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 information about 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 during a Natural Session
- Executing Server Commands
- Logon to a Server Library
- Using the Logon Option
- Using Compression
- Using Secure Socket Layer
- Monitoring the Status of an RPC Session
- Retrieving Runtime Settings of a Server
- Setting/Getting Parameters for EntireX
- Handling Errors
- User Exits before and after Service Execution

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 keyword subparameter DFS of profile parameter RPC or parameter macro NTRPC,
- 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, see *Using 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` or the Natural Object Handler) to move `NATCLTGS` into one of the steplibs defined for the application.

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

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

Using Remote Directory Entries

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

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 keyword subparameter `DFS` of profile parameter `RPC` or parameter macro `NTRPC`, 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. This interface enables you to determine a default server address that is to be used each time a remote program cannot be addressed via the service directory.

To make use of USR2007N

1. Copy the subprogram USR2007N from the library SYSEXT to the library SYSTEM or to the steplib library, or to any application in the server environment.
2. Using the DEFINE DATA statement in structured mode or the RESET statement in reporting mode, specify the following parameters:

Parameter	Format	Description	
function	A1	Function; possible values are:	
		P (Put)	Determines that the server address (composed of the parameters *nodename* and *servername*, see below) is the default address for all subsequent remote procedure calls not defined in the service directory. To remove a default server address, specify a blank for *nodename* and *servername*.
		G (Get)	Retrieves the current default server address as set by the function P.
nodename	A192	Specifies/returns the name of the server node to be addressed. 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*. **Note:** For compatibility reasons, *servername* is defined with the option BY VALUE or BY VALUE RESULT (see the section parameter-data-definition in the description of the DEFINE DATA statement) to support existing callers which pass an A8 field for the *servername*. The sample USR2007P provided in library SYSEXT supports up to 32 characters.	

Parameter	Format	Description	
servername	A192	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*. **Note:** For compatibility reasons, *nodename* is defined with the option BY VALUE or BY VALUE RESULT (see the section parameter-data-definition in the description of the DEFINE DATA statement) to support existing callers which pass an A8 field for the *nodename*. The sample USR2007P provided in library SYSEXT supports up to 32 characters.	
logon	A1	Specifies/returns the Logon option, see *Using the Logon Option*.	
protocol	A1	Specifies/returns the transport protocol. Valid value: B (=EntireX Broker).	
noservdir	A1	Specifies/returns the service directory option, see keyword subparameter DFS of profile parameter RPC or parameter macro NTRPC.	
		Y	Service directory must not be present
		N	Service directory must be present

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

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

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

Using an Alternative Server

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

- The client makes a first attempt to establish the connection.
- If this attempt fails, instead of providing an error message, a second attempt is made, however, this time not on the same server. Instead, the service directory is searched again starting at the current entry to find out whether or not another server is available which offers the desired service.
- If a second entry is found, Natural tries to establish the connection to this server. If the remote procedure call is performed successfully, the client application keeps on running. The user does not notice whether the connection to the first server or to the alternative server produced the result.

- If no further entry is found or if the 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 keyword subparameter TRYALT of profile parameter RPC or parameter macro NTRPC to ON to give permission to use an alternative server.

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

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 LOGBROKER must be used.

  Example:

  ```
  SRVNVODE='LOGBROKER=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, an asterisk (*) must be specified as node name (intentionally left empty), and the server name contains the logical service name.

  Example:

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

  If the Natural Application Programming Interface USR2071N is used, 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 further information about *EntireX Location Transparency*, refer to the EntireX documentation.

The following components refer to node and server names:

- Natural keyword subparameters SRVNODE, SRVNAME, DFS and RDS of profile parameter RPC or parameter macro NTRPC
- Service Directory Maintenance function of the SYSRPC utility
- Service directory (NATCLTGS)
- Natural Application Programming Interfaces USR2007N, USR2071N

See also *Location Transparency* in *Service Directory Maintenance* function of the *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.

If you want to call an EntireX RPC server with a remote CALLNAT execution, it is strongly recommended to use a stub subprogram (interface object). A stub subprogram is required if the IDL (Interface Definition Language) definition of the subprogram you want to call on an EntireX RPC server contains a group structure. In this case, you must define the same group structure during the stub generation on the Stub Generation screen or generate the stub subprogram from the EntireX IDL file (Windows only).

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; see *Stub Generation* in the *SYSRPC Utility* documentation.

The PDA contains the same parameters as used in the CALLNAT statement of the calling program and must be defined 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.

Working with Automatic Natural RPC Execution

You are not required to generate Natural RPC stubs, but you can work with automatic Natural RPC execution (that is, without using Natural stubs). To work with automatic Natural RPC execution, set the keyword subparameter AUTORPC of profile parameter RPC or parameter macro NTRPC 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 enabled (AUTORPC=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 keyword subparameter AUTORPC of profile parameter RPC or parameter macro NTRPC is set to ON, and a Natural stub exists, it will still be used.

Modifying RPC Profile Parameters during a Natural Session

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.

Caution:
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) *and* (LOGONRQ=YES) is not possible.			
4	Lib1	yes	No influence	No influence	NSC	AUTO	Lib1
5	Lib1	yes	No influence	No influence	NSC	N	Lib1
6	Lib1	yes	No influence	No influence	Native Natural	--	Lib1

Explanation of the table columns:

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

   The flag can be set by using

   the Service Directory Maintenance function of the SYSRPC utility,

   or the keyword subparameter DFS of profile parameter RPC or parameter macro NTRPC,

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

Using the Logon Option

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

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

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

Logging on to a Different Library

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

To make use of USR4008N

1. Copy the subprogram USR4008N from the library SYSEXT to the library SYSTEM or to the steplib library, or to any application in the server environment.
2. Using the DEFINE DATA statement, specify the following parameters:

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

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

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

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

Settings Required on the Client Side

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

Settings Required on the Server Side

No setting is required on the server side.

Using Compression

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

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

Using Secure Socket Layer

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

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

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

Example:

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

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

To make use of USR2035N

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

Parameter	I/O	Format	Description	
FUNCTION	I	A01	Function code; possible values are:	

			P (Put)	Specify a new SSL parameter string.
				
				The SSL parameter string is internally saved and passed to EntireX each time an EntireX Broker using SSL communication is referenced the first time. You may use different SSL parameter strings for several EntireX Broker connections by calling application programming interface USR2035N each time before you access the EntireX Broker the first time.
				
				Example:
				
				`FUNCTION := 'P'`
				`SSLPARMS := 'TRUST_STORE=FILE://DDN:CACERT&VERIFY_SERVER=N'`
				`CALLNAT 'USR2035N' USING FUNCTION SSLPARMS`
				
				To set SSL parameters in case of a Natural RPC server, put the name of the calling program onto the Natural stack when starting the server.
				
				Example:
				
				`STACK=(LOGON server-library;set-SSL-parms)`
				
				Where *set-SSL-parms* is a Natural program that invokes the application programming interface USR2035N to set the SSL parameter string.

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

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

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

Monitoring the Status of an RPC Session

This part is organized in the following sections:

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

Using the RPCERR Program

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

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

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

Example of an RPC Error Display: RPCERROR

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

RPC error information:
No additional information available.

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

Using the RPCINFO Subprogram

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

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

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

Example:

```
DEFINE DATA LOCAL USING RPCINFOL
  LOCAL
  1 PARM   (A1)
  1 TEXT   (A80)
  1 REDEFINE TEXT
    2 CLASS  (A4)
    2 REASON (A4)
END-DEFINE
...
```

```
OPEN CONVERSATION USING SUBPROGRAM 'APPLSUB1'
  CALLNAT 'APPLSUB1' PARM
CLOSE CONVERSATION *CONVID
...
ON ERROR
   CALLNAT 'RPCINFO' SERVER-PARMS CLIENT-PARMS
   ASSIGN TEXT=C-ERROR-TEXT
   DISPLAY CLASS REASON
END-ERROR
...
END
```

Parameters of RPC Info

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

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

Parameter	Format	Description
C-NAME	A32	The name of the previously addressed server.
C-ERROR-TEXT	A80	Contains the message text returned from the transport layer.
C-CON-ID	I4	Conversation ID of the last server call. Note that this is the physical ID from EntireX Broker, not the logical Natural ID. If no conversation is open, the value of this parameter is less than or equal to 0. If the physical conversation ID is either non-numeric or greater than I4, a -1 is returned.
C-CON-OPEN	L	Indicates whether there is an open conversation. This parameter contains the value TRUE if a conversation is open, otherwise it contains FALSE.
C-ENTIREX-RPC-ERROR-MESSAGE	A	Contains the message text returned from an EntireX RPC server.

Using the Server Trace Facility

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

Activating/Deactivating the Server Trace Facility

To activate/deactivate the server trace facility, start the server with the option

TRACE=*n*

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

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

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

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

Support of TS=ON for RPC Server Trace

The following information applies to Mainframe environments only:

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

Defining the Trace File

The trace file definition depends on the environment:

- Trace File Handling for Mainframe Environments - General Information
- Trace File Handling in z/OS Batch Mode
- Trace File Handling under CICS
- Trace File Handling in z/VSE Batch Mode
- Trace File Handling in BS2000/OSD Batch Mode
- Trace File Handling for UNIX and OpenVMS Environments
- Trace File Handling for Windows

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 *Parameter Reference* documentation).

Trace File Handling in z/OS Batch Mode

a) Running A Server As Single Task

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

Example:

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

b) Running a Server With Replicas

1. Set the RPC parameter `NTASKS` to a value greater than 1.
2. Assign `CMPRMIN` to a dataset with `DISP=SHR` or to `*`.

3. As each task writes on a separate CMPRINT dataset, define the following DD card names:

   CMPRINT for the main task;

   CMPRINT1 to CMPRINT9 for the first nine subtasks;

   CMPRIN10 to CMPRIN*nn* for the next two-digit numbers of subtask, *nn*=NTASKS-1.

4. If the keyword subparameter TRACE of profile parameter RPC or parameter macro NTRPC is set, the trace facility writes to Printer 10.

   You must define the following DD card names:

   CMPRT10 for the main task;

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

Example:

```
//NATRPC   JOB  CLASS=K,MSGCLASS=X
//NATSTEP  EXEC PGM=NATOS,REGION=8M
//steplib  DD   DISP=SHR,DSN=SAG.NAT.LOAD
//         DD   DISP=SHR,DSN=SAG.EXX.LOAD
//CMPRMIN  DD   *
IM=D,MADIO=0,MT=0,OBJIN=R,AUTO=OFF,MAXCL=0,ID=',',INTENS=1,
PRINT=((10),AM=STD)
/*
//SYSUDUMP DD   SYSOUT=X
//CMPRT10  DD   SYSOUT=X
//CMPRT101 DD   SYSOUT=X
//CMPRT102 DD   SYSOUT=X
//CMPRT103 DD   SYSOUT=X
//CMPRINT  DD   SYSOUT=X
//CMPRINT1 DD   SYSOUT=X
//CMPRINT2 DD   SYSOUT=X
//CMPRINT3 DD   SYSOUT=X
/*
```

Trace File Handling under CICS

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

Examples:

Natural dynamic profile definition:

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

CICS definition:

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

CICS Startup JCL:

```
RPCTRACE DD    SYSOUT=*
```

Trace File Handling in z/VSE Batch Mode

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

Example:

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

where *vrs* stands for version, release, system maintenance level.

Trace File Handling in BS2000/OSD Batch Mode

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

Example:

```
/.NATRPC      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 NATBS2
MADIO=0,IM=D,ID=',',PRINT=((10),AM=STD)
```

Trace File Handling for UNIX and OpenVMS Environments

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

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

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

Example for UNIX:

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

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

Example for OpenVMS:

```
nattmp:filename
```

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

Trace File Handling for Windows

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

1. **Reports**

   Assign the logical device LPT10 to your Report Number 10.

2. **Devices**

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

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

   >>*filename*

Retrieving Runtime Settings of a Server

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

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

To make use of `USR4010N`

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

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

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

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

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

   See also the Syntax Description of the CALLNAT statement.

4. If RPC parameter AUTORPC=OFF, copy the stub USR4010**X** to the client environment.

   If RPC parameter AUTORPC=ON, the API must not be available to the client environment, otherwise the API would be called locally.

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

Setting/Getting Parameters for EntireX

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

- Compression level
- Encryption level

To make use of USR4009N

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

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

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

3. The interface can be called in two ways:

   1. From within a program:

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

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

      Examples:

      ```
      USR4009P P,C,ENCRYPTION-LEVEL=1
      USR4009P P,C,,2
      USR4009P P,C,ENCRYPTION-LEVEL=1,COMPRESSLEVEL=6
      ```

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

Notes:

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

Handling Errors

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

Remote Error Handling

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

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

Note:
If keyword subparameter `AUTORPC` of profile parameter `RPC` or parameter macro `NTRPC` 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

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

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

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

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

Important 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 statement block must not be changed.

User Exits before and after Service Execution

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

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

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

For both exits, the following rules apply:

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

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

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

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

  The use of the dynamic variable allows us to implement future extensions of the passed information without causing problems with existing user written exits.

- The exit is also called inside a conversation.
- The Natural RPC server does not intercept unhandled errors in the exit. If an unhandled error occurs in the exit, the error is propagated to the client.

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

Example for `NATRPC02`:

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