Explore the technical procedures and settings for creating and securing RFMs in ABAP

Part 2 — A comprehensive guide for SAP programmers and administrators

by Masoud Aghadavoodi Jolfaei and Eduard Neuwirt

Masoud Aghadavoodi Jolfaei
Development Architect,
SAP AG

Eduard Neuwirt
Senior Developer,
SAP AG

(Full bios appear on page 66.)

This is the second installment in a two-article series on the remote function call (RFC) protocol, which is the de facto protocol of choice for SAP developers, system architects, and administrators who need to connect their SAP systems.

In the first installment,[1] we introduced key RFC terminology and concepts, including discussions about the five types of RFCs, and we stepped through the RFC communication process.[2] The goal of Part 1 was to provide a solid foundation to help you understand RFC so you could use this knowledge to create better performing and more robust ABAP applications.

Continuing to strive for and reach beyond that goal, in this article we explore how to design and develop remote function modules (RFMs) that fulfill performance and security expectations and maintain RFC destinations that are well prepared for effective network communication while avoiding the misuse of them. We explain the technical procedures and settings necessary to create and secure remote-enabled function modules, including how to define RFC destinations. We also demystify how RFC requests are serialized/deserialized for transmission between systems, including the steps involved in the execution of an RFM and how to handle errors and messages returned by the RFC runtime.

[1] See "Master the five remote function call (RFC) types in ABAP: Part 1 — A comprehensive guide for SAP programmers and administrators" in the September/October 2006 issue of *SAP Professional Journal*.

[2] Recall that the process begins with an RFC call from your ABAP program running within a work process on the SAP server, and proceeds through a dispatcher and gateway. These components coordinate delivery of the request over the network to the target system, as well as delivery of the response to your program. Analogous processes on the remote system coordinate assignments to a work process to make the RFC calls, and both systems contain a roll-in/roll-out mechanism by which work processes are assigned to programs that are free to work on other tasks until further processing is required.

Note!

This article is written for developers and administrators — those who are beginners as well as those who are experts. If you're already familiar with one or more of the topics in this article, feel free to skim those sections for embedded tips that might supplement your knowledge.

Note!

The first part of this series introduced the different kinds of RFCs. The business of these five RFC types is to transmit ABAP data between SAP systems. In this second article, we explain the principles of how the RFC runtime handles security and data transmission. These principles are valid for all five kinds of RFCs.

Before you build custom RFMs

Before you build function modules that can be called remotely via RFC (i.e., RFMs), you need to perform a couple of technical prerequisites and be aware of some performance pitfalls. Let's begin with the technical prerequisites.

Perform these two steps before building an RFM:

1. Enable the remote functionality by selecting the Remote-Enabled Module option on the Attributes tab for the function module (in this case, RFC_SYSTEM_INFO), as shown in **Figure 1**.
2. Implement the logic of this function module in accordance with standard RFC guidelines.[3] As you do with any other function module, use transaction SE37 to develop and maintain your RFMs.

In addition, you should adhere to the following guidelines to avoid performance issues:

- **Avoid marking parameters "by reference."** You can only remote-enable a function module if none of the function's IMPORTING, EXPORTING, or CHANGING parameters are marked as "by reference." TABLES parameters are an exception;[4] they are always passed by reference.
- **Avoid using parameters that are not defined in the Data Dictionary (DDIC).** You can only remote-enable a function module if all parameters are declared as DDIC types or as non-generic built-in ABAP types.[5]
- **Avoid using deep parameters whenever possible.** Deep parameters are variables, structures, or tables that leverage variable length data types (e.g., STRING or XSTRING). Transport of deep parameters requires significantly more memory than flat parameters. It can degrade performance, because the RFC engine has to use a more elaborate XML-based mechanism called XRFC[6] serialization to serialize and to transport these deep parameters. In contrast, using flat parameters — that is, those that contain only data types of a fixed, pre-defined length such as I (integer), F (float), or T (time) — results in better performance and scalability

[3] To find this information, go to the SAP Help Portal at http://help.sap.com and search for RFC in SAP NetWeaver 2004 or SAP R/3 documentation.

[4] RFC does not support reference parameters, with the exception of TABLES parameters. For more information, go to the Parameter Handling in Remote Calls topic in the SAP NetWeaver Library at http://help.sap.com.

[5] Dictionary structures and ABAP types are not acceptable because an RFC needs the serialization/deserialization of the meta information of the associated parameters from both the client and server sides, which neither dictionary structures nor ABAP types provide.

[6] In the next major release following SAP NetWeaver 2004s, the RFC runtime provides a new serialization/deserialization method, called basXML (i.e., binary ABAP serialization XML), as an alternative to XRFC. basXML is an extension of asXML, which will avoid the performance degradation issues raised by XRFC. For more information, see the article "Mastering the asXML Format to Leverage ABAP-XML Serialization" (*SAP Professional Journal*, November/December 2002).

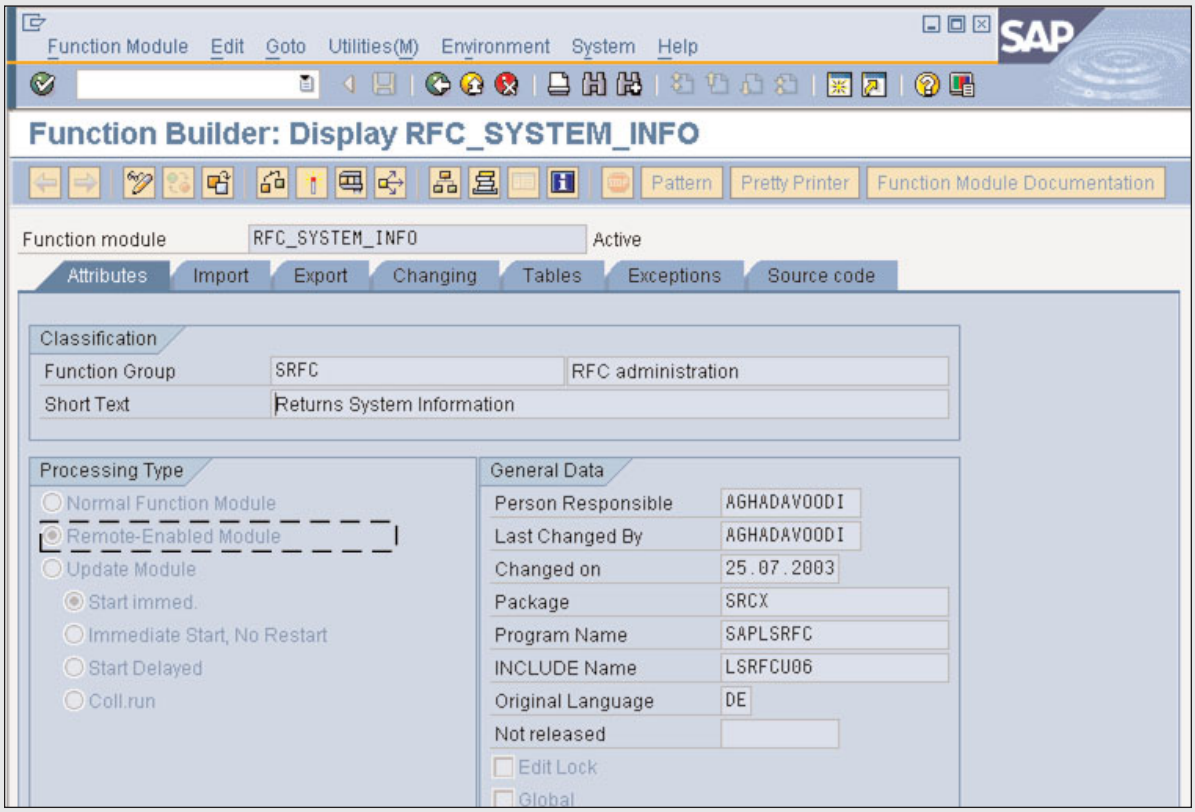


Figure 1 Enable the remote functionality for the RFC_SYSTEM_INFO function module

because you can use a much simpler RFC transport mechanism.[7]

- **Avoid using tables as EXPORTING, IMPORTING, or CHANGING parameters whenever possible.** Using tables as any of these parameters leads to performance degradation, because the RFC engine treats tables as deep parameters. The RFC engine uses the less-efficient XRFC protocol to transport those parameters between the source and target systems.

- **Boost security by organizing your function modules judiciously.** Recall that in our first article we noted that, for system security, the RFC server runtime checks the authority object S_RFC before executing an RFC request. This is to make sure that the user under which the session is running is authorized to execute the requested function module. Recall also that this check (and therefore the user authorization) is at the function *group* level, not the function module level.[8] In other words, you grant or deny remote

[7] A key reason for the performance difference is that the default RFC protocol uses an offset-based transport mechanism. The XRFC protocol uses a name-oriented mechanism analogous to the difference between moving data between two ABAP structures using an equal sign (=) vs. `MOVE-CORRESPONDING`. In other words, the system must locate and access each element within the subcomponents by name, versus simply by its numerical position within the data, which requires much more processing.

[8] In the next major release following SAP NetWeaver 2004s, the RFC runtime extends the authority checks to support the function module level.

Note!

RFC uses a delta managing mechanism for handling table parameters. Through RFC delta management, the RFC runtime exchanges only incremental changes to the tables. In this case only the modified portion of the table rows are transferred, which results in improved performance. This feature is always active for the TABLES parameters, and only for those. If your table is specified in the IMPORTING, EXPORTING, or CHANGING section, the RFC delta management is not active. In the case of CHANGING parameters, the entire table is returned to the RFC client, not just the changed rows in the RFM. For example, your table contains 10,000 rows, and you change only one row in the RFM. By declaring this table in the TABLES section, only the row you changed will be sent back to the client system. On the other hand, if you declare the same table in the CHANGING section, then all 10,000 rows will be sent, negatively affecting performance.

authorization access as a group: Users can either call all RFMs in the group or none at all. It's essential that you assign your function modules to groups strategically to leverage this built-in layer of security.

Note!

In addition to boosting security, you should hard code object-level authorization checks within your function module code (e.g., to particular customer records, plant facilities, or company codes) to further secure your data.

Note!

This RFC authority check is enabled in SAP Basis 4.0B and higher systems. You can activate or deactivate it by modifying the Basis profile parameter auth/rfc_authority_check via transaction RZ11. See this parameter's documentation in RZ11 for more information.

- **Avoid** `LEAVE TO <transaction>` **and other elimination or replacement ABAP transactions.** There is one important restriction regarding the use of certain ABAP statements in remote function modules: Avoid using the following ABAP commands that cause the elimination or replacement of the original internal mode (session) in which the function module is being executed:
  - `LEAVE TO <transaction code>` — Use the ABAP statement `CALL TRANSACTION <transaction code>` instead.
  - `SUBMIT <report>` without the `AND RETURN` statement — Use the ABAP statement `SUBMIT <report> AND RETURN` instead.

  If, in an RFC session, dialog communication is in place and the command field is enabled, then the function code `/n<command code>` leads to the same error behavior — i.e., the RFC communication is interrupted and the RFC client receives an RFC exception.

  These elimination or replacement ABAP commands disrupt the communication pipe between the RFC client and server, because the session in which the function module code is running is terminated. The RFC client system reports this error to your program by raising the exception SYSTEM_FAILURE.
- **Use fixed length character types instead of strings wherever possible.** Strings are data types with a variable length, so the RFC runtime cannot determine the string's length on the receiver side from only the metadata. Many data types have a constant length, so it is possible to reserve enough

memory to store the received data. We recommend that you not use strings when transporting structured data.

- **Use extreme caution when modifying structures that serve as parameters of remote function modules.** Modifying structures that are used either directly or indirectly[9] as parameters can often result in errors that are often very hard to detect. This includes adding, resizing, removing, or re-typing fields. Errors can occur because the code to call a function module might have been written based on an earlier version of the interface. The RFC runtime may not be able to detect the incompatibility.

> ***Note!***
>
> Beware: The application of support packages or portions of upgrades can modify structures, resulting in potential data corruption described below. Regression testing is essential, and you know where to begin troubleshooting if your data appears corrupted.

In simple cases, such as adding a new obligatory import parameter to the function, the RFC call errors out, alerting you to the incompatibility. But consider what would happen if you added a field to the middle structure of an import field. Worse yet: What would happen if you added a field to a structure that serves as the data type of a field nested within a table parameter's structure? In both cases, the RFC runtime does not raise an exception since it is not designed to inspect the internal structure of complex data types such as structures or tables for compatibility (it only does this for simple data types such as integers, strings, or dates). And you may or may not like the answer.

[9] A structure might include a field whose data type is itself a structure. Similarly, a TABLES parameter might include a table whose field is either a structure or a table. We defined this as "deep data" earlier in the article.

In the first case (a parameter with a structured data type), the RFC runtime passes the structure internals by name (i.e., each value is labeled with its field name). If the calling system doesn't pass a value for the new field, the remote system uses the default value for that field's data type (e.g., 0 for an integer field).[10] In the latter case (a structure embedded within another structure), however, data is passed as a raw stream (i.e., no field names are sent along with each data field). The remote system is responsible for dividing this raw stream by position instead of by name. Therefore, the addition of any new field in the center of the structure results in improper data mapping. For example, if the source system sends the fields

```
firstname = "Robert    ",
lastname="Thompson  "
```

but we adjust the target structure to be

```
  firstname(length 10), nickname(length
5), lastname(length 10),
```

then the target system maps this as

```
  firstname="Robert    ",
nickname="Thomp", lastname="son         "
```

The lesson, to be safe, is never remove, reorder, or modify the data types of fields in a structure that an existing RFM uses, unless you review all of the code that calls the function to ensure its compatibility. If you must add fields, add them to the *end* of the structure, not to the middle.

In the next section, we discuss how to create RFC destinations and register them in your SAP system. We also step through some of the more advanced options that even experienced developers may not know about.

Setting up RFC destinations

Destinations are symbolic names that represent a target system — be it an SAP system, non-SAP system, or even another client on the SAP system

[10] If you remove a field from the structure instead, the calling program would mostly still work unless it needed to use the value of this field.

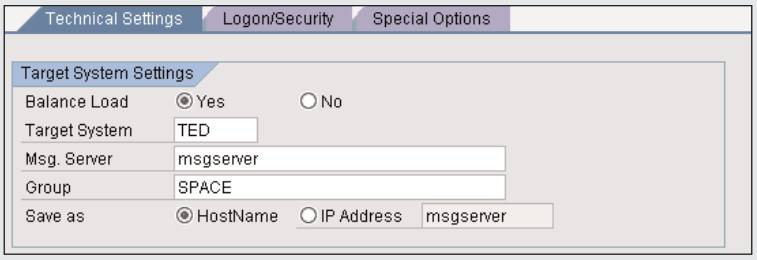


Figure 2 Technical settings for a load-balanced SAP R/3 connection

from which the RFC is made. When programming your RFC calls, you'll use these destinations to identify to which system the RFC should be passed.[11] A key benefit of using destination names versus hard-coding system names or IP addresses is maintainability — you don't need to change your program's code when you want to point the call to a different system or when the system's IP address changes, for example.

> ***Note!***
>
> Since our article so far focuses on making RFC calls between SAP systems, we'll define RFC destinations of type 3 (i.e., R/3 connections and ABAP connections since SAP NetWeaver 2004s). We captured the screenshots for maintaining destinations from an SAP Web Application Server (Web AS) 6.40 system.

You create and maintain destinations using transaction SM59. As you will see shortly, the screen for creating a new destination contains the following tabs:

- **Technical Settings** for entering the essential data about the physical connection to the target system
- **Logon/Security** for entering the necessary logon information to the target SAP system
- **Special Options** for specifying the data needed for error analysis and code page[12] handling

Because some of the fields and settings on these tabs are not particularly intuitive, let's step through the most critical settings and key choices on each tab.

Technical Settings

The Technical Settings tab provides the options and settings you need to define the physical connection to the target system. The first setting on this tab is the Balance Load option. As you can see in **Figure 2**, you have two choices (Yes or No): You can choose to create a *load-balanced* connection by selecting Yes, or create a *direct* connection by selecting No.

When you choose to create a load-balanced connection, you enter the target system (in the example, TED) and the message server (msgserver). You also need to specify a logon group, which determines how client requests to certain applications are load balanced. The RFC infrastructure offers two groups for load balancing:

- **SAPGUI logon group:** Choose to create a load-balanced connection based on this type of group

[11] Refer to Figures 4, 5, 6, 7, and 8 in the September/October 2006 article (on pages 88-95) for the defined destinations for the five RFC types.

[12] A code page is an electronic "menu" of characters that defines the particular mapping between a code and its actual character to enable communication with other systems. We cover code pages and encoding schemes later in the article.

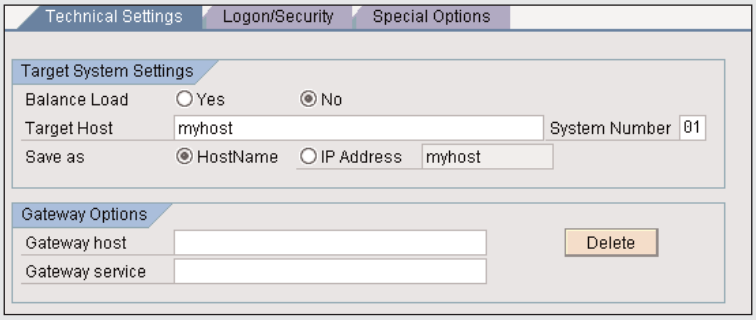


Figure 3 Technical settings of a direct connection

when you want to increase the performance and provide the high availability option. Do this even when certain application servers are not available (e.g., in the example the SAPGUI logon group name is SPACE). You maintain SAPGUI logon groups using transaction SMLG. For more information see the "Maintaining SAPGUI logon groups" sidebar on page 52.

Note!

The SAPGUI group SPACE is a predefined group that addresses all available application servers in a system. We recommend that you define a dedicated group name for each business scenario to achieve better performance and to control the assigned application server.

- **RFC logon group:** Choose to create a load-balanced connection based on this group when you want to increase the performance and provide the high availability option, even when certain application servers are not available. However, if you do, you can use the group only in combination with the ABAP statement for parallel RFC (pRFC). You maintain RFC logon groups using transaction RZ12.

Caution!

The RFC load balancing variants based on SAPGUI logon groups are well suited to low-volume, randomly distributed RFC calls, but not for handling large numbers of RFC requests. One reason is that the message server refreshes the load distribution information for the logon group's participating servers about every five minutes. You can control the frequency through profile parameter rdisp/autoabaptime (the default value is 300 seconds). Therefore, the load is not actually distributed for an application during massive parallel processing. You should not use an RFC load-balanced connection based on SAPGUI logon groups for applications; instead you should use a parallel RFC (pRFC) type if possible.

If you want a *direct* connection, select No, or in other words, you choose *not* to create a load-balanced connection (see **Figure 3**). With a direct connection, you specify the target host (myhost) and its system number (01).[13] Then you save the connection as the host name or as an IP address.

[13] In the SAP Basis arena, a system number is a symbol that identifies the TCP/IP port on which the remote system is listening for requests. In the case of RFC communication, this is the port of the target's server gateway process.

Tip!

Don't be confused — you have to specify the "target system" for the load-balanced connection and the "target host" for a direct connection using transaction SM59. Your SAP system may be running on several computers (i.e., there may be several hosts). In the case of load balancing, you can let the system choose the application server for you. Obviously, you have to address the system during the establishment of the connection. However, if you log on directly to a concrete application server, you have to address the application server that is running on the host. Therefore, you have to identify the host in this case as well. The default name of an application server uses the following syntax: <host_name>_<system_id>_<system_number> (e.g., prdmain_PRD_00, in which prdmain is the host name of the application server).

The next tab provides the options for specifying how your users log on to the specified destination via the RFC connection you just established.

Logon/Security

SAP supports two types of logon options: the traditional user ID/password logon option and a trusted system option. The user ID/password option is the most common and most familiar choice. Generally you'll choose the traditional approach when you want to communicate with another client or SAP system ID (SID), or when you want to use a different user ID to log on.

You'll choose the trusted system logon approach in the following situations and for an increase of security, since with this approach no passwords are exchanged:

- **Single sign-on (SSO) environment:** In an SSO situation, you want all users to have access rights within the target system. You should not specify users in the related R/3 connections option, which tags the trusted option as a logon procedure. Furthermore, a client number transition is also possible (i.e., an SSO from system BIN, client 000 to system B20, client 000 or client 800).
- **Prevention of the misuse of an RFC destination:** If a certain RFC user account is maintained in an RFC destination, we recommend that the use of the user account and destination be restricted to a certain user group in the caller system. With the help of the trusted authority (i.e., S_RFCACL in the target system), you can list all of the users from the caller system who are permitted to use the RFC user account to log on to the system.

Note!

For complete instructions on how to configure and use RFC trust relationships, go to the SAP Help Portal at http://help.sap.com and search for "trusted relationships between SAP systems," or in transaction SM59, follow the menu path Tools → System Administration → Administration → Network → RFC Destinations.

To enable the traditional user ID/password logon approach, you must choose *not* to select the trusted system option, as shown in **Figure 4**. The language, client, user, and password fields are optional for both logon approaches. If you leave the language, client, or user field blank, the system automatically populates the field with appropriate values from the RFC client session. For example, a blank language field would be populated with the value of ABAP runtime (system) field SY-LANGU, a blank client field would be populated with the value of SY-MANDT, and a blank user field would be populated with the value of SY-UNAME.

Let's take a closer look at some of the other options in the top portion of the screen:

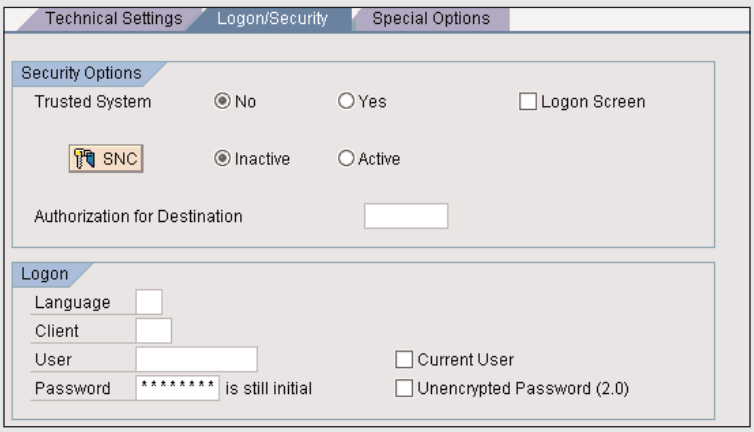


Figure 4 Specify the security settings for a traditional user ID/password logon

- **Logon Screen:** If you select this option, a foreground SAPGUI session makes the RFC call to the destination. The remote SAP system's logon screen appears, giving the user the opportunity to override the default settings.

  If authentication to an SAP system fails (e.g., if the specified user ID or password is no longer valid), and the RFC runtime in the target system determines that the caller session is a SAPGUI session, the system displays the RFC logon screen so the user can correct the information. The system does not update the information in the destination record, however.

- **SNC:** If you choose to activate this option (inactive being the default), you can enhance the security on the communication pipe. Secure Network Communication (SNC) is a software layer in the SAP system architecture that provides an interface to an external security product. SNC secures the data communication paths among the various SAP system components (for more information, go to the SAP Help Portal at http://help.sap.com and search for "SNC").

- **Authorization for Destination:** This is another way to secure your RFC destination from misuse — for example, to avoid the misuse of a defined destination by unauthorized users in other scenarios, or by other user groups. Selecting this option also helps to prevent the use of a named destination in another client number (i.e., to prevent the client-dependent usage of a defined destination). Destinations are client independent — that is, a user from client 600 can use the same destination as a user from client 800. This capability can pose a security risk. You can, however, minimize the risk by specifying the group of users who are authorized to use the destination.

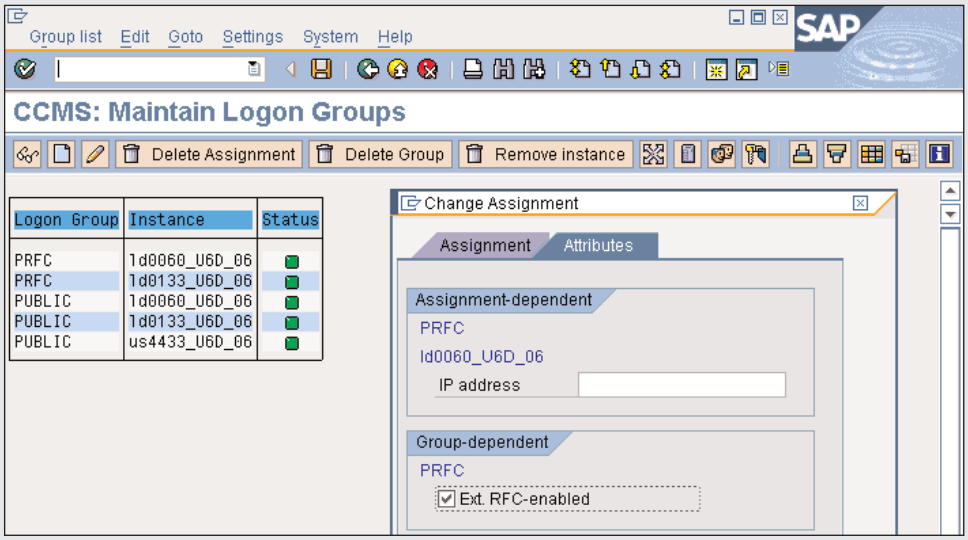
As mentioned earlier, entering values for the language, client, and user fields is optional. However, some considerations may influence your decision to provide values. Let's discuss some of the more critical options in the lower portion of this tab:

- **Language:** The language declaration affects the RFC regarding the code page[14] used by the remote system when executing an RFM. The language you specify here determines how the system transfers and converts character-like data. If the RFC destination does not maintain a language, the system uses the present language (SY-LANGU), that is, the language with which the user is logged on.

[14] We'll discuss code pages in the last section of this article.

Maintaining SAPGUI logon groups

To improve the SAPGUI logon group distributing strategy for SAP Web AS 6.10 and higher systems, SAP recommends you select the Ext. RFC-enabled option for the application servers within the logon group (see the screenshot below). Use transaction SMLG to display the Computing Center Management System (CCMS).



Note!

Character-like data is data of the type CHAR, DATE, NUM, and TIME (CDNT data type in SAP jargon).

- **Current User:** Selecting this option causes the system to ignore any entry in the user name field in the logon dialog because the system takes the specified user ID from the ABAP system field SY-UNAME.
- **Password:** This password field behaves a little differently than you might expect. If you choose the traditional user ID/password logon approach and you do not specify a password, the RFC runtime displays the logon dialog with the specified or default language, client, and user values, so the user only has to enter the password. If you choose the trusted system logon approach and you have properly configured an RFC trust relationship between the two systems, then the password is not required. Note that the specification of the password always is required when a system, user ID, or client number changes. If the user, client, and password are empty and the target server belongs in the same system, a logon under the present user account (i.e., with the user ID SY-UNAME and client SY-MANDT) takes place.
- **Unencrypted Password:** If you activate this option, the system sends the password unencrypted to the server. Choose this option only if you want to create a destination to an SAP R/2 system.

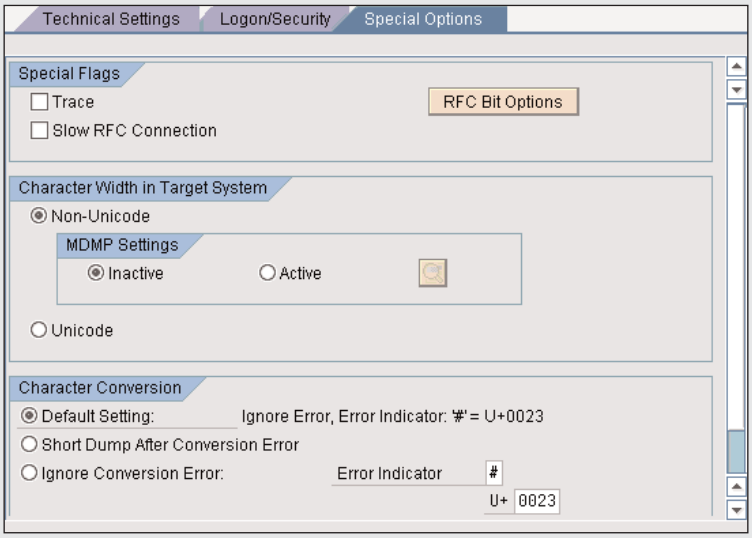


Figure 5 Setting special options for an RFC destination

> ***Note!***
>
> Two connection tests are available for the settings you defined on these tabs. Your first option is the connection test button, which appears on the main screen as part of the function bar. This option only checks the RFC communication pipe without checking the logon settings. For a more complete test, use your second option, which is available on the RFC destination menu bar. For Release 6.40, follow the menu path, Test → Authorization.

Special Options

The final tab is the Special Options tab, which lets you fine-tune several settings of the SAP connections for your RFC destination (see **Figure 5**). The most important of these are the Trace option, the Character Width in Target System options, and the Character Conversion options. Let's take a closer look at each of these.

- **Trace:** You select this option if you need the system to trace your RFC (e.g., when troubleshooting an RFC problem). If you enable this, the system automatically propagates changes to the remote server. Your traces reflect the processing on both the client and server sides. You can review the trace logs by selecting the Display RFC trace option in transaction SM59.

> ***Tip!***
>
> The Trace feature enables you to trace an entire RFC landscape; however, an active trace can have a negative impact on system performance, so make sure to turn it off when you don't need it. The target application server can reject the activation of an RFC trace if you set the profile parameters gw/accept_remote_trace_level and rdisp/accept_remote_trace_level. Refer to the documentation in transaction RZ11 for more information.

- **Character Width in Target System:** You select one of two options to indicate if the target system is a non-Unicode or a Unicode system. If you have a non-Unicode system, you have the option of implementing Multiple Display/Multiple Processing (MDMP),[15] which allows you to use any combination of languages[16] on a single installation. It is very important that you select the correct option. If you choose Unicode, but the server is a non-Unicode system, a short dump occurs on the server side. If you want to use MDMP tables in your communication landscape but the MDMP feature is off, then the content of the language tables (MDMP tables) may be damaged. We will discuss this option in detail in the next section, "Serialization/deserialization principles of ABAP data during RFCs."

Note!

You only can find the Unicode-related settings using transaction SM59 on Unicode SAP systems (i.e., SAP Web AS 6.20 and higher). In the case of non-Unicode systems, both the Character Width in Target System and the Character Conversion sections are grayed out.

- **Character Conversion:** These settings influence the conversion of character-like data. You can decide what happens in the case of conversion errors. You can decide that the system will replace an unconvertible character with a predetermined replacement character, or that an unconvertible character will raise an ABAP runtime error.

[15] MDMP is an SAP solution for language support in which more than one code page can be used on the application server to allow languages to be used together in one system, although the characters used by those languages are not in the same code page.

[16] For a complete listing of languages supported for non-Unicode systems, go to ftp://sapserv3/home/ftp/general/R3server/abap/note.0073606/document/R3_languages.pdf.

The rest of the settings in this section are extremely specific, advanced features that go beyond the scope of this article. For example, the Slow RFC Connection option ignores the default threshold value for a table's compression of 8 kilobytes. By default, tables larger than 8 kilobytes are compressed before sending, and tables smaller than 8 kilobytes are not compressed. If you were to set this flag, the RFC runtime would compress all tables independent of their size. For more information on destination settings, see the "Do not modify SAP's internal, reserved destinations" sidebar on page 55.

Note!

Some characters cannot be translated between languages or Unicode and non-Unicode code pages. With communication between two non-Unicode systems, these non-convertible symbols are not replaced and the system ignores resulting conversion errors. With inhomogeneous communication (one system is Unicode and one isn't), the system either raises the ABAP runtime error RFC_CONVERSION_FIELD, or it replaces the non-convertible character by a special character that you define in the SM59 destination. The default character is # (the pound symbol).

Serialization/deserialization principles of ABAP data during RFCs

In this last section, we discuss seemingly insignificant, but extremely important, options on the Special Options tab — Character Width in Target System and Character Conversion. These settings are so important that if you don't set them correctly, your RFC calls won't work at all (i.e., either the system will raise RFC communication exceptions with each call or some of the data will be damaged after the call).

Do not modify SAP's internal, reserved destinations

SAP automatically maintains a set of predefined, standard destinations for communication with commonly used partner systems (e.g., SAP R/2, SAP R/3, or non-SAP systems).

You can identify these predefined destinations using transaction SM59. These destinations are assigned to the Internal Connections or the Connections via ABAP/4 driver destination type.*

The Internal Connection type includes the following destinations:

- **NONE:** This destination is a "loopback" destination, that is, it refers to the application server on which it is defined. You'll want to use an RFC variant — especially in the case of an asynchronous RFC (aRFC) — that requires this "loopback" destination when you want the call to be executed on the current (local) system. With the help of this destination, the application server can handle RFC communication. This destination is used when RFC requests on the same server should be executed (e.g., processing several aRFCs on the same server). The setting of this destination is aligned with the respective character width of the system (i.e., the Unicode and Non-Unicode options discussed in the "Special Options" section on page 53).
- **BACK:** This destination acts as the "call-back" destination for an RFM. For example, during execution of a synchronous RFC (sRFC) in the RFM, the client system initiates the processing of an RFC in the caller session. You'll want to use this type of destination when certain information from the caller session is required during the execution of the RFM, and due to lack of appropriate importing parameters the caller program does not forward the required information regularly. We recommend you avoid using this destination because its use is restricted to sRFC — you cannot use it with aRFC, queued RFC (qRFC), transactional RFC (tRFC), or parallel RFC (pRFC).

> ***Note!***
>
> The RFC destination SPACE or ' ' (i.e., empty, which is only available for the sRFC type and works as if the extension destination was not specified by call function statement) leads to the local execution of the RFM. In this case the RFM is executed locally and in the same internal mode as the calling program.

- **Destinations with application server names:** When you install a new application server, the installer does two important things.
  - Registers the application server with the message server, so it can begin receiving requests from the application server. The message server extends its list of active application servers in the system to include this server. You can view a list of all available application servers registered on the message server using transaction SM51.

* Connections via ABAP/4 driver mainly are used for transactional and queued RFC (tRFC and qRFC) infrastructures. Internal Connections are NONE, BACK, and all active application servers in a system.

Continues on next page

Continued from previous page

- Creates a destination of type Internal Connections. The installer creates a new destination using the name of the application server. These destinations enable RFC communication among the application servers in a system.

The RFC runtime is able to interpret a list of destination names internally, although the destination names are not explicitly maintained in transaction SM59. The RFC runtime addresses these destinations as destinations of type R/3 Connection. The following destination pattern belongs to this category and cannot be used for destination names for your custom RFC destinations (refer to the information found in the "Technical Settings" section on page 48):

- <application server>_<system id>_<service number> — The purpose of this pattern is to establish an RFC connection to a certain application server. An example of this pattern is usemain_USE_01, where usemain is the target host or application server, USE is the SID, and 01 is the service number of the target server.

Note!

Because the client/server systems use these destinations (i.e., Internal Connections and Connections via ABAP/4 driver destination types), you must not modify or delete any without the guidance of your SAP support team.

Administrators often are surprised by this after upgrading one or more of their remote systems. This section explains what these special options actually do, so you can make better decisions when setting these options and when troubleshooting your installation if you experience RFC communication problems (e.g., all character data in your table has been replaced with #).

To execute an RFM, the RFC runtime has to transmit the function parameters and tables from the client to the server and vice versa. The process of encoding parameters and tables into a stream of bytes is called *serialization*. *Deserialization* is the decoding of such a stream to reconstruct a copy of the parameters and tables on the remote side. Obviously the serialization/deserialization of each parameter or table depends on its ABAP data type.

Although RFC serialization/deserialization is a process handled by the RFC runtime, you need to know the basic principles. If not, your calls to the RFMs may not work properly or may work rather slowly.

Let's look at the following issues — byte ordering schemes, encoding schemes, code pages, and MDMP — which are critical to your understanding of the RFC serialization/deserialization.

Byte ordering scheme

The underlying processor architecture of the installed hardware defines the two byte ordering schemes:

- **Big-endian:** This approach places the most significant byte of a word at the lower address.
- **Little-endian:** This approach places the most significant byte of a word at the higher address.

The byte ordering schemes afffect the internal representation of all data (i.e., integer, float, and even Unicode characters). If the client and the server are running on hardware with different byte ordering schemes, exchanged data has to be converted. The byte order depends on the data type (integers, floats, etc.) and is always converted on the receiver side.

Note!

The terms big-endian and little-endian originate from the different opinions between Lilliput and Blefescu on how to eat a boiled egg in Jonathan Swift's *Gulliver's Travels* (1726). The difference led to a bloody war — that is, in this famous literary work.

Encoding schemes and code pages

To fully grasp the extent of RFC serialization/deserialization, you need to understand what encoding schemes are and what code pages are. You also need to know what these terms mean in the world of SAP. Let's begin with a brief overview of encoding schemes and code pages.

As you may well know, a particularly bothersome technical challenge that arises when systems exchange character data via an RFC is ensuring that both systems properly interpret the characters being passed. Proper interpretation is important because different computer systems can use different codes to refer to the same character. The *code page* defines the code (or value) assigned to each character, and the *encoding scheme* defines the code format that the system implements.

Tip!

The character width is synonymous (for the most part) with the encoding scheme. The Unicode encoding scheme always has a character width of two bytes. Non-Unicode and MDMP encoding schemas always have a character width of one byte.

Now, specific to SAP, you need to know that SAP supports three encoding schemes (and two character widths):

- **Unicode:** Most modern and flexible encoding scheme in which all character-like data types are represented using a two-byte code. All SAP Web AS 6.20 and higher systems are Unicode systems.[17]
- **Non-Unicode:** Scheme in which each character is assigned a one-byte code. All SAP systems earlier than SAP Web AS 6.20 are non-Unicode systems.
- **Multi-Display/Multi-Processing (MDMP):** Alternative to the non-Unicode scheme that uses multiple code pages within the same ABAP session. Basically, from an RFC point of view, it is possible to have character-like data encoded in different code pages within the same table. For example, the first row in the table contains Japanese characters, the second row English characters, the third row Polish, and so on.

When two systems using the same encoding scheme communicate with each other, this is known as *homogeneous communication*. When systems using different encoding schemes communicate, it is called *inhomogeneous communication*.

Note!

You define the encoding scheme during the system installation. You can set up either a Unicode or non-Unicode system. Once you determine the encoding scheme of your system, you cannot change it.

If this weren't complex enough, you have to consider the code page. A code page is a map of the representation of characters to certain numbers. Pre-Unicode,[18] every character is represented by a

[17] Unicode encoding is also known as encoding in the UTF16 format.

[18] For more information on pre-Unicode and Unicode, see the following *SAP Professional Journal* articles: "Globalizing Applications Part 1: Pre-Unicode Solutions" (September/October 2001) and "Looking Forward to the Unicode Advantage: Internationalization and Integration" (January/February 2002) by Michael Redford, and "Processing text-based information in ABAP — best practices for improving performance" by Ralph Benzinger and Björn Mielenhausen (May/June 2006).

byte value. For example, in ASCII, the character A is represented by the number 65. Also, the set of possible characters is limited by 255, so you are not able to represent every character in every language in the world. The solution is to assign to every language its own code page, for example, code page 1100 for English and code page 8000 for Japanese. This means that the same number value may be represented in different characters in different code pages. With Unicode, however, all languages are built into just two code pages: 4102 and 4103.

Note!

Don't be confused by the fact that Unicode has two code pages. Remember, each Unicode character is represented by a two-byte value, and the byte ordering scheme is hardware dependent. The value of the character is the same on each hardware platform, but the byte order differs from hardware platform to hardware platform. On big-endian architectures, the Unicode code page is 4102, and on little-endian architectures, the Unicode code page is 4103.

If the data is encoded in different code pages or even has different encoding schemes, one of the systems must convert it. So, what does this mean to you? Two things:

- First, when defining your RFC destinations, be sure you correctly identify whether the target system uses Unicode, non-Unicode, or MDMP. As you'll learn in the next section, although the RFC runtime includes a sophisticated handshake mechanism by which the client and server systems can automatically agree on a code page to use, determining which character width (i.e., encoding scheme) to use is not automatically determined.
- Second, you need to understand the basic terminology and the procedure/rules the target system uses to determine which character width and code page to use to troubleshoot RFC problems (e.g., interpret the logs or read documentation).

We'll tackle both issues in the next two sections, beginning with character width determination.

Determining the appropriate encoding scheme

Agreement on a character width (i.e., an encoding scheme) is the first of two criteria for two systems to exchange character data via RFC. Life is relatively easy if the two systems use the same character width. Character data can pass straight through, and the target system potentially only has to worry about code page conversions. If the two systems use different character widths, however, one system must covert its output to that of the other to be understood. Furthermore, the Unicode side always converts the data. If a Unicode system initiates the request, it converts the data before sending it. If the Unicode side is the receiver, it converts the data upon receipt. This makes sense because non-Unicode systems (being older) don't not know how to interpret or convert Unicode characters.[19]

In case you forget to select the correct option during installation, the default is that the client assumes the partner is a non-Unicode system and converts the data into the non-Unicode format. For more information, see the "What happens if the Unicode flag is not set correctly?" sidebar on page 59.

[19] Because releases before SAP Web AS 6.20 cannot handle the UTF16 format, the Unicode system must "pretend" to be a non-Unicode system to communicate with the non-Unicode system via RFC. In other words, the non-Unicode system is fooled into believing that it is communicating with another non-Unicode system. This is true whether the Unicode system is the sender or the receiver of the RFC.

Note!

Prior to Basis Release 6.10, code pages were listed in tables that contained the code page number, SAP character ID, and code point for each character. As of Basis Release 6.10, characters are no longer based on SAP character IDs, but on standard Unicode scalar values. In addition, code pages contain a list of segments and a code page building rule. For more information on code pages, go to the SAP Help Portal at http://help.sap.com and search the SAP NetWeaver 2004 library for "code pages."

What happens if the Unicode flag is not set correctly?

There is a bit more to the story than what we have stated on page 58 about forgetting to specify a Unicode or non-Unicode system. The Unicode vs. non-Unicode option also tells the system which RFC version to use. A newer version of the RFC protocol was established after Unicode became an industry standard. This presumes that an older system — that is, a non-Unicode system — cannot understand the Unicode-RFC protocol.

If the Unicode flag is set and the partner is a non-Unicode system, the data is sent using the Unicode protocol. Because the non-Unicode system is not able to handle this protocol version, the ABAP runtime error RFC_INVALID_PROTOCOL_VERSION occurs on the server side.

With a transactional RFC (tRFC), the invalid Unicode flag leads to the failing logical unit of work (LUW). If you set the Unicode flag for the defined destination, but the partner is a non-Unicode system, all tRFCs called with this destination fail. You can view the list of failed LUWs using transaction SM58. Restarting the LUW does not help because the partner system is not able to handle the tRFC data sent by the Unicode protocol.

The situation that results when the server is a Unicode system but the Unicode flag is not set is rather interesting. The server side is able to understand both protocol versions, so the Unicode server accepts the received data. The system converts the received data from the non-Unicode format into the local (Unicode) format. Obviously, the conversion on the client side into the non-Unicode format is not necessary. Also, during this conversion, some data may be lost, so it is useful to switch the connection to the Unicode protocol. According to the received header data, the server recognizes that the data is coming from a Unicode system. The server changes the used protocol version to Unicode and sends the response data back to the client in the Unicode format to avoid the useless and dangerous conversion from Unicode format into non-Unicode format. The client recognizes, according to the received metadata from the server, that the partner is a Unicode system. Recognizing that the received data is already Unicode-encoded, the server accepts the data and switches the protocol version, too. After the handshaking mechanism is complete, the communication through the underlying connection is done using the Unicode protocol, as long as the connection remains active.

Note!

Adjusting the Unicode flag on the target system does not successfully restart the failed LUW. You need to manually restart the entire process for creating the tRFC after maintaining the destination (transaction SM59). You should then manually delete the failed LUW using transaction SM58.

Determining a code page

The second criterion you must determine for successful RFC communication is the correct choice of the code pages. Remember that the character width, and not the code page, of both partners determines whether the communication is homogeneous or inhomogeneous.

The following four guidelines will help you better understand how code page determination is rendered and how the conversion between systems works:

- First, the local code page on a Unicode system is fixed — it is either 4102 or 4103 depending on the underlying processor architecture. It does not depend on the language with which the user is logged on to the system. For example, the Unicode code page on Intel platforms is always 4103; whereas, on the Scalable Processor Architecture (SPARC) platforms[20] the Unicode code page is 4102.
- Second, the code page on a non-Unicode system depends on the logon language used for the RFC session on the target system. For example, code page 1100 contains English characters and code page 8000 is used for encoding Japanese characters.
- Third, in the case of homogeneous communication the receiver converts the data from the remote code page into the local code page. That is, if the code page for system A is 1100 and the code page for system B is 8000, data received by system A is converted from 8000 to 1100, and data received by system B is converted from 1100 to 8000. The same is valid for Unicode-to-Unicode communication.
- Finally, with inhomogeneous communication, the Unicode system handles the conversion. This shouldn't be too surprising since the non-Unicode system doesn't know how to convert characters to the Unicode code pages. The Unicode system converts the data into a communication code page that comes from the local logon language. Additionally, the communication code page may change during the RFC call. We will explain this procedure in a little more detail next.

If all this has you a bit confused, **Figure 6** should clear things up. The top two images cover the homogeneous communication. As you can see, the data is always converted on the receiver side. The bottom image illustrates inhomogeneous communication. In this case, the communication code page of the Unicode system is 8000. Although the non-Unicode system believes that it is talking with another non-Unicode system, the Unicode system converts the data as usual.

> ***Note!***
>
> Remember that the RFC data serialization/deserialization mechanism does not differ between client and server roles. Only the sender/receiver role is significant for this process. Therefore, the scenario Unicode-to-non-Unicode is equivalent to the scenario non-Unicode-to-Unicode.

Recall that the SAP RFC runtime includes a handshaking mechanism by which the RFC client and server systems can automatically agree on a code page to use. Knowing the implemented code page is useful when you are troubleshooting system behavior or interpreting documentation. You don't need to make any specific settings when configuring your RFC destinations. The handshake takes place between the RFC client and server at the first call only. To illustrate how the RFC handshaking mechanism works, let's look at **Figure 7**.

The user logs on to a Unicode system via SAPGUI. The SAPGUI logon language is EN (i.e., English using ISO notation). There is an Intel-based architecture so the fixed local code page is 4103 (which is based on the UTF16 little-endian). On the right side, there is a non-Unicode system and the local logon language is JA (for Japanese); therefore the code page is 8000. Remember that you maintain the

[20] Scalable Processor Architecture (SPARC) is a big-endian reduced instruction set computer (RISC) architecture originally designed in 1985 by Sun Microsystems.

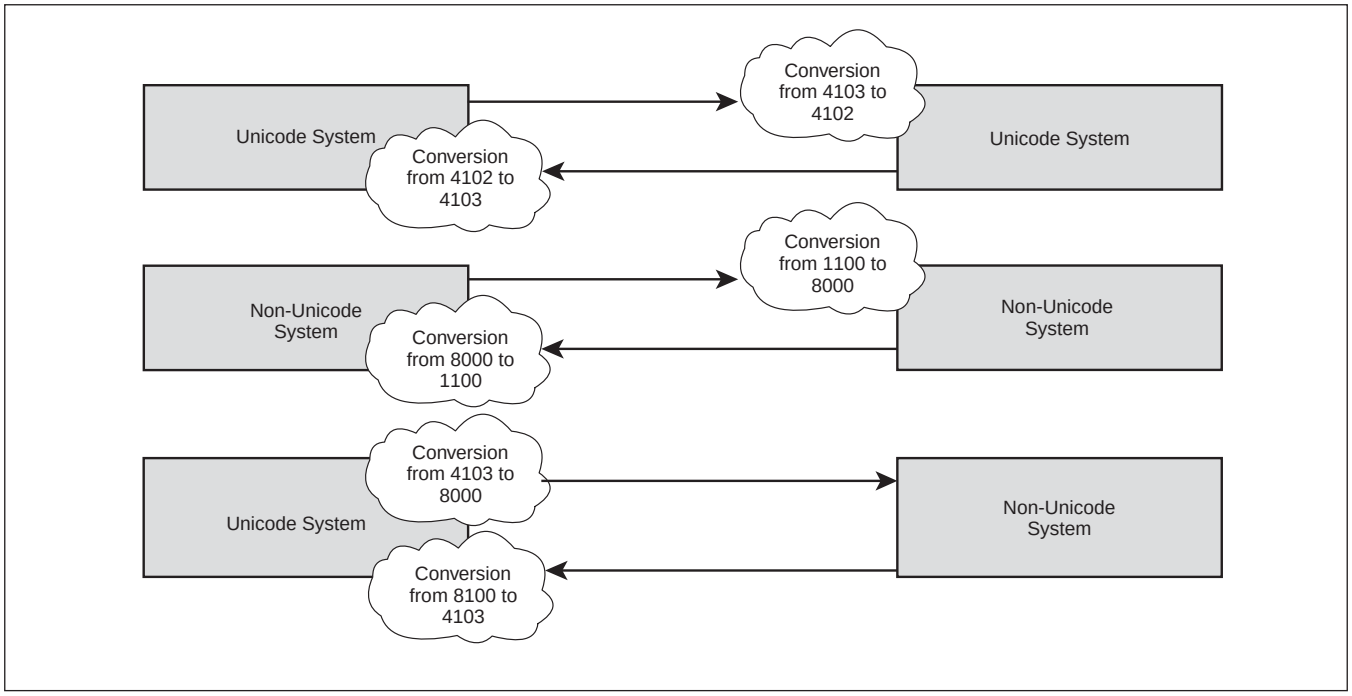


Figure 6 Code page conversion in RFC communication

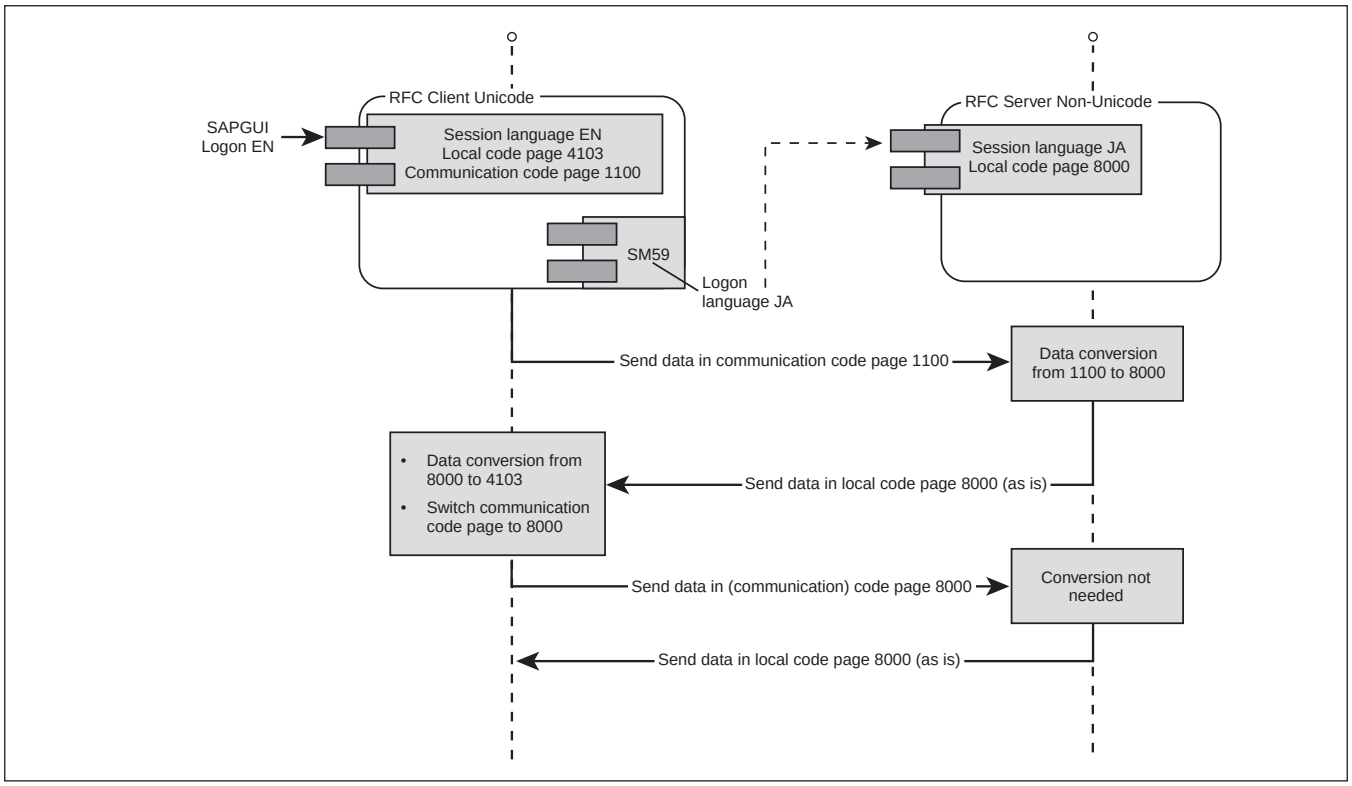


Figure 7 Handshake between RFC communication partners

(local) logon language for the server session in transaction SM59.

When the RFM is called (by the Unicode system), the server session is created with the code page 8000 (associated with the Japanese logon language maintained in SM59). During the first call the communication code page on the client side is derived from the local logon language (EN), so the communication code page is set to 1100.

When the Unicode client initiates the RFM, it converts the data from code page 4103 to code page 1100 and sends it to the server. The server converts the received data from remote code page 1100 to local code page 8000; therefore, the response to the RFM is sent "as is" in local code page 8000. In the receiving step, the client converts data from remote code page 8000 into local code page 4103 and in addition recognizes that local communication code page 1100 does not match server code page 8000. To avoid the useless conversion of the data on the server side, the client switches the communication code page from 1100 to 8000. The next time the Unicode client sends data via an RFM, the Unicode system will convert the character data from code page 4103 to code page 8000.

Note!

The logon language maintained in the RFC destination using transaction SM59 is not relevant for the local session. Neither the local code page nor the communication code page for the client is derived from SM59 values. The language setting in SM59 is relevant for the server only.

Implications for MDMP systems

MDMP is a rather complex topic, so our discussion here is only in relation to the content of this article — that is, RFC communication and in particular the structured data as defined in the ABAP type LANGU field, which enables instances of this data type to be multi-page. In other words, you can choose to store your textual information in a code page that differs from the local session code page. For example, if you want to retain your Japanese data despite the fact that you are logged on with English as the logon language, all you have to do is specify "J" for Japanese as the value for the LANGU field. Then you are able to fill your text data with Japanese content independent of your logon language (English).

RFC communication supports MDMP tables only in inhomogeneous RFC communication (i.e., communication between Unicode and non-Unicode systems) and only for table parameters. Other constellations — for example, MDMP tables such as IMPORTING, EXPORTING, and CHANGING parameters as part of the nested data types — are not supported.

Note!

The language format is the SAP proprietary format and not the ISO format. For example, English is represented by "E" and not by "EN," and Japanese by "J" and not by "JA."

In the case of MDMP tables, you have inhomogeneous communication, so the Unicode system converts all data. This principle of data conversion for flat data types is the same as for non-MDMP tables. Instead of using the communication code page of the current session, the RFC runtime converts the table content row by row from the Unicode format into a specific code page that is derived from the value of the LANGU field in the current row of the table.

Figure 8 illustrates the handling of an MDMP table and non-MDMP table in an inhomogeneous RFC environment. The MDMP table (the upper table) has several rows. The contents of two rows are English and Japanese text. To handle this, the content of the LANGU field is "E" for English and "J" for Japanese, respectively. Another table (the table on the bottom) is

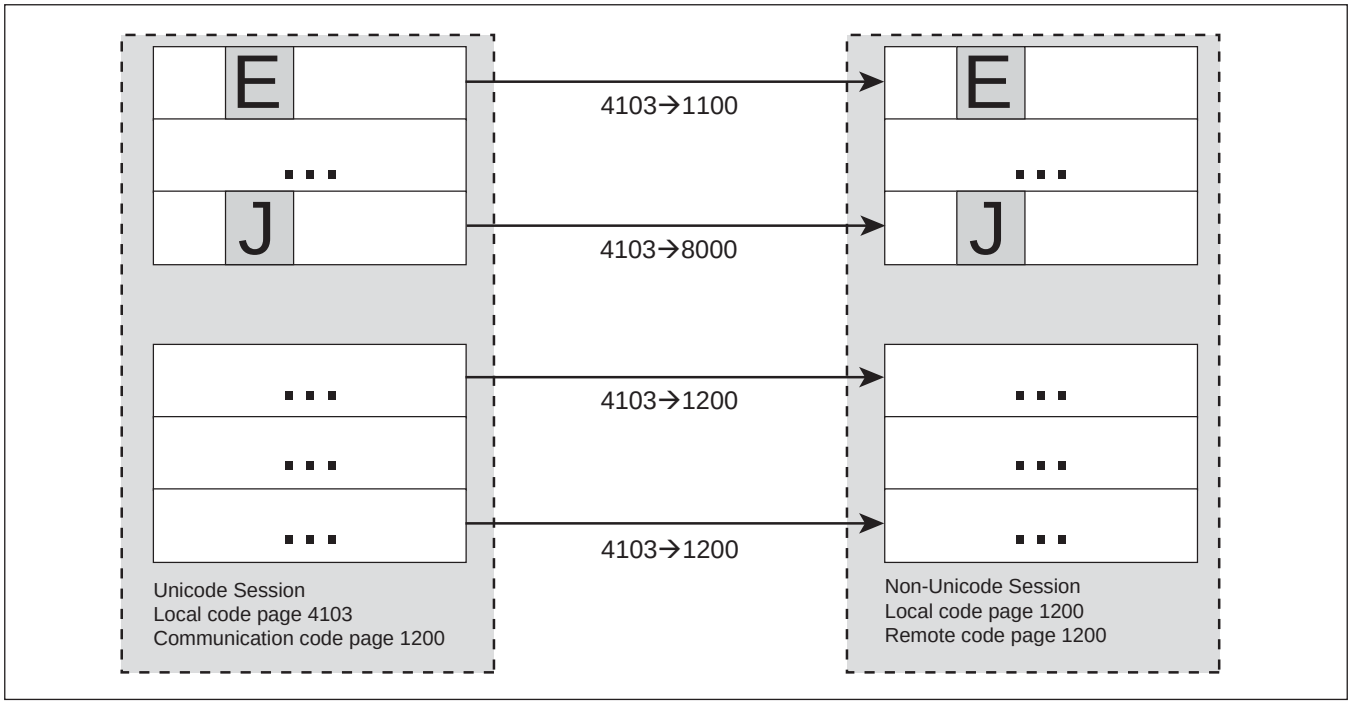


Figure 8 MDMP and non-MDMP table handling in RFC communication

a non-MDMP table (i.e., there are no fields of type LANGU). Rather, there are two sessions: a Unicode session on the left with local code page 4103 (UTF16) and a non-Unicode session on the right with local code page 1200. The Unicode session acts as the sender. The MDMP table is converted row by row from UTF16 format (code page 4103) into a code page determined by the value of the LANGU field. The communication code page is not used. Another table without the LANGU field is converted from code page 4103 (UTF16 format) into a communication code page, in this case 1200.

Note!

MDMP behavior concerns only character-like data types. Other data types such as strings, integers, floats, and bytes are not involved in MDMP at RFC runtime and are converted as usual.

Handling RFC exceptions and messages

During the execution of an RFM, a variety of errors can occur. RFC classifies the potential errors in two categories:

- **Communication failure:** This kind of error usually occurs because of RFC connection interruptions and character set conversion errors. The RFC runtime maps this sort of error to the special RFC-specific exception COMMUNICATION_FAILURE, which has to be treated by the RFC client adequately.
- **System failure:** This kind of error usually occurs when, during execution of an RFM, a message of type E (error), A (abort or termination), or X (exit) is raised, or an unwanted ABAP statement is executed (e.g., invalid input data, wrong program logic, incorrect use of ABAP statements, etc.). The RFC runtime maps these kinds of errors to the RFC-specific exception SYSTEM_FAILURE, which has to be treated by the RFC client adequately.

Note!

The RFC client specifies both the RFC-specific exceptions COMMUNICATION_FAILURE and SYSTEM_FAILURE. With the help of the addition MESSAGE <text>, the application is able to get a description of the error situation. We recommend treating RFC exceptions properly. For example, you should record exception messages in an application log.

Note!

In the case of communication errors, you might find related error information in the affected application server's work process, RFC, or gateway trace file (in the dev_w<work process number>, dev_rfc<work process number>, or dev_rd file, which you can access via transaction ST11).

Generally, in the case of a COMMUNICATION_FAILURE, the location of the communication malfunction is not immediately detectable — it is somewhere between the client and server work processes, including involved gateway processes.[21] This is not the case with SYSTEM_FAILURE. This kind of error occurs most frequently on the RFC server.

Every (remote) function module supports ABAP exceptions and ABAP messages. You specify ABAP exceptions in the interface of the function module and maintain those exceptions via transaction SE37. If the RFM raises an ABAP exception, the control flow of the module is immediately interrupted and the system sends the exception to the RFC client.

The reaction of the RFC runtime when it receives an RFM exception changed after Release 7.00. The client releases from 7.00 and higher receive all available data declared in the TABLES section of a function module if an ABAP exception occurs on the server side. Available data means all data the server wrote into corresponding tables until the exception was thrown. This behavior is similar to the behavior of the locally called function modules. The older client releases ignore all payload data in the TABLES section of a function module if an ABAP exception is raised at the server side.

[21] For more information on the RFC communication process see our first article "Master the five remote function call (RFC) types in ABAP — Part 1: A comprehensive guide for SAP programmers and administrators" (*SAP Professional Journal*, September/October 2006).

The reaction of the RFC runtime on the ABAP messages depends on the message type:

- **Message types E, A, and X:** These message types lead to the termination of the RFM processing. In this case, the RFC runtime propagates to the RFC client both the associated message text maintained in the T100 database table and the corresponding ABAP system fields SY-MSGID, SY-MSGTY, SY-MSGV1, SY-MSGV2, SY-MSGV3, and SY-MSGV4. The RFC client treats those messages with the help of the RFC exception SYSTEM_FAILURE, and with the addition MESSAGE, the RFC client can receive the raised message text.

- **Message types W (warning), I (information), and S (status):** The RFC runtime does not execute any actions when these message types are raised, but rather continues processing the ABAP statements after the MESSAGE statement. If these message types are raised during screen processing in an RFC session, these messages are covered by the screen processing engine.

The RFC runtime treats the message statement with the addition `RAISING` exceptions (i.e., the control flow of the function module is interrupted and both the exception and the corresponding ABAP system fields SY-MSGID, SY-MSGTY, SY-MSGV1, SY-MSGV2, SY-MSGV3, and SY-MSGV4 are sent to the RFC client).

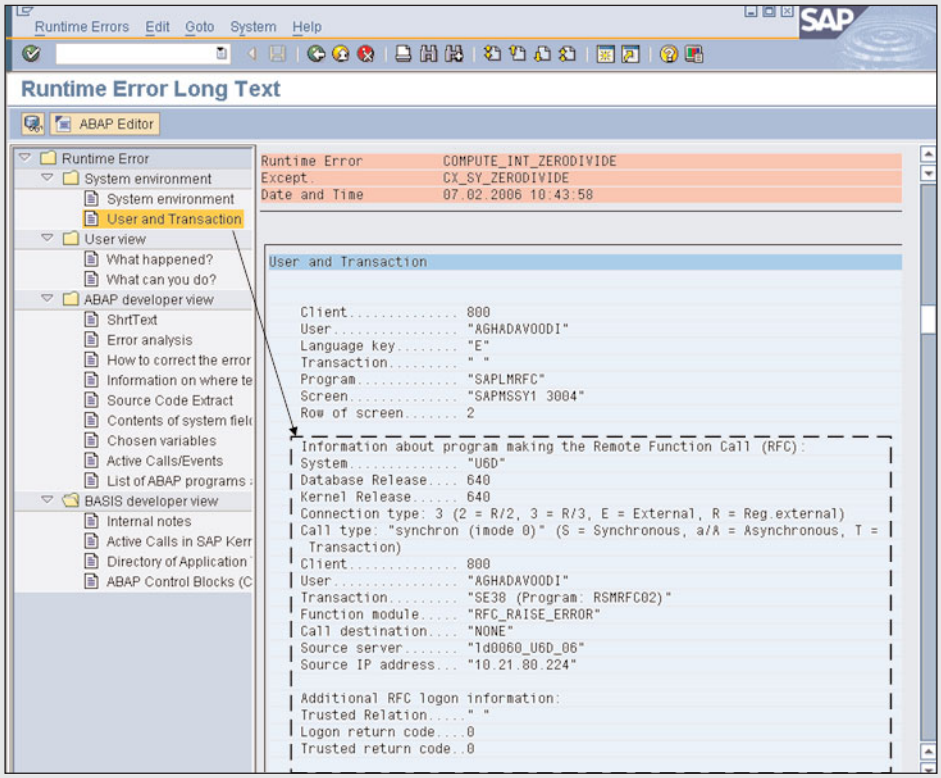


Figure 9 RFC-related section in ABAP runtime errors

The ABAP runtime logs information regarding the cause of the error in different places depending again on the message type:

- **Message type X or ABAP runtime errors:** The ABAP runtime errors include a detailed description of the error situation and the involved ABAP sources. Those messages are recorded in transaction ST22 on the affected system. If an ABAP runtime error occurs during the execution of an RFM, you can find the error in the User and Transaction section, as shown in **Figure 9**.
- **Message type A:** You can find a short description of the error situation in transaction SM21 on the affected application server in the target system.

> ***Note!***
>
> The throwing of an ABAP application exception never leads to the closing of the underlying RFC connection; therefore, the RFC session remains active. However, if a client receives a `COMMUNICATION_FAILURE` or `SYSTEM_FAILURE` error message, it means that the underlying RFC connection is closed, and the RFC session on the server side no longer exists.

- **Message type E:** You might find related error information in the affected application server's

work process or RFC trace file (in the dev_w<work process number> or dev_rfc<work process number> file, which you can access via transaction ST11).

Note!

An RFC engine provides the profile parameter rfc/dump_connection_info to enable the output of additional information related to the error context if this error occurs in an RFC session. The RFC output takes place in the work process and RFC trace files (in the dev_w<work process number> and dev_rfc<work process number> files, which you can access via transaction ST11).

Conclusion

This is the second of two articles — the first of which provided with you a solid foundation about the five types of RFCs. To complement what you learned in the first article, this article has presented information on the following topics as they relate to RFC communication on SAP and non-SAP systems:

- Technical prerequisites and guidelines for building robust custom RFMs
- Technical settings and special options for defining RFC destinations
- Serialization and deserialization principles, including issues such as byte ordering schemes, encoding schemes, code pages, and MDMP
- RFC exception and message handling

Your understanding of the RFC communication process should prepare you for the challenges you may be facing in your own IT environment and help you get more from upcoming RFC articles. Look for an article in a future issue of *SAP Professional Journal* on the new improved type of transactional and queued RFC called background RFC (bgRFC).

Masoud Aghadavoodi Jolfaei studied computer science and received his doctorate in the area of satellite communication at Aachen University of Technology. He joined SAP AG in 1994 and became a member of the ABAP Connectivity group, where he works as a development architect on the design, tools, and rollout of the ABAP communication infrastructure. In addition, Masoud is responsible for the integration of Internet protocols (HTTP, HTTPS, and SMTP) into the ABAP runtime. You may reach him at masoud.aghadavoodi.jolfaei@sap.com.

Eduard Neuwirt joined SAP AG in 1999 and became a member of the ABAP Connectivity group, where he worked on the design, tools, and rollout of the ABAP communication infrastructure. Eduard was also responsible for the development of the remote function call (RFC) tools on the external side, including the RFC library and JRFC. Since September 2005, Eduard has worked for the SAP Defense and Public Security Department. He is responsible for the interfaces to external military non-SAP systems. You may reach him at eduard.neuwirt@sap.com.

