



# natural

---

Natural RPC (Remote Procedure Call) | Version 5.1.1 for Mainframes | Natural RPC (Remote Procedure Call)

This document applies to Natural RPC (Remote Procedure Call) Version 5.1.1 for Mainframes and to all subsequent releases.

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

© Copyright Software AG 1979 - 2003.  
All rights reserved.

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

# Table of Contents

<b>Natural RPC - Overview</b>	1
Natural RPC - Overview	1
<b>Principles of Natural RPC</b>	3
Principles of Natural RPC	3
General Information	3
Purpose	3
Advantages of Natural Remote Procedure Calls	3
Natural RPC Modes of Operation	4
Availability on Various Platforms	4
Support of Non-Natural Environments	5
Prerequisites	5
Natural RPC Operation in Non-Conversational Mode	6
Issuing CALLNATs in an RPC Environment	7
Natural RPC Operation in Conversational Mode	9
General Rules for Local/Remote Subprogram Execution	9
Conversational versus Non-Conversational Mode	10
General Rules for Use of Conversational/Non-Conversational RPC	11
Possible Disadvantage of Using Conversational RPC	11
Database Transactions	11
Non-conversational CALLNAT	11
Conversational CALLNAT	11
Restrictions and Limitations when Using Natural RPC	12
User Context Transfer	12
System Variable Transfer	12
Parameter Handling in Error Situations	12
Variable Arrays in Subprograms	12
Natural Statement Reactions	13
Location of Conversations	13
Future Restrictions of Statement Usage with RPC	13
<b>Setting Up a Natural RPC Environment</b>	15
Setting Up a Natural RPC Environment	15
Setting Up a Natural Client	15
Setting Up A Natural Server	16
Setting Up an EntireX Broker Access	17
Using TCP/IP as a Transport Method	18
Setting Up an EntireX Broker Environment	18
Starting a Natural Server	20
Starting a Natural Server in a Mainframe Online Environment	20
Starting a Batch Server in a Mainframe Environment	20
Starting a Server in a UNIX Environment	22
Starting a Server in a Windows Environment	22
Considerations for Natural RPC Servers with Replicates	23
Natural RPC Batch Server with NTASKS >1	23
Running a Batch Server with Replicates	23
<b>Operating a Natural RPC Environment</b>	24
Operating a Natural RPC Environment	24
Specifying RPC Server Addresses	24
Using Local Directory Entries	24
Using Remote Directory Entries	25
Specifying a Default Server Address at Natural Startup	25
Specifying a Default Server Address within a Natural Session	25
Using an Alternative Server	26
Using EntireX Location Transparency	27

Stubs and Automatic RPC Execution . . . . .	27
Creating Stub Subprograms . . . . .	28
Working with Automatic Natural RPC Execution . . . . .	28
Modifying RPC Profile Parameters Dynamically . . . . .	28
Executing Server Commands . . . . .	28
Logon to a Server Library . . . . .	29
Using the LOGON Option . . . . .	29
Settings Required on the Client Side . . . . .	30
Settings Required on the Server Side . . . . .	30
Using Natural RPC with Natural Security . . . . .	30
Using Natural RPC with EntireX Security . . . . .	31
Client Side . . . . .	31
Server Side . . . . .	32
Using Compression . . . . .	33
Using Secure Socket Layer . . . . .	34
Using Interface USR2035N . . . . .	34
Monitoring the Status of an RPC Session . . . . .	35
Using the RPCERR Program . . . . .	35
Using the RPCINFO Subprogram . . . . .	35
Using the Server Trace Facility . . . . .	37
Defining the Trace File . . . . .	38
Handling Errors . . . . .	41
Remote Error Handling . . . . .	41
Avoiding Error Message NAT3009 from Server Program . . . . .	41
User Exit NATRPC01 . . . . .	41
Terminating a Natural RPC Server . . . . .	42
Using SYSRPC . . . . .	42
Using Entirex Control Center or EntireX System Management Hub . . . . .	42
User Exit NATRPC99 . . . . .	42
<b>Using a Conversational RPC . . . . .</b>	<b>43</b>
Using a Conversational RPC . . . . .	43
Opening a Conversation . . . . .	43
Closing a Conversation . . . . .	44
Defining a Conversation Context . . . . .	45
Modifying the System Variable *CONVID . . . . .	45
<b>Using a Remote Directory Server - RDS . . . . .</b>	<b>46</b>
Using a Remote Directory Server - RDS . . . . .	46
RDS Principles of Operation . . . . .	46
Using a Remote Directory Server . . . . .	48
Creating an RDS Interface . . . . .	50
Creating a Remote Directory Service Routine . . . . .	52
Remote Directory Service Program RDSSCDIR . . . . .	53

# Natural RPC - Overview

Remote procedure call (RPC) techniques establish a framework for communication between server and client systems that can be located 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.

## Related Documentation:

For instructions on the functions provided to maintain remote procedure calls refer to the Natural SYSRPC Utility documentation.

This document is organized in the following sections:

- Principles of Natural RPC
- Setting up a Natural RPC Environment
- Operating a Natural RPC Environment
- Using a Conversational RPC
- Using a Remote Directory Server (RDS)

## Related Products

EntireX RPC for 3GL, Entire Network, EntireX Broker

## Definition of Terms

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

Term	Explanation
Client Stub	<p>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.</p> <p>The client stub is the local subprogram via which the server subprogram is called. The client stub has the same name and contains the same parameters as the corresponding server subprogram.</p>
EntireX Broker Stub	Interface between the Natural RPC runtime and the EntireX Broker transport layer which exchanges marshalled data between client and server.
NATCLTGS	The name of the Natural subprogram generated with the SYSRPC utility to implement the service directory (see below).
Node Name	<p>The name of the node to which the remote CALLNAT is sent.</p> <p>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.</p>
RPC Parameters	<p>All parameters available to control a Natural RPC as described in the Natural Parameter Reference documentation:</p> <p>The RPC parameters are included in the NTRPC macro (static definition) or are defined with the RPC profile parameter (dynamic definition). See NTRPC Macro and Profile Parameters (RPC - Remote-Procedure-Call Settings).</p>
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 referenced by the RDS profile parameter (see the relevant section in the Natural Parameter Reference documentation).
Server Name	<p>The name of the server on which the CALLNAT is to be executed.</p> <p>In case of communication via EntireX Broker, the server name is the name as defined in the EntireX Broker attribute file, located in the field SERVER.</p>
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.

# Principles of Natural RPC

This section covers the following topics:

- General Information
  - Natural RPC Operation in Non-Conversational Mode
  - Natural RPC Operation in Conversational Mode
  - Conversational versus Non-Conversational Mode
  - Database Transactions
  - Restrictions and Limitations when Using Natural RPC
- 

## General Information

The following topics are covered below:

- Purpose
- Advantages of Natural Remote Procedure Calls
- Natural RPC Modes of Operation
- Availability on Various Platforms
- Support of Non-Natural Environments
- Prerequisites

### 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.

#### Example:

A Natural client program on a Windows computer can issue a CALLNAT against a mainframe server in order to retrieve data from a mainframe database. The same Windows computer can act as a server if a Natural client program running under, for example, UNIX issues a CALLNAT requesting data from this server Natural.

### Advantages of Natural 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 mainframe computer. The following advantages arise from that:

- The end user on the client can use a Natural application with a graphical user interface.
- A large database can be accessed on a mainframe 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 Remote Procedure Call offers the following modes of operation:

- non-conversational mode (in the following texts this mode is meant unless otherwise specified)
- 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 the Natural RPC on various platforms under the following operating systems:

### Mainframe Environments

- OS/390
- VSE/ESA
- VM/CMS
- BS2000/OSD

Natural RPC on mainframes is supported under the following TP monitors:

- Com-plete
- CICS
- IMS/TM
- TSO
- UTM

Also, it is available in batch mode.

### Other Environments

- UNIX
- Windows

On all of these platforms, Natural can act as both client and server.

**Exception:** Under Windows 98 and Windows ME, Natural can only act as client.

## Support of Non-Natural Environments

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 SDK.

### Prerequisites

The Natural RPC interface requires the following products:

- Software AG EntireX Broker (including the stubs).
- Software AG Entire Net-work (if the transport method used by EntireX Broker is Entire Net-work)
- TCP/IP (if the transport method used by EntireX Broker is TCP/IP)
- EntireX SDK for non-Natural programming language support.
- Directory services if the location transparency provided by Software AG EntireX Broker is used.

For the supported versions, refer to Natural and Other Software AG Products in the current Natural release Notes for Mainframes.

## 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:



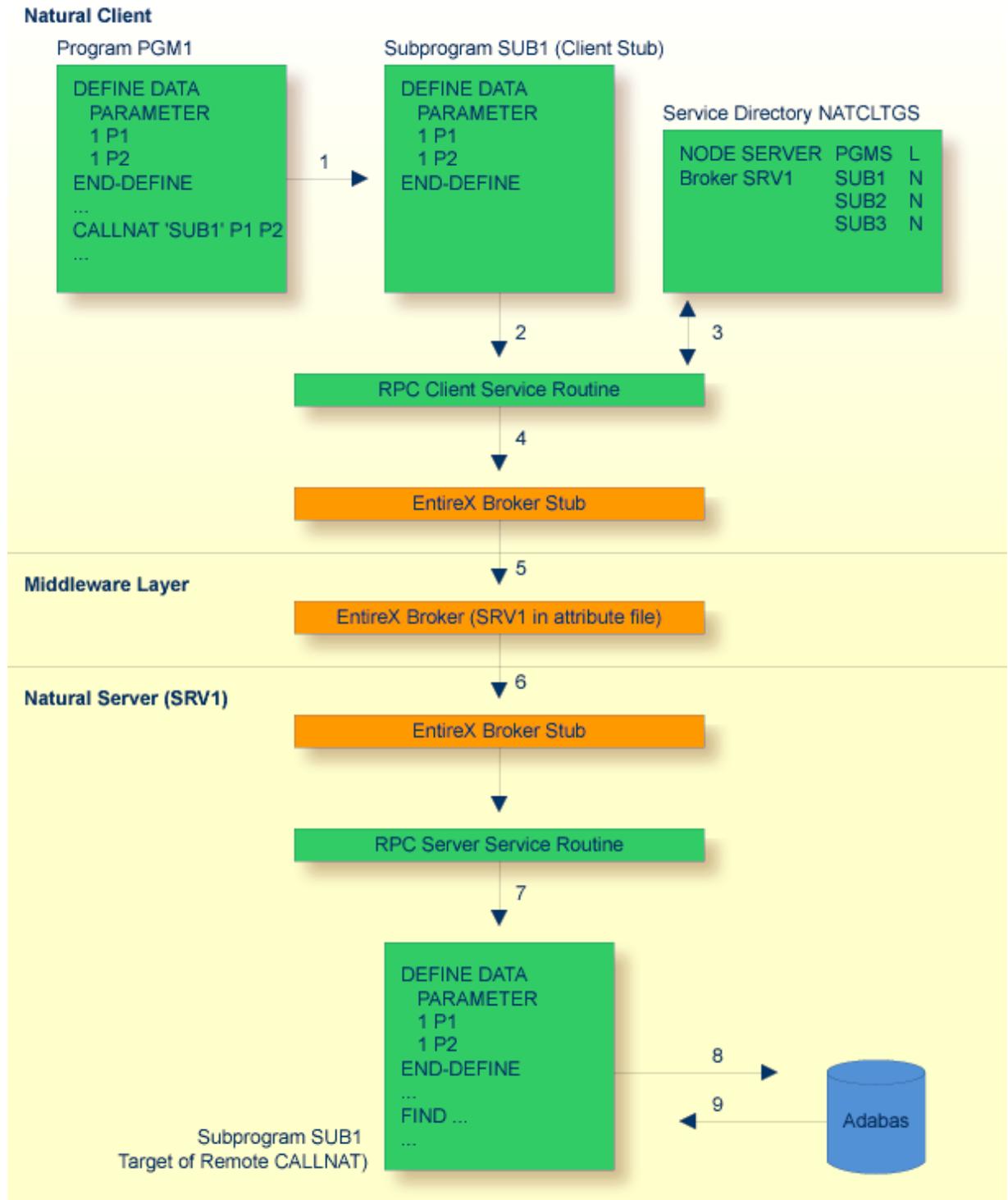
The CALLNAT issued from the Natural Client is routed via a middleware layer to the Natural Server which passes data back to the client.

Usually, the middleware layer consists of the Software AG product 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:



- 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 a "stub" subprogram SUB1 instead. This

client stub subprogram has been created automatically or by using the SYSRPC Utility's Stub Generation (SG) function.

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

If the parameter AUTORPC is set to ON and Natural cannot find the subprogram in the local environment, Natural will interpret this as a remote procedure call and will generate the parameter area dynamically during runtime.

It will also try to find this subprogram in the Service Directory.

For more information on the SYSRPC Stub Generation function, see also *Creating Stub Subprograms*.

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

2. The stub 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 optionally the logon data are passed to a middleware layer.
4. In this example, this middleware layer consists of the Software AG product 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 runtime.  
The server runtime validates the logon data (if present) and performs a logon (if requested).
7. The RPC server runtime 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, SUB1 can issue a FIND statement to the server's Adabas database. SUB1 does not know whether it has been performed by a local or by a remote CALLNAT.
9. Adabas FINDs the data and passes them to SUB1.  
Then, SUB1 returns the Adabas data to the calling server service routine, which passes it back to PGM1 via the middleware layer using 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 process that is exclusively available to the client for the duration of the conversation.

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 process.

During a conversation, you can define and share a data area called context area between the remote subprograms on the server side.

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, i.e. either locally or remotely. You are not allowed to mix up local and remote CALLNATs within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server replicate.

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

## General Rules for Local/Remote Subprogram Execution

### Local Subprogram Execution

If you execute subprograms locally, the following rule applies:

- 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 executed in non-conversational mode.

### Remote Subprogram Execution

If you execute subprograms remotely, the following rule applies:

- A subprogram S1 may call another subprogram S2 which is a member of the conversation.

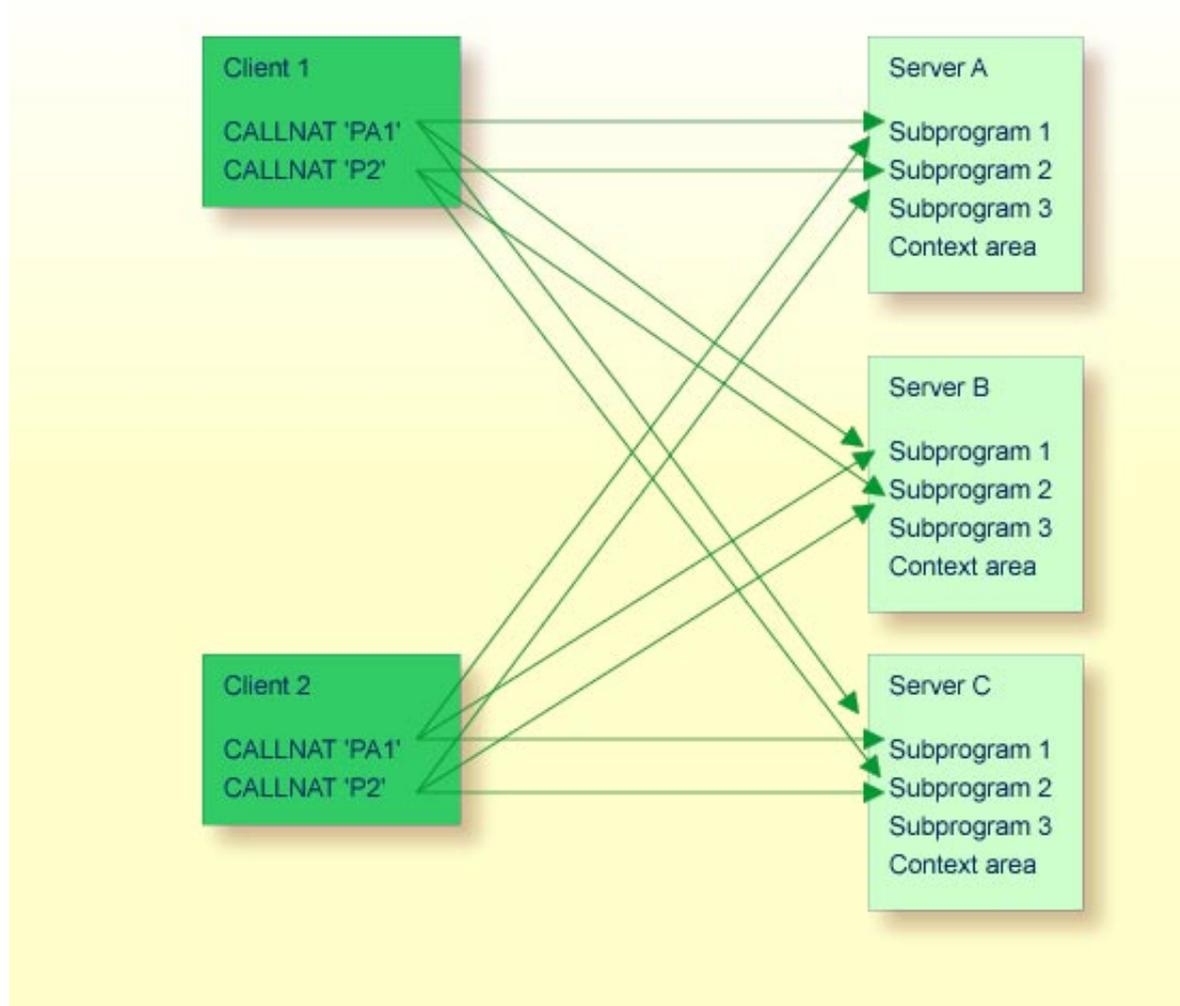
This CALLNAT will be executed in non-conversational mode because it was invoked indirectly. Thus, the subprogram S2 does not have access to the context area.

## 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 'P1' from Client 1 executes Subprogram 1 on server A (P1 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 'P1' to server A, thus overwriting the data from Client 1. If Client 1 then sends a CALLNAT 'P2' (meaning END TRANSACTION), Client 1 thinks its data have been saved correctly while the data from client 2's identical CALLNAT have in fact been saved.

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



CALLNAT 'P1' from Client 1 can access Subprogram 1 on server A, B, or C. CALLNAT 'P1' 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 process context.

You can avoid the potential problems of a non-conversational RPC by defining a more complex RPC transaction. You do this by opening a conversation, for example, on Client 1 comprising CALLNAT 'P1' and CALLNAT 'P2'. Opening such a conversation reserves one entire server replicate (for example, server A) and no other remote CALLNATs may interrupt this conversation on this server until the conversation is closed.

## General Rules for Use of Conversational/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 process 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 replicates.

## Possible Disadvantage of Using Conversational RPC

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

## Database Transactions

The database transactions on the client and server side run independent of each other. That is, an END TRANSACTION or BACKOUT TRANSACTION executed on the server side does not effect 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
- Conversational CALLNAT

### 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.

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

## Restrictions and Limitations when Using Natural RPC

When executing a subprogram by using the Natural RPC facility, several differences to local execution apply.

- User Context Transfer
- System Variable Transfer
- Parameter Handling in Error Situations
- Variable Arrays in Subprograms
- Natural Statement Reactions
- Location of Conversations
- Future Restrictions of Statement Usage with RPC

### 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.

### 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:VŠ notation), you should not use a stub to call this subprogram. As a stub only supports array definitions with a fixed number of occurrences, you cannot vary the number of occurrences from call to call.

## Natural Statement Reactions

Several Natural statements may react in a different way, for example:

Statement	Description
OPEN/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 at the server side only, also for subsequent executions by other users.
SET CONTROL, SET GLOBALS, SET KEY, SET TIME, SET WINDOW	No settings are returned to the caller.
STACK	All stack data are released after execution.
STOP, TERMINATE	These statements do not stop the client session.

## Location of Conversations

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

## Future Restrictions of Statement Usage with RPC

### Current State

With the current Natural version, the use of the following statements in conjunction with 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. <b>Exception:</b> 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).

### Future State

The use of the statements FETCH, INPUT and RUN in conjunction with the Natural RPC will be inhibited.

<b>Statement</b>	<b>Description</b>
FETCH, RUN, INPUT	Not permitted.
STOP, TERMINATE	Same as ESCAPE ROUTINE.

# Setting Up a Natural RPC Environment

To set up a Natural RPC environment, you must perform the following steps for all client and server Naturals:

- Setting Up a Natural Client
  - Setting Up a Natural Server
  - Setting Up an EntireX Broker Access
  - Setting Up an EntireX Broker Environment
  - Starting a Natural Server
  - Considerations for Natural RPC Servers with Replicates
- 

## Setting Up a Natural Client

To set up a Natural client proceed as described below:

1. Define the name of the server to be used.  
Use the SYSRPC utility to define the name of the server to be used for each CALLNAT to be executed remotely. For details, refer to 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, you have to move NATCLTGS to this library or to one of the Steplibs.  
Optionally, you can use the following server selection techniques:
  - Address a default server;  
for more information, see Specifying a Default Server Address Dynamically, or profile parameter DFS.
  - Try alternative servers;  
for more information, see Modifying RPC Profile Parameters Dynamically, or profile parameter TRYALT.
  - Use a Remote Directory Server (RDS),  
for more information, see Using a Remote Directory Server, or profile parameter RDS.**For Windows and UNIX Environments:**  
Predict servers are not maintained in the SYSRPC utility. For information on how to connect to a Predict server, see the profile parameter USEDIC or the Dictionary Server Assignments function in the Global Configuration File.
2. Generate a stub subprogram.  
Skip this step, if you want to work without stub. In this case, set the Natural profile parameter AUTORPC to ON, see Working with Automatic Natural RPC Execution.  
For each CALLNAT to be executed remotely, use the Stub Generation function of the SYSRPC utility, see Creating Stub Subprograms.  
The generated stub must be made available to the Natural client environment. If you have not generated the stub subprogram in your client library, you have to move the stub subprogram to this library or to one of the Steplibs.
3. Set the Natural profile parameters relevant to the client-specific handling of remote procedure calls. These parameters are (all optional, except RPCSIZE on mainframe clients):  
RPCSIZE, MAXBUFF, TIMEOUT, AUTORPC, TRYALT, COMPR, DFS, RDS and the Natural profile parameter CP.

## Setting Up A Natural Server

A Natural server is a Natural 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 proceed as described below:

1. Set the Natural profile parameters relevant to the general and server-specific handling of remote procedure calls in a parameter module for the server NATURAL.

The mandatory profile parameters are:

SERVER, SRVNAME, SRVNODE, RPCSIZE (RPCSIZE refers to mainframe servers only).

Optional parameters are:

RPCSIZE, MAXBUFF, TIMEOUT, LOGONRQ, SRVUSER, TRANSP, TRACE, ACIVERS and the Natural profile parameter CP.

If the EntireX Broker is used, 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

### **For Mainframe Environments:**

If you want to use TCP/IP, you are recommended to set the TRANSP parameter accordingly, as the preferred transport method is using Entire Net-work.

2. Ensure that your Natural server session will enter command mode:
  - Set MENU=OFF in your Natural profile parameters.
  - Do not put a program onto the Natural stack which never terminates.
  - Do not use a STARTUP program which never terminates.
  - Do not disallow NEXT mode in Natural Security for your server library.
3. Ensure that the ADABAS ETID used by the Natural server session is unique within a certain Adabas nucleus.
4. Start a Natural server as described in the section Starting a Natural Server below. This server then waits for remote CALLNAT requests from a client.

### **For OS/390 and VSE/ESA in batch mode:**

For information about servers using the NTASK parameter, refer to Considerations for Natural RPC Servers with Replicates.

## Setting Up an EntireX Broker Access

To set up an EntireX Broker interface proceed as follows:

1. Make the EntireX Broker stub accessible to your Natural environment.

**For Mainframe Environments:**

- If you use the Entire Net-work protocol:  
Link the EntireX Broker stub NATETB23 to your Natural or specify RCA=BROKER to load NATETB23 dynamically at run-time.
- If you use the TCP/IP protocol:  
Specify RCA=BROKER RCALIAS=(BROKER,*stubname*)  
where *stubname* refers to one of the TCP/IP-enabled EntireX Broker stubs BKIMBTSO, BKIMBTIA, EXAAPSB or EXAAPSC.

In OS/390 batch mode and under TSO, you may either use BKIMBTSO or EXAAPSB.

You must use the SMARTS-based EXAAPSB:

- If you want to communicate with the EntireX Broker using SSL.
- If you want to use the location transparency provided by the EntireX Broker.

Refer to the EntireX documentation for details.

**For UNIX:**

The EntireX Broker library stub can be assigned in the Local Configuration File of the Natural parameter module, entry NATEXTLIB.

**For Windows:**

The EntireX Broker stub must be accessible over the registry.

2. Set the RPC parameter ACIVERS according to your requirements:

**Note:** The ACIVERS value set in the parameter module can only work if the EntireX Broker and EntireX Broker stub support this version as well.

Setting	Function
<b>ACIVERS=2</b>	(Default) Support of the EntireX Broker functions LOGON and LOGOFF. The server performs a LOGON to the EntireX Broker before executing the REGISTER, and a LOGOFF after the DEREGISTER. This does not imply any security checks, but it is a pure EntireX Broker management function, see EntireX Broker function LOGON.
<b>ACIVERS=3</b>	Support of EntireX Broker non-numeric conversation IDs. When this Natural parameter is set to 3 or higher, the EntireX Broker will also assign non-numeric conversation IDs. If a Natural client issues an OPEN CONVERSATION and the client's ACIVERS is 3 or higher, the EntireX Broker will be able to automatically assign non-numeric conversation IDs. It will not check whether the associated server does accept non-numeric conversation IDs, but only the ACIVERS of the requestor (a Natural client in this case) will be decisive. <b>Therefore, make sure that both the Natural client and the server support the respective ACI version.</b>
<b>ACIVERS=4</b>	Support of code pages and (for servers only) Natural Security. With EntireX Broker ACI Version 4 or higher, the Natural RPC supports code pages. For this, the name of the code page can be specified in the Natural profile parameter CP for clients and servers. The evaluation of the code page is done by the EntireX Broker. The EntireX Broker translates the RPC data sent according to the code page of client and server to the corresponding target code page. The CP parameter can be set by the client and/or by the server. It applies for the current process. This means that the client code page does not need to be identical with the server code page. The server is enabled to logon to the EntireX Broker using a qualified user ID. If the Natural parameter/subparameter SRVUSER is set to *NSC and the server is running under Natural Security, the Natural RPC will automatically pass the current Natural user ID (*USER) and the password defined in Natural Security to the EntireX Broker, where they are checked for conformity with the EntireX Broker security data.
<b>ACIVERS=6</b>	If you are using the EntireX Broker stub EXAAPSC (CICS only), we strongly recommend that you use the specification of the ACI Version 6. In this case, Natural will use the TERMINATE option for the LOGOFF from the EntireX Broker.

- For additional RPC parameters affecting the EntireX Broker, refer to the Profile Parameters section in the Parameter Reference documentation.

## Using TCP/IP as a Transport Method

If TCP/IP is used as transport method and you use a host name to address the server node, you must either define the server node in the hosts and services directory of your TCP/IP installation or use a DNS.

## Setting Up an EntireX Broker Environment

In the EntireX Broker Attribute File, add the following:

- For each Natural RPC server, a service definition must be specified as follows:  
`CLASS=RPC, SERVICE=CALLNAT, SERVER=servername.`
- If you want to use the conversion services, set `CONVERSION=userexit`. In this case, you must set the Natural profile parameter CP accordingly.
- If the Natural RPC client and the Natural RPC server are of Natural Version 3.1 or higher (mainframe environments) or of Natural Version 4.1 or higher (Windows and UNIX environments), you can set `AUTOLOGON=NO`.  
 In this case, ACIVERS must be 2 or higher.

4. If both the Natural RPC client and the Natural RPC server are of Natural Version 3.1 or higher (mainframe environments) or of Natural Version 4.1 or higher (Windows and UNIX environments) and Natural Security is installed, you can enable EntireX Security by setting:

`SECURITY=YES.`

In this case, `SRVUSER` must be set to `*NSC` on the server side.

## Starting a Natural 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 or as a background task.

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

- Create an RPC-specific Natural parameter module, see Setting Up a Natural Server.  
This parameter module is either specified dynamically with `PARM=serverparm`, where *serverparm* is the name of the parameter module (all platforms) or linked to your Natural (on mainframe only).
- Alternatively, you can also specify the profile parameters dynamically.
- **In mainframe environments only:** The RPC-specific Natural profile parameter may be specified in a profile created with the SYSPARM utility.  
Natural would then be started with  
`PROFILE=serverprofile`  
where *serverprofile* is the name of the profile.

How a Natural server is started depends on the environment, refer to the corresponding paragraph:

- Starting a Natural Server in a Mainframe Online Environment
- Starting a Batch Server in a Mainframe Environment
- Starting a Server in a UNIX Environment
- Starting a Server in a Windows Environment

### Starting a Natural Server in a Mainframe Online Environment

To start a Natural server in a mainframe online environment, enter the following command:

```
<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).

#### For CICS and Com-plete only:

You can also use the Natural program STARTSRV in library SYSRPC to start a Natural server in asynchronous mode. 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 Security (NSC) is used, the user ID of the current Natural session is propagated, too. You may adapt the input to your requirements.

Some Natural profile parameters are implicitly added by RPCSSRV. If you 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 Batch Server in a Mainframe Environment

Batch servers are started correspondingly. A batch server is a standard Natural batch session that is started with the RPC profile parameters above. For a sample JCL see Using the Server Trace Facility.

Under OS/390, a sample JCL for a started task is provided in Installing Natural under OS/390, Create Sample JCL for Natural RPC Server.

**For Batch Mode under OS/390 and VSE/ESA only:**

You can also run a batch server with replicates by setting the RPC parameter `NTASKS` to a value greater than 1. Replicates 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 Server in a UNIX Environment

To start a Natural server under UNIX, enter the following command:

```
natural parm=serverparm >/dev/null </dev/null &
```

or

```
natural server=on, srvname=servername, srvnode=nodename, maxbuff=n >/dev/null </dev/null &
```

## Starting a Server in a Windows Environment

To start a Natural server under Windows, proceed as follows:

1. Create a shortcut for Natural.
2. Enter the shortcut properties.
3. Create a Natural parameter module with the RPC server parameters set.
4. In the Target input field, edit the Natural path and append:

```
parm=serverparm batch
```

or

```
server=on, srvname=servername, srvnode=nodename, maxbuff=n batch
```

## Considerations for Natural RPC Servers with Replicates

For OS/390 and VSE/ESA only.

### Natural RPC Batch Server with NTASKS >1

The main task and all replicates run in the same OS/390 region or VSE/ESA partition.

1. Use the reentrant version ADALNKR of the Adabas link module ADALNK.  
If you use ADAUSER, you must rename ADALNKR to ADALNK.  
**Note:**  
You may need a separate Copy of the reentrant ADALNK module if you are using 3GL programs which do not pass a work area as 7th Adabas parameter to the Adabas interface.
2. In the NATPARM module:
  - Set the NTRPC subparameter NTASKS = *n*, where *n* is the number of parallel servers (< 100) to be started, including the main task.  
**Note for VSE/ESA:**  
The number of subtasks is restricted by the operating system. Ask your system administrator.
  - ETID must be specified as a **blank** character to prevent a NAT3048 (ETID not unique in Adabas nucleus) error when the subtask is started.
3. When using dynamic Natural profile parameters:  
Use the CMPRMIN dataset to pass the dynamic Natural profile parameters to Natural. Do **not** use the PARM card or the CMSYNIN dataset.
4. When using a local buffer pool (OS/390 only):  
Each subtask allocates its own local buffer pool unless you specify a shared local buffer pool. See Natural profile parameter LBPNAME in the section NTOS Macro - Generation Parameters for Natural under OS/390 in the Natural Operations for Mainframes documentation.
5. In the Natural front-end link job (OS/390 only):  
Link the front-end reentrant by using the RENT option of the linkage editor.  
If the front-end is not linked with the RENT option, only the main task will start the communication with the EntireX Broker. All subtasks are set to a WAIT status by OS/390, until the main task has been terminated. If you terminate the RPC server lateron, the address space will hang and must be cancelled.  
**Note:**  
If you use ADAUSER you must not link ADAUSER with your front-end as ADAUSER is non-reentrant. Instead, use the Natural profile parameter ADANAME and set ADANAME to ADAUSER. This will cause Natural to load ADAUSER dynamically at runtime.
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 OS/390:**  
If you cannot make a module reentrant, link the module as non-reusable (do **not** specify the link option RENT or REUS). This will ensure that each subtask gets its own copy.

### Running a Batch Server with Replicates

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

# Operating a Natural RPC Environment

This section mainly describes the tasks required to operate a Natural RPC environment. Some of these tasks are performed with the SYSRPC utility. For instructions on the functions the SYSRPC utility provides, refer to the Natural SYSRPC Utility documentation.

This section covers the following topics:

- Specifying RPC Server Addresses
  - Stubs and Automatic RPC Execution
  - Modifying RPC Profile Parameters Dynamically
  - Executing Server Commands
  - Logon to a Server Library
  - Using the LOGON Option
  - Using Natural RPC with Natural Security
  - Using Natural RPC with EntireX Security
  - Using Compression
  - Using Secure Socket Layer
  - Monitoring the Status of an RPC Session
  - Handling Errors
  - Terminating a Natural RPC Server
- 

## 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 RPC 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.

If EntireX Broker is used, it is also possible to define servers using the EntireX location transparency.

Below is information on:

- 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 EntireX Location Transparency

## 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, SYSTRANS or the Natural Object Handler) to move NATCLTGS into one of the steplib 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.

For information on the SYSRPC Remote Directory Maintenance function, see the relevant section in the SYSRPC Utility documentation.

## 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 RPC profile parameter DFS, as described in your Natural Operations 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** in the library SYSEXT. The 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. It includes the following parameters:

Parameter	Format	Explanation
<i>function</i>	A1	<p><b>P</b> Put: Determines that the server address (composed of the parameters <i>nodename</i> and <i>servername</i>, see below) is the default address for all subsequent remote procedure calls not defined in the service directory.</p> <p>To remove a default server address, specify a "blank" for <i>nodename</i> and <i>servername</i>.</p> <p><b>G</b> Get: Retrieves the current default server address as set by the function P.</p>
<i>nodename</i>	A192	<p>Specifies/returns the name of the server node to be addressed. The node name may have up to 32 characters for physical node names and up to 192 characters for logical node names. See Using EntireX Location Transparency.</p> <p><b>Note:</b> For compatibility reasons, <i>servername</i> is defined with BY VALUE RESULT to support existing callers which pass an A8 field for the <i>servername</i>.</p>
<i>servername</i>	A192	<p>Specifies/returns the server name to be addressed. The server name may have up to 32 characters for physical server names and up to 192 characters for logical service names. See Using EntireX Location Transparency.</p> <p><b>Note:</b> For compatibility reasons, <i>nodename</i> is defined with BY VALUE RESULT to support existing callers which pass an A8 field for the <i>nodename</i>.</p>
<i>logon</i>	A1	Specifies/returns the logon option.
<i>protocol</i>	A1	<p>Specifies/returns the transport method.</p> <p>Valid value: B (=EntireX Broker).</p>

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 connection 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 Natural RPC profile parameter TRYALT to ON to give permission to use an alternative server.

This parameter can also be set dynamically for the current session See the Parameter Maintenance function as described in the SYSRPC Utility documentation.

## Using EntireX Location Transparency

Using EntireX location transparency, you can change physical node and server names without having to configure anything or to change client and/or server programs. Now, instead of using a physical node and physical server name, a server can be addressed by a logical name. The logical name is mapped to the physical node and server names using directory services.

To take advantage of location transparency, the Natural RPC has been enabled to accept a logical name wherever only a node and server name could be specified before. The logical name is passed to the EntireX Broker before it is used the first time.

The maximum length of a logical name is 192 characters. To avoid new Natural profile parameters, a logical name is specified in the server name part of the already existing parameters. There are two kinds of logical names:

- **Logical node names**

With a logical node name you specify a logical name for the node only in conjunction with a real server name. A logical node name can be used in all places where you can also use a real node name. To define a logical node name the keyword LOGBROK must be used.

**Example:**

```
SRVNODE=' LOGBROK=logical_node_name,my_set'
```

- **Logical services**

With a logical service, you specify a logical name for both the node and the server. A logical service can be used in all places where you can also use a real node and server name. To define a logical service, the node name must be set to \* (intentionally left empty), and the server name contains the logical service name.

**Example:**

```
SRVNODE='*' SRVNAME='logical_service_name,my_set'
```

**Note:**

In the case of interface USR2071N, you can LOGON to a logical service name by using the keyword LOGSERVICE together with the logical service name in the field *broker-id*.

For more details about the EntireX location transparency, refer to the EntireX documentation.

The following components refer to node and server names:

- Natural profile parameters SRVNODE, SRVNAME, DFS and RDS
- Service maintenance of the SYSRPC utility
- Service directory (NATCLTGS)
- User application interfaces USR2007N, USR2071N
- Service programs RPCERR, RPCINFO

See also Location Transparency in Service Directory Maintenance in the Natural SYSRPC utility documentation.

## Stubs and Automatic RPC Execution

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

However, generating stubs 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 a stub, for an incorrect CALLNAT you may receive follow-up errors returned from the transport layer or the Natural server.

Below is information on:

- Creating Stub Subprograms
- Working with Automatic Natural RPC Execution

## Creating Stub Subprograms

With the Stub Generation function of the SYSRPC utility, you can generate the Natural stub subprograms used to connect the client's calling program to a subprogram on a server. The stub consists of a parameter data area (PDA) and of the server call logic.

The PDA contains the same parameters as used in the CALLNAT statement of the calling program and must be defined in the Stub Generation screen of the Stub 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 Stub Generation function after the PDA has been defined.

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

For information on the SYSRPC Stub Generation function, see the relevant section in the SYSRPC Utility documentation.

## Working with Automatic Natural RPC Execution

You are not required to generate Natural RPC stubs, but you can work with automatic Natural RPC execution (i.e. without using Natural stubs). To work with automatic Natural RPC execution set the RPC parameter AUTORPC as follows:

```
AUTORPC=ON
```

In that case, you can omit the generation of the client stub during your preparations for RPC usage. When the automatic Natural RPC execution is ON, Natural behaves as follows:

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

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

If AUTORPC is set to ON, and a Natural stub exists, it will still be used.

## Modifying RPC Profile Parameters Dynamically

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

### Attention:

These modifications are retained as long as the user session is active; they are lost when the session is terminated. Static settings are only made using Natural profile parameters.

## 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.

## 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) <b>and</b> ( 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 DFS parameter, or the application programming interface USR2007N.
3. LOGONRQ can be set as a Natural profile parameter 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) or not?
6. The setting of the LOGON option in the NSC library profile (Session options > RPC restrictions) of the NSC server application. If the NSC LOGON option is set to 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.

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 steplib 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 make a logoff to the previous library.

## 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 DFS profile parameter or the Interface USR2007N.

## Settings Required on the Server Side

No setting is required on the server side.

# 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. If security data is to be passed to the server, the LOGON option (see also Using the LOGON Option) must be used.

The user ID and password are established as follows:

- **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 server.

- **For non-Natural Security clients:**

The application programming interface **USR1071N** is provided which the user has to call for specifying logon data which are then passed to the server. USR1071N is contained in the library SYSEXT. The logon data contains the user ID and password from which the so-called security token is generated, and additionally some administrative information.

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

To invoke, for example, USR1071P from within a program, specify:

```
FETCH RETURN 'USR1071P' USERID(A8) PASSWORD(A8).
```

For a more detailed description, see the **USR1071T** member in library SYSEXT.

If the server runs under Natural Security, 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 made current 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 before the conversation was opened.

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/subparameter LOGONRQ to ON for the server.

As part of the Natural RPC Restrictions in library profiles of Natural Security, a 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. See Natural RPC Restrictions in the Natural Security documentation.

## Using Natural RPC with EntireX Security

Natural RPC fully supports EntireX Security on the client side and the server side.

### Client Side

To logon to and logoff from the EntireX Broker, the application programming interface USR2071N is provided in library SYSEXT. 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 feature is fully transparent to the Natural application.

If EntireX Security has been installed or if AUTOLOGON=NO has been 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.

### Using the Application Programming Interface USR2071N

USR2071N has the following parameters:

Parameter	I/O	Format	Description
<i>function</i>	I	A08	Function code.  Values:  LOGON          Logon to EntireX Broker  LOGOFF         Logoff from EntireX Broker
<i>broker-id</i>	I	A192	Broker ID The <i>broker-id</i> may have up to 32 characters for physical node names and up to 192 characters for logical node names or logical service names. See Using EntireX Location Transparency.  <b>Note:</b> For compatibility reasons <i>broker-id</i> is defined with BY VALUE RESULT to support existing callers which pass an A8 field for the <i>broker-id</i> .
<i>user-id</i>	I	A08	User ID.
<i>password</i>	I	A08	User ID's password.
<i>newpassw</i>	I	A08	User ID's new password.
<i>rc</i>	O	N04	Return value:  0            ok  1            invalid function code  9999        EntireX Broker error (see <i>message</i> )
<i>message</i>	O	A80	Message text, returned by EntireX Broker.

The Subprogram USR2071N should be copied to the Library SYSTEM or to the steplib library, or to any application.

The parameters listed above must be defined via DEFINE DATA.

The calling program must contain the following statement:

```
CALLNAT 'USR2071N' FUNCTION BROKER-ID USER-ID PASSWORD NEWPASSW          RC MESSAGE
```

### Special considerations when using location transparency:

- If you want to LOGON using a logical node name, you have to use the LOGBROK keyword.

```
BROKER-ID := 'LOGBROK=my_logical_node,my_set'
```

- If you want to LOGON using a logical service name, you have to use the LOGSERVICE keyword.

```
BROKER-ID := 'LOGSERVICE=my_logical_service,my_set'
```

## 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 \*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 LOGON 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 reLOGON 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 reLOGON 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.

## Server Side

If the value of 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:

- **SRVUSER=\*NSC**

If Natural Security is installed on the server, you can specify `SRVUSER=*NSC` to determine that the current Natural Security userID 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.

- **USR2072N**

The application programming interface `USR2072N` enables you to specify a password which is used for the LOGON in conjunction with `SRVUSER`.

### Using the Interface `USR2072N`

`USR2072N` has the following parameter:

Parameter	I/O	Format	Description
<i>password</i>	I	A08	User ID's password.

The Subprogram `USR2071N` should be copied to the library `SYSTEM` or to the `steplib` library, or to any application.

The parameter listed above must be defined using the `DEFINE DATA` statement.

The calling program must contain the following statement:

```
CALLNAT 'USR2072N' PASSWORD
```

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:

```
STACK=(LOGON server-library;USR2072P password)
```

## Using Compression

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

Compression Type	Description
<b>COMPR=0</b>	All <code>CALLNAT</code> parameter values are sent to and returned from the server, i.e. no compression is performed.
<b>COMPR=1 (default)</b>	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.
<b>COMPR=2</b>	Same as for <code>COMPR = 1</code> , except that the server reply message still contains the format description of the <code>CALLNAT</code> parameters. This might be useful if you want to use certain options for data conversion in the Software AG product <code>EntireX Broker</code> (for more information, see the description of <code>Translation Services</code> in the <code>EntireX Broker</code> 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

## Using Interface USR2035N

USR2035N has the following parameters:

Parameter	I/O	Format	Description
<i>function</i>	I	A01	Function code.
			Values:
			P Put: Specify a new SSL parameter string.
		G	Get: Retrieve previously specified SSL parameter string.
<i>SSLPARMS</i>	I	A128	SSL parameter string as required by the EntireX Broker

The Subprogram USR2035N should be copied to the library SYSTEM or to the steplib library, or to any application.

The parameters listed above must be defined via DEFINE DATA.

The calling program must contain the following statement:

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

## Functionality of Interface USR2035N

### P (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 USR2035N each time before you access the EntireX Broker the first time.

### Example:

```
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 user application programming interface USR2035N to set the SSL parameter string.

**G (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.

## 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 use the RPCERR program from the command line or invoke it via FETCH from within a Natural program.

RPCERR displays the last Natural error number and message if it was RPC related and it also displays the last BROKER reason code and associated message. Additionally, 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 :3:.

RPC error information:
  No additional information available.

Server Node: Library:      SYSRPC
  Server Name:      Program: NATCLT3
                          Line No: 0480
```

### 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 SYSRPC.

**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
```

RPCINFO has the following parameters which are provided in the PDA 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>B</b> (EntireX Broker)
S-NODE	A8	The node name of the server.
S-NAME	A8	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.
S-CON-OPEN	L	Indicates whether there is an open conversation. This parameter contains value TRUE if a conversation is open, otherwise it contains FALSE.
CLIENT-PARMS		Contain information about the Natural session when acting as a client. The CLIENT-PARMS only apply if you execute RPCINFO remotely on an RPC client.
C-BIKE	A1	Transport protocol used.  Possible value: <b>B</b> (EntireX Broker)
C-NODE	A8	The node name of the previously addressed server.
C-NAME	A8	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 value TRUE if a conversation is open, otherwise it contains FALSE.

## 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

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.

## Defining the Trace File

The trace file definition depends on the environment. The following topics are covered below:

- Trace File Handling for Mainframe Environments - General Information
- Trace File Handling in OS/390 Batch Mode
- Trace File Handling under CICS
- Trace File Handling in VSE/ESA Batch Mode
- Trace File Handling in BS2000/OSD Batch Mode

### Trace File Handling for Mainframe Environments - General Information

On the mainframe, define the trace file appropriate to your environment, see also the NTPRINT macro in the Natural Operations Manual for Mainframes.

### Trace File Handling in OS/390 Batch Mode

#### a) Running A Server As Single Task

In the server start job, assign an OS/390 dataset to the Natural additional report CMPRT10.

#### Example:

```
//NAT31     JOB   CLASS=K,MSGCLASS=X
//NATSTEP   EXEC  PGM=NATOS31
//STEPLIB   DD    DISP=SHR,DSN=SAG.NAT.LOAD
//          DD    DISP=SHR,DSN=SAG.ETB.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 Replicates

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 RPC subparameter TRACE 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 $nm$  for all subtasks,  $nm = NTASKS-1$ ;

**Example:**

```
//NAT31      JOB  CLASS = K,MSGCLASS = X
//NATSTEP   EXEC PGM=NATOS31,REGION = 8M
//STEPLIB  DD   DISP = SHR,DSN = SAG.NAT.LOAD
//          DD   DISP = SHR,DSN = SAG.ETB.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, Natural Advanced Facilities is required to write to the additional Report 10. If Natural Advanced Facilities is not installed or not wanted, the trace file is written to the Work File 10, provided that this file exists and Print File 10 has not been defined, otherwise the trace is disabled.

The Natural work file should be assigned to an extra-partitioned transient data queue.

**Examples:****Natural definition:**

```
NTWORK ((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 in VSE/ESA Batch Mode

In VSE/ESA batch mode assign a trace file to the Printer Number 10.

#### Example:

```
// LIBDEF PHASE,SEARCH=(SAGLIB.NATnnn,SAGLIB.ETBnnn),TEMP
// ASSGN SYS000,READER // ASSGN SYSLST,FEE // ASSGN SYS050,FEF
// EXEC RPCVSEE3,RPCSIZE=AUTO,PARM='SYSRDR'
IM=D,MADIO=0,MT=0,OBJIN=R,AUTO=OFF,MAXCL=0,ID=',',INTENS=1,
PRINT=((10),AM=STD,SYSNR=50)
/*
```

### Trace File Handling in BS2000/OSD Batch Mode

In BS2000/OSD batch mode assign a trace file to Printer Number 10.

#### Example:

```
/.RPCSERV          LOGON
/                  SYSFILE   SYSOUT = output-file
/                  SYSFILE   SYSDTA = (SYSCMD)
/                  SYSFILE   SYSIPT = (SYSCMD)
/                  FILE trace-file,LINK = P10,OPEN = EXTEND      */server trace file
/                  STEP
/                  SETSW      ON = 2
/                  EXEC $DC1.NATDB64B
MADIO = 0,IM = D,ID = ',','PRINT = ((10),AM = STD)
```

## 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 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, proceed as follows:

1. Add a FIND FIRST or HISTOGRAM statement in program NATRPC39, library SYSRPC.
2. Copy the updated program to library SYSTEM on FUSER.  
The STEPLIB concatenation of the library to which the server currently is logged on is not evaluated.

### User Exit NATRPC01

This exit 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 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.

**Important Notes:**

1. NATRPC01 must be located in library SYSTEM on FUSER. The STEPLIB concatenation of the library to which the server currently is logged on is **not** evaluated.
2. The DEFINE DATA PARAMETER block must not be changed.

## Terminating a Natural RPC Server

- Using SYSRPC
- Using Entirex Control Center or EntireX System Management Hub
- User Exit NATRPC99

### Using SYSRPC

Use the TE command of the Server Command Execution function.

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 Entirex Control Center or EntireX System Management Hub

Use the Shutdown command (EntireX Control Center) or the Deregister Button (EntireX System Management Hub).

A Natural RPC server can only be terminated if the server is not currently executing a remote CALLNAT request.

### User Exit NATRPC99

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

- If no NATRPC99 is found, the server terminates immediately as usual.
- If 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 (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 (CMSYNIN).

#### Important Notes:

1. NATRPC99 must be a Natural program.
2. NATRPC99 is currently only called if the server is terminated with a TE command issued using the Server Command Execution function of the SYSRPC utility. The exit is not called if the server is terminated via the Entirex Control Center or the EntireX System Management Hub.
3. NATRPC99 must be located in the library SYSTEM on FUSER. The STEPLIB concatenation of the library to which the server currently is logged on is not evaluated to find NATRPC99.
4. 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 FUSER).

# Using a Conversational RPC

This section covers the following topics:

- Opening a Conversation
  - Closing a Conversation
  - Defining a Conversation Context
  - Modifying the System Variable \*CONVID
- 

## 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 process which is exclusively reserved for this conversation.
- If it is not member of the current conversation, it will be executed in a different server process. This also applies to different conversations.

A conversation can be opened on any program level and CALLNATs 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 replicate 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.

### ▶ To 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.

### ▶ To close a conversation with implicit END TRANSACTION (Commit)

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

The exit 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 "+".

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.

# Using a Remote Directory Server - RDS

This section covers the following topics:

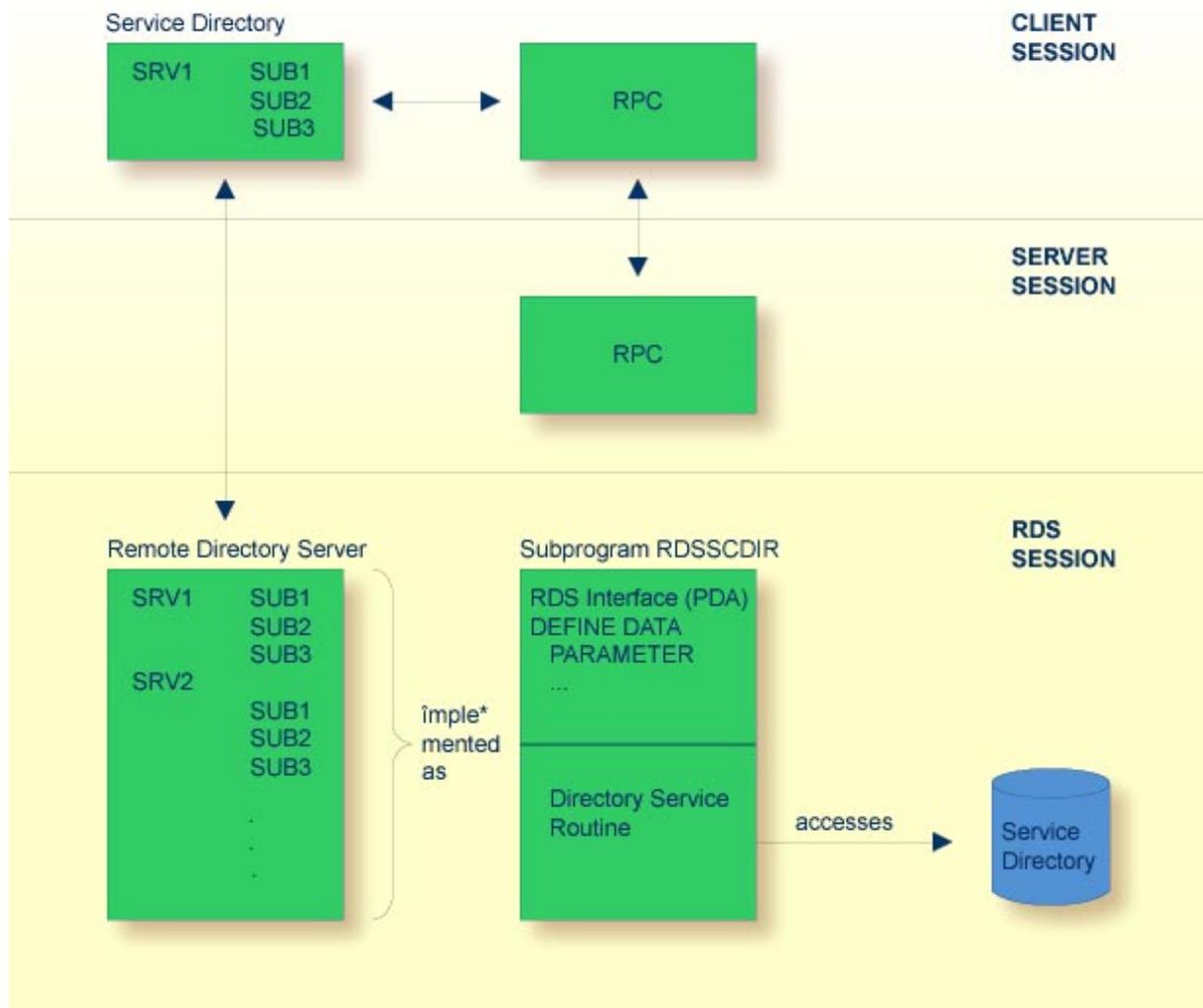
- RDS Principles of Operation
  - Using a Remote Directory Server
  - Creating an RDS Interface
  - Creating a Remote Directory Service Routine
  - Remote Directory Service Program RDSSCDIR
- 

## 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 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 this subprogram is provided in library SYSRPC as subprogram RDSSCDIR. It 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 for the remote directory service using the Remote 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 the CMWKF01 assigned to the appropriate dataset in the server startup JCL.

2. Start the remote directory server and proceed with the following steps.
3. Define the RDS in the profile parameter RDS.

Alternatively, you can 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 RDS subparameter of session parameter RPC. 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. The example below shows how to define a remote directory server in 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 "#" 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	Explanation
<b>P_UDID(B8)</b>	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.
<b>P_UDID_EXPIRATION (I4)</b>	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 to an unnecessarily low value, you may cause a lot of network traffic to the RDS.
<b>P_CURSOR (I4)</b>	The remote procedure call has the option to scan for an alternative server if a connection to the previous one cannot be established (see RPC subparameter 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.
<b>P_ENTRIES (I4)</b>	This parameter contains the number of service definitions in P_REQUEST.
<b>P_REQUEST (A16/1:250)</b>	A list of services for which a server address can be scanned. An entry is structured as program name (A8) library name (A8)
<b>P_EXTENT (A16/1:250)</b>	Reserved for future use.
<b>SRV_NODE (A8)</b>	Contains the server node.
<b>SRV_NODE_EXT (A8)</b>	Contains the server node extension..
<b>SRV_NAME (A8)</b>	Contains the server name.
<b>SRV_NAME_EXT (A8)</b>	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).

For the delivered version of RDSSCDIR, this is the CMWKF01, which is assigned to the appropriate dataset in the server startup JCL.

### 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 TESTS1, TESTS2 and TESTS3 in library SYSTEM.
- The server SERVER2 may execute TESTS4 on library SYSTEM.
- One server SERVER1 on node NODE2 which may execute 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 Remote Directory Maintenance function.