The RFC API

HELP.BCFESDE4

Release 4.6C

Copyright

Icons

Icon	Meaning
⚠	Caution
💬	Example
💡	Note
🧭	Recommendation
SYN	Syntax

Contents

The RFC API

Introduction to the RFC API

This documentation describes how to use RFC (Remote Function Call) from outside an SAP System, that is, how to write your own RFC functions or to call a function module in an R/2 or R/3 System.

The following topics contain basic information:

RFC with External Systems

In the SAP System, the ability to call remote functions is provided by the Remote Function Call (RFC) interface. This interface allows for remote calls between two SAP Systems (R/3 as of Release 2.1 and R/2 as of Release 5.0D), or between an SAP System and a non-SAP system.

The present section describes how to write RFC partner programs that run in non-SAP Systems.

If you are writing RFC programs in an SAP System, see RFC-Programming in ABAP [Ext.].

Client and Server Programs

RFC programs for non-SAP Systems can function as either the caller or the called program in an RFC communication.

There are two kinds of RFC programs: RFC client and RFC server programs:
The **RFC client** is the instance that calls up the RFC to execute the function which is provided by an **RFC server**. In the following, the functions that can be executed remotely will be called **RFC functions**, and the functions provided by the RFC API will be called **RFC calls**.

How to implement external RFC programs

You have two options for implementing external RFC programs:

- Use programs generated by the RFC Interface Generator (see The RFC Generator [Ext.].

  These are stub programs you can install on your workstation or PC to call up SAP function modules. The RFC Interface Generator in the SAP System lets you generate the stubs and download them to your machine.

  RFC stubs are written in C, and are stored either in libraries (in R/3-based UNIX systems) or in DLL's ("dynamic-link libraries" in Windows systems). With dynamic-link libraries, you can call the stubs from any programming language whose compiler offers DLL options.

  With the RFC Interface Generator you can also create example programs that show how to call an RFC stub. These programs can serve as a basis for programming your own application.

- Write your own RFC partner program

  You can write an RFC partner that makes (or receives) the remote call directly. This program can call up any SAP function module or be called by any ABAP program. You must write the program in C.

Both methods above use the RFC API.

What is the RFC API?

The SAP System provides the RFC API (Remote Function Call Application Programming Interface) that you install on non-SAP systems to help you implement RFC partner programs. The RFC API is a set of C-language routines that perform certain communications tasks for you.

The RFC API supports several external systems, such as OS/2, Windows, WindowsNT and Windows95, and all R/3-based UNIX platforms and makes it possible to use the RFC functionality between an SAP System and a C program on the above platforms. It is of no significance whether the remote function is provided in an SAP System or in a C program.

For each supported platform, there is an RFC SDK including the RFC library specific for each of these platforms, SAP RFC header files and some sample RFC programs.

The RFC API is always required

Both methods for implementing RFC programs use the RFC API:

- RFC programs generated by the RFC Interface Generator use API routines to call an SAP function module. In addition, the application you write (that calls the RFC stub program) must also use API routines to establish a connection with the SAP System, prepare table parameter structures, and so on.
- User-created programs (if you write your own) must likewise use API routines. Your program must perform all the same communication tasks as an automatically-generated stub and its caller.

For information about the RFC API components you need for your RFC projects, see Contents of the RFC SDK [Page 14].

Technical Requirements

External Systems

External systems must support TCP/IP.

- **OS/2:** TCP/IP for OS/2 from IBM.
- **Windows 3.1/3.11:** All TCP/IP products that support the socket interface.
- **Windows NT/95:** Microsoft standard.
- **UNIX platforms:** Manufacturer’s standard.

The RFCSDK for the respective platforms contains the following libraries and includes:

• **saprfc.h**	This include file contains all data types and structures required and the prototypes (declarations) of the RFC calls.	
• **sapitab.h**	This include file contains all the RFC calls required to manipulate internal tables	
• **librfc**	Depending on the platform, the following libraries are required:	
	OS/2:	**librfc.dll** and **librfc.lib** for Compile/Link
	Windows 3.1/3.11:	**librfc16.dll, librfc2.dll**, **librfc3.dll, librfc4.dll**
		and **librfc5.dll** and **librfc16.lib** for Compile/Link
	Windows NT/95:	**librfc32.dll** and **librfc32.lib** for Compile/Link
	UNIX-Platforms:	**librfc.a**

SAP R/3 Systems

For RFC between external systems and R/3, there are no specific requirements in the R/3 System, except that the R/3 System has to be at least Release 2.1.

Contrary to this, **RFC between SAP R/2 in an IBM environment and SAP R/3 or external systems** requires an SAP gateway to run on a machine that supports the SNA LU6.2 protocol for the IBM host. The SNA product must also be installed on this machine, and the SAP gateway must be operable with this product. This is necessary, because some SNA products are not compatible on the same machines.

The following SNA products are currently supported:

- SNA services or SNA server on IBM-AIX machines
- SNAplusLink on HP-UX machines

- Communication Manager on OS/2
- SNA server on Windows NT
- SNALink SNA peer-to-peer 8.0 on SUN machines
- TRANSIT-SERVER and TRANSIT-CPIC on SIEMENS-SINIX-machines.

SAP R/2 Systems

- **IBM host (CICS)**: Release 5.0D with the following components:
  - 082

    Communication via Remote Function Call (RFC)
  - 153

    SAP Intersystem Communication
  - 080

    Host communication with DOS, OS/2
  - or 081

    Host communication with other LU6.2 systems
- **IBM host (IMS)**: Probably not before IMS >= 4.1 with MVS/APPC
- **SNI host**: Release 5.0D with the following components:
  - 082

    Communication via Remote Function Call (RFC)
  - 153

    SAP Intersystem Communication
  - 083

    Host communication via TCP/IP (BS2000)

Contents of the RFC SDK

After the Remote Function Call Software Development Kit is installed, the following directory structures and items will be available:

- **.../rfcsdk/bin** contains the executables of all sample programs
  - sapinfo.exe

    RFC client program which receives system information from the SAP System
  - startrfc.exe

    RFC client program which can call any function module in the SAP System
  - rfcexec.exe

    RFC server program which can be started from the SAP System for file and pipe access
  - rfc2abap.exe

    RFC client program which loads and/or just starts an ABAP program in the R/3 System
  - srfctest.exe

    RFC client program which provides both a connection and a performance test
  - srfcserv.exe

    RFC server program which provides both a connection and a performance test
  - trfctest.exe

    Sample RFC client program for transactional RFC which transfers data in an internal table to the R/3 System.

    □

    This program is **not** available with a 16-bit RFC library on Windows.
  - trfcserv.exe

    Sample RFC server program for transactional RFC which receives data in an internal table from the R/3 System.

    □

    This program is **not** available with a 16-bit RFC library on Windows.
- **.../rfcsdk/include** contains all includes-files
  - saprfc.h

    Header file for general RFC API
  - sapitab.h

    Header file for working with SAP internal tables
  - rfcsi.h

    Header file for program sapinfo.c

  - srfctest.h

    Header file for program srfctest.c

  - srfcserv.h

    Header file for program srfcserv.c

  - trfctest.h

    Header file for program trfctest.c

    □

    This file is **not** available with a 16-bit RFC library on Windows.

  - trfcserv.h

    Header file for program trfcserv.c

    □

    This program is **not** available with a 16-bit RFC library on Windows.

- **.../rfcsdk/lib** contains the specific RFC library
  - librfc.a

    RFC library for supported UNIX platforms

  - or librfc.dll

    RFC library for OS/2 (>= Release 2.1),

    librfc.lib

    import library to link RFC programs

  - or librfc32.dll

    32-bit RFC library for Windows NT and Windows 95

    librfc32.lib

    import library to link RFC programs

  - or librfc16.dll, librfc2.dll, librfc3.dll, librfc4.dll, librfc5.dll

    16-bit RFC library for Windows

    librfc16.lib

    import library to link RFC programs

- **.../rfcsdk/text** contains the source code of all sample programs
  - sapinfo.c

    RFC client program which receives system information from an SAP System

  - startrfc.c

    RFC client program which can call any function module in the SAP System

- rfcexec.c

  RFC server program which can be started from the SAP System for file and pipe access

- rfc2abap.c

  RFC client program which loads and/or just starts an ABAP program in the R/3 System

- srfctest.c

  RFC client program which provides both a connection and a performance test

- srfcserv.c

  RFC server program which provides both a connection and a performance test

- trfctest.c

  Sample RFC client program for transactional RFC which transfers data in an internal table to the R/3 System.

  □

  This program is **not** available with a 16-bit RFC library on Windows.

- trfcserv.c

  Sample RFC server program for transactional RFC which receives data in an internal table from the R/3 System.

  □

  This program is **not** available with a 16-bit RFC library on Windows.

Compiling and Linking RFC Programs

In general, you have to use an ANSI C compatible C-compiler and set the include and library search path to the installed RFC SDK include and lib directory.

On some platforms you also have to link the TCP/IP socket libraries explicitly.

Assume.../rfcsdk is the root directory of the unpacked RFC SDK. You must use the following compile/link syntax for the program *sapinfo.c* on different platforms.

- HP-UX:

  ```
  cc -Ae -I.../rfcsdk/include -L.../rfcsdk/lib sapinfo.c librfc.a
  ```

- AIX (RS/6000):

  ```
  ccc -I.../rfcsdk/include -L.../rfcsdk/lib sapinfo.c librfc.a
  ```

- SINIX (RM600):

  ```
  /opt/C/bin/cc -I.../rfcsdk/include -L.../rfcsdk/lib sapinfo.c
  librfc.a -lsocket -lnsl -lusc
  ```

  □

  You ought to use the above compiler because the librfc.a is also compiled with it.

- DEC Alpha AXP:

  ```
  cc -std1 -unsigne -DA_OSF -I.../rfcsdk/include -L.../rfcsdk/lib
  sapinfo.c librfc.a
  ```

- SUN (SunPro):

  ```
  /opt/SUNWspro/bin/cc -Xc -xcg92 -I.../rfcsdk/include -
  L.../rfcsdk/lib sapinfo.c librfc.a -lsocket -lnsl
  ```

  □

  You ought to use the above compiler because the librfc.a is also compiled with it.

- WINDOWS with a 16-bit C-compiler:

  ```
  cl  /nologo /Gs /G2 /W4 /AL /D "_DOS" /Od /D "_DEBUG" /Mq
  /Fesapinfo.exe sapinfo.c librfc16.lib /link
  ```

- WINDOWS with a 32-bit C-compiler:

  ```
  cl  -nologo -Od -G5 -Z7 -Gs -W3 -J -D_X86_ -DWIN32 /MT /FR -
  Fesapinfo.exe sapinfo.c librfc32.lib
  ```

- OS/2 2.1 and higher:

  ```
  icc -Gm+ -Ss+ -Ti -J- -DOS2  B”/E /NOI /ST:0x8000” sapinfo.c
  librfc.lib (sample for IBM VisualAge C++ V3)
  ```

Programming with the RFC API

You must keep the following guidelines in mind when writing RFC programs.

The programs you write can call or be called by ABAP programs of an SAP System.

Establishing an RFC Connection from an R/2 System [Page 55]

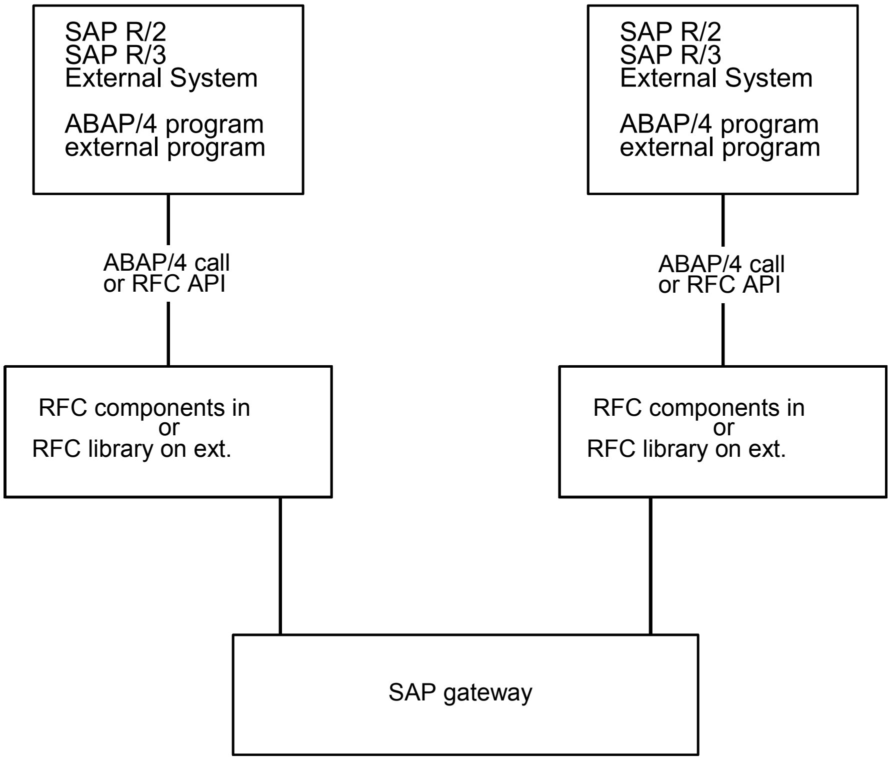
Establishing an RFC Connection from an R/3 System [Page 58]

Passing Parameters [Page 61]

Writing an RFC Function in C [Page 63]

Using Multiple-Client Server Programs [Page 64]

Technical Description



Getting Connected

RFC and programming information for RFC client/server programs can be summarized as follows:

- An RFC connection is always initiated by an RFC client program.
- An RFC connection is always built up in two steps:
  - Connection from an RFC client to the SAP gateway
  - Connection from the SAP gateway to an RFC server
- There are different ways of starting or connecting to an RFC server program:
  - An RFC server program can be started by the SAP gateway, by the currently running SAPGUI or by the currently running application server.
  - From R/3 Release 3.0C onwards, you can register an RFC server program at an SAP gateway. The program waits for RFC requests from SAP Systems.

For program start-up options while establishing an RFC connection, see RFC Client Programs [Page 25] and RFC Server Programs [Page 46].

Supported Data and Transfer

Data format

Only **homogenous** structures or tables are supported. They must consist of character fields only (ABAP-types **C**, **D**, **T**, **N**) or fields which will **not** be converted (ABAP-types **X** or **P**). Integer (ABAP-type **I**) or float (ABAP-type **F**) fields can only be transfered as single fields (IMPORT/EXPORT parameters).

Data Compression

From R/3 Release 3.0 onwards, RFC data will be compressed before sending, if both client and server system are capable of this. Otherwise, only the blank spaces at the end of a table line will be truncated.

Data Conversion

The SAP System or the RFC library will convert received data in its own code page if the two code pages are not the same.

Some standard code pages (such as 1100, 0100, etc.) are already implemented in the RFC library. It can also use a conversion file defined by the environment variable PATH_TO_CODEPAGE. The files can be created and downloaded into a directory accessible from the currently running R/3 application server with transaction SM59.

Limitations of Data in one RFC Function

RFC library delivered with R/3 Release	< 2.1K/2.2E	4 MB
	>= 2.1K/2.2E	64 MB
	>= 3.0	none

Special Destinations in ABAP (R/3 or R/2)

- Destination **BACK**:

  During execution of an RFC function, you can use the destination **BACK** to call another RFC function in the client system (SAP R/2 or R/3) or in the RFC client program (external program via RFC API) using the same RFC connection.

  □

  This destination is **not** available in R/2 Systems.

- Destination **NONE** for calls from SAP R/2 or R/3:

  In this case, the server system is the same SAP System (R/2 or R/3) as the client system.

  – In R/3, it will be treated as a remote function call even if the RFC serveris the same application server (RFC data will be transferred via the SAP gateway).

  – In R/2, however, it will be treated as a “local" CALL FUNCTION.

- Destination **SPACE**

□

This destination is only possible as of R/3 Release 3.0.

This destination is treated like a "local" CALL FUNCTION.

Basic Functionality

The RFC API allows you to call ABAP function modules from C programs as well as receiving call requests issued from an ABAP program by the CALL FUNCTION interface.

The RFC API consists of three main parts:

- the include file saprfc.h

  This file contains data type and structure definitions as well as the prototypes (declarations) of the functions forming the API.

- the include file sapitab.h

  This file defines an interface for manipulating ABAP internal tables.

- the library librfc.a (librfc32.dll, etc)

  This library contains the functions of the API.

Components of the RFC API

The SAP System's RFC API (application programming interface) can be installed on external systems and used to implement RFC programs. The API consists of library routines you call to communicate with an RFC partner. These routines (implemented in C) perform the communications calls and other tasks needed to handle either the caller or receiver side of the communication. (The RFC interface in the SAP System handles the other side.)

SAP's Remote Function Call API consists of:

- Include file *saprfc.h* (data and function definitions)
- An API function library

  This library is named *librfc.a, librfc.dll, librfc.lib, librfc32.dll, librfc32.lib, librfc16.dll, librfc2.dll, librfc3.dll, librfc4.dll, librfc5.dll, librfc16.lib*, depending on your system. Routines in the RFC API are described in the following:

  RFC Client Function Reference [Page 127]

  RFC Server Function Reference [Page 134]

  Table-Handling Function Reference [Page 147]

  Transactional Function Reference [Page 161]

  Extended Function Reference [Page 171]

  □

  For many of these routines, there are extended versions that should be used by Basic programmers. See Extended Function Reference [Page 171] for more information.

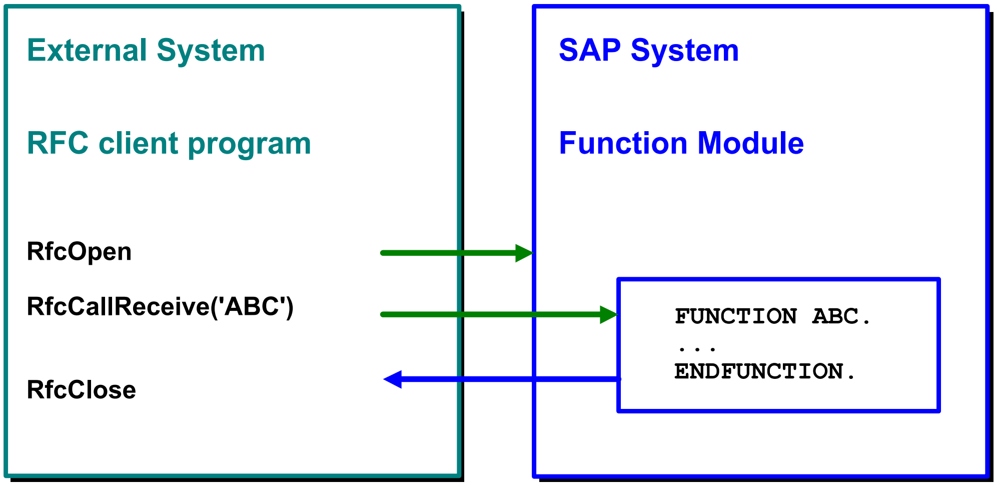
  In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __pascal __far.
  Defined in: SAPITAB.H

RFC Client Programs

The following topics are available:

Introduction to RFC Client Programs

All RFC client programs have to establish an RFC connection to an SAP System:



Possible Logon Users

- Logon to an R/3 System is possible with a DIALOG user or a CPIC user.
- Logon to an R/2 System is only possible with a CPIC user.

Getting Connected with the RFC Library before Release 3.0C

There are different ways of and conditions for establishing an RFC connection to an SAP System (R/2 or R/3):

1. using a **local sideinfo** file
2. using no **local sideinfo** file, but a **sideinfo** file for the SAP gateway
3. without using any **sideinfo** file

A sideinfo file is needed for communication via CPIC.

For RFC connections to an R/2 System in IBM environments, a sideinfo file for the SAP gateway for communication via SNA LU6.2 protocol is always required.

RFC connections in SNI environments require the SAP gateway to run on the BS 2000 host. Apart from that, there are no differences for establishing an RFC connection to an R/2 System (SNI) or to an R/3 System.

Getting Connected with the RFC Library as of Release 3.0C

An RFC connection can be established via an entry in the saprfc.ini file. Using this feature, you need not have any sideinfo file except for connecting to an R/2 System (IBM).

- The **saprfc.ini** file has the same meaning as the **sideinfo** file, but all RFC features, such as RFC with SAPGUI, ABAP-debug,... can be defined in that file.

- Moreover, most new RFC features to be developed in the future can be used without changing the RFC client program sources.

The sideinfo file is needed for communication via CPI-C and includes some parameters necessary for establishing a CPI-C connection.

If the local sideinfo is not used, all the parameters required must be passed on to the RFC library using the call **RfcConnArgv** before the **RfcOpen** is submitted or defined in the parameter RFC_OPTIONS of RfcOpen.

An RFC function is called by RfcCall [Page 128]. The function RfcCall returns after the call request is sent.

The function RfcReceive [Page 132] allows to receive the answer to an RFC call and must be called after RfcCall was issued. The function RfcReceive waits until the answer is received.

Moreover, there is a function which can issue an RFC call synchronously: RfcCallReceive [Page 130] waits until the returned answer will be received.

Typical examples of RFC client programs are sapinfo.c, startrfc.c and srfctest.c which are included in the RFC SDK.

Establishing an RFC Connection to R/2 Systems

Establishing an RFC connection to an R/2 System in an SNI environment is similar to an R/3 System, except that the SAP gateway must run on a BS 2000 host.

Therefore, the following sections contain descriptions on establishing RFC connections to R/2 Systems in IBM environments.

□

A sideinfo file for the SAP gateway is always required.

The RFC client program can work

- with a local sideinfo file (see Programming Example of Working With Local sideinfo File [Page 29])
- without any local sideinfo file (see Programming Example of Working Without Local sideinfo File [Page 31])
- with the saprfc.ini file (see Programming Example of Working With saprfc.ini File [Page 32])

□

The following sections describe only how to establish an RFC connection to R/2 Systems with ConnArgv. For information about how to establish an RFC connection to R/2 Systems in the subroutine RfcConnect, see ‘srfctest.c’ in the RFC SDK.

Programming Example of Working With Local sideinfo File

```
RFC_OPTIONS          rfc_opt;     /* Parameter for RFC connection*/

RfcEnvironment(...);              /* Install error handling function
                     */

rfc_opt.mode         = RFC_MODE_CPIC;  */ RFC to SAP R/2      */


rfc_opt.destination  = "K50";     /* Entry with this destination */
                                  /* must be in local sideinfo   */
                                  /* file and for SAP gateway    */

rfc_opt.connopt      = NULL       /* Connect parameters in       */
                                  /* sideinfo file               */

rfc_opt.client       = "000"      /* Client in SAP R/2           */

rfc_opt.user         = "RFCTEST"  /* CPIC user                   */

rfc_opt.password     = "SECRET"   /* Password                    */

rfc_opt.language     = "E"        /* Logon language              */

rfc_opt.trace        = 0          /* No RFC trace                */

rfc_handle = RfcOpen(&rfc_opt);   /* Open RFC connection         */

...
```

Local sideinfo file

The local sideinfo file must either be in the same directory as the RFC client program or be defined with its full path and file name using the environment variable SIDE_INFO.

□

Windows: `set SIDE_INFO=d:\rfctest\sideinfo`

The entry for the above test can be defined as follows:

```
DEST=K50
GWHOST=is0001
GWSERV=sapgw00
PROTOCOL=C
```

sideinfo file for the SAP gateway running on IBM-AIX machines

```
DEST=K50_1
LU=K50T1
TP=X1SA

DEST=K50_2
LU=K50T2
TP=X1SA
```

Programming Example of Working With Local sideinfo File

```
DEST=K50_3
LU=K50T3
TP=X1SA
```

The extension _n(n=1,2,,...) makes it possible for the SAP gateway to establish a LU6.2 connection to the R/2 System via different SNA connections.

sideinfo file for the SAP gateway running on HP-UX machines

```
DEST=K50
LU=K50T
TP=X1SA
LOCAL_LU=LU0001
MODE_NAME=LU62SAP1
```

- Specifications about LOCAL_LU are not required if at least one local LU is defined in the **LU pool** in the SNA-configuration on HP-UX.
- The LU6.2-mode ’LU62SAP1’ defined as MODE_NAME above must be defined in the SNA-configuration on HP-UX and in IBM-Host.

Programming Example of Working Without Local sideinfo File

```
RFC_OPTIONS              rfc_opt;  /*Parameter for RFC connection*/
RFC_CONNOPT_CPIC         rfc_connopt_cpic;
                                   /*Specific param. for RFC to R/2
                         */
RfcEnvironment(...); /*            Install error handling function*/


rfc_connopt_cpic.gateway_host = “iw1009”
                                   /*host name of the SAP gateway*/
rfc_connopt_cpic.gateway_service = “sapgw00”
                                   /*service no. of the SAP gateway
                         */
rfc_opt.mode = RFC_MODE_CPIC;      /*RFC to SAP R/2              */
rfc_opt.destination = “K50”        /*Entry with this destination */
                                   /*be in  the sideinfofile for */
                                   /*the SAP gateway             */
rfc_opt.connopt = rfc_connopt_cpic;
                                   /*includes connect parameters */
rfc_opt.client = “000”;            /*Client in SAP R/2           */
rfc_opt.user = “RFCTEST”;          /*CPIC user                   */
rfc_opt.password = “SECRET”;       /*Password                    */
rfc_opt.language = “E”;            /*Language                    */
rfc_opt.trace = 0;                 /*No RFC trace                */
rfc_handle = RfcOpen(&rfc_opt);    /*Open RFC connection         */
...
```

You can also issue **RfcConnArgv** or **RfcConnArgv3** before **RfcOpen** to pass on the information about the SAP gateway to the RFC library. The **RfcOpenExt** and **RfcOpenExtV3** are for non-C programs, such as Visual Basic programs. For more details, see the delivered ‘sapinfo.c’ and ‘srfctest.c’.

Programming Example of Working With saprfc.ini File

```
RFC_OPTIONS             rfc_opt;      /* Parameter for RFC connection*/
RfcEnvironment(...); /*             Install error handling function*/
rfc_opt.destination   = “K50”;      /* Destination in saprfc.ini   */
                                    /* and in sideinfo for the SAP */
                                    /* gateway                     */
rfc_opt.connopt       = NULL;       /* Connect parameters in       */
                                    /* saprfc.ini file             */
rfc_opt.client        = “000”;      /* Client in SAP R/2           */
rfc_opt.user          = “RFCTEST”;  /* CPIC user                   */
rfc_opt.password      = “SECRET”;   /* Password                    */
rfc_opt.language      = “E”;        /* Logon language              */
rfc_opt.trace         = 0;          /* No RFC trace                */
rfc_handle = RfcOpen(&rfc_opt);     /* Open RFC connection         */
...
```

Entry in saprfc.ini file

The saprfc.ini file must be in the same directory as the RFC client program or be defined with its path and file name by the environment variable RFC_INI.

Windows: `set RFC_INI=d:\rfctest\saprfc.ini`

The sideinfo entry for the above test could be as follows:

```
DEST=K50
TYPE=2
GWHOST=is0001
GWSERV=sapgw00
```

For more information on the saprfc.ini file, see The SAPRFC.INI File [Page 66].

Establishing an RFC Connection to R/3 Systems

Contrary to SAP R/2 (IBM), the SAP gateway does not need a specific entry in its sideinfo file if all needed parameters are specified in the local sideinfo file or if they are defined in the RFC client programs.

An RFC client program can work

- with a local sideinfo file (see Programming Example of Working With Local sideinfo file [Page 34])
- without any local sideinfo file (see Programming Example of Working Without Local sideinfo File [Page 35])
- with the saprfc.ini file (see Programming Example of Working With saprfc.ini File [Page 37])

Programming Example of Working With Local sideinfo file

```
RFC_OPTIONS          rfc_opt;     /* Parameter for RFC connection*/

RfcEnvironment(...);              /* Install error handling function
                     */

rfc_opt.mode         = RFC_MODE_R3ONLY;*/ RFC to R/3 System   */

rfc_opt.destination  = “BIN”;     /* Destination in local sideinfo*/

rfc_opt.connopt      = NULL;      /* Connect parameters in       */
                                  /* sideinfo file               */

rfc_opt.client       = “000”;     /* Client in SAP R/3           */

rfc_opt.user         = “RFCTEST”; /* CPIC or dialog user         */

rfc_opt.password     = “SECRET”;  /* Password                    */

rfc_opt.language     = “E”;       /* Logon language              */

rfc_opt.trace        = 0;         /* No RFC trace                */

rfc_handle = RfcOpen(&rfc_opt);   /* Open RFC connection         */

...
```

Local sideinfo file:

The local sideinfo file must be in the same directory as the RFC client program or must be defined with its path and file name by the environment variable SIDE_INFO.

□

Windows: `set SIDE_INFO=d:\rfctest\sideinfo`

The sideinfo entry for the above test could be as follows:

```
DEST=BIN
GWHOST=hs0311
GWSERV=sapgw53
PROTOCOL=I
LU=hs0311
TP=sapdp53
```

Programming Example of Working Without Local sideinfo File

```
RFC_OPTIONS                   rfc_opt;
                                  /*Parameter for RFC connection*/
RFC_CONNOPT_VERSION_3         rfc_connopt_version_3;
                                  /*Spec. parameter for R/3 System
                              */
RfcEnvironment(...);              /*Install error handling function
                                  */

if (use_load_balancing)
{
      rfc_connopt_version_3.use_load_balancing = 1;
      rfc_connopt_version-3.lb_host = “hs0311”;
                                  /*host name of message server*/
      rfc_connopt_version_3.lb_system_name = “BIN”;
                                  /*Name of the R/3 System      */
      rfc_connopt_version_3.lb_group = “PUBLIC”;
                                  /*Application server group    */
}
else
{
      rfc_connopt_version_3.use_load_balancing = 0;
                                  /*Connect to spec. appl. server*/
      rfc_connopt_version_3.hostname = “hs0011”;
                                  /*Host name of an appl. server*/
      rfc_connopt_version_3.sysnr = “53”;
                                  /*System number of the R/3 System
      */
}

if (use_sapgui)
      rfc_connopt_version_3.use_sapgui = 1;
                                  /*Work with SAPGUI            */
else

      rfc_connopt_version_3.use_sapgui = 0;       /*or not      */

rfc_opt.mode = RFC_MODE_VERSION_3;                /*RFC to R/3  */

rfc_opt.destination = “BIN”;      /*Destination in local sideinfo*/
rfc_opt.connopt = rfc_connopt_version_3;
                                  /*Includes connect parameters*/
rfc_opt.client = “000”;           /*Client in SAP R/3           */
rfc_opt.user = “RFCTEST”;         /*CPIC or dialog user         */
rfc_opt.password = “SECRET”;      /*Password                    */
rfc_opt.language = “E”;           /*Language                    */

if (ABAP_debug)
      rfc_opt.trace = ‘D’;        /*Working with ABAP-debugger  */
```

```
else
      rfc_opt.trace = 0;          /* or not                    */

if (rfc_trace)
rfc_opt.trace = rfc_opt.trace + 1;/* 0/1: Rfc-trace ON/OFF      */
                                  /* 'D': ABAP-debug            */
                                  /* 'E': ABAP-debug plus RFC trace
      */

rfc_handle = RfcOpen(&rfc_opt);

...
```

You can also issue **RfcConnArgv** or **RfcConnArgv3** before **RfcOpen** to pass on the information about the SAP gateway to the RFC library. The **RfcOpenExt** and **RfcOpenExtV3** are for non-C programs, such as Visual Basic programs.

For more details, see the *sapinfo.c* and *srfctest.c* in the RFC SDK.

Programming Example of Working With saprfc.ini File

```
RFC_OPTIONS          rfc_opt;     /* Parameter for RFC connection*/

RfcEnvironment(...);              /* Install error handling function
                     */

rfc_opt.mode         = RFC_MODE_PARAMETER;
                                  */ saprfc.ini file is used    */

rfc_opt.destination  = “BIN”;     /* Destination in saprfc.ini  */

rfc_opt.connopt      = NULL;      /* Connect parameters in      */
                                  /* saprfc.ini file            */

rfc_opt.client       = “000”;     /* Client in SAP R/2          */
rfc_opt.user         = “RFCTEST”; /* CPIC or dialog user        */
rfc_opt.password     = “SECRET”;  /* Password                   */
rfc_opt.language     = “E”;       /* Logon language             */
rfc_opt.trace        = 0;         /* No RFC trace               */

rfc_handle = RfcOpen(&rfc_opt);   /* Open RFC connection        */

...
```

Entry in saprfc.ini file

The saprfc.ini file must be in the same directory as the RFC client program, or it can be defined with its path and file name by the environment variable RFC_INI

□

Windows: `set RFC_INI=d:\rfctest\saprfc.ini`

The following example contains a reference to a specific application server:

□

```
DEST=BIN
TYPE=A
ASHOST=hs0311
sysnr=53
RFC_TRACE=0
ABAP_DEBUG=1
USE_SAPGUI=1
```

The following example illustrates the load balancing feature available as of R/3 Release 3.0:

□

```
DEST=BIN
TYPE=B
R3NAME=BIN
GROUP=PUBLIC
RFC_TRACE=0
```

Programming Example of Working With saprfc.ini File

```
ABAP_DEBUG=1
USE_SAPGUI=1
```

The new RFC features in Release 3.0C, such as RFC with SAPGUI or RFC with ABAP-debug, can be used via some more definitions in the saprfc.ini file not shown in the examples above. For more defails, see The SAPRFC.INI File [Page 66].

Load Balancing

From Release 3.0 onwards an RFC client program can call a function module in an R/3 System without specifying the application server for establishing the connection. In an R/3 System it is therefore possible to make use of load-balancing for RFC as well, a feature initially introduced for SAPGUI connections. This is done by means of the two functions RfcOpen [Page 126] or RfcConnect [Page 123], or, as of 4.0, by RfcOpenEx [Ext.]: The system first builds up a connection to the Message Server of the R/3 System and tries to find the application server with the least load (**LOAD BALANCING principle**). On the basis of this information, the RFC library internally builds up the connection to the selected application server.

This load balancing feature has the following advantages:

- The load in an R/3 System is distributed to different R/3 application servers. The RFC connection is always established to an application server with the least load.
- Using load balancing, the RFC server will be determined at run time from the application servers available. Therefore, RFC connections are independent of a specific application server. (In R/3 Release prior to 3.0, an RFC connection could only be established to a specific application server.)
- Only the host name of the R/3 message server and its port number are required in the *hosts* and *services* file. Information about the SAP gateway, application server, system number for RfcOpen or sideinfo entry and entries for the application server and SAP gateway are no longer required there.

Programming Examples

The following sections contain examples in short form only. For more details, see the RFC client programs *srfctest.c* or *sapinfo.c* in the RFC SDK.

RFC Client Program Transferring Internal Tables

```
RFC_TABLE   tables[1];            /* Work with one internal table*/

RfcEnvironment(...);              /* Install error handling function
                                  */

rfc_handle = RfcOpen(...);        /* Open RFC connection          */

if (rfc_handle == RFC_HANDLE_NULL)/* Check return code            */

{

  rfc_error_handling("RfcOpen");  /* Handle error and get error  */

  return 1;                       /* specification via RfcLastError
                                  */

}

tables[0].name = “ITAB1000";      /* Must fit with definition     */
                                  /* in SAP-FM                    */
tables[0].nlen     = 8;           /* Length of name               */
tables[0].type     = TYPEC;       /* Character only               */
tables[0].leng     = 1000;        /* Lenth of a table line        */
tables[0].itmode   = RFC_ITMODE_BYREFERENCE;  /* Recommended      */
tables[0].ithandle = ItCreate(...);    /* Allocate storage        */
                                  /* for internal table           */

if (rfc_rc != RFC_OK)             /* Check return code            */

{

  rfc_error_handling("ItCreated");/* Get specific error via       */
                                  /* RfcLastError                 */
  return 1;                       /* and handle error             */

}

fill_table(table[0].ithandle, 10); /* Fill internal table with */
                                  */ 10 lines of text             */

rfc_rc = RfcCallReceive(...);     /* Call up function module      */

if (rfc_rc != RFC_OK)             /* Check return code            */

{

  rfc_error_handling("RfcCallReceive"); /* Get specific error via*/
                                  /* RfcLastError                 */
  return 1;                       /* and handle error             */

}
```

```
ItDelete(...);                    /* Free storage of internal table
                                  */
RfcClose(...);                    /* Close RFC connection        */


/* Fill internal table with text as requested */
void fill_table(ITAB_H itab_h, int table_leng)
{
  int    linenr;                  /* Actual line number of a table*/
  int    lineleng;                /* Length of a table line      */
  char   *ptr;                    /* Pointer to a table line     */
  char   table_data[]="This is a test";    /* Text for test      */
if (table_leng == 0) return 0;    /* Table with no entry         */
lineleng = ItLeng(itab_h);        /* Determine length of a table */
                                  /* line                        */
  for (linenr = 1; linenr <= table_leng; linenr++)
                                  /*  Fill table as requested    */
  {
    ptr = (char *) ItAppLine(itab_h);
                                  /* Get address of next line    */
    if (ptr == NULL)              /* Check return code           */
    {                             /* Output error message and    */
      printf("\nMemory insufficient\n");   /* output error message
                                     */
      exit(1);                    /* and terminate program       */
    }
    memcpy(ptr, table_data, lineleng);  /*Transfer data to internal
                                     */                          /*
                                     table                       */
  }
  return;                         /* Back to main program        */
}
```

RFC Client Program Receiving Internal Tables

```
RFC_TABLE   tables[1];            /* Work with one internal table*/
RfcEnvironment(...);              /* Install error_handling function
                                  */
rfc_handle = RfcOpen(...);        /* Open RFC connection         */
if (rfc_handle == RFC_HANDLE_NULL) /* Check return code          */
```

```
{
  rfc_error_handling("RfcOpen");  /* Handle error and get error  */
  return 1;                       /* specification via RfcLastError
                                  */
}
tables[0].name   = “ITAB1000”;    /* Must fit with definition    */
                                  /* in SAP-FM                   */
tables[0].nlen     = 8;           /* Length of name              */
tables[0].type     = TYPEC;       /* Character only              */
tables[0].leng     = 1000;        /* Lenth of a table line       */
tables[0].itmode   = RFC_ITMODE_BYREFERENCE;  /* Recommended     */
tables[0].ithandle = ItCreate(...);   /* Allocate storage        */
                                  /* for internal table          */
if (rfc_rc != RFC_OK)             /* Check return code           */
{
  rfc_error_handling("ItCreated");/* Get specific error via      */
                                  /* RfcLastError                */
  return 1;                       /* and handle error            */
}
rfc_rc = RfcCallReceive(...);     /* Call up function module     */
if (rfc_rc != RFC_OK)             /* Check return code           */
{
  rfc_error_handling("RfcCallReceive"); /* Get specific error via*/
                                  /* RfcLastError                */
  return 1;                       /* and handle error            */
}
display_table(table[0].ithandle); /* output received internal */
                                  /* table on screen             */
ItDelete(...);                    /* Free storage of internal table
                                  */
RfcClose(...);                    /* Close RFC connection        */


/* Output received internal table on screen */
void display_table(ITAB_H itab_h)
{
  int       linenr;               /* Actual line number of a table*/
  int       lineleng;             /* Length of a table line      */
  char      ptr;                  /* Pointer to a table line     */
  char      table_data[8193];     /* Max. length of an internal  */
                                  /* table (8192 B)              */
lineleng = ItLeng(itab_h);        /* Get length of a table line */
  for (linenr = 1; ; linenr++)    /* Loop at internal table     */
  {
    ptr = (char *) ItGetLine(itab_h);
                                  /* Get address of next line   */
    if (ptr == NULL) break;       /* NULL: End of table reached */
    memcpy(table_data, ptr, lineleng);
                                  /* Read a table line into buffer*/
    table_data[lineleng] = '\0';  /* Set string end in buffer for*/
                                  /* output                     */
    printf("'%s'\n", table_data); /* Output on screen           */
  }
  return;                         /* Back to main program       */
}
```

Remote Function Calls Using SAPGUI

Starting with R/3 Release 3.0C, it is possible for an RFC client program to call ABAP function modules which are using **'Dynpros'** or **SAP graphics**. This allows you to include normal SAP screens into your programs. Consequently, it is possible to call complete SAP transactions from external programs.

To use this functionality, a SAPGUI front end server (version 3.0C or later) must be installed on the external system where the RFC client program is running.

There are three methods to activate this new functionality:

1. Use RfcOpen

   You can call the function RfcOpen [Page 126] with the special mode RFC_MODE_VERSION_3 and then set the field **use_sapgui** to a non-zero value in the structure RFC_CONNOPT_VERSION_3 [Page 196].

2. Use the saprfc.ini file

   You can define an entry in the **saprfc.ini** file which includes all necessary connection parameters and the RFC-specific parameter USE_SAPGUI (=1) in the saprfc.ini file. An RFC client program can then issue the RfcOpen [Page 126] call with the mode RFC_MODE_PARAMETER and a destination pointed to the defined entry. See The SAPRFC.INI File [Page 66] for more details.

3. Use SYSTEM_ATTACH_GUI in R/3

   You can call the function module **RFC_ATTACH_GUI** (without any parameter) within the ABAP funtion module you want to use before using any SAPGUI functionality.

   □

   This method can be used without modifying the C code of the RFC client program.

   □

   - This feature is available on all supported UNIX platforms (Motif).
   - On Windows NT or Windows 95, it is only available with the 32-bit RFC library together with a 32-bit SAPGUI. In particular, this functionality is **not** supported for the 16-bit RFC library or for 16-bit SAPGUI (Windows 3.x, as well as the 16-bit subsystems of Windows NT and Windows 95).
   - Under Windows NT and Windows 95 the SAPGUI program and its DLLs and auxiliary programs can be installed anywhere. However, you must have started SAPGUI once before it can be started automatically by RFC, because the SAPGUI must register itself in the Windows registry.
   - On R/3-based UNIX systems, the SAPGUI program must be installed on the default 'SAP path' **/usr/sap/<system name>/SYS/exe/run**.

RFC Using the ABAP Debugger

As of R/3 Release 3.0C you can use the full functionality of the ABAP debugger when developing or testing an application using RFC.

To use this functionality, a 3.0C (or later) SAPGUI must be installed on the external system where the RFC client program is running.

An RFC client program can use this feature with one of the following options:

- **environment variable RFC_DEBUG**

  Set this environment variable to any value to enter debugging mode.

- **set -debug on the command line**

  If the RFC client program uses RfcConnArgv [Page 119] for scanning the command line, set **-debug** on the command line to enter debugging mode.

- **set the trace field to D or E**

  If you can modify the coding of the RFC client program you are using, you can set the trace field in the structure RFC_OPTIONS [Page 201] to the value D or E before calling RfcOpen [Page 126]:

  - D: debugging without activated RFC trace
  - E: debugging with activated RFC trace

- **use the saprfc.ini file**

  Define an entry in the **saprfc.ini** file which includes all necessary connection parameters and the RFC-specific parameter RFC_DEBUG (=1). An RFC client program can then issue the RfcOpen [Page 126] call with the mode RFC_MODE_PARAMETER and a destination pointed to this defined entry. See The SAPRFC.INI File [Page 66] for more details.

  - This feature is available on all supported UNIX platforms (Motif).
  - On Windows NT or Windows 95, it is only available with the 32-bit RFC library together with a 32-bit SAPGUI. In particular, this functionality is **not** supported for the 16-bit RFC library or for the 16-bit SAPGUI (Windows 3.x, as well as the 16-bit subsystems of Windows NT and Windows 95).
  - Under Windows NT and Windows 95 the SAPGUI program and its DLLs and auxiliary programs can be installed anywhere. However, you must have started SAPGUI once before it can be started automatically by RFC, because the SAPGUI must register itself in the Windows registry.
  - On R/3-based UNIX systems, the SAPGUI program must be installed on the default ‘SAP path’ **/usr/sap/<system name>/SYS/exe/run**.

RFC Server Programs

The following topics are available:

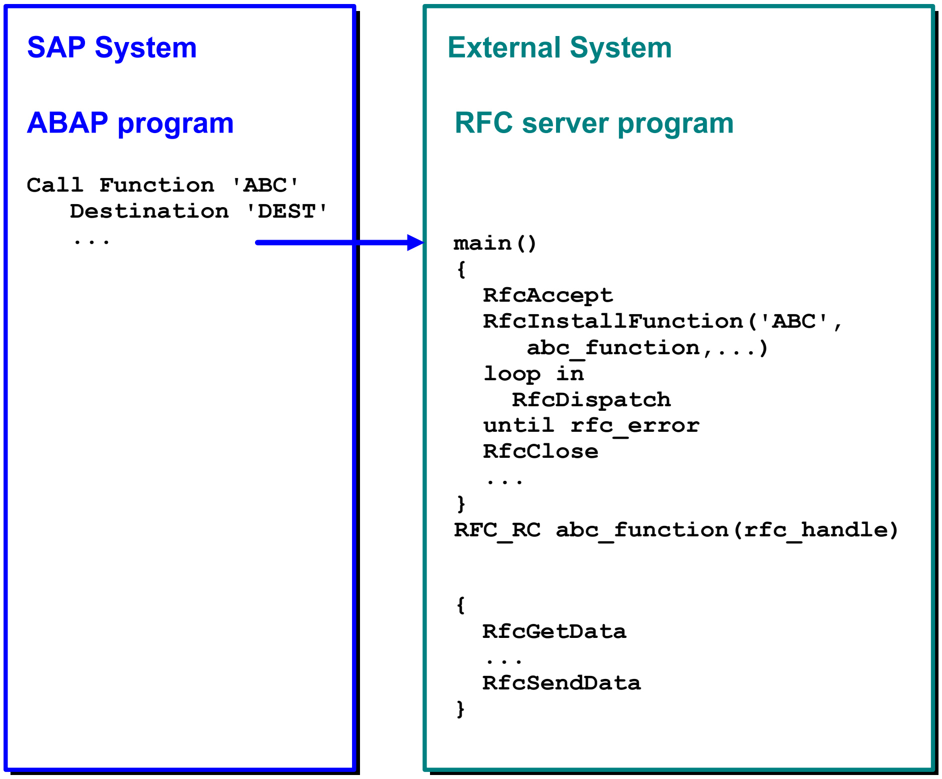
Introduction to RFC Server Programs

An RFC server program is a program which offers RFC functions to be called by ABAP programs.

The RFC API offers routines for implementing RFC server programs. After having started, the RFC server programs must inform the RFC library about all RFC functions which can be called within this server program. It can then wait for incoming call requests, and the RFC library will dispatch the requested calls (using RfcDispatch in a loop).

As an alternative, the RFC server program can use RfcGetName to identify the name of the required RFC function, and then it must dispatch itself to the supported function.

A typical server looks as follows:



SAP System

ABAP program

```
Call Function 'ABC'
   Destination 'DEST'
   ...
```

External System

RFC server program

```
main()
{
  RfcAccept
  RfcInstallFunction('ABC',
     abc_function,...)
  loop in
    RfcDispatch
  until rfc_error
  RfcClose
  ...
}
RFC_RC abc_function(rfc_handle)


{
  RfcGetData
  ...
  RfcSendData
}
```

Since the RFC server program **never** builds up a connection actively, the sideinfo file is **never** used.

There are two methods of receiving an RFC call. The most simple way of receiving an RFC call in an external program is to register a C function to be called when a call request is received. The function RfcInstallFunction [Page 140] registers a C function to be called when receiving the request for an RFC call. After RfcAccept [Page 116] or after receiving the return code RFC_CALL when calling RfcReceive [Page 132], the program must use RfcDispatch [Page 135] to internally

Introduction to RFC Server Programs

call the corresponding registered function. The return code of the registered function is again returned by RfcDispatch.

There are always some standard functions which are installed automatically. Apart from some internally-used functions, the function modules are as follows:

- RFC_PING

  This function does nothing by itself. You can use it to test the connection.

- RFC_SYSTEM_INFO

  This function returns some information about the library and its environment.

- RFC_DOCU

  This function returns the function documentation which was installed during the calls of RfcInstallFunction [Page 140].
  It is also possible to receive RFC calls directly. The function RfcGetName [Page 139] is used to get the name of the called function. The calling program has to determine the interface of the requested function module itself. It must then receive the parameters as within a function being installed via RfcInstallFunction.

Registering Server Programs with the SAP Gateway

Up to R/3 Release 3.0C, RFC server programs could only be started by an SAP gateway, either by the SAPGUI or by the currently running application server. As of R/3 Release 3.0C, an RFC server program can be registered with the SAP gateway and wait for incoming RFC call requests.

This new registering feature has the following features:

- An RFC server program registers itself under a **program-ID** at an SAP gateway and **not** for a specific SAP System (R/2 or R/3).
- If an RFC call request from any R/2 or R/3 System is transferred to this SAP gateway with the option **'connect to an already registered program'** with the same **program-ID**, it will be connected to this RFC server program.
- In **R/2,** this option requires the parameter setting **PROTOCOL=R** in the *sideinfo* file for the *gwhost* program.
- In **R/3**, the destination must be defined with transaction SM59, using connection type **T** and **Register Mode**. Moreover, this entry must contain information on the SAP gateway at which the RFC server program is registered.
- After having executed an RFC function, the RFC connection will be closed. If this RFC server program works with **RfcDispatch in a loop** (this procedure is recommended), it will be automatically registered again at the same SAP gateway under the same **program-ID** and can then wait for further RFC call requests from the same SAP System or from other SAP Systems.
- An RFC server program can be run on all Windows PCs with this new functionality. There are no restrictions for server programs on systems that do not support **remote shell**. In addition, you can now run RFC server programs easily in debugging mode.

How to Work with this Functionality

In order to use this new functionality, no changes in ABAP programs are needed. Only the destination in an ABAP program must have the new feature defined with transaction SM59, using connection type T and Register Mode.

There are two ways of working with this new registering functionality in external systems:

- **With a few additional parameters:**

  All correctly running RFC server programs can work with this new feature without changing the source code. They must only be started with a few additional parameters needed to connect to the SAP gateway.

- **Using the *saprfc.ini* file:**

  The RFC call *RfcAccept* must include a destination referring to an entry in the *saprfc.ini* file with type *R* and containing some parameters needed by the SAP gateway.

See also Using Multiple-Client Server Programs [Page 64].

Programming Examples

The following sections contain examples in short form only. For more details, see the RFC server programs *srfcserv.c* or *rfcexec.c* in the RFC SDK.

RFC Server Program Working With RfcDispatch [Page 51]

RFC Server Program Working With RfcDispatch and RfcListen [Page 53]

RFC Server Programs Sending/Receiving Internal Tables [Page 54]

RFC Server Program Working With RfcDispatch

```
/* main program */

rfc_handle = RfcAccept(...);       /* Accept RFC connection      */

if (rfc_handle == RFC_HANDLE_NULL) /* Check return code          */
{
  rfc_error_handling("RfcAccept"); /* Handle error and get error */
  return 1;                        /* specification via RfcLastError
                                   */
}

rfc_rc = RfcInstallFunction(‘ABC’, abc_function,...);
                                   /* Install RFC function ‘ABC’ */
                                   /* under the C-routine        */
                                   /* ‘abc_function’             */

if (rfc_rc != RFC_OK)              /* Check return code          */
{
  rfc_error_handling("Install function ‘ABC’);
                                   /* Handle error and get error */
  return 1;                        /* specification via RfcLastError
                                   */
}

do                                 /* Wait for call or execution of*/
{
  rfc_rc = RfcDispatch(...);       /* installed RFC functions until*/

} while ( rfc_rc == RFC_OK);       /* connection is closed or    */
                                   /* terminated                 */

RfcClose(...);
                                   /* Close RFC connection       */


/* RFC function: ‘ABC’ */

static RFC_RC abc_function(RFC_HANDLE rfc_handle)
{
  rfc_rc = RfcGetData(...);        /* Get associated RFC data    */
  if (rfc_rc != RFC_OK)            /* Check return code          */
  {

    rfc_error_handling("RfcGetData");
                                   /* Handle error and get error */
    return 1;                      /* specification via RfcLastError
                                   */
  }

...                                /* Process RFC data           */

rfc_rc = RfcSendData(...);         /*Report result to ABAP program*/
  if (rfc_rc != RFC_OK)            /* Check return code          */
```

RFC Server Program Working With RfcDispatch

```
   {
     rfc_error_handling("RfcSendData");
                                  /* Handle error and get error  */
     return 1;                    /* specification via RfcLastError
                                  */
   }
return 0;                         /* Back to RFC Library         */
}
```

RFC Server Program Working With RfcDispatch and RfcListen

Instead of waiting for further RFC call requests, RFC server programs can also ask for further RFC call requests with **RfcListen** and then issue **RfcDispatch** for an incoming RFC request. You must only change the main program as follows:

```
/* main program */

rfc_handle = RfcAccept(...);     /* Accept RFC connection       */

if (rfc_handle == RFC_HANDLE_NULL) /* Check acceptance of RFC conn.*/
{
  rfc_error_handling("RfcAccept"); /* Handle error and get error  */
  return 1;                        /* specification via RfcLastError
                                   */
}

rfc_rc = RfcInstallFunction('ABC', abc_function,...);
                                   /* Install RFC function 'ABC'  */
                                   /* under the C-routine         */
                                   /* 'abc_function'              */

if (rfc_rc != RFC_OK)              /* Check return code           */
{
  rfc_error_handling("Install function 'ABC');
                                   /* Handle error and get error  */
  return 1;                        /* specification via RfcLastError
                                   */
}


/* Wait for the next RFC request */

do
{
  for (rfc_rc = RFC_RETRY; rfc_rc == RFC_RETRY;)
  {
    rfc_rc = RfcListen(rfc_handle);
    if (rfc_rc == RFC_RETRY)
    {
      /* while waiting for the next RFC request, do something else */
   ...
    }
  }
  if (rfc_rc != RFC_OK)
    break;

  rfc_rc = RfcDispatch(rfc_handle);

}while(rfc_rc == RFC_OK);

RfcClose(...);                     /* Close RFC connection        */
```

RFC Server Programs Sending/Receiving Internal Tables

In RFC functions offered by RFC server programs, all importing and exporting parameters must be defined and also all internal tables as described in *RFC Client Program Transferring Internal Tables* and *RFC Client Program Receiving Internal Tables*" in Programming Examples [Page 40], but it is not necessary to call *ItCreate*.

This will be done automatically by the RFC library. After an RFC function is ended, the RFC library will free the storage for all internal tables used in this function.

Establishing an RFC Connection from an R/2 System

Direct communication from an R/2 System (IBM or SNI environments) to an SAP gateway is not possible. An RFC connection from R/2 Systems can only be established via a *gwhost* program which is delivered with the SAP gateway software. This program will connect to the RFC server program via the SAP gateway specified.

How you can configure the *gwhost* programs depends on the environment (IBM-host or SNI-host).

The RFC syntax in ABAP in R/2 Systems is always the same.

Call from an ABAP program:

```
CALL FUNCTION "ABC" DESTINATION "RFCEXTERNAL"

       IMPORT...
       EXPORT...
       TABLES...
       EXCEPTIONS...
```

The destination “RFCEXTERNAL” identifies an entry in the RFCD table.

Tables:

Table RFCD

Destination	Client	User	Password	S	Argument in Table XCOM
RFCEXTERNAL	000	TEST	xxxxxxxx	E	RFCEXT

The argument “RFCEXT” identifies an entry in the table XCOM.

Table XCOM

R/2 in an IBM environment (with CICS)

Symbolic destination	Logical Unit (LU)	Transaction Program (TP)
RFCEXT	LU02	GWRFCSRV

LU02 is a four-byte identifier defined in a CICS environment which represents an SNA LU6.2 in CICS. This LU6.2 must be defined in the IBM host and on the external computer with a LU6.2-supported SNA product.

R/2 in an SNI environment

Symbolic destination	Logical Unit (LU)	Transaction Program (TP)	Type
RFCEXT		GWRFCSRV	H

Type H must be set for communication in BS2000 via TCP-IP (component 083). Information about the LU is not needed.

GWHOST Configuration

R/2 in an IBM Environment (with CICS)

On the external computer, a “local program” (also called“remotely attachable” or “remotely invokable” program on some SNA products) named GWRFCSRV must be defined in the SNA configuration.

GWRFCSRV is a softlink (on a UNIX platform) or copy (PC platform) of the ‘gwhost’ program which is included in the delivery of an R/3 System.

After GWRFCSRV was started by the installed SNA LU6.2, GWRFCSRV tries to establish a connection to the RFC server program via the SAP gateway. For this purpose, an entry in the sideinfo file must exist for this program:

sideinfo for GWRFCSRV

`DEST =`	`GWRFCSRV`	
`GWHOST =`	`<host name of the SAP gateway,`	`e.g. hs0311>`
`GWSERV =`	`<service name of the SAP gateway,`	`e.g. sapgw53>`
`PROTOCOL =`	`<E (or R): server will be started by (or is already registered at) the SAP gateway>`	
`LU =`	`<host name of the RFC server program,`	`e.g. hs0311>`
`TP =`	`<name of the RFC server program,`	`e.g. /rfctest/srfcserv>`

Because of this complete entry, a corresponding entry in the sideinfo file for the SAP gateway is not required.

This sideinfo file must be in the work directory of the GWRFCSRV program:

- For SNA server of SNA services on IBM-AIX, this is the home directory of the user which is specified in the SNA configuration for the local program GWRFCSRV.
- For SNAplusLink on HP-UX, this is the home directory of the owner of the GWRFCSRV program.

R/2 in an SNI Environment

From within an R/2 System, an RFC connection will not be established via the SAP gateway running on BS2000. It must first connect to a ‘gwhost’ program and then work with an RFC server program via another SAP gateway running on any other supported platforms.

This ‘gwhost’ program is always running as a SAPGWHO-task on BS2000 and waits for an RFC request from the R/2 System. The destination defined in the XCOM table as described above will be sent to this program, and it will connect to the RFC server system/program as defined in the sideinfo file.

This ‘gwhost’ program can only provide one RFC connection at a time. Therefore, you have to configure the number of parallel running ‘gwhost’ programs while installing and configuring the SAPGWHO-jobs for working with more than one RFC connection at one time.

sideinfo for ‘gwhost’ programs

`DEST =`	`GWRFCSRV`	
`GWHOST =`	`<host name of the SAP gateway,`	`e.g. hs0311>`
`GWSERV =`	`<service name of the SAP gateway,`	`e.g. sapgw53>`
`PROTOCOL =`	`<E (or R): server will be started by (or is already registered at) the SAP gateway>`	
`LU =`	`<host name of the RFC server program,`	`e.g. hs0311>`
`TP =`	`<name of the RFC server program,`	`e.g. /rfctest/srfcserv>`

Because of this complete entry, a corresponding entry in the sideinfo file for the SAP gateway is not required.

For more details, see the SAP gateway installation guide.

Starting an RFC Server Program

From within an R/2 System (IBM), an RFC server program can only be started by an SAP gateway. See the configurations section of Establishing an RFC Connection from an R/3 System [Page 58] for details on how to start an RFC server program via the SAP gateway.

By using the registering feature of the SAP gateway (from Release 3.0C onwards), RFC server programs can be started before, register at this SAP gateway and then wait for RFC call requests from R/2 Systems (IBM or SNI host) via the gwhost program.

In this case, the entry for this gwhost program in the sideinfo file must have protocol type **R** instead of **E** (PROTOCOL=R). See The SAPRFC.INI File [Page 66] for more details.

Establishing an RFC Connection from an R/3 System

Call from an ABAP program:

```
CALL FUNCTION “ABC” DESTINATION “RFCEXTERNAL”

      IMPORT...
      EXPORT...
      TABLES...
      EXCEPTIONS...
```

The destination “RFCEXTERNAL” identifies an entry in the RFCDES table.

The RFCDES table in R/3 up to Release 3.0C:

From within an R/3 System, up to Release 3.0C, you can define an entry in the RFCDES table using transaction code sm59:

•	*RFC destination*	RFCEXTERNAL
•	*Connection type*	T
•	*Program location*	explicit or server or user
•	*Target host*	hs0311
•	*Program*	/rfctest/srfcserv
•	*Trace*	not selected
•	*Gateway*	no info (default gateway will be used)

Connection type “T”

The RFC server program is an external program based on the RFC API and communicates via TCP/IP.

Program location “explicit”

The computer on which the RFC server program runs can be specified via *Target host*. In this case, the RFC server program will be started by the SAP either locally or via remote shell. This RFC server program will run on supported UNIX platforms under the gateway user-ID.

The program location “explicit” is not possible for RFC server programs with a 16-bit RFC library on Windows. See “The RFCDES table in R/3 Release 3.0C onwards” for new functionality in Releases as of 3.0C.

Program location “Server”

The RFC server program is started by the current R/3 application server. This improves performance, because the RFC server program always runs on the same computer as the ABAP program (work process of the current ABAP program).

This RFC server program will run on supported UNIX platforms under the user-ID of the respective R/3 application server.

The program location “Server” is **not** possible for RFC server programs with a 16-bit RFC library on Windows. See “The RFCDES table in R/3 Release 3.0C onwards” for new functionality in Release >= 3.0C.

Program location “User”

The RFC server program is started by the current SAPGUI (SAP frontend). The program location “User” is the only option available to start an RFC server program with a 16-bit RFC library on Windows. See *The RFCDES Table in R/3 Releases as of 3.0C* below for new functionality.

Trace

When this field is selected, trace data can be displayed in the SAP System using the ABAP program RFRFCTRC table. On extended systems, the trace data are stored in the ‘dev_rfc’ file. With the 16-bit RFC library, a trace file ‘RFCxxxx.TRC’ will be created for each connection.

Gateway

The gateway specification is optional. If nothing is specified, the default gateway will be used. The default gateway depends on the R/3 Release:

- Release < 2.1I/2.2C

  Call up transaction se38 and execute the program RSPARAM: The parameters rdisp/gateway and rdisp/gw_service contain data about the default SAP gateway.

- Release >= 2.1I/2.2C/3.0

  The local gateway on each R/3 application server since these Releases is the default gateway.

Testing a Connection

After having entered and saved a new destination using transaction sm59, you can test the RFC connection by double-clicking on *Test connection*. However, this is only possible if the RFC server program was implemented by using **RfcDispatch** in a loop and not with **RfcGetName**.

The RFCDES table in R/3 Release 3.0C onwards

From R/3 Release 3.0C onwards, you can define an RFC server program which is already started, registered at an SAP gateway and waiting for RFC call requests by using transaction sm59.

You must define an RFC server program with registering as activating action, and a program-ID.

Program-ID

The program-ID is an identifier of the RFC server program for the SAP gateway to distinguish between different RFC server programs. It is recommended to use both the name of the RFC server program and the host name of the RFC server program.

Information about the SAP Gateway

□

The SAP gateway of the relevant application server will be used if nothing else is specified. It is recommended to define the SAP gateway explicitly, because an RFC server program usually registers at a specific SAP gateway.

Configurations

SAP recommends you to specify the complete name of the RFC server program (including the full path name) when defining an entry in the RFCDES table using transaction sm59.

The RFC server program can be started by the currently running SAPGUI, by the currently running application server or by an SAP gateway, but it has to communicate via a specified SAP gateway.

Consequently, the following prerequisites must be met:

- The user under which the SAPGUI or the application server or the SAP gateway runs must have execution rights for the RFC server program.
- Both SAP gateway and RFC server program are running on the same computer:
  - The IP address of this computer must be specified in the *hosts* file.
  - The service name of the SAP gateway must be specified in the *services* file.
- The SAP gateway and the RFC server program are running on different computers:
  - The IP-addresses of both computers must be specified in both *hosts* files.
  - The service name of the SAP gateway must be specified in the services file.
  - The SAP gateway must have the authority to start the RFC server program on the target computer via remote shell:

    i) The user of the SAP gateway must be defined on the target computer.

    ii) The**.rhosts** file which contains the host name of the gateway computer must exist in this user's home directory on the target computer

    iii) Since the remote shell command is different on different UNIX platforms (remsh, rsh, etc.), the command can be defined in the gateway profile parameter gw/remsh, if necessary (e.g. gw/remsh=/usr/ucb/remsh).
    The default is '**remsh**'.

    □

    For very large *hosts* files on OS/2, it is recommended to define the entries for all computers involved at the beginning of this file, because it may take a long time to get all IP-addresses used.

Starting an RFC server program

An RFC server program can be started by the currently running SAPGUI, by the currently running R/3 application server or by an SAP gateway, as described in the above section.

Using the registering feature of the SAP gateway (from Release 3.0C onwards), RFC server programs can be started before, register at this SAP gateway and then wait for RFC call requests from R/3 System Release 3.0C onwards.

Here, the RFC server program of the RFCDES table (transaction sm59) must be defined as registering for activating action. See The SAPRFC.INI File [Page 66] for more details.

Passing Parameters

Five API routines pass parameters back and forth to the remote function. These routines are:

RfcCall and RfcReceive

RfcCallReceive

RfcGetData and RfcSendData

The parameters are passed in one of two structures: RFC_PARAMETER and RFC_TABLE. When your program calls one of these routines, it must provide information about all parameters being passed by filling in these structures.

Passing Import/Export Parameters

Exporting and importing fields are passed to the remote function in an array of RFC_PARAMETER [Page 204] structures:

```
typedef struct
{
   void *   name;
   unsigned nlen;
   unsigned type;
   void *   addr;
   unsigned leng;
}
RFC_PARAMETER;
```

Arrays must be terminated by an entry with name equal to NULL. The supported ABAP datatypes (see Table 4-1) are defined in **saprfc.h** (see ABAP Data Types [Page 96]).

ABAP Data Types Supported by the RFC API

Data type	Typedef	Length in Bytes	Description
TYPC	RFC_CHAR[]	1-65535	Characters, blank padded at the end
TYPX	RFC_BYTE[]	1-65535	Binary data
TYPP	RFC_BCD[]	1-16	BCD numbers (Binary Coded Decimals)
TYPINT	RFC_INT	4	Integer
TYPFLOAT	RFC_FLOAT	8	Floating point
TYPDATE	RFC_DATE	8	Date ("YYYYMMDD")
TYPTIME	RFC_TIME	6	Time ("HHMMSS")

Currently, only scalar data types are supported. Record structures will be supported in the future.

Passing Internal Tables

Almost all the RFC sending and receiving functions have two parameters where the exported or imported fields and the referenced ‘internal tables’ are specified. The parameters are pointers to arrays of the structure RFC_PARAMETER [Page 204]. The arrays have to be terminated by an entry with *name* equal NULL.

Internal tables parameters are passed to the remote function in an array of RFC_TABLE [Page 207] structures:

```
typedef struct {

   void *      name;
   unsigned    nlen;
   unsigned    type;
   unsigned    leng;
   ITAB_H      ithandle;
   RFC_ITMODE itmode;
   int         newitab;
} RFC_TABLE;
```

Currently, only scalar data types are possible. That is, only single-column tables and tables with similar columns (only TYPC or only TYPX fields) are supported.

In order to send or receive inhomogenous structures or tables use the function RfcInstallStructure [Page 142]. See also the example SRFCTEST.C.

The type definition for ITAB_H is void *. Note that the *itab_h* table handle must be supplied by the calling program in *RfcCall*. Use the routine *ItCreate* to create an *itab_h* table handle (see ItLeng [Page 159] for more information).

When receiving a call with *RfcGetData*, *itab_h* is filled automatically by *RfcGetData*.

For the field *itmode*, you should specify that the table is to be passed by reference (value ITMODE_BYREFERENCE). This notion has a special meaning in the context of remote calls that differs slightly from the conventional meaning (see Parameter Handling in Remote Calls [Ext.]for more information).

Creating and Manipulating Table Parameters

Tables passed as parameters to an RFC function must match SAP internal tables in their structure and handling.

As a result:

- To create a table parameter for sending, use the routine *ItCreate*.
  This routine creates a control structure (defined as ITAB_H) for a table just like the one the SAP System creates automatically for internal tables. This address of the control structure is stored in the *handle* field of the RFC_TABLE structure.

- To manipulate a table passed to or from the SAP System (for example: to access, append or delete rows, and so on), use the table-handling routines provided by the API. Do not attempt to manipulate the table using normal C mechanisms.

Writing an RFC Function in C

To receive the parameters of a remote function call, use the function RfcGetData [Page 138] within a registered function or after receiving the name of the called function by RfcGetName.

To send the answer back to a caller, use the function RfcSendData [Page 145].

To raise an exception while processing a received RFC call, use the function RfcRaise [Page 143] or RfcRaiseTables [Page 144].

Using Multiple-Client Server Programs

As of R/3 Release 3.0C, an RFC server program can register itself at an SAP gateway under a program identifier and then wait for further RFC requests from any other R/3 System.

There are three ways of doing this:

1. Registration will be carried out by RfcAccept [Page 116], and the parameter *argv* must include the following data:

- a<program-ID>

  e.g. <host name>.<program name>

- g<SAP gateway host name>

  e.g. hs0311

- x<SAP gateway service>

  e.g. sapgw00

- t (optional)

  RFC trace on

2. You can define an entry in the *saprfc.ini* file which includes all necessary connection parameters and optionally the RFC-specific parameter RFC_TRACE. An RFC client program can then issue the RfcOpen [Page 126] call with the mode RFC_MODE_PARAMETER and a destination pointed to the defined entry. See The SAPRFC.INI File [Page 66] for more details.

   In this case, the parameter argv in *RfcAccept* must include the following data:

- D<destination>

  This destination must point to a valid entry in the saprfc.ini file.

- t (optional)

  RFC trace on

  Until now the contents of *argv* were defined by the SAP instance which started the server program.

3. With RfcDispatch [Page 135] or RfcListen [Page 190] or RfcGetName [Page 139] it can wait or look for the following RFC request. In this case, you must define the destination in the R/3 System using transaction SM59:

- connection type

  T

- activate kind

  register mode

- program-ID

  same program-ID as in RfcAccept [Page 116]

- gateway host

  same gateway host name as in *RfcAccept*

– gateway service

  same gateway service as in *RfcAccept*

To work with this functionality no changes in both client or server porgram are necessary. Only the configuration with SM59 must be changed as mentioned above, and the server program must be started with other parameters.

□

You must use the program name as program-ID, because there may be more programs registered with the same name at an SAP gateway. Therefore, use at least the host name of the computer where your RFC server program is running as part of the program-ID. For example, you can start the *rfcexec* program in register mode with RFC trace from the command line as follows:

```
rfcexec -ap13387.rfcexec -ghs0311 -xsapgw53 -t
```

or

```
rfcexec -Drfctest
```

The *saprfc.ini* file contains the entry rfctest.

Transactional RFC (tRFC) server programs can also work in register mode.

Additionally, this functionality both enables and eases the use of tRFC server programs in debugging mode.

For further information on the transactional RFC, see Transactional Remote Function Calls [Ext.].

See also Registering Server Programs with the SAP Gateway [Page 49].

The SAPRFC.INI File

This section contains the following topics:

Introduction

If you want to change or add parameters, you need not change the program code. These parameters can be specified in a file called *saprfc.ini*.

The RFC library will read the *saprfc.ini* file to find out the connection type and all RFC-specific parameters needed to connect to an SAP System (R/2 or R/3), or to register an RFC server program at an SAP gateway and wait for RFC calls from any SAP System.

All RFC-specific parameters, both currently known (load balancing, ABAP-debug, RFC with SAPGUI) or becoming available in the future, can be used without changing the RFC programs.

The *saprfc.ini* file must be in the same directory as the RFC client/server program, or you can define it with full path and file name by the environment variable RFC_INI.

□

Windows: `set RFC_INI = d:\rfctest\saprfc.ini`

□

You can work with the *saprfc.ini* file only if your RFC SDK was delivered with an R/3 Release as of 3.0C. However, the *saprfc.ini* file is merely a new interface in the RFC library. Therefore, it can work with any R/2 or R/3 System.

RFC client programs

To use this new feature, an RFC client program must issue the **RfcOpen** with RFC_MODE_PARAMETER as *mode* parameter. The destination must point to an entry of type **B**, **A**, **2**, **E** or **R** in this *saprfc.ini* file.

□

No sideinfo file is necessary when using the new feature.

RFC server programs

To use this new feature, an RFC server program must issue the **RfcAccept** with -D<destination> as parameter. The destination must point to an entry of type **R** in the *saprfc.ini* file.

Restrictions

- The parameters **ABAP_DEBUG** and **USE_SAPGUI** are only possible if the RFC server system is an R/3 System as of Release 3.0C, and the RFC client program does **not** work with the 16-bit RFC library on Windows (SAPGUI must also be of an R/3 Release as of 3.0C). For more details, see Load Balancing [Page 39].
- The Load Balancing feature is only available for R/3 Releases as of 3.0.

Possible connection (entry) types

Five connection types are available:

- Type **R** is for RFC server programs or for a client program working with another external program as RFC server program which is already registered at an SAP gateway.

- Type **B** is recommended for connecting to an R/3 System (using Load Balancing).
- Type **A** is only to be used if you want to connect to a specific application server.
- Type **2** is for connecting to an R/2 System only.
- Type **E** is for RFC client programs working with another external program as RFC server program.

See RFC Between External Programs [Page 235] for more details about RFC between external programs.

Possible Parameters

Type R

Registers an RFC server program at an SAP gateway and lets it wait for RFC calls by an R/2 or R/3 System.

The following parameters can be used:

- `DEST = <destination in RfcAccept>`
- `TYPE = <R: Register at SAP gateway>`
- `PROGID = <program-ID, optional; default: destination >`
- `GWHOST = <host name of the SAP gateway>`
- `GWSERV = <service name of the SAP gateway>`
- `RFC_TRACE = <0/1: OFF/ON, optional; default:0(OFF)>`

In an R/3 System the **program-ID** and this **SAP gateway** must be specified in the *Destination* entry defined with transaction sm59: use connection type **T** and **register mode**.

To work with R/2 Systems, the entry in the *sideinfo* file for the *gwhost* program must have **R** instead of **E** as PROTOCOL parameter.

The host service names of the SAP gateway must be specified in the *hosts* and *services* files (<service name> = **sapgw**<R/3 system number>).

Type B

Connects to an R/3 System using load balancing (as of R/3 3.0).
The application server will be determined at runtime.

The following parameters can be used:

- `DEST = <destination in RfcOpen>`
- `TYPE = <B: use Load Balancing feature>`
- `R3NAME = <name of R/3 System, optional; default: destination>`
- `MSHOST = <host name of the message server>`
- `GROUP = <group name of the application servers, optional; default: PUBLIC>`
- `RFC_TRACE = <0/1: OFF/ON, optional; default:0(OFF)>`
- `ABAP_DEBUG = <0/1: OFF/ON, optional; default:0(OFF)>`
- `USE_SAPGUI = <0/1: OFF/ON, optional; default:0(OFF)>`

The host name and the service name of the message server must be defined in the ‘hosts’ and ‘service’ files (<service name> = **sapms**<R/3 system name>).

Possible Parameters

Type A

Connects to a specific R/3 application server.

The following parameters can be used:

- `DEST = <destination in RfcOpen>`
- `TYPE = <A: RFC server is a specific R/3 application server>`
- `ASHOST = <host name of a specific R/3 application server>`
- `SYSNR = <R/3 system number>`
- `GWHOST = <optional; default: gateway on application server>`
- `GWSERV = <optional; default: gateway on application server>`
- `RFC_TRACE = <0/1: OFF/ON, optional; default:0(OFF)>`
- `ABAP_DEBUG = <0/1: OFF/ON, optional; default:0(OFF)>`
- `USE_SAPGUI = <0/1: OFF/ON, optional; default:0(OFF)>`

The host name and the service name of the specific application server must be defined in the *hosts* and *services* files (<service name> = **sapdp**<R/3 system number>).

The host name and the service name of the SAP gateway must be defined in the *hosts* and *services* files. If GWHOST and GWSERV are not specified, the service name of the SAP gateway must be defined in the *services* file (<service name> = **sapgw**<R/3 system number>).

Type 2

Connects to an R/2 System.

The following parameters can be used:

- `DEST = <destination in RfcOpen and in the 'sideinfo' file for the SAP gateway>`
- `TYPE = <2: RFC server is an R/2 System>`
- `GWHOST = <host name of the SAP gateway>`
- `GWSERV = <service name of the SAP gateway>`
- `RFC_TRACE = <0/1: OFF/ON, optional; default:0(OFF)>`

The host name and the service name of the SAP gateway must be defined in the *hosts* and *services* files. (<service name> = **sapgw**<R/3 system number>).

Type E

Connect to another external program as RFC server program.

- `DEST = <destination in RfcOpen>`
- `TYPE = <E: RFC server is an external server>`
- `GWHOST = <host name of the SAP gateway>`
- `GWSERV = <service name of the SAP gateway>`

- `TPHOST = <host name of the RFC server program>`
- `TPNAME = <name of the RFC server program>`
- `RFC_TRACE = <0/1: OFF/ON, optional; default:0(OFF)>`

The host name and the service name of the SAP gateway must be defined in the *hosts* and *services* files (<service name> = **sapgw**<R/3 system number>).

See “Configurations” in Establishing an RFC Connection from an R/3 System [Page 58] for details about how to start an RFC server program by an SAP gateway.

Examples

Type R: Register an RFC server program at an SAP gateway

```
DEST=rfctest
TYPE=R
PROGID=hw1145.rfcexec
GWHOST=hs0311
GWSERV=sapgw53
RFC_TRACE=1
```

Type B: R/3 System - Load Balancing feature

```
DEST=BIN
TYPE=B
R3NAME=BIN
MSHOST=hs0311
GROUP=PUBLIC
RFC_TRACE=0
ABAP_DEBUG=0
USE_SAPGUI=0
```

Type A: R/3 System - specific application server

```
DEST=BIN_HS0011
TYPE=A
ASHOST=hs0011
SYSNR=53
RFC_TRACE=0
ABAP_DEBUG=0
USE_SAPGUI=0
```

Type 2: R/2 System (IBM)

```
DEST=K50
TYPE=2
GWHOST=is0001
GWSERV=sapgw00
RFC_TRACE=0
```

Type E: External Program (will be started by SAP gateway)

```
DEST=RFCEXT
TYPE=E
GWHOST=hs0311
GWSERV=sapgw53
TPHOST=hs0311
TPNAME=/rfctest/srfcserv
RFC_TRACE=0
```

Call-Back Feature with R/3 and External Systems

This section contains the following topics:

Introduction

During the execution of an RFC function, it is sometimes necessary and useful to call another RFC function in the original/source RFC client system to get some more data before continuing with the current RFC function. This functionality is called **call-back** and will use the same RFC connection established by the first RFC call request.

The following sections describe how to use and carry out this feature both in ABAP function modules and in external programs.

Call-Back from an ABAP Function Module [Page 75]

Call-Back from an RFC Server Program [Page 76]

Call-Back from an ABAP Function Module

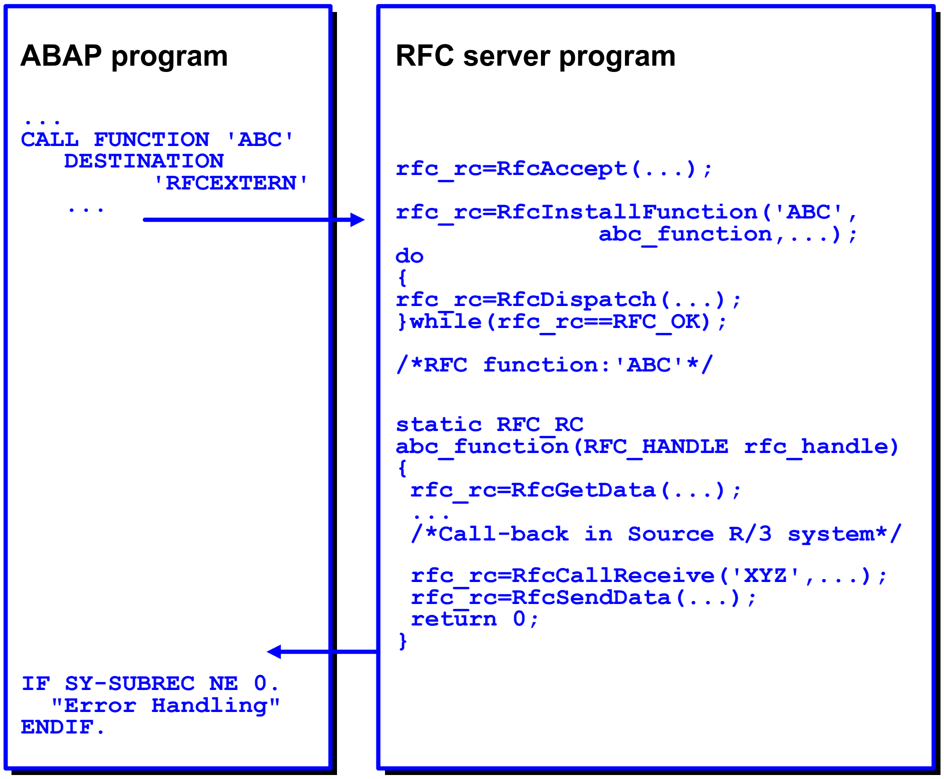
The following programming example shows you how to use this feature:

RFC client program		Function module in an R/3 System
rfc_rc = **RfcOpen**(...);		**FUNCTION ABC.**
rfc_rc = **RfcInstallFunction(‘XYZ’, xyz_function**,...);		
rfc_rc = **RfcCallReceive**(‘ABC’,...);	----->	...
		CALL FUNCTION ‘XYZ’ DESTINATION ‘BACK’
If(rfc_rc==**RFC_CALL**)	<-----	...
{		
rfc_rc = **RfcDispatch**(...);		
if(rfc_rc!=RFC_OK)		
exit(1);	----->	...
}		
rfc_rc=RfcReceive(...);	<-----	**ENDFUNCTION**
...		

```
/* RFC function: ‘ABC’ */
static RFC_RC xyz_function(RFC_HANDLE rfc_handle)
{
                            rfc_rc = RfcGetData(...);           /*
                            Get RFC data                        */
                            ...                                 /*
                            Process RFC data                    */
                            rfc_rc = RfcSendData(...);          /*
                            Report result to ABAP               */
      return 0;
}
```

Call-Back from an RFC Server Program

The following programming example shows you how to use this feature:



ABAP program

```
...
CALL FUNCTION 'ABC'
   DESTINATION
         'RFCEXTERN'
   ...


IF SY-SUBREC NE 0.
  "Error Handling"
ENDIF.
```

RFC server program

```
rfc_rc=RfcAccept(...);

rfc_rc=RfcInstallFunction('ABC',
              abc_function,...);
do
{
rfc_rc=RfcDispatch(...);
}while(rfc_rc==RFC_OK);

/*RFC function:'ABC'*/


static RFC_RC
abc_function(RFC_HANDLE rfc_handle)
{
 rfc_rc=RfcGetData(...);
 ...
 /*Call-back in Source R/3 system*/

 rfc_rc=RfcCallReceive('XYZ',...);
 rfc_rc=RfcSendData(...);
 return 0;
}
```

Function module ‘XYZ’ is any RFC-supported function module in this source R/3 System.

FUNCTION XYZ.

...

ENDFUNCTION.

Transactional RFCs and External Systems

The following topics are available:

Introduction

From Release 3.0 onwards, data can be transferred between two R/3 Systems **reliably** and **safely** via **transactional RFC (tRFC)**.

□

This RFC was renamed from **asynchronous** to **transactional** RFC, because asynchronous RFC has another meaning in R/3 Systems.

The remote system need not be available at the time when the RFC client program is executing a tRFC. In R/3 Systems, the tRFC component stores the called RFC function together with the corresponding data in the database, including a unique transaction identifier (TID). This ensures that the called function module executed **exactly once** in the RFC server system.

For more information on tRFC in ABAP, see Transactional Remote Function Calls [Ext.].

For external systems, the RFC library provides some special RFC calls. However, the transactional RFC cannot be fully implemented in the RFC library.

tRFC between R/3 and External Systems

On external systems, the transactional RFC cannot be fully implemented in the RFC library, because of the following reasons:

- A database is not always available in external systems.
- The RFC library cannot always repeat the RFC call in case of errors, such as network errors.

Therefore, the transactional RFC interface from external systems to an R/3 System is currently implemented as follows:

- **RFC library**

  The RFC library provides some special RFC calls, such as **RfcCreateTransID**, **RfcIndirectCall** and **RfcInstallTransactionControl** for working with tRFC. It will pack and unpack RFC data between RFC format and tRFC format.

  For an R/3 System, there is no difference whether these calls are requested from another R/3 System or from an external system. For an RFC server program, the RFC function itself (only the RFC function, not the whole RFC server program) can be executed normally, as it is called via ‘normal’ RFC with RfcGetData and RfcSendData.

- **RFC client programs and RFC server programs**

  Both programs have to manage the TIDs themselves for checking and executing the requested RFC functions **exactly once** as the tRFC component in an R/3 System does.

- **R/3 Systems**

  In R/3 Systems, no additional changes are necessary in ABAP programs working with external RFC programs which use the tRFC interface. For ABAP programs, such as RFC client programs, the destination defined in CALL FUNCTION must have **‘T’** as connection type.

  □

  The tRFC interface is not available with a 16-bit RFC library on Windows platforms.

Transactional RFC Client Programs

After having been connected to an R/3 System (via **RfcOpen**), an RFC client program must use the following two RFC calls for working with the tRFC interface:

- **RfcCreateTransID**

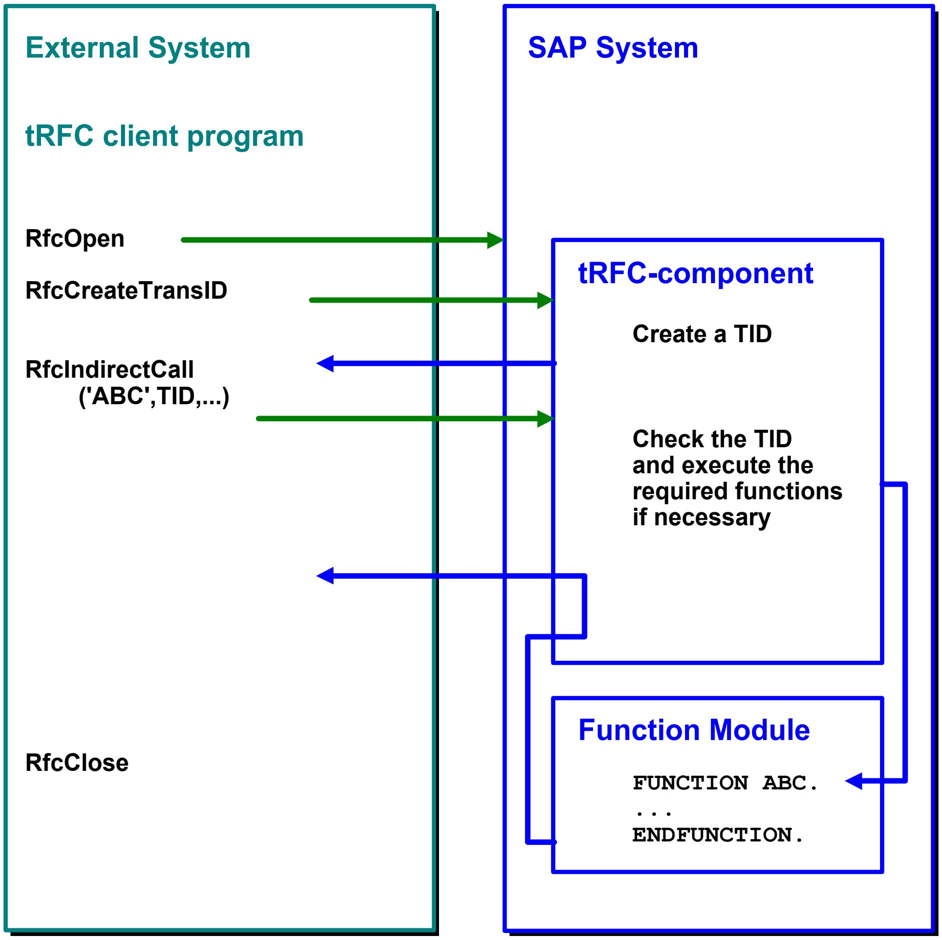
  With this call, the RFC library tries to get a TID created by the R/3 System. If errors occur, the RFC client program has to reconnect later and must try to repeat this call. Otherwise, the RFC client program can assign this TID with the RFC data, and if the next call is not successful, it can be repeated later.

- **RfcIndirectCall**

  With this call, the RFC library will pack all RFC data belonging to an RFC function together with the TID and send them to the R/3 System using the tRFC protocol.
  If an error occurs, the RFC client program has to reconnect later and must try to repeat this RFC call (**RfcIndirectCall**). In this case, it has to use the old TID and **must not** get a new TID with **RfcCreateTransID**. Otherwise, there is no guarantee that this RFC function will be executed **exactly once** in the R/3 System.
  After this call is executed successfully, the transaction will be completed once and for all. The RFC client program can then update its own TID management (e.g. delete the TID entry).

Contrary to tRFC between R/3 Systems, a transaction from an RFC client program contains **only one** RFC function.

Technical Description



Sample test program *trfctest.c*

The C program **trfctest.c** which is delivered in the RFC SDK (executable and source code) is an example of a tRFC client program.

To connect to an R/3 System, a saprfc.ini file is needed.

Data which is to be transferred to an R/3 System via tRFC must be in a file. The file name will be required when the program is started. Each line in this file is one line in an internal table. Only one internal table with a 72-bit line length is used.

All data which is received in the R/3 System will be written in the TCPIC table in an R/3 System (only the first 40 bytes), and the function module **STRFC_WRITE_TO_TCPIC** will be activated.

The *trfctest.c* program uses the file I/O on the running platform to manage the TIDs.

For each TID there is an entry in the TID management. This entry contains the date and time, the TID itself, the state of the transaction (CREATED, CONFIRMED,...) and the name of the data file.

It is possible to break up the program trfctest.c in order to simulate error cases.

Transactional RFC Client Programs

Whenever this program is started, it will look at the TID management for aborted transactions. If any transactions exist it will first try to repeat these transactions.

Since the program *trfctest.c* can be run on different platforms, a corresponding flag (SAPonUNIX, SAPonNT,...) must be set if you want to compile and link this program in your environment.

For more details, see the source code delivered with the RFC SDK.

Transactional RFC Server Programs

After having connected to an R/3 System (via **RfcAccept**) and after having installed the supported RFC functions, the RFC server program has to use the RFC call **RfcInstallTransactionControl** for working with the TIDs to check and execute the real RFC function it supports before entering in the loop with **RfcDispatch**.

This function installs the following four functions (e.g. C routines) to control transactional behavior:

- **onCheckTid**

  This function will be activated if a tRFC is called from an R/3 System. The current TID is passed. The function has to store this TID in permanent storage and return 0.
  If the same function will be called again later with the same TID, it has to make sure that it will return a value <>0. If the same TID is already running by another process but is not completed, the function has to wait until the transaction finishes or to stop the RFC connection with **RfcAbort**.

- **onCommit**

  This function will be called if all RFC functions belonging to this transaction are done and the local transaction can be completed. It should be used to locally commit the transaction if necessary (working with database).

- **onRollback**

  This function is called instead of the function **onCommit** if there is an error occurring in the RFC library while processing the local transaction. This function can be used to roll back the local transaction (working with database).

- **onConfirmTid**

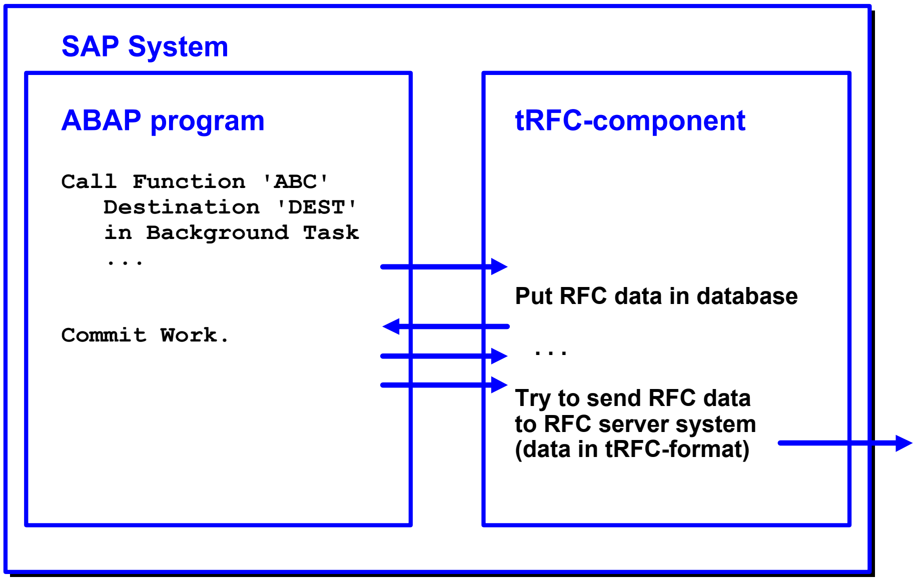
  This function will be called if the local transaction is completed. All information about this TID can be deleted.

  - These four functions must be realized in **any** tRFC server program and are independent of the real RFC function offered in an RFC server program. A server program can offer more than one RFC function but only the four functions above and not four functions for each RFC function.

  - If you do not install and work with these four functions it can only be guaranteed that an RFC call issued by **'Call Function... In Background Task'** is executed by this RFC server program **at least once**. In this case, all RFC functions offered by such a server program can be called **more than once** by using the tRFC interface.

Technical Description

tRFC within an R/3 System



```
Call Function 'ABC'
   Destination 'DEST'
   in Background Task
   ...


Commit Work.
```

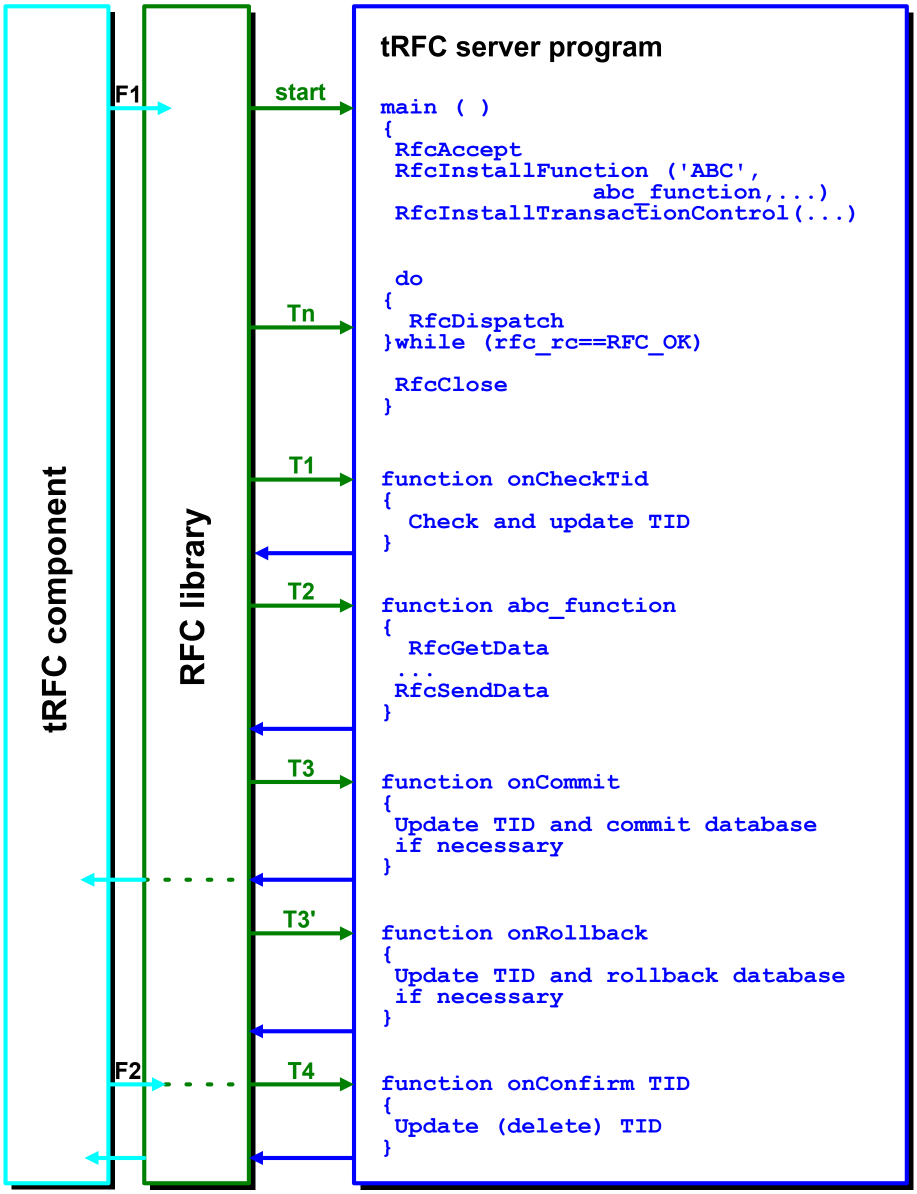
The following parameters can be configured with transaction SM59:

- try or do not try to connect to an RFC server program in cases of error
- number of times for trying
- time between two tries

The program location ‘User’ defined with this transaction (start RFC server program via currently using SAPGUI) is not available, because the tRFC component is not assigned to any SAPGUI while running.

Transaction SM58 displays the running state of a transaction if the transaction was not executed successfully already.

tRFC between tRFC component, RFC library and tRFC server program



tRFC server program

```
main ( )
{
 RfcAccept
 RfcInstallFunction ('ABC',
                    abc_function,...)
 RfcInstallTransactionControl(...)


 do
{
  RfcDispatch
}while (rfc_rc==RFC_OK)

 RfcClose
}


function onCheckTid
{
  Check and update TID
}


function abc_function
{
  RfcGetData
 ...
 RfcSendData
}


function onCommit
{
 Update TID and commit database
 if necessary
}


function onRollback
{
 Update TID and rollback database
 if necessary
}


function onConfirm TID
{
 Update (delete) TID
}
```

The tRFC component, the RFC library and the tRFC server communicate with each other in two phases:

- Phase 1/F1: Function transfer
- Phase 2/F2: Confirmation

F1: Function transfer

After tRFC data is received, the RFC library will activate the tRFC server program (action: **start**). Using **RfcDispatch** in a loop, the library will call up the following functions within this function transfer phase:

- **T1**: onCheckTid
- **T2**: the required RFC function ‘abc_function’
- **T3**: onCommit

As described above, the function transfer phase will be repeated by the tRFC component in the R/3 System if any CPI-C error (network errors, for example) occurs during this phase. The maximum number of tries and the time between two tries can be defined by using transaction **SM59** and **TRFC options**.

F2: Confirmation

After the RFC library informs the tRFC components in the R/3 System about a successful T3, it will immediately receive confirmation of the current transaction. The RFC library will then call up the function

- **T4**: onConfirmTID

After this phase, the current transaction is successfully completed on both sides.

The Sample Test Program *trfcserv.c*

The C-program **trfcserv.c**, delivered in the form of executable and source code in the RFC SDK, is an example of a tRFC server program.

For testing, the ABAP program SRFCTEST, option *Transactional RFC*, can be used with this server program.

The data received from the R/3 System will be written in ‘trnn... n.dat’ on the running platform. Each line in this file is one line in an internal table. Only one internal table with a line length of 72 bit is used.

To manage the TIDs, the trfcserv.c program uses the file I/O on the running platform.

For each TID there is an entry in the TID-management which contains the date and time, the TID itself, the state of this transaction (CREATED, CONFIRMED,...) and the name of the data file.

Since this program can run on different platforms, a corresponding flag (SAPonUNIX, SAPonNT,...) must be set if you want to compile and link this program in your environment.

For more details, see the source code delivered in the RFC SDK.

Function Interface: Summary

The following functions form the programming interface to call ABAP function modules from a C environment:

Administration

- **RfcOpen**

  Open RFC connection (RFC client program)
- **RfcConnect**

  Open RFC connection to R/3 via LOAD BALANCING (Release 3.0 onwards)
- **RfcAccept**

  Accept RFC connection (RFC server program)
- **RfcClose**

  Close RFC connection
- **RfcAbort**

  Terminate RFC connection and send error text to ABAP program
- **RfcConnArgv**

  Set the parameters required for RfcOpen
- **RfcConnArgv3**

  Set the parameters required for RfcOpen
- **RfcEnvironment**

  Set RFC parameters for RFC library
- **RfcLastError**

  Get extended error specification after an RFC error

Calling and Accepting RFC Functions

Description

The **Remote Function Call Application Programming Interface** (**RFC API**) allows - both remotely or locally - you to call ABAP function modules from C programs as well as receiving call requests issued from an ABAP program by the CALL FUNCTION interface.

In ABAP function modules are special routines with well-defined and documented interfaces which are developed within a library development workbench.

ABAP function modules use named parameter passing. There are also exceptions which can be raised by a function module to indicate errors. These exceptions can be caught by the caller of a function module.

The following functions form the programming interface to call ABAP function modules from a C environment.

Functions for an RFC Client Program

- **RFCOpen**

  Open an RFC connection

- **RFCOpenExt**

  Another way to open an RFC connection, more appropriate for non-C environments than Visual Basic

- **RFCOpenExtV3**

  Another way to open an RFC connection, more appropriate for non-C environments than Visual Basic (using RFC Version 3)

- **RfcCall**

  Call an ABAP function module without waiting for the result

- **RfcReceive**

  Wait for execution of an RFC function called and receive the return values from the ABAP function module

- **RfcCallReceive**

  Call a function module and receive the return values in one step

Functions for an RFC Server Program

- **RfcAccept**

  Accept an RFC request or register at an SAP gateway

- **RfcInstallFunction**

  Register functions as callable function modules

- **RfcInstallFunctionExt**

  Communicate function as an RFC function (Windows 3.x.)

Calling and Accepting RFC Functions

- **RfcDispatch**

  Wait for the next function call

- **RfcGetAttributes**

  Return information about an RFC connection

- **RfcGetName**

  Read the symbolic function name

- **RfcGetData**

  Receive the parameters of a function

- **RfcSendData**

  Send back the return values

- **RfcRaise**

  Report error that occured during the execution of an RFC function

- **RfcRaiseTables**

  Raise an exception while processing an RFC call if the function module being called has tables parameters

- **RfcGetTransID**

  Get associated TID for execute an RFC-function via aRFC (3.0 onwards)

- **RfcCallReceive**

  Call a function module and receive the return values in one step. It can be used for calling back a function module in R/3 using the same RFC connection while executing an RFC function in this server program. It is also possible to use RfcCall and RfcReceive.

- **RFCInstallTransactionControl**

  Register functions as callable function modules for transactional RFC

RFC Calls for Manipulating Internal Tables

Description

These functions allow processing of ABAP internal tables in C.

ABAP internal tables follow the model of a relational database table.

ABAP internal tables consist of identical rows. When it is created, a table is empty. In ABAP you can fill rows into a table by the statements ‘Insert’ or ‘Append’. You can access a row by ‘Read Table’, and you can delete a row by ‘Delete’. You can free a table by ‘Free Table’, and you can receive information about tables by ‘Describe’.

These language constructs correspond to the following C routines:

- **ItCreate**

  creates a new internal table

- **ITAB_H**

  handle of an internal table

- **ItDelete**

  deletes the content of a complete internal table and frees storage

- **ItGetLine**

  reads a line from an internal table

- **ItInsLine**

  inserts a line into the given position of an internal table

- **ItAppLine**

  appends a line at the end of an internal table

- **ItDelLine**

  deletes a line from an internal table

- **ItGupLine**

  reads a line for update

- **ItFree**

  resets an internal table to initial state

- **ItFill**

  returns the number of lines in a table

- **ItLeng**

  returns the width of a table, i.e. the size of a row of the table

  □

  The syntax and semantics of the above RFC calls are identical for all platforms supported.

RFC Calls for Manipulating Internal Tables

The syntax of the RFC calls is described in **saprfc.h**. The syntax of the RFC calls for manipulating internal tables is described in **sapitab.h**.

□

Creating and filling a new internal table

```
const int myTableSize = 200;
ITAB_H   handle;
void *   ptr;

handle = IitCreate(“MyTable”, myTableSize, 0,0);
if(handle == ITAB_NULL)
{
... error
}

do
{
  ptr = ItAppLine(handle);;

  if(ptr != NULL)
  {
     memcpy(ptr,..., myTableSize);
  }
  while(ptr != NULL);
```

Transactional Remote Function Calls

- **RfcInstallTransactionControl**
  installs four functions to control transactional behaviour.
- **RFC_ON_CHECK_TID**
  is called when a local transaction is starting.
- **RfcCreateTransID**
  Get a unique transaction-ID for calling an ABAP function module using the transactional RFC Interface
- **RfcIndirectCall**
  Call an ABAP function module using the transactional RFC Interface
- **RFC_ON_COMMIT**
  is called when a local transaction ends.
- **RFC_ON_CONFIRM_TID**
  is called when a local transaction is completed.
- **RFC_ON_ROLLBACK**
  is called when a local transaction ends with failure.
- **RFC_ONCALL**

Extended Remote Function Calls

- **RfcAddExportParam**

  adds export parameters to the stack.

- **RfcAddImportParam**

  adds import parameters to the stack.

- **RfcAddTable**

  adds table parameters to the stack.

- **RfcAllocParamSpace**

  allocates stack space.

- **RfcCallExt**

  sends a call request and returns after sending the call request..

- **RfcCallReceiveExt**

  makes an RFC call and receives return vaues (it waits until it receives an answer before returning).

- **RfcOpenExt**

  opens an RFC connection.

- **RfcOpenExtV3**

  opens an RFC connection.

- **RfcFreeParamSpace**

  frees stack space.

- **RfcGetDataExt**

  gets incoming parameter values when the function is being called.

- **RfcReceiveExt**

  receives return values from an RFC call.

- **RfcSendDataExt**

  sends the result parameters back to the caller.

- **RfcInstallFunctionExt**

  installs a function to be callable as RFC function module for Windows 3.x (16-Bit).

Special Functions

- **RfcListen**

  Check whether the result of a previously submitted RfcCall is available

- **RfcWaitForRequest**

  Wait for incoming RFC requests.

  This function is only available after **RfcAccept** in **register mode**.

ABAP Data Types

Description:

The RFC API uses only ABAP data representation for data transported between function module interfaces.

The following data types are supported:

- RFC_CHAR [Page 97]
- RFC_HANDLE [Page 98]
- RFC_FUNCTIONNAME [Page 99]
- RFC_TID [Page 100]
- RFC_NUM [Page 101]
- RFC_INT [Page 102]
- RFC_INT1 [Page 103]
- RFC_INT2 [Page 104]
- RFC_BYTE [Page 105]
- RFC_BCD [Page 106]
- RFC_DATE [Page 107]
- RFC_TIME [Page 108]
- RFC_FLOAT [Page 109]

RFC_CHAR

ABAP data type C, blank padded character string of fixed length.

This data type is defined in SAPRFC.H.

RFC_HANDLE

ABAP handle for RFC connection.

This data type is defined in SAPRFC.H.

- RFC_HANDLE_NULL

  RFC handle is not set

RFC_FUNCTIONNAME

RFC function name. This type is a character field big enough to contain the name of any ABAP function module.

This data type is defined in SAPRFC.H.

RFC_TID

ABAP null-terminated string giving an RFC transaction a globally unique identifier

This data type is defined in SAPRFC.H.

RFC_NUM

ABAP data type N, character string of fixed length containing only digits.

This data type is defined in SAPRFC.H.

RFC_INT

ABAP data type **I**, four-byte signed integer.

This data type is defined in SAPRFC.H.

RFC_INT1

Data Dictionary data type INT1, one-byte unsigned integer.

This data type is defined in SAPRFC.H.

RFC_INT2

Data Dictionary data type INT2, two-byte signed integer.

This data type is defined in SAPRFC.H.

RFC_BYTE

ABAP data type X, raw data.

This data type is defined in SAPRFC.H.

RFC_BCD

ABAP data type P, packed number in BCD (Binary Coded Decimal) format.

This data type is defined in SAPRFC.H.

RFC_DATE

ABAP data type D, eight-byte date (YYYYMMDD).

This data type is defined in SAPRFC.H.

RFC_TIME

ABAP data type T, six-byte time (HHMMSS).

This data type is defined in SAPRFC.H.

RFC_FLOAT

ABAP data type F, eight-byte IEEE floating point number.

This data type is defined in SAPRFC.H.

Function Reference

The following function references are available:

Administration Function Reference

This section contains the following topics:

In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __loadds __pascal __far.

RfcAbort

You can abort a connection (close it with an error) with:

```
void  SAP_API  RfcAbort(RFC_HANDLE handle, char * text)
```

Thus this function sends an error message, if possible, and closes the connection.

A given error message cannot be sent if the receiver is not in state where it expects to receive some RFC data.

This function is defined in SAPRFC.H.

Function Parameters:

- *handle*

  Handle to RFC connection. If *handle* is RFC_HANDLE_NULL, all connections are aborted.

- *text*

  Error message.

  If you supply a *text*, the text is used as error message on the receiving side. If the *text* field is NULL, the connection is only closed.

RfcAccept

The function

```
RFC_HANDLE  SAP_API  RfcAccept(char ** argv)
```

accepts an incoming connection. You must use this routine if your program will be started by an RFC call from an SAP System (or by any other program using this API). The command line (**argv**) has to be passed to this function.

With R/3 Release 3.0C onwards this function can be used to register at an SAP gateway, and the server program can wait for the next RFC request by issueing RfcDispatch [Page 135] or RfcListen [Page 190] or RfcGetName [Page 139].

Using this functionality, an RFC server program can now already run and work as an RFC daemon. Starting any server program by an R/3 application server, by SAPGUI or via remote shell by an SAP gateway is no longer necessary.

There are two ways to define the input parameter ‘argv’

1. Working with the ‘saprfc.ini’-file:

– D

<destination pointed to an entry in ‘saprfc.ini’>

– t

Use RFC-trace

2. Working without the ‘saprfc.ini’-file:

– a

<program ID>   e.g. own_host_name.program_name

– g

<host name of the SAP gateway>

– s

<service of the SAP gateway>   e.g. sapgw00

– t

Use RFC-trace

The first of these two ways is recommended, because this way allows you to use some RFC features today and in the future without changing your RFC programs. See saprfc.ini for more details.

The three parameters above must fit with the configuration in the R/3 System (via SM59, connection type T and register mode).

For working with register mode no further changes in RFC programs are necessary.

□

You must be careful to use only the program name as program-ID, because more programs (e.g. rfcexec) might be registered at the same SAP gateway. You ought to use at least the host name of the computer where your RFC server program is running as part of the program-ID.

□

The well-known rfcexec program can be started from the command line with RFC-trace as follows:

```
rfcexec -ap10234.rfcexec -ghs0311 -xsapgw53 -t
```

or

```
rfcexec -Drfctest
```

and an entry in saprfc.ini can be defined as follows:

```
DEST=rfctest
TYPE=R
PRGOGID=p10234.rfcexec

GWHOST=hs0311
GWSERV=sapgw53
RFC_TRACE=1
```

RFC server programs working with this functionality will be called RFC server programs running in register mode.

This function is defined in SAPRFC.H.

Return value:

- Returns a valid RFC_HANDLE [Page 98] or RFC_HANDLE_NULL.

Function Parameter:

- *argv*

  command line, the line the program is started with

RfcClose

You can close an RFC connection using:

```
void  SAP_API  RfcClose(RFC_HANDLE handle)
```

This function is defined in SAPRFC.H.

Function Parameter:

- *handle*

  Handle to RFC connection.

RfcConnArgv

This function offers the same functionality as RfcConnArgv3 [Page 120], allocating a static instance of RFC_CONNOPT_VERSION_3 [Page 196] internally, however.The syntax is:

```
int  SAP_API  RfcConnArgv(char **             argv,
                    RFC_OPTIONS *         rfc_opt,
                    RFC_CONNOPT_R3ONLY *  connopt_r3only);
```

This function was used to build up the structures RFC_OPTIONS [Page 201], RFC_CONNOPT_R3ONLY [Page 195] or RFC_CONNOPT_CPIC [Page 194] from the command line in former releases and is included for backward compatibility.

Use the function RfcConnArgv3 [Page 120] instead, if you are not working on 16-bit Windows. On 16-bit Windows only RfcConnArgv is supported.

This function is defined in SAPRFC.H.

Return Value:

- returns 0.

Function Parameters:

- *argv*

  Command line to be parsed.

- *rfc_opt*

  Option structure to be filled.

- *connopt_r3only*

  Option extension for R/3 connection (a valid pointer must be supplied all the time).

See also RfcConnArgv3 [Page 120].

RfcConnArgv3

The function

```
  int  SAP_API  RfcConnArgv3(char **          argv,
                      RFC_OPTIONS *         rfc_opt,
                      RFC_CONNOPT_R3ONLY *  connopt_r3only
                      RFC_CONNOPT_VERSION_3 connopt_version_3);
```

can be used to build up the structures RFC_OPTIONS [Page 201], RFC_CONNOPT_R3ONLY [Page 195], RFC_CONNOPT_VERSION_3 [Page 196] or RFC_CONNOPT_CPIC [Page 194] from the command line.

This function is defined in SAPRFC.H.

Return Value:

- returns 0.

Function Parameter:

- *argv*

  Command line to be parsed.

- *rfc_opt*

  Option structure to be filled.

- *connopt_r3only*

  Option extension for R/3 connection (a valid pointer must be supplied all the time)

- *connopt_version_3*

  Option extension for a connection to an R/3 System of Release 3.0 or later (a valid pointer must be supplied all the time).

Comments:

The following tokens are recognized in the argv array, which must be terminated by a NULL entry:

- d <Destination>

  name of the destination

- c <NNN>

  client (sign on data)

- u <User ID>

  user ID

- p <Password>

  password

- l <Language>

  language

- 3

  R/3 mode
- 2

  CPIC mode
- t

  turn trace on
- h <Hostname>

  name of the target host
- s <NN>

  system number of the target SAP System
- g <Gateway host>

  gateway host (if not specified, the h option is used)
- x <Gateway service>

  TCP/IP service of the gateway (default: sapgwNN, where NN is the system number (-s))
- gui

  start ‘sapgui’ to be able to display SAP Dypros or Graphics (R/3 mode only)
- debug

  start communication in debug mode (R/3 mode only).

  All tokens that were interpreted by RfcConnArgv3 are removed from the argv array.

See also RfcConnArgv [Page 119].

RfcEnvironment

The *RfcEnvironment* function lets you supply your own routines for error handling and memory allocation:

```
void  SAP_API   RfcEnvironment(RFC_ENV * new_env);
typedef struct
{
   void * (* allocate)( void * old_ptr, size_t new_size);
   void   (* errorhandler)( void);
}
RFC_ENV;
```

Supplying an Allocate Routine

If you specify an *allocate* routine (that is, the address is not NULL), RFC calls your routine (instead of the default one) whenever it allocates, resizes, or frees memory. Your allocate routine should be able to perform three operations:

```
     allocate( NULL, size)             /* to allocate memory         */
     allocate( address, 0)             /* to free memory             */
     allocate( old_addr, new_size)     /* to reallocate memory       */
```

If you don't specify an allocation function, RFC uses the standard C library routines *malloc, free*, and *realloc*.

Supplying an Error Handler

If you supply an *errorhandler* function, it is always called when an error occurs within RFC.

Clearing the Environment Structure

You must clear the environment structure RFC_ENV when using the function *RfcEnvironment*. You can do this either implicitly:

```
static RFC_ENV rfcenv;
rfcenv.allocate =....;
RfcEnvironment( &rfcenv);
```

```
or explicitly:
```

```
RFC_ENV rfcenv;
memset( &rfcenv, 0, sizeof( rfcenv));
rfcenv.allocate =....;
RfcEnvironment( &rfcenv);
```

Either method maintains program flexibility, in case of future extensions to the RFC_ENV structure.

RfcConnect

The function

```
RFC_HANDLE   SAP_API   RfcConnect(char *           system_name,
                                  char *           ms_hostname,
                                  char *           group_name,
                                  char *           client,
                                  char *           user,
                                  char *           password,
                                  char *           language,
                                  int              trace,
                                  RFC_HOSTNAME     as_hostname,
                                  int *            sysnr);
```

opens an RFC connection via LOAD BALANCING.

With this function the RFC library will try to connect to an application server with the least load (LOAD BALANCING principle) within a group of predefined application servers.

□

This function is only available for R/3 Release 3.0 onwards.

Better use the call RfcOpenEx [Ext.].

This function has the following advantages:

- The load in the R/3 System is distributed to different application servers.
- RFC connections are thus independent of a specific application server (with RfcOpen [Page 126] or RfcOpenExt [Page 179] an RFC connection could only be established to a predefined application server).
- Only the host name of the message server and its port number of the according R/3 System are required in the host file and the services file.

Information about the SAP gateway, application server, system number, etc. as parameters for RfcOpen [Page 126] or RfcOpenExt [Page 179] or as parameters in sideinfo are no longer necessary. Even the sideinfo is no longer required.

□

You ought to use RfcOpen [Page 126] or RfcOpenExtV3 [Page 181] with RFC_MODE [Page 200], RFC_MODE_VERSION3 or RFC_MODE_PARAMETER instead of this function.

This function is defined in SAPRFC.H.

Return Value:

- returns a handle to the RFC connection (RFC_HANDLE [Page 98]) or RFC_HANDLE_NULL if the connection cannot be opened.

Function Parameters:

- *system_name*

name of the R/3 System

- *ms_hostname*

  host name of the message server

- *group_name*

  name of a specific group of application servers

- *client*

  signon data: client

- *user*

  signon data: user-ID

- *password*

  signon data: password

- *language*

  signon data: language

- *trace*

  trace /0/1)

- *as_hostname*

  name of the connected application server (output parameter)

- *sysnr*

  system number of the connected R/3 System (output parameter)

See also RfcOpen [Page 126], RFC_OPTIONS [Page 201], RfcOpenExtV3 [Page 181] and RFC_OPTIONS [Page 201].

RfcLastError

You can use the following function to get more information on errors that have occurred:

```
int  SAP_API   RfcLastError(RFC_ERROR_INFO * error_info);
```

This function thus describes the last error reported by some function of the RFC API.

The *errorinfo* structure is:

```
typedef struct
{
char    key[32];
char    status[128];
char    message[256];
char    intstat[128];
}
RFC_ERROR_INFO;
```

The structure RFC_ERROR_INFO is filled with more information describing the error.

This function is defined in SAPRFC.H.

Return Value:

- Returns 1 if no error occurred and 0 elsewhere.

Function Parameter:

- *error info*

  structure RFC_ERROR_INFO [Page 198] describing the error.

RfcOpen

The following function opens an RFC connection

```
RFC_HANDLE  SAP_API   RfcOpen(RFC_OPTIONS * options)
```

according to the given options and returns a handle for the connection established. The structure RFC_OPTIONS [Page 201] contains the data needed to open the connection:

```
typedef struct
{
      char *      destination;
      RFC_MODE    mode;
      void *      connopt;
      char *      client;
      char *      user;
      char *      password;
      char *      language;
      int   trace;
}
RFC_OPTIONS;
```

This function is defined in SAPRFC.H.

Return Value:

- Returns a handle to the RFC connection (RFC_HANDLE [Page 98]) or RFC_HANDLE_NULL if the connection cannot be opened.

Function Parameter:

- *options*

  connection parameters as described at structure RFC_OPTIONS [Page 201].

  □

  If you call *RfcOpen* with an invalid password, the function does not immediately fail. However, the subsequent call to *RfcCall* (or *RfcCallReceive*) will fail.

See also RfcOpenExt [Page 179] and RfcOpenExtV3 [Page 181].

RFC Client Function Reference

This section contains the following topics:

In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __loadds __pascal __far.

RfcCall

This functin calls an ABAP function module via RFC:

```
RFC_RC   SAP_API   RfcCall(RFC_HANDLE         handle,
                           char *             function,
                           RFC_PARAMETER *    parameters,
                           RFC_TABLE *        tables);
```

The structures RFC_PARAMETER [Page 204] and RFC_TABLE [Page 207] contain names and descriptions of the ‘exporting’ parameters and tables specified in the function's interface. Tables can only be internal ABAP tables. The function returns after the call-request is sent. If the function returns RFC_OK there is no guarantee that the call has already been received by the target system.

This function is defined in SAPRFC.H.

Return Value:

- Returns after sending the call request and returns either RFC_OK or RFC_FAILURE

Function Parameters:

- *handle*

  connection handle

- *function*

  function module to call

- *parameters*

  ‘exporting’ parameters

- *tables*

  ‘tables’ parameters

  □

  RfcCall can fail because you called *RfcOpen* with an invalid password. See RfcOpen [Page 126] for more information.

  □

  Calling the function module RFC_SYSTEM_INFO

```
RFC_RC                     rfc_rc;
RFC_PARAMETER              exporting[32];
RFC_TABLE                  tables[32];

exporting[0].name = NULL;
tables[0].name = NULL;

rfc_rc = RfcCall(handle, “RFC_SYSTEM_INFO”, exporting,
tables);

if(rfc_rc != RFC_OK)
```

```
{
. . .
```

RfcCallReceive

You can make an RFC call and receive return values using a single function:

```
RFC_RC  SAP_API   RfcCallReceive(RFC_HANDLE     handle,
                                 char *         function,
                                 RFC_PARAMETER *   exporting,
                                 RFC_PARAMETER *   importing,
                                 RFC_TABLE *       tables,
                                 char **        exception)
```

This function waits till it receives an answer before returning. The return values are just the same as those you would receive by calling *RfcReceive*. Note that *RfcCallReceive* can fail because you called *RfcOpen* with an invalid password. See RfcOpen [Page 126] for more information.

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK

  The call was successfully completed, and the values of the returned parameters were filled into the fields being supplied by the RFC_PARAMETER [Page 204] array.

- RFC_FAILURE

  An internal error has occurred. RfcLastError [Page 125] may give more information.

- RFC_EXCEPTION

  The callee has raised an exception. The field '*exception' points to the name of the exception. No data were transported.

- RFC_SYS_EXCEPTION

  The local or remote RFC system has raised an exception. Also, '*exception' points to the name of the exception. The connection was automatically closed by the system and RfcLastError [Page 125] may give more information on the origin of the error. Two exceptions may occur now: SYSTEM_FAILURE and COMMUNICATION_ FAILURE.

- RFC_CALL

  The callee has issued an RFC call to the caller of RfcReceive. No data are transported. The call request must be handled by using the functions RfcDispatch [Page 135] or by RfcGetName [Page 139], RfcGetData [Page 138] and RfcSendData [Page 145] before an other call to RfcReceive can be done.

Function Parameters:

- *handle*

  connection handle

- *function*

  function module to call

- *exporting*

  'exporting' parameters

- *importing*

  ‘importing’ parameters
- *tables*

  ‘tables’ parameters
- *exception*

  output parameter: pointer to exception string. This parameter is only set if RFC_EXCEPTION or RFC_SYS_EXCEPTION is returned

See also RfcCall [Page 128] and RfcReceive [Page 132].

RfcReceive

The function

```
RFC_RC  SAP_API  RfcReceive(RFC_HANDLE handle,
               RFC_PARAMETER *       parameters,
               RFC_TABLE *           tables,
               char **               exception);
```

receives the return values from a function call issued by RfcCall [Page 128].

This function thus allows you to receive the answer to an RFC call and must be called after calling *RfcCall*. The tables' description (RFC_TABLE [Page 207]) must be identical to the one used in *RfcCall*.

RfcReceive waits till the answer is received before returning. If you want to check for incoming events without waiting, use the routine *RfcListen*.

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK

  The call was successfully completed, and the values of the returned parameters were filled into the fields being supplied by the RFC_PARAMETER [Page 204] array.

- RFC_FAILURE

  An internal error has occurred. RfcLastError [Page 125] may give more information.

- RFC_EXCEPTION

  The callee has raised an exception. The field '*exception' points to the name of the exception. No data were transported.

- RFC_SYS_EXCEPTION

  The local or remote RFC system has raised an exception. Also, '*exception' points to the name of the exception. The connection was automatically closed by the system and RfcLastError [Page 125] may give more information on the origin of the error. Two exceptions may occur now: SYSTEM_FAILURE and COMMUNICATION_ FAILURE.

- RFC_CALL

  The callee has issued an RFC call to the caller of RfcReceive. No data are transported. The call request must be handled by using the functions RfcDispatch [Page 135] or by RfcGetName [Page 139], RfcGetData [Page 138] and RfcSendData [Page 145] before an other call to RfcReceive can be done.

Function Parameters:

- *handle*

  connection handle

- *parameters*

  'importing' parameters

- *tables*

‘tables’ parameters

- *exception*

  output parameter: pointer to exception string. This parameter is only set if RFC_EXCEPTION or RFC_SYS_EXCEPTION is returned

See also RfcCall [Page 128] and RfcCallReceive [Page 130].

RFC Server Function Reference

This section contains the following topics:

In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __loadds __pascal __far.

RfcDispatch

The routine *RfcDispatch* receives a single RFC request and causes it to be executed. The calling syntax is:

```
RFC_RC  SAP_API   RfcDispatch(RFC_HANDLE handle);
```

Use *RfcDispatch* after calling RfcAccept [Page 116], or after receiving an RFC_CALL return code from RfcReceive [Page 132] or RfcCallReceive [Page 130]. In order to dispatch RFC requests, you must also have registered the remote function with *RfcInstallFunction* (or *RfcWinInstallFunction*).

This function internally calls the registered function corresponding the the RFC call. The return code of the registered function is again returned by RfcDispatch.

The sample programs provide examples of how to use *RfcDispatch*. (In particular, the program *rfcexec.c* which is included in the RFC SDK) implements an RFC server.)

This function is defined in SAPRFC.H.

Return Values:

- returns RFC_OK or other RFC_RC [Page 205] return code

Function Parameter:

- *handle*

  handle to the RFC connection

Comments:

There are some function modules which are always offered automatically when using RfcDispatch. These are:

- RFC_DOCU

  Get a list of the installed function modules.

- RFC_PING

  Do nothing (used for connection test).

- RFC_SYSTEM_INFO

  Deliver some information to the RFC library used.

A typical RFC server:

```
int main(int argv, char** argv)
{
  RFC_HANDLE handle;
  RFC_RC     rc;

  handle = RfcAccept(argv);
  if(handle == RFC_HANDLE_NULL)
  {
   ... error processing
     return 1;

  }

  rc = RfcInstallFunction(...);
  if(rc != RFC_OK)
  {
     error processing
     return 1;
  }

  do
  {
     rc = RfcDispatch(handle);
  }
  while(rc == RFC_OK);

  RfcClose(handle);

  return 0;

}
```

□

When RfcDispatch(handle) calls a C-function, it passes the function a different handle than the handle passed RfcDispatch(). Trying to do an RfcGetData() on the original handle passed to RfcDispatch() malfunctions. This is due to the internal structure of the transactional RFC: Since the function call is done indirectly, and not directly, from librfc, it internally passes RFC handles of its own in order to carry out the RFC calls.

RfcGetAttributes

```
int   SAP_API   RfcGetAttributes(RFC_HANDLE      handle,
                                 RFC_ATTRIBUTES*    rfc_attributes);
```

This call returns some information about an RFC connection, such as host name, service of the connected application server and SAP gateway, the R/3 system number, client, user and language.

RfcGetAttributes can be used in an RFC client or server program.

The RFC library can only know these attributes after this RFC connection is established and at least one RFC function is called. Therefore, it should be called after an RfcReceive [Page 132] in an RFC client program or after RfcGetData [Page 138] in an RFC server program.

Return Values:

- returns 0 if no error occurred and 1 elsewhere

Function Parameters:

- *handle*

  connection handle

- *rfc_attributes*

  structure RFC_ATTRIBUTES [Page 208] describing some information about this RFC connection

See the delivered files saprfc.hlp, saprfc.h, srfctest.c or srfcserv.c for more details.

RfcGetData

To get the parameter values for the called function, use the following:

```
RFC_RC   SAP_API   RfcGetData(RFC_HANDLE        handle,
                              RFC_PARAMETER *   parameters,
                              RFC_TABLE *       tables);
```

Within a function registered via RfcInstallFunction [Page 140] or after receiving the name of the called function by RfcGetName [Page 139], the function RfcGetData can be used to receive the parameters of the function call.

Here the ITAB_H [Page 151] field in the RFC_TABLE [Page 207] record has to be initialized to NULL. The function *RfcGetData* fills in the corresponding table handle. (This is either a newly created table or an already existing one sent to the caller via another RFC call). The field itmode in an RFC_TABLE [Page 207] record determines if a received table is passed by reference or by value.

This function is defined in SAPRFC.H.

Return Values:

- returns RFC_OK or
- returns RFC_FAILURE

Function Parameters:

Function parameters:

- *handle*

  RFC connection handle
- *parameters*

  'importing' parameters (RFC_PARAMETER [Page 204])
- *tables*

  'tables' parameters (RFC_TABLE [Page 207])

RfcGetName

This call is used by the server program to identify the RFC function to be performed, and the name of the function is returned:

```
RFC_RC  SAP_API   RfcGetName(RFC_HANDLE        handle,
                             RFC_FUNCTIONNAME   functionname);
```

Besides using RfcDispatch [Page 135], it is also possible to receive RFC calls directly. The function RfcGetName must therefore be used to get the name of the called function. The calling program then has to determine the interface of the requested function module and to receive the parameters as within a function being installed via RfcInstallFunction [Page 140].

The server program will wait for the next RFC requests in register mode (see RfcAccept [Page 116]).

This function is defined in SAPRFC.H.

Return Values:

- returns RFC_OK or
- returns RFC_FAILURE

Function Parameters:

Function parameters:

- *handle*

  RFC connection handle

- *functionname*

  output parameter, name of the called function module (RFC_FUNCTIONNAME [Page 99])

See also RfcDispatch [Page 135], RfcInstallFunction [Page 140], RfcGetData [Page 138] and RfcSendData [Page 145].

□

RfcGetName is obsolete and exists only for reasons of compatibility.

A server program that uses this RFC call can only be tested completely by using transaction sm59 from an R/3 System with the R/3 library from R/3 Release 3.0D onwards.

RfcInstallFunction

To implement an RFC server, you must register every callable function with RFC. This allows the *RfcDispatch* routine to route RFC requests properly. Use the *RfcInstallFunction* routine to register callable functions.

To call *RfcInstallFunction*, use the calling syntax:

```
RFC_RC   SAP_API   RfcInstallFunction(RFC_FUNCTIONNAME
                                      functionname,
                                      RFC_ONCALL        f_ptr,
                                      char *            docu);
```

There are two possible ways to receive an RFC call. The most simple way to receive an RFC call in an external program is to register a C function to be called when a call request is received.

The function RfcInstallFunction registers a C function to be called when receiving the request for an RFC call.

The function pointer points to a function of type RFC_ONCALL [Page 170], which contains the functionality being offered as an RFC function module. 'functionname' is the name of the offered RFC function module, and the description should contain a description of the functionality as well as the interface. Newline characters can be used to start new lines.

The descriptions of the registered functions can be requested by calling the standard function module RFC_DOCU which is available in every RFC server program using the RfcDispatch [Page 135] interface.

After RfcAccept [Page 116] or after receiving the return code RFC_CALL, when calling RfcReceive [Page 132], the program has to call RfcDispatch [Page 135] which internally calls the corresponding registered function.

This function is defined in SAPRFC.H.

Return Values:

- returns RFC_OK or
- returns RFC_FAILURE (if there is no memory available to register the function)

Function Parameters:

Function parameters:

- *functionname*

  name of function as it can be called from ABAP environment (null-terminated string)
- *f_ptr*

  function to be called (must be of type RFC_ONCALL [Page 170])
- *docu*

  text describing the functionality and the parameters of the function module

The sample programs provide examples of how to use *RfcInstallFunction*. (In particular, the program *rfcexec.c* which is included in the RFC SDK implements an RFC server.)

□

RfcInstallFunctionExt [Page 188] is for RFC programs on Windows 3.x.

If your RFC server program is working with **RfcGetName** but you want to see the docu_function belonging to your RFC function (transaction sm59, system function, function list), you can use this function with NULL as function pointer.

See also RfcDispatch [Page 135] and RfcGetName [Page 139].

RfcInstallStructure

The call *RfcInstallStructure* can be used in an RFC client or server program.

If structures and/or internal tables are to be transferred from an R/3 System to external programs (that is, to programs which display or run the SAP RFC Library), only homogeneous structures/tables can be transferred. These may only consist of character-like fields (type C, D, T or N) or fields to be converted (type X or P). Integer or float fields can only be transferred as individual fields.

For functions with structured parameters or tables it is therefore necessary to install a description of the structures used to allow automatic conversion of different data representations by the RFC library.

The call *RfcInstallStructure* installs a description for a structure being used in an RFC interface.

The description must contain all (scalar) fields of the structure in correct order. A type handle is returned which can be used in RFC_PARAMETER [Page 204] and RFC_TABLE [Page 207] descriptions.

To call *RfcInstallStructure*, use the calling syntax:

```
RFC_RC  SAP_API  RfcInstallStructure(char            * name,
                                     RFC_TYPE_ELEMENT * elements,
                                     unsigned         entries,
                                     RFC_TYPEHANDLE   * pTypeHandle;
```

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK

  The structure was successfully installed. The returned type handle (RFC_TYPEHANDLE) can be used in RFC_PARAMETER or RFC_TABLE arrays.

- RFC_MEMORY_INSUFFICIENT

  Not enough memory available to register the structure.

Function Parameters:

- name

  Name of the structure. It is used only in the trace file.

- elements

  Description of the elements of the structure

- entries

  Count of (scalar) elements in the structure. The elements parameter points to an array of size entries * sizeof(RFC_TYPE_ELEMENT).

- pTypeHandle

  Returned type handle.

RfcRaise

To raise an exception while processing a received RFC call, use the function:

```
RFC_RC  SAP_API  RfcRaise(RFC_HANDLE handle, char * exception);
```

□

RfcRaise should be used instead of RfcSendData if an exception is to be raised. If RfcRaise is used, then do not send off an additional RfcSendData.

If there are tables parameters in your function module use RfcRaiseTables [Page 144] instead.

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK
- RFC_FAILURE

Function Parameters:

Function parameters:

- *handle*

  Rfc connection handle

- *exception*

  exception to be raised (null-terminated string)

See also RfcRaiseTables [Page 144].

RfcRaiseTables

```
RFC_RC  SAP_API   RfcRaiseTables(RFC_HANDLE  handle,
                                 char *      exception,
                                 RFC_TABLE * tables);
```

This function is used to raise an exception while processing an RFC call if the function module which is called has tables parameters.

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK
- RFC_FAILURE

Function Parameters:

Function parameters:

- *handle*

  Rfc connection handle
- *exception*

  exception to be raised (null-terminated string)
- *tables*

  tables parameters as received by RfcGetData [Page 138]

See also RfcRaise [Page 143].

RfcSendData

To send the result parameters back to the caller, use this function:

```
RFC_RC  SAP_API   RfcSendData(RFC_HANDLE         handle,
                            RFC_PARAMETER *      parameters,
                            RFC_TABLE *          tables);
```

The tables description (RFC_TABLE [Page 207]) must be the same as in the previous *RfcGetData* call.

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK
- RFC_FAILURE

Function Parameters:

Function parameters:

- *handle*

  Rfc connection handle

- *parameters*

  'exporting' parameters

- *tables*

  'tables' parameters (must be in the same structure as passed to RfcGetData [Page 138])

  □

  RfcRaise should be used instead of RfcSendData if an exception is to be raised.
  If RfcRaise is used, then do not send off an additional RfcSendData.

RfcWinInstallFunction

To call *RfcWinInstallFunction,* use the calling syntax:

```
RFC_RC  SAP_API
        RfcWinInstallFunction(RFC_HANDLE      handle,
                              RFC_FUNCTIONNAME functionname,
                              RFC_ONCALL *     f_ptr,
                              char *           description);
```

To implement an RFC server, you must register every callable function with RFC. This allows the RfcDispatch [Page 135] routine to route RFC requests properly. To register callable functions when you are programming in the Windows operating system, use the *RfcWinInstallFunction* routine.

RFC provides both the RfcInstallFunction [Page 140] and *RfcWinInstallFunction* routines for registering callable functions. Windows programmers may **only** use *RfcWinInstallFunction*; all other programmers can use either routine. The only difference between them is that *RfcWinInstallFunction* requires a handle parameter.

This function is defined in SAPRFC.H.

Return Values:

- RFC_OK
- RFC_FAILURE

Function Parameters:

Function parameters:

- *handle*

  Rfc connection handle
- *functionname*

  name of the RFC function
- *f_ptr*

  function pointer
- description

  help text for the function

Once you have used and registered the function with *RfcWinInstallFunction*, you can use the *RfcDispatch* routine to execute RFC requests that come in.

The sample programs provide examples of how to use *RfcInstallFunction*, and you can use *RfcWinInstallFunction* in the same way. (In particular, the program *rfcexec.c* which is included in the RFC SDK implements an RFC server.)

Table-Handling Function Reference

The following table-handling functions let you create and manipulate internal table variables:

□

Note that for all table-handling functions, the indexing of tables starts at 1,not 0.

□

In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __loadds __pascal __far.

ItAppLine

The *ItAppLine* routine appends a new row at the end of an internal table, and returns the address of the row. The new line is initialized with 0. The calling syntax for *ItAppLine* is:

```
void * ItAppLine(ITAB_H itab);
```

Lines appended to a table can be updated immediately. You do not need to use *ItGupLine* to update them.

The corresponding ABAP operation is **Append...**

This function is defined in SAPITAB.H.

Return Values:

- pointer to the appended table row, if successful
- NULL otherwise (no space available)

Function Parameter:

- *itab*

  handle of an internal table

See also ItInsLine [Page 158].

ItCpyLine

The *ItCpyLine* routine copies the contents of the row into the destination area. The length of the copy operation is implicitly assumed as the size of one row. Memory must be supplied by the caller, no check is done. The row must not be updated. The syntax is:

```
int SAP_API ItCpyLine(ITAB_H itab, unsigned line, void* dest)
```

The corresponding ABAP operation is **`Read Table...Index...`**

This function is defined in SAPITAB.H.

Return Values:

- 0

  ok
- -1

  line does not exist

Function Parameter:

- *itab*

  handle of an internal table
- *line*

  number of the desired row (row numbers start with 1, similar to Sy-Tabix in ABAP)
- *dest*

  pointer to destination area

See also ItGetLine [Page 156] and ItPutLine [Page 160].

ItCreate

The *ItCreate* routine creates an internal table. That is, it creates a control structure for an internal table, and allocates table space for the size and shape you request. The syntax is:

```
ITAB_H ItCreate(char *          name,
                unsigned        leng,
                unsigned        occu,
                unsigned        memo);
```

This function is defined in SAPITAB.H.

Return Values:

- returns handle to a table control structure (ITAB_H [Page 151]), if successful
- ITAB_NULL if there is not enough memory available

Function Parameters:

- *name*

  name of the internal table (null-terminated string) which is used to identify the table when tracing
- *leng*

  line length for the internal table
- *occu*

  occurs-value for internal table (amount of lines being allocated when first appending a line to the table)
- *memo*

  Only use heap memory for allocating table lines. This field is for internal use only. Set the parmeter to 0.

ITAB_H

Handle of an internal tablte returned by ItCreate [Page 150] and used to access a table by all the other routines of the internal table API.

This handle is defined in SAPITAB.H.

- ITAB_NULL

  invalid table handle

ItDelete

The *ItDelete* routine deletes the internal table with the given handle. That is, it frees all table space and deletes the table control structure. Then the table handle is not valid and must not be used any more. The syntax is:

```
int  SAP_API   ItDelete(ITAB_H itab);
```

The corresponding ABAP operation is **`Free...`**

This function is defined in SAPITAB.H.

Return Value:

- returns 0, if successful

Function Parameter:

- *itab*

  handle of an internal table

See also ItFree [Page 155].

ItDelLine

To delete a row from an internal table, use:

```
int  SAP_API  ItDelLine(ITAB_H itab,  unsigned line);
```

The corresponding ABAP operation is `Delete...`

This function is defined in SAPITAB.H.

Return Values:

- returns 0, if successful
- returns >0

  row does not exist
- returns <0

  other error (no space available for index, etc.)

Function Parameters:

- *itab*

  handle of an internal table
- *line*

  rom number of the line to be deleted

ItFill

You can determine the number of rows in a table by calling *ItFill*:

```
unsigned  SAP_API   ItFill(ITAB_H  itab);
```

Return Value:

- returns number of rows in an internal table

Function Parameter:

- *itab*

  handle of an internal table

ItFree

The *ItFree* routine empties a table; it deletes all rows from a table, and frees the memory allocated for these rows. The syntax is:

```
int SAP_API   ItFree(ITAB_H  itab);
```

ItFree does not destroy the table itself, however, since the *itab* control structure still exists after the call. Still, the table handle remains valid. You can add new lines to the table after using *ItFree*.

The corresponding ABAP operation is `Free...`

This function is defined in SAPITAB.H.

Return Values:

- returns 0, if successful

Function Parameter:

- *itab*

  handle of an internal table

See also ItCreate [Page 150] and ItDelete [Page 152].

ItGetLine

Use the *ItGetLine* routine to access a specific row in a table:

```
void  SAP_API   ItGetLine(ITAB_H itab,   unsigned line);
```

This routine returns the address of the *line*th line in the internal table specified by the handle *itab*.

The *ItGetLine* function is only for reading table rows: you cannot update the data in the row. If you want to update table data, use the *ItGupLine* routine.

The corresponding ABAP operation is **`Read Table... Index...`**

This function is defined in SAPITAB.H.

Return Values:

- returns pointer to a table row, if successful
- returns NULL otherwise (table row does not exist)

Function Parameters:

- *itab*

  handle of an internal table
- *line*

  number of the desired row, row numbers start with 1 similar than Sy-Tabix in ABAP

See also ItGupLine [Page 157], ItCpyLine [Page 149] and ItPutLine [Page 160].

ItGupLine

Use the *ItGupLine* routine to access and update a specific row in a table:

```
void  SAP_API   ItGupLine(ITAB_H itab,   unsigned line);
```

This routine returns the address of the requested line (specified by line). *ItGupLine* functions just like *ItGetLine*, except that you may update the data at the returned address.

The corresponding ABAP operation is **`Read Table... Index..., Modify...`**

This function is defined in SAPITAB.H.

Return Values:

- returns pointer to a table row, if successful
- returns NULL otherwise (table row does not exist)

Function Parameters:

- *itab*

  handle of an internal table
- *line*

  number of the desired row, row numbers start with 1 similar than Sy-Tabix in ABAP

See also ItGetLine [Page 156], ItCpyLine [Page 149] and ItPutLine [Page 160].

ItInsLine

The *ItInsLine* routine inserts a new row into a table. The new row is inserted immediately before the specified line number, and is not initialized in any way. *ItInsLine* returns the address of the new line.

The syntax for *ItInsLine* is:

```
void  SAP_API   ItInsLine(ITAB_H itab,   unsigned line);
```

Lines inserted with *ItInsLine* can be updated right away. You do not need to call *ItGupLine* to update them.

The corresponding ABAP operation is **`Insert... Index...`**

This function is defined in SAPITAB.H.

Return Values:

- returns pointer to a table row, if successful
- returns NULL otherwise (no space available)

Function Parameters:

- *itab*

  handle of an internal table
- *line*

  row number before which the line is to be inserted

See also ItAppLine [Page 148].

ItLeng

You can determine the length of a table row by calling *ItLeng*:

```
unsigned  SAP_API   ItLeng(ITAB_H   itab);
```

- Return value:

  Returns length of a row of the internal table.

- function parameters:

  itab (handle of an internal table).

This function is defined in SAPITAB.H.

Return Value:

- returns length of a table row

Function Parameter:

- *itab*

  handle of an internal table

ItPutLine

The *ItPutLine* routine copies the source onto the line ‘line’ of an internal table. The syntax is:

```
int  SAP_API   ItPutLine(ITAB_H itab, unsigned line, void* src)
```

The corresponding ABAP operation is **`Modify... Index...`**

This function is defined in SAPITAB.H.

Return Values:

- returns 0, if successful
- returns -1 if an error occurred

Function Parameters:

- *itab*

  handle of an internal table
- *line*

  row number onto which ‘src’ is copied
- *src*

  pointer to source, source is copied in the length of a table row

See also ItGetLine [Page 156], ItCpyLine [Page 149] and ItGupLine [Page 157].

Transactional Function Reference

This section contains the following topics:

In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __loadds __pascal __far.

RfcCreateTransID

The function

```
RFC_RC  SAP_API  RfcCreateTransID(void)
```

gets a transaction-ID for a following call of a function module in R/3 using the transactional RFC interface in R/3.

With this function a new TransID will be produced from the R/3 System. The RFC client program has to call a function module in R/3 via RfcIndirectCall [Page 163] with this TransID.

If an error occurs (e.g. communication error) during the call of a function module in R/3 via RfcIndirectCall, the RFC client program has to reconnect the RFC connection and repeat the RfcIndirectCall without creating a new TransId.

This function is defined in SAPRFC.H.

See also RfcIndirectCall [Page 163] and RFC_TID [Page 100].

RfcIndirectCall

The function

```
RFC_RC   SAP_API   RfcIndirectCall(RFC_HANDLE        handle,
                                   char *            function,
                                   RFC_PARAMETER *   exporting,
                                   RFC_TABLE *       tables,
                                   RFC_TID           tid);
```

calls a function module in R/3 indirectly.

With this function, the call of a function module in R/3 will use the transactional RFC interface in R/3.

Importing parameters are not supported.

If an error occurs (almost only communication errors), the RFC client program has to reconnect to R/3 later and repeat this RFC call with the specific TransID. It must **not** create a new TransID via RfcCreateTransID [Page 162].

This function is defined in SAPRFC.H.

Function Parameters:

- *handle*

  connection handle
- *function*

  function module to call
- *exporting*

  'exporting' parameters
- *tables*

  'tables' parameters
- *tid*

  corresponding transaction-ID

See also RfcCreateTransID [Page 162] and RFC_TID [Page 100].

RfcInstallTransactionControl

This function installs four functions to control transactional behaviour. It is only to be used with transactional RFC. The syntax is:

```
void SAP_API RfcInstallTransactionControl
                   (RFC_ON_CHECK_TID     onCheckTid,
                   RFC_ON_COMMIT        onCommit,
                   RFC_ON_ROLLBACK      onRollback,
                   RFC_ON_CONFIRM_TID   onConfirmTid)
```

This function was introduced to allow an RFC server program to ensure exactly-once behaviour for functions being called via tRFC in ABAP: RfcInstallTransactionControl must thus be called by RFC server program before the RfcDispatch loop is entered if this program wants to receive transactional RFC calls and must ensure that RFC calls are done excatly once.

Without installing these functions it can only be guaranteed that an RFC function call issued by 'Call Function... In Background Task' is done at least once. Then all function modules offered by such a server program which are called via 'Call Function... In Background Task' must cope with being called more then once.

If installed, the first function onCheckTid is called, if a transactional RFC is to be called. The actual Transaction ID is passed (RFC_ON_CHECK_TID [Page 166]). The function has to store this transaction-ID in permanent storage and return 0. If the same function will be called later again with the same transaction-ID, it has to make sure that it will return a non-zero value. If the same transaction is already running by another process but is not completed, the function has to wait until the transaction completes.

The second function is called if all the RFC function module calls are done and the local transaction can be completed. It should be used to locally commit the transaction, if necessary (RFC_ON_COMMIT [Page 167]). This function is 'void'.

The third function is called instead of the second function, if, from the point of view of the RFC library, there occurred an error while processing the local transaction. This function can be used to roll back the local transaction (RFC_ON_ROLLBACK [Page 169]). This function is 'void'.

The fourth function will be called if the local transaction is completed also from the point of view of the calling system and all information on this transaction-ID can be discarded. This function is 'void': RFC_ON_CONFIRM_TID [Page 168].

This function is defined in SAPRFC.H.

Function Parameters:

- *onCheckTid*

  Function to be called when local transaction starts. Must be used to check if the transaction is already running or was already completed.

- *onCommit*

  Function to be called when local transaction ends. Should be used to commit the transaction locally.

- *onRollback*

  Function to be called if local transaction fails due to an error found while the processing is done inside the RFC library. Should be used to roll back a local transaction.

- *onConfirmTid*

  Function to be called when local transaction is confirmed. All information stored locally about this transaction can be deleted.

See also RFC_TID [Page 100].

RFC_ON_CHECK_TID

This function is to be installed by RfcInstallTransactionControl [Page 164]. The syntax is:

```
int RFC_ON_CHECK_TID(RFC_TID transactionId)
```

The function is called when a local transaction is starting. Since a transactional RFC call can be issued many times by the client system, the function is responsible for storing the transaction-ID in permanent storage. If the client system tries starting the same transaction a second time, the function has to return 1.

Return Values:

- Returns 0

  transaction-ID stored, transaction can be started

- Returns 1

  transaction already done, skip the request

- Returns < 0

  cannot lock transaction, internal error

Function Parameter:

- *transactionId*

  actual transaction-ID

  □

  If this functions has access to a SQL database, it should perform an operation like Insert Into SomeTable values (:transactionId);
  where the table 'SomeTable' must have a unique index over the transaction-ID. If another process has also inserted the same transaction-ID, the second process has to wait until the transaction is completed by the first process. If the transaction is completed by another process (the Insert command returns with some 'Duplicate Record' error code), the function must return 1 to indicate to skip the transaction.

RFC_ON_COMMIT

This function is to be installed by RfcInstallTransactionControl [Page 164]. The syntax is:

```
int RFC_ON_COMMIT(RFC_TID transactionId)
```

The function is called when a local transaction ends.

The function is to be used to commit the local transaction, if necessary.

This function is defined in SAPRFC.H.

Function Parameter:

- *transactionId*

  Transaction-ID

  □

  If this functions has access to a SQL database, it should perform an operation like Commit work;

RFC_ON_CONFIRM_TID

This function is to be installed by RfcInstallTransactionControl [Page 164]. The syntax is:

```
void RFC_ON_CONFIRM_TID(RFC_TID transactionId)
```

The function is called when a local transaction is completed. All informations stored about that transaction can be discarded by the server.

In general, this function can be used to delete the transaction-ID from permanent storage.

This function is defined in SAPRFC.H.

Function Parameter:

- *transactionId*

  Transaction-ID

  □

  If this functions has access to a SQL database, it should perform an operation like
  Delete From SomeTable where key = :transactionId; Commit Work;
  where the table 'SomeTable' should have a unique index over the transaction-ID.

RFC_ON_ROLLBACK

This function is to be installed by RfcInstallTransactionControl [Page 164]. The syntax is:

```
int RFC_ON_ROLLBACK(RFC_TID transactionId)
```

The function is called when a local transaction ends with failure.

The function is to be used to roll back the local transaction, if necessary.

This function is defined in SAPRFC.H.

Function Parameter:

- *transactionId*

  Transaction-ID

  □

  If this functions has access to a SQL database, it should perform an operation like Rollback work;

RFC_ONCALL

RFC_ONCALL is a type of C-function to be installed by RfcInstallFunction [Page 140] or RfcInstallFunctionExt [Page 188]. The syntax is:

```
RFC_RC RFC_ONCALL(void)
```

This function is defined in SAPRFC.H.

Extended Function Reference

Visual Basic programs that make RFC calls without using SAP-generated stub programs need the extended functions described in this section. (SAP-generated stub programs are described in The RFC Generator [Ext.]).

□

If possible, the extended functions described in this section should only be used in Visual Basic stubs, except for RfcOpenExt.

SAP offers the extended functions because Basic does not pass table structures in a manner compatible with RFC API functions. (In particular, structures containing character strings are problematic). Structure-passing is required when you use the non-extended API functions. Both Basic and C programs may call the extended functions.

The extended functions perform the same RFC tasks, but include specialized parameter-passing. Routines are provided for:

- allocating and freeing stack space
  - RfcAllocParamSpace [Page 176]
  - RfcFreeParamSpace [Page 184]
- storing and retrieving parameters in stack storage
  - RfcAddImportParam [Page 174]
  - RfcAddExportParam [Page 173]
  - RfcAddTable [Page 175]
- performing RFC tasks,

  passing the stack storage instead of RFC_PARAMETER [Page 204] / RFC_TABLE [Page 207] structures:
  - RfcOpenExt [Page 179]
  - RfcOpenExtV3 [Page 181]
  - RfcCallExt [Page 177]
  - RfcReceiveExt [Page 186]
  - RfcCallReceiveExt [Page 178]
  - RfcGetDataExt [Page 185]
  - RfcSendDataExt [Page 187]
  - RfcInstallFunctionExt [Page 188]

Entry points for the extended functions are all declared in *saprfc.h*. The program *testrfc.c* which is included in the RFC SDK shows how to use the RFC Extended Functions to make an RFC call.

□

In many of these routines, the token SAP_API is included. SAP_API contains platform-dependent keywords which are neccesary to allow dynamic linking of these function from various environments. On Windows, for example, SAP_API is __extern __loadds __pascal __far.

RfcAddExportParam

Before transferring data with the extended functions, you must add all parameters and tables to the parameter stack.

A calling program that exports parameters (RfcCallExt [Page 177] /RfcCallReceiveExt [Page 178]), or a called program that returns them (RfcSendData [Page 145]), must add export parameters to the stack with *RfcAddExportParam*:

```
RFC_RC  SAP_API
   RfcAddExportParam(RFC_PARAM_SPACE     PSpace,
                   unsigned               ParamNo,
                   void *                 name,
                   unsigned               nlen,
                   unsigned               type,
                   unsigned               leng,
                   void *                 addr);
```

Function Parameters:

- Pspace

  Address of the stack space area.

- ParamNo

  Index for the parameter in the export-parameter stack space.

- name, nlen

  type, leng

  addr

These fields have the same meanings as the corresponding RFC_PARAMETER [Page 204] fields. (See ItInsLine [Page 158])

□

Remember to allocate the stack space with RfcAllocParamSpace [Page 176] before calling *RfcAddExportParam*.

RfcAddImportParam

Before transferring data with the extended functions, you must add all parameters and tables to the parameter stack.

A calling program that receives return values (RfcReceiveExt [Page 186] /RfcCallReceiveExt [Page 178]), or a called program that receives import parameters (RfcGetData [Page 138]), must add import parameters to the stack with *RfcAddImportParam*:

```
RFC_RC  SAP_API
   RfcAddImportParam(RFC_PARAM_SPACE     PSpace,
                     unsigned            ParamNo,
                     void *              name,
                     unsigned            nlen,
                     unsigned            type,
                     unsigned            leng,
                     void *              addr);
```

Function Parameters:

Parameters for *RfcAddImportParam* have the same meanings as for *RfcAddExportParam*.

- Pspace

  Address of the stack space area.

- ParamNo

  Index for the parameter in the import-parameter stack space.

- name, nlen

  type, leng

  addr

These fields have the same meanings as the corresponding RFC_PARAMETER [Page 204] fields. (See ItInsLine [Page 158])

□

Remember to allocate the stack space with RfcAllocParamSpace [Page 176] before calling *RfcAddImportParam*.

RfcAddTable

Before transferring data with the extended functions, you must add all parameters and tables to the parameter stack.

Both calling and called programs must add table parameters to the stack, since tables are both exported and imported. To do this, use the function:

```
RFC_RC  SAP_API  RfcAddTable(RFC_PARAM_SPACE PSpace,
                             unsigned        TableNo,
                             void *          name,
                             unsigned        nlen,
                             unsigned        type,
                             unsigned        leng,
                             ITAB_H          ithandle);
```

Function Parameters:

- Pspace

  Address of the stack space area.

- TableNo

  Index for the parameter in the table stack space.

- name, nlen

  type, leng

These fields have the same meanings as the corresponding RFC_PARAMETER [Page 204] fields. (See ItInsLine [Page 158])

- ithandle

  Table handle. (Corresponds to the *itab_h* field in the RFC_TABLE [Page 207] structure.) Each table parameter must have a table handle: if you are initiating an RFC call, call *ItCreate* to create a handle. (See ItCreate [Page 150])

  □

  Remember to allocate the stack space with RfcAllocParamSpace [Page 176] before calling *RfcAddTable*.

RfcAllocParamSpace

Before storing parameters in the parameter stack, you must allocate the stack space. To do this, call:

```
RFC_PARAM_SPACE  SAP_API   RfcAllocParamSpace(unsigned   MaxEx,
                                    unsigned   MaxIm,
                                    unsigned   MaxTab);
```

The parameters *MaxEx, MaxIm*, and *MaxTab* tell how many export, import and table parameters the calling program will use when invoking the function module. *RfcAllocParamSpace* then allocates enough space (in each category) for the requested number of parameters.

As in ABAP, the meaning of the terms "export" and "import" depend on point of view. The exporting parameters for a calling program are the importing parameters for the function being called. Thus an RFC caller that sends two export parameters, no import parameters, and a table should call:

```
    RfcAllocParamSpace(2, 0, 1);
```

to create its stack structure. The function called, for the same set of parameters, would allocate stack space with:

```
    RfcAllocParamSpace(0, 2, 1);
```

The return value (RFC_PARAM_SPACE) is a pointer to the stack space allocated.

RfcCallExt

Call the remote function using:

```
RFC_RC  SAP_API   RfcCallExt(RFC_HANDLE      handle,
                             RFC_PARAM_SPACE   PSpace,
                             char *            function);
```

The parameter *PSpace* is a pointer to the stack area where parameters have been stored.

The function *RfcCallExt* returns after sending the call request, and returns either RFC_OK or RFC_FAILURE.

□

RfcCallExt can fail because you called *RfcOpenExt* with an invalid password. See RfcOpenExt [Page 179] for more information.

RfcCallReceiveExt

You can make an RFC call and receive return values using a single function:

```
RFC_RC  SAP_API  RfcCallReceiveExt(RFC_HANDLE     handle,
                                   RFC_PARAM_SPACE   PSpace,
                                   char *            function,
                                   char **           exception);
```

This function waits till it receives an answer before returning. The return values are just the same as those you would receive by calling *RfcReceiveExt*. (See the RfcReceiveExt [Page 186] function for a list of return values.)

□

RfcCallReceiveExt can fail because you called *RfcOpenExt* with an invalid password. See RfcOpenExt [Page 179] for more information.

RfcOpenExt

□

Better use the call RfcOpenEx [Ext.].

The following function opens an RFC connection. All parameters are passed as single fields:

```
RFC_HANDLE  SAP_API   RfcOpenExt(char*     destination,
                                 RFC_MODE  mode,
                                 char*     hostname,
                                 int       sysnr,
                                 char*     gateway_host,
                                 char*     gateway_service,
                                 char*     client,
                                 char*     user,
                                 char*     password,
                                 char*     language,
                                 int       trace);
```

RFC opens the connection using the given options, and returns a handle for the connection established.

The input parameters to this function carry the same information as that sent to RfcOpen, but not packed in structures. Instead, individual fields are sent. At any given time, some of these fields are empty, depending on the value of mode:

- RFC_MODE_R3ONLY

  The fields *hostname, sysnr, gateway_host*, and *gateway_service* have valid information in them.

- RFC_MODE_CPIC

  Only the fields *gateway_host*, and *gateway_service* have valid information in them.

This function is defined in SAPRFC.H.

Return Values:

- Returns a handle to the RFC connection (RFC_HANDLE [Page 98]) or
- Returns RFC_HANDLE_NULL

Function Parameters:

- *destination*

  name of destination

- *mode*

  connection mode (RFC_MODE [Page 200])

- *hostname*

  hostname of target system

- *sysnr*

system number (0-99)

- *gateway_host*

  gateway hostname or NULL

- *gateway_service*

  gateway service or NULL

- *client*

  signon data: client

- *user*

  signon data: user

- *password*

  signon data: password

- *language*

  signon data: language

- *trace*

  If the field *trace* contains a non-zero value, the outgoing and incoming RFC data are written to a trace file *dev_rfc* in the current directory. A trace file is also written on the target system. On R/3 Systems, you can view this file using the utility program RSRFCTRC.

See also RfcOpen [Page 126], RfcOpenExtV3 [Page 181] and RFC_OPTIONS [Page 201] for further description of these fields. In general, the input parameters allow the system to avoid accessing the *sideinfo* table. If you send the relevant parameters as null fields (*hostname* for R/3, the *gateway* fields for R/2), the *sideinfo* table will be accessed to get the needed information.

If you call *RfcOpenExt* with an invalid password, the function does not immediately fail. However, the subsequent call to RfcCallExt [Page 177] (or RfcCallReceiveExt [Page 178]) will fail.

RfcOpenExtV3

□

Better use the call RfcOpenEx [Ext.].

The following function opens an RFC connection.

```
RFC_HANDLE  SAP_API  RfcOpenExtV3(char*      destination,
                                  RFC_MODE    mode,
                                  char*       hostname,
                                  int         sysnr,
                                  char*       gateway_host,
                                  char*       gateway_service,
                                  char*       client,
                                  char*       user,
                                  char*       password,
                                  char*       language,
                                  int         trace
                                  RFC_INT     use_load_balancing,
                                  char *      lb_host,
                                  char *      lb_system_name
                                  char *      lb_group
                                  RFC_INT     use_sapgui);
```

All parameters are passed as single fields (using RFC_MODE [Page 200]).

The following parameters are always needed in this call: RFC_MODE [Page 200] mode and SAP logon information: client, user, password, language, trace, and use_sapgui.

The following parameters are necessary, depending on RFC_MODE [Page 200]:

- RFC_MODE_CPIC:
  - destination must be defined
  - gateway-host and gateway_service can be defined
  - sideinfo is necessary.
- RFC_MODE_R3ONLY/RFC_MODE_VERSION_3
  - use_load_balancing is 0 (OFF)
    - destination is not NULL:

      gateway_host, gateway_service, hostname and sysnr can be defined. sideinfo is necessary.
    - destination is NULL:

      gateway_host and gateway_service can be defined. hostname and sysnr must be defined.
  - use_load_balancing is 1 (ON)

    destination, gateway_host, gateway_service, hostname and sysnr will not be evaluated.

lb_host and lb_system_name must be defined. If lb_group is NULL, the group 'PUBLIC' will be used.

□

use_sapgui and ABAP-Debug are only available with R/3 3.0C or later and not on Windows with a 16-bit-RFC-library.

use_sapgui and ABAP-Debug are not available with R/2.

This function is defined in SAPRFC.H.

Return Values:

- Returns a handle to the RFC connection (RFC_HANDLE [Page 98]) or
- Returns RFC_HANDLE_NULL

Function Parameters:

- *destination*

  name of destination or NULL
- *mode*

  connection mode (RFC_MODE [Page 200])
- *hostname*

  hostname of target system or NULL
- *sysnr*

  system number (0-99)
- *gateway_host*

  gateway hostname or NULL
- *gateway_service*

  gateway service or NULL
- *client*

  signon data: client
- *user*

  signon data: user
- *password*

  signon data: password
- *language*

  signon data: language
- *trace*

  trace (OFF/ON/ABAP-DEBUG or also trace ON and ABAP-DEBUG: 0/1/2/3)
- *use_load_balancing*

using load balancing feature (OFF/ON/0/1)

- *lb_host*

  host name of the ‘message server’

- *lb_system_name*

  name of the target system (e.g. C11)

- *lb_group*

  application server group or NULL for PUBLIC

- *use_sapgui*

  using sapgui to display SAP dynpros and graphics (OFF/ON: 0/1)

See also RfcOpen [Page 126] and RFC_OPTIONS [Page 201].

RfcFreeParamSpace

The *RfcFreeParamSpace* function de-allocates the storage allocated with *RfcFreeParamSpace*. The syntax is:

```
RFC_RC  SAP_API   RfcFreeParamSpace(RFC_PARAM_SPACE  PSpace);
```

RfcGetDataExt

To get incoming parameter values when a function is being called, use the following:

```
RFC_RC  SAP_API  RfcGetDataExt(RFC_HANDLE     handle,
                               RFC_PARAM_SPACE    PSpace);
```

Before calling *RfcGetDataExt*, you must have set up parameter and table structures in *PSpace* that are defined like RFC_PARAMETER [Page 204] and RFC_TABLE [Page 207]. The program *testrfc.c* which is included in the RFC SDK gives an example of how to define and fill these structures before adding them to *PSpace*.

RfcGetDataExt stores all parameter values at the addresses specified by the structures in *PSpace*. If table parameters are being passed in, *RfcGetDataExt* fills in the corresponding table handle. (The table can be either newly created or an already existing one sent to the caller via another RFC call).

RfcReceiveExt

The function

```
  RFC_RC  SAP_API  RfcReceiveExt(RFC_HANDLE       handle,
                                 RFC_PARAM_SPACE  PSpace,
                                 char **          exception);
```

allows you to receive return values from an RFC call. *RfcReceiveExt* must be called after calling RfcCallExt [Page 177].

RfcReceiveExt waits till the answer is received before returning. If you want to check for incoming events without waiting, use the routine RfcListen [Page 190].

Return Values:

RfcReceiveExt returns the same values as RfcReceive [Page 132].

- RFC_OK

  The call was successfully completed. Return parameter values have been placed in *PSpace*, in the appropriate RFC_PARAMETER_STACK areas. (See RfcAllocParamSpace [Page 176] for more information.)

- RFC_FAILURE

  An internal error has occurred. You can use the function RfcLastError [Page 125] to get more information.

- RFC_EXCEPTION

  The remote function you called has raised an exception. The field **exception* points to the name of the exception. No data was returned.

- RFC_SYS_EXCEPTION

  The local or remote RFC system has raised an exception. The field **exception* points to the name of the exception. The connection was automatically closed by the system; you can call RfcLastError [Page 125] to get information on the origin of the error. Two exceptions are possible: SYSTEM_FAILURE and COMMUNICATION_FAILURE.

- RFC_CALL

  The callee has issued an RFC call (a "call back") to the caller of *RfcReceiveExt*. No data has been returned. The call request must be handled by using the functions *RfcGetNameExt*, RfcGetDataExt [Page 185] and RfcSendDataExt [Page 187] before another call to *RfcReceiveExt* can be done.

RfcSendDataExt

To send the result parameters back to the caller, use the function *RfcSendDataExt*:

```
RFC_RC  SAP_API   RfcSendDataExt(RFC_HANDLE       handle,
                                 RFC_PARAM_SPACE  PSpace);
```

RfcInstallFunctionExt

The *RfcInstallFunctionExt* function is an alternative function, replacing RfcInstallFunction [Page 140] for Windows 3.x (16-bit). The function needs a valid RFC handle as an additional parameter. The syntax is:

```
RFC_RC SAP_API RfcInstallFunctionExt(RFC_HANDLE     handle,
                                     RFC_FUNCTIONNAME functionname,
                                     RFC_ONCALL     f_ptr,
                                     char*          docu)
```

The function module is installed only for the given RFC connection.

This function is defined in SAPRFC.H.

Return Values:

- Returns RFC_OK or
- Returns RFC_FAILURE if there is no memory available to register the function

Function Parameters:

- *handle*

  Valid RFC connection handle, as returned by RfcAccept [Page 116].
- *functionname*

  Name of function as it can be called from ABAP environment. Null terminated string.
- *f_ptr*

  Function to be called. Must be of type RFC_ONCALL [Page 170].
- *docu*

  Text describing the functionality and the parameters of the function module.

  □

  The function is available on all supported platforms to ease porting.

See also RfcInstallFunction [Page 140], RfcDispatch [Page 135] and RfcGetName [Page 139].

Special Function Reference

The following special functions are available:

RfcListen [Page 190]

RfcWaitForRequest [Page 192]

RfcListen

```
RFC_RC  SAP_API  RfcListen(RFC_HANDLE  handle);
```

can be used to check if there is an RFC request available.

You may not always want to wait for the answer to an RFC call. You can use the following function (instead of RfcReceive [Page 132]) to poll for incoming RFC events.

The function always returns immediately. If RfcListen returns RFC_OK, RfcReceive has to be called to handle the event. It is only possible to listen for one incoming RFC message at a time.

It is also possible to use RfcListen while waiting for a new RFC request by RfcDispatch [Page 135] with or without register mode (see RfcAccept [Page 116]). Because RfcListen will return immediately, it is recommended to use RfcWaitForRequest [Page 192] to wait for new RFC requests.

This function is defined in SAPRFC.H.

Return Values:

- returns RFC_OK

  an RFC event is pending (call or return)

- returns RFC_RETRY

  nothing has arrived yet

- returns RFC_FAILURE

  an error has occurred

Function Parameter:

- *handle*

  RFC connection handle

□

Using RfcListen while waiting for the answer of a remote function call:

```
RFC_RC  rc;

rc = RfcCall(handle,...);

do
{
  rc = RfcListen(handle);
  if(rc == RFC_RETRY)
  {
    // while waiting for the RFC answer, do something...
  ...
  }
} while (rc == RFC_RETRY);

if(rc == RFC_OK)
  rc = RfcReceive(handle,...);
```

```
...
```

□

Using RfcListen and RfcDispatch:

```
RFC_RC  rc;
RFC_HANDLE handle = RfcAccept(argv);

...

do
{
  // Waiting for the next RFC request
  for(rc = RFC_RETRY; rc == RFC_RETRY;)
  {
    rc = RfcListen(handle);
    if(rc == RFC_RETRY)
    {
      // while waiting for the next RFC request, do
something...
    ...
    }
  }
  if (rc != RFC_OK)
     break;

  rc = RfcDispatch(handle);

} while(rc == RFC_OK);

RfcClose(handle);
exit(0);
```

□

A lot of calls to the RFC library “block”, that is, they do not return until communications are complete. You can overcome this obstacle by using the RfcListen function: RfcListen checks whether any requests are present. If there are none, then programs can be started and run. If, however, a call is being processed, then RfcListen puts the results in a buffer and dispatches them, so that no blocking occurs. Thus, the results will not be lost.

RfcWaitForRequest

```
RFC_RC   SAP_API   RfcWaitForRequest(RFC_HANDLEhandle,
                                    RFC_INT     wtime);
```

Using the registering feature [Page 49] of an R/3 System Release 3.0C onwards, an RFC server program can issue RfcAccept [Page 116] to register itself at an SAP gateway. After that, it can wait for any RFC request by issueing RfcGetName [Page 139], RfcDispatch [Page 135] or RfcListen [Page 190].

RfcGetName and RfcDispatch are blocking calls. The server has to wait until any RFC request is incoming.

In R/3 Release 3.0C and 3.0D, RfcListen is also a blocking call when using registering mode.

From R/3 Release 3.0E onwards, RfcListen is a **non**-blocking call. However, if you do not issue the RfcListen call quick enough, the SAP gateway will return an error to the RFC client, because no server program was available at that time.

Instead of RfcListen, you can use this new call RfcWaitForRequest with a time interval in seconds as a parameter to define how long you want to wait for RFC requests.

This function is defined in SAPRFC.H.

Function Parameters:

- *handle*

  RFC connection handle

- *wtime*

  Wait time

  □

  This call is available with the RFC library from R/3 Release 3.0E onwards.

For more details, see saprfc.h, srfcserv.c or saprfc.hlp in the delivered RFC SDK.

Structures and Enumerations

This section contains the following topics:

RFC_CONNOPT_CPIC

This structure provides options for an SNA connection via the SAP gateway. The connection data must be specified at the SAP gateway. At the SAP gateway, ‘destination’ is used as key for the ‘sideinfo’ file.

```
typedef struct {
      char * gateway_host;
      char * gateway_service;
} RFC_CONNECT_CPIC;
```

This structure is defined in SAPRFC.H.

Members:0

- gateway_host

  gateway hostname

- gateway_service

  gateway service (in general sapgw00)

RFC_CONNOPT_R3ONLY

This structure provides options for a connection to an R/3 System.

```
typedef struct {
      char * hostname;
      int sysnr;
      char * gateway_host;
      char * gateway_service;
} RFC_CONNOPT_R3ONLY;
```

This structure is defined in SAPRFC.H.

Members:

- hostname

  host name of target system

  Host names can be regular host names defined in a ‘hosts’ file, an IP address like 123.123.123.123 or a saprouter address as /H/hostname/S/port/H/host/S/port/...

- sysnr

  system number (0-99)

  This is the number by which the target system is identified. In general, this is 0.

- gateway_host

  gateway hostname

  If the pointer is NULL, the gateway is assumed to run at ‘hostname’.

- gateway_service

  gateway service

  If the pointer is NULL, the service “sapgw##” with ## = ‘sysnr’ is assumed.

RFC_CONNOPT_VERSION_3

This structure provides a connection to an R/3 System. The target system must be Release 3.0C or later.

```
typedef struct {
      char * hostname;
      RFC_INT sysnr;
      RFC_INT use_load_balancing;
      char * lb_host
      char * lb_system_name;
      char *lb_group;
      RFC_INT use_sapgui;
} RFC_CONNOPT_VERSION_3;
```

Since an R/3 System of version 3.0 always has its own ‘gateway’ process, no gateway options are necessary any more.

This structure is defined in SAPRFC.H.

Members:

- hostname

  host name of target system

  Host names can be regular host names defined in a ‘hosts’ file, an IP address like 123.123.123.123 or a saprouter address as /H/hostname/S/port/H/host/S/port/...

  If ‘use_load_balancing’ is set to a non-zero value, this field can be NULL.

- sysnr

  system number (0-99)

  This is the number by which the target system is identified. In general, this is 0.

  If ‘use_load_balancing’ is set to a non-zero value, this field is ignored.

- use_load_balancing

  Use the load balancing features of an SAP System.

  If you set this to a non-zero value, ‘hostname’ and ‘sysnr’ are ignored.

  You must set the fields ‘lb_host’ and ‘lb_system_name’ instead.

  The target system is then determined dynamically.

  lb_host

  Host name where the ‘message server’ of the target system is running.

  This field must only be filled if ‘use_load’balancing’ is set to a non-zero value

- lb_system_name

  name of the target system (e.g. C11)

  This field must only be filled if ‘use_load_balancing’ is set to a non-zero value.

- lb_group

application server group

The group of application servers to be used.

This field must only be filled if ‘use_load_balancing’ is set to a non-zero value.

- use_sapgui

use ‘sapgui’ processes to display SAP dynpros and graphics.

Set this to a non-zero value to activate this functionality.

□

Starting sapgui takes some time, so you should not set that value if you do not need it.

This field is set automatically as soon as you activate the ABAP debugger by entering ‘d’ in the **trace** field or by setting RFC_DEBUG in the system environment.

RFC_ERROR_INFO

The structure

```
typedef struct {
      char key[33] ;
      char status[128];
      char message[256];
      char intstat[128];
} RFC_ERROR_INFO;
```

is returned by RfcLastError [Page 125] describing the last RFC error that occurred.

This structure is defined in SAPRFC.H.

Members:

- key[33]

  error code to identify the error

- status128]

  state of the RFC connection

- message[256]

  text describing the error

- intstat[128]

  internal description of the RFC connection

RFC_ITMODE

The structure

```
enum RFC_ITMODE {
      RFC_ITMODE_BYREFERENCE,
      RFC_ITMODE_BYVALUE,
      RFC_ITMODE_KEEPALIVE,
};
```

provides a mode how to pass an internal table.

This structure is defined in SAPRFC.H.

Members:

- RFC_ITMODE_BYREFERENCE

  table is passed by reference

  □

  You must always use RFC_ITMODE_BYREFERENCE.

- RFC_ITMODE_BYVALUE

  table is passed by value, changes are not transported back

  □

  RFC_ITMODE_BYVALUE is for internal use only.

- RFC_ITMODE_KEEPALIVE

  table is passed by reference, but is kept alive after returning (i.e. after RfcSendData)

  □

  RFC_ITMODE_KEEPALIVE is for internal use only.

RFC_MODE

The structure

```
enum RFC_MODE {
      RFC_MODE_R3ONLY,
      RFC_MODE_CPIC,
      RFC_MODE_VERSION_3
      RFC_MODE_PARAMETER
};
```

provides a type of connection to be openend by RfcOpen [Page 126].

This structure is defined in SAPRFC.H.

Members:

- RFC_MODE_R3ONLY

  Use R/3 protocol and addressing scheme. Only for R/3 Systems. Any kind of user-ID (dialog user-ID, cpic user-ID) is possible.

  connopt must point to a structure of type RFC_CONNOPT_R3ONLY [Page 195].

- RFC_MODE_CPIC

  Use R72 protocol and addressing scheme. Must be used for R/2, only cpic user-IDs are allowed. Since an R73 System is also capable of understanding the R/2 RFC protocol, you can also reach an R/3 System with that mode. However, you must use a 'sideinfo' file for addressing.

  connopt must point to a structure of type RFC_CONNOPT_CPIC [Page 194].

- RFC_MODE_VERSION3

  Use R/3 protocol version 3. The receiving SAP System must have at least Release 3.0C to be able to serve every kind of options.

  connopt must point to a structure of type RFC_CONNOPT_VERSION_3 [Page 196].

- RFC_MODE_PARAMETER

  Use R/3 protocol version 3 or R/2 protocol and addressing scheme. This mode includes all three modes above and all necessary parameters will be read from an INI-file (saprfc.ini). See The SAPRFC.INI File [Page 66] for more details.

  In some cases the receiving SAP system must have at least Release 3.0C to be able to serve every kind of options.

  destination must point to a valid entry in the saprfc.ini. This file can be in the current directory where RFC programs are running or it must be defined by the environment variable RFC_INI (e.g. in /usr/rfctest/saprfc.ini).

  connopt must be set to NULL.

RFC_OPTIONS

The structure

```
typedef struct {
      char * destination;
      RFC_MODE mode;
      void * connopt;
      char * client;
      char * user;
      char * password;
      char * language;
      int trace;
} RFC_OPTIONS;
```

provides parameters for RfcOpen [Page 126].

Depending on the type of connection, various data have to be supplied to open an RFC connection.

There are three ways to supply this information:

1. You can enter a destination name pointed to an entry in a 'saprfc.ini' file which contains the necessary network parameters and RFC-specific parameters for opening the connection at RfcOpen [Page 126].
2. You can enter a destination name pointed to an entry in a 'sideinfo' file which only contains the necessary network parameters for opening the connection at RfcOpen [Page 126]
3. In your program you supply all the data needed for opening the connection at RfcOpen [Page 126].

The first of these methods is recommended (i.e. working with saprfc.ini), because it allows you to use some RFC features today as well as in the future without changing your RFC programs. See sprfc.ini for more details.

Fields that are not supplied must be set to NULL.

The RFC_OPTIONS structure consists of three groups.

- The data needed to establish the physical connection depend on the type of connection (R/2 or R/3 connection, version,...). Depending on the contents of the field 'mode', the field 'connopt' must point to different structures.
- The signon data (client, user, password, language) are needed for authentication. Since RfcOpen [Page 126] only opens the physical connection directly, these data are only checked if the first RFC call is sent.
- The third field contains a trace flag. If not zero, a trace file is written for all the operations corresponding to this connection to ease error analysis.

  □

  This structure must be completely initialized with 0. This will ensure that in the future SAP can add more connection parameters.

This structure is defined in SAPRFC.H.

RFC_OPTIONS

Members:

- destination

  name of destination

  If the connection is not described completely, this name is used as a key for a ‘sideinfo’ where the connection should then be described. You always have to fill in this field.

- mode

  connection mode

  There are two different protocol types for RFC, depending on whether your target system is an R/2 or R/3 System.

  If your target system is an SAP system of Release 3.0C or later, you can use various special options if you enter the value RFC_MODE_VERSION_3 here.

  Depending on the contents of this field, connopt must point to different structures (see RFC_MODE [Page 200]).

- connopt

  If connopt is NULL, the ‘sideinfo’ or the ‘saprfc.ini’ file is used to determine the connection parameters.

  Without ‘sideinfo.ini’ file connopt must point to a structure of type RFC_CONNOPT_R3ONLY [Page 195], RFC_CONNOPT_VERSION_3 [Page 196] or RFC_CONNOPT_CPIC [Page 194] depending in the value of ‘mode’.

- client

  signon data: client

- user

  signon data: user

- password

  signon data: password

- language

  signon data: language

- trace

  trace

  If 0, no trace is written. If not 0, the RFC library traces all activities into a file ‘dev_rfc’ in the actual working directory.

  If your target system is of Release 3.0C or later, you can enter the value ‘D’ here to start the ABAP debugger on the target system.

  The ABAP debugger can also be activated by setting the environment varialbe RFC_DEBUG before the call to RfcOpen [Page 126] is done.

  □

  Options for an R/3 connection.

```
// static = initialized structures
static RFC_OPTIONS        options;
static RFC_CONNOPT_R3ONLY rfc_connopt_r3only;
RFC_HANDLE                handle

options.destination          = "TEST";
options.mode      = RFC_MODE_R3ONLY;
options.client    = "000";
options.user      = "SAP*";
options.language = "E";
options.password = "PASS";
options.trace     = 0;         // turn trace off

options.connopt   = &rfc_connopt_r3only;

rfc_connopt_r3only.hostname            = "some_host";   //
some host name
rfc_connopt_r3only.sysnr= 0;           // system 00

handle = RfcOpen(&options);
if(handle == RFC_HANDLE_NULL)
{
...
```

RFC_PARAMETER

The structure

```
typedef struct {
      void * name;
      unsigned nlen;
      unsigned type;
      unsigned leng;
      void * addr;
} RFC_PARAMETER;
```

provides RFC parameters. It describes ‘Exporting’ and ‘Importing’ parameters of a function module.

This structure is defined in SAPRFC.H.

Members:

- name

  name of the field (in the interface definition of the function)

- nlen

  length of the name (should be strlen(name))

- type

  data type of the field

- leng

  length of the field in bytes

- addr

  address of the field to be exported or imported

See also RFC_CHAR [Page 97], RFC_NUM [Page 101], RFC_BYTE [Page 105], RFC_BCD [Page 106], RFC_INT [Page 102], RFC_INT1 [Page 103], RFC_INT2 [Page 104], RFC_FLOAT [Page 109], RFC_DATE [Page 107], and RFC_TIME [Page 108].

RFC_RC

The structure

```
enum RFC_RC {
      RFC_OK,
      RFC_FAILURE,
      RFC_EXCEPTION,
      RFC_SYS_EXCEPTION,
      RFC_CALL,
      RFC_INTERNAL_COM,
      RFC_CLOSED,
      RFC_RETRY,
      RFC_NO_TID,
      RFC_EXECUTED,
      RFC_SYNCHRONIZE,
      RFC_MEMORY_INSUFFICIENT,
      RFC_VERSION_MISMATCH,
      RFC_NOT_FOUND,
      RFC_CALL_NOT_SUPPORTED
};
```

provides RFC return codes.

This structure is defined in SAPRFC.H.

Members:

- RFC_OK

  ok

- RFC_FAILURE

  error occurred

- RFC_EXCEPTION

  exception raised

- RFC_SYS_EXCEPTION

  system exception raised, connection closed

- RFC_CALL

  call received

- RFC_INTERNAL_COM

  internal communication, repeat (internal use only)

- RFC_CLOSED

  connection closed by the other side

- RFC_RETRY

  no data yet (RfcListen only)

- RFC_NO_TID

no transaction-ID available

- RFC_EXECUTED

  function already executed

- RFC_SYNCHRONIZE

  synchronous call in progress (only for Windows)

- RFC_MEMORY_INSUFFICIENT

  memory insufficient

- RFC_VERSION_MISMATCH

  version mismatch

- RFC_NOT_FOUND

  function not found (internal use only)

- RFC_CALL_NOT_SUPPORTED

  call is not supported on Windows

RFC_TABLE

The structure

```
typedef struct {
   void * name;
   unsigned nlen;
   unsigned type;
   unsigned leng;
   ITAB_H ithandle;
   RFC_ITMODE itmode;
   int newitab;
} RFC_TABLE;
```

provides RFC tables. It describes tables parameters of the interface of a function module.

This structure is defined in SAPRFC.H.

Members:

- name

  name of the table (in the interface of the function)

- nlen

  length of the name (should be strlen(name))

- type

  data type of the lines of the table

- leng

  length of a row in bytes

- ithandle

  table handle ITAB_H [Page 151], i.e. the address of the control structure of the internal table
  This is an input field at RfcCall [Page 128] and an output field at RfcGetData [Page 138].

- itmode

  mode (RFC_ITMODE [Page 199])

  This table has to be received either by ‘call by reference’ or by ‘call by value’ (only for RfcGetData, has no effect in RfcCall).

- newitab

  If the value after RfcGetData is not 0, the table was created by RfcGetData. This is for internal use only. This field must not be modified between RfcGetData [Page 138] and RfcSendData [Page 145].

RFC_ATTRIBUTES

The structure

```
typedef struct {
      char dest[64+1];
      char own_host[100+1];
      char partner_host[100+1];
      char systnr[2+1];
      char sysid[8+1];
      char client[3+1];
      char user[12+1];
      char language[1+1];
      char reserved[256];
 } RFC_ATTRIBUTES;
```

is returned by RfcGetAttributes [Page 137] describing some information about this RFC connection.

This structure is defined in SAPRFC.H.

Members:

- dest[64+1]

  RFC destination

- own_host[100+1]

  Own host name

- partner_host[100+1]

  Partner host name

- systnr[2+1]

  R/3 system number

- sysid[8+1]

  R/3 system name

- client[3+1]

  Client

- user[12+1]

  User

- language[1+1]

  Language

- reserved[256]

  Reserved

Platform-Specific Features of the RFC API

This section deals with specific features of the **RFC API** on different platforms.

OS/2

Under OS/2 a "shared" RFC library (DLL) is available that provides the complete RFC functionality and is currently operable only with the TCP/IP product for OS/2 from IBM.

Windows 3.x

On Windows 3.x, a "shared" RFC Library (DLL) is also provided with the following functionality:

- Any RFC module in SAP R/2 or R/3 can be called from within Windows.
- From ABAP programs in R/3 Systems, RFC server programs can only be called up via SAPGUI. However, this is possible only with SAPGUI version >= 2.1J / 2.2 D.
- All RFC calls in an RFC client or server program must be performed in one task. RFC calls in different Windows tasks are not allowed.

The RFC library can be run with all TCP/IP products for Windows that support sockets.

For more information, refer to SAP Network Supported Products.

Windows NT/95

On Windows NT, a "shared" RFC library (DLL) is also provided. This DLL offers the complete RFC functionality except for one restriction and currently runs only with the TCP/IP product for Windows NT from Microsoft.

The following restrictions apply:

Since no **remote shell demon** currently exists under Windows NT, there are only the following two options to start an RFC server program from within SAP R/2 or R/3:

- Via SAPGUI under Windows NT
- Via an SAP gateway that runs under Windows NT. The SAP gateway then starts the RFC server program locally.

As an alternative, if you want to run an RFC server program on an explicit NT machine, you can use the Registering Server Programs with the SAP Gateway [Page 49].

R/3-Based UNIX Platforms

On R/3-based UNIX platforms a “non-shared” RFC library is available that provides the complete RFC functionality and runs on the standard TCP/IP product installed on these computers.

Sample Programs

Sample programs are provided in the **RFC SDK**:

- **RFC client programs**:
  - sapinfo.c
  - srfctest.c
  - rfc2abap.c
  - startrfc.c
  - and trfctest.c for transactional RFC client programs.
- **RFC server programs**:
  - rfcexec.c
  - srfcserv.c
  - and trfcserv.c for transactional RFC server programs.

You can find these programs in**.../rfcsdk/text**.

The syntax for establishing connections is the same with **R/3 and R/2 Systems**. Therefore, the example programs delivered in the RFC SDK can be used for RFC with **R/3 and R/2**.

Error Handling

This sections contains the following topics:

Debugging

As of R/3 Release 3.0C, you can use the full functionality of the ABAP debugger when developing an application using RFC.

Depending on the structure of the program to debug, you can use different techniques:

- setting RFC_DEBUG in the environment

  You can set the environment variable to any value to enter the debug mode.

- setting `-debug` on the command line

  If the program uses RfcConnArgv [Page 119] for scanning the command line, set `–debug` on the command line to enter debugging mode.

- setting the trace field to D or E

  If you can modify the C code you are using you can set the trace field in the structure RFC_OPTIONS [Page 201] to the value D or E before calling RfcOpen [Page 126] or any other function opening an RFC connection (D: debugging without activated RFC trace; E: debugging with activated RFC trace).

  Another way of doing this is to define an entry in the *saprfc.in*' file which includes all necessary parameters and the RFC-specific parameter RFC_TRACE in the *saprfc.in*' file. An RFC client program can then issue the RfcOpen [Page 126] call with the mode RFC_MODE_PARAMETER and a destination pointed to the defined entry. See The SAPRFC.INI File [Page 66] for more details.

  In R/3 Release 3.0C, this functionality is only supported for Windows NT, Windows 95 and all supported UNIX platforms (Motif). In particular, this functionality is not available for 16-bit Windows environments (Windows 3.x, as well as the 16-bit subsystems of Windows NT and Windows 95).

  Moreover, you have to install a SAPGUI program on your system which must be of Release 3.0C or later.

  Under Windows NT and Windows 95, the SAPGUI program and its DLLs and auxiliary programs can be installed anywhere. However, you must have started 'SAPGUI' once before you can use it via RFC, because the program must register itself in the Windows registry.

  Under R/3-based UNIX, the SAPGUI program must be installed on the normal SAP path **/usr/sap/<system name>/SYS/exe/run**.

Error Handling in ABAP

Causes

There are different causes for ABAP errors, depending on the program. Causes include:

- Incorrect or incomplete entries in the RFC destination (transaction SM59)
- Network problems
- Authorization problems
- Error in the RFC program

Solution

Check the following:

1. Check all the entries made in transaction SM59. Do all the entries match the settings of the target system?
2. Is the hostname correct?
3. Is the service/system number correct?
4. Is the user/password set correctly?
5. Is the target system active?
6. Did you save the destination?
7. Make sure that the target machine can be addressed by the host on which the gateway process is running.
8. Check that the gateway processes are running on both systems.
9. Check the network connection.

Error Handling in RFC Server Programs

Causes

There are different causes for ABAP errors, depending on the program. Causes include:

- Incorrect or incomplete entries in the RFC destination (transaction SM59)
- Network problems
- Authorization problems
- Error in the RFC program

Solution

Check the following:

1. Check all the entries made in transaction SM59. The destination used must be entered with category T. Enter a complete path name, if possible, for the program to be started. Make sure to save the destination.
2. Configuration on operating system level

   a) The host name specification in the destination

      i) A name is entered into ‘host name’ (radio button in Release 3.0 is selected ‘explicitely’) or a non-standard gateway is stored (in gateway options). In the latter case, the program is started by the standard gateway program of the system or by the gateway (**`gwrd`**) specified explicitely via ‘remote shell’.

      Make sure that the entered computer can be addressed by the computer of gateway process. Enter the following command on this computer:

      `/etc/ping <host name>` or **`ping <host name>`**

      To start a program with ‘remote shell’ on another computer, it is necessary that the user-ID of the gateway process exists on the target system and that the **`HOME`** directory of this user contains a file.**`rhosts`** in the target system. The name of the calling computer must be stored in this file. To test this, log in on the computer of the gateway process under the user-ID of this process and call the command:

      ```
      remsh <host name> <program name>
      ```

      In this case, you have to enter the same as in transaction SM59 for **`<host name>`** and **`<program name>`**.

      (If an RFC server program is called without parameters, the call **`RfcAccept`** returns an error code (**`RFC_HANDLE_NULL`**) in any case and the program should terminate immediately.)

      ii) No entry is made in ‘*host name*’ (or in 3.0: *Server* is selected)

      In this case, the program is started from the SAP application server.

      Make sure that the program can be addressed from the SAP application server.

Make sure that the SAP application server has the authorization for starting the program.

To do this, log on under the user-ID of the SAP application server (generally `c11adm`). Go into the ‘work’ directory of the application server (`/usr/sap/.../D.../work`), if possible, and start the RFC server program manually from there.

(If an RFC server program is called without parameters, the call `RfcAccept` returns an error code (`RFC_HANDLE_NULL`) in any case and the program should terminate immediately.)

In Release 3.0, you can also select ‘User’. In this case, the program is started by ‘Sapgui’, that is, under the user-ID and the environment of the user.

Make sure that the program can be addressed by `sapgui/saptemu`.

Make sure that `sapgui` has the authorization to start the program.

To do this, call the RFC server program.

b) Problems in the RFC server program itself

To pick up `stderr` output of the RFC server program, you can enter a control program instead of the actual server program into the destination which in turn starts the actual server program with the same command line and which in this case redirects the standard output of the program into a file.

□

RFC server program is /xxx/xxxx.

However, the C script is called (do not forget the specification of the shell in the first line):

```
#!/bin/csh

date >> /tmp/rfclog

/xxx/xxxx $* >>& /tmp/rfclog

echo $status >>& /tmp/rfclog
```

Display the log file `/tmp/rfclog` for further error analysis.

You can activate the trace flag in the destination (remember to save). A file `dev_rfc` is then written by the RFC server program in its current directory containing all data received, operations and errors that occurred.

Debugging an RFC server program

Using Registering Server Programs with the SAP Gateway [Page 49], you can start your RFC server program with a C debugger, such as dbx, xdb or code view, etc.

If your RFC server program will be started by an SAP component (i.e. SAPGUI, SAP Gateway, Application Server), you can do as follows:

1. Enter a program for your destination in SM59 which simply writes the transferred command line to a defined file:

   `Example (C-Shell)`

```
#!/bin/csh

echo $* > /tmp/rfc
```

2. Call the C-debugger with your RFC server program. For example, **`xdb <program name>`**
3. Set a breakpoint after the **`RfcAccept`** call.
4. Start the calling ABAP program. The control program writes its command line to its output file (**`/tmp/rfc`** in this example).
5. Read this file and start the RFC server program loaded into the debugger immediately using the same command line.

More Information in Error Cases

This information refers to possible errors in R/3 Release 3.0D and 3.0E.

Trace Files

For better error analysis, the RFC library and the R/3 System Release 3.0E onwards will write some more error information into the TRACE file:

- dev_rfc0, dev_rfc1,... within an R/3 System or
- dev_rfc or RFCnnnnn.TRC (16-bit RFC library) in non-SAP systems

RFC Library 3.0C on Windows (16- and 32-bit)

When working with the **saprfc.ini** file on Windows (librfc16.dll and librfc32.dll), you cannot connect to an R/3 System Release 3.0D onwards with dialog user, because of a security error (system log with error code 152!).

In this case, use the RFC library version 3.0D onwards or CPIC user.

Stack Overflow while Working with librfc16.dll in R/3 3.0D onwards

Because of some extensions in the “network interface” (NI) layer, some RFC programs could be broken by Windows Operating System with the error STACK OVERFLOW in some RFC error cases while working with the 16-bit RFC library 3.0D onwards.

In this case, recompile your RFC program with at least 9000 bytes stack size. The default value in “cl.def” of the Microsoft Visual C++ 1.5x is 8096 bytes.

Getting Version of the RFC Library

Until now it has been possible to get the version of the **RFC library** and all SAP internal libraries, such as cpic, ni,..., by using the call **RfcGetAllLibVersions**, but this call does not return any information about the R/3 Release to which this RFC library belongs.

From R/3 Release 3.0E onwards, the RFC library will also return the release of the R/3 System.

To get this information, you can start `sapinfo -v` from the command line.

RFC and SAProuter

This section contains the following topics:

Introduction to SAProuter

SAProuter is an SAP software product which is available on all R/3-based UNIX platforms and Windows NT/95. It acts like a firewall system by regulating access from/to your network.

SAProuter can be used

- to establish an indirect connection between two programs running on different machines. The network configuration does not allow a direct communication between these machines due to missing IP addresses (or the same IP addresses as well) or firewall restrictions.
- to improve network security by allowing accesses from/to your network with or without password-protection only from a specified machine where the SAProuter is running.
- to control and log all connections between your network and the rest of the world.

Important SAProuter Commands

saprouter	Online help (display list of all supported options)
saprouter -r	Start SAProuter with default values
saprouter -s	Stop SAPRouter

Route String

A route string can have one or more substrings. Each substring contains parameters how to reach the next SAProuter or the target host or program on the target host.

Such parameters are:

- name or IP-address of the target host
- service (port number) of the program running on the target host
- password for this connection, if needed

Example of one substring: **/H/host/S/service/P/password**

H:	Identifier for host name
S:	Identifier for service (port number)
P:	Identifier for password

Route Permission Table

The SAProuter regulates access to your network via the route permission table in form of a file. You can start your SAProuter with this file name.

An entry in a route permission table has the following structure:

```
<P/D> <source host> <target host> <target service> <password>
```

P(ermit):	allows connection
D(eny):	prevents connection
<source host>:	host name or IP-address, could be a SAProuter
<target host>:	host name or IP-address, could be a SAProuter
<target service>:	service (port number) of the program of the target host The default service of SAProuter is ‘3299’.

□

If no route permission table was explicitly assigned to the SAProuter while starting (option -R <name of a route permission table>), the file ‘saprouttab’ in the current directory will be used. If this file is not available, all connections are allowed.

You can use wildcarts (‘*’) to define hosts, services and passwords in your route permission table.

□

See **SAP note 30289** for more details about SAProuter.

A Typical Use of SAProuter (remote support)

Network_1 (SAP-Walldorf)	**Network_2 (Customer)**		
host_11	**host_r1**	**host_r2**	**host_21**
SAPGUI ————>	**SAProuter ———— >** (“3299”)	**SAProuter ——— —>** (“3299”)	**SAP R/3**

Route Permission Tables

Entry in the route permission table of **SAProuter on host_r1 in Network_1**:
P host_11 host_r2 3299

Entry in the route permission table of **SAProuter on host_r2 in Network_2**:
P host_r1 host_21 sapdp<R/3 system number>

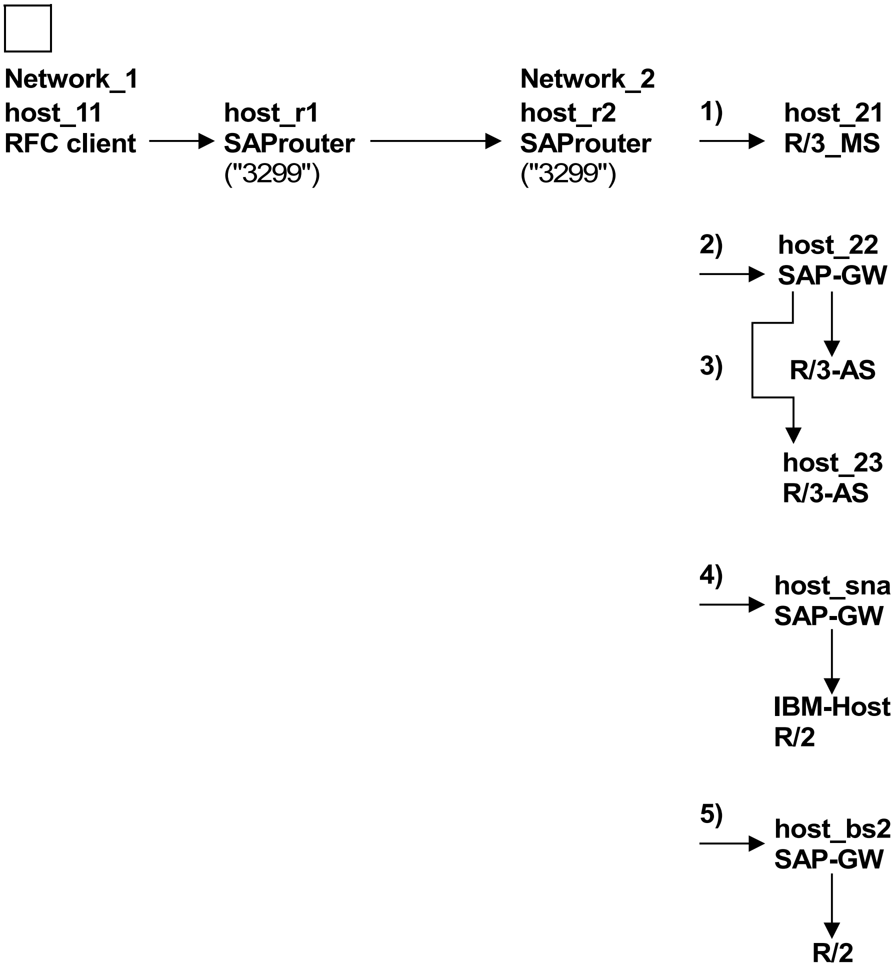
The SAPGUI on host_11 will connect to the R/3 application server on host_21 with the following route string, host name and dispatcher service:
/H/host_r1/H/host_r2/H/host_21/S/sapdp<R/3 system number>

The information about services of both SAProuters are not necessary, because they are running with the default service (“3299”).

RFC Client Program and SAProuter

Any RFC client program can connect to an SAP System via SAProuter. The feature SAP using SAPGUI [Page 44] which is available from R/3 Release 3.0C onwards is also possible via SAProuter. You must also allow the corresponding SAP dispatcher to work with SAProuters.

One or more SAProuters can be used. The following example shows how the client program can work with two SAProuters.



Route Permission Tables

In the route permission table of **SAProuter on host_r1 in Network_1** only one entry is necessary:

P host_11 host_r2 3299

The entries in the route permission table of **SAProuter on host_r2 in Network_2** are dependent on the type of RFC connection which is established with the SAP System. The RFC client program must inform the RFC library about all used SAProuters for connecting to the R/3 System via the parameter ‘host name’ in **RfcOpen**.

RFC Client Program and SAProuter

1. Using Load Balancing

The following applies only to R/3 Release 3.0 onwards.

The host name of the message server must contain the route string. For the example described in the last section, the RFC client must set the host name of the message server as follows:
MS-Host: **/H/host_r1/H/host_r2/H/host_21.**

Entries in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_21	sapms<R/3 name>
P	host_r1	<R/3-AS host 1>	sapgw<R/3 system number>
...			
P	host_r1	<R/3-AS host n>	sapgw<R/3 system number>

2. Specified R/3 Application Server and Default SAP Gateway

The host name of the specified application server must contain the route string. For the example described in the last section, the RFC client must set the host name of the application server as follows:
AS-Host: **/H/host_r1/H/host_r2/H/host_22.**

Entries in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_22	sapgw<R/3 system number>

3. Specified R/3 Application Server and Specified SAP Gateway (R/3)

If you are working with a specified SAP gateway (not recommended for R/3 Release >= 2.1I, 2.2C or 3.0 because of bad performance), the host name of the SAP gateway must contain the route string. The host name of the application server may not contain the route string. For the example described in the last section, the RFC client must set the host name of the SAP gateway as follows:
GW-Host: **/H/host_r1/H/host_r2/H/host_22**
AS-Host: **host_23**

Entries in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_22	sapgw<GW service number>

4. R/2 in an IBM Environment

The host name of the SAP gateway must contain the route string. For the example described in the last section, the RFC client must set the host name of the SAP gateway as follows:
GW-Host: **/H/host_r1/H/host_r2/H/host_sna**
host-sna is a machine where a SNA LU6.2-product is running.

Entries in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_sna	sapgw<GW service number>

5. R/2 in an SNI Environment (BS2000)

The host name of the SAP gateway must contain the route string. For the example described in the last section, the RFC client must set the host name of the SAP gateway as follows:
GW-Host: **/H/host_r1/H/host_r2/H/host_bs2**
host-bs2 is a BS2000 host where the SAP R/2 is running.

Entries in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_bs2	sapgw<GW service number>

Starting an RFC Server Program Via SAProuter

An RFC server program can be started by a SAP gateway, by the currently running R/3 application server or by the currently running SAPGUI. Using the registering feature of an R/3 System Release 3.0C onwards, the RFC server program can be started before and then it registers at an SAP gateway and then waits for RFC requests.

□

From R/3 (RFC library and SAPGUI) Release 3.0D onwards, it is possible to start an RFC server program by a SAPGUI on a workstation (which is connected to an R/3 System) via one or more SAProuters.

According to the way of starting an RFC server program and to the Release of the SAP System, there are different ways for RFC server programs to work with SAProuters.

Using the Registering Feature [Page 229]

Program Start by Application Server [Page 230]

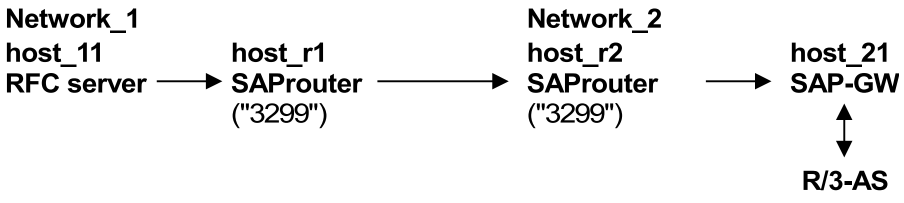
Program Start by SAP Gateway [Page 231]

Program Start by SAPGUI [Page 233]

Using the Registering Feature

As described in Registering Feature [Page 49], the registering feature is only available with an R/3 System and SAP gateway of Release 3.0C onwards.

One or more SAProuters can be used. The following example shows how a server program can work with two SAProuters using the registering feature:



Route Permission Tables

Entry in the route permission table of **SAProuter on host_r1 in Network_1:**

P	host_11	host_r2	3299

Entry in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_21	sapgw<GW service number>

The host name of the SAP gateway must contain the route string. For the example above, the RFC server must set the host name of the SAP gateway as follows:
GW-Name: /H/host_r1/H/host_r2/H/host_21

The delivered RFC server program srfctest.c can be started at the command line as follows:
srfcserv -ahost_11.srfcserv -g/H/host_r1/H/host_r2/H/host_21 -xsapgw<GW serv.-nr.>

A destination in transaction sm59 can be defined as follows:

Connection type:	**T**
Activate type:	**Registering**
Program-ID:	**host_11.srfcserv**
Gateway host:	**host_21**
Gateway service:	**sapgw<GW service number>**

Program Start by Application Server

Program start by application server means that the RFC server program will be run on the same machine as the application server. In this case, you have different application servers of an R/3 System running on different networks connected via SAProuter.

From R/3 (RFC library and SAPGUI) Release 3.0D onwards, it is posssible to start an RFC server program by a SAPGUI on a workstation which is connected to an R/3 application server.
This application server has two networks:

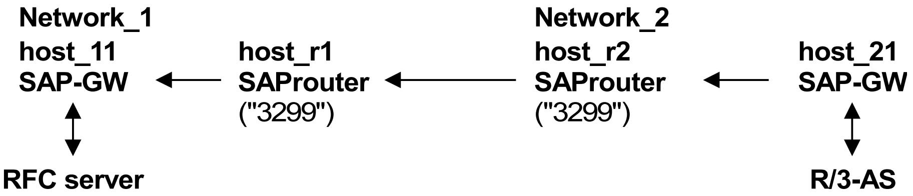
- one for the network with all R/3 application servers and
- one for the network with this application server and the frontend workstation with SAPGUI.

Program Start by SAP Gateway

Because a SAP gateway cannot start an RFC server program with remote shell on another machine via SAProuter, it is necessary to install a SAP gateway on a machine in the network where the RFC server program will be run. It should be the same machine for a better performance.

One or more SAProuters can be used. The following example shows

- two different networks with two SAProuters and
- how an RFC server program can be started by an SAP gateway and how it communicates with an R/3 System running on another network.



Route Permission Tables

Entry in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_21	host_r1	3299

Entry in the route permission table of **SAProuter on host_r1 in Network_1:**

P	host_r2	host_11	sapgw<GW service number>

A destination in transaction sm59 can be defined as follows:

Connection type:	**T**
Activate type:	**Start**
Program location:	**explicit host**
Program:	**/rfctest/srfcserv**
Target host:	**host_11**
Gateway host:	**/H/host_r2/H/host_r1/H/host_11**
Gateway service:	**sapgw<GW service number>**

Program Start by SAP Gateway

□

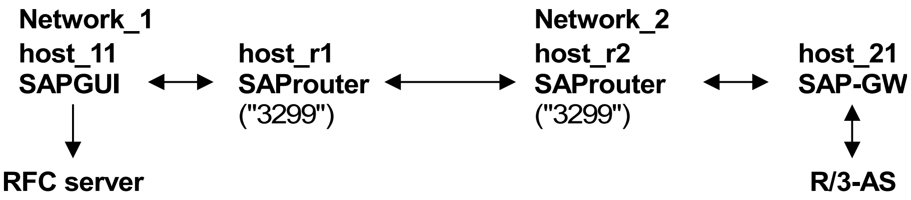
The maximum length of the gateway host (transaction sm59) is:

- 32 bytes for R/3 Release <= 3.0C
- 64 bytes for R/3 Release 3.0D
- 100 bytes for R/3 Release >= 3.0E.

Program Start by SAPGUI

One or more SAProuters can be used. The following example shows

- two different networks with two SAProuters and
- how an RFC server program can be started by the currently running SAPGUI and how it communicates with an R/3 System running on another network.



Route Permission Tables

Entry in the route permission table of **SAProuter on host_r1 in Network_1:**

P	host_11	host_r2	3299

Entry in the route permission table of **SAProuter on host_r2 in Network_2:**
– Entry for RFC server program

P	host_r1	host_21	sapdp<R/3 service number>

– Entry for SAPGUI

P	host_r1	host_21	sapgw<R/3 service number>

A destination in transaction sm59 can be defined as follows:

Connection type:	**T**
Activate type:	**Start**
Program location:	**Front-end workstation**
Program:	**/rfctest/srfcserv**

The maximum length of the gateway host (transaction sm59) is:

- 32 bytes for R/3 Release <= 3.0C
- 64 bytes for R/3 Release 3.0D

- 100 bytes for R/3 Release >= 3.0E.

The maximum length of the host name of the application server (transaction sm59) is:

- 64 bytes for R/3 Release <= 3.0D
- 100 bytes for R/3 Release >= 3.0E.

RFC server program and SAPGUI use the same host name of an application server

An application server runs with two network adapters, one for the network with R/3 application servers and one for the network with this application server and the frontend workstation with SAPGUI. This application server has two host names in two different networks.

Problem

Starting an RFC server program via SAPGUI is not an available option, because after having been startet, the RFC server program will connect to the application server with the host name only known in the R/3 network.

Solution

The RFC server program will use the same host name as the SAPGUI does. This functionality is available with R/3 Release 3.0D onwards and SAPGUI 3.0D onwards.

□

For more information about problems and error handling, see Error Handling [Page 215].

RFC Between External Programs

With the **RFC API** SAP provides an interface for communications with external systems using the same internal function calls in SAP Systems. For a local test (outside SAP Systems), you can write both RFC client and server programs as external programs and you can let these programs communicate with each other via an **SAP gateway**.

In this case, an RFC server program can only be started by an SAP gateway, or it can use the registering feature described in Registering Feature [Page 49].

An RFC client program can work either with a local *sideinfo* or with the *saprfc.ini* file.

The sample programs **srfctest.c** and **srfcserv.c** which are delivered in the RFC SDK can be used for this local test if you have an SAP gateway running in your test environment.

The following sections describe how the parameters in these files should be set:

Using A Local sideinfo File [Page 236]

Using the saprfc.ini File [Page 237]

Using A Local sideinfo File

Sample entry in the *sideinfo* file:

```
DEST = RFCEXT
GWHOST = <host name of the SAP gateway, e.g. hs0311>
GWSERV = <service name of the SAP gateway, e.g. sapgw53>
PROTOCOL = <C/E (or R): server will be started by (or is already
registered at) SAP-GW>
LU = <host name of the RFC server program, e.g. hs0311>
TP = <name of the RFC server program, e.g. /rfctest/srfcserv>
```

Moreover, the parameter *mode* in **RfcOpen** must be set to RFC_MODE_CPIC.

With the entry above, no entry is necessary in the *sideinfo* file for the SAP gateway.

Using the saprfc.ini File

This functionality is only available with the RFC library of R/3 Release 3.0D onwards. The SAP gateway can be from Release 2.1I, 2.2C or 3.0 onwards. Working with type E is only possible with Release 3.0D onwards.

Sample entry in the *saprfc.ini* file:

```
DEST = RFCEXT
TYPE = <E (or R): server will be started by (or is already registered
at) SAP-GW>
GWHOST = <host name of the SAP gateway, e.g. hs0311>
GWSERV = <service name of the SAP gateway, e.g. sapgw53>
TPHOST = <host name of the RFC server program, e.g. hs0311>
TPNAME = <name of the RFC server program, e.g. /rfctest/srfcserv>
```

The parameter *mode* in **RfcOpen** must be set to RFC_MODE_PARAMETER as usual.