |

## Examples

- localhost
- localhost:1971
- tcpip://myhost.com:1971
- tcpip://127.0.0.1:1971
- ssl://localhost:22101?trust_store=C:\SoftwareAG\EntireX/etc/ExxCACert.jks&key_store=C:\SoftwareAG\EntireX/etc/ExxJavaAppCert.jks&key_passwd=ExxJavaAppCert
- http://www.yourhost.com/servlets/tunnel
- https://www.yourhost.com/servlets/tunnel

## Examples of Parameters

### Java Programming Language

1. poolsize=*n* (*n*: number of connections)
2. pooltimeout=*n* (*n*: number of seconds until timeout)
3. compresslevel=[0|1|2|3|4|5|6|7|8|9|DEFAULT_COMPRESSION|NO_COMPRESSION|BEST_SPEED|DEFLATED|BEST_COMPRESSION|N|Y]

   (set the level of compression; N is mapped to NO_COMPRESSION; Y is mapped to 6, see *Using Compression* under *Writing Advanced Applications - EntireX Java ACI*
4. encryptionlevel=[0|1|2] (set the encryption level, see *Encryption* under *Writing Applications using EntireX Security* in the ACI Programming documentation)
5. For http, https: checkheaders=[yes|no] (check http headers if yes)
6. For http, https: log=[yes|no] (enable tracing if yes)
7. For ssl: verify_client=[yes|no] (SSL client has to send certificate if yes)
8. For ssl: verify_server=[yes|no] (verify that the host name of the Broker is the common name of the certificate, if yes.)

### Other Programming Languages

1. For ssl: verify_client=[yes|no] (SSL client has to send certificate if yes)
2. For ssl: verify_server=[yes|no] (verify that the host name of the Broker is the common name of the certificate if yes)

- EntireX RPC Server under Windows / UNIX and SSL

  You may use either the keyword SSL_file in the configuration file to specify parameters for SSL or use SSL://<host><port>?ssl_file=MySSLfile.

- CICS RPC Server and SSL

  Use the keyword SSL_file to specify the memory block with the parameters for SSL.

- EntireX RPC under C and SSL

  You may use either SSL://<host><port>?ssl_file=MySSLfile or specify parameters for SSL in the ERX structure ERX_CLIENT_IDENTIFICATION.

## Transport-method-style Broker ID

Transport methods TCP, SSL and NET are available. The transport method may be omitted, whereby certain rules apply. See *Default Rules*. The transport methods TCP and NET may be also combined. See *Examples* below.

### Transport Method TCP

**<host><port>:TCP**

| Element | Description | Permitted Values | Default |
|---|---|---|---|
| <host> | The host where the Broker operates. | Valid host name consisting of a domain name or a numerical IP address. | localhost |
| <port> | The port where the Broker listens. | Valid port number. | The default port is resolved by the domain name service (DNS). If the DNS cannot resolve the port, 1971 is used. |

### Transport Method SSL

**<host><port>:SSL**

| Element | Description | Permitted Values | Default |
|---|---|---|---|
| <host> | The host where the Broker operates. | Valid host name consisting of a domain name or a numerical IP address. | localhost |
| <port> | The port where the Broker listens. | Valid port number. | The default port is resolved by the domain name service (DNS). If the DNS cannot resolve the port, 1958 is used. |

### Transport Method NET (Entire Net-Work) under z/OS, BS2000/OSD and z/VSE

**<name><node>:[<svc>]:NET**

| Element | Description | Permitted Values | | Default |
|---|---|---|---|---|
| <name> | Sequence of letters | Any sequence of letters is allowed. | | none |
| <node> | Sequence of digits | A node number for Entire Net-Work or a database ID. The node number is required. | | none |
| <svc> | SVC number | **z/OS, z/VSE** | SVC*nnn*, where *nnn* is a valid SVC number. SVC must be uppercase. When omitted, the default SVC number is used. | none |
| | | **BS2000/OSD** | Not applicable. | |

### Examples

- Myhost.com:65534:SSL
- ETB024::TCP tells the Broker to use TCP/IP. ETB024 will be used to look up the host TCP address. Because the port number is not specified, the Broker ID ETB024 will be used by default to look up the port number.
- ETB024:3800:TCP tells the Broker to use TCP/IP. ETB024 will be used to look up the host TCP address. Because the port number is specified, no lookup for the port number takes place; 3800 is used directly for the port number.
- ETB024::NET tells the Broker to use Entire Net-Work. Under z/OS: this format is used if the SVC number must not be changed.
- ETB024:SVC252:NET tells the Broker to use Entire Net-Work, SVC number 252, as the preferred transport. This form applies to z/OS (due to the SVC number).

### Default Rules

- If broker ID does not specify a transport method, environment variable `ETB_TRANSPORT` is used.
- If environment variable `ETB_TRANSPORT` is also not specified, TCP is used.
- If the port number is not specified, 1971 is used for TCP and 1958 is used for SSL.

### Technical Limitations

#### Java

- The transport method is not supported for the programming language Java and EntireX components based on the programming language Java such as Broker Agent, Java Wrapper, Java RPC Server, etc.

#### Other Programming Languages

- For all programming languages and for EntireX components under z/OS it depends on the broker stub module used if the SVC number can be specified as part of the Broker ID. See *SVC Number for Broker Communication* under *Administration of Broker Stubs under z/OS* in the z/OS administration documentation.
- For all programming languages except Java and for EntireX components not based on the programming language Java - such as EntireX RPC Server under z/OS, CICS, UNIX and Windows, DCOM Wrapper, C Wrapper etc. - Broker ID has a maximum length of 32 characters (unless the `LONG-BROKER-ID` is used; see *`LONG-BROKER-ID-LENGTH`* under *Broker ACI Fields*).
- For the URL style the supported protocols are:
  - tcpip://
  - ssl://

# 3 Software AG IDL File

A Software AG IDL file contains definitions of the interface between client and server. The IDL file is used by Software AG wrappers to generate RPC clients, RPC servers and tester etc. on the basis of these definitions.

The IDL file can be edited by the IDL Editor provided by plug-ins for Eclipse.

This document contains a descriptive introduction to IDL files. The syntax of IDL files in a formal notation is given under *Software AG IDL Grammar* in the *IDL Editor* documentation.

## Introduction to the IDL File

The IDL's syntax looks similar to a Software AG Natural parameter data definition statement.

```
Library 'EXAMPLE' Is
        Program 'CALC' Is
                Define Data Parameter
                1 Operator         (A1) In
                1 Operand_1        (I4) In
                1 Operand_2        (I4) In
                1 Function_Result  (I4) Out
                End-Define
```

The syntax is described in a formal notation under *Software AG IDL Grammar* in the *IDL Editor* documentation.

## IDL Data Types

The table below uses the following metasymbols and informal terms for the IDL.

- The metasymbols [ and ] surround optional lexical entities.
- The informal term *number* (or in some cases *number*.*number*) is a sequence of numeric characters, for example 123.

| Type and Length | Description | Example | See Notes |
|---|---|---|---|
| A*number* | Alphanumeric | A100 | 1, 2, 7, 17, 20 |
| AV | Alphanumeric variable length | AV | 1, 2, 7, 17, 20 |
| AV*number* | Alphanumeric variable length with maximum length | AV100 | 1, 2, 7, 17, 20 |
| B*number* | Binary | B10 | 1, 2, 15 |
| BV | Binary variable length | BV | 1, 2, 15 |
| BV*number* | Binary variable length with maximum length | BV128 | 1, 2, 15 |

| Type and Length | Description | Example | See Notes |
|---|---|---|---|
| D | Date | D | 3, 4, 13 |
| F4 | Floating point (small) | F4 | 11, 13, 16 |
| F8 | Floating point (large) | F8 | 12, 13, 16 |
| I1 | Integer (small) | I1 | 8 |
| I2 | Integer (medium) | I2 | 9 |
| I4 | Integer (large) | I4 | 10 |
| K*number* | Kanji | K20 | 1, 2, 7, 17, 18, 20 |
| KV | Kanji variable length | KV | 1, 2, 7, 17, 18, 20 |
| KV*number* | Kanji variable length with maximum length | KV200 | 1, 2, 7, 17, 18, 20 |
| L | Logical | L | 3, 14 |
| N*number*[ . *number*] | Unpacked decimal | N8 or N8.2 | 6, 13 |
| NU*number*[ . *number*] | Unpacked decimal unsigned | NU2 or NU6.2 | 6, 13 |
| P*number*[ . *number*] | Packed decimal | P12 or P10.3 | 6, 13 |
| PU*number*[ . *number*] | Packed decimal unsigned | PU3 or PU4.2 | 6, 13 |
| T | Time | T | 3, 5, 13 |
| U*number* | Unicode | U100 | 2, 19 |
| UV | Unicode variable length | UV | 2, 19 |
| UV*number* | Unicode variable length with maximum length | UV200 | 2, 19 |

Note that equivalents of the data types are not necessarily supported in every target programming language environment. Also, value ranges of the mapped data type can differ. See *Mapping Software AG IDL Data Types* in the respective Wrapper or language-specific documentation.

**Notes:**

1. There is, however, an absolute limit (1 GB) which cannot be exceeded.
2. The maximum length you can specify depends on your hardware and software configuration (apart from this product).
3. The length is implicit and must not be specified.
4. The supported range is from 1.1.0001 up to 31.12.9999. Dates BC (before the birth of Christ) are not supported.

   It is also possible to transfer 1.1.0000 as a value. This is a special value (because there is no year 0) and denotes “no date” is given. The no date value is the internal state of a `#DATE` variable (Natural type D) after a `RESET #DATE` is executed within Natural programs. The target language environment determines how 'no date' is handled.

   See the notes under data type D in the section *Mapping Software AG IDL Data Types* to the target language environment C | Java | .NET.

5. The data type T has two different meanings:
   - A time-only meaning, which transfers a time without a date. The time-only meaning always uses the invalid date 1.1.000 for the date part. The time part has a value range from 00:00:00.0 to 23:59:59.9. This time-only meaning is not supported.
   - A timestamp meaning, consisting of a date and time.

     The supported range is from 1.1.0001 0:00:00.0 up to 31.12.9999 23:59:59.9. Dates BC (before the birth of Christ) are not supported.

     It is also possible to transfer 1.1.0000 0:00:00.0 as a value. This is a special value (because there is no year 0) and denotes "no time" is given. The "no time" value is the internal state of a `#TIME` (Natural type T) variable after a `RESET #TIME` is executed within Natural programs. The target language environment determines how "no time" is handled.

     See the notes under data type T in the section *Mapping Software AG IDL Data Types* to the target language C | Java | .NET.
6. The term *`number`*[ . *`number`*] describes the number as it is: The first number is the number of digits before the decimal point and the second number is the number of digits after the decimal point. The total number of digits (before and after the decimal point) must not exceed 29. The number of digits after the decimal point must not exceed 7.
7. The length is given in bytes, not in number of characters.
8. The valid integer range is from -128 up to +127.
9. The valid integer range is from -32768 up to +32767.
10. The valid integer range is from -2147483648 up to +2147483647.
11. The following term restricts the valid range which can be transferred from -n.nnnnnn+Enn up to +n.nnnnnn+Enn. A mantissa of 7 decimal digits and an exponent of 2 decimal digits.
12. The following term restricts the valid range which can be transferred from -n.nnnnnnnnnnnnnnn+Enn up to +n.nnnnnnnnnnnnnnn+Enn. A mantissa of 16 decimal digits and an exponent of 2 decimal digits.
13. The real valid range and precision can be restricted by the mapping to the target language environment.
14. Valid values are TRUE and FALSE.
15. The length is given in bytes.
16. When using floating-point values, rounding errors can occur when converting to the target language environment. Thus, values from sender and receiver might differ slightly.
17. In environments where multibyte, double-byte or other complex codepages are used, alphanumeric data may increase or decrease during conversion. Thus, to match the field length restriction given by the IDL types A and AV with maximum length, data must be truncated, otherwise unpredictable results will occur. The most popular internationalization approach *ICU Conversion*

under *Introduction to Internationalization* with `CONVERSION=SAGTRPC` takes care of data increase/decrease.

We recommend always using SAGTRPC for RPC data streams. *Conversion with Multibyte, Double-Byte and other Complex Codepages* will always be correct, and *Conversion with Single-byte Codepages* is also efficient because SAGTRPC detects single-byte codepages automatically. See *Conversion Details*.

See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific administration documentation.

18. In environments that use EBCDIC stateful codepages, encoded with escape technique (SI/SO bytes), and where the most popular internationalization approach *ICU Conversion* under *Introduction to Internationalization* with `CONVERSION=SAGTRPC` is used, the IDL types K and KV fields allow you to transfer double-byte data without SO and SI bytes. This feature is designed for use in Asian countries. For more information see *Conversion with Multibyte, Double-Byte and other Complex Codepages*.
19. The length is given in 2-byte Unicode code units following the Unicode standard. UTF-16. The maximum length is restricted to 805306367 2-byte code units.

    Depending on your target environment and target programming language, the mapping may follow a different Unicode standard, for example UTF-32.
20. If *SAGTRPC User Exit* under *Introduction to Internationalization* is used as the internationalization approach, the handling of the different IDL types depends on the implementation of the SAGTRPC user exit. This is your responsibility as user. See *Writing SAGTRPC User Exits* in the platform-specific administration documentation.

## Rules for Coding IDL Files

1. Statements and their lexical entities can begin in any column and are separated by any number of whitespace characters: blank, new line carriage return, horizontal tab, and form feed.
2. The maximum line length allowed in an IDL file is 256 characters.
3. Comments can be entered in the following ways:
   - If the entire line is to be used for a user comment, enter an asterisk or a slash and an asterisk in columns 1 and 2 of the line:

```
*    USER COMMENT
/*   USER COMMENT
```

- If only the latter part of a line is to be used for a user comment, enter an asterisk or slash asterisk.

```
  1 NAME    (A20)      * USER COMMENT
  1 NUMBER  (A15)     /* USER COMMENT
```

## Rules for Coding Group and Parameter Names

Group and parameter names

1. can be defined with the following characters:
   - characters: a to z
   - characters: A to Z
   - digits: 0 to 9 (a digit must not be the first character)
   - special characters: - _ $ # & @ + /

   other characters are not allowed.
2. are limited to a maximum length of 31 characters
3. are not allowed to be the same as a valid type-length specification.

   For example:

   ```
   1 P1 (P1) In Out
   ```

   is invalid and will cause an error because the name `P1` is identical to the type-length `P1`.
4. must adhere to the rules of the target programming language, for example to permitted special characters or reserved keywords.
5. cannot be defined as the following reserved names:

   `ALIGNED`, `CALLNAT`, `DATA`, `DEFINE`, `END-DEFINE`, `IMS`, `IN`, `INOUT`, `IS`, `LIBRARY`, `OUT`, `PARAMETER`, `PROGRAM`, `RCODE`, `STRUCT`, `VERSION`.
6. must be unique and must not conflict with those of the target programming language, see the following portion of an IDL file

```
Define Data Parameter
1 AA   (I2)
1 AA   (I4)
1 long (I4)
End-define
```

and the output generated with the client.tpl as the template for target language C:

```
short int AA;
long      AA;      /*erroneous, double declaration*/
long      long;    /*erroneous, double declaration*/
```

The ambiguous declaration of `AA` and `long` is passed unchecked and the stub will be generated. As you can see, this is not valid C syntax.

## Rules for Coding Library, Library Alias, Program, Program Alias and Structure Names

The following rules apply to library, library alias, program, program alias and structure names:

1. Names are restricted by length. Library, library alias, program and program alias are restricted to a maximum length of 128 characters. A structure name is restricted to a maximum length of 31 characters.
2. Names must adhere to the rules of the target programming language, for example regarding permitted special characters or reserved keywords.
3. Names should not start with the prefix "`SAG`". The prefix "`SAG`" is used within the delivered IDL files. See ***Change RPC Password by Wrappers and RPC Clients*** and ***Command and Info Services IDLs*** for more information.
4. Names must be unique and different within the IDL file after conversion of the name to lowercase or uppercase characters. You cannot use the same name for a library, library alias, program, program alias and structure

   Example: The following names are not allowed within an IDL file:

   - `MYLIBRARY` and `MyLibrary`
   - `CALC` and `Calc`
   - `MYSTRUCTURE` and `mystructure`

# 4 Command and Info Services IDLs

The Broker-internal RPC CIS Server provides the Command and Information Services using the Remote Procedure Call (RPC) protocol.

Two CIS IDL files are available in directory *EntireX\etc\idl*.

RPC CIS is a complete implementation of the Command and Information Services.

See *Broker CIS Data Structures* in the ACI Programming documentation for a description of the CIS API. The names of the fields can also be found in the IDL (with case-insensitive and insignificant modifications).

The service names `SAG/ETBCIS/RPCCIS` and `RPC/RPCCIS/CALLNAT` can be used for all versions of the CIS IDLs.

This chapter covers the following topics:

## Command Service IDL

The files *SagCmdServiceV6.idl* to *SagCmdServiceV8.idl* are contained in directory *etc*. They provide an interface description of CIS version 6 and above. Brokers with more recent CIS versions are backward compatible down to version 6 via RPC CIS.

File *SagCmdServiceV8.idl* provides the interface description for all Command Services of CIS version 8.

This section covers the following topics:

- Stucture COMMAND_REQUEST
- Program COMMAND

### Stucture COMMAND_REQUEST

```
struct 'COMMAND_REQUEST' is  /*Broker CIS: command request structure
  define data parameter
  1 Command                (I2)
  1 ObjectType             (I2)
  1 Option                 (I2)
  1 puid                   (B28)
  1 uowid                  (A16)
  1 Topic                  (A96)
  1 uid                    (A32)
  1 Token                  (A32)
  1 ServerClass            (A32)
  1 ServerName             (A32)
  1 ServiceName            (A32)
  1 reserved_etbinfo_v73_2 (A32)
```

```
 1 convid                (A16)
 1 transportId           (A3)
 1 nSequenceNumber       (I4)
 1 cExcludeAttachServers (I1)
 1 nErrorNumber          (I4)
 end-define
```

The request structure is described under *Command Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation. Note also the *Command Request Parameter Combinations* under *Broker CIS Data Structures* in the ACI Programming documentation.

### Program COMMAND

```
Program 'COMMAND':'command' is  /*command request
 define data parameter
 1 CmdRequest      ('COMMAND_REQUEST')  In
 1 Function_Result (I4)                 Out
 end-define
```

You can call the command service using program `COMMAND`, with the structure `COMMAND_REQUEST` as argument. See *Command Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation. Alternatively, you can enter the functions listed in the table directly:

| Program Short Name [1] | Long Name [2] | Number | Command |
|---|---|---|---|
| COMMAND | command | *n* | all available commands |
| CALLOW | cmdAllow | 13 | ALLOW-NEWUOWMSGS |
| CCLECLF | cmdClearCmdLogFilter | 20 | CLEAR-CMDLOG-FILTER |
| CNOP | cmdNoOperation | 88 | NO-OPERATION |
| CCONPST | cmdConnectPStore | 17 | CONNECT-PSTORE |
| CDISACC | cmdDisableAccounting | 28 | DISABLE-ACCOUNTING |
| CDISCL | cmdDisableCmdLog | 24 | DISABLE-CMDLOG |
| CDISCLF | cmdDisableCmdLogFilter | 22 | DISABLE-CMDLOG-FILTER |
| CDISDWK | cmdDisableDynWorker | 37 | DISABLE-DYN-WORKER |
| CDISPST | cmdDisconnectPStore | 18 | DISCONNECT-PSTORE |
| CENAACC | cmdEnableAccounting | 27 | ENABLE-ACCOUNTING |
| CENACL | cmdEnableCmdLog | 23 | ENABLE-CMDLOG |
| CENACLF | cmdEnableCmdLogFilter | 21 | ENABLE-CMDLOG-FILTER |
| CENADWK | cmdEnableDynWorker | 38 | ENABLE-DYN-WORKER |
| CFORBID | cmdForbid | 14 | FORBID-NEWUOWMSGS |
| CPROSTA | cmdProduceStatistics | 25 | PRODUCE-STATISTICS |
| CPURGE | cmdPurge | 12 | PURGE |
| CRSTUSR | cmdResetUser | 29 | RESET-USER |

| Program Short Name [1] | Long Name [2] | Number | Command |
|---|---|---|---|
| CTRARES | cmdResume | 31 | RESUME |
| CSETCLF | cmdSetCmdLogFilter | 19 | SET-CMDLOG-FILTER |
| CSHUTB | cmdShutdownBroker | 8 | SHUTDOWN [3] |
| CSHUTC | cmdShutdownConversation | 8 | SHUTDOWN |
| CSHUTP | cmdShutdownParticipant | 8 | SHUTDOWN |
| CSHUTS | cmdShutdownServer | 8 | SHUTDOWN |
| CTRASTR | cmdStart | 33 | START |
| CTRASTA | cmdStatus | 36 | STATUS |
| CTRASTP | cmdStop | 32 | STOP |
| CSUB | cmdSubscribe | 15 | SUBSCRIBE [4] |
| CTRASUS | cmdSuspend | 30 | SUSPEND |
| CSWICL | cmdSwitchCmdLog | 26 | SWITCH-CMDLOG |
| CTRCFLU | cmdTraceFlush | 35 | TRACE-FLUSH |
| CTRCOFF | cmdTraceOff | 2 | TRACE-OFF |
| CTRCON | cmdTraceOn | 1 | TRACE-ON |
| CTRPERR | cmdTrapError | 34 | TRAP-ERROR |
| CUNSUB | cmdUnsubscribe | 16 | UNSUBSCRIBE [4] |

**Notes:**

1. Short name as used, for example, by C programs.
2. Long name as used, for example, by Java programs.
3. You cannot execute `CSHUTB` (`cmdShudtownBroker`) in a non-secure mode.
4. See *Writing Applications: Publish and Subscribe* in the ACI Programming documentation.

The prototypes and source code can be found in the generated files:

| Language | File(s) |
|---|---|
| C | CSAGCCV8.h, CSAGCCV8.c |
| Java | SagCmdServiceV8.java |
| .NET | Sagccv8.cs |

## Info Service IDL

Files *SagInfServiceV6.idl* to *SagInfServiceV8.idl* are contained in directory *etc*. They provide an interface description of CIS version 6 and above. Brokers with more recent CIS versions are backward compatible down to version 6 via RPC CIS.

File *SagInfServiceV8.idl* provides the interface description for all Information Services. The following functions are used to receive an unbounded array of the corresponding Broker Information Service.

See also *Using Unbounded Arrays* under *Writing Advanced Applications with the C Wrapper*.

### Structure INFORMATION_REQUEST

```
struct 'INFORMATION_REQUEST' is  /*CIS: information request
  define data parameter
  1 ObjectType       (I2)
  1 uid              (A32)
  1 puid             (B28)
  1 Token            (A32)
  1 ServerClass      (A32)
  1 ServerName       (A32)
  1 ServiceName      (A32)
  1 convid           (A16)
  1 uowid            (A16)
  1 uowStatus        (I1)
  1 userStatus       (A32)
  1 recvUID          (A32)
  1 recvToken        (A32)
  1 recvClass        (A32)
  1 recvServer       (A32)
  1 recvService      (A32)
  1 Topic            (A96)
  1 publicationId    (A16)
  1 subscriptionType (I2)
  1 conversationType (I2)
  1 level            (I1)
  end-define
```

The request structure is described under *Information Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation.

### Program: INFO

```
program 'INFO':'info' is  /*all
  define data parameter
  1 InfRequest       ('INFORMATION_REQUEST')   In
  1 InfBroker        ('INFO_BRK'/V)            Out
  1 InfWorker        ('INFO_WRK'/V)            Out
  1 InfService       ('INFO_SV'/V)             Out
  1 InfClient        ('INFO_CS'/V)             Out
  1 InfServer        ('INFO_CS'/V)             Out
  1 InfParticipant   ('INFO_CS'/V)             Out
  1 InfConversation  ('INFO_CV'/V)             Out
  1 InfPSF           ('INFO_PSF'/V)            Out
  1 InfPSFADA        ('INFO_PSFADA'/V)         Out
  1 InfPSFDIV        ('INFO_PSFDIV'/V)         Out
  1 InfPSFFile       ('INFO_PSFFILE'/V)        Out
  1 InfPSFCTree      ('INFO_PSFCTREE'/V)       Out
  1 InfSubscriber    ('INFO_SUBSCRIBER'/V)     Out
  1 InfPublisher     ('INFO_PUBLISHER'/V)      Out
  1 InfPublication   ('INFO_PUBLICATION'/V)    Out
  1 InfTopic         ('INFO_TOPIC'/V)          Out
  1 InfTcp           ('INFO_TCP'/V)            Out
  1 InfSecurity      ('INFO_SECURITY'/V)       Out
  1 InfSsl           ('INFO_SSL'/V)            Out
  1 InfCmdLogFilter  ('INFO_CMDLOG_FILTER'/V)  Out
  1 InfNet           ('INFO_NET'/V)            Out
  1 InfPoolUsage     ('INFO_POOL_USAGE'/V)     Out
  1 InfResourceUsage ('INFO_RESOURCE_USAGE'/V) Out
  1 InfStatistics    ('INFO_STATISTICS'/V)     Out
  1 InfUser          ('INFO_USER'/V)           Out
  1 InfWorkerUsage   ('INFO_WRK_USAGE'/V)      Out
  1 Function_Result  (I4)                      Out
  end-define
```

You can call the information service using program `INFO`, with the structure `INFORMATION_REQUEST` as argument. See *Information Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation. Depending on the object type, the reply will contain the corresponding `INFO_` structure containing one or more records. The variable array contains all available data. No segmentation takes place. Alternatively, you can call directly the functions listed below for the individual object types.

All functions below return the corresponding structure from the information reply structures.

| Program Short Name [1] | Long Name [2] | See |
|---|---|---|
| INFO | info | One of available reply structures; see *Information Reply Structures* under *Broker CIS Data Structures* in the ACI Programming documentation |
| IBROKER | infoBroker | *BROKER-OBJECT* |
| ICMDLGF | infoCmdLogFilter | *CMDLOG_FILTER-OBJECT* |
| ICLIENT | infoClient | *CLIENT-SERVER-PARTICIPANT-OBJECT* |
| ICONV | infoConversation | *CONVERSATION-OBJECT* |
| INET | infoNet | *NET-OBJECT* |
| IPOOLUS | infoPoolUsage | *POOL-USAGE-OBJECT* |
| IPARTI | infoParticipant | *CLIENT-SERVER-PARTICIPANT-OBJECT* |
| IPSF | infoPsf | *PSF-OBJECT* |
| IPSFADA | infoPsfAda | *PSFADA-OBJECT* |
| IPSFCTR | infoPsfCtr | *PSFCTREE-OBJECT* |
| IPSFDIV | infoPsfDiv | *PSFDIV-OBJECT* |
| IPSFFIL | infoPsfFil | *PSFFILE-OBJECT* |
| IPUBLIC | infoPublication | *PUBLICATION-OBJECT* |
| IPUBLIS | infoPublisher | *PUBLISHER-OBJECT* |
| IRESUS | infoResourceUsage | *RESOURCE-USAGE-OBJECT* |
| ISECUR | infoSecurity | *SECURITY-OBJECT* |
| ISERVER | infoServer | *CLIENT-SERVER-PARTICIPANT-OBJECT* |
| ISERVIC | infoService | *SERVICE-OBJECT* |
| ISSL | infoSSL | *SSL-OBJECT* |
| ISTAT | infoStatistics | *STATISTICS-OBJECT* |
| ISUBSCR | infoSubscriber | *SUBSCRIBER-OBJECT* |
| ITCP | infoTcp | *TCP-OBJECT* |
| ITOPIC | infoTopic | *TOPIC-OBJECT* |
| IUSER | infoUser | *USER-OBJECT* |
| IWORKER | infoWorke | *WORKER-OBJECT* |
| IWORKUS | infoWorkerUsage | *WORKER-USAGE-OBJECT* |

**Notes:**

1. Short name as used, for example, by C programs.
2. Long name as used, for example, by Java programs.

The prototypes and source code can be found in the generated files:

| Language | File(s) |
|---|---|
| C | CSAGCIV8.h, CSAGCIV8.c |
| Java | SagInfServiceV8.java |
| .NET | Sagciv8.cs |

## Examples

The following examples are available:

- .NET CIS Client Example
- Java CIS Client Example

### .NET CIS Client Example

The .NET CIS Client Example can be found in the examples directory *NetWrapper\Client\SagCisClient.cs*. The directory includes the source file *SagCisClient.cs* and the *README.TXT*.

Information required to prepare and configure the *EntireX Workbench*:

- *EntireX Workbench*.
- *Using EntireX Custom Wrappers* or
- *Using the EntireX Workbench in Command-line Mode*.

Information required to to write a .NET CIS Client:

- *Writing Applications with the .NET Wrapper*.
- *Using Unbounded Arrays* under *Writing Advanced Applications with the C Wrapper*.

Build steps:

1. Prepare Eclipse.
2. Start Eclipse.
3. Create a new Java Project.
4. Add files *SagCmdServiceV8.idl* and *SagInfServiceV8.idl* to the project.
5. Generate the .NET RPC Client from *SagCmdServiceV8.idl* using the context menu.
6. Generate the .NET RPC Client from *SagInfServiceV8.idl* using the context menu.
7. Start Visual Studio.
8. Create a new Visual Studio Project. (.NET Console Application).
9. Insert the generated sources *SAGCCV8.cs* and *SAGCIV8.cs*.

10. Delete *Program.cs*.
11. Insert .NET example file *SagCisClient.cs* to the project.
12. Add Reference (*<drive>:\SoftwareAG\common\EntireX\bin\SoftwareAG.EntireX.NETWrapper.Runtime.dll*).
13. Build the solution.
14. Start the .NET CIS Client.

### Java CIS Client Example

The Java CIS Client Example can be found in the examples directory *EntireX\Examples\JavaWrapper\Client\SagCisService\mainSagCisClient.java*. This directory includes the source file *mainSagCisClient.java* and the *README.TXT*.

Information required to prepare and configure the *EntireX Workbench*:

- *EntireX Workbench*.
- *Using EntireX Custom Wrappers* or
- *Using the EntireX Workbench in Command-line Mode*

Information required to to write a Java CIS Client:

- *Writing Applications with the Java Wrapper*

Build steps:

1. Prepare Eclipse.
2. Start Eclipse.
3. Create a new Java Project.
4. Add files *SagCmdServiceV8.idl* and *SagInfServiceV8.idl* to the project.
5. Add source file *mainSagCisClient.java* to the project.
6. Generate the Java RPC Client from *SagCmdServiceV8.idl* using the context menu.
7. Generate the Java RPC Client from *SagInfServiceV8.idl* using the context menu.
8. Build the Java CIS Client project.
9. Start the Java CIS Client.

# 5 Common Features of Wrappers and RPC-based Components

This chapter provides additional information on concepts and features common to all wrappers and RPC-based components.

## Change RPC Password by Wrappers and RPC Clients

The application programmer can embed an RPC password change in an application. This is useful if the application programmer wants to provide this functionality to end users of RPC applications. It is necessary if the RPC server forces alteration of the RPC password, otherwise denying use of the RPC server.

The functionality is provided with a special-purpose IDL:

```
Library 'SAGCRPW' : 'SagChangeRPCPassword' is
  Program 'SAGCRPW' : 'changeRPCPassword' is
    Define Data Parameter
       1 newRPCPassword (A8) in
    End-Define
```

The prefix “SAG” is reserved and is used for Software AG delivered IDL files and must not be used by customer applications; see *Rules for Coding Library, Library Alias, Program, Program Alias and Structure Names* under *Software AG IDL File* in the IDL Editor documentation.

Proceed as follows:

- Define the IDL in the Workbench Editor and generate a wrapper as you would for any other IDL.
- Write a wrapper client program and issue an RPC request as you would for any other IDL. See the documentation on EntireX wrappers for an example.
- Specify the old RPC password in the same way as for any other RPC request issued. See the wrapper documentation on how to specify the password.

**Natural RPC Server running under Natural Security**

- may force the user of an application to alter the RPC Password, e.g. in the following situations:
  - NAT838:

    ```
    Change your password. Enter the old and a new password
    ```

  - NAT873:

```
User ID or password invalid
```

**Other RPC Servers**

- do not support this functionality.

## Natural Logon or Changing the Library Name

The library name sent with the RPC request to the EntireX RPC or the Natural RPC Server is specified in the Software AG IDL file (see `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation). The library name can be overridden by wrapper-specific methods, see your wrapper documentation.

For EntireX RPC Servers, depending on the target server, the library name

- is used by an EntireX Java RPC server. The program name is a method within the class called as the name of the class called. See *Administering the EntireX Java RPC Server* in the UNIX and Windows administration documentation.
- is used by an EntireX RPC server under Windows as the name of the dynamic-link library (DLL). The program name is a function export within the DLL called. See *Administering the EntireX Java RPC Server* in the UNIX and Windows administration documentation.
- is used by an EntireX RPC server under UNIX as the name of the shared library or shared object called. The program name is a function export within the shared library or shared object called. See *Administering the EntireX Java RPC Server* in the UNIX and Windows administration documentation.
- is customizable if a CICS RPC Server is used. See *Locating and Calling the Target Server*.
- for ***Conversational RPC*** is considered for every remote procedure call that belongs to the conversation.

For Natural RPC servers, the library name

- is used as the Natural library name
- can have a maximum length of 8 characters
- is considered only if Natural Logon is forced, even to Natural RPC Server running without Natural Security. If Natural Logon is not given, a Natural RPC Server (under Natural Security or non-security) does not consider the library name. See your EntireX Wrapper documentation for information on how Natural Logon can be forced.
- for ***Conversational RPC*** is evaluated at the time the conversation is opened. During an ongoing RPC conversation the Natural library cannot be changed due to Natural RPC rules.

## Conversational RPC

EntireX RPC and Natural RPC also supports conversational communication (also known as connection-oriented communication), where the two partners (client and server) retain a communication link over several remote procedure calls.

A context can be maintained on the server side when a Natural RPC Server is in use. See the `DEFINE DATA CONTEXT` statement in the appropriate Natural documentation.

EntireX Wrappers and RPC clients allow termination of an RPC conversation either successfully or abnormally by offering two different methods or function calls for ending an RPC conversation. See the appropriate EntireX Wrapper or RPC client documentation for information on how to initiate the end of an RPC communication.

If communicating with a Natural RPC Server and

- the RPC conversation is ended normally,

  the Natural RPC Server executes a Natural `END TRANSACTION` statement, resulting in a commit of all database manipulations at the server side done within the RPC conversation;
- the RPC conversation is aborted,

  the Natural RPC Server executes a Natural `BACKOUT TRANSACTION` statement, resulting in a backout of all database manipulations done at the server side within the RPC conversation.

See your Natural and Natural RPC documentation for more information.

If communicating with an EntireX RPC Server

- no automatic database processing is initiated. Aborting and closing an RPC conversation are the same and have no effect if database manipulations were done at the server side within the RPC conversation.

## Non-conversational RPC

The basic method of communication for both the EntireX and the Natural RPC is non-conversational (also known as connectionless communication).

Using this method,

- each RPC request is isolated and has no relationship to any other RPC request.
- there is no context and no context could be maintained by the RPC Server.

## Natural Security

A Natural RPC Server may run under Natural Security to protect RPC requests. RPC clients need to be

- **authenticated**
  i.e. the RPC client needs to be defined within Natural Security. Authentication is done with a user ID/password check.
- **authorized**
  i.e. the RPC client needs to be allowed to access programs in the target Natural library, otherwise a security violation error will be returned.

See your Natural Security documentation for more information.

## RPC Compression

RPC compression is a feature used to reduce network data sizes. EntireX tries to select RPC compression automatically. If RPC compression is supported by your RPC server, we recommend using it.

### Natural RPC Servers

Natural RPC Servers running under Natural version 3.1.6 and later support RPC compression. If you are communicating with Natural RPC version 5.1.1 and later, RPC compression is selected automatically. To enable connection to earlier versions of Natural RPC Servers, EntireX wrappers and RPC technology allow you to switch off RPC compression.

### EntireX RPC Servers

All versions of the EntireX RPC server support RPC compression. If you are communicating with an RPC server using EntireX 5.3.1 and later, RPC compression is selected automatically.