webMethods EntireX

EntireX RPC Programming

Version 10.7

October 2020

Document ID: EXX-RPC-107-20220422

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 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 RPC, Microsoft RPC, and DCE RPC.

Software AG's Natural RPC uses the same RPC protocol as EntireX RPC, which means they are fully compatible.

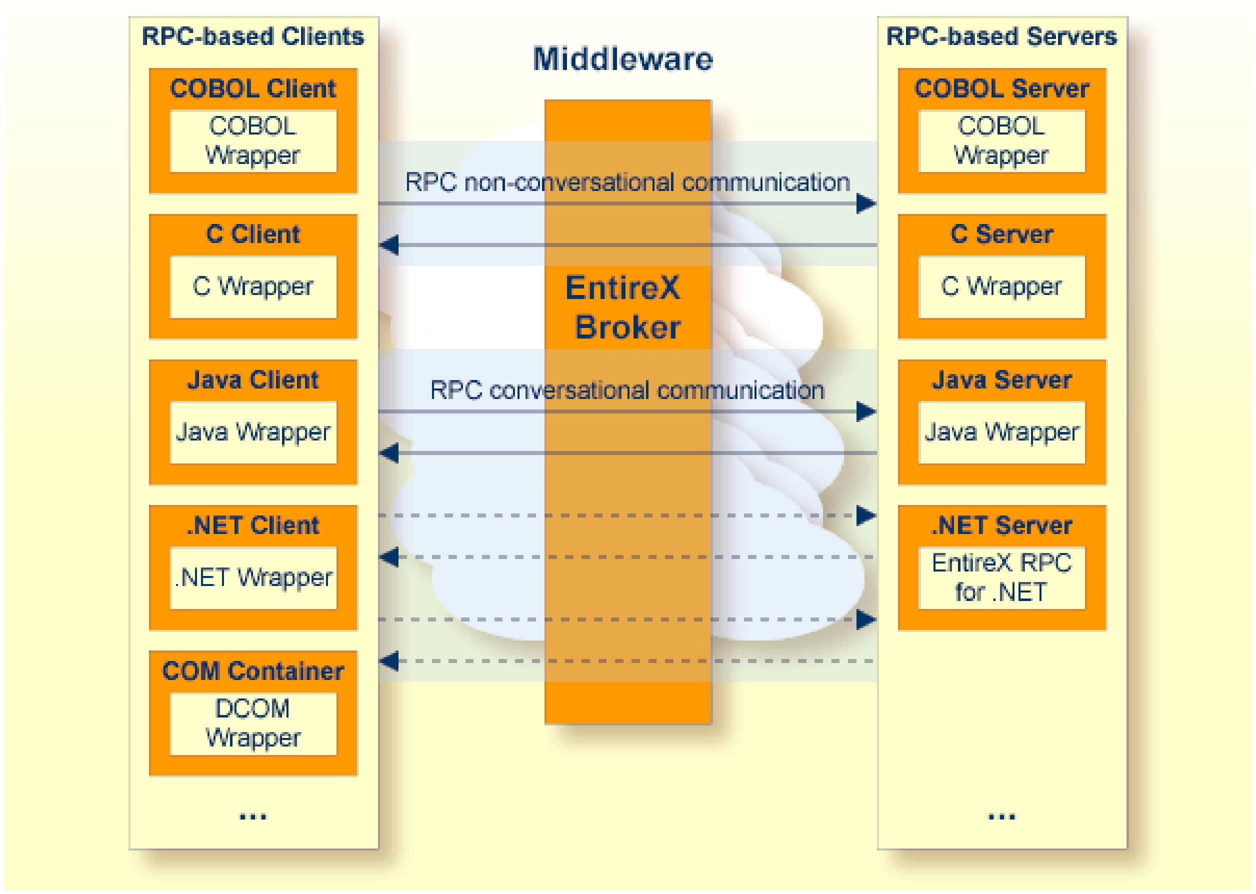
RPC-based Components

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 character conversion.
- 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 in the IDL Editor documentation.
- The Software AG IDL file can be automatically created from Natural subprograms, COBOL, PL/I, XML, Integration Server etc. See *Software AG IDL Extractors*.
- There are generation tools for the RPC-based client and server components (client interface objects, server interface objects, skeletons, etc.). See *EntireX Wrappers*.
- 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

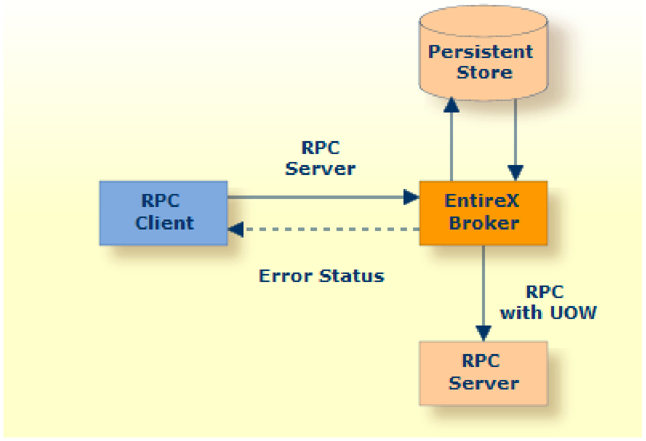
Connection	Feature					
	Client Extraction	Client Generation	Server Extraction	Server Generation	Client Connectivity	Server Connectivity
Web Service	x	x	x	x	x	x
XML Service	x	x	x	x	x	x
Java		x		x	x	x
.NET		x		x	x	x
PHP, Perl, Ruby	via WSDL/XSD				x	
COBOL		x	x	x	x	x
PL/I		x	x	x	x	x
Natural	x	x	x	x	x	x
RPG						x
CL						x
C		x		x	x	x
Assembler						
IBM® 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



See also separate section *Reliable RPC*.

3 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

Syntax

The URL syntax is described in RFC1738 and related RFCs.

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

Element	Description	Permitted Values	Default	Notes
`<protocol>`	The transport protocol.	`tcpip://`, `ssl://`, `http://`, `https://`, or none;	`tcpip://`	case-insensitive. ■ For SSL, see *Using SSL/TLS with EntireX Components*. ■ For HTTP(S), see *Using HTTP(S) Tunneling* (EntireX Java ACI | Java Wrapper).
`<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: The 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. Java-based components:	

Element	Description	Permitted Values	Default	Notes
			The default depends on the protocol: `tcpip://` 1971 `ssl://` 1958 `http://` 80 `https://` 443	
`<parameter>`	Parameters in the form `?<parm1>&<parm2>&...`	The keys and the permitted values depend on the protocol.	none	See *Examples of Parameters for Java-based Components*.

Notes

Use the URL-style Broker ID for Java-based EntireX components such as

- EntireX Java ACI | Java Wrapper
- Broker TCP Agent in the UNIX | Windows Administration documentation, Broker SSL Agent in the UNIX | Windows Administration documentation, Broker HTTP(S) Agent in the UNIX | Windows Administration documentation
- RPC Servers: CICS ECI | CICS Socket Listener | IMS Connect | Java | XML/SOAP | IBM MQ
- RPC Listeners: XML/SOAP | IBM MQ
- *EntireX Adapter*
- etc.

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 for Java-based Components

- **Socket Pooling**
  - `poolsize=`*n*, where *n*=number of connections
  - `pooltimeout=`*n*, where *n*=number of seconds until timeout; see *Socket Pooling Parameters for TCP and SSL/TLS Communication*
- **Compression**
  - `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*
- **HTTP, HTTPS**
  - `checkheaders=[yes|no]`. If `yes`: check HTTP headers
  - `log=[yes|no]`. If `yes`: enable tracing; see *Using HTTP(S) Tunneling* (EntireX Java ACI | Java Wrapper)
- **SSL**
  - `verify_client=[yes|no]`. If `yes`: SSL client has to send certificate
  - `verify_server=[yes|no]`. If `yes`: verify that the host name of the SSL server is the common name of the certificate. The SSL server can be EntireX Broker Broker SSL Agent or Direct RPC in Integration Server (IS inbound). See:
    - *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
    - Broker SSL Agent in the UNIX | Windows Administration documentation
    - *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

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	Notes
`<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.	1971	The port is resolved by the domain name service (DNS). If the DNS cannot resolve the port, the default 1971 is used.

Transport Method SSL

<host><port>:SSL

Element	Description	Permitted Values	Default	Notes
`<host>`	The host where the broker operates.	Valid host name consisting of a domain name or a numerical IP address.	localhost	See *Using SSL/TLS with EntireX Components*.
`<port>`	The port where the broker listens.	Valid port number.	1958	The port is resolved by the domain name service (DNS). If the DNS cannot resolve the port, the default 1958 is used.

Transport Method NET (Entire Net-Work) under z/OS and BS2000

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

Element	Description	Permitted Values		Default	Notes
`<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	The broker stub module you use determines whether the SVC number can be specified as part of the broker ID. See *SVC Number for Broker Communication* in the z/OS Administration documentation.
		BS2000	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

- The transport method is not supported for Java-based EntireX components such as:
  - EntireX Java ACI | Java Wrapper
  - Broker TCP Agent in the UNIX | Windows Administration documentation, Broker SSL Agent in the UNIX | Windows Administration documentation, Broker HTTP(S) Agent in the UNIX | Windows Administration documentation
  - RPC Servers: CICS ECI | CICS Socket Listener | IMS Connect | Java | XML/SOAP | IBM MQ
  - RPC Listeners: XML/SOAP | IBM MQ
  - *EntireX Adapter*
  - etc.

  Use the *URL-style Broker ID* instead.
- For *EntireX Broker ACI Programming* with programming languages such as Natural, COBOL, C etc, the broker ID has a maximum length of 32 characters unless the `LONG-BROKER-ID-LENGTH` is used (Assembler | C | COBOL | PL/I | Natural). See `LONG-BROKER-ID-LENGTH` under *Broker ACI Fields*.

4 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

In the table below, the following metasymbols and informal terms are used for the IDL.

- The metasymbols "[" and "]" enclose optional lexical entities.
- The informal term *number* (or in some cases *number1*.*number2*) is a sequence of numeric characters, for example 123.

Type and Length	Description	Example	See Notes
A*number*	Alphanumeric	A100	1, 2, 7, 16, 19
AV	Alphanumeric variable length	AV	1, 2, 7, 16, 19, 20
AV*number*	Alphanumeric variable length with maximum length	AV100	1, 2, 7, 16, 19, 20
B*number*	Binary	B10	1, 2, 14
BV	Binary variable length	BV	1, 2, 14, 20
BV*number*	Binary variable length with maximum length	BV128	1, 2, 14, 20
D	Date	D	3, 4
F4	Floating point (small)	F4	11, 15
F8	Floating point (large)	F8	12, 15
I1	Integer (small)	I1	8
I2	Integer (medium)	I2	9
I4	Integer (large)	I4	10
K*number*	Kanji	K20	1, 2, 7, 16, 17, 19
KV	Kanji variable length	KV	1, 2, 7, 16, 17, 19, 20
KV*number*	Kanji variable length with maximum length	KV200	1, 2, 7, 16, 17, 19, 20
L	Logical	L	3, 13
N*number1*[.*number2*]	Unpacked decimal	N8 or N8.2	6
NU*number1*[.*number2*]	Unpacked decimal unsigned	NU2 or NU6.2	6
P*number1*[.*number2*]	Packed decimal	P12 or P10.3	6
PU*number1*[.*number2*]	Packed decimal unsigned	PU3 or PU4.2	6
T	Time	T	3, 5
U*number*	Unicode	U100	2, 18
UV	Unicode variable length	UV	2, 18, 20
UV*number*	Unicode variable length with maximum length	UV200	2, 18, 20

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 IDL*

Data Types in the respective Wrapper or language-specific documentation and also *Integration Server Data Types to IDL Mapping*.

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 *`number1`*[.*`number2`*] 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 (*`number1`*+*`number2`*) must not exceed 99. Depending on your target programming language, the total number of digits can be more restricted.

   If you connect two endpoints, the total number of digits used must be lower or equal than the maxima of both endpoints. For the supported total number of digits for endpoints, see the notes under data types N, NU, P and PU in section *Mapping IDL Data Types* in the respective Wrapper or language-specific documentation.

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. Valid values are TRUE and FALSE.
14. The length is given in bytes.
15. 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.
16. In environments that use *Multibyte or Double-byte Codepages*, *EBCDIC Stateful Codepages* or *Arabic Shaping*, alphanumeric data may increase or decrease during character 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. See also *Rules for Data Length Changes due to Character Conversion* under *Introduction to Internationalization*. This feature is not supported on BS2000. It requires the following broker configuration:

    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*.
17. In environments that use EBCDIC stateful codepages (Asian countries) encoded with escape technique (SI/SO bytes), the IDL types K and KV fields allow you to transfer double-byte data without SO and SI bytes. See also *EBCDIC Stateful Codepages* under *Introduction to Internationalization*. This feature is not supported on BS2000. It requires the following broker configuration:

    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*.
18. 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.
19. If *SAGTRPC User Exit* is used as the character conversion 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* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation.

20. Variable-length (e.g. AV, AV`n`) fields are transferred in the RPC data stream in the length specified. A defined maximum in the IDL file limits the number of elements that can be transferred.

    Variable-length fields with maximum (e.g. AV`n`) are important for connections to endpoints that have no concept of variable-length data, such as COBOL (see *Software AG IDL to COBOL Mapping*) and PL/I (see *Software AG IDL to PL/I Mapping*).

Fixed and Unbounded Arrays

A fixed array is transferred in the RPC data stream with all its elements.

With an unbounded array, the current number of elements and their contents are transferred in the RPC data stream. A defined maximum in the IDL file limits the number of elements that can be transferred.

For the formal syntax of arrays, refer to `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Unbounded arrays with a maximum are important for connections to COBOL, which supports a similar concept with the `OCCURS DEPENDING ON` clause. See *COBOL Tables with Variable Size - `DEPENDING ON` Clause* under *COBOL to IDL Mapping* in the IDL Extractor for COBOL 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

The following rules apply to group and parameter names:

1. Names 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: - _ $ # & @ + / £ æ Æ ø Ø å Å

   Other characters are not allowed.
2. Names must adhere to the rules of the target programming language, for example to permitted special characters or reserved keywords.
3. They cannot be defined as the following reserved names:

   `ALIGNED`, `DATA`, `DEFINE`, `END-DEFINE`, `IMS`, `IN`, `INOUT`, `IS`, `LIBRARY`, `OUT`, `PARAMETER`, `PROGRAM`, `STRUCT`.
4. Names 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   (I4)
   1 long (I4)
   End-define
   ```

   and the output generated with the *C Wrapper*:

   ```
   short int AA;
   long      long;    /*erroneous*/
   ```

   The declaration of `long` is passed unchecked and the interface objects 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:

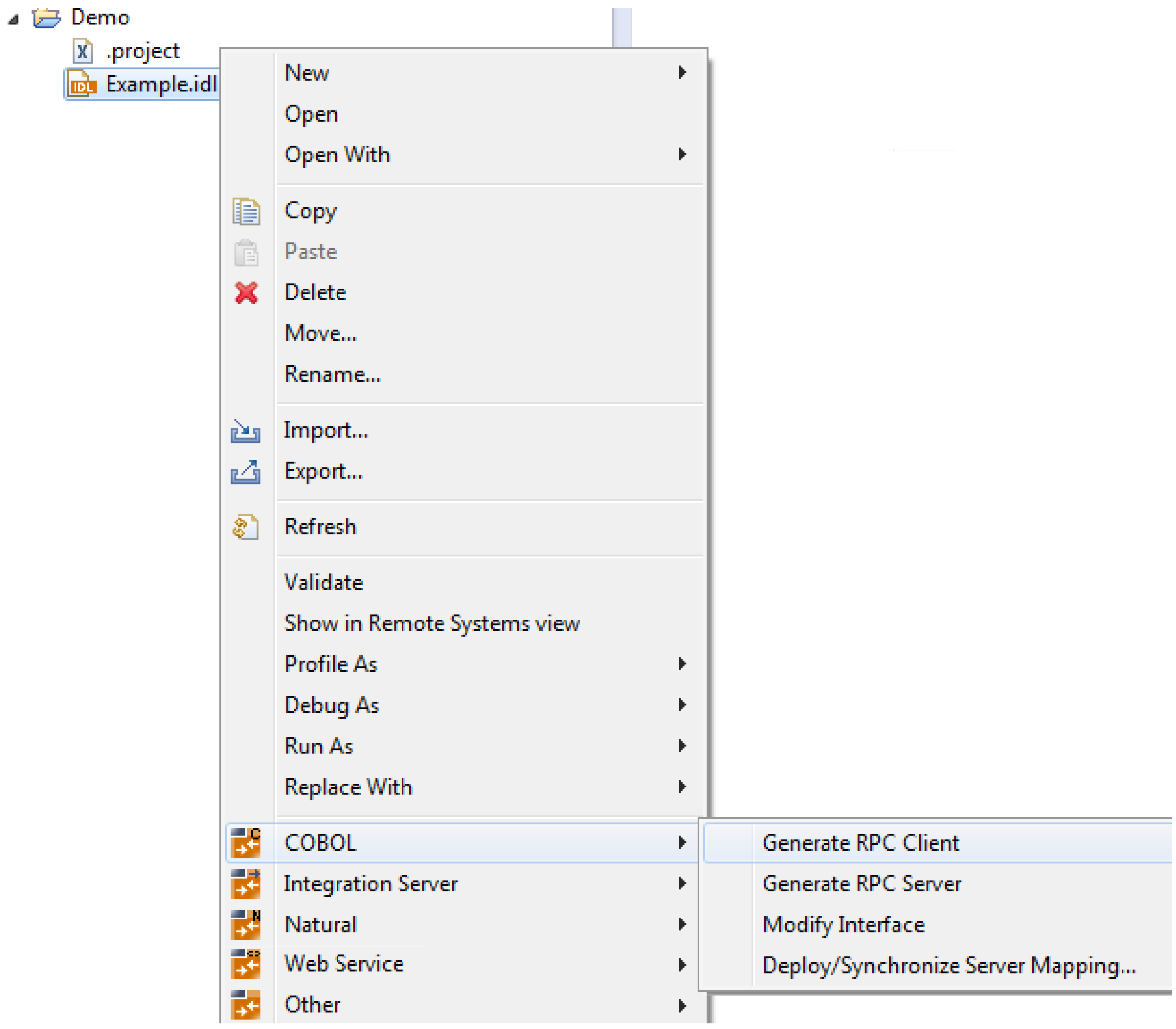
1. Names are restricted by length. Library, library alias, program and program alias are restricted to a maximum length of 128 characters. A structure name is restricted to a maximum length of 31 characters.
2. Names must adhere to the rules of the target programming language, for example regarding permitted 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 adapting them 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`

Context Menu

The context menu for IDL files in the Designer has the most commonly used target environments - COBOL, Integration Server, Natural and Web Services - on the first menu level. Under **Other** you can specify additional supported targets such as C or Java, start the IDL Tester or refactor the IDL file. See example for COBOL below:


Demo
.project
Example.idl
New
Open
Open With
Copy
Paste
Delete
Move...
Rename...
Import...
Export...
Refresh
Validate
Show in Remote Systems view
Profile As
Debug As
Run As
Replace With
COBOL
Integration Server
Natural
Web Service
Other
Generate RPC Client
Generate RPC Server
Modify Interface
Deploy/Synchronize Server Mapping...

5 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* 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)
  1 collectorBrokerId      (A64)
  1 hostName               (A256)
  1 processId              (A16)
  end-define
```

The request structure is described under *Command Request Structure*. Note also the *Command Request Parameter Combinations*.

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*. 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
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

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.

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*.

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 conversationType (I2)
  1 level            (I1)
  end-define
```

The request structure is described under *Information Request Structure*.

The optional parameter `level`, which only appears in the Info Service IDL, determines the service used:

- If `level=0` the RPC request is passed to the broker-internal service `USER-INFO` (default).
- If `level=1` the RPC request is passed to the service `INFO`.

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 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*. 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*
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
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

Program Short Name [(1)]	Long Name [(2)]	See
ITCP	infoTcp	TCP-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

6 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*.

Proceed as follows:

- Define the IDL in the Designer 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 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 *RPC Server for Java*. The program name is a method within the class called as the name of the library called.
- is used by an *RPC Server for C* under
  - Windows as the name of the dynamic-link library (DLL). The program name is a function export within the DLL called.
  - 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.
- is customizable if COBOL is the programming language of your target server. See *Locating and Calling the Target Server* under z/OS (CICS, Batch, IMS) | BS2000 in the respective Administration or RPC Server documentation.
- 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**
  that is, the RPC client needs to be defined within Natural Security. Authentication is done with a user ID/password check.
- **authorized**
  that is, 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.

7 Supported RPC Protocols

RPC Protocol[1]	Feature[2]	EntireX	EntireX Adapter for Integration Server	PI Adapter for EntireX	Natural			More Information
					Mainframe	ONE	Open Systems	
2000	IDL unbounded arrays	5.4	7.1	4.1.0	4.1.3	8.1	6.1.1 PL13	Unbounded arrays are arrays with variable upper bounds, see `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation. For the programming language COBOL they can be defined with an upper maximum and mapped to COBOL data items with the `DEPENDING ON` clause. See *COBOL Tables with Variable Size - `DEPENDING ON` Clause*. If a server is wrapped with the COBOL Wrapper or extracted with the IDL Extractor for COBOL a server mapping file [3] is automatically created for IDL unbounded arrays.
	IDL types PU, NU	5.4	7.1	4.1.0	4.1.3	8.1	6.1.1 PL13	The data types packed and unpacked decimal unsigned support the programming languages COBOL and PL/I. See *IDL Data Types* in the IDL Editor documentation.
	IDL structs	5.4	7.1	4.1.0	4.1.3	8.1	6.1.1 PL13	An IDL struct describes a user-defined type for reusability. For RPC server with a CICS channel

RPC Protocol[(1)]	Feature[(2)]	EntireX	EntireX Adapter for Integration Server	PI Adapter for EntireX	Natural			More Information
					Mainframe	ONE	Open Systems	
								container interface, it describes the layout of container. See `structure-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.
	IDL aligned attribute	5.4	7.1	4.1.0	n/a	n/a	n/a	The aligned attribute is relevant for the programming languages COBOL and PL/I and the server side. For other languages it is not relevant. See `attribute-list` under *Software AG IDL Grammar* in the IDL Editor documentation.
	IDL parameters with more than 3 indices (all indices up to the parent on level 1 counted)	5.4	7.1	4.1.0	n/a	n/a	n/a	See `array-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.
	IDL library or IDL program longer than 8 characters for COBOL	5.4	7.1	4.1.0	4.2.6	8.1	6.3.7	If a server is wrapped with COBOL Wrapper or extracted with IDL Extractor for COBOL with a long IDL program name (longer than 8 characters) or extracted with the or extracted with IDL Extractor for COBOL and renamed to a different IDL program name, a server mapping file [(3)] is created automatically.
	Reliable RPC	8.0	7.2	4.1.1	4.2.5	8.1	6.3.6	*Reliable RPC*.
	CICS Channel Container	8.2.2	7.1	4.1.0	n/a	n/a	n/a	If a COBOL server with interface type CICS with Channel Container Calling Convention is wrapped with the COBOL Wrapper or extracted with the IDL Extractor for COBOL, a server mapping file [3] is automatically created.

RPC Protocol[(1)]	Feature[(2)]	EntireX	EntireX Adapter for Integration Server	PI Adapter for EntireX	Natural			More Information
					Mainframe	ONE	Open Systems	
	Mixed case RPC passwords	8.2.2	8.2.2	no	4.1.4	8.1	6.1.1 PL21	Allow RPC passwords with lowercase characters. See ***Natural Security*** under *Common Features of Wrappers and RPC-based Components* in the RPC Programming documentation.
2010	Natural optional parameters	n/a	n/a	n/a	4.1.3	8.1	6.1.1 PL13	Optional parameters are a special Natural feature. See Natural documentation.
2020	IDL types U, UV	7.3	7.1	4.1.1	4.2.1	8.1	6.2.1	The Unicode data types support all programming languages with separate Unicode and character types such as COBOL, Natural and C. See *IDL Data Types* in the IDL Editor documentation.
	Up to 99 IDL Levels	7.3	7.1	4.1.1	4.2.1	8.1	6.2.1	IDL levels are used in conjunction with parameter grouping. Parameters are either scalar or a member of the immediately preceding group that has been assigned a lower level number. See `simple-parameter-definition` under *Software AG IDL Grammar* in the IDL Editor documentation.
2030	Natural Redefine, IDL program name longer than 8 characters for Natural, server mapping file for Natural (Designer file with extension .cvm), and other features	8.2	8.2	4.1.2	4.2.7 and 8.2.2	8.2.4	6.3.9	`REDEFINE`s are supported by IDL Extractor for Natural on the server side and force a server mapping file [(4)] to be created. See *When is a Server Mapping File Required?* in the Designer documentation for full list of features.

RPC Protocol[1]	Feature[2]	EntireX	EntireX Adapter for Integration Server	PI Adapter for EntireX	Natural			More Information
					Mainframe	ONE	Open Systems	
	Server mapping file (Designer file with extension .cvm) for COBOL	9.7	9.7	4.1.2	n/a	n/a	n/a	If a server is wrapped with the COBOL Wrapper or extracted with the IDL Extractor for COBOL, a server mapping file [3] is automatically created if one is required. See *When is a Server Mapping File Required?* under *Server Mapping Files for COBOL* in the Designer documentation.
2040	Increased precision for IDL types N, NU, P, PU	9.7	9.7	no	8.2.2	8.4.1	8.4.1	If more than 7 digits after the decimal point or more than 29 digits in total (before and after the decimal point) are used. For EntireX, the maximum number of supported digits depends on language and operating system. See notes for data types N, NU, P and PU in chapter *Mapping IDL Data Types* in the respective Wrapper or language-specific documentation.
2050	Long RPC user ID and RPC password	10.3	10.3	no	9.1.1	9.1.1	9.1.1	The RPC user ID/password pair is designed to be used by the receiving RPC server. This component's configuration determines whether the pair is considered or not. Useful scenarios are: ■ Credentials for Natural Security ■ Impersonation in the respective RPC Server documentation ■ Web Service Transport Security with the RPC Server for XML/SOAP, see *XML Mapping Files* ■ Service execution with client credentials for EntireX Adapter Listeners, see *Configuring Listeners* ■ etc.

RPC Protocol[(1)]	Feature[(2)]	EntireX	EntireX Adapter for Integration Server	PI Adapter for EntireX	Natural			More Information
					Mainframe	ONE	Open Systems	
								There is no limitation on the length of the RPC user ID and RPC password sent by an RPC client. A length restriction may come into account by the receiving RPC server when the RPC server validates the credentials from the calling RPC client against a security system.

n/a	=	not applicable
no	=	not supported

Notes:

1. To enable communication it is not required that both partners (RPC client and RPC server) support the same level of RPC protocol. There is a handshake to negotiate the highest protocol level supported by both ends. The RPC protocols 1110 thru 1140 (which are not documented here) may occur in communications as well if older RPC components are used.
2. To enable communication the feature used must be supported by both partners (RPC client and RPC server), otherwise communication is not possible.
3. If a server mapping file with extension .cvm is used at runtime, protocol 2030 is forced.
   If a server mapping file with extension .svm is used at runtime, the protocol level is not increased. For more information see *Server Mapping Files for COBOL*.

   Server mapping files with extension .svm are no longer supported at design time by the Designer. You can still use them at runtime in a server-side mapping container. All special COBOL syntax and features supported by server mapping files with extension .svm are also covered by server mapping files with extension .cvm. See *When is a Server Mapping File Required?* We recommend migrating .svm files to .cvm files. See *Migrating Server Mapping Files* under *Server Mapping Files for COBOL* in the Designer documentation.
4. For more information see *Server Mapping Files for Natural* in the Designer documentation.