Natural

Natural RPC (Remote Procedure Call)

Version 9.3.4

October 2026

ADABAS & NATURAL

This document applies to Natural Version 9.3.4 and all subsequent releases.

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

Copyright © 1992-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: NATUX-NNATRPC-934-20261013

Table of Contents

Preface vii
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Introducing Natural RPC 5
About this Documentation 6
General Information 6
Natural RPC Operation in Non-Conversational Mode 8
Natural RPC Operation in Conversational Mode 12
Conversational versus Non-Conversational Mode 13
Database Transactions 16
Handling of Limits Set with Profile Parameters LT, MAXCL, MADIO, and MT on Server 17
Location of Conversations 17
Natural RPC Terminology 18
3 Prerequisites and Preparatory Information 21
Products Involved 22
Natural Statements Involved 23
Natural Utilities for Use with Natural RPC 23
Application Programming Interfaces for Use with Natural RPC 24
Software AG IDL to Natural Mapping 24
4 Restrictions and Limitations 31
User Context Transfer 32
System Variable Transfer 32
Application-Independent Variables 32
Parameter Handling in Error Situations 33
Variable Arrays in Subprograms 33
X-Arrays 33
Groups and Interface Objects 34
Group Arrays on the RPC Server Side 34
Unsupported Natural Data Formats 34
EntireX RPC Server 35
Using VSAM 35
Natural Statement Reactions 35
Notes on Natural Statements on the Server 36
5 Setting Up a Natural RPC Environment 37
Setting Up Natural RPC with Natural Security 38
Setting Up a Natural Client 38
Setting Up a Natural Server 40
Setting Up an EntireX Broker Access 43
Setting Up an EntireX Broker Environment 44
Set Application Error on RPC Server 45

6 Starting a Natural RPC Server 47
Preliminaries before Starting a Natural RPC Server 48
Starting a Natural RPC Server in a z/OS Online Environment (all TP Monitors) 49
Starting a Natural RPC Server in a z/OS Online Environment (CICS only) 49
Starting a Natural RPC Server in a z/OS Online Environment (Com-plete only) 51
Starting a Batch Server in a z/OS Environment 52
Starting a Natural RPC Server in a Windows Environment 53
Starting a Natural RPC Server in a Linux Environment 54
Considerations for z/OS Natural RPC Servers with Replicas 54
Starting a Natural RPC Server Using the RPC Server Front-End (z/OS Batch Mode only) 55
Starting a Natural RPC Server Using the RPC Server Front-End (CICS only) 58
7 Terminating a Natural RPC Server 61
Using SYSRPC 62
Using EntireX System Management Hub 62
Using Application Programming Interface USR2073N 63
Using Application Programming Interface USR2075N 64
Using Application Programming Interface USR8208N 65
Using Application Programming Interface USR8213N 66
Using Application Programming Interface USR8220N 66
Using Application Programming Interface USR9175N 67
Using Application Programming Interface USR9176N 67
Using Application Programming Interface USR9177N 68
Server Termination When Using an Attach Manager 68
User Exit NATRPC99 68
8 Operating a Natural RPC Environment 71
Specifying RPC Server Addresses 72
Interface Objects and Automatic RPC Execution 75
Getting or Setting RPC Profile Parameters during a Natural Session 77
Executing Server Commands 77
Sending Command to RPC Server 78
Logon to a Server Library 79
Using the Logon Option 80
Using Compression 81
Using Secure Socket Layer 82
Monitoring the Status of an RPC Session 83
Retrieving Runtime Settings of a Server 90
Setting or Getting Parameters for EntireX 91
Maintain Message ID and Correlation ID of EntireX 93
Handling Errors 94
User Exits before and after Service Execution 95
9 Using a Conversational RPC 97
Opening a Conversation 98

Closing a Conversation ... 99
Defining a Conversation Context ... 100
Modifying the System Variable *CONVID ... 100
10 Reliable RPC ... 101
General Information ... 102
Reliable RPC on the Natural RPC Client Side ... 103
Reliable RPC on the Natural RPC Server Side ... 106
Viewing the Status of Reliable RPC Messages ... 106
11 Using a Remote Directory Server - RDS ... 109
RDS Principles of Operation ... 110
Using a Remote Directory Server ... 112
Creating an RDS Interface ... 113
Creating a Remote Directory Service Routine ... 114
Remote Directory Service Program RDSSCDIR ... 115
12 Using Security ... 119
Using Natural RPC with Natural Security ... 120
Impersonation (z/OS Batch Mode) ... 126
Impersonation (CICS) ... 130
Using Natural RPC with EntireX Security ... 136
13 EntireX Broker Support ... 141
Security ... 142
Logging and Accounting ... 142
14 API for Providing an RPC Context from the Natural Client Side ... 143
RPC-CNTX ... 144
15 APIs for Use with Natural RPC ... 147

Preface

This documentation is organized under the following headings:

Introducing Natural RPC	Provides basic information, such as Natural RPC operation in non-conversational and conversational mode; describes the database transactions on the client and server side and contains a list of important key terms used in the `SYSRPC` utility and the Natural RPC documentation.
Prerequisites and Preparatory Information	Provides an overview of the general prerequisites and, a short description of the facilities that are available in Natural for implementing a Natural RPC (Remote Procedure Call) environment, and information on the specific mapping of IDL data types, groups, arrays and structures to the Natural programming language.
Restrictions and Limitations	Informs you about some restrictions and limitations that you should observe when using the Natural RPC.
Setting up a Natural RPC Environment	Describes the fundamental steps you must perform for all client and server Naturals to set up a Natural RPC environment.
Starting a Natural RPC Server	How to start a Natural RPC server on the different platforms.
Terminating a Natural RPC Server	Describes the various methods of terminating a Natural RPC server, for example by terminating the EntireX Broker service supporting the RPC connection.
Operating a Natural RPC Environment	How to operate a Natural RPC environment.
Using a Conversational RPC	How to use the Natural RPC in conversational mode: opening/closing a conversation, defining a conversation context, modifying the system variable `*CONVID` if multiple conversations are used in parallel.
Reliable RPC	Describes the Reliable RPC, which is the Natural RPC implementation of a reliable messaging system.
Using a Remote Directory Server (RDS)	Describes the principle and the usage of an RDS: how to create an RDS interface, a remote directory service routine; information about the RDS directory service program `RDSSCDIR` required to read directory information from a work file.
Using Security	How to use the Natural RPC with Natural Security or EntireX Security.
EntireX Broker Support	Special considerations concerning EntireX Broker support.
API for Providing an RPC Context from the Natural RPC Client Side	A description of the API RPC-CNTX.
APIs for Use with Natural RPC	Describes the Application Programming Interfaces available in the Natural library SYSEXT, which are intended for being used with the Natural RPC.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

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**.

Product Training

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

Tech Community

You can collaborate with Software GmbH 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 GmbH 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 GmbH resources.

Product Support

Support for Software GmbH 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 Introducing Natural RPC

- About this Documentation 6
- General Information 6
- Natural RPC Operation in Non-Conversational Mode 8
- Natural RPC Operation in Conversational Mode 12
- Conversational versus Non-Conversational Mode 13
- Database Transactions 16
- Handling of Limits Set with Profile Parameters LT, MAXCL, MADIO, and MT on Server 17
- Location of Conversations 17
- Natural RPC Terminology 18

About this Documentation

Remote procedure call (RPC) techniques establish a framework for communication between server and client systems that can be collocated on the same computer or based on a network of identical or heterogeneous machines and operating systems. Several basically similar methods are known.

This documentation describes the theory of operation and the use of the RPC techniques provided by Natural to enable the design and to simplify the application of distributed software systems. For information on other products that may be involved in a Natural RPC-based environment, see the documentation of EntireX RPC for 3GL, Entire Network, EntireX Broker.

For full details of the functions provided to maintain remote procedure calls, refer to the Natural *SYSRPC Utility* documentation.

This documentation applies to all platforms on which Natural can be used. But, depending on the Natural documentation set you are currently using, the following differences exist:

- The examples of using the Natural utility `SYSRPC` exhibit platform-specific maps (with either GUI or CUI interface).
- Under Natural for Windows and Linux, the RPC-specific parameters are available as profile parameters.
- Under Natural for z/OS, the parameters are available as keyword subparameters of profile parameter `RPC` or parameter macro `NTRPC`.

Notation *vrs* or *vr*

When used in this documentation, the notation *vrs* or *vr* represents the relevant product version (see also *Version* in the *Glossary*).

General Information

- Purpose
- Advantages of Natural RPC (Remote Procedure) Calls
- Natural RPC Modes of Operation
- Availability on Various Platforms

- Support of Non-Natural Environments (EntireX RPC)

Purpose

The Natural RPC facility enables a client Natural program to issue a `CALLNAT` statement to invoke a subprogram in a server Natural. The Natural client and server sessions may run on the same or on a different computer. For example, a Natural client program on a Windows computer can issue a `CALLNAT` statement against a z/OS server in order to retrieve data from a z/OS database. The same Windows computer can act as a server if a Natural client program running under, for example, Linux issues a `CALLNAT` statement requesting data from this server Natural.

Advantages of Natural RPC (Remote Procedure) Calls

Natural RPC exploits the advantages of client server computing. In a typical scenario, Natural on a Windows client computer accesses server data (using a middleware layer) from a Natural on a z/OS computer. The following advantages arise from that:

- The end user on the client side can use a Natural application with a graphical user interface.
- A large database can be accessed on a z/OS server.
- Network traffic can be minimized when only relevant data are sent from client to server and back.

Natural RPC Modes of Operation

The Natural RPC (Remote Procedure Call) offers the following modes of operation:

- **non-conversational mode** (unless otherwise specified, the following text refers to this mode)
- **conversational mode**

These modes are described in detail in the following sections. For a comparison of the advantages and disadvantages of these modes, refer to ***Conversational versus Non-Conversational Mode***.

Availability on Various Platforms

You can use Natural RPC on various platforms under the following operating systems:

- **z/OS Environments**
  Natural RPC on z/OS is supported under the following TP monitors:
  - Com-Plete
  - CICS
  - TSO

  It is also available in batch mode.

Natural can act as both client and server on all these platforms.

- **Other Environments**
  - Windows
  - Linux

  Natural can act as both client and server on all these platforms.

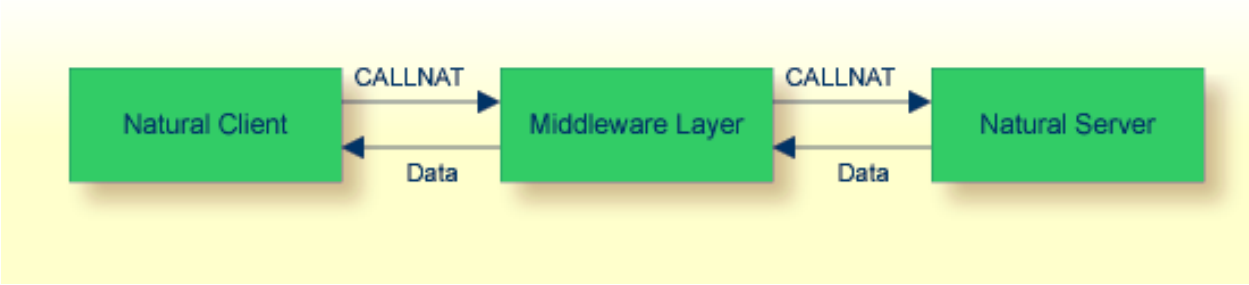
Support of Non-Natural Environments (EntireX RPC)

Non-Natural environments (3GL and other programming languages) are supported on the client and the server side. Thus, a non-Natural client can communicate with a Natural RPC server, and a Natural client can communicate with a non-Natural RPC server. This is enabled by the use of the EntireX RPC.

Natural RPC Operation in Non-Conversational Mode

The non-conversational mode should be used only to accomplish a single exchange of data with a partner. See also **Conversational versus Non-Conversational Mode.**

The Natural RPC technique uses the Natural statement `CALLNAT`, so that both local and remote subprogram calls can be issued in parallel. Remote program calls work synchronously. As a remote procedure call, a `CALLNAT` would, simply speaking, take the following route:



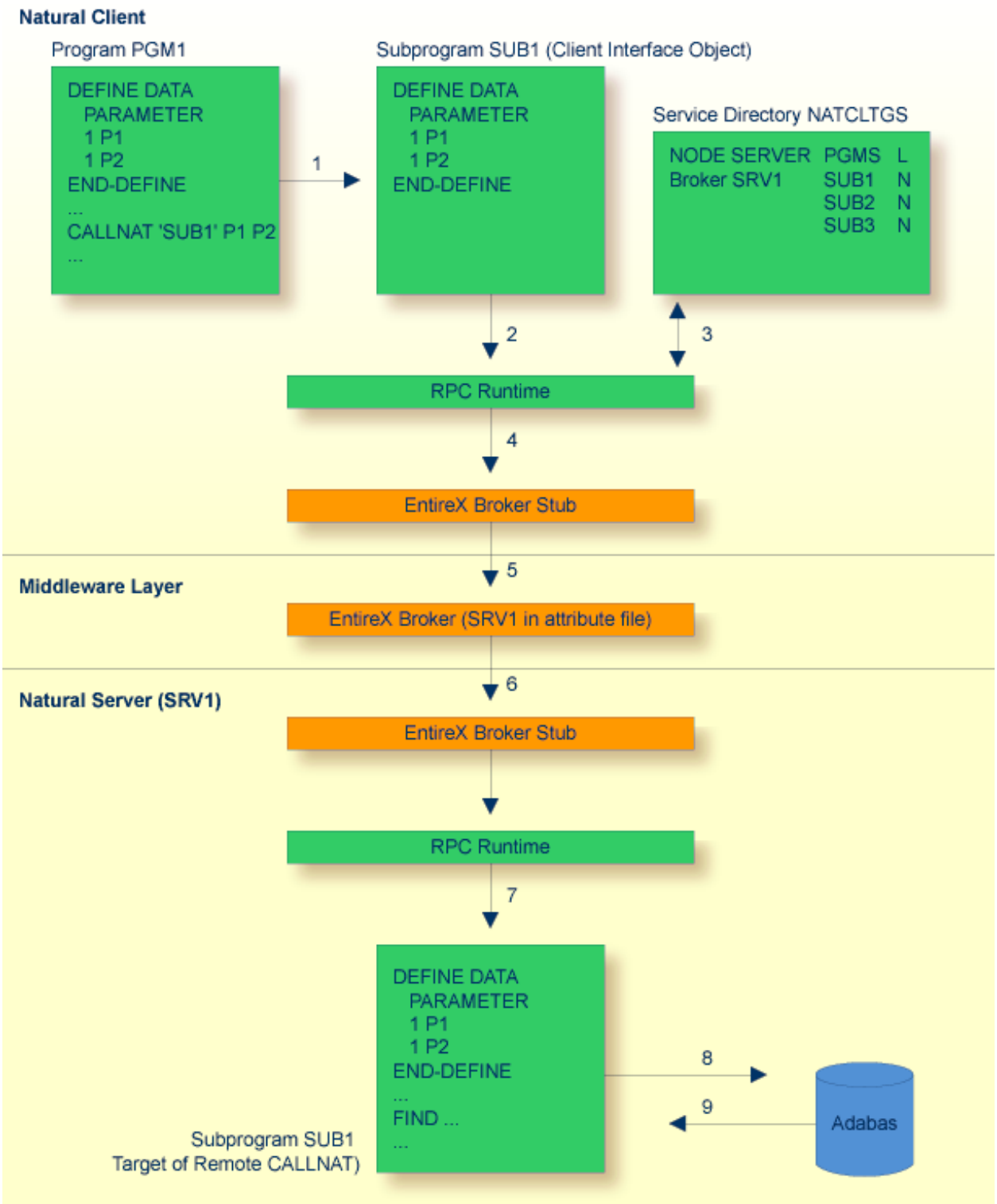
The `CALLNAT` issued from the Natural client is routed via a middleware layer to the Natural server which passes data back to the client.

Usually, the middleware layer consists of the EntireX Broker which uses the ACI protocol. EntireX Broker uses either Entire Net-Work or TCP/IP as communication layer.

A detailed example of the RPC control flow is described below.

Issuing CALLNATs in an RPC Environment

CALLNAT control flow details in a remote procedure are illustrated below. For greater clarity, the return path is not shown, but it is analogous; the numbers refer to the description:



1. From the Natural client, the program `PGM1` issues a `CALLNAT` to the subprogram `SUB1`. `PGM1` does not know if its `CALLNAT` will result in a local or in a remote `CALLNAT`.

As the target SUB1 resides on a server, the CALLNAT accesses an **interface object** SUB1 instead. This client interface object has been created automatically or manually (by using the SYSRPC utility's interface object generation (IG) function).

The interface object has the same name as the target subprogram and contains parameters identical with those used in program PGM1 and in the target subprogram SUB1 on the server. It also contains control information used internally by the RPC.

If the AUTORPC profile parameter is set to ON and Natural cannot find the subprogram in the local environment, Natural interprets this as a remote procedure call and generates the parameter data area (PDA) dynamically during runtime.

Natural also tries to find this subprogram in the service directory NATCLTGS.

For further information on the SYSRPC interface object generation function, see ***Creating Interface Objects***.

If you want to work without interface objects, see ***Working with Automatic Natural RPC Execution***.

2. The interface object then sets up a CALLNAT to an RPC client service routine.
3. The client RPC runtime checks in the service directory NATCLTGS on which node and server the CALLNAT is to be performed and whether a logon is required.

   The CALLNAT data including the parameter list and, if required, the logon data are passed to a middleware layer.
4. In this example, this middleware layer consists of the EntireX Broker. Therefore, the CALLNAT data is first passed to an EntireX Broker stub on the client.
5. From the EntireX Broker stub, the CALLNAT data is passed to the EntireX Broker. The EntireX Broker is a product that can reside:
   - on the client computer
   - on the server computer or
   - on a third platform.

   For the data to be passed on successfully, the server SRV1 must be defined in the EntireX Broker attribute file and SRV1 must be already up, thus having registered with EntireX Broker.

   For information on how to define servers in the EntireX Broker attribute file, see the EntireX Broker documentation.
6. From the middleware layer, the CALLNAT data is passed on to the EntireX Broker stub on the Natural Server platform and from there to the RPC server service routine.

   The RPC server service routine validates the logon data (if present) and performs a logon (if requested).

7. The RPC server service routine invokes the target subprogram `SUB1` and passes the data, if requested.

   At this point, the target subprogram `SUB1` has all the required data to execute just as if it had been invoked by a local program `PGM1`.
8. Then, for example, the subprogram `SUB1` can issue a `FIND` statement to the server's Adabas database. `SUB1` does not know whether it has been started by a local or by a remote `CALLNAT`.
9. Adabas `FIND`s the data and passes them to `SUB1`.

   Then, `SUB1` returns the Adabas data to the calling server service routine. From there, it is passed it back to `PGM1` via the middleware layer. It takes the same route as described in Steps 1 to 8, but in reverse order.

Natural RPC Operation in Conversational Mode

A conversational RPC is a static connection of limited duration between a client and a server. It provides a number of services (subprograms) defined by the client, which are all executed within one server task that is exclusively available to the client for the duration of the conversation. It is implemented in a program using an `OPEN CONVERSATION` statement and a `CLOSE CONVERSATION` statement.

Multiple connections (conversations) can exist at the same time. They are maintained by the client by means of conversation IDs, and each of them is performed on a different server. Remote procedure calls which do not belong to a given conversation are executed on a different server, within a different server task.

During a conversation, you can define and share a data area called context area between the remote subprograms on the server side. For further information, see *Defining Context Variables for Natural RPC* in the Natural *Statements* documentation.

A conversation may be local or remote.

Example:

```
OPEN CONVERSATION USING SUBPROGRAM 'S1''S2'
     CALLNAT 'S1' PARMS1
     CALLNAT 'S2' PARMS2
CLOSE CONVERSATION ALL
```

Both subprograms (`S1` and `S2`) must be accessed at the same location, that is, either locally or remotely. It is not admissible to mix up local and remote `CALLNAT`s within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server task.

Analogously to non-conversational RPC `CALLNAT`s, conversations may first be written and tested locally and can then be transferred to the servers.

General Rules for Local and Remote Subprogram Execution

- **Local Subprogram Execution**
  If you execute subprograms locally, a subprogram may not call another subprogram which is a member of the conversation.

  Other subprograms not listed in the `OPEN CONVERSATION` statement may be called. They are however executed in **non-conversational mode**.
- **Remote Subprogram Execution**
  If you execute subprograms remotely, a subprogram `S1` may call another subprogram `S2` which is a member of the conversation.

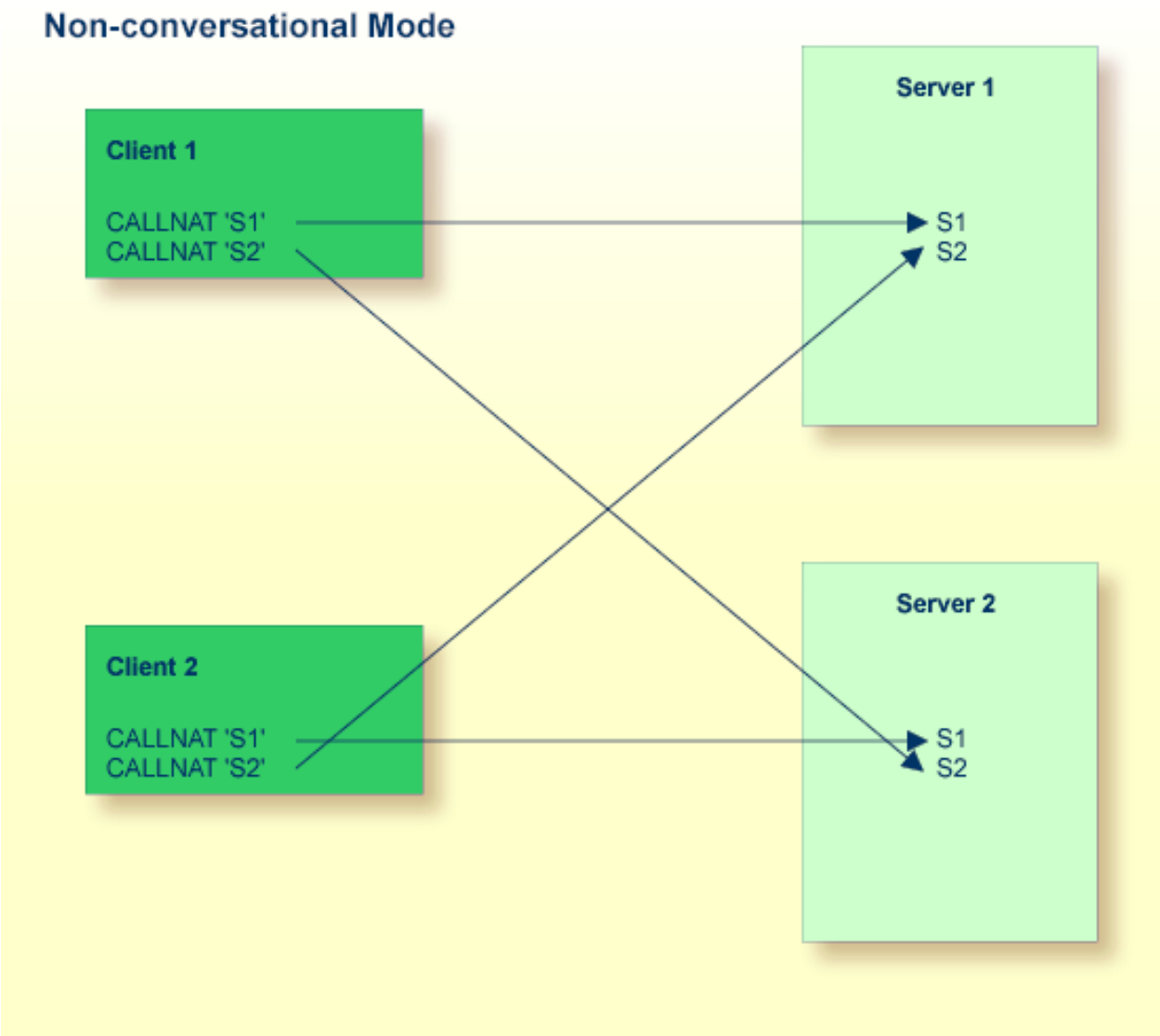
  This `CALLNAT` will be executed in non-conversational mode because it was invoked indirectly. Thus, the subprogram S2 does not have access to the context area.

Conversational versus Non-Conversational Mode

In a client-server environment where several clients access several servers in non-conversational mode, there may be the problem that identical `CALLNAT` requests from different clients are executed on the same server.

This means, for example, that a `CALLNAT 'S1'` from Client 1 executes Subprogram `S1` on Server 1 (`S1` is writing a record to the database). The transaction for Client 1 is not yet complete (no `END TRANSACTION`) when Client 2 also sends a `CALLNAT 'S1'` to Server 1, thus overwriting the data from Client 1. If Client 1 then sends a `CALLNAT 'S2'` (meaning `END TRANSACTION`), Client 1 supposes its data have been saved correctly, although in fact the data from Client 2's identical `CALLNAT` were saved.

The diagram below illustrates this with two clients and two servers. In such a scenario, you cannot control whether two identical `CALLNAT`s from two different clients access the same subprogram on the same server:

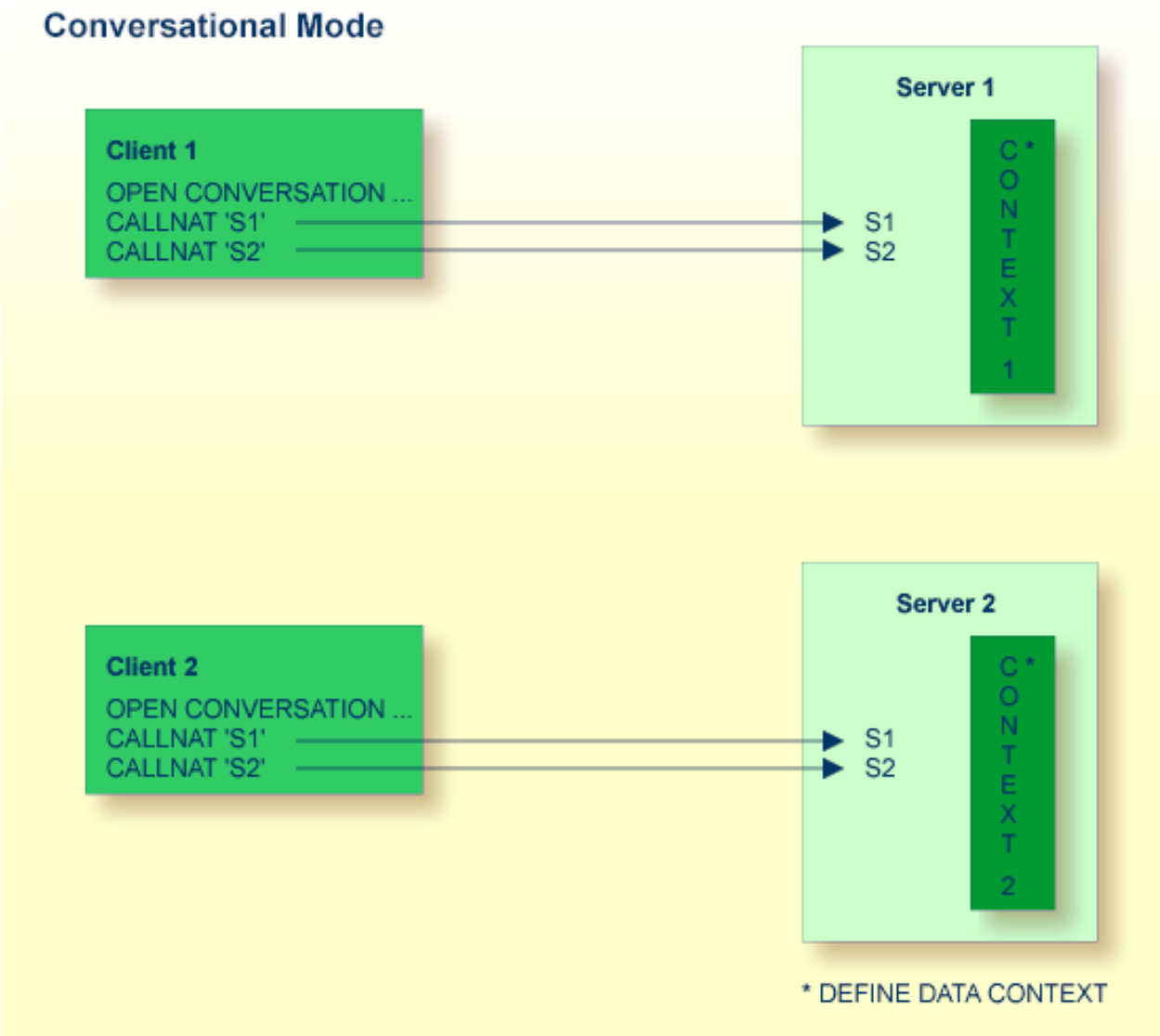


In the above example, `CALLNAT 'S2'` from Client 1 can access subprogram `S2` on Server 1 and on Server 2. `CALLNAT 'S2'` from Client 2 has the same choice.

Similarly, `CALLNAT 'S1'` from Client 1 could access Subprogram `S1` on Server 1 and on Server 2, while `CALLNAT 'S1'` from Client 2 has the same choice.

It is obvious that interference can be a problem here if the subprograms are designed to be executed within one server task context.

You can avoid the potential problems of a non-conversational RPC by defining a more complex RPC transaction in conversational mode:



You do this by opening a conversation. This involves the use of the `OPEN CONVERSATION` statement on the client side, referring to `CALLNAT 'S1'` and `CALLNAT 'S2'`. Opening such a conversation reserves one entire server task (for example, Server 1) and no other remote `CALLNAT`s may interrupt this conversation on this server before this conversation has been closed. In addition, you can define a common context area for the two subprograms on the server side by using the `DEFINE DATA CONTEXT` statement.

General Rules for Use of Conversational and Non-Conversational RPC

As a general rule, the following applies:

- Use the **conversational RPC** to ensure that a defined list of subprograms is executed exclusively within one context.
- Use the **non-conversational RPC** if each of your subprograms can be used within a different server task or if the transaction does not extend over more than one server call. The advantage of this is that no server blocks over a significant amount of time and you only need a relatively small number of server tasks.

Possible Disadvantage of Using Conversational RPC

A possible disadvantage of conversational RPCs is that you reserve an entire server task, thus blocking all other subprograms on this server. As a consequence, other `CALLNAT`s might have to wait or more server tasks must be started.

Database Transactions

The database transactions on the client and server sides run independent of each other. That is, an `END TRANSACTION` or `BACKOUT TRANSACTION` executed on the server side does not influence the database transaction on the client side and vice-versa.

At the end of each non-conversational `CALLNAT` and at the end of each conversation, an implicit `BACKOUT TRANSACTION` is executed on the server side. To commit the changes made by the remote `CALLNAT`(s), you have the following options:

Non-conversational CALLNAT

1. Execute an explicit `END TRANSACTION` before leaving the `CALLNAT`.
2. Set the Natural profile parameter `ETEOP` to `ON`. This results in an implicit `END TRANSACTION` at the end of each non-conversational `CALLNAT`.

   Depending on the setting of the parameter `SRVCMIT`, the `END TRANSACTION` is executed either before the reply is sent to the client (`SRVCMIT=B`) or after the reply has been successfully sent to the client (`SRVCMIT=A`). `SRVCMIT=B` is the default and is compatible with earlier versions of the RPC.

Conversational CALLNAT

1. Execute an explicit `END TRANSACTION` on the server before the conversation is terminated by the client
2. Set the Natural profile parameter `ETEOP` to `ON`. This results in an implicit `END TRANSACTION` at the end of each conversation.

Depending on the setting of the parameter SRVCMIT, the END TRANSACTION is executed either before the reply is sent to the client (SRVCMIT=B) or after the reply has been successfully sent to the client (SRVCMIT=A). SRVCMIT=B is the default and is compatible with earlier versions of the RPC.

3. Before executing the CLOSE CONVERSATION statement, call the application programming interface USR2032N on the client side. This will cause an implicit END TRANSACTION at the end of the individual conversation.

Handling of Limits Set with Profile Parameters LT, MAXCL, MADIO, and MT on Server

The following Natural limits apply to each individual remote CALLNAT execution on the Natural RPC server:

Natural Profile Parameter	Type of Limit
LT	Limit for Processing Loops
MADIO	Maximum DBMS Calls between Screen I/O Operations
MAXCL	Maximum Number of Program Calls

These limits are reset after each remote CALLNAT execution, and you can enforce certain limits on each remote CALLNAT execution.

This measure helps to avoid unpredictable error situations when the execution of a client request reaches one of the limits due to the processing of a previous client request.

Location of Conversations

Both subprograms S1 and S2 (shown in the figure above) must be accessed at the same location, i.e. either locally or remotely. You may not mix up local and remote CALLNATs within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server task.

Natural RPC Terminology

The following table provides an overview of important key terms used in the SYSRPC Utility and the Natural RPC documentation:

Term	Explanation
Client Stub	Obsolete term, see ***Client Interface Object*** below.
EntireX Broker Stub	Interface between the Natural RPC runtime and the EntireX Broker transport layer which exchanges marshalled data between client and server.
Impersonation	Impersonation assumes that access to the operating system on which a Natural RPC server is running is controlled by an SAF-compliant external security system. User authentication is performed by this external security system. Impersonation means that after the authentication has been successful and the user's identity is established, any subsequent authorization checks will be performed based on this identity. This includes authorization checks for access to external resources (for example, databases or work files). After successful authentication the user cannot "change his/her identity", that is, he/she cannot use a different user ID. See ***Impersonation*** in ***Using Security***.
Interface Object	In earlier versions of EntireX, the term "stub" was also used to refer to application-dependent, Workbench-generated pieces of code for issuing and receiving remote procedure calls. These objects are now referred to as *interface objects*.
Client Interface Object	Accepts the CALLNAT requests on the client side, marshalls the parameters passed, transmits the data through the Natural RPC runtime and the transport layer to the remote server, unmarshalls the result and returns it to the caller. The interface object stub is the local subprogram via which the server subprogram is called. The client interface object has the same name and contains the same parameters as the corresponding server subprogram.
NATCLTGS	The name of the Natural subprogram generated with the SYSRPC utility to implement the **service directory** (see below).
Node Name	The name of the node to which the remote CALLNAT is sent. In case of communication via the EntireX Broker, the node name is the name of the EntireX Broker for example, as defined in the EntireX Broker attribute file, in the field BROKER-ID.
RPC Parameters	All parameters available to control a Natural RPC are described in detail in the Natural Parameter Reference documentation. See the section *Profile Parameters*.
RPC Server	An RPC server is either a Natural or an EntireX RPC server.
Service Directory	The service directory contains information on the services (subprograms) that a server provides. It can be locally available on each client node, or it can be located on a remote directory server, which is referenced on Windows and Linux by profile parameter RDS, and on z/OS by the corresponding keyword subparameter RDS of profile parameter RPC or parameter macro NTRPC.
Server Name	The name of the server on which the CALLNAT is to be executed.

	In case of communication via EntireX Broker, the server name is the name as defined in the EntireX Broker attribute file in the field `SERVER`.
Server Task	A Natural task which offers services (subprograms). This is typically a batch task or asynchronous task. It is identified by a server name.

3 Prerequisites and Preparatory Information

- Products Involved 22
- Natural Statements Involved 23
- Natural Utilities for Use with Natural RPC 23
- Application Programming Interfaces for Use with Natural RPC 24
- Software AG IDL to Natural Mapping 24

This document provides an overview of the general prerequisites and a short description of the facilities that are available in Natural for implementing a Natural RPC (Remote Procedure Call) environment.

Products Involved

If the RPC environment is to be implemented on different platforms, the corresponding current versions of Natural for z/OS, Windows or Linux will be required. In addition, the following required or optional products, subproducts and facilities are available for use in a Natural RPC environment:

Product	Purpose
EntireX Communicator (EntireX Broker)	The EntireX Broker of the EntireX Communicator usually establishes the middleware layer between the Natural client and a Natural server. It uses the ACI protocol and comprises the appropriate client/server stubs.
Entire Net-Work	This product is required if the transport method used by EntireX Broker is Entire Net-work. This is the preferred transport method.
TCP/IP	Required if the transport method used by EntireX Broker is TCP/IP. See also ***Using TCP/IP as a Transport Method*** in ***Setting Up a Natural RPC Environment***.
EntireX Developer's Kit	Included in the EntireX Communicator product, this kit is required for non-Natural programming language support.
Directory Services	A remote directory server (RDS) enables you to define directory definitions in one place so that its services can be used by all clients in an RPC environment.
Natural Security	Optional. This add-on product is required on the server side to protect Natural RPC servers and services when security is active on the client and vice versa.
International Components for Unicode (ICU)	Optional. This add-on product is required on the server side if the Natural RPC server is protected by Natural Security and the RPC clients have to provide user ID and password to access services.
EntireX RPC	Optional. Natural RPC fully supports EntireX RPC for 3GL. EntireX RPC is included in the EntireX Communicator product.
EntireX Security	Optional. Natural RPC fully supports EntireX Security on the client and on the server side. EntireX Security is included in the EntireX Broker product.

For information on the supported operating system platforms and other requirements for the current version of Natural, see *System Requirements* in the *Installation* documentation.

For information on other products that may be involved in a Natural RPC-based environment, see the corresponding product documentation.

Natural Statements Involved

The following Natural statements are used in the creation of a Natural RPC environment:

- `CALLNAT`
- `DEFINE DATA PARAMETER`
- `DEFINE DATA CONTEXT`
- `OPEN CONVERSATION`
- `CLOSE CONVERSATION`

In the section ***Restrictions and Limitations***, the paragraphs ***Natural Statement Reactions*** and ***Notes on Natural Statements on the Server*** provide information on what you should know about any deviating behavior of these statements when they are used in a Natural RPC environment.

Natural Utilities for Use with Natural RPC

The following Natural utilities are used in the creation and maintenance of a Natural RPC environment:

- `SYSRPC`

  This utility is used to maintain remote procedure call environments.

- `SYSEXT`

  This utility is used to locate and test Natural Application Programming Interfaces (APIs, see below) contained in the current system library `SYSEXT`.

- `SYSPARM`

  On z/OS, this utility is used for creating and maintaining a set of Natural profile parameters that is stored under a profile name.

- Configuration Utility

  Under Windows and Linux, this utility is used to modify global and local configuration files and to create or modify parameter files.

Application Programming Interfaces for Use with Natural RPC

The purpose of Natural Application Programming Interfaces (API) is to retrieve or modify information or use services that are not accessible by Natural statements.

The APIs in the Natural library `SYSEXT` which are used with the Natural RPC are listed in ***APIs for Use with Natural RPC***.

For further information about how to use a Natural API, see the *Utilities > SYSEXT Utility - Natural Application Programming Interfaces*.

For a list of all RPC-relevant APIs, enter the keyword `RPC` in the **Keyword** field in the *SYSEXT utility*.

Software AG IDL to Natural Mapping

This section describes the specific mapping of Software AG IDL data types, groups, arrays and structures to the Natural programming language. Please note also the remarks and hints on the IDL data types valid for all language bindings found in the Software AG IDL File (in the *EntireX* documentation).

- Mapping Software AG IDL Data Types to Natural Data Formats
- Mapping Library Name and Alias
- Mapping Program Name and Alias
- Mapping Parameter Names
- Mapping Fixed and Unbounded Arrays
- Mapping Groups and Periodic Groups
- Mapping Structures
- Mapping the Direction Attributes IN, OUT, INOUT
- Mapping the ALIGNED Attribute
- Calling Servers as Procedures or Functions

Mapping Software AG IDL Data Types to Natural Data Formats

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

Software AG IDL	Description	Natural Data Format	Note
A*number*	Alphanumeric	A*number*	
AV	Alphanumeric variable length	A DYNAMIC	
AV*number*	Alphanumeric variable length with maximum length	A DYNAMIC	
B*number*	Binary	B*number*	
BV	Binary variable length	B DYNAMIC	
BV*number*	Binary variable length with maximum length	B DYNAMIC	
D	Date	D	3,5
F4	Floating point (small)	F4	2
F8	Floating point (large)	F8	2
I1	Integer (small)	I1	
I2	Integer (medium)	I2	
I4	Integer (large)	I4	
K*number*	Kanji	A*number*	1
KV	Kanji variable length	A DYNAMIC	1
KV*number*	Kanji variable length with maximum length	A DYNAMIC	1
L	Logical	L	
N*number*[.*number*]	Unpacked decimal	N*number*[.*number*]	
NU*number*[.*number*]	Unpacked decimal unsigned	N*number*[.*number*]	
P*number*[.*number*]	Packed decimal	P*number*[.*number*]	
PU*number*[.*number*]	Packed decimal unsigned	P*number*[.*number*]	
T	Time	T	3,4
U*number*	Unicode	U*number*	
UV	Unicode variable length	U DYNAMIC	
UV*number*	Unicode variable length with maximum length	U DYNAMIC	

See also the hints and restrictions on the Software AG IDL Data Types (in the *EntireX* documentation) valid for all language bindings.

Notes:

1. Data type K is an RPC-specific data format that is not part of the Natural language.
2. When floating-point data types are used, errors due to rounding can occur, so that the values of senders and receivers might differ slightly. This is especially true if client and server use different representations for floating point data (IEEE, HFP).
3. Count of days AD (anno domini, after the birth of Christ). The valid range is from 1.1.0001 up to 28.11.2737. Mapping of the number to the date in the complete range from 1.1.0001 on, follows the Julian and Gregorian calendar, taking into consideration the following rules:

   a. Years that are evenly divisible by 4 are leap years.

b. Years that are evenly divisible by 100 are not leap years unless rule 3, below, is true.

c. Years that are evenly divisible by 400 are leap years.

d. Before the year 1582 AD, rule 1 from the Julian calendar is used. After the year 1582 AD, rules 1, 2 and 3 of the Gregorian calendar are used.

See the following table for the relation of the packed number to a real date:

Date / Range of Dates	Value / Range of Values
1.1.0000	0 (special value - no date)
undefined dates	1 - 364 (do not use)
1.1.0001	365
1.1.1970	719527 (start of C-time functions)
28.11.2737	999999 (maximum date)

4. Count of tenth of seconds AD (anno domini, after the birth of Christ). The valid range is from 1.1.0001 00:00:00.0 up to 16.11.3168 9:46:39 plus 0.9 seconds. See the following table for the relation of the packed number to a real time:

Time / Range of Times	Value / Range of Values
1.1.0000 00:00:00.0	0 (special value - no date)
undefined times	1 - 315359999
1.1.0001 00:00:00.0	315360000
1.1.1970 00:00:00.0	621671328000 (start of C-time functions)

5. The relation between the packed number of a Date and Time data type is as follows:

tenths of a second per day = 24*60*60*10 = 864000

number of time	= number of date	* 864000
315360000	= 365	* 864000 1.1.0001 00:00:00.0
621671328000	= 719527	* 864000 1.1.1970 00:00:00.0
number of date	= number of time	/ 864000
365	= 315360000	/ 864000 1.1.0001
719527	= 621671328000	/ 864000 1.1.1970

Mapping Library Name and Alias

The library name as specified in the IDL file is not supported by Natural. By default, a Natural client sends the library name `SYSTEM` to the server. To send a library name other than `SYSTEM` from a client to a server, the following steps are required for the client:

To send a library name other than `SYSTEM` from a client to a server

1. On the client, turn on the logon option.
2. Call application programming interface `USR4008N` to specify the name of the library, otherwise the name of the current library is sent

The length of the library name is limited to 8 characters.

Mapping Program Name and Alias

The program name is sent from a client to the server. Special characters are not replaced. The program alias is not sent to the server.

The generated Natural interface object has the same name.

In the RPC server, the IDL program name sent is used to locate the Natural subprogram.

The length of the program name is limited to 8 characters.

Mapping Parameter Names

The parameter names as given in the parameter-data-definition of the IDL file are replaced by artificial names in the generated Natural interface object.

See *parameter-data-definition* in the section *Software AG IDL Grammar* in the *EntireX* documentation.

Mapping Fixed and Unbounded Arrays

- Fixed arrays within the IDL file are mapped to fixed Natural arrays. The lower bound is set to 1 and the upper bound is set to the upper bound given in the IDL file.

  See the *array-definition* (in the section *Software AG IDL Grammar* in the *EntireX* documentation) for the syntax on how to describe fixed arrays within the IDL file and refer to fixed-bound-array-index.

- Unbounded arrays within the IDL file are mapped to Natural X-arrays. The lower bound is always fixed and set to 1.

  See the *array-definition* (in the section *Software AG IDL Grammar* in the *EntireX* documentation) for the syntax of unbounded arrays within the IDL file and refer to unbounded-array-index.

Note: Natural variable arrays (Natural notation (../1:V)) can be used on the Natural RPC server side instead of Natural fixed arrays or X-arrays. An RPC client can pass either an IDL fixed array or IDL unbounded array to a Natural RPC server with such a Natural variable array. In the RPC server, the variable array cannot be resized; this means the number of array occurrences cannot be changed, and the Natural RPC server will always pass back the same number of occurrences.

Mapping Groups and Periodic Groups

Groups within the IDL file are mapped to Natural groups. See *group-parameter-definition* (in the section *Software AG IDL Grammar* in the *EntireX* documentation) for the syntax on how to describe groups within the IDL file.

Mapping Structures

Structures within the IDL file are mapped to Natural groups. See *structure-definition* (in the section *Software AG IDL Grammar* in the *EntireX* documentation) for the syntax on how to describe structures within the IDL file.

Mapping the Direction Attributes IN, OUT, INOUT

The IDL syntax allows you to define parameters as IN parameters, OUT parameters, or INOUT parameters (which is the default if nothing is specified). This direction specification is reflected by Natural as follows:

- Parameters with the `OUT` attribute are sent from the RPC client to the RPC server. They are always provided with the call by reference method.
- Parameters with the `IN` attribute are sent from the RPC server to the RPC client. They are always provided with the call by reference method.
- Parameters with the `INOUT` attribute are sent from the RPC client to the RPC server and then back to the RPC client.
- Only the direction information of the top-level fields (level 1) is relevant. Group fields always inherit the specification from their parent. A different specification is ignored.

See *attribute-list* (in the section *Software AG IDL Grammar* in the *EntireX* documentation) for the syntax on how to describe attributes within the IDL file and refer to direction-attribute.

Note: If you define an interface object layout in the Natural application `SYSRPC`, the meaning of the direction attributes IN and OUT are reversed compared to the IDL:

- `IN` in `SYSTRPC` is `OUT` in IDL
- `OUT` in `SYSTRPC` is `IN` in IDL

Mapping the ALIGNED Attribute

The `ALIGNED` attribute is not relevant for the programming language Natural. However, a Natural client can send the `ALIGNED` attribute to an RPC server where it might be needed. To do this you need a Natural interface object that has been generated from an IDL file.

See *attribute-list* (in the section *Software AG IDL Grammar* in the *EntireX* documentation) for the syntax of attributes in the IDL file and refer to the aligned-attribute.

Calling Servers as Procedures or Functions

The IDL syntax allows definitions of procedures only. It does not have the concept of a function. A function is a procedure which, in addition to the parameters, returns a value. Procedures and functions are transparent between clients and server. This means a client using a function can call a server implemented as a procedure, and vice versa.

Client and Server Side

The Natural RPC does not support functions.

4 Restrictions and Limitations

- User Context Transfer 32
- System Variable Transfer 32
- Application-Independent Variables 32
- Parameter Handling in Error Situations 33
- Variable Arrays in Subprograms 33
- X-Arrays 33
- Groups and Interface Objects 34
- Group Arrays on the RPC Server Side 34
- Unsupported Natural Data Formats 34
- EntireX RPC Server 35
- Using VSAM 35
- Natural Statement Reactions 35
- Notes on Natural Statements on the Server 36

This document informs you about some restrictions and limitations that you should observe when you are using the Natural RPC (Remote Procedure Call) facility.

When executing a subprogram via the RPC, certain differences to local execution apply. These are described in the following section.

User Context Transfer

Excepting the user identification, no user context is transferred to the server session, for example:

- all client session parameters remain unchanged and do not affect the execution on the server side;
- open transactions on the client side cannot be closed by the server and vice versa;
- client report handling and work file processing cannot be continued on the server side and vice versa;
- the handling of the Natural stack cannot be continued either.

System Variable Transfer

No system variables except `*USER` can be transferred from the client to the server side.

Application-Independent Variables

In an RPC server, application-independent variables (AIVs) are not deallocated implicitly, but stay active across RPC requests, because different clients may have access to the same variables on the RPC server. This means they must be deallocated explicitly using the `RELEASE VARIABLES` statement.

If you want to release AIVs at the end of an RPC request, you may use the Natural RPC user exit `NATRPC03` for this purpose.

Parameter Handling in Error Situations

Parameter handling in error situations is different:

- If an error occurs during local execution, all parameter modifications performed so far are in effect, because parameters are passed via “call by reference”.
- If an error occurs during remote execution, however, all parameters remain unchanged.

Variable Arrays in Subprograms

If the parameter data area of the subprogram contains a variable number of occurrences (`1:V` notation), you should not use an interface object to call this subprogram. As an interface object only supports array definitions with a fixed number of occurrences, you cannot vary the number of occurrences from call to call.

If you have to use an **interface object**, for example, because you want to call an EntireX RPC server with the same program, you should use an X-array on the Natural client side. With X-arrays, it is possible to vary the number of occurrences from call to call even when using an interface object. In this case, the X-array on the client side is passed to the (fixed) variable array on the server side. The variable array is fixed because the server program may receive a varying number of occurrences from call to call but cannot change the number of occurrences.

X-Arrays

X-arrays are supported in the parameter list of a remote `CALLNAT` statement execution. The server may increase or decrease the number of occurrences.

Restrictions

- In case of a multidimensional array, all dimensions of the array must be extensible.
- The lower bound must not be extensible, that is, only extensible upper bounds are allowed.
- If you want to use an X-group array that contains an array with constant bounds or a group array that contains an X-array, you must use an interface object. When generating the interface object, you must define the group structure in the Interface Object Generation screen.

  Examples:

```
01 X-Group-Array (/1:*)
02 Array (A10/1:10)
*
01 Group-Array (/1:10)
02 X-Array (A10/1:*)
```

Groups and Interface Objects

If group arrays or X-group arrays are present in the parameter list of a remote `CALLNAT` statement execution and an interface object is used, the following restrictions exist.

Restrictions

- You must not use the `AD=O` or `AD=A` session parameter settings (attribute definition) in the `CALLNAT` statement.
- Group arrays and X-group arrays passed from a client Natural object to an interface object must be contiguous. We therefore strongly recommend that you always pass a complete array to the interface object by using asterisk (*) notation for all dimensions. We also strongly recommend that you use identical data definitions in the client Natural program, in the interface object, and in the server program.

Group Arrays on the RPC Server Side

The storage layout of group arrays in the `DEFINE DATA PARAMETER` area of subprograms on the RPC server side is not necessarily identical with respect to the syntax. Do not redefine fields within a group array or pass the group array to a 3GL program. If you need to do so, copy the group array to a group array with the same layout in the `DEFINE DATA LOCAL` area and use this local group array in the call to the 3GL program.

Unsupported Natural Data Formats

The Natural data formats C (attribute control) and Handle are not allowed in the parameter list of a remote `CALLNAT` statement execution.

EntireX RPC Server

If you want to call an EntireX RPC server with a remote `CALLNAT` statement execution, it is strongly recommended to use an interface object. There are two ways to generate such an interface object:

- Generate the interface object from the EntireX IDL (Interface Definition Language) file. This is only supported with Natural Studio (Windows only) or the EntireX Workbench (Linux and Windows only).
- Define the parameters of the IDL definition of the subprogram you want to call on an EntireX RPC server manually using the Interface Object Generation screen of the `SYSRPC` utility. If the IDL (Interface Definition Language) contains a group structure, you have to define the same group structure on the Interface Object Generation screen.

Using VSAM

If you access a VSAM dataset in a subtasking environment or if you share a VSAM dataset across regions, you must consider the required share options. For example, you may have to set cross-region `SHAREOPTIONS 4` instead of `SHAREOPTIONS 2` to enforce buffer invalidation if records are inserted in a VSAM dataset in one address space and the same VSAM dataset is read in another address space. Otherwise, records recently inserted may not be found.

You should also consider using record level sharing (RLS).

For further information, refer to the relevant VSAM documentation of IBM.

Natural Statement Reactions

Several Natural statements may react in a different way when used in a Natural RPC context, for example:

Statement	Description
`OPEN CONVERSATION` `CLOSE CONVERSATION`	If executed on a server, these statements do not affect the client session. When the server itself acts as a client for another server (as agent), these statements only affect the conversations on the second server.
`PASSW`	The password setting remains active on the server side only, also for subsequent executions by other users.

`SET CONTROL`
`SET GLOBALS`
`SET KEY`
	No settings are returned to the caller.

Statement	Description
SET TIME SET WINDOW	
STACK	All stack data are released after execution.
STOP TERMINATE	These statements *do not* stop the client session. For information on how to terminate a Natural RPC server, see ***Terminating a Natural RPC Server***.

Notes on Natural Statements on the Server

The use of the following statements on an Natural RPC is theoretically possible, but not recommended, as it causes undesired effects:

Statement	Description
TERMINATE	Using this statement causes the server to be terminated, regardless of conversations that may still be open.
FETCH RUN STOP	Using these statements causes the CALLNAT context to be lost. Upon a FETCH, RUN or STOP statement, the server detects that it has lost its CALLNAT context and returns a corresponding Natural error message to the client; at that time, however, the statement has already been executed by the server. *Exception:* This does not apply to FETCH RETURN.
INPUT	Input values are unpredictable when the input data are read from a file (and not from the stack).

5 Setting Up a Natural RPC Environment

- Setting Up Natural RPC with Natural Security 38
- Setting Up a Natural Client 38
- Setting Up a Natural Server 40
- Setting Up an EntireX Broker Access 43
- Setting Up an EntireX Broker Environment 44
- Set Application Error on RPC Server 45

To set up a Natural RPC environment, you must perform the steps described below for all client Naturals and server Naturals and read the platform-specific notes and considerations.

Setting Up Natural RPC with Natural Security

If the Natural RPC (client as well as server) is protected by Natural Security, the Natural RPC client or the Natural RPC server respectively must enable International Components for Unicode (ICU) by specifying:

```
CFICU=ON,CP=<valid code page>
```

Setting Up a Natural Client

Unless otherwise noted, this instruction applies to all environments.

To set up a Natural client, perform the following steps:

- **Define the Server Name**
- **Generate an Interface Object**
- **Set the RPC Client-Specific Natural Parameters**

Define the Server Name

Use the *Service Directory Maintenance* function of the `SYSRPC` utility to define the name of the server to be used for each `CALLNAT` statement to be executed remotely.

For details and example screens, refer to *Invoking Service Directory Maintenance* (in the `SYSRPC` utility documentation).

The generated directory subprogram `NATCLTGS` must be made available to the Natural client application. If you have not generated `NATCLTGS` in your client library, move `NATCLTGS` to this library or to one of the steplibs.

Optionally, you can use one of the following server selection techniques:

- **Address a default server**

  See ***Specifying a Default Server Address within a Natural Session***, or profile parameter `DFS`.
- **Use a remote directory server**

  See **Using a Remote Directory Server**, or profile parameter `RDS`.

Generate a Client Interface Object

This step applies only if you do not want to or cannot work with automatic Natural RPC execution (see *Operating a Natural RPC Environment*, ***Working with Automatic Natural RPC Execution***).

For each `CALLNAT` statement to be executed remotely, use the Interface Object Generation function of the `SYSRPC` utility; see ***Creating Interface Objects***.

Note that the generated interface object must be made available to the Natural client environment. If you did not generate the **interface object** in your client library, move the interface object to this library or to one of the steplibs.

Set the RPC Client-Specific Natural Parameters

Set the Natural profile parameters which are relevant to the client-specific handling of remote procedure calls.

Mandatory Parameters:

Parameter	Function
`MAXBUFF`	Maximum buffer size (for automatic RPC execution only)
`RPCSIZE`	Size of buffer used by Natural RPC (for z/OS clients only)

Optional Parameters:

Parameter	Function
`ACIVERS`	This parameter is obsoleted and ignored. The highest ACI version is negotiated by the RPC nucleus and EntireX
`AUTORPC`	Automatic Natural RPC execution
`COMPR`	Set RPC buffer compression See also *Operating a Natural RPC Environment*, ***Using Compression***.
`CPRPC`	Define code page name
`DFS`	Specify RPC client's default server address
`RDS`	Define remote directory server
`RPCSDIR`	Specify name of Natural library in which the Service Directory is located
`TIMEOUT`	Wait time for RPC server response
`TRYALT`	Try alternative server address

The following notes apply to the use of the EntireX Broker.

Notes:

1. The names specified with the `DFS` parameter must identify an active EntireX Broker and must match a server definition in the EntireX Broker Attribute File, see ***Setting Up an EntireX Broker Environment***.
2. The wait time specified with `TIMEOUT` is used to set the `WAIT` field of the EntireX Broker ACI. If `TIMEOUT` is set to zero, `WAIT=YES` is set and the client will wait for the `CLIENT-NONACT` time. If the wait time has elapsed, the remote procedure call is terminated with a corresponding error message. The use of `TIMEOUT` enables you to take advantage of the transport timeout mechanism provided by the EntireX Broker stubs.

Setting Up a Natural Server

A Natural server is a Natural task (**server task**) that can execute Natural subprograms (services). This Natural task is typically an asynchronous or background task (detached process). The EntireX Broker and the client identify it by using a *`nodename`* and a *`servername`*.

To set up a Natural server perform the steps described below:

- Set the RPC Server-Specific Natural Parameters
- Ensure Command Mode Usage in Server Session
- Ensure Unique Adabas ETID Usage
- Start a Natural Server

Set the RPC Server-Specific Natural Parameters

Set the platform-dependent Natural parameters which are relevant to the general and server-specific handling of remote procedure calls for the server Natural.

For z/OS Servers:

1. Create an RPC-specific Natural parameter module.
2. Set the keyword subparameters of profile parameter `RPC` or parameter macro `NTRPC` (see table below) as desired.

For Windows and Linux Servers:

1. Create an RPC-specific Natural parameter file.
2. Set the Natural profile parameters (see table below) as desired.

Mandatory Parameters:

Parameter	Function
MAXBUFF	Maximum buffer size
RPCSIZE	Size of buffer used by Natural RPC (for z/OS servers only)
SERVER	Start Natural session as an RPC server session
SRVNAME	Name of RPC server; see *Note for EntireX Broker* below.
SRVNODE	Name of node; see *Note for EntireX Broker* below.

Optional Parameters:

Parameter	Function
ACIVERS	This parameter is obsoleted and ignored. The highest ACI version is negotiated by the RPC nucleus and EntireX
CPRPC	Define code page name
LOGONRQ	Logon for RPC server request required
NTASKS	Minimum and maximum of the number or server replicas (for z/OS servers only)
SRVCMIT	Time at which a Natural RPC server automatically commits an RPC conversation or a non-conversational RPC request
SRVRTRY	Number of attempts for an RPC server to connect/reconnect (REGISTER) to an EntireX Broker that is not active, and the wait time between two successive attempts.
SRVTERM	Server termination event
SRVUSER	User ID for RPC server registry
SRVWAIT	Wait time of RPC server for client request
TRACE	Define trace level for Natural RPC servers
TRANSP	Server transport protocol (no longer required)

The following notes apply to the use of the EntireX Broker.

Notes:

1. The name specified with SRVNODE must identify an active EntireX Broker and the name specified with SRVNAME must match a server definition in the EntireX Broker Attribute File, see ***Setting Up an EntireX Broker Environment***.
2. The wait time specified with the SRVWAIT parameter is used to set the WAIT field of the EntireX Broker ACI. If SRVWAIT is not specified or set to zero, WAIT=YES is set, and the server will wait for the SERVER-NONACT time. If the wait time has elapsed, a corresponding message is written to the RPC server trace file, and the RPC server continues to wait for the next client request. The use of the SRVWAIT parameter enables you to take advantage of the transport timeout mechanism provided by the EntireX Broker stubs.

3. If you have set the number of attempts for an RPC server to connect or reconnect to an EntireX Broker to a value greater than zero, the RPC server will no longer terminate immediately if the EntireX Broker is shut down (for example, for an IPL or a prolonged maintenance period). In this case, you must explicitly terminate the RPC server before you shut down the EntireX Broker using one of the following ways: Use the **Terminate Service** (`TS`) command of the **Server Command Execution** function of the `SYSRPC` utility, or use the application programming interface `USR2075N` or `USR8208N`, or use the System Management Hub for EntireX.

Ensure Command Mode Usage in Server Session

To ensure that your Natural server session will enter command mode

- disable Natural menu mode by setting the Natural profile parameter `MENU=OFF` (applies to z/OS servers only).

Warning

Do not:

- put a program onto the Natural stack which never terminates.
- use a `STARTUP` program which never terminates.
- disallow `NEXT` mode in Natural Security for your server library.

Ensure Unique Adabas ETID Usage

Ensure that the Adabas `ETID` used by the Natural server session is unique within a certain Adabas nucleus.

Start a Natural Server

To start a Natural server, proceed as described in the section *Starting a Natural RPC Server*.

This server then waits for remote `CALLNAT` requests from a client.

Note for Natural in Batch Mode on z/OS:

For information about servers using the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC`, refer to *Considerations for z/OS Natural RPC Servers with Replicas*.

Setting Up an EntireX Broker Access

To set up an EntireX Broker interface, perform the steps described below:

- Provide Access to the EntireX Broker Stub
- The API Version
- Using TCP/IP as Transport Method

Provide Access to the EntireX Broker Stub

Make the EntireX Broker stub accessible to your Natural environment. This step depends on the platform used.

- **Providing Access to the EntireX Broker Stub on z/OS**
  Link the EntireX Broker stub `NATETB23` to your Natural or specify the profile parameter `RCA=BROKER` to load `NATETB23` dynamically at run-time.

  In the following cases `NATETB23` cannot be used and you have to use a different EntireX Broker stub:

  - If you want to use impersonation in z/OS batch mode, you have to use `BROKER` instead.
  - If you want to use impersonation under CICS, you have to use `CICSETB` instead.

  To load `BKIMBTIA`, `BROKER` or `CICSETB` dynamically at run-time, specify `RCA=BROKER RCALIAS=(BROKER,stubname)`.

  Refer to the EntireX Communicator documentation for further details.

  Note: Please check the prerequisites for the use of `CICSETB` regarding the required PPT entries and the Adabas link routine.

- **Providing Access to the EntireX Broker Stub on Linux**
  The EntireX Broker stub is made available automatically in the course of the EntireX installation.
- **Providing Access to the EntireX Broker Stub on Windows**
  The EntireX Broker stub is made available automatically in the course of the EntireX installation.

The API Version

The possible ACI version is negotiated by the RPC nucleus and EntireX and set to the highest possible value. This version of Natural restricts the ACI version 13, even if EntireX supports a higher ACI version.

Using TCP/IP as Transport Method

If TCP/IP is used as transport method and you use a host name to address the server node, you have the following alternatives:

- Define the server node in the Hosts and Services Directory of your TCP/IP installation.
- Use a Domain Name System (DNS) for domain name resolution.

Setting Up an EntireX Broker Environment

In the EntireX Broker Attribute File, add the following:

1. For each Natural RPC server, a service definition must be specified as follows:

   `CLASS=RPC, SERVICE=CALLNAT, SERVER=`*`servername`*.

2. If you want to use the conversion services, set `CONVERSION=`*`userexit`*. In this case, you must set the profile parameter `CPRPC` accordingly.

If you want to use the reliable RPC, additional settings are required for each Natural RPC server that should support reliable RPC:

- The EntireX Broker attribute `MAX-UOWS` must be set to value greater zero.
- The EntireX Broker attribute `DEFERRED` must be set to `YES` if the client should be able to send reliable RPC messages to an RPC server that is known to the EntireX Broker but has not yet been started.
- The EntireX Broker attribute `STORE` must be set to `BROKER` if recovery of reliable RPC messages after a system failure should be possible. In addition, the EntireX Broker persistent store must be enabled.
- The lifetime of the reliable RPC message itself (EntireX Broker attribute `UWTIME`) and the lifetime of its status (EntireX Broker attribute `UWSTAT-LIFETIME`) must be adapted to your needs.

Set Application Error on RPC Server

You can use the application programming interface `USR4012N` to force the Natural NAT1999 error message to be returned to the RPC client with the text you specify in the API for the case of an error in the application on the Natural RPC server.

For further information, refer to the description in text object `USR4012T` in library `SYSEXT`.

6 Starting a Natural RPC Server

- Preliminaries before Starting a Natural RPC Server 48
- Starting a Natural RPC Server in a z/OS Online Environment (all TP Monitors) 49
- Starting a Natural RPC Server in a z/OS Online Environment (CICS only) 49
- Starting a Natural RPC Server in a z/OS Online Environment (Com-plete only) 51
- Starting a Batch Server in a z/OS Environment 52
- Starting a Natural RPC Server in a Windows Environment 53
- Starting a Natural RPC Server in a Linux Environment 54
- Considerations for z/OS Natural RPC Servers with Replicas 54
- Starting a Natural RPC Server Using the RPC Server Front-End (z/OS Batch Mode only) 55
- Starting a Natural RPC Server Using the RPC Server Front-End (CICS only) 58

This section describes how to start a Natural RPC server on the different platforms.

Preliminaries before Starting a Natural RPC Server

Any kind of Natural session can be used as a Natural RPC server, but typically a Natural server is a Natural session which is started as an asynchronous task or as a background task.

On z/OS:

For the purpose of starting a server, you have the following options:

- Create an RPC server-specific Natural parameter module.

  For a list of the relevant parameters, refer to the section ***Setting Up a Natural RPC Environment***, ***Set the RPC Server-Specific Natural Parameters***.

  This parameter module is linked to your Natural.
- Alternatively, you can also specify the RPC server-specific Natural profile parameters dynamically.

  The RPC server-specific Natural profile parameters may also be specified in a Natural profile created with the `SYSPARM` utility. Natural would then be started with

  ```
  PROFILE=serverprofile
  ```

  where *`serverprofile`* is the name of the Natural profile.

On Windows or Linux:

For the purpose of starting a server, you have the following options:

- Create an RPC server-specific Natural parameter file.

  For a list of the relevant parameters, refer to the section ***Setting Up a Natural RPC Environment***, ***Set the RPC Server-Specific Natural Parameters***. As it is strongly recommended to start a Natural RPC server with the Natural profile parameter `BATCHMODE`, you should additionally specify the Natural profile parameters `CMSYNIN`, `CMOBJIN` and `CMPRINT`.

  Natural will then be started with

  ```
  PARM=serverparm
  ```

  where *`serverparm`* is the name of the parameter file.
- Alternatively, you can specify the RPC server-specific Natural profile parameters dynamically.

How a Natural server is started depends on the environment and is described in the corresponding paragraphs below. For the sake of simplicity, only the mandatory RPC server-specific profile parameters are shown. The optional RPC server-specific profile parameters may be specified analogously.

Starting a Natural RPC Server in a z/OS Online Environment (all TP Monitors)

To start a Natural server in a z/OS online environment

- In your TP monitor environment, enter the following command:

```
<natural> PROFILE=serverprofile
```

Or:

```
<natural> RPC=(SERVER=ON,SRVNAME=servername,SRVNODE=nodename,RPCSIZE=n,MAXBUFF=n)
```

Where `<natural>` is the name with which you start your Natural (transaction code, transaction ID, environment-dependent nucleus name) and `serverprofile` is the name of the Natural profile.

Starting a Natural RPC Server in a z/OS Online Environment (CICS only)

In a CICS environment, you will typically start a Natural RPC server in asynchronous mode. In addition to the options described above for all TP monitors, you have the following options for this purpose:

- You can use the Natural program `STARTSRV` in library `SYSRPC` to start a Natural server in asynchronous mode (see **below**).
- You can start a Natural RPC server in asynchronous mode during startup of CICS (see **below**).

When starting the asynchronous Natural RPC server, it is recommended to specify the Natural profile parameters `TTYPE`, `INTENS`, `SENDER` in addition, using the following settings:

```
TTYPE=ASYL,INTENS=1,SENDER=CSSL
```

This will cause each output to the primary output destination to be written in line mode rather than in 3270 mode. Instead of `CSSL`, you may specify any other CICS output destination.

Starting a Natural RPC Server in Asynchronous Mode with STARTSRV

`STARTSRV` is a sample front-end for `RPCSSRV` that starts the asynchronous Natural session.

By default, the asynchronous Natural is started with the same transaction ID in the same library as the current session.

If Natural Security (NSC) is used, the user ID of the current Natural session is propagated, too. You may adapt the input to your requirements.

Note that some Natural profile parameters are implicitly added by `RPCSSRV`. This applies especially to the keyword subparameter `RPCSIZE` of profile parameter `RPC`. `RPCSIZE` defaults to `MAXBUFF+4`, where `MAXBUFF` is the value entered in the map field **Receiving buffer**. You may overwrite the default value for `RPCSIZE` by entering `RPC=(RPCSIZE=n)` in the map field **Session parameter**.

If you want to execute a Natural program during startup of the Natural session, you may use the map field **User Stack**. The content of **User Stack** is put on the Natural `STACK` and executed before the Natural server is activated.

To show the Natural profile parameters involved, enter `*SHOW*` in the **Transaction ID** field. `STARTSRV` will show you all dynamic profile parameters that will be used by `RPCSSRV` to start the asynchronous Natural session.

Starting a Natural RPC Server Session in Asynchronous Mode during Startup of CICS

To start a Natural RPC server session in asynchronous mode during startup of CICS, proceed as follows:

Use the `PLTPI` to start a program that uses `EXEC CICS START` to start a Natural session with all required RPC specific Natural profile parameters. You may adapt the sample `PLTPI` program `XNCIFRNP` that is provided in the Natural CICS source library.

Please note that the Natural session is started under the CICS default user ID, which, by default, is `CICSUSER`. This user ID is assigned to the Natural system variables `*INIT-USER` and `*USER`. If your Natural session is running under Natural Security, you may therefore have to put a Natural `LOGON` command on the Natural `STACK`.

Note: To avoid errors, do not start multiple RPC server instances with the same transaction ID. It is recommended that you assign a unique transaction ID to each RPC server instance, especially when operating against the same Roll Server.

Starting a Natural RPC Server in a z/OS Online Environment (Com-plete only)

In a Com-plete environment, you will typically start a Natural RPC server in asynchronous mode. In addition to the options described above for all TP monitors, you have the following options for this purpose:

- You can use the Natural program `STARTSRV` in library `SYSRPC` to start a Natural server in asynchronous mode (see **below**).
- You can start a Natural RPC server in asynchronous mode during startup of Com-plete (see **below**).

Starting a Natural RPC Server in Asynchronous Mode with STARTSRV

`STARTSRV` is a sample front-end for `RPCSSRV` that starts the asynchronous Natural session.

By default, the asynchronous Natural is started with the same Natural name in the same library as the current session.

If Natural Security (NSC) is used, the user ID of the current Natural session is propagated, too. You may adapt the input to your requirements.

Note that some Natural profile parameters are implicitly added by `RPCSSRV`. This applies especially to the keyword subparameter `RPCSIZE` of profile parameter `RPC`. `RPCSIZE` defaults to `MAXBUFF+4`, where `MAXBUFF` is the value entered in the map field **Receiving buffer**. You may overwrite the default value for `RPCSIZE` by entering `RPC=(RPCSIZE=`*`n`*`)` in the map field **Session parameter**.

If you want to execute a Natural program during startup of the Natural session, you may use the map field **User Stack**. The content of **User Stack** is put on the Natural `STACK` and executed before the Natural server is activated.

To show the Natural profile parameters involved, enter `*SHOW*` in the **Transaction ID** field. `STARTSRV` will show you all dynamic profile parameters that will be used by `RPCSSRV` to start the asynchronous Natural session.

Starting a Natural RPC Server Session in Asynchronous Mode during Start-up of Com-plete

To start a Natural RPC server session in asynchronous mode during startup of Com-plete, proceed as follows:

Use the startup option (sysparms) `STARTUPPGM` to start a Natural session with all required RPC-specific Natural profile parameters. You may either specify the required RPC-specific parameters individually or use a Natural profile to specify them as follows:

```
STARTUPPGM='<natural> PROFILE=<serverprofile>'
```

Please note that the Natural session is started under the user ID under which Com-plete is started. This user ID is assigned to the Natural system variables `*INIT-USER` and `*USER`. If your Natural session is running under Natural Security, you may therefore have to put a Natural `LOGON` command on the Natural `STACK`.

Starting a Batch Server in a z/OS Environment

A batch server is a standard Natural batch session that is started with the RPC parameters described in the section ***Setting Up a Natural RPC Environment***, ***Set the RPC Server-Specific Natural Parameters***.

The following topics are covered below:

- Starting a Batch Server under z/OS

Note: For a sample JCL using the trace facility, refer to ***Operating a Natural RPC Environment***, ***Using the Server Trace Facility***.

Starting a Batch Server under z/OS

Sample JCL for z/OS

```
//NATRPC   JOB  CLASS=K,MSGCLASS=X
//         EXEC PGM=NATOS,REGION=8M                <== Note 1
//STEPLIB  DD   DISP=SHR,DSN=SAG.NAT.LOAD
//         DD   DISP=SHR,DSN=SAG.EXX.LOAD
//         DD   DISP=SHR,DSN=SAG.ADA.LOAD          <== Note 2
//         DD   DISP=SHR,DSN=DB2_load_library      <== Note 3
//CMPRMIN  DD   *
IM=D,MADIO=0,MT=0,OBJIN=R,AUTO=OFF,MAXCL=0,ID=',',INTENS=1,
RPC=(SERVER=ON,SRVNAME=servername,SRVNODE=nodename)
RPC=(RPCSIZE=m,MAXBUFF=n),
STACK=(LOGON serverlibrary,userID,password)
/*
//CEEOPTS  DD *                                    <== Note 4
POSIX(ON)
/*
//SYSUDUMP DD SYSOUT=X
//CMPRINT  DD SYSOUT=X
/*
```

Notes:

1. If the RPC server is running with subtasks, the `EXEC PARM` statement will pass Natural parameters only to the main task. Instead, use the `CMPRMIN` statement to pass parameters to the main task and the subtasks simultaneously.
2. Applies only if the Adabas link routine `ADAUSER` or the Natural profile parameter `ADANAME` is used.
3. Applies to Db2 users only.
4. Applies only if SSL is used.

Sample JCL for a Started Task

A sample JCL for a started task is provided in the Natural for z/OS installation documentation; see *Installing Natural on z/OS*.

Running a Batch Server with Replicas

You can also run a batch server with replicas by setting the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC` to a value greater than 1.

Replicas are attached to a Natural main task as additional server tasks. They enable you to start several identical servers in the same region.

Starting a Natural RPC Server in a Windows Environment

To start a Natural RPC server under Windows

1 Create a shortcut for Natural.

2 Enter the shortcut properties.

3 In the **Target** text box, edit the Natural path and append:

```
"<Path>\natural.exe" batchmode parm=serverparm
```

where *`serverparm`* is the name of the parameter file.

Or:

```
"<Path>\natural.exe" batchmode
server=on ↩
srvname=servername srvnode=nodename maxbuff=n cmsynin=cmsynin cmobjin=cmobjin cmprint=cmprint
```

Starting a Natural RPC Server in a Linux Environment

To start a Natural RPC server under Linux

- Enter the following command:

```
natural batchmode parm=serverparm &
```

where *serverparm* is the name of the parameter file.

Or:

```
natural batchmode
server=on ↩
srvname=servername srvnode=nodename maxbuff=n cmsynin=cmsynin cmobjin=cmobjin cmprint=cmprint
 &
```

Considerations for z/OS Natural RPC Servers with Replicas

- **Natural RPC Batch Server with NTASKS >1**
- **Running a Batch Server with Replicas**

Natural RPC Batch Server with NTASKS >1

The main task and all replicas run in the same z/OS region.

1. Use the reentrant batch link routine ADALNKR instead of ADALNK.

   If you want to use ADAUSER, you must not link ADAUSER with your front-end, because ADAUSER is non-reentrant (see Item 5). Instead, use the Natural profile parameter ADANAME and set ADANAME=ADAUSER. This will cause Natural to load ADAUSER dynamically at runtime.
2. In the Natural parameter module:
   - Set the keyword subparameter NTASKS=*n* of profile parameter RPC or parameter macro NTRPC, where *n* is the number of parallel servers (< 100) to be started, including the main task.
   - Use the Natural profile parameter ETID to specify the Adabas user identification as a blank character. This is necessary to prevent a NAT3048 error (ETID not unique in Adabas nucleus) when the subtask is started.
3. When using dynamic Natural profile parameters:

Use the dynamic parameter dataset `CMPRMIN` to pass the dynamic Natural profile parameters to Natural. Do *not* use the `PARM` card or the primary command input dataset `CMSYNIN`.

4. When using a local buffer pool (z/OS only):

   Each subtask allocates its own local buffer pool unless you specify a shared local buffer pool. See subparameter `LBPNAME` of profile parameter `OSP` or parameter macro `NTOSP` (in the *Parameter Reference* documentation).

5. In the Natural front-end link job (z/OS only):

   Link the front-end reentrant by using the `RENT` option of the linkage editor.

   If the front-end were not linked with the `RENT` option, only the main task would start the communication with the EntireX Broker. All subtasks would be set to a `WAIT` status by z/OS, until the main task would have been terminated. If you would terminate the RPC server lateron, the address space would hang and would have to be cancelled.

6. Make sure that any other modules that are additionally linked to the Natural nucleus are reentrant. Any dynamically loaded programs must also be reentrant.

   Note for z/OS: If you cannot make a module reentrant, link the module as non-reusable; this means, you should *not* specify the link option `RENT` or `REUS`. This is to ensure that each subtask will get its own copy.

Running a Batch Server with Replicas

For a sample JCL, see ***Using the Server Trace Facility***.

Starting a Natural RPC Server Using the RPC Server Front-End (z/OS Batch Mode only)

In z/OS batch mode, a Natural RPC server may alternatively be started using the RPC server front-end. This approach is required with **impersonation** and is optional in other cases.

If you use the RPC server front-end without impersonation, you are recommended to set the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC` to a value greater than 1. Otherwise, there will be no benefit. The ***Considerations for z/OS Natural RPC Servers with Replicas*** apply also when you are using the RPC server front-end.

The RPC server front-end uses the Natural Server functionality; see *Natural as a Server under z/OS* (in the Natural *Operations* documentation). It is characterized by the following features:

- The Natural RPC server front-end starts a number of Natural RPC server sessions as specified by the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC`.
- All Natural RPC server sessions run with the same Natural profile parameter settings.

The Natural profile parameter settings are taken from the Natural parameter module and may be overwritten by dynamically specified profile parameters that are passed with the transaction ID. The dynamically specified profile parameters must follow the startup parameter.

- If all Natural RPC server sessions are currently in use by clients (executing a client request or waiting for the next request within a conversation) and if the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC` is set to a value greater than one, auxiliary Natural RPC server sessions are started. These Natural RPC server sessions are automatically terminated on the first EntireX Broker timeout, provided that there is at least one other Natural RPC server session not in use by a client. If all other Natural RPC server sessions are being used by clients, the auxiliary RPC server session will stay up until the next EntireX Broker timeout. This makes sure that there is always a Natural RPC server available to process a new client request.
- The Natural RPC server sessions are executed in a thread environment that is similar to Natural sessions executing in a TP monitor system.
- **With impersonation:**

  At the end of a non-conversational `CALLNAT` and at the end of a conversation, all database sessions and all work files are closed. This ensures that the next client request will open the database and the work files with its own user ID.
- **Without impersonation:**

  After the first EntireX Broker timeout, all database sessions and all work files are closed. This ensures that no resources are blocked during wait times.

Startup Parameters:

The required startup parameters are passed in the `PARM=` parameter of the `EXEC` statement in the JCL. These parameters are:

- The name of the Natural z/OS batch nucleus;
- The size of a storage thread;
- The optional keyword `UCTRAN`.

  `UCTRAN` indicates that all messages of the RPC Server front-end are converted into upper case.

These parameters must be separated by commas and must be entered without leading or trailing blanks:

PARM='*Natural-z/OS-batch-nucleus,size-of-thread*[,,*UCTRAN*]'

Note: For compatibility reasons `UCTRAN` must be preceded by two consecutive commas.

See also the ***Sample JCL*** below.

Execution Notes:

- The Natural z/OS batch nucleus is dynamically loaded.

  The load library containing the z/OS batch nucleus must be available in the steplib concatenation.
- The EntireX Broker stub `NATETB23` must not be used.

  The EntireX Broker is accessed outside the Natural context. Therefore, you must use the EntireX Broker stub `BROKER`, see ***Providing Access to the EntireX Broker Stub on z/OS***.

With impersonation only:

When the **impersonation** feature is used, the Natural RPC server calls Natural Autorized Services Manager (NATASMvv) to do the impersonation. No APF authorization is required for the Natural RPC server itself.

With Natural Security only:

If the Natural RPC server front-end is started with profile parameter `AUTO=OFF`, you must provide a Natural `LOGON` command with library ID, user ID and password on the Natural stack: `STACK=(LOGON library-id;user-id;password)`.

Sample JCL:

```
//NATRPC   JOB  CLASS=K,MSGCLASS=X
//         EXEC PGM=RPC-FRONT,REGION=8M           <== Note 1
//         PARM='Natural-z/OS-interface-module,1000'
//STEPLIB  DD   DISP=SHR,DSN=SAG.NAT.LOAD
//         DD   DISP=SHR,DSN=SAG.EXX.LOAD
//         DD   DISP=SHR,DSN=SAG.ADA.LOAD         <== Note 2
//         DD   DISP=SHR,DSN=DB2_load_library     <== Note 3
//CMPRMIN  DD   *
IM=D,MADIO=0,MT=0,OBJIN=R,AUTO=OFF,MAXCL=0,ID=',',INTENS=1,
PRINT=((10),AM=STD)
RPC=(SERVER=ON,SRVNAME=servername,SRVNODE=nodename,NTASKS=3)
RPC=(RPCSIZE=m,MAXBUFF=n,TRACE=2),
RCA=BROKER,RCALIAS=(BROKER,BROKER)
STACK=(LOGON serverlibrary,userID,password)
/*
//CEEOPTS  DD *                                   <== Note 4
POSIX(ON)
*
//SYSUDUMP DD SYSOUT=X
```

```
//CMPRT10  DD SYSOUT=X
//CMPRT101 DD SYSOUT=X
//CMPRT102 DD SYSOUT=X
//CMPRT199 DD SYSOUT=X
//CMPRINT  DD SYSOUT=X
//CMPRINT1 DD SYSOUT=X
//CMPRINT2 DD SYSOUT=X
//CMPRIN99 DD SYSOUT=X
/*
```

Notes:

1. If the RPC server is running with subtasks, the EXEC PARM statement will pass Natural parameters only to the main task. Instead, use the CMPRMIN statement to pass parameters to the main task and the subtasks simultaneously.
2. Applies only if the Adabas link routine `ADAUSER` or the Natural profile parameter `ADANAME` is used.
3. Applies to DB2 users only.
4. Applies only if SSL is used.

Starting a Natural RPC Server Using the RPC Server Front-End (CICS only)

In CICS, a Natural RPC server may alternatively be started using the RPC server front-end. This approach is required when impersonation is used and is optional in other cases.

If you use the RPC server front-end without impersonation, you are recommended to set the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC` to a value greater than 1. Otherwise, there will be no benefit.

The RPC server front-end uses the Natural Server functionality. It is characterized by the following features:

- The Natural RPC server front-end is started via the transaction ID defined in the *Customize CICS* step for the Natural RPC server front-end;

  see *Installing the Natural CICS Interface on z/OS*.

  The transaction ID may either be entered at a terminal or you may use the Natural program `STARTSFE` in library `SYSRPC` to start the Natural RPC server front-end in asynchronous mode.

  The Natural RPC server front-end requires the name of the Natural CICS Interface nucleus as startup parameter. This startup parameter is passed with the transaction ID.
- The Natural RPC server front-end starts a number of Natural RPC server sessions as specified by the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC`.

- All Natural RPC server sessions run with the same Natural profile parameter settings.

  The Natural profile parameter settings are taken from the Natural parameter module and may be overwritten by dynamically specified profile parameters that are passed with the transaction ID. The dynamically specified profile parameters must follow the startup parameter.
- If all Natural RPC server sessions are currently in use by clients (executing a client request or waiting for the next request within a conversation) and if the keyword subparameter `NTASKS` of profile parameter `RPC` or parameter macro `NTRPC` is set to a value greater than one, auxiliary Natural RPC server sessions are started. These Natural RPC server sessions are automatically terminated on the first EntireX Broker timeout, provided that there is at least one other Natural RPC server session not in use by a client. If all other Natural RPC server sessions are being used by clients, the auxiliary RPC server session will stay up until the next EntireX Broker timeout. This will ensure that there is always a Natural RPC server available to process a new client request.
- The Natural RPC server sessions are executed in a thread environment that is similar to Natural sessions executing in a TP monitor system.
- All inactive Natural RPC server sessions (sessions that wait for a client request) are rolled out using the Natural Roll Facilities under CICS which are defined for the Natural CICS Interface nucleus in use.
- **With impersonation:**

  At the start of a non-conversational `CALLNAT` and at the start of a conversation, a new CICS worker task is started under the user ID of the client by using the `USERID()` option of the `EXEC CICS START TRANSID()` command. The client request is executed by Natural in this worker task. While the client request is executed, the Natural RPC server session waits for the worker task to finish.

  At the end of a non-conversational `CALLNAT` and at the end of a conversation, the worker task is terminated and all databases are closed and all CICS resources are freed. This ensures that the next client request will open the database and access the CICS resources with its own user ID.
- **Without impersonation:**

  The client request is executed by the Natural RPC server session itself.

  After the first EntireX Broker timeout, all databases are closed and all CICS resources are freed. This will ensure that no resources are blocked during wait times.

Startup Parameters:

The required startup parameters are passed with the transaction ID. These parameters are:

- The name of the Natural CICS Interface nucleus `<ncistart>`.
- The optional keyword `UCTRAN`.

UCTRAN indicates that all messages of the RPC Server front-end are converted into upper case.

- An optional Natural profile parameter string.

Sample Start at a Terminal:

```
<natural> <ncistart>[,UCTRAN]
RPC=(SERVER=ON,SRVNAME=servername,SRVNODE=nodename,RPCSIZE=n, MAXBUFF=n)
RCA=BROKER,RCALIAS=(BROKER,CICSETB)
```

Where <natural> is the transaction ID with which you start your Natural RPC server front-end and <ncistart> is the name of your Natural CICS Interface nucleus.

Execution Notes:

- If the Natural Roll Server is required (NCMDIR parameter ROLLSRV is set to YES), you must start a Natural Roll Server for the used subsystem-id (as defined by Natural profile parameter SUBSID) before the Natural RPC server front-end is started.
- The transaction ID of the RPC server front-end is used to identify the RPC server environment.

  Do not to start more than one Natural RPC server front-end with the same transaction ID.
- The executable NCI module is dynamically loaded.

  The load library containing the executable NCI module must be available in the DFHRPL concatenation.
- The EntireX Broker stub NATETB23 must not be used.

  The EntireX Broker is accessed outside the Natural context. Therefore, you must use the EntireX Broker stub CICSETB, see ***Providing Access to the EntireX Broker Stub on z/OS***.

With impersonation only:

When the impersonation feature is used, the RPC server front-end starts worker tasks. Ensure that the setting of the CICS system initialization parameter MXT of your CICS installation is high enough.

With Natural Security only:

If the Natural RPC server front-end is started with profile parameter AUTO=OFF, you must provide a Natural LOGON command with library ID, user ID and password on the Natural stack: STACK=(LOGON *library-id;user-id;password*).

7 Terminating a Natural RPC Server

- Using SYSRPC 62
- Using EntireX System Management Hub 62
- Using Application Programming Interface USR2073N 63
- Using Application Programming Interface USR2075N 64
- Using Application Programming Interface USR8208N 65
- Using Application Programming Interface USR8213N 66
- Using Application Programming Interface USR8220N 66
- Using Application Programming Interface USR9175N 67
- Using Application Programming Interface USR9176N 67
- Using Application Programming Interface USR9177N 68
- Server Termination When Using an Attach Manager 68
- User Exit NATRPC99 68

This section describes how to terminate a Natural RPC server. This can be done either by terminating a single RPC server replica or the EntireX Broker service supporting the RPC connection. Several methods exist.

Using SYSRPC

This method can either be applied to a single RPC server replica or the EntireX Broker service:

- Using SYSRPC to Terminate a Single RPC Server Replica
- Using SYSRPC to Terminate a Single RPC Replica by Sequence Number
- Using SYSRPC to Terminate an EntireX Broker Service

Using SYSRPC to Terminate a Single RPC Server Replica

Use the `TE` (Terminate Server) command of the `SYSRPC` utility as described in *Terminating an RPC Server* in the *SYSRPC Utility* documentation.

Note: A Natural RPC server can only be terminated if the server is currently neither executing a remote `CALLNAT` nor waiting for the next `CALLNAT` request in a conversation.

Using SYSRPC to Terminate a Single RPC Replica by Sequence Number

You can use the `LN` (List Server with Sequence Number) command to select the severs to be terminated with the `TE` (Terminate Server) command from a detailed list of replicas that contains the sequence number assigned to a server. For more information, see *Terminating an RPC Server* in the *SYSRPC Utility* documentation.

Using SYSRPC to Terminate an EntireX Broker Service

Use the `TS` (Terminate EntireX Broker Service) command of the `SYSRPC` utility as described in *Terminating an RPC Server* in the *SYSRPC Utility* documentation.

Using EntireX System Management Hub

Use the **Deregister** button in the Server subtree of the EntireX System Management Hub. You can also use the **Deregister** button in the Service subtree of the EntireX System Management Hub.

Using Application Programming Interface USR2073N

The Application Programming Interface (API) USR2073N enables you to ping or terminate a Natural RPC server.

The interface sends a ping or terminate command to an RPC server specified by node name and server name. The returned message contains the following information:

- the version of the running server (PING) or
- the acknowledgment of termination (TERMINATE) or
- an error message.

To make use of USR2073N

1. Copy the subprogram USR2073N from library SYSEXT to the library SYSTEM or to the steplib library or to any application in the server environment.
2. Using a DEFINE DATA statement in structured mode or a RESET statement in reporting mode, specify the following parameters:

Parameter	Format	I/O	Description	
FUNCTION	A10	I	PING	Asks the RPC server for its version.
			TERMINATE	Initiates the finishing of the Natural RPC server task.
SRVNODE	A192	I	Node name. See the profile parameter SRVNODE.	
SRVNAME	A32	I	Server name. See the profile parameter SRVNAME .	
LOGON	A1	I	The logon flag Y (yes) indicates that logon data must be transmitted to the Natural RPC server.	
USER-ID PASSWORD	A8	I	If LOGON=Y, then user ID and password are the logon data for Natural Security. If the client does not run under Natural Security (NSC) and the logon flag is set, the user ID and password have to be specified unless the data have been entered via Application Programming Interface USR1071N. See *Using Security*, ***Using Natural RPC with Natural Security***.	
MSG	A79	O	Message returned.	
RC	I2	O	Response code; possible values are:	
			0	MSG contains a normal message from RPC server or EntireX Broker.

Parameter	Format	I/O	Description	
			1	MSG contains an error message from RPC server or EntireX Broker.
			2	MSG contains an error message from the interface.
			3	Natural Security error.

3 Before invoking the API, fill the input variables listed above.

Using Application Programming Interface USR2075N

The Application Programming Interface (API) USR2075N enables you to terminate an EntireX Broker Service from within your application.

With the command TERMINATE-SERVICE the interface uses the command and information service of EntireX to fulfill the task. First of all, it sends a logon request to the EntireX Broker. Then it gets a list of all servers specified by server class, server name, and service type. Finally, it sends a shutdown request to each server. A message indicates how many servers were terminated.

With the command PING the identification string of a server is returned.

To make use of USR2075N

1 Copy the subprogram USR2075N from library SYSEXT to the library SYSTEM or to the steplib library or to any application in the server environment.

2 Using a DEFINE DATA statement in structured mode or a RESET statement in reporting mode, specify the following parameters:

Parameter	Format	I/O	Description
FUNCTION	A17	I	Specify TERMINATE-SERVICE. This sends a shutdown request to each server specified with the parameters SRVCLASS, SRVNAME, and SERVICE (service type), see below. or: Specify PING This returns the identification string of a server.
SRVNODE	A192	I	Specify the node name. See the profile parameter SRVNODE.
SRVCLASS	A32	I	Specify the server class. For Natural RPC servers, this is RPC.

Parameter	Format	I/O	Description	
SRVNAME	A32	I	Server name. See the profile parameter SRVNAME	
SERVICE	A32	I	Specify the type of service. For Natural RPC servers, this is CALLNAT.	
IMMEDIATE	L		TRUE	Immediately terminate all conversations for the specified server.
			FALSE	Existing conversations are allowed to end normally; no new conversations are accepted.
USER-ID	A32	I	Specify the user ID to logon to the EntireX Broker.	
PASSWORD	A32	I	Specify the password to the EntireX Broker if EntireX Broker security is active. If EntireX Broker security is active, you must specify a password. If EntireX Broker security uses the trusted user ID mechanism (on z/OS only), you can use the reserved password *TRUSTED instead of a user password.	
MSG	A (dynamic)	O	Message returned.	
RC	I2	O	Response code; possible values:	
			0	MSG contains a normal message from the EntireX Broker.
			1	MSG contains an error message from the EntireX Broker.
			3	MSG contains an error message from Natural Security on the client side.

3 Before invoking the API, fill the input variables listed above.

Using Application Programming Interface USR8208N

The Application Programming Interface (API) USR8208N is an enhanced version of API USR2075N with the following new features:

- With the command TERMINATE-SERVICE the interface uses the Command and Information Service of EntireX to fulfill the task. The SHUTDOWN-SERVICE command is sent to the EntireX Broker if its version is 5 or above. It returns version information on the EntireX Broker and the Command and Information Service itself. Otherwise, the same method is used as in USR2075N.
- With the command INFO-SERVICE additional information on the specified service(s) is returned from the EntireX Broker such as SERVER-CLASS, SERVER-NAME, SERVICE, TRANS, CONV-NONACT etc. For a detailed description, see *webMethods EntireX V9.0.1 documentation > Components and Features*

of EntireX > ACI Programming > Broker CIS Data Structures > Information Reply Structures > SERVICE-OBJECT (Struct INFO_SV).

To make use of `USR8208N`

- See the similar instruction **To make use of** `USR2075N`.

Using Application Programming Interface USR8213N

The Application Programming Interface (API) `USR8213N` shuts down a server on an EntireX Broker node.

This interface firstly retrieves a list of RPC server replica registered on a specified EntireX Broker. For each server, the list indicates the start date/time, host name, application name and IP address (if available). It also indicates the sequence number assigned to each server during registration. The interface then sends a shutdown request with the sequence number to the EntireX Broker.

To make use of `USR8213N`

- Copy the subprogram `USR8213N` from the library SYSEXT to the library SYSTEM or to the steplib library, or to any application in the Natural RPC server environment and proceed as described on the text object `USR8213T` in the library `SYSEXT`.

Using Application Programming Interface USR8220N

The Application Programming Interface (API) `USR8220N` triggers the server-side termination of the Natural RPC server that is running this API. After calling `USR8220N`, you can send a confirmation message to the RPC client. This API is useful to explicitly terminate the Natural RPC server when severe errors occur.

To make use of `USR8220N`

1. Copy the subprogram `USR8220N` from the library SYSEXT to the library SYSTEM or to the steplib library, or to any application in the Natural RPC server environment.
2. Create a Natural RPC server subprogram that calls `USR8220N` and that sends an optional confirmation message to the RPC client program.

Using Application Programming Interface USR9175N

The Application Programming Interface (API) USR9175N is an enhanced version of API USR2073N with an extended PASSWORD parameter (format A128) and the additional values for the FUNCTION parameter:

Parameter	Format	I/O	Description	
FUNCTION	A10	I	PING	Asks the RPC server for its version.
			TERMINATE	Initiates the finishing of the Natural RPC server task.
			TRACEINFO	Retrieves the state of the Natural RPC server trace.
			TRACE0	Disables the Natural RPC server trace.
			TRACEn	Enables the Natural RPC server trace with level n, where n=1-9.
			TRACEnE	Enables the Natural RPC server trace with level n, where n=1-9, and sets the flag to trace in the event of an error only.

For further information about the possible trace settings, see TRACE - Define Trace Level for Natural RPC Servers.

To make use of USR9175N

1 See the similar instruction **To make use of** USR2073N.

2 When using trace, the print file for the trace must be defined with the start of the Natural RPC server.

Using Application Programming Interface USR9176N

The Application Programming Interface (API) USR9176N is an enhanced version of API USR8208N with an extended PASSWORD parameter (format A128).

To make use of USR9176N

- See the similar instruction **To make use of** USR8208N.

Using Application Programming Interface USR9177N

The Application Programming Interface (API) USR9177N is an enhanced version of API USR8213N with the an extended PASSWORD parameter (format A128).

To make use of USR9177N

- See the similar instruction **To make use of USR8213N**.

Server Termination When Using an Attach Manager

The profile parameter SRVTERM influences the termination behavior of a Natural RPC server. By default, a server is never terminated (SRVTERM=NEVER) unless one of the termination methods described before is applied.

If you use an Attach Manager to dynamically start Natural RPC servers on request, you should set SRVTERM to TIMEOUT. With this parameter setting, a Natural RPC server is automatically terminated if the wait time for the next client request outside of an RPC conversation is exceeded.

User Exit NATRPC99

This exit is called after the Natural RPC server has deregistered and logged off from the server node.

- If no NATRPC99 program is found, the server terminates immediately as usual.
- If the program NATRPC99 is found, the server continues to run as a normal Natural session.

NATRPC99 is called with a FETCH statement without any parameters, that is, no data is put on the Natural stack before NATRPC99 has been called.

You may add any coding to NATRPC99, including transfer control statements (FETCH, CALLNAT, PERFORM) and statements that terminate the program (STOP, ESCAPE, TERMINATE).

If NATRPC99 is terminated with a RETURN or STOP statement, Natural returns to the NEXT prompt. If the NEXT prompt is not supported in the environment used (profile parameter CM=OFF, asynchronous Natural session, etc.) the session terminates. Otherwise, the session tries to read the next command from the primary input file/dataset for Natural commands and INPUT data.

Important Notes:

1. NATRPC99 must be a Natural program.
2. NATRPC99 must be located in the library SYSTEM on system file FUSER. The steplib concatenation of the library to which the server currently is logged on is not evaluated to find NATRPC99.
3. Natural objects that are called by NATRPC99 (FETCH, CALLNAT, PERFORM) must be located either in the library to which the server is logged on or in one of its steplibs (including SYSTEM on system file FUSER).
4. NATRPC99 is called for all kinds of normal Natural RPC server terminations. It is not called in case of abends.

8 Operating a Natural RPC Environment

- Specifying RPC Server Addresses 72
- Interface Objects and Automatic RPC Execution 75
- Getting or Setting RPC Profile Parameters during a Natural Session 77
- Executing Server Commands 77
- Sending Command to RPC Server 78
- Logon to a Server Library 79
- Using the Logon Option 80
- Using Compression 81
- Using Secure Socket Layer 82
- Monitoring the Status of an RPC Session 83
- Retrieving Runtime Settings of a Server 90
- Setting or Getting Parameters for EntireX 91
- Maintain Message ID and Correlation ID of EntireX 93
- Handling Errors 94
- User Exits before and after Service Execution 95

This section mainly describes the tasks required to operate a Natural RPC environment.

Some of these tasks are performed with the SYSRPC utility. For information about the functions the `SYSRPC` utility provides, refer to the Natural *SYSRPC Utility* documentation.

Specifying RPC Server Addresses

To each remote `CALLNAT` request, a server must be assigned (identified by *`servername`* and *`nodename`*) on which the `CALLNAT` is to be executed. Therefore, all subprograms to be accessed remotely must be defined

- in a local service directory on the client side,
- or in a remote directory accessed via a remote directory server,
- or by way of default server addressing with the profile parameter `DFS`,
- or within the client application itself by way of default server addressing.

In addition to the methods mentioned above, you can specify alternative servers.

- Using Local Directory Entries
- Using Remote Directory Entries
- Specifying a Default Server Address at Natural Startup
- Specifying a Default Server Address within a Natural Session
- Using an Alternative Server

Using Local Directory Entries

All data of a client's local service directory is stored in the subprogram `NATCLTGS`. At execution time, this subprogram is used to retrieve the target server. As a consequence, `NATCLTGS` must be available in the client application or in one of the Natural steplibs defined for the application.

If `NATCLTGS` has not been generated into a steplib or resides on another machine, use the appropriate Natural utility (`SYSMAIN` or the Natural Object Handler) to move `NATCLTGS` into one of the steplibs defined for the application.

If you are using a `NATCLTGS` for joint usage, you must make it available to all client environments, for example by copying it to the library `SYSTEM`, or, if an individual copy is used for a client, it must be maintained for this client using the Service Directory Maintenance function of the `SYSRPC` utility.

To define and edit RPC service entries, see the section *Service Directory Maintenance* in the *SYSRPC Utility* documentation.

Using Remote Directory Entries

A remote directory contains service entries that can be made available to several Natural clients. The Natural clients can retrieve these service entries from remote directory servers. For information on the purpose and on the installation of remote directory servers; see ***Using a Remote Directory Server***.

Specifying a Default Server Address at Natural Startup

Instead of addressing a server by using a local or remote service directory, you can preset a default server with the profile parameter `DFS`, as described in the Natural *Parameter Reference* documentation. This server address is used if the subprogram can be found in neither the local nor the remote service directory.

The `DFS` setting determines the default server for the whole session or until it is overwritten dynamically.

If no `DFS` setting exists and the server address of a given remote procedure call could not be found in the service directory, a Natural error message is returned.

A default server address defined within a client application remains active even if you log on to another library or if a Natural error occurs.

Specifying a Default Server Address within a Natural Session

The client application itself may dynamically specify a default server address at runtime. For this purpose, Natural provides the application programming interface `USR2007N`. This interface enables you to determine a default server address that is to be used each time a remote program cannot be addressed via the service directory.

To make use of `USR2007N`

1 Copy the subprogram `USR2007N` from the library `SYSEXT` to the library `SYSTEM` or to the steplib library, or to any application in the server environment.

2 Using the `DEFINE DATA` statement in structured mode or the `RESET` statement in reporting mode, specify the following parameters:

Parameter	Format	Description
function	A1	Function; possible values are:

Parameter	Format	Description	
		P (Put)	Determines that the server address (composed of the parameters *nodename* and *servername*, see below) is the default address for all subsequent remote procedure calls not defined in the service directory. To remove a default server address, specify a blank for *nodename* and *servername*.
		G (Get)	Retrieves the current default server address as set by the function P.
nodename	A192	Specifies/returns the name of the server node to be addressed. **Note:** For compatibility reasons, *nodename* is defined with the option BY VALUE or BY VALUE RESULT (see the section parameter-data-definition in the description of the DEFINE DATA statement) to support existing callers which pass an A8 or A32 field for *nodename*. The sample USR2007P provided in the library SYSEXT supports up to 32 characters.	
servername	A32	Specifies/returns the server name to be addressed. **Note:** For compatibility reasons, *servername* is defined with the option BY VALUE or BY VALUE RESULT (see the section parameter-data-definition in the description of the DEFINE DATA statement) to support existing callers which pass an A8 field for *servername*.	
logon	A1	Specifies/returns the Logon option, see ***Using the Logon Option***.	
protocol	A1	Specifies/returns the transport protocol. Valid value: B (=EntireX Broker).	
noservdir	A1	Specifies/returns the service directory option, see profile parameter DFS.	
		Y	Service directory must not be present
		N	Service directory must be present

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR2007N' function nodename servername logon protocol [noservdir]
```

Note: The Natural subprogram NATCLTPS in the library SYSRPC is only maintained for compatibility reasons.

Using an Alternative Server

To avoid connection failures, you may want to define several alternative servers for a remote `CALLNAT`. If you specify such alternative servers, Natural proceeds as follows:

- The client makes a first attempt to establish the connection.
- If this attempt fails, instead of providing an error message, a second attempt is made, however, this time not on the same server. Instead, the service directory is searched again starting at the current entry to find out whether or not another server is available which offers the desired service.
- If a second entry is found, Natural tries to establish the connection to this server. If the remote procedure call is performed successfully, the client application keeps on running. The user does not notice whether the connection to the first server or to the alternative server produced the result.
- If no further entry is found or if the connections to alternative servers fail, Natural issues a corresponding error message.

To enable the use of an alternative server

1. Define more than one server in the service directory for the same service.
2. Set the profile parameter `TRYALT` to `ON` to give permission to use an alternative server.

This parameter can also be set dynamically for the current session with the Parameter Maintenance function (described in the *SYSRPC Utility* documentation).

Interface Objects and Automatic RPC Execution

Interface objects are no longer required if automatic Natural RPC execution is used, as described in ***Working with Automatic Natural RPC Execution*** below.

However, generating interface objects provides the advantage of controlling the `CALLNAT`(s) executed remotely and facilitates error diagnoses. Should a remote call fail due to an incorrect `CALLNAT` name, the Natural error message issued then helps to immediately identify the problem cause. Without an interface object, for an incorrect `CALLNAT` you may receive follow-up errors returned from the transport layer or the Natural server.

If you want to use a remote `CALLNAT` statement to execute a subprogram on an EntireX RPC server, we strongly recommend that you set `AUTORPC=OFF` and use an **interface object**. If the IDL (Interface Definition Language) of the subprogram you want to call on an EntireX RPC server contains a group structure, you must define the same group structure during the interface object generation on the Interface Object Generation screen or generate the interface object from the EntireX IDL file (Windows only) or use the EntireX Workbench (Linux and Windows only).

- Creating Interface Objects
- Working with Automatic Natural RPC Execution

Creating Interface Objects

With the **Interface Object Generation** function of the `SYSRPC` utility, you can generate the Natural interface objects used to connect the client's calling program to a subprogram on a server. The interface object consists of a parameter data area (PDA) and the server call logic; see *Interface Object Generation - General Considerations* in the *SYSRPC Utility* documentation.

The PDA contains the same parameters as used in the `CALLNAT` statement of the calling program and must be defined on the **Interface Object Generation** screen of the **Interface Object Generation** function. If a compiled Natural subprogram with the same name already exists, the PDA used by this subprogram is used to preset the screen. The server call logic is generated automatically by the **Interface Object Generation** function after the PDA has been defined.

At execution time, the Natural application program containing the `CALLNAT` statement and the interface object must exist on the client side. The Natural application subprogram must exist on the server side. Both the interface object subprogram and the server subprogram must have the same name.

Working with Automatic Natural RPC Execution

This section applies only if you do not want to call an EntireX RPC server.

You are not required to generate Natural RPC interface objects, but you can work with automatic Natural RPC execution (that is, without using Natural interface objects). To work with automatic Natural RPC execution, set the profile parameter `AUTORPC` as follows:

```
AUTORPC=ON
```

In that case, you can omit the generation of the client interface object during your preparations for RPC usage. When the automatic Natural RPC execution is enabled (`AUTORPC=ON`), Natural behaves as follows:

- if a subprogram cannot be found locally, Natural tries to execute it remotely (an interface object is not needed),
- the parameter data area will then be generated dynamically during runtime.

As interface objects only exist for client programs, this feature has no effect on the `CALLNAT` program on the server.

If profile parameter `AUTORPC` is set to `ON`, and a Natural interface object exists, it will still be used.

Getting or Setting RPC Profile Parameters during a Natural Session

- Modifying RPC Profile Parameters during a Natural Session
- Get or Set RPC TIMEOUT Value Using API USR2076N

Modifying RPC Profile Parameters during a Natural Session

With the Parameter Maintenance function of the SYSRPC utility, you can dynamically modify some of the RPC profile parameters set in the Natural profile parameter module for the current session.

Caution: These modifications are retained as long as the user session is active; they are lost when the session is terminated.

Get or Set RPC TIMEOUT Value Using API USR2076N

You can use the Application Programming Interface `USR2076N` to dynamically (within a session) get (function: `G`=Get) or set (function: `S`=Set) the value of the Natural profile parameter `TIMEOUT` in the parameter file for the current session.

`TIMEOUT=`*`value`* specifies the number of seconds the client should wait for an RPC server response. If this time is exceeded, the remote procedure call is aborted with a corresponding error message.

Caution: This modification is retained as long as the user session is active; it is lost when the session is terminated.

For further information, refer to the description contained in the text object `USR2076T` in the library `SYSEXT`.

Executing Server Commands

Active servers that have been defined in the service directory (see ***Specifying RPC Server Addresses***) can be controlled with the `SYSRPC` server command execution function as described in the relevant section in the *SYSRPC Utility* documentation.

Sending Command to RPC Server

The Application Programming Interface (API) `USR9204N` enables you to send a command to a server on an EntireX Broker node.

First, this interface retrieves a list of RPC server replicas registered on a specified EntireX Broker. For each server, the list indicates the start date/time, host name, application name, and IP address (if available). It also indicates the sequence number assigned to each server during registration.

The interface then sends a send-command request with the sequence number to the EntireX Broker. The command transports a subprogram or an RPC command with the sequence number for an RPC server (or to all sequence numbers of an RPC service).

To make use of `USR9204N`

1. Copy the subprogram USR9204N from the library SYSEXT to the library SYSTEM or to the steplib library, or to any application in the Natural RPC server environment and proceed as described on the text object `USR9204T` in the library SYSEXT.
2. Supported commands:

   - `P,`*`subprogram`*`,`*`parameter-values`*

     The subprogram will be directly executed in the current library of the RPC server. A copy of the object module of the subprogram must be present on the client side to retrieve its parameter definitions. For parameter-values, all values should be separated with a comma.

   - `C,`*`RPC-command`*

     The RPC command will be directly executed in the current library of the RPC server. The supported RPC commands are `PING`, `TERMINATE`, `TRACE0`, `TRACE1` - `TRACE9`, `TRACE1E` - `TRACE9E`, and `TRACEINFO`.

     Note: The command is rejected if the CIS version used by the current Natural session is not high enough to support the requested EntireX command (required: EntireX Broker 11.0 with CIS version 13 and higher).

Logon to a Server Library

The server library on which the CALLNAT is executed depends on the RPC ***Logon Option*** on the client side and a couple of parameters on the server side.

The following table shows which the relevant parameters are and how they influence the library setting:

	Client		**Server**				
	1	2	3	4	5	6	7
	*library-id	RPC LOGON flag for server entry set?	LOGONRQ set?	Server started with STACK=	NSC or native Natural?	NSC: RPC Logon option in library profile	Server *library-id
1	Lib1	no	no	logon lib1	No influence	N/--	Lib1
2	Lib1	no	no	logon lib2	No influence	N/--	Lib2
3	Lib1	no	yes	(Client LOGON flag = NO) *and* (LOGONRQ=YES) is not possible.			
4	Lib1	yes	No influence	No influence	NSC	AUTO	Lib1
5	Lib1	yes	No influence	No influence	NSC	N	Lib1
6	Lib1	yes	No influence	No influence	Native Natural	--	Lib1

Explanation of the table columns:

1. The library ID of the client application where the CALLNAT is initiated.
2. The value of the RPC LOGON flag. Can be set for a whole node or a server.

   The flag can be set by using

   the Service Directory Maintenance function of the SYSRPC utility,

   or the profile parameter DFS,

   or the application programming interface USR2007N.
3. The profile parameter LOGONRQ can be set at server startup.
4. The library ID to which the server is positioned at its startup.
5. Does the server run under Natural Security (NSC) (see ***Using Natural RPC with Natural Security***) or not?
6. The setting of the Logon option in the NSC *Library Profile Items* (*Session options* > *Natural RPC Restrictions*) of the NSC server application. If the NSC *Logon Option* is set to A (AUTO), only library and user ID are taken. If set to N (default), the library, user ID and password parameters are evaluated.

7. The library on the server where the CALLNAT program is finally executed.

Using the Logon Option

The Logon option defines on which library the remote subprogram is to be executed. See also *Logon to a Server Library*.

Note: When you do not use the Logon option, the CALLNAT is executed on the library to which the server is currently logged on. This server logon is defined with the Natural profile parameter STACK=(LOGON *library*). The server will search for the CALLNATs to be executed in *library* (and all associated steplibs defined for *library*).

A client application can be enabled to execute a subprogram on a different library by setting the Logon option for this subprogram. This causes the client to pass the name of its current library to the server, together with this Logon option. The server will then logon to this library, searching it for the desired subprogram and, if the latter is found, it will execute it. After that, it will logoff from the previous library.

Logging on to a Different Library

If the server should logon to a library other than the client's current library, the client has to call the application programming interface USR4008N before the remote CALLNAT is executed. With USR4008N the client specifies an alternate name of a library to which the server will logon. The name of this library will be used for all subsequent calls to remote subprograms for which the Logon option applies. If blank is specified for the library name, the name of the current client library will be used again.

To make use of USR4008N

1 Copy the subprogram USR4008N from the library SYSEXT to the library SYSTEM or to the steplib library, or to any application in the server environment.

2 Using the DEFINE DATA statement, specify the following parameters:

Parameter	I/O	Format	Description	
P-FUNC	I	A01	Function code; possible values are:	
			P (Put)	Specify a new library for remote CALLNAT execution.
			G (Get)	Retrieve previously specified library for remote CALLNAT execution.
P-LIB	I	A8	Library on server for remote CALLNAT execution.	

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR4008N' P-FUNC P-LIB
```

Note: The calling program must be executed before the Natural RPC client invokes a remote CALLNAT.

Settings Required on the Client Side

To set the Logon option, you can use either the SYSRPC Service Directory maintenance function (see the relevant section in the SYSRPC Utility documentation) or - when using a default server - the profile parameter DFS or the application programming interface USR2007N.

Settings Required on the Server Side

No setting is required on the server side.

Using Compression

Compression types may be: 0, 1 or 2. Interface objects generated with COMPR=1 or 2 can help reduce the data transfer rate.

Compression Type	Description
COMPR=0	All CALLNAT parameter values are sent to and returned from the server, i.e. no compression is performed.
COMPR=1	M-type parameters are sent to and returned from the server, whereas O-type parameters are only transferred in the send buffer. A-type parameters are only included in the reply buffer. The reply buffer does not contain the Format description. This is the default setting.
COMPR=2	Same as for COMP=1, except that the server reply message still contains the format description of the CALLNAT parameters. This might be useful if you want to use certain options for data conversion by EntireX Broker (for more information, see the description of Translation Services in the EntireX Broker documentation).

Using Secure Socket Layer

The Natural RPC supports Secure Socket Layer (SSL) for the TCP/IP communication to the EntireX Broker.

To enable the EntireX Broker to recognize that the TCP/IP communication should use SSL, you must use one of the following methods:

- Append the string `:SSL` to the node name. If the node name has already been postfixed by the string `:TCP`, `:TCP` must be replaced by `:SSL`.
- Prefix the node name with the string `//SSL:`

Example:

```
SRVNODE='157.189.160.95:1971:SSL'
```

Before you access an EntireX Broker using SSL, you must first invoke the application programming interface `USR2035N` to set the required SSL parameter string.

To make use of `USR2035N`

1. Copy the subprogram `USR2035N` from the library `SYSEXT` to the library `SYSTEM` or to the steplib library, or to any application in the server environment.
2. Using the `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description	
FUNCTION	I	A01	Function code; possible values are:	
			`P` (Put)	Specify a new SSL parameter string. The SSL parameter string is internally saved and passed to EntireX each time an EntireX Broker using SSL communication is referenced the first time. You may use different SSL parameter strings for several EntireX Broker connections by calling application programming interface `USR2035N` each time before you access the EntireX Broker the first time. **Example:**

Parameter	I/O	Format	Description	

				`FUNCTION := 'P'`
				`SSLPARMS := ↩`
				`'TRUST_STORE=FILE://DDN:CACERT&VERIFY_SERVER=N'`
				`CALLNAT 'USR2035N' USING FUNCTION SSLPARMS ↩`
				
				To set SSL parameters in case of a Natural RPC server, put the name of the calling program onto the Natural stack when starting the server.
				
				Example:
				
				`STACK=(LOGON server-library;set-SSL-parms)`
				
				Where *`set-SSL-parms`* is a Natural program that invokes the application programming interface `USR2035N` to set the SSL parameter string.

			`G` (Get)	Retrieve previously specified SSL parameter string. The previously put SSL parameter string is returned to the caller. For more information about the SSL parameter string, refer to the EntireX documentation.
`SSLPARMS`	I	A128	SSL parameter string as required by the EntireX Broker	

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR2035N' FUNCTION SSLPARMS
```

Monitoring the Status of an RPC Session

This part is organized in the following sections:

- Using the RPCERR Program
- Using the RPCINFO Subprogram
- Using the Server Trace Facility

- Defining the Trace File

Using the RPCERR Program

You can run the RPCERR program from the Command line or invoke it by using a FETCH statement from within a Natural program. RPCERR displays the following information:

- The last Natural error number and message if it was RPC related.
- The last EntireX Broker message associated with this error.
- The last EntireX RPC server error message if the Natural error number is related to the EntireX RPC server error.

In addition, the node and server name from the last EntireX Broker call can be retrieved.

Example of an RPC Error Display: RPCERROR

```
Natural error number: NAT6972
Natural error text  :
Directory error on Client, reason 3.

RPC error information:
No additional information available.

Server Node:                                        Library: SYSTEM
Server Name:                                        Program: NATCLT3
                                                    Line No: 1010        ↩
```

Using the RPCINFO Subprogram

You can use the subprogram RPCINFO in your application program to retrieve information on the state of the current RPC session. This also enables you to handle errors more appropriately by reacting to a specific error class.

The subprogram RPCINFO is included in the library SYSTEM and can be called by any user application.

A sample program TESTINFO is included in the library SYSRPC together with the parameter data area RPCINFOL for calling RPCINFO.

Example:

```
DEFINE DATA LOCAL USING RPCINFOL
  LOCAL
  1 PARM   (A1)
  1 TEXT   (A80)
  1 REDEFINE TEXT
    2 CLASS  (A4)
    2 REASON (A4)
END-DEFINE
...
OPEN CONVERSATION USING SUBPROGRAM 'APPLSUB1'
  CALLNAT 'APPLSUB1' PARM
CLOSE CONVERSATION *CONVID
...
ON ERROR
   CALLNAT 'RPCINFO' SERVER-PARMS CLIENT-PARMS
   ASSIGN TEXT=C-ERROR-TEXT
   DISPLAY CLASS REASON
END-ERROR
...
END
```

Parameters of RPC Info

`RPCINFO` has the following parameters which are provided in the parameter data area `RPCINFOL`:

Parameter	Format	Description
`SERVER-PARMS`		Contains information about the Natural session when acting as a server. The `SERVER-PARMS` only apply if you execute `RPCINFO` remotely on an RPC server.
`S-BIKE`	A1	Transport protocol used; possible value:
		`B` — EntireX Broker
`S-NODE`	A32	The node name of the server.
`S-NAME`	A32	The name of the server.
`S-ERROR-TEXT`	A80	Contains the message text returned from the transport layer.
`S-CON-ID`	I4	Current conversation ID. Note that this is the physical ID from EntireX Broker, not the logical Natural ID. This parameter always contains a value as EntireX Broker generates IDs for both conversational and non-conversational calls. If the physical conversation ID is either non-numeric or greater than I4, a `-1` is returned.

Parameter	Format	Description
S-CON-OPEN	L	Indicates whether there is an open conversation. This parameter contains the value TRUE if a conversation is open, otherwise it contains FALSE.
CLIENT-PARMS		Contain information about the Natural session when acting as a client.
C-BIKE	A1	Transport protocol used; possible value: B — EntireX Broker
C-NODE	A32	The node name of the previously addressed server.
C-NAME	A32	The name of the previously addressed server.
C-ERROR-TEXT	A80	Contains the message text returned from the transport layer.
C-CON-ID	I4	Conversation ID of the last server call. Note that this is the physical ID from EntireX Broker, not the logical Natural ID. If no conversation is open, the value of this parameter is less than or equal to 0. If the physical conversation ID is either non-numeric or greater than I4, a -1 is returned.
C-CON-OPEN	L	Indicates whether there is an open conversation. This parameter contains the value TRUE if a conversation is open, otherwise it contains FALSE.
C-ENTIREX-RPC-ERROR-MESSAGE	A	Contains the message text returned from an EntireX RPC server.

Using the Server Trace Facility

Natural RPC includes a trace facility that enables you to monitor server activities and trace possible error situations.

Activating/Deactivating the Server Trace Facility

To activate/deactivate the server trace facility, start the server with the option or set the trace level dynamically with help of the SYSRPC utility.

```
TRACE=n
```

The integer value *n* represents the desired trace level; that is, the level of detail in which you want your server to be traced. The following values are possible:

Value	Trace Level
0	No trace is performed (default).
1	All client requests and corresponding server responses are traced and documented.
2	All client requests and corresponding server responses are traced and documented; in addition, all RPC data are written to the trace file.

The RPC trace facility writes the trace data to the Natural Report Number 10.

In case of a conversion error which is reported with Natural error number NAT6974 and reason codes 2 and 3, the position of the erroneous data in the buffer is indicated.

Support of TS=ON for RPC Server Trace

The following information applies to z/OS environments only:

All messages in the Natural RPC server trace are converted into upper case if `TS=ON` is specified in the Natural RPC server session. The trace of the data from/to the client is not affected by `TS=ON` and remains unchanged.

Defining the Trace File

The trace file definition depends on the environment:

- Trace File Handling for z/OS Environments - General Information
- Trace File Handling in z/OS Batch Mode
- Trace File Handling under CICS
- Trace File Handling for Linux Environments
- Trace File Handling for Windows

Trace File Handling for z/OS Environments - General Information

On z/OS, define the trace file appropriate to your environment, see also the `NTPRINT` macro (in the *Parameter Reference* documentation).

Trace File Handling in z/OS Batch Mode

a) Running A Server As Single Task

In the server start job, assign a z/OS dataset to the Natural additional Report `CMPRT10`.

Example:

```
//NATRPC   JOB  CLASS=K,MSGCLASS=X
//NATSTEP  EXEC PGM=NATOS
//STEPLIB  DD   DISP=SHR,DSN=SAG.NAT.LOAD
//         DD   DISP=SHR,DSN=SAG.EXX.LOAD
//CMPRMIN  DD   *
IM=D,MADIO=0,MT=0,OBJIN=R,AUTO=OFF,MAXCL=0,ID=',',INTENS=1,
PRINT=((10),AM=STD)
/*
//SYSUDUMP DD   SYSOUT=X
//CMPRT10  DD   SYSOUT=X
//CMPRINT  DD   SYSOUT=X
/*
```

b) Running a Server With Replicas

1. Set the RPC parameter `NTASKS` to a value greater than 1.
2. Assign `CMPRMIN` to a dataset with `DISP=SHR` or to `*`.
3. As each task writes on a separate `CMPRINT` dataset, define the following DD card names:

   `CMPRINT` for the main task;

   `CMPRINT1` to `CMPRINT9` for the first nine subtasks;

   `CMPRIN10` to `CMPRIN`*nn* for the next two-digit numbers of subtask, *nn*=`NTASKS-1`.
4. If the keyword subparameter `TRACE` of profile parameter `RPC` or parameter macro `NTRPC` is set, the trace facility writes to Printer 10.

   You must define the following DD card names:

   `CMPRT10` for the main task;

   `CMPRT101` to `CMPRT1`*nn* for all subtasks, *nn*=`NTASKS-1`;

Example:

```
//NATRPC   JOB  CLASS=K,MSGCLASS=X
//NATSTEP  EXEC PGM=NATOS,REGION=8M
//steplib  DD   DISP=SHR,DSN=SAG.NAT.LOAD
//         DD   DISP=SHR,DSN=SAG.EXX.LOAD
//CMPRMIN  DD   *
IM=D,MADIO=0,MT=0,OBJIN=R,AUTO=OFF,MAXCL=0,ID=',',INTENS=1,
PRINT=((10),AM=STD)
/*
//SYSUDUMP DD   SYSOUT=X
//CMPRT10  DD   SYSOUT=X
//CMPRT101 DD   SYSOUT=X
//CMPRT102 DD   SYSOUT=X
```

```
//CMPRT103 DD   SYSOUT=X
//CMPRINT  DD   SYSOUT=X
//CMPRINT1 DD   SYSOUT=X
//CMPRINT2 DD   SYSOUT=X
//CMPRINT3 DD   SYSOUT=X
/*
```

Trace File Handling under CICS

Under CICS, assign Print File 10 to a CICS extra-partitioned transient data queue.

Examples:

Natural dynamic profile definition:

```
PRINT=((10),AM=CICS,DEST=RPCT,TYPE=TD)
```

CICS definition:

```
RPCTRAC DFHDCT TYPE=SDSCI,                      X
               BLKSIZE=136,                     X
               BUFNO=1,                         X
               DSCNAME=RPCTRACE,                X
               RECFORM=VARUNB,                  X
               RECSIZE=132,                     X
               TYPEFLE=OUTPUT
        SPACE
RPCT    DFHDCT TYPE=EXTRA,                      X
               DSCNAME=RPCTRACE,                X
               DESTID=RPCT,                     X
               OPEN=INITIAL
```

CICS Startup JCL:

```
RPCTRACE DD    SYSOUT=*
```

Trace File Handling for Linux Environments

It is recommended that you use a different file name (that is, a different `NATPARM` parameter file) for each server so that you can trace them individually. The trace file is defined in the `NATPARM` parameter file of the Natural server:

1. **Report Assignments**
   Assign the logical device LPT10 to your Report Number 10.

2. **Device Parameter Assignments**
   Instead of selecting a physical printer specification for LPT10, specify a file name that represents the name of your trace file.

Example for Linux:

```
/bin/sh -c cat>>/filename
```

where *filename* represents the name of the trace file.

Trace File Handling for Windows

It is recommended that you use a different file name (that is, a different NATPARM parameter file) for each server so that you can trace them individually. The trace file is defined in the NATPARM parameter file of the Natural server (see Device/Report Assignments in the Configuration Utility):

1. **Reports**

   Assign the logical device LPT10 to your Report Number 10.
2. **Devices**

   Instead of selecting a physical printer specification for LPT10, specify a file name that represents the name of your trace file. As default, old trace files are deleted when a new file with the same name is created.

   If you wish to append the new log to the existing one, specify:

   ```
   >>filename
   ```

Retrieving Runtime Settings of a Server

The Natural application programming interface (API) USR4010N enables you to retrieve the runtime settings of a server:

- the system file assignments for FUSER, FNAT, and FSEC,
- the steplib chain.

To make use of USR4010N

1 Server environment:

   Copy the subprogram USR4010N from the library SYSEXT to the library SYSTEM or to the steplib library or to any application in the server environment.

2 Client environment:

In a DEFINE DATA statement, specify the following parameters as a redefinition of USR4010-PARM (A300):

Parameter	Format	Description
FUSER-DBID	N5	Database ID of system file FUSER.
FUSER-FNR	N5	File number of system file FUSER.
FNAT-DBID	N5	Database ID of system file FNAT.
FNAT-FNR	N5	File number of system file FNAT.
FSEC-DBID	N5	Database ID of system file FSEC.
FSEC-FNR	N5	File number of system file FSEC.
STEP-NAME	A8/15	Name of steplib.
STEP-DBID	N5/15	Database ID of steplib.
STEP-FNR	N5/15	File number of steplib.

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR4010N' USR4010-PARM
```

See also the Syntax Description of the CALLNAT statement.

4 If the RPC parameter AUTORPC=OFF is set, copy the interface USR4010X to the client environment with the name USR4010N.

If the RPC parameter AUTORPC=ON is set, the API must not be available in the client environment to avoid that the interface is called locally.

When USR4010N is called, the values of the parameter specified above are output in the group of fields USR4010-PARM.

Setting or Getting Parameters for EntireX

The Application Programming Interface (API) USR4009N enables you to set or to get the EntireX parameters that are currently supported by the Natural RPC. These are:

- Compression level
- Encryption level

To make use of `USR4009N`

1 Copy the subprogram `USR4009N` from the library `SYSEXT` to the library `SYSTEM` or to the steplib library or to any application in the server environment.

2 Using a `DEFINE DATA` statement, specify the following parameters:

Parameter	Format	I/O	Description	
`FUNCTION`	A01	I	Function; possible values are:	
			`G` (Get)	The values already set for the EntireX parameters are returned. If no `PUT` has been called before in the Natural session, all values are zero or blank.
			`P` (Put)	The values specified for the EntireX parameters are saved and used in all subsequent calls to EntireX.
`ENVIRONMENT`	A01	I	Environment; possible values are:	
			`S`	Server
			`C`	Client
			`B`	Both
`COMPRESSLEVEL`	A01	I/O	Compression level.	
`ENCRYPTION-LEVEL`	I01	I/O	Encyption level.	
`ACIVERS`	B02	O	API version used.	
`RC`	B01	O	Return code, unless equal to zero. Contains the API version required to set the requested parameter:	
			`0`	Function successful.
			`6`	Compression level requires API version 6.
			`7`	Compression level requires API version 7.

3 The interface can be called in two ways:

1. From within a program:

```
CALLNAT 'USR4009N' FUNCTION ENVIRONMENT
                   COMPRESSLEVEL
                   ENCRYPTION-LEVEL
                   ACIVERS RC
```

2. From the command prompt or by using the statement `STACK` with values for the above parameters.

   Examples:

```
USR4009P P,C,ENCRYPTION-LEVEL=1,COMPRESSLEVEL=6
USR4009P P,C,,6
```

In command mode, you may use the *keyword=value* notation to set only a subset of the EntireX parameters. The values for parameters that are not referenced remain unchanged.

Notes:

- The request is rejected and no values are saved if the API version used by the current Natural session is not high enough to support the requested EntireX parameter. In this case the RC contains the required API version.
- The EntireX parameters are only honored by the Natural RPC.

Maintain Message ID and Correlation ID of EntireX

The Application Programming Interface (API) USR8225N enables you to maintain message ID and correlation ID of EntireX in an RPC client or server environment.

To make use of USR8225N

1. Copy the subprogram USR8225N from library SYSEXT to the library SYSTEM or to the steplib library or to any application in the server environment.
2. Using a DEFINE DATA statement, specify the following parameters:

Parameter	Format	I/O	Description
FUNCTION	A08	I	Function code: ■ GENERATE ■ RETRIEVE
ENVIRONMENT	A06	I	Environment setting: ■ CLIENT ■ SERVER
MESSAGE_ID	A64	O	Message ID
CORRELATION_ID	A64	O	Correlation ID
RC	N04	O	Return value: ■ 0 - No errors ■ 1 - Invalid function code

Parameter	Format	I/O	Description
			■ 2 - Invalid environment ■ 3 - Not supported by Natural RPC

Handling Errors

- Remote Error Handling
- Avoiding Error Message NAT3009 from Server Program
- User Exit NATRPC01

Remote Error Handling

Any Natural error on the server side is returned to the client as follows:

- Natural RPC moves the appropriate error number to the `*ERROR-NR` system variable.
- Natural reacts as if the error had occurred locally.

Note: If theprofile parameter `AUTORPC` is set to `ON` and a subprogram cannot be found in the local environment, Natural will interpret this as a remote procedure call. It will then try to find this subprogram in the service directory. If it is not found there, a NAT6972 error will be issued. As a consequence, no NAT0082 error will be issued if a subprogram cannot be found.

See also ***Using the RPCERR Program***.

Avoiding Error Message NAT3009 from Server Program

If a server application program does not issue a database call during a longer period of time, the next database call might return a NAT3009 error message.

To avoid this problem, the optional user exit `NATRPC39` is provided. This exit is called in the following cases:

1. After a Ping command;
2. In case of communication via the EntireX Broker: each time the `SERVER-NONACT` time is exceeded.

To activate the user exit `NATRPC39`

1 Copy the sample exit `NATRPC39` from library `SYSRPC` to library `SYSTEM` on system file `FUSER`.

The steplib concatenation of the library to which the server currently is logged on is not evaluated.

2 Adapt the database ID which is assigned to field `ACB-RSP` to your needs.

3 Add additional `CALL 'CMADA'` statements, which reference additional database IDs to your `NATRPC39` if more than one Adabas database is involved.

User Exit NATRPC01

The user exit `NATRPC01` is called when a Natural error has occurred, actually after the error has been handled by the Natural RPC runtime and immediately before the response is sent back to the client. This means, the exit is called at the same logical point as an error transaction, that is, at the end of the Natural error handling, after all `ON ERROR` statement blocks have been processed.

In contrast to an error transaction, this exit is called with a `CALLNAT` statement and must therefore be a subprogram which must return to its caller.

The interface to this exit is similar to the interface of an error transaction. In addition, the exit can pass back up to 10 lines of information which will be traced by the Natural RPC runtime. Only lines which begin with a non-blank character will be traced.

A sample user exit `NATRPC01` can be found in the library `SYSRPC`.

Important Note:

The `DEFINE DATA PARAMETER` statement block must not be changed.

User Exits before and after Service Execution

To give administrators more control over the execution of services (remote CALLNATs), two optional user exits are called on the Natural RPC server side.

User Exit	Purpose
`NATRPC02`	The optional before-service-execution exit `NATRPC02` is called immediately before the service is executed. At this point in time, the request has already passed all security checks and the data is unmarshalled.
`NATRPC03`	The optional after-service-execution exit `NATRPC03` is called immediately after successful return from the service. At this point in time, the data is not yet marshalled. The exit is not called if an unhandled error has occurred.

These exits are independent of each other and can be used separately.

For both exits, the following rules apply:

- The exit must be located in library `SYSTEM` on the `FUSER` system file.

If the exit is found during startup of the Natural RPC server, a message is written to the Natural RPC server trace to indicate the activation of the exit. The exit is afterwards called unconditionally. If the exit is removed during the lifetime of the server session, a permanent NAT0082 error will occur.

If the exit is not found during startup of the Natural RPC server, the exit is never called during the lifetime of the server session. The exit cannot be enabled dynamically.

- The exit must be implemented by the user as a subprogram. The exit is called with a single dynamic variable as parameter. The content of the dynamic variable is the eight character long name of the remote subprogram.

  The use of the dynamic variable allows us to implement future extensions of the passed information without causing problems with existing user written exits.
- The exit is also called inside a conversation.
- The Natural RPC server does not intercept unhandled errors in the exit. If an unhandled error occurs in the exit, the error is propagated to the client.

The exits may be used for auditing or tracing purposes. `NATRPC02` may also be used for additional security checks.

Example for `NATRPC02`:

```
DEFINE DATA PARAMETER
1 SUBPROGRAM (A8) BY VALUE
END-DEFINE
IF *USER <> 'DBA' AND SUBPROGRAM = 'PRIVATE'
  *ERROR-NR := 999
END-IF
END
```

9 Using a Conversational RPC

- Opening a Conversation 98
- Closing a Conversation 99
- Defining a Conversation Context 100
- Modifying the System Variable *CONVID 100

Opening a Conversation

To open a conversation

1 Specify an `OPEN CONVERSATION` statement on the client side.

2 In the `OPEN CONVERSATION` statement, specify a list of services (subprograms) as members of this conversation.

The `OPEN CONVERSATION` statement assigns a unique conversation identifier to the system variable `*CONVID`.

More than one conversation may be open in parallel. If subprograms interfere with each other, the application programs are responsible to manage the various conversations by setting the appropriate `*CONVID`, which is evaluated by the `CALLNAT` instruction.

- If the subprogram is a member of the current conversation (referred to by `*CONVID`), it will be executed at the server task which is exclusively reserved for this conversation.
- If it is not member of the current conversation, it will be executed in a different server task. This also applies to different conversations.

A conversation can be opened on any program level and `CALLNAT`s within this conversation can be executed on any other program level below or above.

It is possible to open a client conversation within a remote `CALLNAT` executed on a server so the server acts as an agent. As the client only controls its own conversations, and not the server's, it is the application programmer's responsibility to ensure that the conversation on the server is closed properly before the main client is closed.

Additional Restrictions

The conversational RPC can still be tested locally. To keep the behavior identical if you execute a conversational `CALLNAT` remotely or locally, the following additional restrictions apply:

- A `CLOSE CONVERSATION` is not possible within an object which is currently running as a member of this conversation. This corresponds to the restriction that it is not possible to close a conversation from within a remotely running program.
- It is not possible to execute a conversational `CALLNAT` which is member of the conversation from within another (or the same) member of this conversation. This corresponds to the restriction that it is not possible to execute a conversational `CALLNAT` which is member of the client's conversation from a server subprogram.
- It is not recommended to open a conversation from within another conversation's subprogram.

Closing a Conversation

To close a conversation

- Specify a `CLOSE CONVERSATION` statement on the client side.

This enables the client to close a specific conversation or all conversations. All context variables of the closed conversation are then released and the server task will be available again for another client.

If you terminate Natural, you implicitly close all conversations.

When a server receives a `CLOSE CONVERSATION` request, it issues a `CLOSE CONVERSATION ALL` statement so that all conversations the server might have opened (as agent) are also closed.

Close a conversation with implicit `BACKOUT TRANSACTION` (Rollback)

By default, when a `CLOSE CONVERSATION` statement is executed, the Rollback option will be sent to the server together with the `CLOSE CONVERSATION` statement. This will cause an implicit `BACKOUT TRANSACTION` on the server side at the end of the conversation processing.

Close a conversation with implicit `END TRANSACTION` (Commit)

You can use the application programming interface `USR2032N` available in library `SYSEXT` to cause an implicit `END TRANSACTION` on the server side.

The application programming interface has to be called before the next `CLOSE CONVERSATION` statement is executed. The result is that the commit option is sent to the server together with the `CLOSE CONVERSATION` statement and that the server executes an `END TRANSACTION` statement at the end of the conversation processing.

The commit option applies to the next `CLOSE CONVERSATION` statement executed by the client application. After the conversation(s) has (have) been closed, the default option is used again. This means, that the following `CLOSE CONVERSATION` statements will result again in a `BACKOUT TRANSACTION` statement.

Defining a Conversation Context

During a conversation the subprograms that are members of this conversation may share a context area on this server.

To do so, declare a data area with the `DEFINE DATA CONTEXT` statement in each of the concerned subprograms.

The subprograms, using a context area, behave in the same way if the conversation were local or remote. The `DEFINE DATA CONTEXT` statement closely corresponds to the `DEFINE DATA INDEPENDENT` statement. All rules which apply to the definition of AIV variables also apply to context variables, with the exception that a context variable does not need to be prefixed by a plus sign (+).

The compiler does not check format/length definition because this requires that the variables be created by running a program which includes all definitions for this application (as usual with AIVs). This makes no sense for context variables, because a library containing RPC service routines is usually not application-dependent.

In contrast to AIVs, the caller's context variables are not passed across `CALLNAT` boundaries. Context variables are referenced by their name and the context ID they apply to. A context variable is shared by all service routines referring to the same variable name within one conversation. Therefore each conversation has its own set of context variables. Context variables cannot be shared between different conversations even if they have the same variable name.

The context area will be reset to initial values when an `OPEN CONVERSATION` statement or a non-conversational `CALLNAT` statement is performed.

Modifying the System Variable *CONVID

The system variable `*CONVID` (format I4) is set by the `OPEN CONVERSATION` statement and may be modified by the application program.

Modifying `*CONVID` is only necessary if you are using multiple conversations in parallel.

10 Reliable RPC

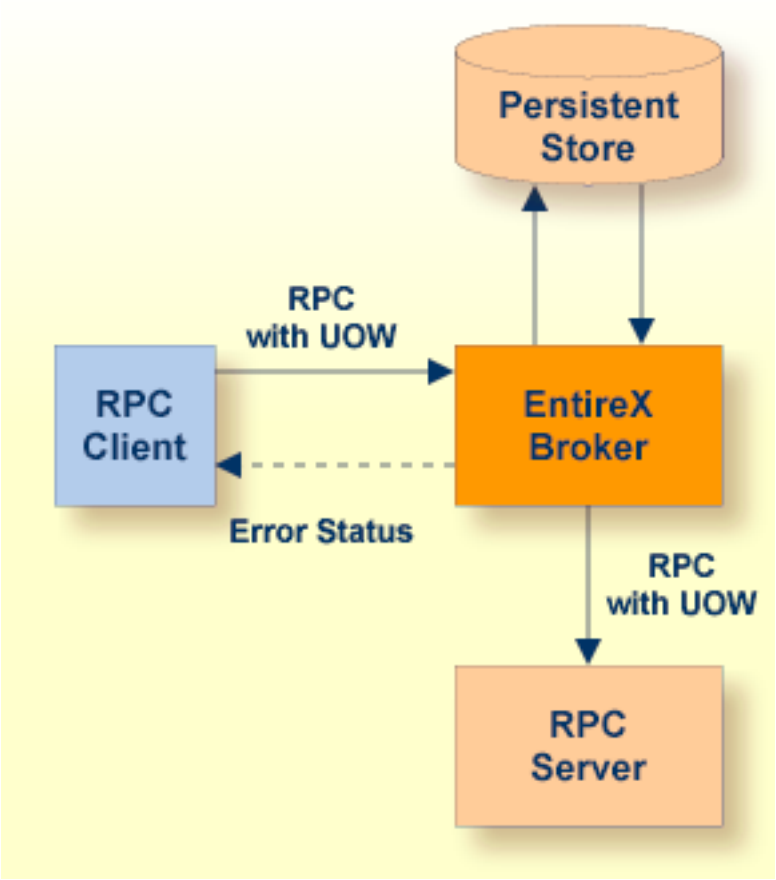
- General Information 102
- Reliable RPC on the Natural RPC Client Side 103
- Reliable RPC on the Natural RPC Server Side 106
- Viewing the Status of Reliable RPC Messages 106

General Information

In the architecture of modern e-business applications (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 Natural RPC implementation of a reliable messaging system. It combines the Natural RPC technology and persistence, which is implemented by means of units of work (UOWs) that are offered by the EntireX Broker. Reliable RPC is characterized by following features:

- The Natural RPC client executes a `CALLNAT` statement without waiting for a reply from the server (the RPC message is sent in asynchronous mode).
- An RPC server needs not be active at the time the `CALLNAT` is executed.
- The reliable RPC message is stored in the EntireX Broker's persistent store until an RPC server is available.
- The Natural RPC server executes the reliable RPC by calling the requested subprogram but does not send a reply to the RPC client.
- A Natural RPC client may ask the status of the sent reliable RPC messages.
- A Natural RPC client may send a reliable RPC message to an EntireX RPC server.
- A Natural RPC server may receive a reliable RPC message from an EntireX RPC client.



Reliable RPC on the Natural RPC Client Side

The Natural RPC client for a reliable RPC is configured in the same way as for a normal Natural RPC. The same Natural RPC client session can send standard RPC requests and reliable RPC messages.

To enable a Natural RPC client to use reliable RPC, the Natural RPC client must use application programming interface `USR2071N` for an explicit EntireX Broker logon. This implies that the profile parameter `ACIVERS` must be set to `2` or higher.

Reliable RPC is used to send messages to a persistent EntireX Broker service. The messages are described by the PDA of the caller and may only contain output parameters. A parameter is defined as "output" in one of the following ways:

- If a Natural **interface object** is used:

  In the `Attr` field on the **Interface Object Generation** screen of the `SYSRPC` utility, set the attribute of the parameter to `O` (output).

Note: If your parameter definitions do not contain group structures, you must set the attribute COMPAT to IDL before generating the interface object; see *Generating Interface Objects - General Considerations* in the SYSRPC Utility documentation.

- If no Natural interface object is used:

  Use the AD=O session parameter in the CALLNAT statement.

  Note: If you want to call an EntireX RPC server and if the corresponding IDL file contains group structures, then you must use a Natural interface object, and the parameter definition for the Natural interface object must correspond to the group structure of the IDL file.

- If you generate a Natural interface object from an IDL file, the attribute of the parameter is taken from the IDL file. In this case, the IDL file must only contain inbound (from the server's point of view) parameters.

Reliable RPC is enabled at runtime. The client has to set one of two different modes before issuing a reliable RPC request:

- AUTO_COMMIT
- CLIENT_COMMIT

While AUTO_COMMIT commits each message implicitly after sending it, a series of RPC messages sent in a unit of work (UOW) can be committed or rolled back explicitly using CLIENT_COMMIT mode.

For this purpose, Natural provides the two application programming interfaces USR6304N and USR6305N. With interface USR6304N, the mode for reliable RPC is set. With interface USR6305N, a unit of work that has been created with CLIENT_COMMIT can be committed or rolled back.

To make use of USR6304N

1. Copy the subprogram USR6304N from the library SYSEXT to the library SYSTEM or to the steplib library or to any application in the client environment.
2. Using the DEFINE DATA statement, specify the following parameters:

Parameter	I/O	Format	Description	
P-FUNC	I	A01	Function code; possible values are:	
			P (Put)	Set the mode for reliable RPC. The mode applies to all subsequent calls.
			G (Get)	Get the previously specified mode.
P-MODE	I/O	N01	Mode for reliable RPC:	

Parameter	I/O	Format	Description	
			0	No reliable RPC (standard RPC execution)
			1	Reliable RPC AUTO_COMMIT
			2	Reliable RPC CLIENT_COMMIT
P-RC	O	N04	Return code	
P-MESSAGE	O	A80	Message text	

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR6304N' P-FUNC P-MODE P-RC P-MESSAGE
```

Note: The mode CLIENT_COMMIT cannot be changed if reliable RPC messages have been sent but not yet committed or rolled back.

To make use of USR6305N

1 Copy the subprogram USR6305N from the library SYSEXT to the library SYSTEM or to the steplib library or to any application in the client environment.

2 Using the DEFINE DATA statement, specify the following parameters:

Parameter	I/O	Format	Description	
P-FUNC	I	A08	Function code; possible values are:	
			COMMIT	Commit the sent reliable RPC messages. The messages are now available for an RPC server.
			ROLLBACK	Discard the already sent reliable RPC messages.
P-RC	O	N04	Return code	
P-MESSAGE	O	A80	Message text	

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR6305N' P-FUNC P-MODE P-RC P-MESSAGE
```

Reliable RPC on the Natural RPC Server Side

The Natural RPC server for reliable RPC is configured in the same way as for normal Natural RPC. The same Natural RPC server session can process standard RPC requests and reliable RPC messages.

To enable the processing of reliable RPC messages, the profile parameter `ACIVERS` must be set to `2` or higher.

Viewing the Status of Reliable RPC Messages

To view the status of sent reliable RPC messages, Natural provides the application programming interface `USR6306N`. With `USR6306N` you can get the status of all reliable RPC messages that you have previously sent under your user ID. `USR6306N` must not necessarily be called within the Natural session in which the reliable RPC messages have been sent. If `USR6306N` is used in a different Natural session, the application programming interface `USR2071N` must first be used to log on to the EntireX Broker with the same user ID that has been used to send the reliable RPC messages.

The reliable RPC messages are implemented by EntireX Broker unit of works (UOWs). The information about reliable RPC messages is therefore information about UOWs.

To make use of `USR6306N`

1. Copy the subprogram `USR6306N` from the library `SYSEXT` to the library `SYSTEM` or to the steplib library or to any application in the client environment.
2. Using the `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description
`P-UOW-ID-IN`	I	A16	ID for the UOW to be retrieved. Possible values are: UOWID of a UOW `LAST`: the UOW for the last reliable RPC message in the current Natural session `ALL` or blank: all UOWs for the logged on EntireX Broker user
`P-USER-ID`	O	A32	User ID of the user who has created the UOWs.
`P-BROKER-ID`	O	A32	Broker ID of the EntireX Broker that hosts the UOWs.
`P-UOW-COUNT`	O	I4	Number of UOWs in the `P-UOW-INFO` array.
`P-UOW-INFO (/1:*)`			X-array with information about each UOW
`P-UOW-ID`	O	A32	ID of the UOW

Parameter	I/O	Format	Description
P-UOW-STATUS	O	A10	Status of the UOW according to EntireX Broker The status depends on the processing state and is assigned by the EntireX Broker.
P-USERR-STATUS	O	A32	User information about the UOW. This is typically error information that has been set by the RPC server.
P-CREATE-TIME	O	A32	Creation time of the UOW according to EntireX Broker.
P-RC	O	N04	Return code
P-MESSAGE	O	A80	Message text

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR6306N' P-UOW-ID-IN P-USER-ID P-BROKER-ID P-UOW-COUNT P-OUW-INFO(*) ↩
P-RC P-MESSAGE
```

11 Using a Remote Directory Server - RDS

- RDS Principles of Operation 110
- Using a Remote Directory Server 112
- Creating an RDS Interface 113
- Creating a Remote Directory Service Routine 114
- Remote Directory Service Program RDSSCDIR 115

RDS Principles of Operation

You have two options to use a service directory:

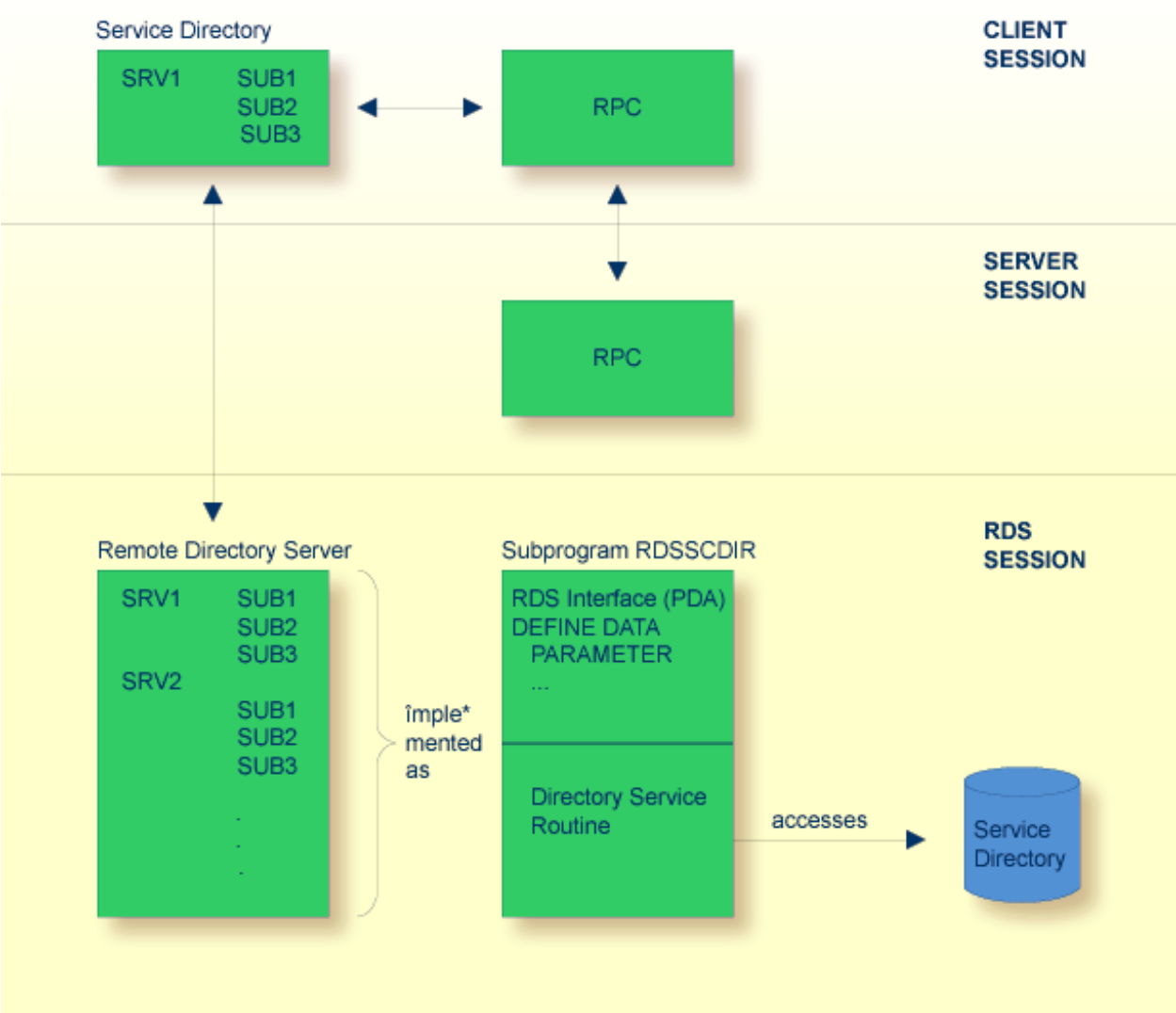
1. **Using a service directory in a Natural subprogram.**

   Normally, to locate a service, the Natural RPC uses a service directory in a Natural subprogram. This directory is an initialized LDA data structure in program `NATCLTGS` generated by the `SYSRPC` *Service Directory Maintenance* function and has to be available to every RPC client application.

2. **Using a remote directory.**

   You can use a remote directory to locate a service. A remote directory server (RDS) enables you to define directory definitions in one place so that the RDS's services can be used by all clients in your environment.

This section describes **how to use a remote directory server to locate a service**.



The remote directory server is implemented as a Natural subprogram.

A sample of such a subprogram is provided in library `SYSRPC`. It is named `RDSSCDIR` and reads the required directory information from a work file. The interface of this subprogram is documented, which enables you to develop your own remote directory service. For more information, see the section *Creating an RDS Interface*.

The RDS interface is the Natural parameter data area of the Natural subprogram and the directory service routine is the code section of the Natural subprogram. If a remote `CALLNAT` is not found within the client's local service directory, the RPC runtime contacts the remote directory server by executing an internal remote `CALLNAT`.

An internal directory cache minimizes the access to the remote directory. The cache information is controlled by an expiration time which is defined by the remote directory server.

Using a Remote Directory Server

To use a remote directory server

1 **Create a Directory File**

Create a directory file for the remote directory service using the *Service Directory Maintenance* function of the `SYSRPC` utility. The subprogram `RDSSCDIR` is provided in the library SYSRPC and reads the directory information from a Natural work file (fixed-block, record length 80 bytes). This is a file on `<install-dir>`/*etc* named `servername`.*DIR* where `servername` is the name of the directory server.

2 **Start the Remote Directory Server**

Start the remote directory server and proceed with the following steps.

3 **Define RDS**

You have two options:

- Specify the RDS in the profile parameter `RDS`.
- Or use the maintenance function of the `SYSRPC` utility to define remote directory servers (refer to *Service Directory Maintenance* in the *SYSRPC Utility* documentation). The definition of remote directory servers is still supported for reasons of compatibility. You should, however, define your RDS in the profile parameter `RDS`. For this purpose, entries are provided that allow to define the location of the directory server. This enables you to expand existing local directory information by one or more remote directory server definitions.

Below is an example of how to define a remote directory server in the service directory `NATCLTGS`.

Service Directory					
	NODE	**SERVER**	**LIBRARY**	**PROGRAM**	**LOGON**
1	NODE1				
2		SERVER1			
3			SYSTEM		
4				TESTS1	
5				TESTS2	
6	RDSNODE				
7		DIRSRV1			
8			#ACI		
9				RDSSCDIR	

This example locally defines a server named SERVER1. This server may execute the services TESTS1 and TESTS2.

Additionally, there are definitions for the remote directory server DIRSRV1. A remote directory server is identified by a preceding hash (#) sign for the library definition.

The definitions of NODE and SERVER are used as usual in Natural RPC. The library definition defines the transport protocol (ACI) which has to be used to connect the RDS.

Finally, the PROGRAM entry contains the name of the remote subprogram which represents the remote directory service (in this case, it refers to the sample subprogram RDSSCDIR).

Creating an RDS Interface

The RDS interface is the parameter data area (PDA) of a Natural subprogram.

To create your own RDS interface you can use the parameter data area shown below.

```
DEFINE DATA PARAMETER
  1 P_UDID(B8)                      /* OUT
  1 P_UDID_EXPIRATION(I4)           /* OUT
  1 P_CURSOR(I4)                    /* INOUT
  1 P_ENTRIES(I4)                   /* IN
  1 P_REQUEST(A16/1:250)            /* IN
  1 P_EXTENT (A16/1:250)            /* OUT
  1 P_RESULT(A32)                   /* OUT
  1 REDEFINE P_RESULT
    2 SRV_NODE(A8)
    2 SRV_NODE_EXT(A8)
    2 SRV_NAME(A8)
    2 SRV_NAME_EXT(A8)
END-DEFINE
```

For an explanation of the parameters, refer to the table below.

Parameter	Format/Length	Explanation
P_UDID	B8	Unique directory identifier, should be increased after changing the directory information. The client saves this identifier in its cache. If the binary number increases from one client request to the next, it causes the client to delete its local cache information, because it no longer corresponds to the remote directory information.
P_UDID_EXPIRATION	I4	This defines the expiration time in seconds, that is, the number of seconds during which the client can use its local cache information without connecting the RDS to validate the UDID setting. It allows you to define a time limit after which you can be sure that your directory modifications are active for all clients. If you set this time

Parameter	Format/Length	Explanation
		to an unnecessarily low value, you may cause a lot of network traffic to the RDS.
P_CURSOR	I4	The remote procedure call has the option to scan for an alternative server if a connection to the previous one cannot be established, see the profile parameter TRYALT. This parameter contains zero for a scan from the top and may be modified by the RDS to remember the record location to continue the scan. The value will not be evaluated by the client, it will only be inserted from the cache to continue scanning.
P_ENTRIES	I4	This parameter contains the number of service definitions in P_REQUEST.
P_REQUEST	A16/1:250	A list of services for which a server address can be scanned. An entry is structured as: program name (A8) library name (A8)
P_EXTENT	A16/1:250	Reserved for future use.
SRV_NODE	A8	Contains the server node.
SRV_NODE_EXT	A8	Contains the server node extension.
SRV_NAME	A8	Contains the server name.
SRV_NAME_EXT	A8	Contains the server name extension.

Creating a Remote Directory Service Routine

The Remote Directory Service Routine is the code area of a Natural subprogram (the default version of this code area is subprogram RDSSCDIR in library SYSRPC).

To create your own RDS routine

- Modify the pseudo-code documented below.

```
Set UDID and UDID_EXPIRATION values
IF P_ENTRIES = 0
    ESCAPE ROUTINE
IF P_CURSOR != 0
   position to next server entry after P_CURSOR
   Scan for server which may execute P_REQUEST(*)
IF found
   SRV_NODE        = found node name
   SRV_NODE_EXT    = node extension
   SRV_NAME        = found server name
   SRV_NAME_EXT    = server extension
```

```
   P_CURSOR        = position of found server
ELSE
   P_CURSOR = 0
```

Remote Directory Service Program RDSSCDIR

This program is to be found in library `SYSRPC`. It reads the directory information from a work file (fixed-block, record length 80 byte).

Your program could also read the directory information from elsewhere (from a database, for example). This is a file in `<install-dir>`/*etc* named `servername`.*DIR*, where `servername` is the name of the directory server.

Structure of the Directory Work File

```
* comment
UDID definition
UDID_EXPIRATION definition
node definition
...
node definition
```

UDID Definition

```
(UDID)
   binary number            ↩
```

UDID_EXPIRATION Definition

```
(UDID_EXPIRATION)
   number of seconds        ↩
```

Node Definition

```
(NODE)
   namevalue     (logon-option)
   server definition
   ...
   server definition
```

Server Definition

```
(SERVER)
   namevalue     (logon-option)
   library definition
   ...
   library definition
```

Library Definition

```
(LIBRARY)
   namevalue
   program definition
   ...
   program definition
```

Program Definition

```
(PROGRAM)
   namevalue
   ...
   namevalue
```

Namevalue

```
Max. 8 characters in uppercase
```

The *logon-option* after *namevalue* as well as the following definition lines are optional. For the possible values of *logon-option*, refer to *Service Directory Maintenance* in the SYSRPC utility documentation.

Example Directory Read from the Work File:

```
(UDID)
ACB8AAB4777CA000
  (UDID_EXPIRATION)
  3600
  * this is a comment
  (NODE)
  NODE1
         (SERVER)
         SERVER1
             (LIBRARY)
             SYSTEM
                  (PROGRAM)
                  TESTS1
                  TESTS2
                  TESTS3
         (SERVER)
         SERVER2   (logon-option)
```

```
              (LIBRARY)
              SYSTEM
                   (PROGRAM)
                   TESTS4
(NODE)
NODE2   (logon-option)
      (SERVER)
      SERVER1
            (LIBRARY)
            SYSTEM
                 (PROGRAM)
                 TESTS1
                 TESTS2
                 TESTS3
                 TESTS4    ↩
```

In the above example, the directory contains:

- Two servers `SERVER1` and `SERVER2` running on node `NODE1`.

  The server `SERVER1` may execute the programs `TESTS1`, `TESTS2` and `TESTS3` in library `SYSTEM`.

  The server `SERVER2` may execute the program `TESTS4` on library `SYSTEM`.
- One server `SERVER1` on node `NODE2` which may execute the programs `TESTS1` - `TESTS4` in library `SYSTEM`.

The indentation of the lines in the example above is not required. All lines may start at any position (one). You can modify this file manually or generate it using the `SYSRPC` *Service Directory Maintenance* function.

12 Using Security

- Using Natural RPC with Natural Security 120
- Impersonation (z/OS Batch Mode) 126
- Impersonation (CICS) 130
- Using Natural RPC with EntireX Security 136

Using Natural RPC with Natural Security

Natural RPC also supports Natural Security in client/server environments, where security may be active on either (or both) sides.

For general information, refer to the *Natural Security* documentation.

For information on how to control the use of Natural RPC (Remote Procedure) calls in a client/server environment, see *Protecting Natural RPC Servers and Services* in the *Natural Security* documentation.

Client Side

The client must send logon data together with the RPC request. The logon data includes a user ID, password and a library.

- User ID and password are used to perform the authentication of the client on the Natural RPC server side.
- The library is used to perform a Natural Security protected logon to the requested library.

The following applies to Natural RPC clients only. For EntireX RPC clients that access a Natural Security protected Natural RPC server, refer to the EntireX Developer's Kit documentation.

To send logon data to the Natural RPC server, the Logon option must be used. See *Operating a Natural RPC Environment*, ***Using the Logon Option***. The logon data parts are established as follows:

1. The user ID and password:

   If the client runs under Natural Security

   The user ID and password from the Natural Security logon on the client are used and passed to the Natural RPC server.

   If you want to use a different user ID and/or password for the Natural Security logon on the server side, you can use the application programming interface `USR1071N`.

   Note: You can disallow the use of `USR1071N` in the Natural RPC restrictions part of the *Session Parameters* restrictions of the Natural Security library profile.

   If the client does not run under Natural Security

   To specify the user ID and password that are passed to the Natural RPC server, the client must call application programming interface `USR1071N` before the first RPC request is sent.

2. The library:

By default, the name of the library to which the client is currently logged on is used. If you want to pass another library name to the Natural RPC server, you may use the application programming interface USR4008N.

If impersonation without password check is active for the Natural RPC server (field Impersonation described in the section *Components of an RPC Server Profile* in the *Natural Security* documentation is set to A), the client may optionally pass an ETID to the Natural RPC server. This ETID will be used by the Natural RPC server to access Adabas on behalf of the client. To specify an ETID on the Natural RPC client side, you may use the application programming interface USR4371N.

USR1071N

The application programming interface USR1071N is provided in the library SYSEXT. It is used to specify the user ID and password that are passed to the Natural RPC server.

To make use of USR1071N

1 Copy the subprogram USR1071N and the program USR1071P from library SYSEXT to the library SYSTEM in the system file FNAT in the server environment. For more information, see *Using a Natural API* in the *SYSEXT Utility* documentation.

2 Using a DEFINE DATA statement, specify the following parameters:

Parameter	I/O	Format	Description	
USERID	I	A08	User ID to be used.	
PASSWORD	I	A08	Password to validate the user ID. This password is not validated on the client side.	
MIXEDCASE	I	A01	Mixed case option for password (optional).	
			Y	Allow mixed case password.
			N	Convert passwords to upper case.

3 In the calling program on the client side, specify the following statement:

```
FETCH RETURN 'USR1071P' USERID PASSWORD [MIXEDCASE]
```

Alternatively, you can invoke USR1071P from the command line, and enter the user ID and password in the displayed window.

For more information, see the USR1071T text object in the system library SYSEXT.

Note: Two samples are provided to call USR1071N: USR1071P, which is passing just user ID and password, and USR1071X (extended version), which in addition enables the user to set/retrieve various data.

USR2071N

To make use of `USR2071N`

1 Copy the subprogram `USR2071N` from library `SYSEXT` to the library `SYSTEM` or to the steplib library or to any application in the server environment.

2 Using a `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description	
`function`	I	A08	Function code; possible values are:	
			`LOGON`	Logon to EntireX Broker.
			`LOGOFF`	Logoff from EntireX Broker.
`broker-id`	I	A192	Broker ID. **Note:** For compatibility reasons `broker-id` is defined with `BY VALUE RESULT` to support existing callers which pass an A8 or A32 field for the `broker-id`. The sample `USR2071P` provided in the library `SYSEXT` supports up to 32 characters.	
`user-id`	I	A32	User ID.	
`password`	I	A128	User ID's password. **Note:** For compatibility reasons `password` is defined with `BY VALUE RESULT` to support existing callers which pass an A32 field for the `password`. The sample `USR2071P` provided in the library `SYSEXT` supports up to 32 characters.	
`newpassw`	I	A128	User ID's new password. **Note:** For compatibility reasons `newpassw` is defined with `BY VALUE RESULT` to support existing callers which pass an A32 field for the `newpassw`. The sample `USR2071P` provided in the library `SYSEXT` supports up to 32 characters.	
`rc`	O	N04	Return value:	
			0	OK
			1	invalid function code
			9999	EntireX Broker error (see `message`)
`message`	O	A80	Message text returned by EntireX Broker.	

USR2072N

To make use of `USR2072N`

1 Copy the subprogram `USR2072N` and optionally program `USR2072P` from library `SYSEXT` to the library `SYSTEM` or to the steplib library or to any application in the server environment.

2 Using a `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description
password	I	A128	User ID's password. **Note:** For compatibility reasons password is defined with BY VALUE RESULT to support existing callers which pass an A32 field for the password. The sample USR2072P provided in the library SYSEXT supports up to 32 characters.

USR4371N

The application programming interface USR4371N is provided in the library SYSEXT. It is used to specify the user ID and the ETID that are passed to the Natural RPC server.

To make use of USR4371N

1 Copy the subprogram USR4371N and the program USR4371P from library SYSEXT to the library SYSTEM in the system file FNAT in the client environment. For more information, see *Using a Natural API* in the *SYSEXT Utility* documentation.

2 Using a `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description
USERID	I	A08	User ID to be used.
ETID	I	A08	ETID to be used.

3 In the calling program on the client side, specify the following statement:

```
FETCH RETURN 'USR4371P' USERID ETID
```

Alternatively, you can invoke USR4371P from the command line, and enter the user ID and ETID in the displayed window.

For more information, see the text object USR4371T in the system library SYSEXT.

USR9171N

The application programming interface USR9171N is provided in the library SYSEXT. You can use it to specify the user ID and passphrase that are passed to the Natural RPC server.

To make use of `USR9171N`

1 Copy the subprogram `USR9171N` and the program `USR9171P` from library `SYSEXT` to the library `SYSTEM` in the system file `FNAT` in the server environment. For more information, see *Using a Natural API* in the *SYSEXT Utility* documentation.

2 Using a `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description	
`USERID`	I	A32	User ID to be used.	
`PASSWORD_PHRASE`	I	A128	Case-sensitive passphrase to validate the user ID. This passphrase is not validated on the client side.	
`MSG-TEXT`	O	A79	Message text if PRC is not 0.	
`PRC`	O	N04	**Return Value**	
			`0`	OK
			`4`	Internal Error
			`2x`	Error in write handling
			`3x`	Error in read handling
			`9999`	Invalid function code

3 In the calling program on the client side, specify the following statement:

```
FETCH RETURN 'USR9171P' USERID PASSWORD_PHRASE MSG-TEXT PRC
```

You may alternatively invoke `USR9171P` from the command line and enter `USERID` and `PASSWORD_PHRASE` in the displayed window.

For a more detailed description, see the `USR9171T` text object in the system library `SYSEXT`.

Server Side

If Natural Security is installed on the server side and `AUTO=ON` is not specified, a Natural logon with user ID and password is required. It is recommended to use the Natural profile parameter `STACK` to pass the Natural system command `LOGON`. If `AUTO=ON` is specified the contents of `*INIT-USER` is used for an internal logon as usual.

To enforce the Logon option - that is, if you want a server to accept only requests from clients where the Logon option is set - set the profile parameter `LOGONRQ` to `ON` for the server. If the Logon option is not enforced, client request without logon data are accepted and executed in the server library or one of its steplibs. This allows you to provide public as well as secured services.

If the client passes logon data, the user ID and password from the client are verified against the corresponding user security profile on the server, and the logon to the requested library and the

execution of the subprogram are performed according to the corresponding Natural Security library and user profile definitions on the server.

After the execution of the subprogram, the library used before the `CALLNAT` request is updated again on the server. In the case of a conversational RPC, the first `CALLNAT` request within the conversation sets the library ID on the server, and the `CLOSE CONVERSATION` statement resets the library ID on the server to the one used before the conversation was opened.

As part of the *Natural RPC Restrictions* in the library profiles of Natural Security, a server session option `Close all databases` is provided. It causes all databases which have been opened by remote subprograms contained in the library to be closed when a Natural logon/logoff to/from the libraries is performed. This means that each client uses its own database session.

If the `Close all databases` option is set, it is also possible to use a client specific ETID for all Adabas accesses which are executed by the server for this client. In this case, you should start the Natural RPC server with `ETID=OFF` and define an appropriate ETID in the user profile for each client that needs an ETID, for example, by specifying the ETID `*USER`. Please note that in this case two clients with the same name cannot issue two concurrent requests with Adabas calls.

Changing Password

It is possible to change the Natural Security password on the Natural RPC server via a Natural RPC service request. For this purpose, the application programming interface `USR2074N` is provided in the library `SYSEXT`.

To make use of `USR2074N`

1. Copy the subprogram `USR2074N`, and optionally program `USR2074P`, from library `SYSEXT` to the library `SYSTEM` or to the steplib library or to any application in the server environment.
2. Using a `DEFINE DATA` statement, specify the following parameters:

Parameter	I/O	Format	Description	
`USERID`	I	A08	User ID to be used.	
`PASSWORD`	I	A08	Password to validate the user ID. This password is not validated on the client side.	
`NEWPASSWORD`	I	A08	New password for the user ID. This password is not validated on the client side.	
`NODE-NAME`	I	A192	Name of the server node to be addressed.	
`SERVER-NAME`	I	A32	Name of the server to be addressed.	
`PROTOCOL`	I	A1	The transport protocol to address the server node. Valid value:	
			`B`	EntireX Broker.
`RC`	O	I2	Return value:	
			`0`	OK, `MESSAGE` contains a confirmation message.

Parameter	I/O	Format	Description	
			1	Error from RPC or server node, MESSAGE contains the error message.
			2	Error from the interface, MESSAGE contains the error message.
			3	Natural Security error, MESSAGE# contains the Natural error number and MESSAGE contains the corresponding message text.
MESSAGE#	O	N4	Message number returned.	
MESSAGE	O	A80	Message text returned.	

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR2074N' user-id password newpassword node-name server-name protocol
rc message# message
```

You may alternatively use program USR2074P from library SYSEXT. Invoke USR2074P from the command line and enter the required data in the displayed window. In this case, all input except for the passwords are converted into upper case. For the passwords, you have the option to enter them in mixed case or not.

Impersonation (z/OS Batch Mode)

- Purpose of Impersonation in z/OS Batch Mode
- Steps to Activate Impersonation in z/OS Batch Mode (Server Side)
- Steps to Use Impersonation in z/OS Batch Mode (Client Side)
- Rules for Impersonation in z/OS Batch Mode

Purpose of Impersonation in z/OS Batch Mode

Impersonation is an optional feature on the Natural RPC server side and is only available if the Natural RPC server runs under Natural Security. The impersonation feature is controlled by the *Security Profiles for Natural RPC Servers*. See the field Impersonation described under the heading *Components of an RPC Server Profile* in the section *Protecting Natural RPC Servers and Services* in the *Natural Security* documentation.

Impersonation in z/OS batch mode requires the use of the Natural RPC server front-end under z/OS and uses the SAF interface provided by z/OS.

If impersonation is active for the Natural RPC server, a client request that uses the **Logon option** is from the perspective of the operating system executed under the user ID that the client passes in the LOGON data (called Natural RPC user ID). Impersonation assumes that access to the operating system on which a Natural RPC server is running is controlled by an SAF-compliant external se-

curity system. User authentication (verification of the Natural RPC user ID and password or passphrase) is performed by this external security system. After successful authentication, the user's identity is established for the operating system (that is, an ACEE is created and linked to the TCB under which the current client request is executed). Any subsequent authorization checks will be performed based on this identity. This means that all accesses to resources that are controlled by the SAF compliant external security system are authorized for this identity. This applies especially to accesses to work files and to databases.

Impersonation does not turn off Natural Security. After successful authentication of the user's identity by the external security system, a Natural Security logon takes place using the same `LOGON` data but without password or passphrase verification.

To start a Natural RPC server using impersonation, see ***Starting a Natural Server Using the RPC Server Front-End*** in ***Starting a Natural RPC Server***.

Note: Without impersonation, a client request that uses the Logon option is from the perspective of the operating system executed with the user ID under which the Natural RPC server has been started.

Steps to Activate Impersonation in z/OS Batch Mode (Server Side)

1. **Install RPC server front-end**

   Proceed as described in the corresponding steps of the Natural for z/OS installation documentation; see *Installing Natural on z/OS*.

2. **Link Natural z/OS batch nucleus with Db2 interface** `DSNRLI`

   This step applies to Natural for Db2 users only.

3. **Use reentrant Adabas batch link routine** `ADALNKR` **instead of** `ADALNK`

   Refer to ***Considerations for z/OS Natural RPC Servers with Replicas*** in ***Starting a Natural RPC Server***.

4. **Use EntireX Broker Stub** `BROKER` **instead of** `NATETB23`

   See ***Provide Access to the EntireX Broker Stub on z/OS*** in ***Setting Up a Natural RPC Environment***.

5. **Define all required RPC server-specific Natural profile parameters**

   Refer to ***Set the RPC Server-Specific Natural Parameters*** in ***Setting Up a Natural RPC Environment***. The parameters are either defined in the Natural parameter module or in the `CMPRMIN` dataset. The parameter `PARM=` of the `JCL EXEC` statement is not used to provide Natural profile parameters.

6. **Define an RPC server profile in Natural Security**

   Define an RPC server profile in Natural Security (NSC) for the server name that is used by the RPC server (`SRVNAME`) and activate the impersonation.

   Refer to *Security Profiles for Natural RPC Servers* in *Protecting Natural RPC Servers and Services* of the *Natural Security* documentation.

7. **Check SAF definitions**

   (This step applies to Natural for Db2 users only.)

   If the SAF resource class `DSNR` is active, you must check whether you need the following SAF definitions:

   ```
   RDEFINE DSNR (subsys.RRSAF) OWNER(DB2owner)

   PERMIT subsys.RRSAF CLASS(DSNR) ID(DB2group) ACCESS(READ)
   ```

   where *`subsys`* is your Db2 subsystem ID.

   Each user who wants to access Db2 must be a member of group *`DB2group`*.

   For further information, refer to the relevant Db2 documentation of IBM.

8. **Create user exit `NATRPC02`**

   (This step applies to Natural for Db2 users only.)

   Create the Natural RPC user exit `NATRPC02` with a call to `NATPLAN` to set the required Db2 plan.

   Make sure that you use a `NATPLAN` of your current Natural for Db2 version.

   Sample `NATRPC02`:

   ```
   DEFINE DATA PARAMETER
     1 SUBPROGRAM (A8) BY VALUE END-DEFINE
   FETCH RETURN 'NATPLAN' 'planname'
   ```

9. **Start Natural RPC server front-end**

   Start the Natural RPC server front-end.

   Refer to ***Starting a Natural RPC Server Using the RPC Server Front-End*** in ***Starting a Natural RPC Server***.

   Make sure you have added all required load libraries to your `STEPLIB` concatenation. You will especially need the following:

   - Natural load library

- EntireX load library
- Adabas load library (if you use the Adabas link routine `ADAUSER`)
- Db2 load library (if you want to access Db2)

The impersonation is successfully activated if you see the following messages:

- In the job log:

```
RPC0010 Authorized environment for impersonation established
```

- In the RPC trace file:

```
M *** Server is running under NSC with impersonation
```

Steps to Use Impersonation in z/OS Batch Mode (Client Side)

The client must send logon data together with the RPC request as it is already done for a standard Natural Security (NSC) protected Natural RPC server. In contrast to a standard Natural RPC server, the user ID must also be a valid SAF user ID and the password or passphrase must be the corresponding SAF password or passphrase. User ID and password or passphrase are validated by the Natural RPC server against the external security system on the z/OS system under which the server is executing. After successful authentication of the client's identity by the external security system, the user ID is validated by NSC according to the defined rules. The password or passphrase is ignored. Therefore, it is not required to set the NSC password to your SAF password.

When the field `Impersonation` described in the section *Components of an RPC Server Profile* in the *Natural Security* documentation is set to `A`, no password or passphrase is used to authenticate the client against the external security system. This setting may be appropriate if the client has already been authenticated by the EntireX Broker.

Depending on the kind of client, the logon data are set differently:

Natural Clients

1. Turn on the logon option in the *Service Directory Maintenance* function or in the profile parameter `DFS`.

   Alternatively, you can use the `USR2007N` to turn it on.

   Refer to ***Using the Logon Option*** in ***Operating a Natural RPC Environment***.

2. Set the SAF user ID and the SAF password or passphrase, using the application programming interface `USR1071N` for password or the application programming interface `USR9171N` for passphrase.

   If your client runs under Natural Security (NSC) and the user ID and password of NSC are identical to the SAF user ID and the SAF password, then `USR1071N` is not required.

EntireX RPC Clients

1. Turn on the Natural logon option according to your application environment.
2. Set the RPC user ID and the RPC password to the SAF user ID and SAF password according to your application environment.

Rules for Impersonation in z/OS Batch Mode

- Impersonation takes place at the start of each non-conversational `CALLNAT` and at the start of each conversation.
- The authentication of the Natural RPC user ID and password or passphrase is performed by the external security system. The password on the `FSEC` system file is not used.
- After successful authentication, the Natural RPC user ID is established for the operating system (user is impersonated).
- After successful impersonation:
  1. A Natural security logon is performed for the Natural RPC user ID without password or passphrase check.
  2. All work files with a `DDNAME` that does not start with CM are opened with the Natural RPC user ID.
  3. All Adabas databases are opened with the Natural RPC user ID (applies to Adabas external security only).
  4. If an `ETID` is specified in the NSC user profile, this `ETID` is used in the Adabas open request.
  5. The Db2 connection is opened with the Natural RPC user ID (applies to Natural for Db2 users only).
- At the end of each non-conversational `CALLNAT` and at the end of each conversation, the Natural RPC user ID is logged off from the operating system.
- After log off:
  1. All work files with a `DDNAME` that does not start with `CM` are closed.
  2. All Adabas databases are closed.

Impersonation (CICS)

The following topics are covered below:

- Purpose of Impersonation under CICS
- Steps to Activate Impersonation under CICS (Server Side)
- Steps to Use Impersonation under CICS (Client Side)
- Rules for Impersonation under CICS

- RPCSFEX1 - User Exit for Impersonation under CICS

Purpose of Impersonation under CICS

Impersonation is an optional feature on the Natural RPC server side and is only available if the Natural RPC server runs under Natural Security. The impersonation feature is controlled by the *Security Profiles for Natural RPC Servers*. See the field `Impersonation` described under the heading *Components of an RPC Server Profile* in the section *Protecting Natural RPC Servers and Services* in the *Natural Security* documentation.

Impersonation under CICS requires the use of the Natural RPC server front-end under CICS and uses the interface provided by CICS.

If impersonation is active for the Natural RPC server, a client request that uses the **Logon Option** is from the perspective of CICS executed under the user ID that the client passes in the `LOGON` data (called Natural RPC user ID). Impersonation under CICS uses the CICS option to start a CICS task under a given user ID. After a client request has arrived the Natural RPC server front-end starts a new CICS task using the `USERID()` option of the `EXEC CICS START TRANSID()` command, where `USERID` is the Natural RPC user ID. The User authentication (verification of the Natural RPC user ID and password or passphrase) is performed by CICS, typically by using the underlying external security system. After successful authentication, the user's identity is established for the CICS task. Any subsequent authorization checks will be performed based on this identity. This means that all accesses to resources that are controlled by CICS are authorized for this identity. This applies especially to accesses to CICS resources and to databases. By default, the newly established CICS task runs with the same CICS transaction ID with which you started the RPC server front-end. You can use the `RPCSFEX1` **user exit** to set a request-specific transaction ID.

Impersonation does not turn off Natural Security. After successful authentication of the user's identity by CICS, a Natural Security logon takes place using the same `LOGON` data without password or passphrase verification.

To start a Natural RPC server using impersonation, see **Starting a Natural Server Using the RPC Server Front-End** (CICS only) in **Starting a Natural RPC Server**.

Note: Without impersonation, a client request that uses the Logon option is from the perspective of the operating system executed with the user ID with which the Natural RPC server has been started.

Steps to Activate Impersonation under CICS (Server Side)

1. **Install the RPC server front-end under CICS**

   Proceed as described in the corresponding steps of the Natural for z/OS installation documentation; see *Installing the Natural CICS Interface on z/OS*.

2. **Install the Adabas link routine for Adabas external security**

   For further information, refer to the relevant Adabas documentation (applies to Adabas external security users only).

3. **Use EntireX Broker Stub `CICSETB` instead of `NATETB23`**

   See ***Providing Access to the EntireX Broker Stub on z/OS*** in ***Setting Up a Natural RPC Environment***.

4. **Define all required RPC server-specific Natural profile parameters**

   Refer to ***Set the RPC Server-Specific Natural Parameters*** in ***Setting Up a Natural RPC Environment***.

   The parameters are either defined in the Natural parameter module or together with the transaction ID.

5. **Define an RPC Server Profile in Natural Security**

   Define an RPC Server Profile in Natural Security (NSC) for the server name that is used by the RPC server (`SRVNAME`) and activate the impersonation.

   Refer to*Security Profiles for Natural RPC Servers* in *Protecting Natural RPC Servers and Services* of the *Natural Security* documentation.

6. **If CICS startup parameter `XUSER=YES`**

   If the CICS startup parameter `XUSER=YES` is specified you must define surrogate users for each client user:

   ```
   RDEFINE SURROGATE userid1.DFHSTART UACC(NONE) OWNER(userid1) PERMIT
   userid1.DFHSTART CLASS(SURROGATE) ID(userid2) ACCESS(READ)
   ```

   where

   `userid1` is the user ID of the client,

   `userid2` is the user ID under which the Natural RPC server front-end is started.

   For further information, refer to the relevant CICS documentation of IBM.

7. **Define a CICS `PROGRAM` entry for the RPC server front-end**

   Refer to the corresponding step in *Installing the Natural CICS Interface on z/OS*.

8. Define a CICS `TRANSACTION` entry for the transaction ID that invokes the RPC server front-end.

   Refer to the corresponding step in *Installing the Natural CICS Interface on z/OS*.

9. **Define a `DB2TRAN` and `DB2ENTRY` entry**

   (This step applies to Natural for Db2 users only.)

   Define a `DB2TRAN` and `DB2ENTRY` entry for the transaction ID that invokes the RPC server front-end.

10. **Start the Roll Server**

    Start the *Roll Server* for the subsystem used by the Natural RPC server.

    (This step applies only if the `NCMDIR` macro parameter `ROLLSRV` is set to `YES`.)

11. **Start the Natural RPC server front-end under CICS**

    Refer to ***Starting a Natural RPC Server Using the RPC Server Front-End (CICS only)*** in ***Starting a Natural RPC Server***.

    The impersonation is successfully activated if you see the following message in the RPC trace file:

    ```
    M *** Server is running under NSC with impersonation
    ```

Steps to Use Impersonation under CICS (Client Side)

The client must send logon data together with the RPC request as it is already done for a standard Natural Security (NSC) protected Natural RPC server. In contrast to a standard Natural RPC server, the user ID must also be a valid CICS user ID and the password or passphrase must be the corresponding password or passphrase of the external security system. User ID and password or passphrase are validated by CICS against the external security system on the z/OS system under which CICS is executing. After successful authentication of the client's identity by the external security system, the user ID is validated by Natural Security according to the defined rules. The password or passphrase is ignored. Therefore, it is not required to set the NSC password to your SAF password.

When the field `Impersonation` described in the section *Components of an RPC Server Profile* in the *Natural Security* documentation is set to `A`, no password or passphrase is used to validate the client against the external security system. This setting may be appropriate if the client has already been authenticated by the EntireX Broker.

Depending on the kind of client, the logon data are set differently:

Natural Clients

1. **Turn on the logon option**

   Turn on the logon option in the *Service Directory Maintenance* function or in the profile parameter `DFS`.

   Alternatively, you can use the application programming interface `USR2007N` to turn it on.

   Refer to ***Using the Logon Option*** in ***Operating a Natural RPC Environment***.

2. **Set user ID and password or passphrase**

   Set the user ID and the password, using application programming interface `USR1071P`. To set the passphrase use application programming interface `USR9171P`.

   If your client runs under Natural Security (NSC) and the user ID and password of NSC are identical to the user ID and password on the server side, then `USR1071P` is not required.

EntireX RPC Clients

1. **Turn on the Natural logon option**

   Turn on the Natural logon option according to your application environment.

2. **Set RPC user ID and password**

   Set the RPC user ID and the RPC password according to your application environment.

Rules for Impersonation under CICS

- Impersonation takes place at the start of each non-conversational `CALLNAT` and at the start of each conversation.
- The optional `RPCSFEX1` **user exit** is called to set a request-specific CICS transaction ID.
- The authentication of the Natural RPC user ID and password or passphrase are performed by CICS. The password on the FSEC system file is not used.
- After successful authentication, the Natural RPC user ID is established for CICS (user is impersonated).
- After successful impersonation:
  1. A Natural security logon is performed for the Natural RPC user ID without password or passphrase check.
  2. All CICS resources are accessed with the Natural RPC user ID.
  3. All Adabas databases are opened with the Natural RPC user ID (applies to Adabas external security only).
  4. If an `ETID` is specified in the NSC user profile, this `ETID` is used in the Adabas open request.

5. The Db2 connection is opened with the Natural RPC user ID (applies to Natural for Db2 users only).

- At the end of each non-conversational `CALLNAT` and at the end of each conversation, the Natural RPC user ID is logged off from CICS.
- After log off:
  1. All CICS resources are closed.
  2. All Adabas databases are closed.
  3. The connection to Db2 is closed (applies to Natural for Db2 users only).

RPCSFEX1 - User Exit for Impersonation under CICS

By default, the CICS transaction ID with which the impersonated RPC request is executed is the same as the CICS transaction ID used to start the RPC server front-end. You can set a request-specific transaction ID with the `RPCSFEX1` user exit.

The `RPCSFEX1` user exit is called when the logon data of a client is evaluated after an RPC request has been received. The logon data is then used to set the CICS transaction ID under which the impersonated RPC request executes.

You must link `RPCSFEX1` to the RPC server front-end for CICS and call the user exit by using the following standard conventions:

`RPCSFEX1` is called using standard conventions:

Register	Contents
15	Entry address of `NATSFEX1`
14	Return address of Natural RPC server front-end
13	Address of a save area of 18 words
1	Address of a parameter list

The parameter list contains the following addresses:

Address	Parameter	I/O	Natural Data Format/Length
1	CICS transaction ID	I/O	A04
2	EntireX user ID of client	I	A32
3	RPC user ID of client	I	A8
4	RPC password of client	I	A8
5	Name of the Natural library where to execute the RPC request on the Natural RPC server	I	A8
6	Name of the subprogram to be executed on the Natural RPC server	I	A8

You can only modify the CICS transaction ID with the user exit.

Using Natural RPC with EntireX Security

Natural RPC fully supports EntireX Security on the client side and on the server side.

- EntireX Security on the Client Side
- EntireX Security on the Server Side

EntireX Security on the Client Side

To logon to and logoff from the EntireX Broker, the Natural Application Programming Interface USR2071N is provided. To logon to EntireX Broker, you use the logon function of USR2071N and pass your user ID and password to the selected EntireX Broker. After a successful logon, the security token returned is saved by Natural and passed to the EntireX Broker on each subsequent call. The Logon option is fully transparent to the Natural application.

If EntireX Security is installed or if AUTOLOGON=NO is specified in the EntireX Broker attribute file, you must invoke USR2071N with the logon function before the very first remote CALLNAT execution.

You are recommended to invoke USR2071N with the logoff function as soon as you no longer intend to use a remote CALLNAT.

To make use of USR2071N

1 Copy the subprogram USR2071N from library SYSEXT to the library SYSTEM or to the steplib library or to any application in the server environment.

2 Using a DEFINE DATA statement, specify the following parameters:

Parameter	I/O	Format	Description	
function	I	A08	Function code; possible values are:	
			LOGON	Logon to EntireX Broker
			LOGOFF	Logoff from EntireX Broker
broker-id	I	A192	Broker ID. **Note:** For compatibility reasons *broker-id* is defined with BY VALUE RESULT to support existing callers which pass an A8 or A32 field for the *broker-id*. The sample USR2071P provided in the library SYSEXT supports up to 32 characters.	
user-id	I	A32	User ID.	
password	I	A32	User ID's password.	
newpassw	I	A32	User ID's new password.	

Parameter	I/O	Format	Description	
rc	O	N04	Return value:	
			0	OK
			1	invalid function code
			9999	EntireX Broker error (see *message*)
message	O	A80	Message text returned by EntireX Broker.	

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR2071N' function broker-id user-id password newpassword rc message
```

See also the *Syntax Description* of the CALLNAT statement.

You may alternatively invoke USR2071P from the command line and enter User ID and password in the displayed window. In this case, all input except for the passwords is converted into upper case. For the passwords, you have the option to enter them in mixed case. On Linux and Windows, you also have the option to enter the input parameters User ID and Broker ID in mixed case.

Functionality:

LOGON	An EntireX Broker LOGON function is executed to the named *broker-id* with the *user-id* and the *password* passed. After a successful LOGON call, the client can communicate with the EntireX Broker *broker-id* as usual. With *newpassw* the client user can change her/his password via the EntireX Security features. **Notes:** ■ If a successful logon has been performed, the user ID used in this LOGON will be passed to the named EntireX Broker on all subsequent remote procedure CALLNATs which are routed via this EntireX Broker. Without an explicit LOGON, the current contents of system variable *USER is used. The same applies if you have issued a LOGON to EntireX Broker 1, but your remote procedure CALLNAT is routed via EntireX Broker 2. ■ It is possible to concurrently log on to multiple EntireX Brokers. For each LOGON, a different user ID may be used. ■ The user ID used for the LOGON to the EntireX Broker may be different from the Natural user ID under which the client application runs. ■ An internal re-logon is done after an EntireX Broker timeout has occurred, if the original LOGON was done without a password (the password used in the LOGON is not saved). If no internal re-logon is possible after a timeout has occurred, the client has to explicitly reissue the LOGON. ■ At the end of the Natural session, an implicit LOGOFF is executed to all EntireX Brokers to which a logon has been performed.

LOGOFF	An EntireX Broker LOGOFF function is executed to the *broker-id* named.

Special Considerations when the Client Request is Executed on the Server Side:

If an RPC client request is executed on the Natural RPC server side, a logon to the EntireX Broker, using the Application Programming Interface USR2071N, must also be performed before executing the RPC client request. The logon data of the Natural RPC server itself are not used for RPC client requests.

If the RPC client request is sent to the same EntireX Broker where the Natural RPC server is registered, the user ID must be different from the value of the Natural profile parameter SRVUSER.

EntireX Security on the Server Side

If the value of profile parameter ACIVERS is 2 or higher, the server will log on to the EntireX Broker at the session start using the LOGON function. The user ID is the same as the user ID defined by SRVUSER.

If EntireX Security has been installed and if the EntireX trusted user ID feature is not available, there are two alternative ways to specify the required password:

- Setting SRVUSER='*NSC'
- Using Application Programming Interface USR2072N to Specify a Password

These alternatives are described below.

Setting SRVUSER='*NSC'

If Natural Security is installed on the server, you can set the profile parameter SRVUSER to '*NSC' to specify that the current Natural Security user ID which was used when the server was started is used for the LOGON in conjunction with the accompanying Natural Security password. In this case, the value set for ACIVERS must be at least 4.

Note:

The password defined with the application programming interface USR2072N is ignored.

Using Application Programming Interface USR2072N to Specify a Password

The Application Programming Interface USR2072N enables you to specify a password which is used for the LOGON in conjunction with profile parameter SRVUSER.

To make use of USR2072N

1 Copy the subprogram USR2072N and optionally program USR2072P from library SYSEXT to the library SYSTEM or to the steplib library or to any application in the server environment.

2 Using a DEFINE DATA statement, specify the following parameter:

Parameter	I/O	Format	Description
password	I	A32	User ID's password.

3 In the calling program on the client side, specify the following statement:

```
CALLNAT 'USR2072' password
```

See also the *Syntax Description* of the CALLNAT statement.

4 The calling program must be executed before the Natural RPC server has started its initialization. To accomplish this, put the name of the calling program on the Natural stack when starting the server. For this purpose, you may also use the program USR2072P from library SYSEXT. In this case, the password is converted into upper case by default. You have the option to enter the password in mixed case by passing the mixed case option Y as second parameter.

```
STACK=(LOGON server-library;USR2072P password [Y])
```

13 EntireX Broker Support

- Security 142
- Logging and Accounting 142

Security

Natural RPC client and Natural RPC server support EntireX Security.

If a Natural RPC client or a Natural RPC server is started with `ACIVERS=8`, EntireX Security without stub exits (z/OS only) and without `SECUEXIT` (all platforms) are supported. In this case, Natural issues a `KERNELVERS` call before any other broker call to get the current setting of the ACI field `KERNELSECURITY`. This `KERNELSECURITY` setting is passed with all subsequent broker calls. This feature also allows a Natural RPC client to access a secured and non-secured EntireX Broker within the same Natural session.

Logging and Accounting

Natural RPC client and Natural RPC server support logging and accounting of the RPC program and the RPC library within the EntireX Broker.

The Natural RPC client provides the name of the subprogram that is to be executed and the name of the library from which the subprogram is to be executed to the EntireX Broker.

The Natural RPC server returns the name of the subprogram that has been executed and the name of the library from which the subprogram has actually been executed.

14 API for Providing an RPC Context from the Natural Client Side

- RPC-CNTX 144

This chapter presents the API RPC-CNTX to provide an RPC context from the Natural client side.

RPC-CNTX

The API RPC-CNTX can be used for providing a context for RPC calls as it is done in test programs generated by the *webMethods EntireX* Natural Wrapper.

RPC-CNTX combines the functionality of several APIs and is available in library `SYSTEM`. There is no need for extra preparations such as setting a STEPLIB or to copy APIs from SYSEXT to user libraries.

To make use of RPC-CNTX

1 Specify a `DEFINE DATA` statement

where level `01` is defined by `01 RPC-CNTX-AREA`, and specify the following parameters on level `02`:

Parameter	Format	I/O	Description
FUNCTION	A2	I	`'SL'` Set the RPC context depending on the parameters given and log on to EntireX Broker. `'SC'` Set the RPC context depending on the parameters given. `'GC'` Get RPC context. By default, the required information relates to the default server. `'LO'` Log on to EntireX Broker `'LF'` Log off from EntireX Broker
BROKERID	A dynamic	I/O	Specify the broker name. See the profile parameter `SRVNODE`.
CLASS	A32	I	Specify the server class. For Natural RPC servers, this is `RPC`. See the keyword subparameter SRVNAME of the RPC profile parameter.
SERVER	A32	I/O	Specify the server name. See the profile parameter `SRVNAME`
SERVICE	A32	I	Specify the type of RPC. For Natural RPC servers, this is `CALLNAT`.
COMPRESSLEVEL	A1	I/O	Compression level requires API version 7.
RESERVED	I1	I/O	Reserved for future use.
RPC-LIBRARY	A08	I	

Parameter	Format	I/O	Description
RPC-RELIABLE-STATE	N1	I/O	`0` No reliable RPC (standard RPC execution) `1` Reliable RPC (`AUTO_COMMIT`) `2` Reliable RPC (`Client_COMMIT`)
NAT-LOGON	A11	I/O	
EXX-USERID	A32	I	
EXX-PASSWORD	A32	I	
RPC-USERID	A32	I	
RPC-PASSWORD	A32	I	
ERR-CODE	I4	O	The origin of some error code can be detected by the following identifiers: USR1071N,(USR4371N) 1nnnn USR6304N 2nnnn USR2007N 3nnnn USR4008N 5nnnn USR4009N 8nnnn USR2071N 9nnnn
ERR-TEXT	A dynamic	O	

Note: The most important function codes are `'SL'` and `'LF'`. Function code `'SL'` is a combination of setting the RPC context (`'SC'`) and a logon to EntireX Broker (`'LO'`).

2 In the calling program on the client side, specify the following statement:

```
CALLNAT 'RPC-CNTX' RPC-CNTX-AREA
```

For further information refer to *Syntax Description* in *CALLNAT* in the *Statements* documentation.

The setting of the RPC context is processed as follows:

1. Set the credentials for RPC server by specifying `RPC-USERID` and `RPC-PASSWORD` (API USR1071N).
2. Set the mode for reliable RPC by specifying `RPC-RELIABLE-STATE` (API USR6304N).
3. Set the data for the RPC default server including the **logon option** with a specification for `BROKERID`, `SERVER` and `NAT-LOGON` (API USR2007N).
4. Change the library name on the RPC server for the logon by specifying `RPC-LIBRARY` (API USR4008N). If the name is left blank, the API is not called.

5. Set parameters for EntireX `COMPRESSLEVEL` by using API USR4009N. If no values are specified, the API is not called.
6. For logging on to the EntireX Broker, specify `BROKERID`, `EXX-USERID`, and `EXX-PASSWORD`, which are then passed to USR2071N.

After you have finished your work with the RPC connection, you need to log off from the EntireX Broker using RPC-CNTX with function code `'LF'`.

15 APIs for Use with Natural RPC

The purpose of Natural Application Programming Interfaces (API) is to retrieve or modify information or use services that are not accessible by Natural statements. The following Application Programming Interfaces available in the Natural library SYSEXT are intended for being used with the Natural RPC. Entries with a hyperlink correspond to APIs that are documented in the Natural documentation.

You can also find a description of an API in library SYSEXT under the name USRnnnnT or in the SYSEXT utility with the line command `T` for a required API.

For a list of all RPC-relevant APIs, enter the keyword **RPC** in the **Keyword** field in the SYSEXT utility.

For more information on how to use a Natural API, see *SYSEXT Utility - Natural Application Programming Interfaces*.

API	Purpose
`USR1071N`	Set credentials for RPC server: user ID, password and ticket criteria for Natural RPC. Send logon data to the Natural RPC server if the client has no NSC, or if the client has NSC: for using a different user ID and password on server side. See ***Using Security***. For an enhanced version of this API, refer to API `USR9171N`.
`USR2007N`	Get or set data for RPC default server, including the logon option. Specify a default server address that is to be used each time a remote program cannot be addressed via the service directory. See ***Specifying a Default Server Address within a Natural Session*** in ***Operating a Natural RPC Environment***.

API	Purpose
USR2032N	Support the commit for a CLOSE CONVERSATION statement on the client side. When called, this API will cause an implicit END TRANSACTION at the end of the individual conversation. See *Database Transactions* in *Introducing Natural RPC*, section *Conversational CALLNAT*.
USR2035N	Get or set parameters for SSL support. Set the required SSL parameter string if the Secure Socket Layer (SSL) for the TCP/IP communication to the EntireX Broker is used. See *Using Secure Socket Layer* in *Operating a Natural RPC Environment*.
USR2071N	Support EntireX Security on client side. Logon to EntireX Broker. Has to be called in case of non-Natural Security clients for specifying logon data which are then passed to the server. See *Using Natural RPC with Natural Security* in *Using Security*.
USR2072N	Support EntireX Security on server side. Specify a password which is used for the LOGON in conjunction with profile parameter SRVUSER. See *Server Side* in *Using Security*, section *Using Natural RPC with EntireX Security*.
USR2073N	Ping or terminate a Natural RPC server from within your application. See *Using Application Programming Interface USR2073N* in *Terminating a Natural RPC Server*. For an enhanced version of this API, refer to API USR9175N.
USR2074N	Change the Natural Security password on the RPC server via a Natural RPC service request. See *Change Password* in *Using Security*.
USR2075N	Terminate an EntireX Broker Service from within your application. See *Using Application Programming Interface USR2075N* in *Terminating a Natural RPC Server*. For an enhanced version of this API, refer to USR8208N and USR9176N.
USR2076N	Get or set RPC TIMEOUT value. See *Get or Set RPC TIMEOUT Value Using API USR2076N* in *Operating a Natural RPC Environment*.
USR4008N	On the client side, pass an alternate name of a library to the Natural RPC server to log on to. See *Logging on to a Different Library* in *Operating a Natural RPC Environment*.
USR4009N	Set or get parameters for EntireX in a Natural RPC client or server environment. See *Setting/Getting Parameters for EntireX* in *Operating a Natural RPC Environment*.
USR4010N	On the client side, retrieve the runtime settings of a server. See *Retrieving the Runtime Settings of a Server* in *Operating a Natural RPC Environment*.

API	Purpose
USR4012N	Set application error on RPC server. See *Set Application Error on RPC Server* in *Setting Up a Natural RPC Environment*.
USR4371N	On the client side, set the user ID and ETID for Natural RPC servers which were configured with Impersonation = A (automatic logon). See *Using Natural RPC with Natural Security* in *Using Security*.
USR6304N	Set or get the reliable state for the reliable Natural RPC. See *Reliable RPC on the Natural RPC Client Side* in *Reliable RPC*.
USR6305N	Commit or roll back a unit of work that has been created with `CLIENT_COMMIT`. This API is required if the reliable RPC state has been set to `client commit`. See *Reliable RPC on the Natural RPC Client Side* in *Reliable RPC*.
USR6306N	Get the status of all reliable RPC messages of the user who is currently logged on to the EntireX Broker. See *Reliable RPC on the Natural RPC Server Side* in *Reliable RPC*.
USR8208N	Terminate an EntireX Broker service. Pings or terminates an RPC server or terminates an EntireX Broker service. Displays the EntireX Broker version and other information from the EntireX Broker Command and Information Service (CIS). Enhanced version of USR2075N with the use of the more performant 'shutdown service' command on EntireX. See *Using Application Programming Interface USR8208N* in *Terminating a Natural RPC Server*. For an enhanced version of this API, refer to API USR9176N.
USR8213N	Shutdown server with sequence number on EntireX Broker. Retrieves a list of RPC server replica registered on a specified EntireX Broker. The interface then sends a shutdown request with the sequence number to the EntireX Broker. See *Using Application Programming Interface USR8213N* in *Terminating a Natural RPC Server*. For an enhanced version of this API, refer to API USR9177N.
USR8216N	Maintain the RPC service directory. See *Service Directory Maintenance* in *SYSRPC Utility*.
USR8220N	Trigger the server-side termination of the Natural RPC server that is running this API. See *Using Application Programming Interface USR8220N* in *Terminating a Natural RPC Server*.
USR8225N	Maintain message ID and correlation ID of EntireX. See *Maintain Message ID and Correlation ID of EntireX* in *Operating a Natural RPC Environment*.

API	Purpose
USR9171N	Set user's passphrase. Set credentials for RPC server. Enhanced version of USR1071N with the capability of using longer passwords. See ***Using Natural RPC with Natural Security*** in ***Using Security***.
USR9175N	Ping, terminate, or trace a Natural RPC server. Enhanced version of API USR2073N with the capability of using longer passwords and getting or setting the trace level of a Natural RPC server. See ***Using Application Programming Interface USR9175N*** in ***Terminating a Natural RPC Server***.
USR9176N	Terminate an EntireX Broker service. Enhanced version of USR8208N with the capability of using longer passwords. See ***Using Application Programming Interface USR9176N*** in ***Terminating a Natural RPC Server***.
USR9177N	Shutdown server with sequence number on EntireX Broker. Enhanced version of USR8213N with the capability of using longer passwords. See ***Using Application Programming Interface USR9177N*** in ***Terminating a Natural RPC Server***.
USR9204N	Send command to Natural RPC server with sequence number on EntireX Broker. See ***Sending Command to RPC Server*** in ***Operating a Natural RPC Environment***.

Note that the RPC-specific APIs accept all values also in mixed mode. An uppercase translation will take place only if the example program USR*nnnn*P (source object) is used to invoke the corresponding subprogram USR*nnnn*N. Exception: All USR*nnnn*P programs that deal with passwords provide an option to enter the passwords in mixed case mode.