

# BC415

## Remote Function Calls in ABAP

THE BEST-RUN BUSINESSES RUN SAP

© SAP AG 2007

- System R/3
- Collection 62
- Materialnummer 50086522

**Internal Use SAP Partner Only**

**Internal Use SAP Partner Only**

**Copyright 2006 SAP AG. All rights reserved.**

**No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.**

© SAP AG 2006

- Some software products marketed by SAP AG and its distributors may contain proprietary software components of other software manufacturers.
- Microsoft, Windows, Outlook and PowerPoint are registered trademarks of Microsoft Corporation.
- IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation in the USA and/or in other countries.
- Oracle is a registered trademark of Oracle Corporation.
- UNIX, X/Open, OSF/1 and Motif are registered trademarks of the Open Group.
- Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

- 
- HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- Java is a registered trademark of Sun Microsystems, Inc.
- JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- MaxDB is a trademark of MySQL AB, Sweden.
- SAP, R/3, mySAP, mySAP.com, xApps, xApp and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in other countries around the world. All other product and service names are trademarks of their respective companies. The information provided herein is non-binding and is intended for information purposes only. Products may have country-specific variations.
- The information contained herein may be changed without prior notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only. The SAP Group does not assume any liability or provide guarantees for errors or incompleteness in this publication. The SAP Group is solely responsible for products and services according to requirements explicitly defined in the agreement regarding the respective products and services. No further liability is assumed as a result of the information contained in this publication.

## Prerequisites



- **SAPTEC training course**
- **BC400 training course**
- **BC414 training course (highly recommended as preparation for remote database changes)**

## Target Audience



- **Participants:** ABAP developers involved in cross-system communication through RFC from the R/3 System
- **Duration:** 2 days



© SAP AG 2002

### Notes to the participant

- The training materials are **not self-teach programs**. They **complement the course instructor's explanations**. On the sheets, there is space for you to write down additional information.

- **General Goals**
- **Course Objectives**
- **Course Content**

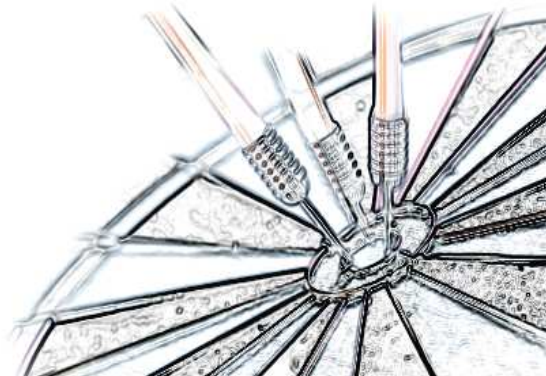


© SAP AG 2002



**This course will provide you with:**

- Knowledge of the use of the RFC interface on the R/3 system side for data exchange between the R/3 System and other application systems
- Knowledge of the basics of using BAPIs



© SAP AG 2002





- The implementation of RFCs in the R/3 System for data exchange between R/3 and other application systems
- Overview of the concept and use of BAPIs as an interface to R/3 data

© SAP AG 2002

## Course Content



### Preface

---

- Unit 1      **Course Goals and Contents**
  - Unit 2      **Introduction**
  - Unit 3      **Remote Function Call**
  - Unit 4      **BAPI - Business Application Programming Interface**
- 

**Exercises**

**Solutions**

**Appendix**

© SAP AG 2002

### **Cross-system data distribution:**

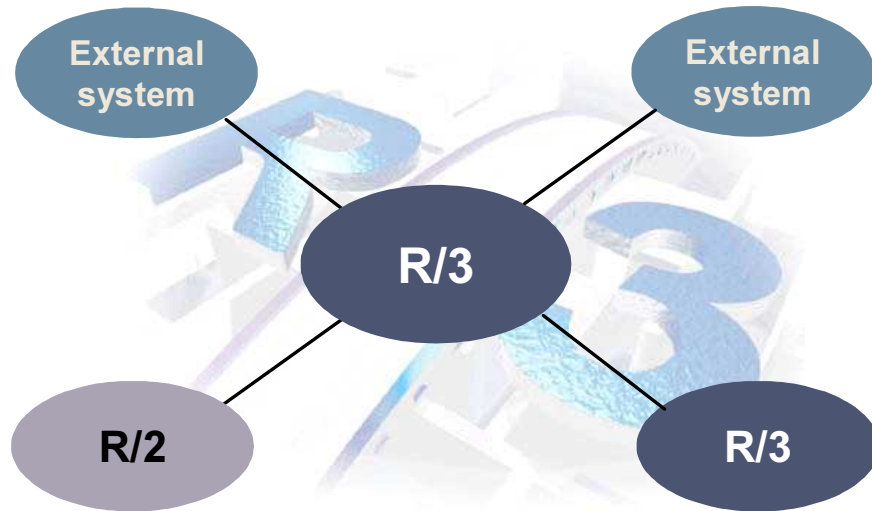
- **Basic terms**
- **Environment and architecture**
- **Technical background**

© SAP AG 2002



- You will learn about the basic concepts, the environment, and the architecture of cross-system data distribution.
- You will receive an overview of existing interfaces that you can use to maintain data consistency in distributed systems.
- You will learn where to get support when choosing, designing, and implementing interfaces.
- You will become familiar with the technical background of cross-system communication.

© SAP AG 2002



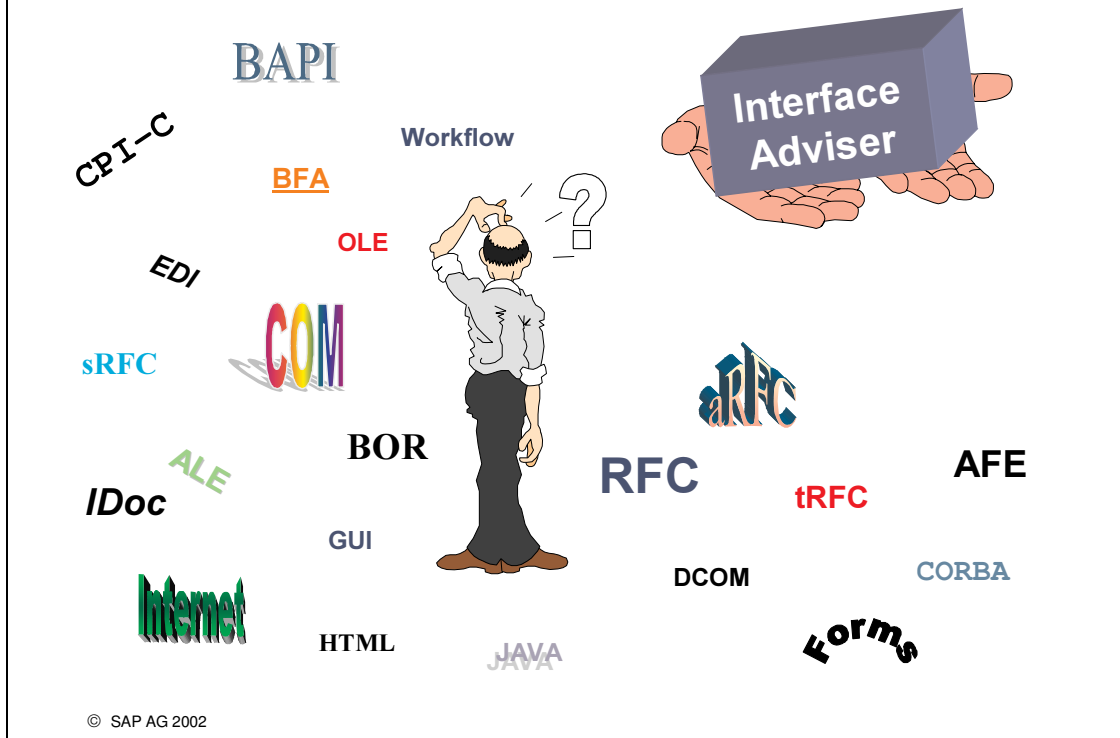
© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only

## Choosing and Designing the Interface

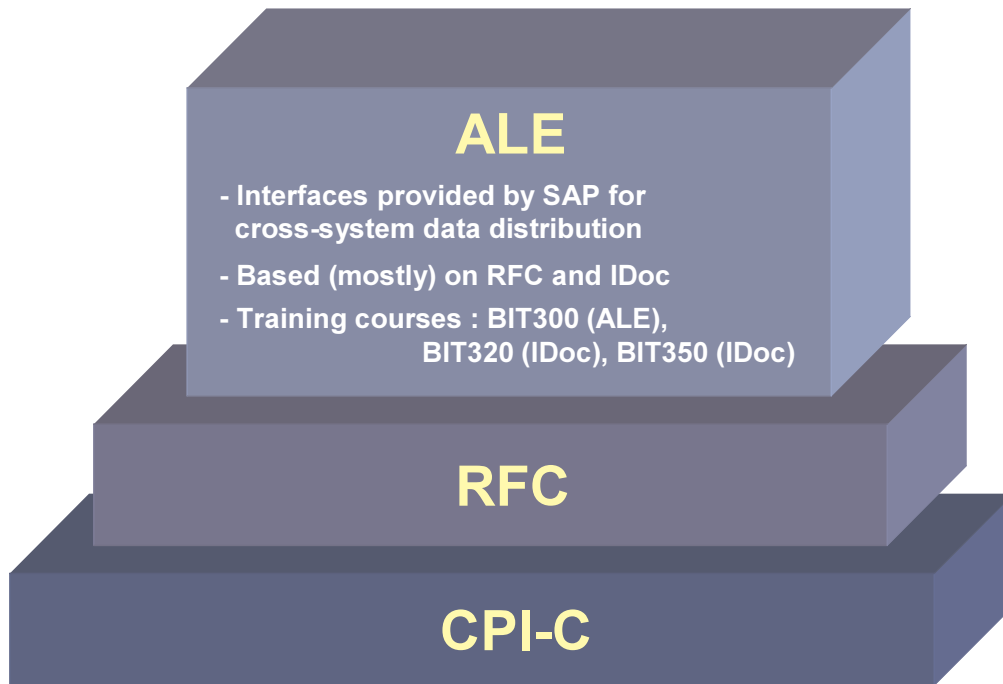
SAP



- Choosing a suitable interface technology is a central task in an integration project. The **Interface Adviser** supports you when choosing the interface technology, and when designing your interface.
- **SAP Integration Technologies**, a Knowledge Product, gives you detailed information on implementing the interfaces.

## Overview of Interfaces for Data Distribution

SAP

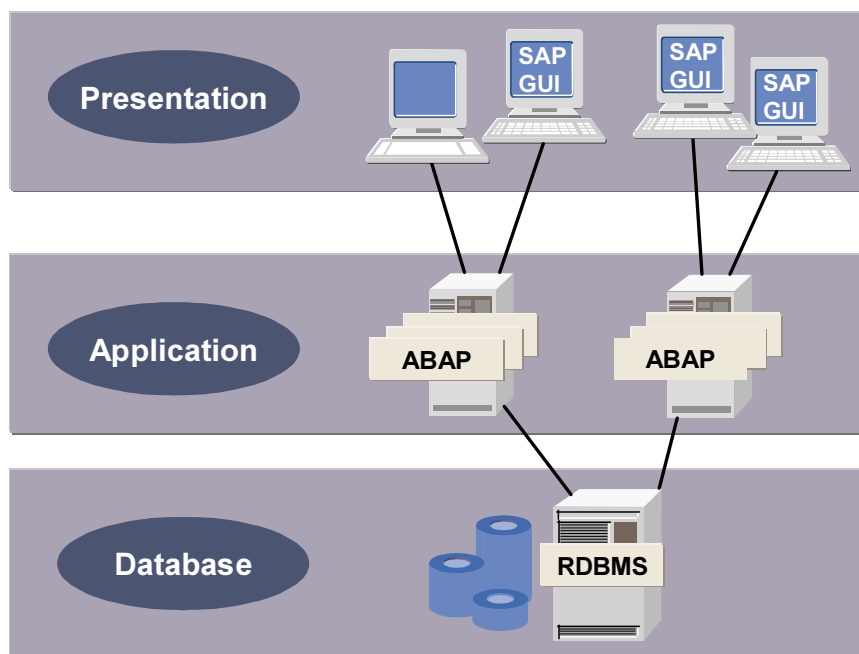


© SAP AG 2002

- RFC is based on CPI-C, which means an RFC call is always transmitted through CPI-C calls.
- CPI-C (Common Programming Interface - Communication) is a program-to-program communication interface that allows communication between programs. It consists of so-called CPI-C calls that a program uses to set up a connection to another program, transmit or receive data, and then disconnect the connection again.
- Since CPI-C is more complicated to use than RFC, it should only be used under the following conditions:
  - If RFC is not available (few platforms)
  - If older programs that communicate through CPI-C need to be maintained
  - If a more complex communication protocol is required (with RFC, only simple input/output handling is possible)
- In the appendix, you will find additional information about CPI-C with exercises and solutions.

## R/3 Architecture

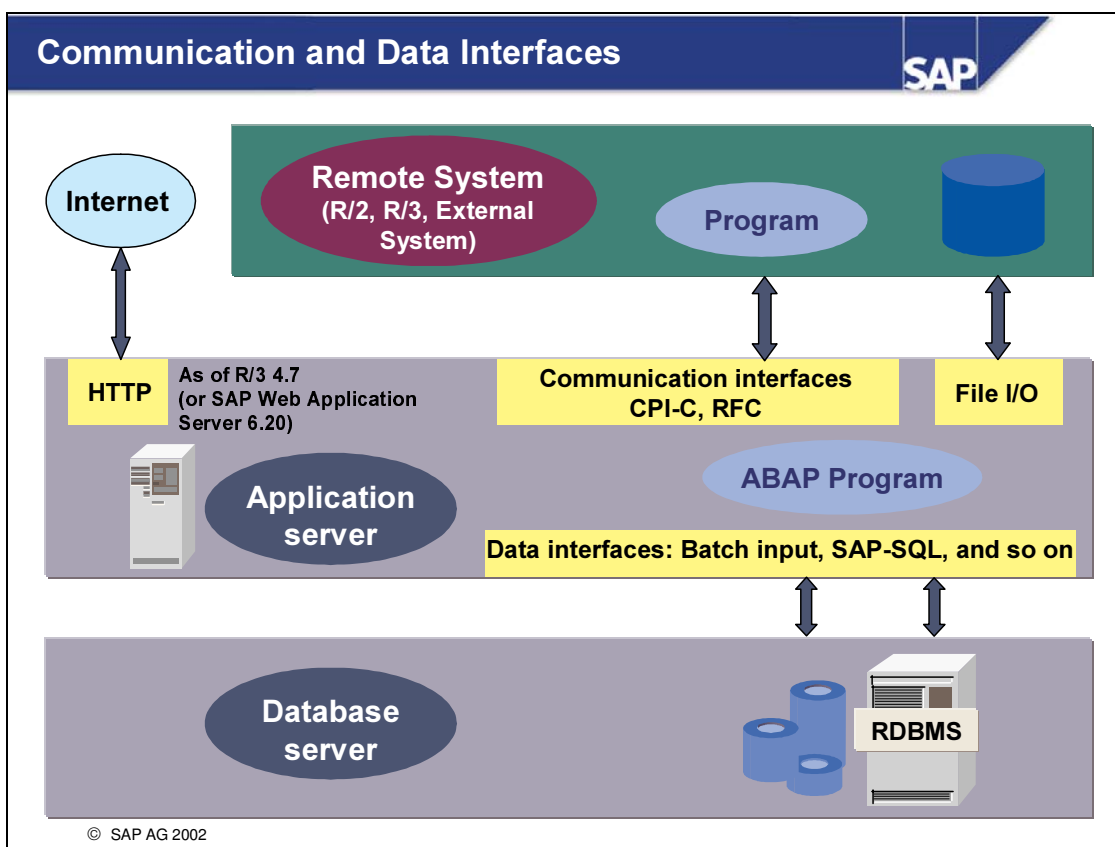
SAP



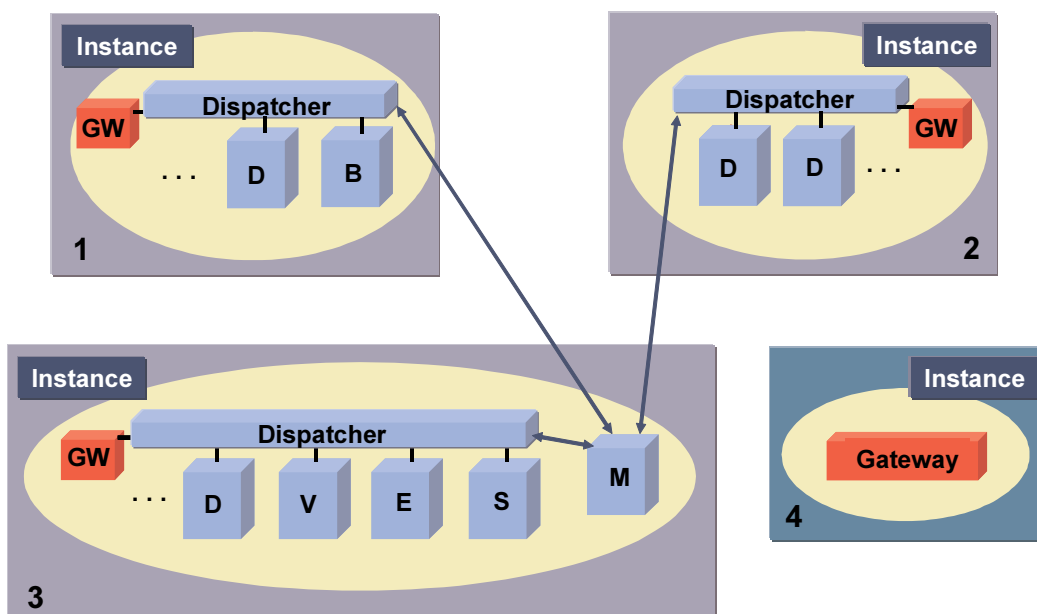
© SAP AG 2002

- The SAP R/3 System has a modular software architecture that follows the software-oriented client/server principle.
- The system distributes presentation, application logic, and data across hosts at different levels. This forms the basis for the high degree of scalability in the R/3 System.
- The lowest level is the **database level**. The data is managed at this level using a relational database management system (RDBMS). In addition to master and transaction data, the database also stores programs and metadata that describes the R/3 System and the structure of its data.
- The second level is the **application layer**, which consists of ABAP programs. You can complement the existing SAP functions by writing your own programs using the ABAP Workbench.
- The third level is the **presentation layer** (SAPGUI), which contains the user interface through which end users access the application functions, enter data, and display the results of their work. You can also use alternative front ends (AFEs) or interfaces at this level instead of the SAPGUI.





- Communication interfaces and data interfaces are different. In addition to the file I/O, the communication interfaces are the gate to external communication, whereas the data interfaces are used for read and write access to the R/3 database.
- The data interfaces include Batch Input, Direct Input, CALL TRANSACTION USING, SAP-SQL (OpenSQL), Native SQL  
For more information on these interfaces, refer to the training courses BC420 and BC414.
- As of SAP R/3 Release 4.7 or SAP Web Application Server 6.20, it is possible to exchange data between ABAP programs and applications over the Internet using the HTTP protocol (instead of the RFC protocol). This topic is covered in training course BC416.

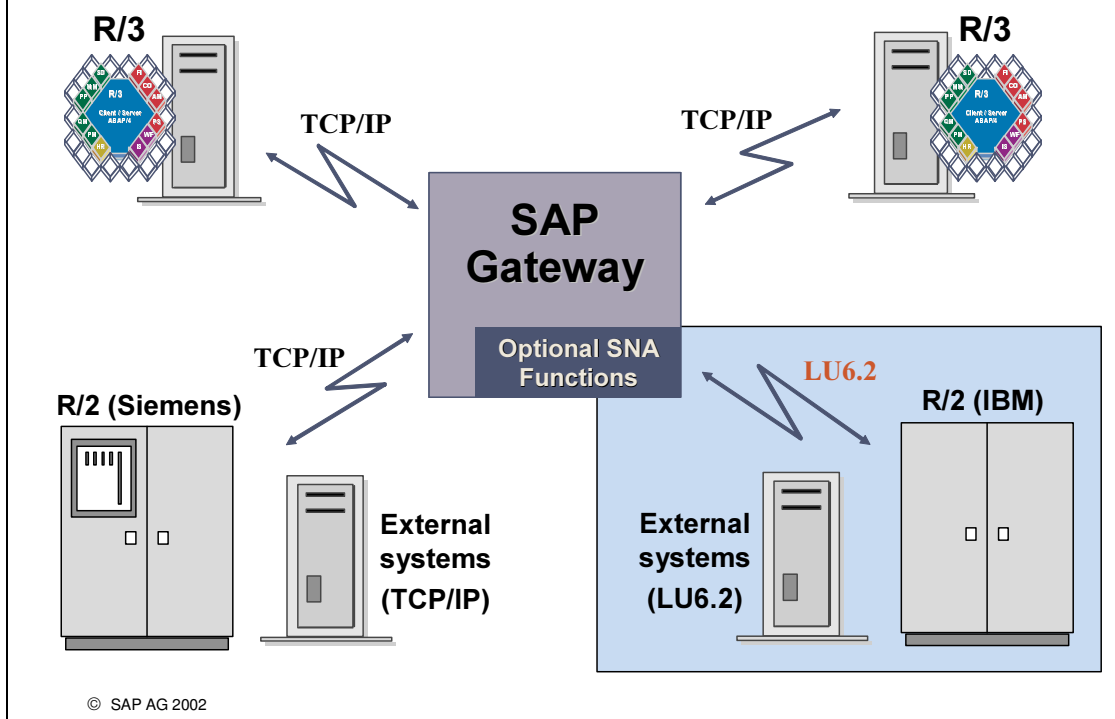


© SAP AG 2002

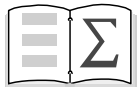
- An **instance** is an administrative unit in the R/3 System. It describes which services (dialog, update, enqueue, background, spool, gateway) are available on the corresponding application server.
- The **dispatcher** process on an application server coordinates tasks within the respective instance. It **communicates** with other instances (application servers) in the same R/3 System through the message server and, through the gateway service, with external systems. As of Release 3.0, a gateway service (application server) belongs to each instance (application server). You can also install a gateway service on a non-R/3 host.
- When an application server (dispatcher) communicates with external systems, it usually uses its own gateway work process. However, it may be more useful to use a gateway that runs on a dedicated host, for communication with SNA, where a gateway with an add-on is required. The reason is to minimize the effort needed for configuration and maintenance (the gateway add-on is only on one host instead of each application server). You may have to use a remote gateway, for example, with remote Windows NT applications because the remote program can only be started using this gateway.

## Architecture and Use of the SAP Gateway

SAP



- R/3 communication always runs through the SAP gateway and is based on TCP/IP.
- For R/3 communication with an IBM R/2 System or other LU6.2 systems where the LU6.2 transport protocol is used, you need an SAP gateway with additional SNA functions. The optional SNA functions on the SAP gateway mainly consist of the protocol converter for the TCP/IP LU6.2 conversion (SAP) and LU6.2 functions (external software). An SAP gateway with SNA functions is also known as an (SAP) SNA gateway.




- The *Interface Adviser* supports you when choosing the interface technology and when designing your interface.
- SAP offers, together with the ALE component, predefined interfaces for maintaining data consistency in distributed systems.
- RFC is based on CPI-C. However, CPI-C should be used only if absolutely necessary, due to its complicated usage.
- There is a difference between communication interfaces and data interfaces.
- R/3 communication always runs through a gateway.


© SAP AG 2002

- Overview
- RFC Destinations
- Synchronous RFC
- Asynchronous RFC
- Transactional RFC
- Queued RFC
- RFC with External Programs
- Use of RFC Types and RFC Authorizations

© SAP AG 2002

# Overview





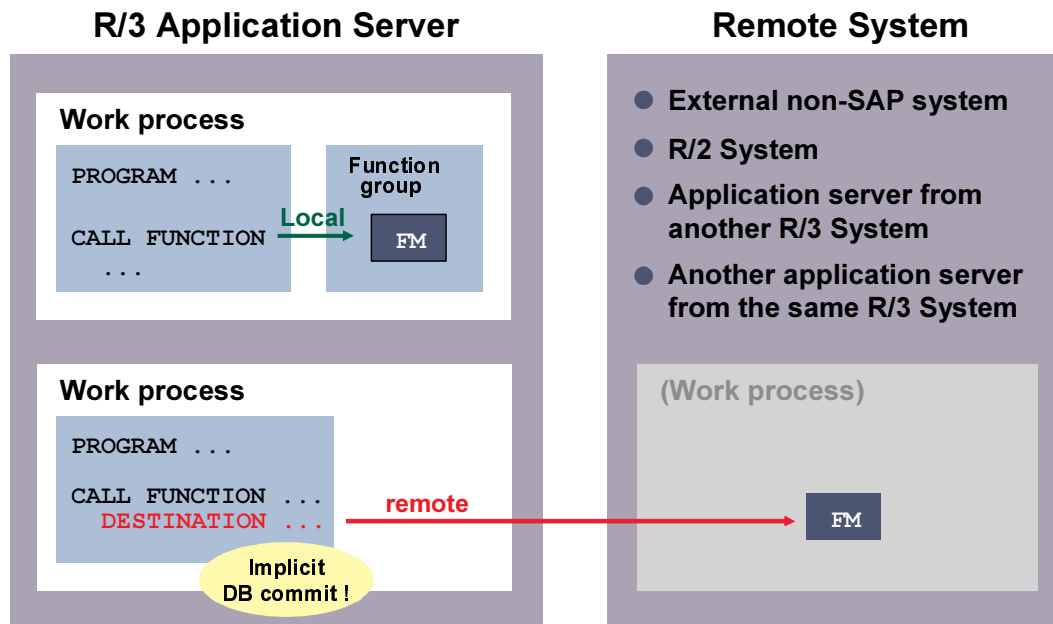
- Overview
- RFC Destinations
- Synchronous RFC
- Asynchronous RFC
- Transactional RFC
- Queued RFC
- RFC with External Programs
- Use of RFC Types and RFC Authorizations

© SAP AG 2002



- This topic will provide you with an overview of the RFC environment and the various RFC types.

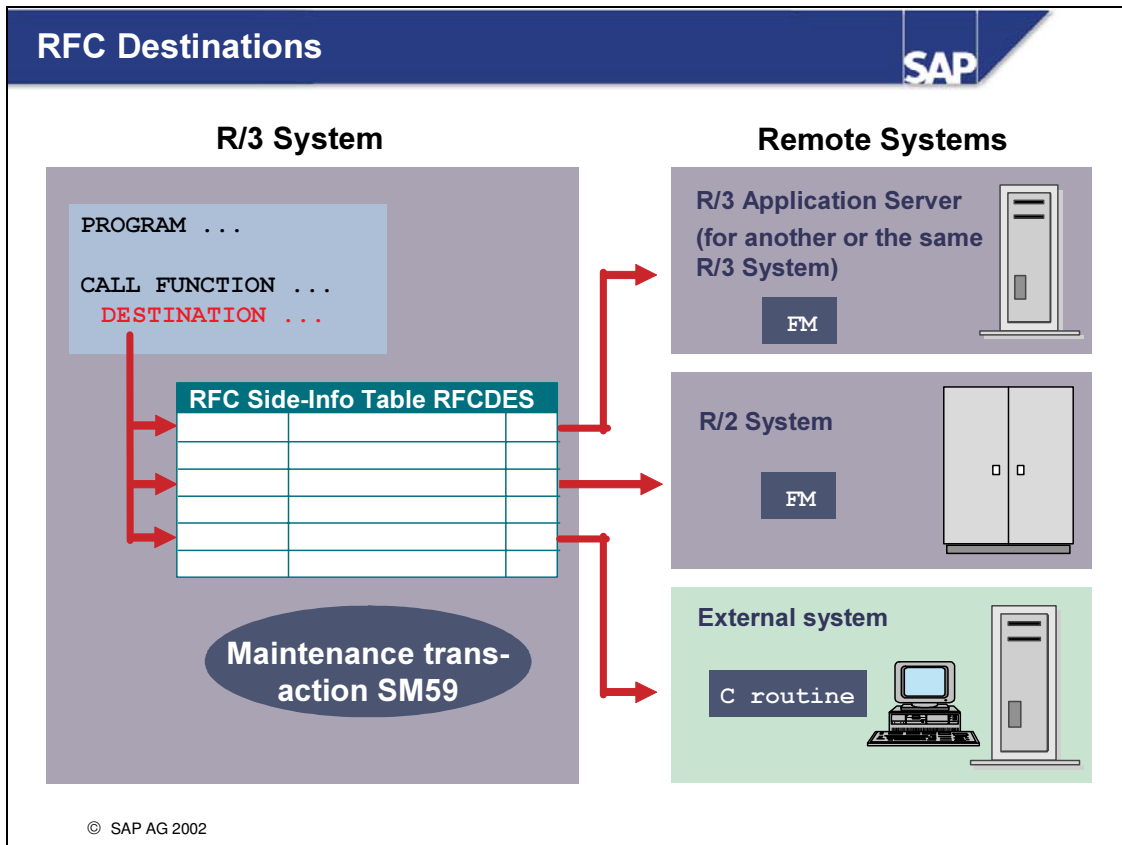
© SAP AG 2002



© SAP AG 2002

- When you call a function module locally, it runs in the same work process as the calling program.
- If a function module is called remotely, it runs in its own work process (its own SAP LUW) if the remote system is an SAP R/3 System.
- The remote destination can be another application server in the same or a different SAP R/3 System, an SAP R/2 System, or an external non-SAP system.
- Note that the calling program is rolled out for each remote function call (RFC), which triggers an implicit database (DB) commit.





- Use transaction SM59 to maintain RFC destinations that must specify an ABAP program when calling remote function modules in the RFC side information table, RFCDES.
- For each partner system, you need to maintain one entry in this information table.

## R/3 Function Modules That Can Be Called Remotely

SAP

### Properties

#### Processing type

- ☐ Normal function module
- ☒ Remote-enabled module
- ☐ Update module

### interface

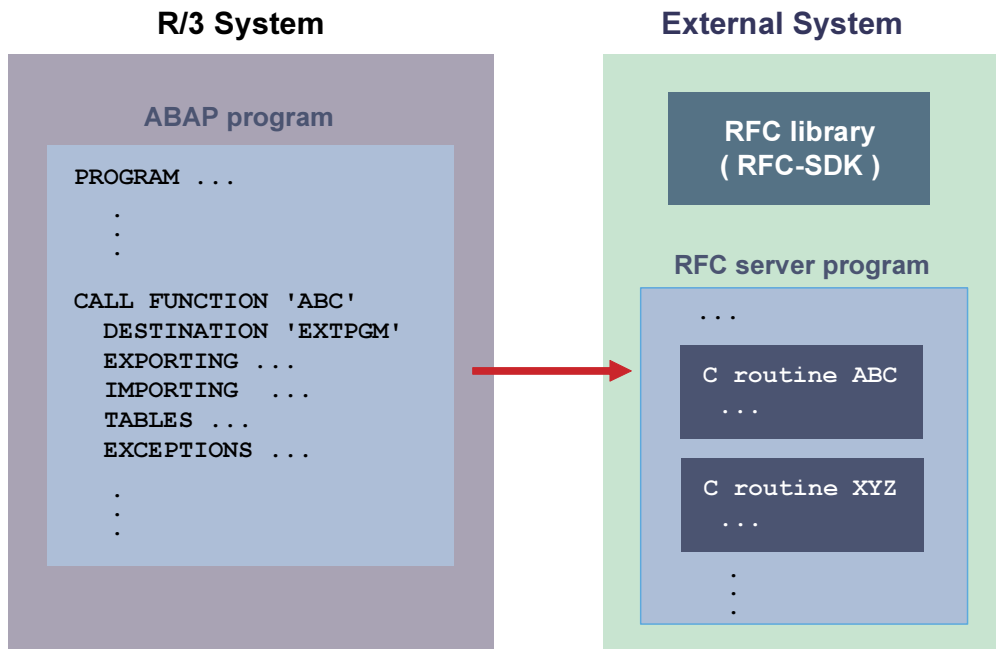
- ★ CHANGING parameter not allowed \*)
- ★ Type assignment for all interface parameters is required (by specifying a Dictionary field/structure)

© SAP AG 2002

\*) Possible as of SAP R/3 Release 4.7 or SAP Web Application Server 6.20

- As of SAP R/3 Release 4.6C, nested structures can be transferred even for remote-enabled function modules.

## Calling External C Routines Using RFC

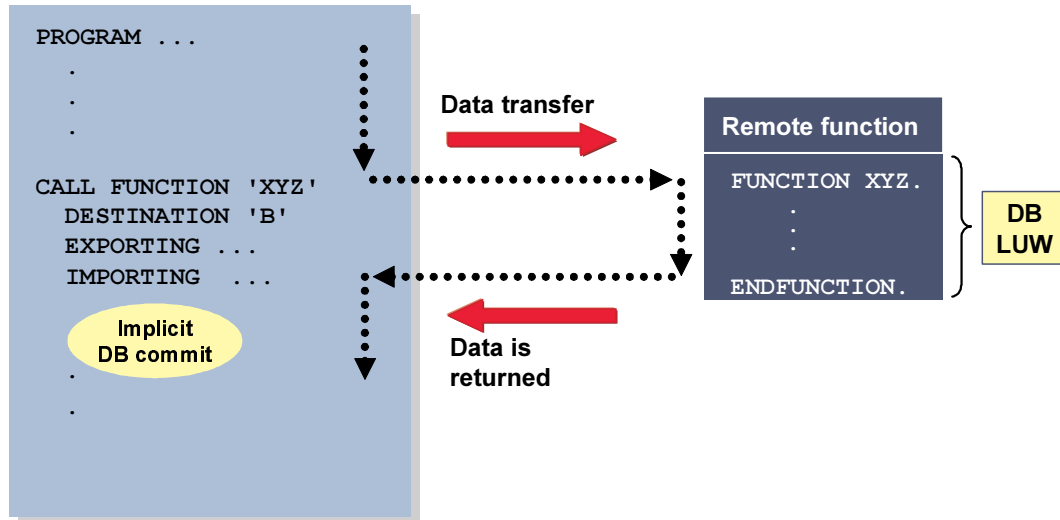


© SAP AG 2002

- SAP delivers the RFC Library for all standard external platforms that contain RFC calls for C programs. After installing the RFC Library on your external platform, you can use the RFC client or RFC server programs with RFC calls.

## Process Flow: Synchronous RFC (sRFC)

SAP

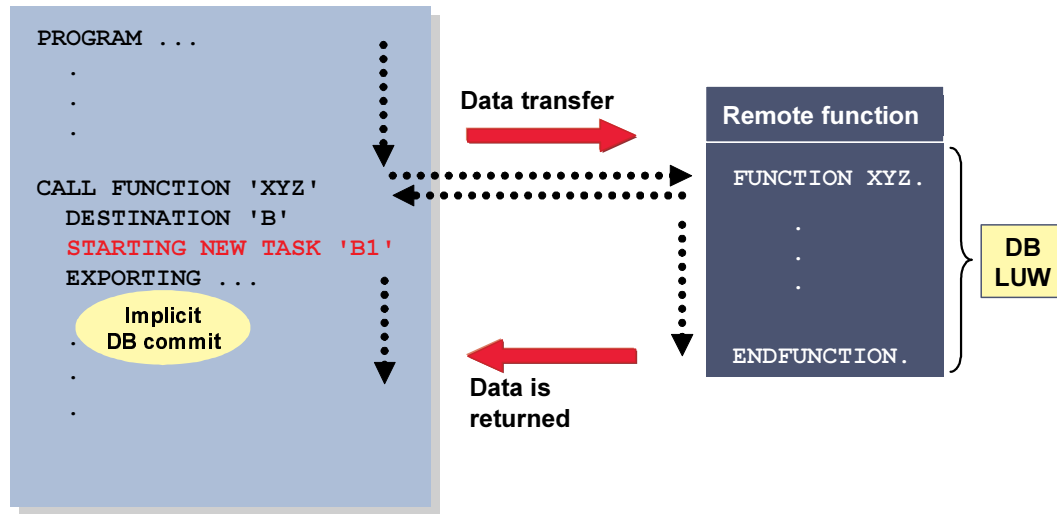


© SAP AG 2006

- With synchronous RFCs, processing is stalled in the calling program until the called remote function is processed and its output is returned. Then processing continues in the calling program after the call.

## Process Flow: Asynchronous RFC ( aRFC )

SAP

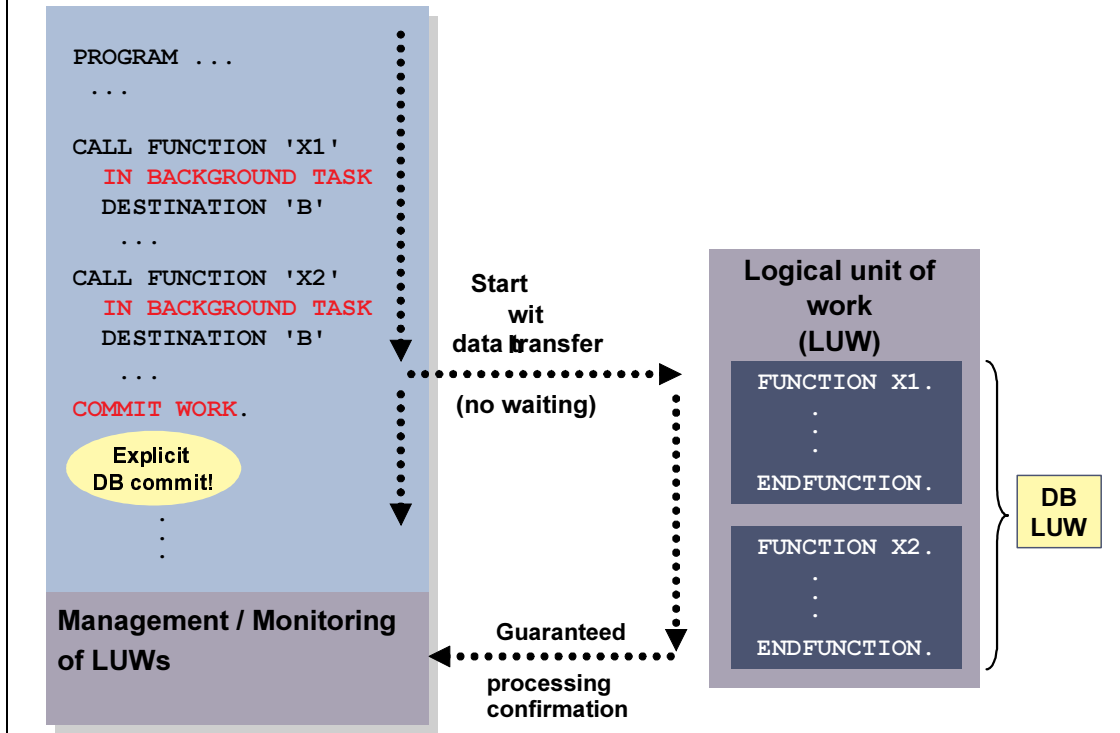


© SAP AG 2006

- In an asynchronous Remote Function Call, the called remote function is started and processing continues immediately afterwards in the calling program. The remote function itself is processed separately from the calling program. The output of the function can be received later in the program.
- Asynchronous RFCs are intended for parallel processing of processes.

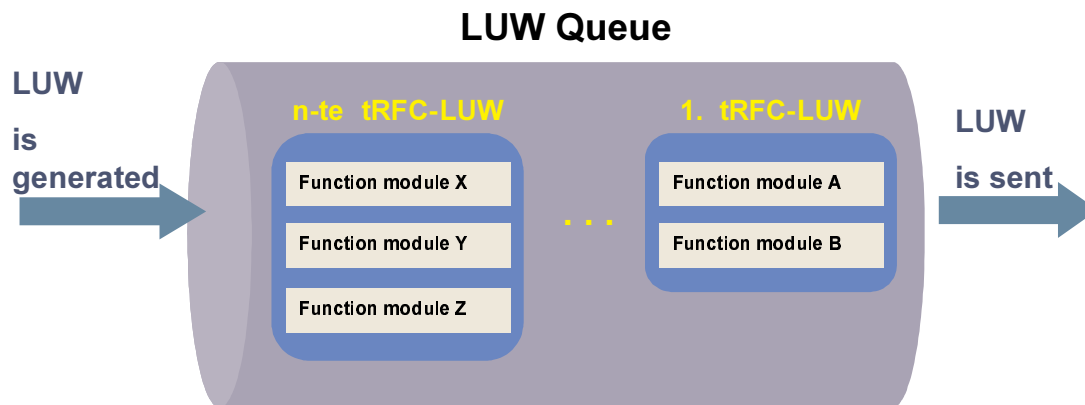
## Process Flow: Transactional RFC (tRFC):

SAP



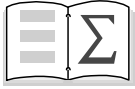
- Whereas with synchronous and asynchronous RFCs each call makes up a single logical unit of work (LUW) in the remote system, for transactional RFCs you can group several remote functions in one LUW (with automatic rollback mechanism if there is an error).

## Serialization of tRFC LUWs



© SAP AG 2002

- LUWs generated using tRFC are processed independently of each other. This means the order in which they are processed is not always the order in which they are generated.
- To ensure that tRFC-LUWs are processed in the same order in which they were generated, use qRFC as an extension of tRFC.
- Queued RFC (qRFC) is available as of release 4.6A and can be used in SAP R/3-R/3 connections and in SAP R/3 external connections.





- You can use RFC to call ABAP function modules in SAP systems and C routines in external server programs.
- To do this, you need an RFC destination that you maintain in the side information table RFCDES.
- There are four call variants: synchronous, asynchronous, transactional, and queued RFC. Select the appropriate RFC type depending on the purpose and requirements involved.

© SAP AG 2002



## RFC Destinations





- Overview
- RFC Destinations**
- Synchronous RFC
- Asynchronous RFC
- Transactional RFC
- Queued RFC
- RFC with External Programs
- Use of RFC Types and RFC Authorizations

© SAP AG 2002



- You are able to maintain RFC destinations for communication processes stemming from an SAP R/3 System.
- You are familiar with the standard destinations SPACE, NONE, and BACK

© SAP AG 2002

## Transaction ?SM59

### RFC Destinations

- ▷ R/2 connections
- ▷ R/3 connections / ABAP connections
- ▷ HTTP connections to R/3 system / ABAP system
- ▷ HTTP connections to ext. server
- ▷ Internal connections
- ▷ Logical connections
- ▷ TCP/IP connections
- ▷ Connections using ABAP drivers

\*)

© SAP AG 2006

\*) Possible as of SAP R/3 Release 4.7 or SAP Web Application Server 6.20

- Using transaction SM59, you maintain the RFC destination in the RFC side information table RFCDES.
- You can maintain RFC destinations in the source system only.
- Depending on the remote system, maintain one destination for the corresponding type.
- As of SAP R/3 Release 4.7 or SAP Web Application Server 6.20, there are two new HTTP destination types. Using these, it is possible to exchange data between ABAP programs and application in the Internet on the basis of the HTTP protocol (instead of the RFC protocol). For further information, see training course BC416.

## Destination for R/3-R/3 Connection

SAP


Type 3 Destination

|  |   |                        |                                  |
|--|---|------------------------|----------------------------------|
| RFC destination  | <input type="text" value="ABC"/>                              | <b>Gateway options</b> |                                  |
| Connection type  | <input type="text" value="3"/> <b>R/3 connection</b>          | Gateway host           | <input type="text" value="..."/> |
| Load distribution  | <input type="radio"/> Yes <input checked="" type="radio"/> No | Gateway service        | <input type="text" value="..."/> |
| Target host  | <input type="text" value="hs5001"/>                           | System number          | <input type="text" value="00"/>  |
| <b>Description</b>   |   |                        |                                  |
| <input type="text" value="RFC connection to remote R/3 System ABC"/> |   |                        |                                  |
| <input type="text"/>   |   |                        |                                  |
| <input type="text"/>   |   |                        |                                  |
| Language   | <input type="text" value="EN"/>                               |                        |                                  |
| Client   | <input type="text" value="800"/>                              |                        |                                  |
| User   | <input type="text" value="SMITH"/>                            |                        |                                  |
| Password   | <input type="text" value="*****"/>                            |                        |                                  |

© SAP AG 2002

- If you want to connect to another SAP R/3 System, you need a destination with type 3. Note that the name of a type 3 destination is case sensitive, that is, a distinction is made between lowercase and uppercase letters.
- If you want to ensure that a particular R/3 application server is addressed, set the *Load distribution* option to *No*. In this case, you must specify the R/3 application server using the *Target host* and *System number* parameters.  
However, if you do want to use load distribution when you log on, set the *Load distribution* option to *Yes*. In this case, you must specify the *system ID*, the *message server*, and the required *logon group*. You can find out this information from transactions SM51 (overview of R/3 servers) and SMLG (overview of logon groups) in the SAP R/3 target system. You determine the message server of an SAP R/3 System using the profile parameter *rdisp/mshosou*. You can display this using transaction RZ11 or report RSPFPAR.
- In the destination system, you can specify the logon data of a user in the target system. This must be a user of the type *dialog* or *CPIC* (or of the type "communication" in more recent releases). When the destination is used, the logon to the target system occurs automatically.  
If you do not enter the logon language in an RFC destination, the logon language of the current caller is used.  
If the logon data is not complete or is faulty, a logon screen from the target system appears at runtime.

## Destination for R/3-R/2 Connection



### Type 2 destination

|  |  |                 |                                  |
|--|--|-----------------|----------------------------------|
| RFC destination  | <input type="text" value="K50"/>   | Gateway options |                                  |
| Connection type  | <input type="text" value="2"/> <input type="text" value="R/2 connection"/> | Gateway host    | <input type="text" value="..."/> |
|  |  | Gateway service | <input type="text" value="..."/> |
| Description  |  |                 |                                  |
| <input type="text" value="RFC connection to remote R/2 System K50"/> |  |                 |                                  |
| <input type="text"/>   |  |                 |                                  |
| <input type="text"/>   |  |                 |                                  |
| Language   | <input type="text" value="EN"/>  |                 |                                  |
| Client   | <input type="text" value="004"/>   |                 |                                  |
| User   | <input type="text" value="SMITH"/>   |                 |                                  |
| Password   | <input type="text" value="*****"/>   |                 |                                  |

© SAP AG 2002

- If you want to connect to an SAP R/2 system, you need a destination with type 2. In the destination, you need to specify the logon data of an R/2 user with the type CPIC. You must also specify the SNA gateway (for IBM R/2) or the host gateway (for Siemens R/2) through which the connection is to be made.
- Note that the destination name (in the figure, it is K50) for a type 2 destination cannot be chosen at random. It must match the name of the corresponding destination in the gateway side information table (for more information, see the online documentation on maintaining gateways).

## Destination for R/3 External Connection

SAP

### Type T destination

RFC destination

Connection type

Gateway options

Gateway host

Gateway service

Activation type

☒ Start

☐ Registration

Start on

☐ Application server

☒ Explicit host

☐ Front end workstation

Explicit host

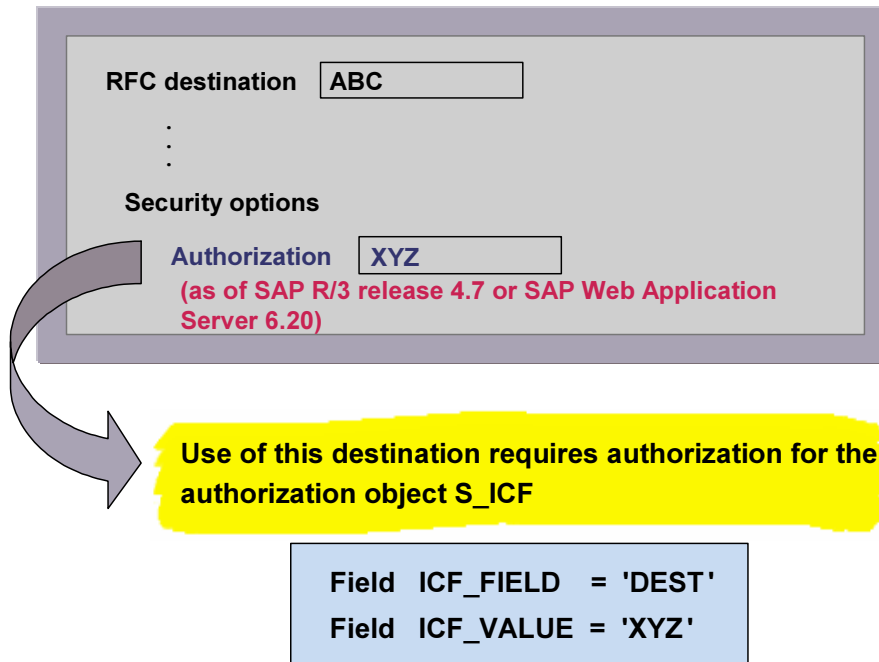
Program

Target host

Description

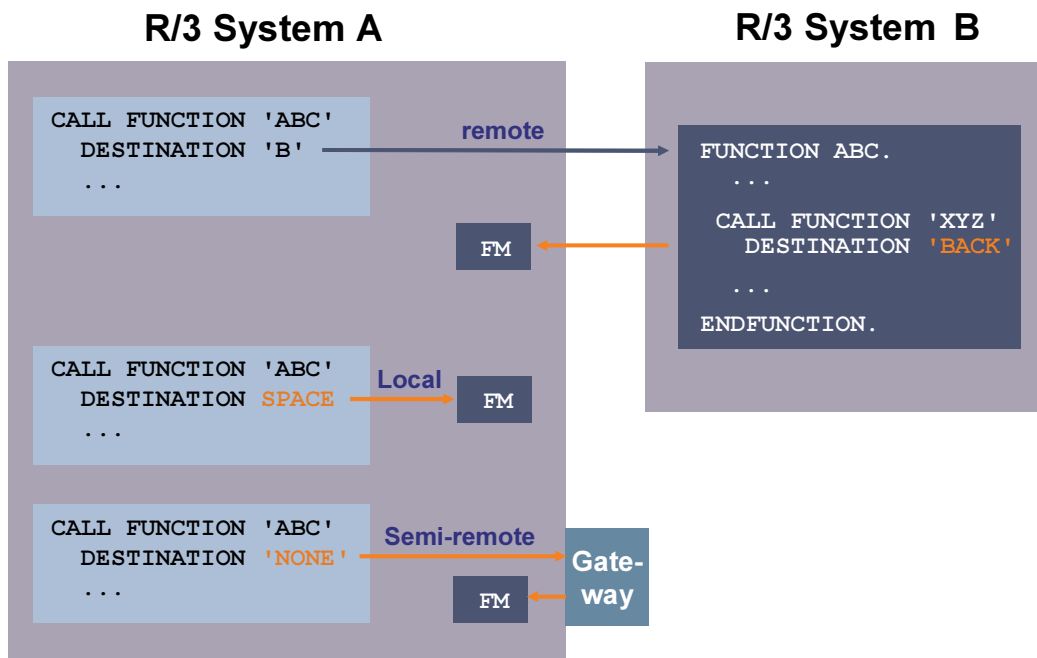
© SAP AG 2002

- For connections to external RFC server programs, you need a destination with the type T. There are two variations of type T:
  1. **Start:** The external program is started when the actual communication is being established.
    - a) *On the application server:* The external program is started on the current SAP R/3 application server. You must give the complete program path including the program names.
    - b) *On the explicit host:* The external program is started on the specified host. You must specify the following:
      - Host name --> Name of the machine where the external program is located
      - Path --> Complete path including program names on an external host
    - c) *Front end workstation:* The external program is started on the current front end at runtime through the SAPGUI. You must enter the complete (absolute) path of the program including program names.
  2. **Registration:** These destination variations let you access a program that has already started and has registered itself on a gateway. This program must be started manually beforehand and registers itself using an RFC call with a program ID on a gateway (see the online documentation). With this variant, the start time is not required. It is suited for starting a program frequently. You must specify the following:
    - Program-ID --> ID under which the external server program has registered on the SAP gateway
    - Gateway --> Host and TCP/IP service of the gateway where the external program has registered (If the gateway is the one for the current application server, this information is not required.)



© SAP AG 2002

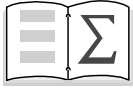
- As of SAP R/3 Release 4.7 or SAP Web Application Server 6.20, it is possible to have an automatic check of the required authorization for destination use. It can be implemented for all destination types.



© SAP AG 2002

- The destinations listed above are predefined standard destinations and have the following meanings:
  - SPACE: Local call of a function module
  - NONE: Also a local call, but the call request is sent to the default gateway of the current application server and is handled there like an external call request. This destination is suited to the RFC test if you only have one system to test.
  - BACK: You can use this destination if you want to start a function module from the called function module in the current system. (This functions with synchronous RFC only.)
- You cannot use these standard destinations from R/2 systems.  
Destination BACK cannot be used for R/3-R/2 connections.





- To make a remote call to a function module, you require an RFC destination in the source system. This is an entry in the RFC side information table RFCDES, which you can maintain using transaction SM59.
- This type of RFC destination contains communication parameters and, if required, the logon data necessary for setting up the connection or the logon in the remote system.
- Depending on the remote partner system, you need to define an RFC destination of the appropriate type.
- SPACE, NONE, and BACK are predefined, standard destinations with special meanings.

© SAP AG 2002

## Synchronous RFC



Overview

RFC Destinations

Synchronous RFC

Asynchronous RFC

Transactional RFC

Queued RFC

RFC with External Programs

Use of RFC Types / RFC Authorizations

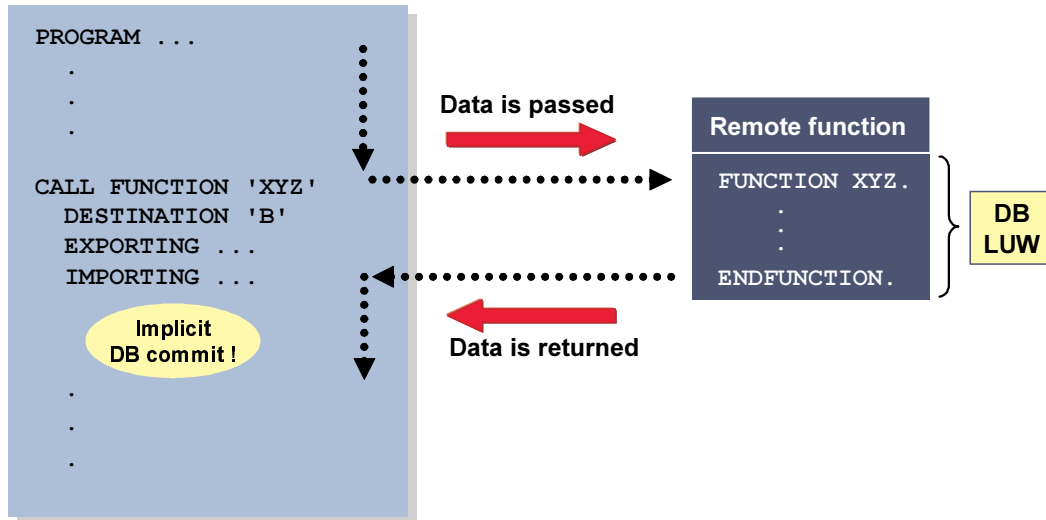
© SAP AG 2002



- In this topic you will learn the techniques of synchronous RFC.

## Synchronous RFC (Process Flow)

SAP

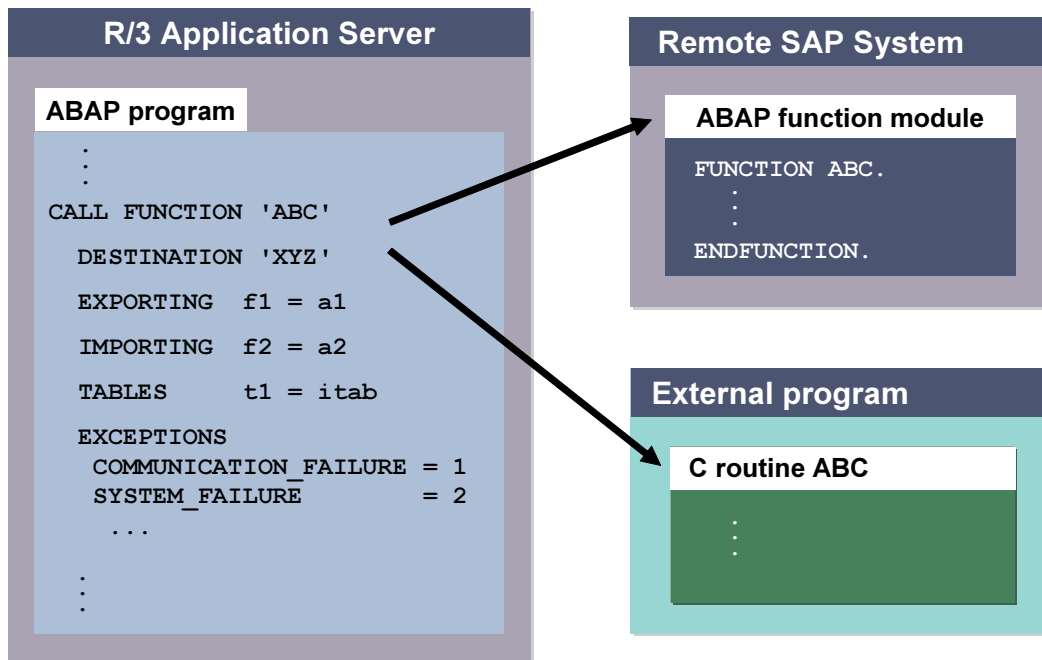


© SAP AG 2002

- With synchronous RFCs, processing is stalled in the calling program until the called remote function is processed and its output is returned. Processing in the calling program continues after the call.

## Synchronous RFC (Syntax)

SAP

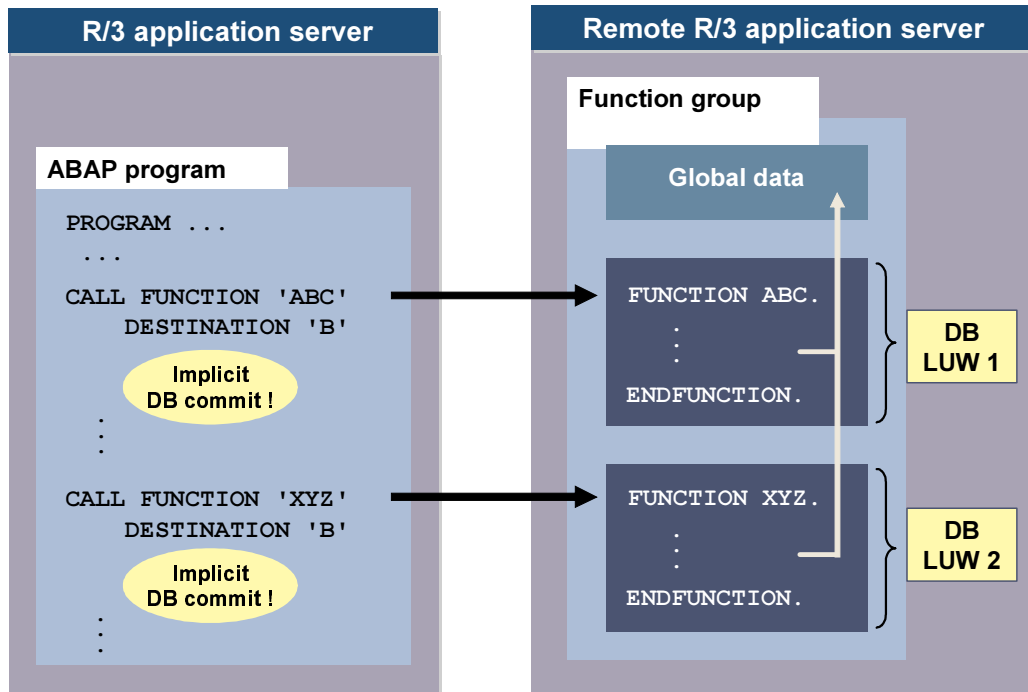


© SAP AG 2002

- You can call the remote ABAP function modules or even C routines in external server programs using the command `CALL FUNCTION ... DESTINATION`.
- When you call a function in this way, always include handling for the standard exceptions `COMMUNICATION_FAILURE` and `SYSTEM_FAILURE`.
- The exception `COMMUNICATION_FAILURE` is triggered by the system if the specified destination in the side information table `RFCDES` is not maintained, or if the connection to the remote system cannot be established.
- The exception `SYSTEM_FAILURE` is triggered if the function module or C routine that you want to start in the remote system is not available.

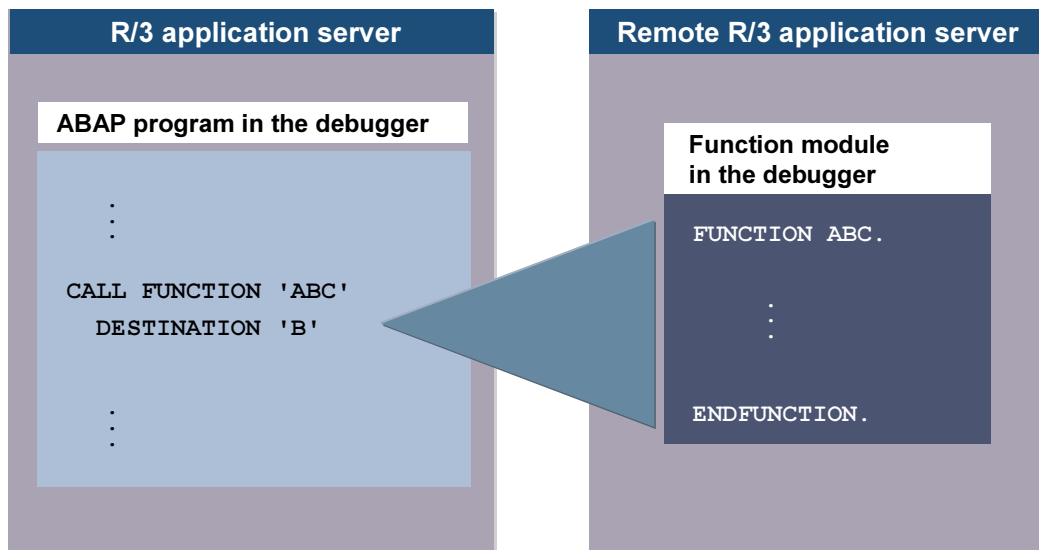
## Logical Units and Remote Program Context

SAP



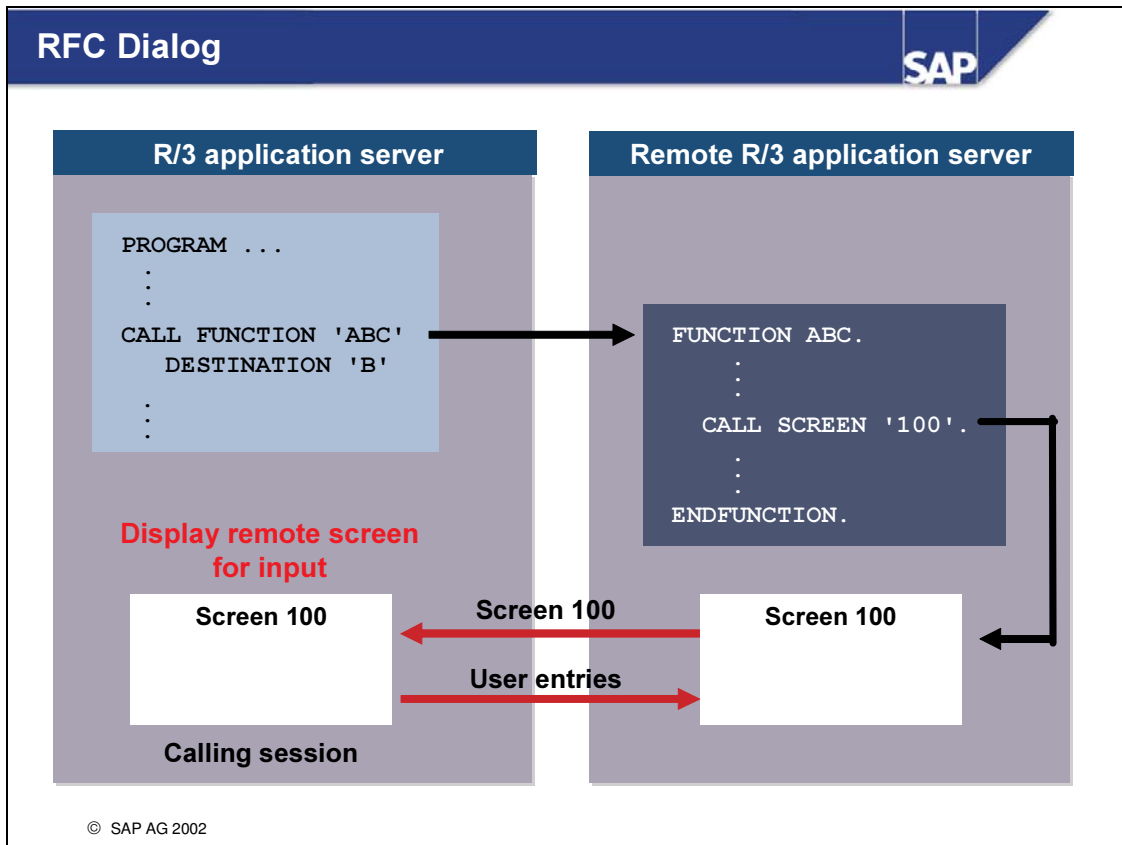
© SAP AG 2002

- The connection to a remote destination remains intact for as long as the context of the calling program remains active. The function groups addressed in the remote destination remain active for as long as the calling program itself remains active (this is the same as with local calls). This means that if you call two function modules from the same function group one after the other, they can both access the same global data of the function group.
- Each function module called using synchronous RFC forms its own logical unit of work (LUW).



© SAP AG 2002

- You can debug function modules called remotely in R/3 - R/3 connections.
- In this way, you can easily access, step by step, the remote function module for processing through the debugger of your calling program
- Also, you can have a breakpoint set automatically before each RFC statement in the debugger of your calling program. From the Debugger menu, choose:  
*Breakpoints → Breakpoint for statement → Enter RFC*
- During remote debugging of a synchronous RFC, the remote logon user must be type *dialog* and must have debug authorization. If this is not the case, the remote function module is executed in one step (background processing).



- If a remotely-called function module uses dialogs (for example, CALL SCREEN, CALL TRANSACTION, or list display), the screens concerned are displayed with all the functions in the session of the caller.
- Here, the remote logon must be type *dialog*. Otherwise, the RFC exception SYSTEM\_FAILURE will be triggered.
- Remember that these RFC dialogs in background processing cause program termination with the exception SYSTEM\_FAILURE.





- **With synchronous RFC, the calling program waits until the called function module has been processed.**
- **In a remote function call between two R/3 Systems, you can debug function modules that are running remotely.**
- **In an RFC between R/3 Systems, all remote dialogs are displayed on the calling program side.**

© SAP AG 2002

## Asynchronous RFC



Overview

RFC Destinations

Synchronous RFC



Asynchronous RFC

Transactional RFC

Queued RFC

RFC with External Programs

Use of RFC Types and RFC Authorizations

© SAP AG 2002

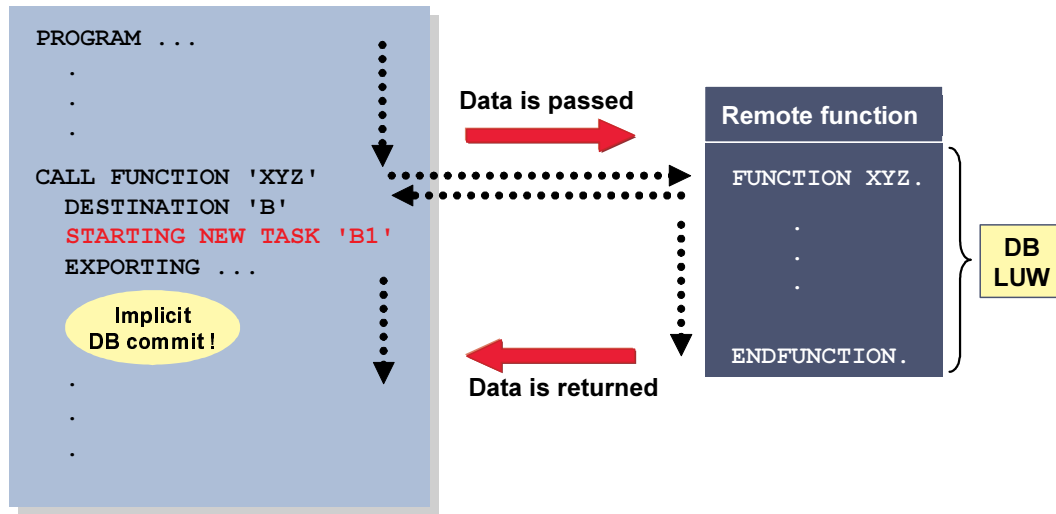


- You will learn how to call function modules asynchronously in order to implement parallel processing.
- You will be able to explain the internal resource management techniques used in this context.

© SAP AG 2002

## Asynchronous RFC (Process Flow)

SAP

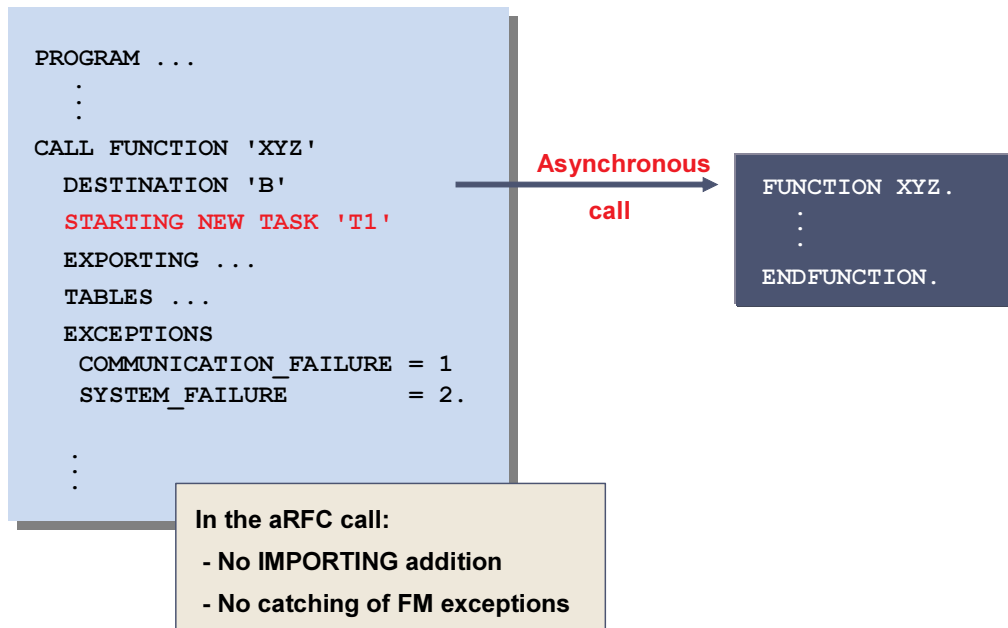


© SAP AG 2002

- In an asynchronous remote function call, the called remote function is started and processing continues immediately afterwards in the calling program. Remote function processing itself is separate from the calling program. The output of the function can be received later in the program.
- Asynchronous RFCs are intended for parallel processing of processes.

## Asynchronous RFC (Basic Syntax)

SAP



© SAP AG 2002

- With the `STARTING NEW TASK <task name>` addition, you can call a remote function module asynchronously. You can use any task name.
- The function module that you call is executed in its own work process.
- You can also use aRFC in background processing. Note, however, that in this context, too, the RFC calls are each executed in a work process.
- In the side information table RFCDES, you can set the number of aRFC calls for each destination using the aRFC options. After these aRFC calls, an automatic load check is performed on the target server. If resource bottlenecks are detected, the system waits for a short time for the next aRFC call meant for the same remote system, and the client program is rolled out of its work process. The client program can then receive results from previous aRFC calls.
- During the call, you may not specify an `IMPORTING` addition since the call does not wait for the end of the function module. Here, for the same reason, you can handle only the two system exceptions `COMMUNICATION_FAILURE` and `SYSTEM_FAILURE`. The function module output must be received and the function module-specific exceptions must be handled later on in a different place.
- Receiving the function module output and handling the function module-specific exceptions are optional tasks, however.

## aRFC with Confirmation (1)

SAP

```

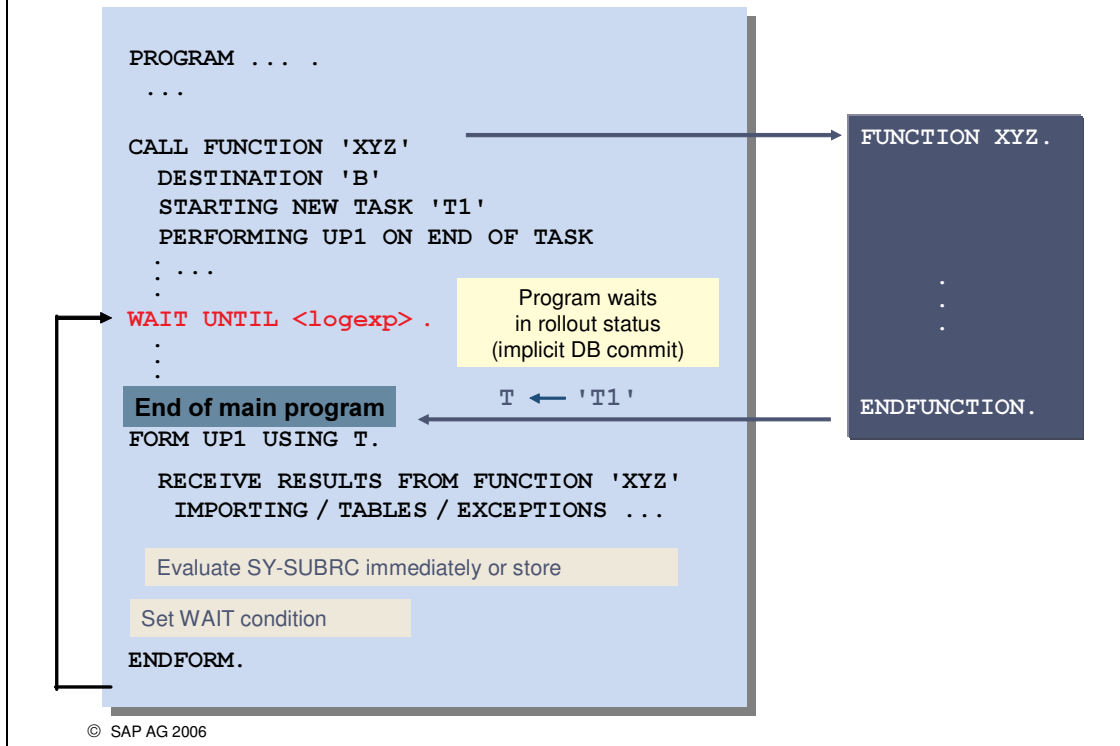
PROGRAM ...
...
CALL FUNCTION 'XYZ'
  DESTINATION 'B'
  STARTING NEW TASK 'T1'
  PERFORMING UP1 ON END OF TASK
  EXPORTING ...
  TABLES ...
  EXCEPTIONS
    COMMUNICATION_FAILURE = 1
    SYSTEM_FAILURE        = 2.
...
FORM UP1 USING T.
  RECEIVE RESULTS FROM FUNCTION
  'XYZ'
  IMPORTING ...
  TABLES ...
  EXCEPTIONS
    COMMUNICATION_FAILURE = 1
    SYSTEM_FAILURE        = 2
    FLIGHT_NOT_FOUND      = 3.
...
ENDFORM.
  
```

```

FUNCTION XYZ.
.
.
.
ENDFUNCTION.
  
```

© SAP AG 2006

- A program can receive output (parameters and exceptions) from a function module that is running asynchronously.
- To implement this, use the addition **"PERFORMING <subprogram> ON END OF TASK"** for the call. The subprogram you specify must exist in your program and the output of the function module should be received through the command **"RECEIVE RESULTS FROM FUNCTION <fbname> ..."**. When the function module ends, the subprogram is called automatically.
- The subprogram must also have a formal parameter with any name you choose. When the subroutine is called, it receives the accompanying task name. This parameter lets you see which aRFC call just ended, and you can receive the relevant output using the **"RECEIVE RESULTS FROM FUNCTION"** command.
- The subroutine cannot contain any statements that interrupt the program execution (such as CALL SCREEN, SUBMIT, COMMIT WORK, WAIT, RFCs, W or I messages). WRITE statements in this subroutine, which is specially defined for aRFC, do not work.
- The subroutine can only be executed automatically if the calling program is in rollout status. -> "WAIT UNTIL" statements (see next slide).
- The addition **KEEPING TASK** with the **RECEIVE** statement causes the function group context that was loaded remotely to wait until the calling program has ended. This lets you use the old remote context with later aRFC calls under the same task name.



- The language element WAIT UNTIL with the addition PERFORMING is only useful with aRFC, and otherwise has no effects.
- At the execution of the "WAIT UNTIL" statement, the specified condition is checked. If the condition is met, the program continues directly after the "WAIT UNTIL" statement. Otherwise, the program waits in rollout status for the output delivery of the aRFC.??? If an aRFC now returns its output, the form routine specified when it was called is executed and then it branches back to the "WAIT UNTIL" statement. This check/wait process repeats until the "WAIT UNTIL" condition is met or until no more open aRFC calls are left.
- Note that the "WAIT UNTIL" statement sets the SY-SUBRC. Therefore, store the SY-SUBRC value (set in the form routine by exceptions-handling in RECEIVE RESULTS) in its own global variable before leaving the form routine if you need this value again later (after WAIT UNTIL).

## Example: Waiting for multiple aRFCs

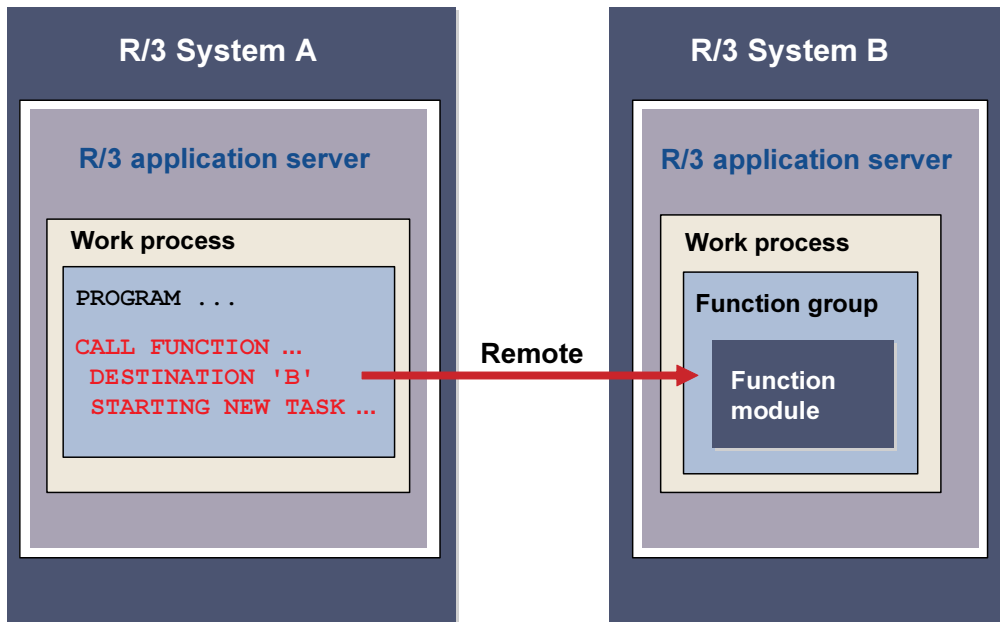


```
PROGRAM ...  
...  
  
CALL FUNCTION 'X1'  
  DESTINATION 'B'  
  STARTING NEW TASK 'T1'  
  PERFORMING UP1 ON END OF TASK  
  ...  
  
CALL FUNCTION 'X2'  
  DESTINATION 'B'  
  STARTING NEW TASK 'T2'  
  PERFORMING UP1 ON END OF TASK  
  ...  
:  
:  
:  
  
WAIT UNTIL NOT flag1 IS INITIAL AND NOT flag2 IS INITIAL.  
:  
:  
:  
  
FORM UP1 USING T.  
  CASE T.  
    WHEN 'T1'. <RECEIVE RESULTS + Output processing of X1>.  flag1 = 'X'.  
    WHEN 'T2'. <RECEIVE RESULTS + Output processing of X2>.  flag2 = 'X'.  
  ENDCASE.  
ENDFORM.
```

© SAP AG 2006

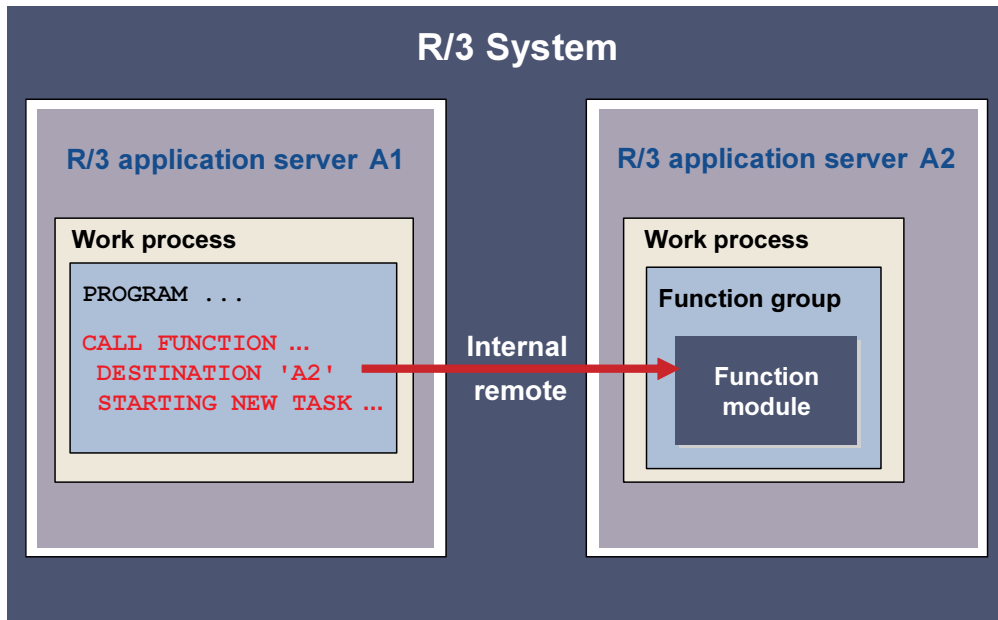
- In this program example, there are two aRFCs and, when the aRFCs are confirmed, the subroutine UP1 is always executed.
- This subroutine uses parameter T (which is automatically transferred by the system at the call) to separately decide which function module just returned its output. Then the corresponding RECEIVE RESULTS statement is used to receive (and process) the output.???
- After processing an aRFC output, you must adjust the WAIT UNTIL condition accordingly. (here: set flag1 or flag2)





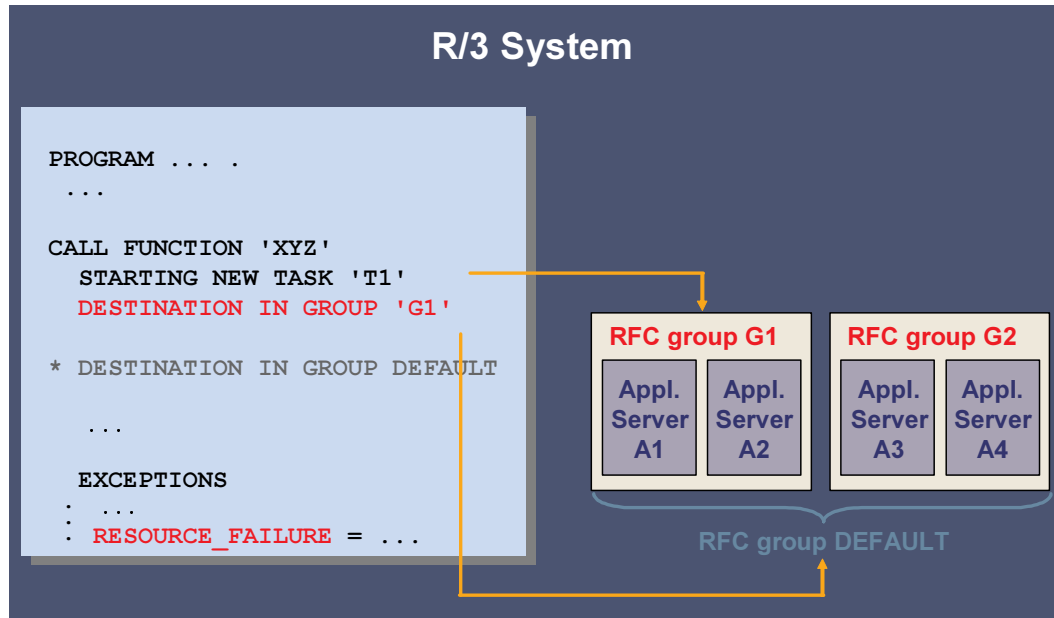
© SAP AG 2002

- aRFC is particularly suited for implementing parallel processing in several R/3 Systems.



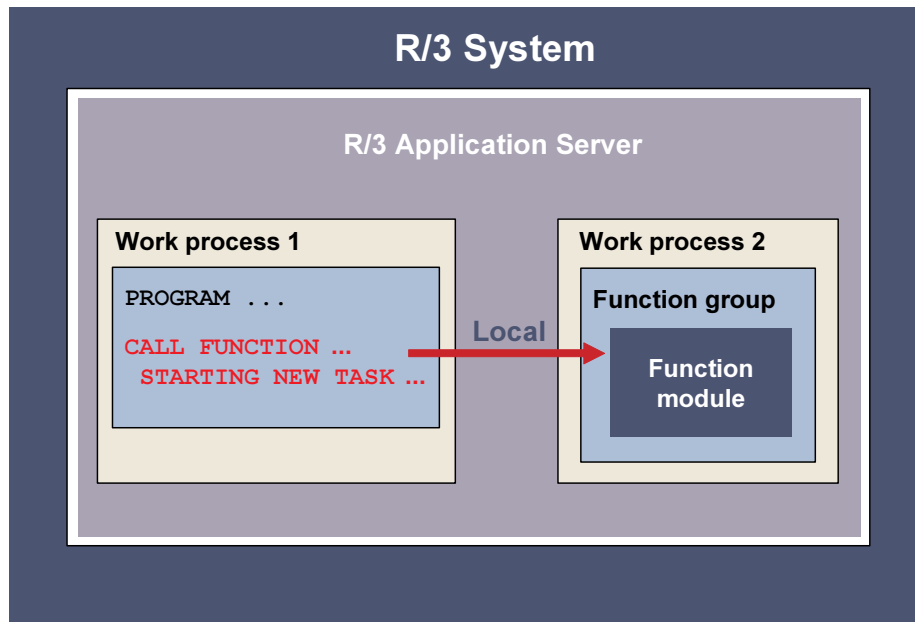
© SAP AG 2002

- You can also use aRFC within the same SAP R/3 System, for example, to move some of the processing load to an application server specially used for this.
- Enter the RFC destination that refers to the corresponding application server. (You can find this under *Internal connections* in transaction SM59.)



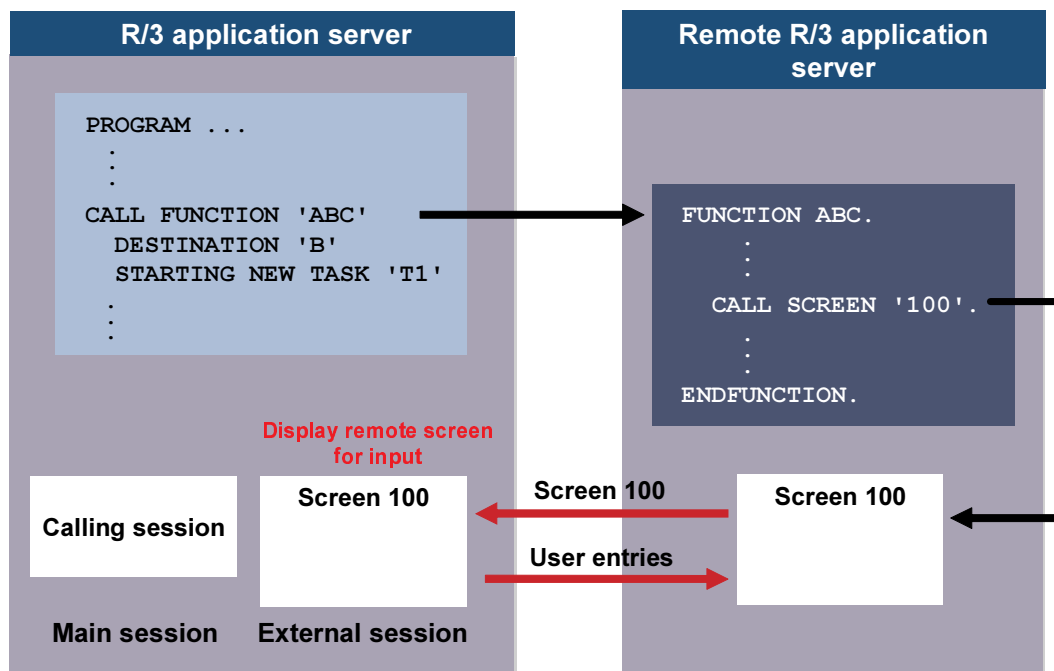
© SAP AG 2006

- You can use transaction SM59 to divide the application servers for an SAP R/3 system into different RFC groups.
- At a function module call in this R/3 system you can use the addition **DESTINATION IN GROUP** <RFC group> to specify one of the defined RFC groups. The server in this group that has the lowest load is then selected to execute the function module.  
You can also specify the 'DEFAULT' RFC group. The server is then selected from amongst all the application servers of the R/3 system.
- When assigning an application server to an RFC group, you can specify conditions under which the application server is to be considered as "available for aRFCs within the same R/3 system". If none of the servers belonging to the specified RFC group are available for this aRFC, the system triggers the exception **RESOURCE\_FAILURE**.
- Notes about the addition **"DESTINATION IN GROUP"**-
  - It is generally to be considered different from the previous addition **DESTINATION**,
  - In contrast to the previous addition **DESTINATION** you must specify it **after "STARTING NEW TASK"** and
  - **can only be used in the same R/3 system**, that is, it cannot be used together with the previous addition **DESTINATION**.



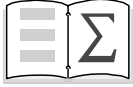
© SAP AG 2002

- You can also use aRFC locally within the **same application server** to implement parallel processing in several work processes.
- Here you do not need to specify a destination.
- Note, however, that several work processes will be occupied by your program at the same time.



© SAP AG 2002

- If you have called a function module asynchronously and it contains dialogs, the relevant screens are displayed in an additional (new external session) window on the calling side.
- Here, the remote logon use must be type *dialog*. Otherwise, the RFC exception `SYSTEM_FAILURE` will be triggered.
- As with synchronous RFC, remote debugging is also possible for asynchronous calls. The debugger screen then appears as a new external mode. Here, too, the remote logon use must be type *dialog* and must also have debugging authorization.
- When you write the program, limit the number of external sessions for each dialog user to six. If the calling dialog user already has six external sessions open, the exception `SYSTEM_FAILURE` is triggered.
- Remember, too, that the aRFC dialog described here will not function if you have an SAP router set in between. The reason is that, for each aRFC dialog, a return link to the source system is specially set up, but this is not possible if an SAP router is implemented.



- aRFC enables you to implement parallel processing.
- You can use aRFC system-wide, for multiple application servers, or locally on the same application server.
- Function modules running asynchronously can return data to the calling program. To do this, the calling program must remain active until the function module output has been returned.

© SAP AG 2002

## Transactional RFC



Overview

RFC Destinations

Synchronous RFC

Asynchronous RFC



Transactional RFC

Queued RFC

RFC with External Programs

Use of RFC Types and RFC Authorizations

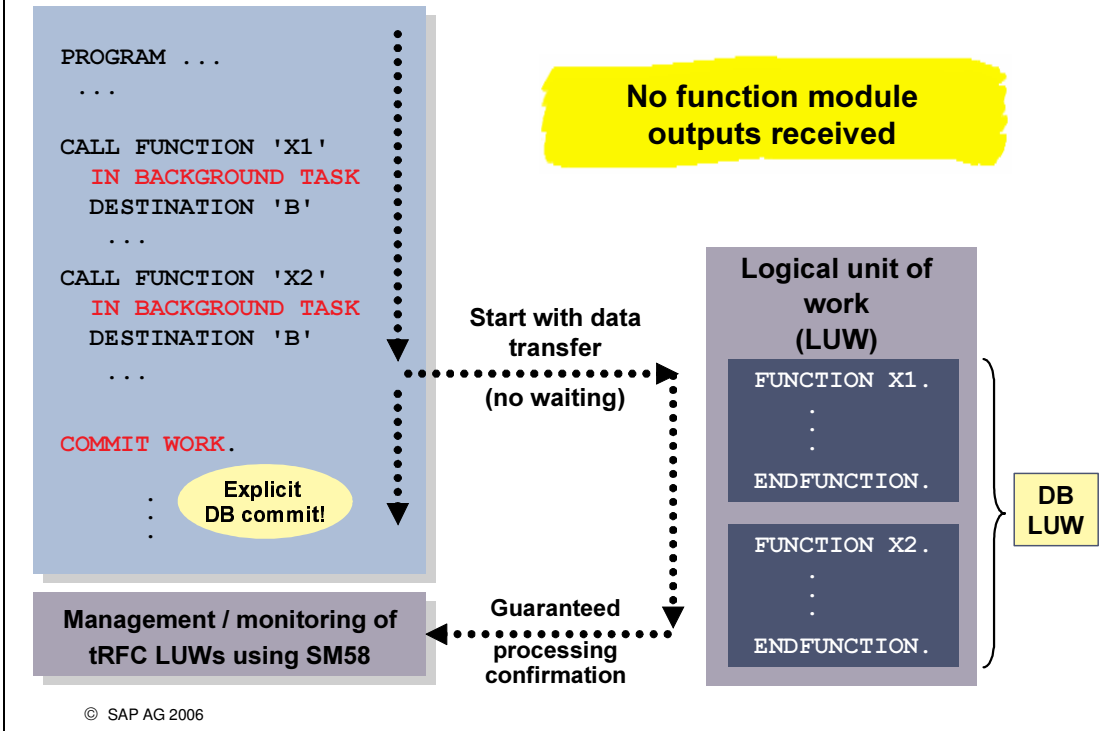
© SAP AG 2002



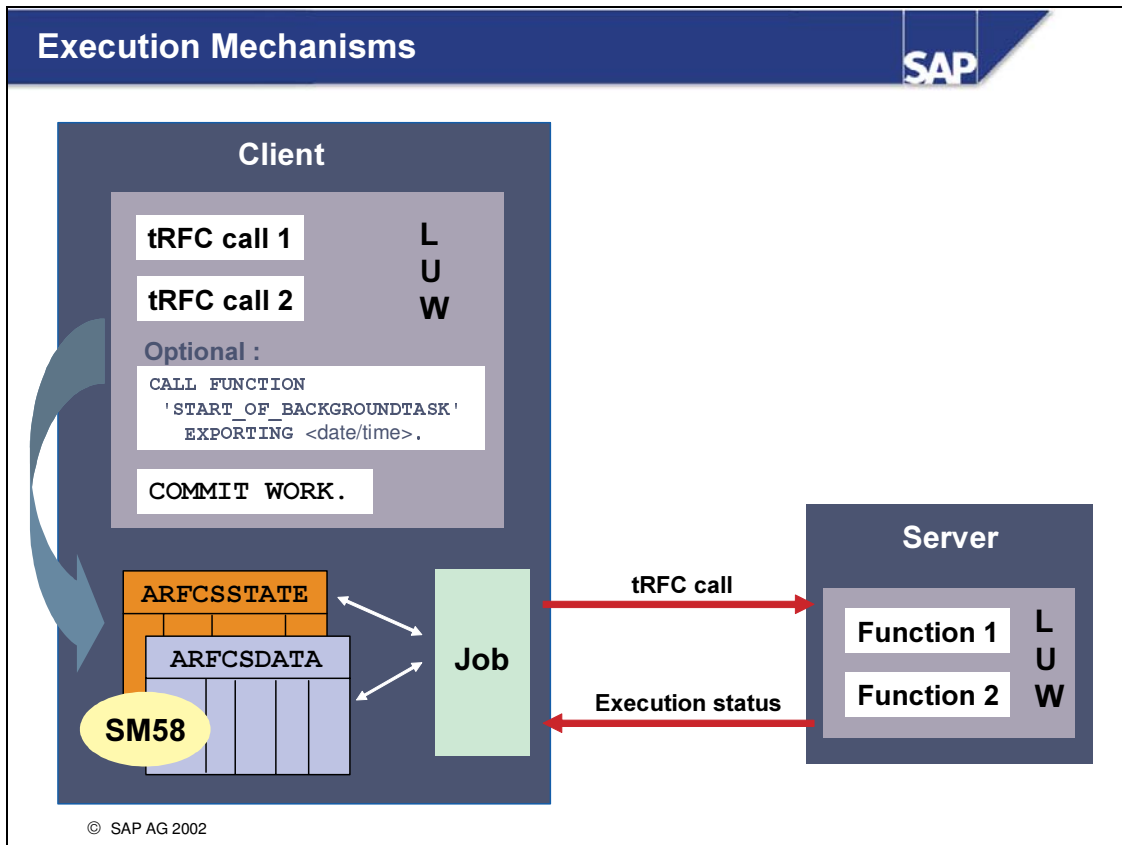
- How to combine several remote function modules (for each destination) into a database LUW with automatic rollback and commit mechanisms using tRFC.
- How to program tRFC in ABAP.
- To be aware that tRFC guarantees return confirmation of the remote processing status and to be able to manage tRFC LUWs using transaction SM58.

© SAP AG 2002





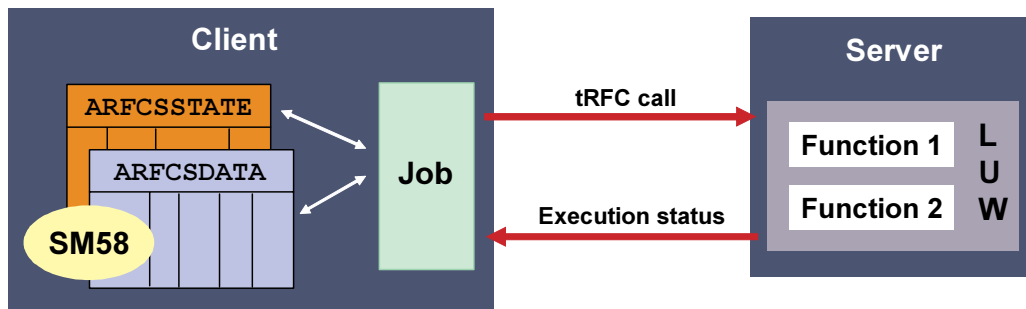
- Whereas with synchronous and asynchronous RFCs each call makes up a single logical unit of work (LUW) in the remote system, for transactional RFCs you can group several remote functions in one LUW (with automatic rollback mechanism if there is an error).
- You can implement a tRFC with the addition "IN BACKGROUND TASK", which you must place **before** the DESTINATION entry. You can use "COMMIT WORK" to bundle all the previously executed RFCs in one LUW.
- tRFCs are also called asynchronously. Unlike aRFC, the output from the called function module cannot be received.
  - > No IMPORTING ... / PERFORMING ... ON END OF TASK when called;
  - > No RECEIVE RESULTS FROM FUNCTION ...
- In the source system, you can use the administration transaction SM58 to display and modify tRFC LUWs.



- tRFC calls are first stored in the local tRFC tables ARFCSSTATE and ARFCSDATA. The execution status of the LUWs is logged in table ARFCSSTATE, while ARFCSDATA contains the input data for the tRFCs.
- If you do not want to execute a remote LUW immediately, but rather trigger it at a later time, call the function module **START\_OF\_BACKGROUNDTASK** **before** the **COMMIT WORK** statement locally. Here, you must enter the date and time of execution.
- The **COMMIT WORK** statement automatically schedules an immediate job or a job set for a later start time to remotely call the LUW. During job execution, the relevant data is read from the tRFC tables, and the corresponding tRFCs are transmitted. If the remote LUW is executed successfully, the accompanying entries are deleted from the tRFC tables. If an error occurs, an automatic repeat mechanism or rollback mechanism is activated.
- If the update is also triggered locally by the **COMMIT WORK** statement, the tRFCs are executed only when the local update is successfully completed.

## Automatic Mechanisms: Successful or Error

SAP



- **If successful:** Relevant entries deleted from tRFC tables
- **If partner not reachable:** Automatic job rescheduling  
(You can set the number of repeat attempts and interval length in the destination  
→ tRFC options)
- **If error with termination in one of remote functions modules:\*)**  
**Automatic rollback in the target system + error note in ARFCSSTATE**  
(No deletion + no automatic job rescheduling in source system)

\*) A message / RAISE <exception>

© SAP AG 2002

- If no connection can be made to the partner, this is logged in the tRFC status table ARFCSSTATE (viewable by calling transaction SM58), and the job is rescheduled. In the destination, you can set the number of times the system repeats the effort to connect, and the time intervals, using the tRFC options. The default setting is 30 times maximum, with a 15-minute interval.
- If the program terminates with an A/X-message (MESSAGE A/X...) or triggers an exception (RAISE...) after a tRFC-LUW has been successfully executed in the partner system in one of the function modules, the following will happen:
  - All the changes executed in the remote system in the current LUW are rolled back.
  - These changes are logged in the source system of the remote program termination in the tRFC status table ARFCSSTATE (can be viewed using SM58).

The entries relevant to the LUW remain in the tRFC tables, and the execution job is not rescheduled. In this case, you can find the remote error using transaction SM58, and you have to correct the problem in the remote system. Afterwards, you must trigger the remote execution again in transaction SM58 (function *Execute LUW*).

- If you want to cancel and roll back this remote execution from a remote function module of a tRFC LUW, but at the same time reschedule the job in the source system (for example, because a master record that is to be processed is locked and the LUW must be executed again), call the function module `RESTART_OF_BACKGROUNDTASK` in the remote function call module instead of `MESSAGE A...` or `RAISE...`

## Client program

```
PROGRAM ...
...
CALL FUNCTION 'X1'
  IN BACKGROUND TASK
  DESTINATION 'B'

CALL FUNCTION 'X2'
  IN BACKGROUND TASK
  DESTINATION 'B'

CALL FUNCTION 'X3'
  IN BACKGROUND TASK
  DESTINATION 'C'

CALL FUNCTION 'X4'
  IN BACKGROUND TASK
  DESTINATION 'C'
...
COMMIT WORK.
```

### Destination B

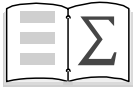
|    |                  |
|----|------------------|
| X1 | L<br>U<br>W<br>1 |
| X2 |                  |

### Destination C

|    |                  |
|----|------------------|
| X3 | L<br>U<br>W<br>2 |
| X4 |                  |

© SAP AG 2002


- If tRFCs are transmitted with different destinations before the `COMMIT WORK`, they are grouped into different LUWs according to their destination.
- A tRFC call with the `AS SEPARATE UNIT` addition is always processed as a separate LUW independently of the other tRFCs.
- Each tRFC-LUW is assigned a unique transaction ID (viewable using transaction SM58). The function module `ID_OF_BACKGROUNDTASK` (called before the `COMMIT WORK`) returns this ID. You can determine the LUW execution status using the function module `STATUS_OF_BACKGROUNDTASK` under the transaction ID (which is called after the `COMMIT WORK`).
- If you also called the update function modules locally before the `COMMIT WORK`, tRFC calls will be processed only after the local update function modules have been processed successfully.
- You can also call transactional C functions. However, you must use the rollback mechanism in the external server program (see RFC-SDK documentation and example programs).



- You can use tRFC to process several function modules remotely in the same LUW with the automatic rollback mechanism.
- tRFC allows you to ensure that the function modules you call are executed even if the target system is unavailable when the call is made.
- With tRFCs, you are guaranteed a confirmation of the remote processing status.
- You can use transaction SM58 to display and process (delete and execute) incorrect or non-executed tRFC-LUWs.

© SAP AG 2002

## Queued RFC



- Overview
- RFC Destinations
- Synchronous RFC
- Asynchronous RFC
- Transactional RFC
- Queued RFC
- RFC with External Programs
- Use of RFC Types and RFC Authorizations

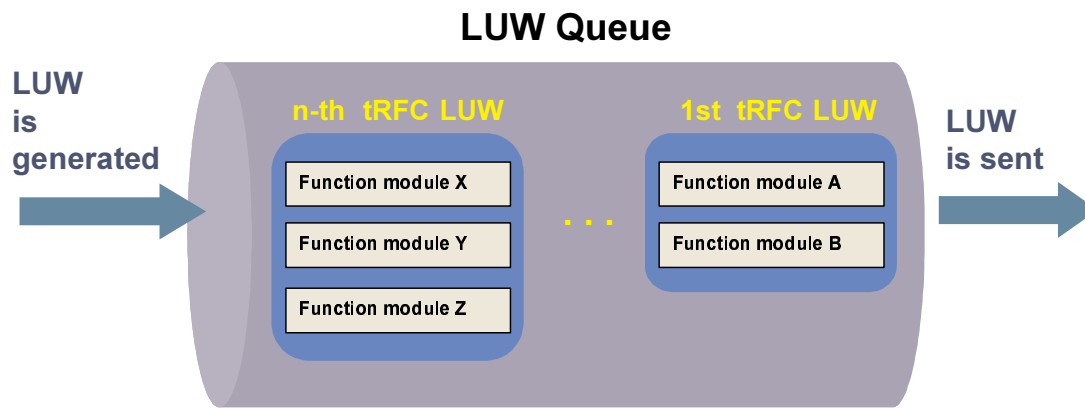
© SAP AG 2002



- You know that you can serialize tRFC-LUWs in a queue using queued RFC.
- You can explain the architecture and programming of queued RFC.
- You know how to manage RFC queues using transaction SMQ1.

© SAP AG 2002

### Serialization of tRFC-LUWs



© SAP AG 2002

- LUWs generated using tRFC are processed independently of each other. This means the order in which they are processed is not always the order in which they are generated.
- To ensure that tRFC-LUWs are processed in the same order as they were generated, use qRFC as an extension of tRFC.
- Queued RFC (qRFC) is available as of Release 4.6A, and can be used in SAP R/3-R/3 connections and in SAP R/3 external connections.



```

PROGRAM ...
...
CALL FUNCTION 'TRFC_SET_QUEUE_NAME' EXPORTING qname = 'Q1'.
CALL FUNCTION 'A'
  IN BACKGROUND TASK
  DESTINATION 'XYZ' ...

CALL FUNCTION 'TRFC_SET_QUEUE_NAME' EXPORTING qname = 'Q1'.
CALL FUNCTION 'B'
  IN BACKGROUND TASK
  DESTINATION 'XYZ' ...

COMMIT WORK.

CALL FUNCTION 'TRFC_SET_QUEUE_NAME' EXPORTING qname = 'Q1'.
CALL FUNCTION 'C'
  IN BACKGROUND TASK
  DESTINATION 'XYZ' ...

CALL FUNCTION 'TRFC_SET_QUEUE_NAME' EXPORTING qname = 'Q1'.
CALL FUNCTION 'D'
  IN BACKGROUND TASK
  DESTINATION 'XYZ' ...

COMMIT WORK.
...

```

1st place  
tRFC-LUW in  
queue Q1

2nd place  
tRFC-LUW in  
queue Q1

© SAP AG 2002

- To place tRFC-LUWs in a first-in-first-out (FIFO) queue, you must specify the queue you want to use through the function module TRFC\_SET\_QUEUE\_NAME **before every single tRFC call**.
- You can choose the queue name you need to specify, and it can have up to 24 characters. The queue name may **not** start with an empty character or the percent (%) symbol, and may not contain an asterisk (\*).

## Queue data

### TRFCQOUT

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## tRFC data

### ARFCSSTATE

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### ARFCSDATA

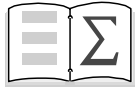
|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Do not maintain using SM58!**

**Maintain using Transaction SMQ1**

© SAP AG 2002

- The tRFC LUWs belonging to a queue are stored in the tRFC tables, ARFCSSTATE and ARFCSDATA. However, you cannot maintain or display these LUWs using Transaction SM58. You **must** use the queue administration transaction SMQ1.
- The queue data (queue attributes + pointers to the relevant tRFC-LUWs in the tRFC tables) is stored in the TRFCQOUT table, which you can maintain using transaction SMQ1.  
The following functions are available:
  - List queues
  - Display queues with a list of accompanying LUWs and function modules including input data
  - Start or stop transmission of queues
  - Delete queues
- If an LUW error occurs in the remote system, the queue is rolled back there. The relevant LUW and all of the other LUWs in the queue remain in the queue and in the tRFC tables. After correcting the error, you can use transaction SMQ1 to transmit the queue again.



- With qRFC, you can ensure that tRFC LUWs are processed in the order they were generated (serialization).
- This is implemented using a queue for tRFC LUWs.
- The queue attributes are stored in the queue table TRFCQOUT and the tRFC input data in the tRFC tables ARFCSSTATE and ARFCSDATA. You can maintain and display this information using transaction SMQ1.

© SAP AG 2002

## RFC with External Programs



Overview

RFC Destinations

Synchronous RFC

Asynchronous RFC

Transactional RFC

Queued RFC



RFC with External Programs

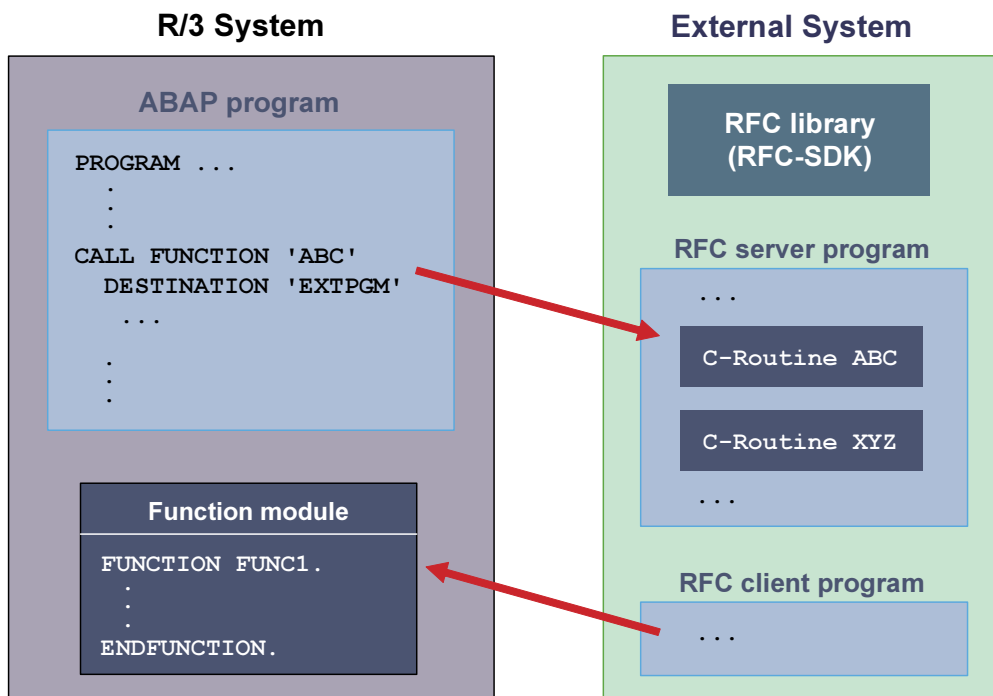
Use of RFC Types and RFC Authorizations

© SAP AG 2002



- You will learn how external RFC clients and RFC servers are structured in C.

© SAP AG 2002



© SAP AG 2002

- SAP delivers the RFC Library for all standard external platforms that contain RFC calls for C programs. After installing the RFC Library on your external platform, you can use the RFC client or RFC server programs with RFC calls.
- In addition to the RFC Library, you also have several sample programs in C source together with documentation supplied with your system.

**saprfc.ini (text file)**

```
DEST=T71_AS1
TYPE=A
ASHOST=iwdf5071
SYSNR=00

DEST=T71_LB
TYPE=B
R3NAME=T71
MSHOST=iwdf5071
GROUP=SPACE

DEST=EXTPGM
TYPE=E
GWHOST=hs0815
GWSERV=sapgw17
TPHOST=hs0815
TPNAME=D:\abc\srfcserv

.
```

© SAP AG 2002

- You maintain RFC destinations for C programs in the text file saprfc.ini .
- In addition to the RFC Library, there is also an saprfc.ini with sample entries and documentation for maintaining the different RFC destination types supplied with your system.
- For details, refer to the online documentation.

```

RfcOpen (...);
...
RfcCall (...);
...
RfcReceive (...);
...
RfcCall (...);
...
RfcReceive (...);
...
RfcClose (...);
...
    
```

} RfcCallReceive (...);

© SAP AG 2002

- This slide shows you the structure of an external RFC client.
- The connection to the SAP R/3 System is made using the RFC call *RfcOpen* with the destination and logon data.
- *RfcCall* calls an R/3 function module. *RfcReceive* receives the output of the function module. This enables you to use an asynchronous call. If you want to use a synchronous call, use *RfcCallReceive* instead of these two RFC calls.
- To close the RFC connection to the SAP R/3 System, use *RfcClose*.
- You will find the semantics and syntax for the RFC calls listed in the RFC library in the online documentation.
- The *sapinfo* program provides an example of an RFC client in C source that is delivered with the RFC Library. This program calls the function module RFC\_SYSTEM\_INFO in a remote R/3 System to get R/3 system data from it.



```
main(...)
{
    ....

    RfcAccept(...)

    RfcInstallFunction(abc,...)
    RfcInstallFunction(xyz,...)

    RfcDispatch(...)

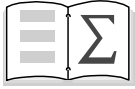
    RfcClose(...)
}

static RFC_RC abc(...)
{
    ...
    RfcGetData(...)
    Data processing
    RfcSendData(...)
}

static RFC_RC xyz(...)
{
    ...
}
```

© SAP AG 2002

- This slide shows you the structure of an external RFC server.
- The call ***RfcAccept*** accepts the incoming RFC connection.
- All C routines for the RFC server program that should be called remotely must be registered as being remote-enabled using the call ***RfcInstallFunction***.
- The call ***RfcDispatch*** sends the incoming call to the corresponding C routine.
- The call ***RfcGetData*** enables the C routine to receive the input sent from the calling program.
- The call ***RfcSendData***, returns the result to the calling program.
- The call ***RfcClose*** ends the RFC connection.
- You will find the semantics and syntax for the RFC calls listed in the RFC Library in the online documentation.
- The *sapinfo* program provides an example of an RFC client in C source that is delivered with the RFC library. This program contains various, interesting remote-enabled routines and can, for example, be called in the standard system with the supplied ABAP program RSRFCDOC through a corresponding RFC destination.



- The RFC SDK must be installed for RFC on the external platform.
- The text file saprfc.ini contains RFC destinations for C clients.
- You can call R/3 function modules using an external RFC client program.
- All remote-enabled, registered C routines of an RFC server program can be called from within ABAP programs.

© SAP AG 2002

## Use of RFC Types / RFC Authorizations



Overview

RFC Destinations

Synchronous RFC

Asynchronous RFC

Transactional RFC

Queued RFC

RFC with External Programs



Use of RFC Types and RFC Authorizations

© SAP AG 2002

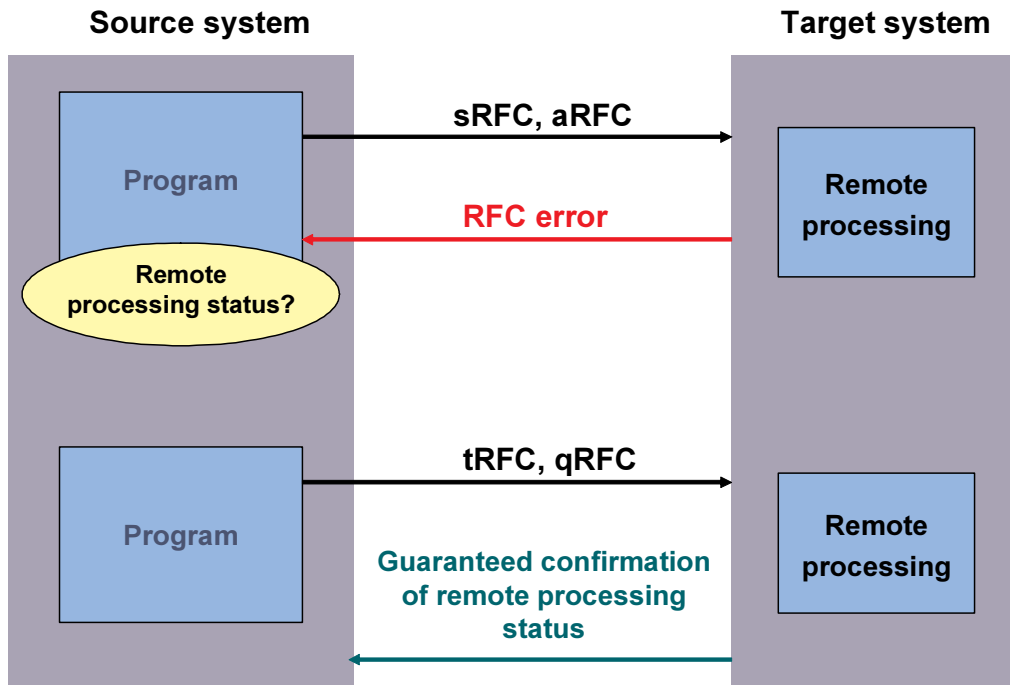


- You will learn which RFC types are to be used for which purpose.
- You can explain which authorization checks exist for RFC.
- You know which authorizations the remote user needs for the RFC login to call the remote function module successfully.
- You can define trust relationships between SAP R/3 Systems, and also explain - for existing trust relationships - how users in the source system can use the released user IDs in the target system without needing a password for RFC login.
- You know how to release user IDs in the target system for specific user IDs in the source system with an existing trust relationship.
- You know the RFC utilities that are useful.

© SAP AG 2002

## Processing Status for Communication Errors

SAP



© SAP AG 2002

- If a communication error occurs during synchronous or asynchronous RFC, the remote processing status is unknown. Therefore, these two RFC types are suitable for remote data retrieval where knowledge of the remote processing status is not of primary importance, even if a communication error occurs.
- Transaction or queued RFCs, on the other hand, always guarantee confirmation of the remote processing status (continued confirmation attempts until confirmation is successful). Therefore, these two RFC types are particularly suitable for remote data changes where knowledge of the remote processing status is important.

### Remote Data Retrieval:

- Synchronous RFC
- Asynchronous RFC

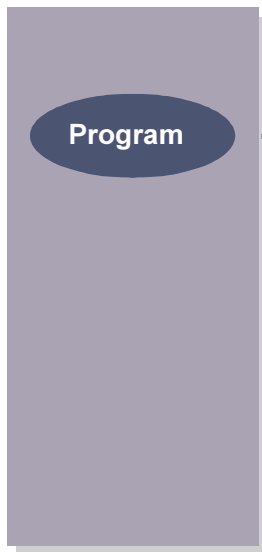
### Remote Data Changes:

- Transactional RFC
- Queued RFC

© SAP AG 2002

## Source System

(R/2, R/3, external)



RFC

RFC logon with  
remote user  
SMITH

## Target SAP R/3 System

User SMITH

- Authorization for RFC start  
of desired function module  
  
- Authorizations for actions  
on the function module

Function module

© SAP AG 2002

- The authorization object S\_RFC lets you define, for each RFC logon user, in which function groups the user is authorized to start RFC. This user can only start function modules in these function groups with RFC.
- This authorization check is active by default. However, you can deactivate this default setting using a profile parameter.
- Authorization checks for actions that call a function module remotely (for example, transaction authorization for CALL TRANSACTION) are compared against the user master record of the corresponding RFC logon user.

## The Authorization Object S\_RFC

SAP

**Authorization for RFC logon user  
to start a function module**

### Object S\_RFC

RFC\_NAME (Function group)

ACTVT (Activity)

RFC\_TYPE (RFC object type)

### Authorization ALL

RFC\_NAME : \*

ACTVT : 16

RFC\_TYPE : FUGR

### Authorization ABCD

RFC\_NAME : ABCD

ACTVT : 16

RFC\_TYPE : FUGR

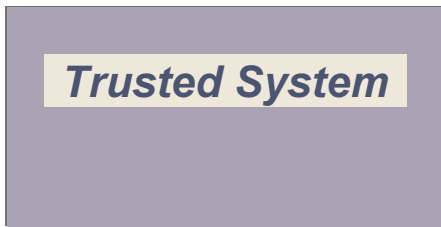
© SAP AG 2006

- The authorization object S\_RFC contains three fields:
  - **RFC\_NAME** : Can currently contain only names of function groups
  - **ACTVT** : Can currently accept only the value 16 (execute)
  - **RFC\_TYPE** : Can currently accept only the value FUGR (function group)
- The above examples show authorizations for the object S\_RFC :
  - If you use an RFC logon user with authorization "ALL", you can call all function modules.
  - .
  - However, if you use an RFC logon user that only has authorization "ABCD", you can only call function modules of function group ABCD.

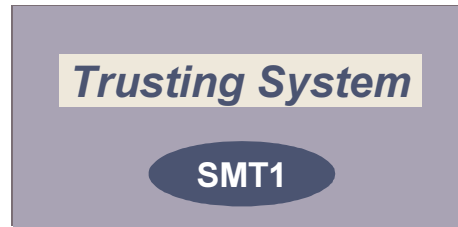


**Declaration of a trust relationship  
using SMT1 in the system that is to  
become the trusting system.**

**R/3 System A**

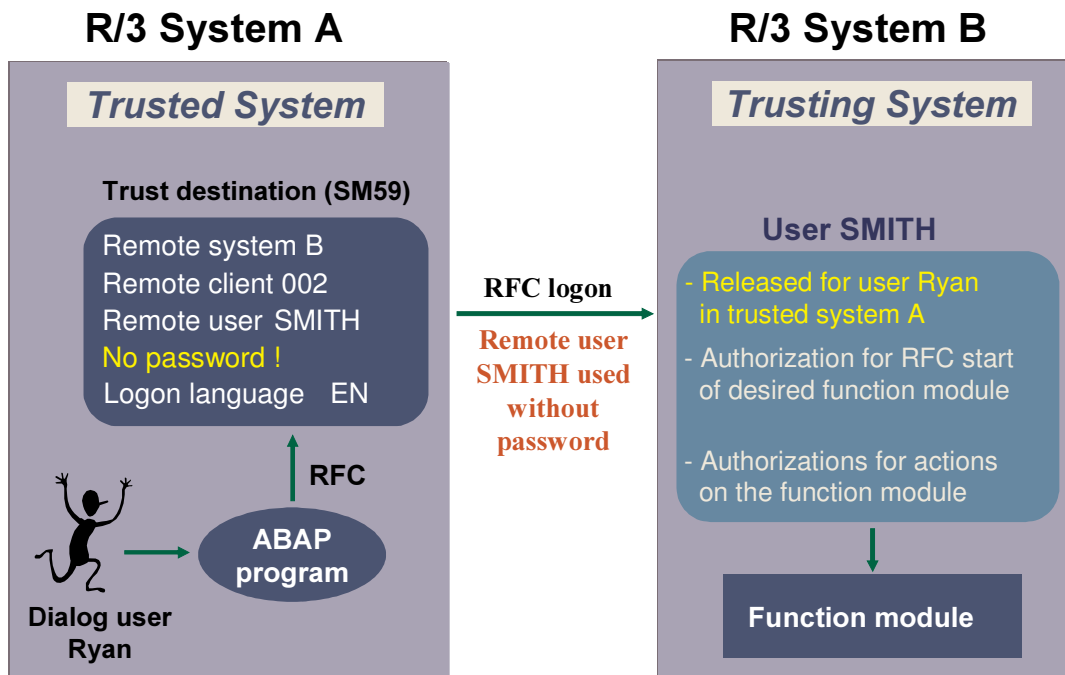


**R/3 System B**



© SAP AG 2002

- You can declare a trust relationship between two SAP R/3 Systems so that you can perform certain RFC logons in the trusting system without requiring a password for released users.
- You always declare a trust relationship in the system that is to become the trusting system, which means, it is in the RFC target system.
- You use transaction SMT1 to define a trust relationship.
- You can use transaction SMT2 in the trusted system to display which trusting systems are declared for the local system.



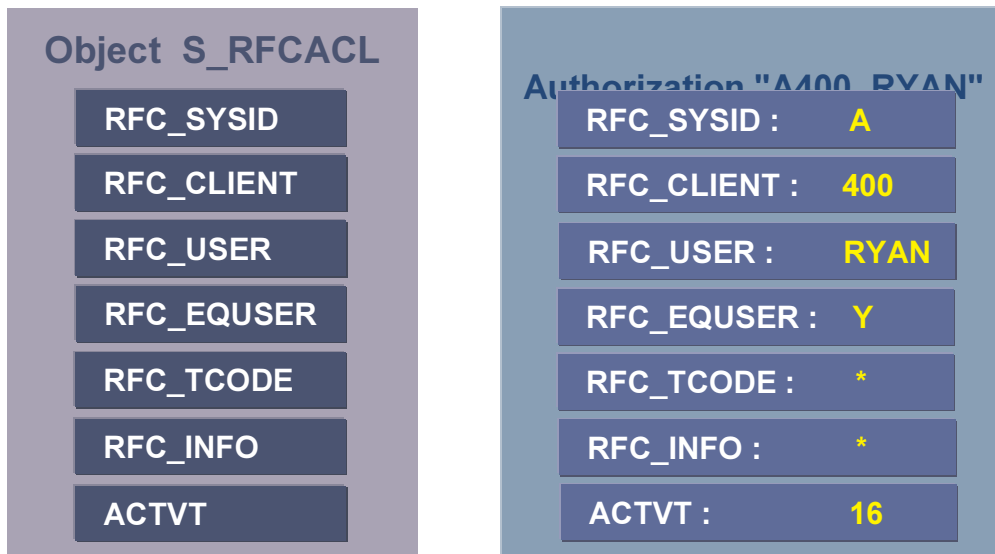
© SAP AG 2002

- A dialog user in the trusted system (RFC source system) can use a specially released, defined user ID for the RFC logon in the trusting system (RFC target system) without a password. To do this, the user requires a trust destination, created using transaction SM59 (*Trusted system* = 'Yes'), in which only the name of the remote user is entered, not the accompanying password.  
**Note:** You can also use regular destinations (with a password) with an existing trust relationship.
- To be able to use RFC logon without a password for a specific user ID in the trusted system, you need to release a user in the trusting system. You do this by giving an authorization for the authorization object S\_RFCACL to the user you want to release in the trusting system.
- For a detailed description of the definition and usage of trust relationships, refer to note number 128447.

## The Authorization Object S\_RFCACL

SAP

Release a user in the **Trusting System** by assigning authorization



© SAP AG 2006

- The authorization object **S\_RFCACL** contains seven fields :
  - **RFC\_SYSID** : Calling RFC source system (trusted system)
  - **RFC\_CLIENT** : Calling client
  - **RFC\_USER** : Calling user
  - **RFC\_EQUSER** : Indicator that shows if the user that has this authorization is released for the user of the same name in the RFC source system.
  - **RFC\_TCODE** : Calling transaction
  - **RFC\_INFO** : Additional information about the RFC source system (not currently used)
  - **ACTVT** : Can only have the value 16 (execute)
- Example:

If user **SMITH** in the trusting system has the authorization "A400\_RYAN", he is released for the user **RYAN** in the trusted system A. **RYAN** is allowed to use remote user **SMITH** for RFC logon without a password using any transaction from system A, client 400. **RFC\_EQUSER** = \* means that user **SMITH** is released for the user of the same name in trusted system A.

- **Short dumps caused by remote function modules:**  
    **ST22** in the remote system
- **Gateway monitor (displays all running remote connections that run on the current R/3 gateway):** **SMGW**
- **Traces :**
  - User-related RFC trace : **ST05**
  - Destination-related RFC trace : **SM59**
  - Gateway trace : **SMGW**
- **In the debugger, have a breakpoint set in front of each RFC:**  
    **Breakpoint menu in the debugger**
- **General tips for RFC:**     **i** -key in ABAP editor  
                                    → Enter ABAP term RFC

© SAP AG 2002

- You can analyze dumps caused by RFC function modules in the remote system using transaction ST22.
- The gateway monitor (transaction SMGW) shows all the remote connections that run through the gateway of the current application server.
- Using transaction ST05, you can switch on/off and display a user-related RFC trace for the current user.
- To display a destination-related trace, proceed as follows:
  - Set *trace* flag in the definition (SM59) of the required destination
  - Display trace using SM59 then choose RFC → Display trace
- To display the gateway trace, use transaction SMGW or choose from the menu as follows:  
*Goto → Trace → Gateway → Display file.*
- To have a breakpoint set automatically before each RFC statement in the debugger of your calling program, from the Debugger menu, choose:  
*Breakpoints → Breakpoint for statement → Enter RFC*
- Through the i key (information key) in your ABAP editor, you can specify the ABAP term RFC and call up useful information on the RFC topic.



- Synchronous and asynchronous RFCs are suitable for remote data retrieval. For remote data changes, transactional or queued RFC is more suitable (depending on requirements) because of guaranteed remote processing confirmation.
- To start a function module using RFC, the RFC logon user in the target system needs a corresponding authorization for the authorization object S\_RFC.
- For an existing trust relationship between two SAP R/3 Systems, specific users in the source system can use the user released for them in the target system for RFC logons without a password.
- To release a user in the target system, you need to give an authorization for the authorization object S\_RFCACL.

© SAP AG 2002



# Exercises



## Unit: Remote Function Call

### Topic: Remote Destinations



- Creating a logical destination for RFC communication with another R/3 System.

RFC destination name: DEST##  
User in source system: BC415-##  
Remote user for RFC logon: BC415-## (## = your group number)

1-1 Create and maintain logical destinations for Remote Function Call using transaction SM59.

Create a destination in R/3 System A through which it can communicate with R/3 System B using RFC. (The names of systems A and B will be given to you by your instructor.)  
You can find out the information you need using transaction SM51 in the target system.

1-2 Test your destination.

#### *Tips & Tricks*



When maintaining the destination, you can press ENTER after entering the connection type to refresh the screen where fields relating to the connection type appear for maintenance.







## Unit: Remote Function Call

### Topic: Synchronous RFC



- Calling a remote function module synchronously and passing data.

Template: SAPBC415\_RFCSYNC\_T1  
Name of your solution program: ZBC415\_##\_RFCSYNC  
Name of the function module to be called: BC415\_RFC\_READ\_SPFLI

2-1 In system B, learn to use the functions of the **BC415\_RFC\_READ\_SPFLI** function module (interface and source text).

The function module reads an entry from table SPFLI. In the function call, you pass the relevant key values required for the read access using the interface parameters.

If no table entry was found during the read access, an exception is triggered in the function module.

If the function module reads an entry successfully, it passes the table entry and system ID back to the caller through the interface parameters.

2-2 In system A, create a program by first copying the sample program listed above.

From your program, call the function module BC415\_RFC\_READ\_SPFLI in destination system B.

Check the exceptions defined in the function module.

If an error occurs, display the relevant error message. If the program is successful, output the data returned by the function module.





## Unit: Remote Function Call

### Topic: Asynchronous RFC



- Calling an RFC function module synchronously and passing data.

Template: ZBC415\_##\_RFCSYNC (or SAPBC415\_RFCSYNC\_S1)

Name of your solution program: ZBC415\_##\_RFCASYNC

#### 3-1 Calling a function module remotely and asynchronously

Copy your solution from the Synchronous RFC exercise or the model solution SAPBC415\_RFCSYNC\_S1, and convert the function call, data transfer, and exception handling to asynchronous processing.





## Unit: Remote Function Call

### Topic: Transactional RFC



- Making identical changes to a customer's telephone number in two R/3 Systems using transactional RFC.

Template: SAPBC415\_RFCTRANS\_T1  
Name of your dialog program : SAPMZ\_BC415\_##\_RFCTRANS  
Name of your transaction : ZBC415\_##

- 4-1 Making the same changes to customer data in two R/3 Systems.
- 4-1-1 Copy the program SAPBC415\_RFCTRANS\_T1 to your program.  
This program is a module pool, with screens and user interfaces. Ensure that you copy all of the program components!  
You need to define a transaction code for your dialog program (see name above).
- 4-1-2 Go to the module USER\_COMMAND\_0200. There you will find a CASE statement for the field OKCODE. Enter the processing step you want to supplement under *WHEN 'SAVE'*.
- 4-1-3 The function module BC415\_TRFC\_CHANGE\_SCUSTOM\_ENTR allows you to change the telephone number of a customer (table SCUSTOM ). This function module contains two import parameters:  
CUSTOMER\_ID → customer number  
I\_TELEPHONE → telephone number  
Call the function module locally (→ local change) and then call it again remotely using transactional RFC (→ identical remote change).
- 4-1-4 In the remote system, use the Data Browser (transaction SE16) to check that the changes have been made correctly. If not, use transaction SM58 (tRFC monitor) to find out why.
- 4-1-5 Set the processing of the tRFC LUW for a future time. To do this, use the START\_OF\_BACKGROUNDTASK function module. In screen 200, you see the entry field for the time of execution that you pass along when you call the function module.
- 4-1-6 In the source system, use transaction SM58 to check that the tRFC LUW has been correctly scheduled and executed.





## Unit: Remote Function Call

### Topic: Remote Destination



- Creating a logical destination for RFC communication with another R/3 system.

1. Call transaction **SM59**.
2. Choose *Create*. A screen appears, on which you should enter the name of the new destination. In the "Connection Type" field, enter **3** as the connection type (→ the target system is an R/3 system). Enter a description. Choose *Enter* to continue.
3. In the next step, you must specify an R/3 application server for the target system by entering the relevant host name and the R/3 system number. You can get the required information by running transaction SM51 in the target system. There you will see a list of all the instances in the target system. The instance ID consists of the:
  - Name of the host on which the instance is running (for example, hs0330)
  - System ID (for example, ABC)
  - System number for the instance (for example, 17)This example would therefore be displayed as follows: hs0330\_ABC\_17.
4. Choose "Unicode Test" to check if the target system is a Unicode system, and make the appropriate settings according to the result.
5. You must now enter the RFC logon data ( language, client, user, password), and save the destination.
6. Test your RFC destination by choosing "Connection Test" or "Remote Logon".



## Unit: Remote Function Call

### Topic: Synchronous RFC



- Calling an RFC function module synchronously, with data transfer.

#### Source text of the solution program

```
REPORT sapbc415_rfcsync_s1 MESSAGE-ID bc415.

PARAMETERS:
    icarr TYPE sflight-carrid DEFAULT 'LH',
    iconn TYPE sflight-connid DEFAULT '0400',
    dest  TYPE rfcdes-rfcdest DEFAULT 'NONE'.

DATA: wa_spfli TYPE spfli,
      sysid    LIKE sy-sysid,
      mess(80).

START-OF-SELECTION.

    CALL FUNCTION 'BC415_RFC_READ_SPFLI'
      DESTINATION dest
      EXPORTING
        carrid      = icarr
        connid      = iconn
      IMPORTING
        ex_spfli    = wa_spfli
        sys         = sysid
      EXCEPTIONS
        invalid_data      = 1
        communication_failure = 2 MESSAGE mess
        system_failure    = 3 MESSAGE mess.

CASE sy-subrc.

  WHEN 0.
    WRITE: / text-004, sysid COLOR 5.
```



SKIP.  
WRITE: wa\_spfli-carrid, wa\_spfli-connid,  
      wa\_spfli-cityfrom, wa\_spfli-cityto.  
WHEN 1.  
    WRITE / text-001.  
WHEN 2.  
    WRITE / mess.  
WHEN 3.  
    WRITE / mess.  
ENDCASE.





## Unit: Remote Function Call

### Topic: Asynchronous RFC



- Calling an RFC function module asynchronously with data transfer.

### Source text of the solution program

```
REPORT sapbc415_rfcasync_s1 MESSAGE-ID bc415.
```

```
PARAMETERS:
```

```
  icarr TYPE sflight-carrid DEFAULT 'LH',
  iconn TYPE sflight-connid DEFAULT '0400',
  dest  TYPE rfcdes-rfcdest DEFAULT 'NONE'.
```

```
DATA: wa_spfli TYPE spfli,
      sysid    LIKE sy-sysid,
      mess(80),
      flag,
      retcode  LIKE sy-subrc.
```

```
START-OF-SELECTION.
```

```
CALL FUNCTION 'BC415_RFC_READ_SPFLI'
```

```
  DESTINATION dest
```

```
  STARTING NEW TASK 'T1'
```

```
  PERFORMING back ON END OF TASK
```

```
  EXPORTING
```

```
    carrid          = icarr
```

```
    connid          = iconn
```

```
  EXCEPTIONS
```

```
    communication_failure = 2 MESSAGE mess
```

```
    system_failure       = 3 MESSAGE mess.
```

```
CASE sy-subrc.
  WHEN 2.
    WRITE: / mess.
    EXIT.
  WHEN 3.
    WRITE: / mess.
    EXIT.
ENDCASE.

WAIT UNTIL flag IS NOT INITIAL.

CASE retcode.
  WHEN 0.
    WRITE: / text-004, sysid COLOR 5.
    SKIP.
    WRITE: wa_spfli-carriid, wa_spfli-connid,
           wa_spfli-cityfrom, wa_spfli-cityto.
  WHEN 1.
    WRITE: / text-001, / text-002.
  WHEN 2 OR 3.
    WRITE mess.
ENDCASE.
```

```
*-----*
*          FORM BACK                                *
*-----*
*          -->  T                                    *
*-----*

FORM back USING t.

  RECEIVE RESULTS FROM FUNCTION 'BC415_RFC_READ_SPFLI'
  IMPORTING
    ex_spfli = wa_spfli
    sys      = sysid
  EXCEPTIONS
    invalid_data          = 1
    communication_failure = 2  message mess
    system_failure        = 3  message mess.

  retcode = sy-subrc.

  flag = 'X'.

ENDFORM.                "BACK
```





## Unit: Remote Function Call

### Topic: Transactional RFC



- Making identical changes to a customer's telephone number in two R/3 systems using transactional RFC.

#### Screen 200 (Flow Logic)

```
PROCESS BEFORE OUTPUT.
  MODULE status_0200.
  MODULE set_values.

PROCESS AFTER INPUT.
  MODULE user_command_0200.
```

#### Modul user\_command\_0200

```
MODULE user_command_0200 INPUT.

  CASE okcode.

    WHEN 'SAVE'.

      CALL FUNCTION 'BC415_TRFC_CHANGE_SCUSTOM_ENTR'
        EXPORTING
          customer_id = scustom-id
          i_telephone = scustom-telephone
        EXCEPTIONS
          no_data_found = 1.

      IF sy-subrc <> 0.
        MESSAGE a514 WITH scustom-id.
      ELSE.
        MESSAGE i515.
      ENDIF.

      CALL FUNCTION 'BC415_TRFC_CHANGE_SCUSTOM_ENTR'
        IN BACKGROUND TASK
        DESTINATION destination
        EXPORTING
          customer_id = scustom-id
```

```
        i_telephone = scustom-telephone.

CALL FUNCTION 'START_OF_BACKGROUNDTASK'
  EXPORTING
    startdate = sy-datum
    starttime.= time.

COMMIT WORK.
MESSAGE i517 WITH time.

WHEN 'SM58'.
  CALL TRANSACTION 'SM58'.

WHEN 'BACK'.
  LEAVE TO SCREEN 100.

WHEN 'END'.
  LEAVE PROGRAM.

WHEN 'CANC'.
  LEAVE TO SCREEN 100.

ENDCASE.

ENDMODULE.                                " user_command_0200 INPUT
```



- **BAPI Overview**
- **Using the BAPI Explorer to Search For and Display BAPIs**
- **Calling BAPIs in Other R/3 Systems**

© SAP AG 2002

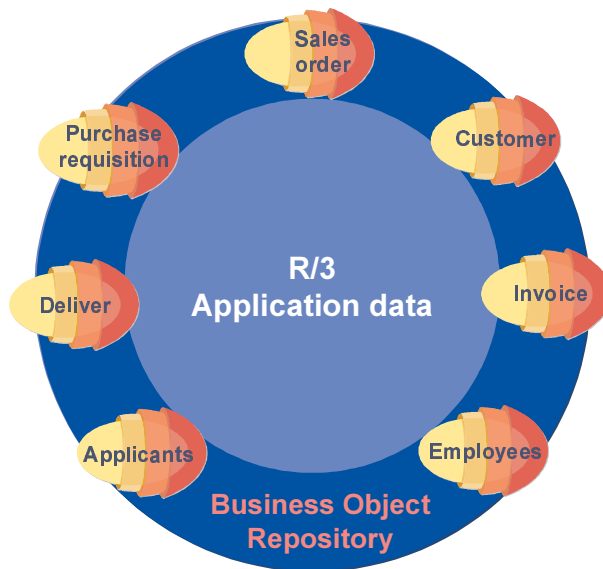
Internal Use SAP Partner Only

Internal Use SAP Partner Only



- Overview of business objects, business object repository (BOR), and the architecture of BAPIs
- Using the BAPI Explorer to search for and display BAPIs
- Calling BAPIs in other R/3 systems

© SAP AG 2006



### Business Objects

- Object-oriented view of R/3 application data
- Methods for displaying/changing application data
- Administration in the Business Object Repository (BOR)

© SAP AG 2006

- SAP business objects provide an object-oriented view of business data and processes in the R/3 System.
- The Business Object Repository (BOR) manages business objects, organizational objects, and technical objects in a hierarchical display corresponding to the component hierarchy in the R/3 system.

## Examples of Business Objects

SAP



### Employees

#### Attributes:

- Name
- Address
- Salary

#### Methods:

- Change address
- Increase salary

### Sales Order

#### Attributes:

- Customer
- Date
- Item list

#### Methods:

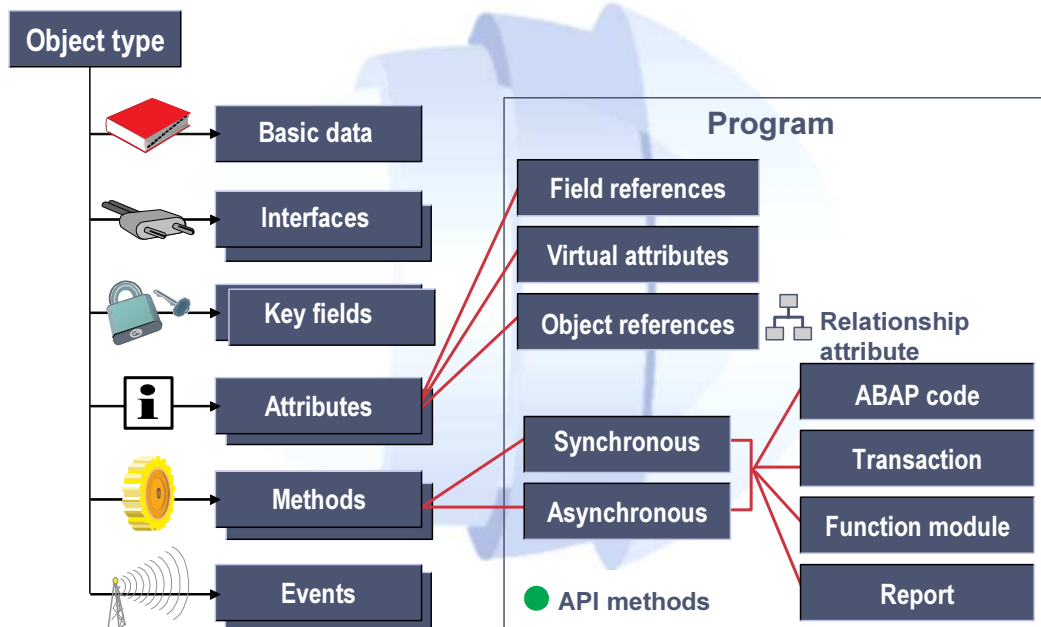
- Create
- Display
- Lock

© SAP AG 2006

- You can use the methods of a business object to display or change its attributes.

## Components of an Object Type

SAP



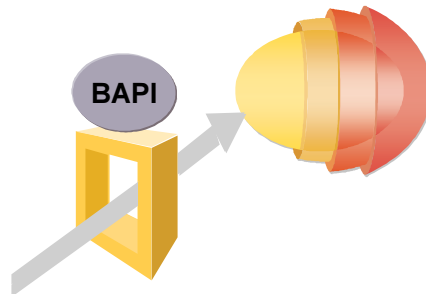
© SAP AG 2002

### ■ Components of an object type in the BOR

- **Basic data:** Technical details such as an internal ID, release level, transport data, and so on.
- **Interfaces:** List of interfaces from which the object type adopts attributes, methods, and events.
- **Key fields:** Attributes that uniquely identify an object of this type. Usually the key fields of the underlying database tables.
- **Attributes:** Object type attributes. These are either values from database fields (field references), values that are calculated at runtime (virtual attributes), or pointers to other objects (object references). Object references are particularly useful in Workflow definitions, allowing you to navigate easily between objects that belong together in a particular business context.
- **Methods:** Calls to R/3 transactions, function modules, or other ABAP code. BAPIs are known as API methods.
- **Events:** To be used in workflow definitions. The events are only defined in the BOR. To trigger an event, use the following links (for details, see the online documentation on workflow definition):
  - Links to change documents for the underlying database fields
  - Links to the status administration
  - Links to message control
  - Calling the function module SWE\_EVENT\_CREATE

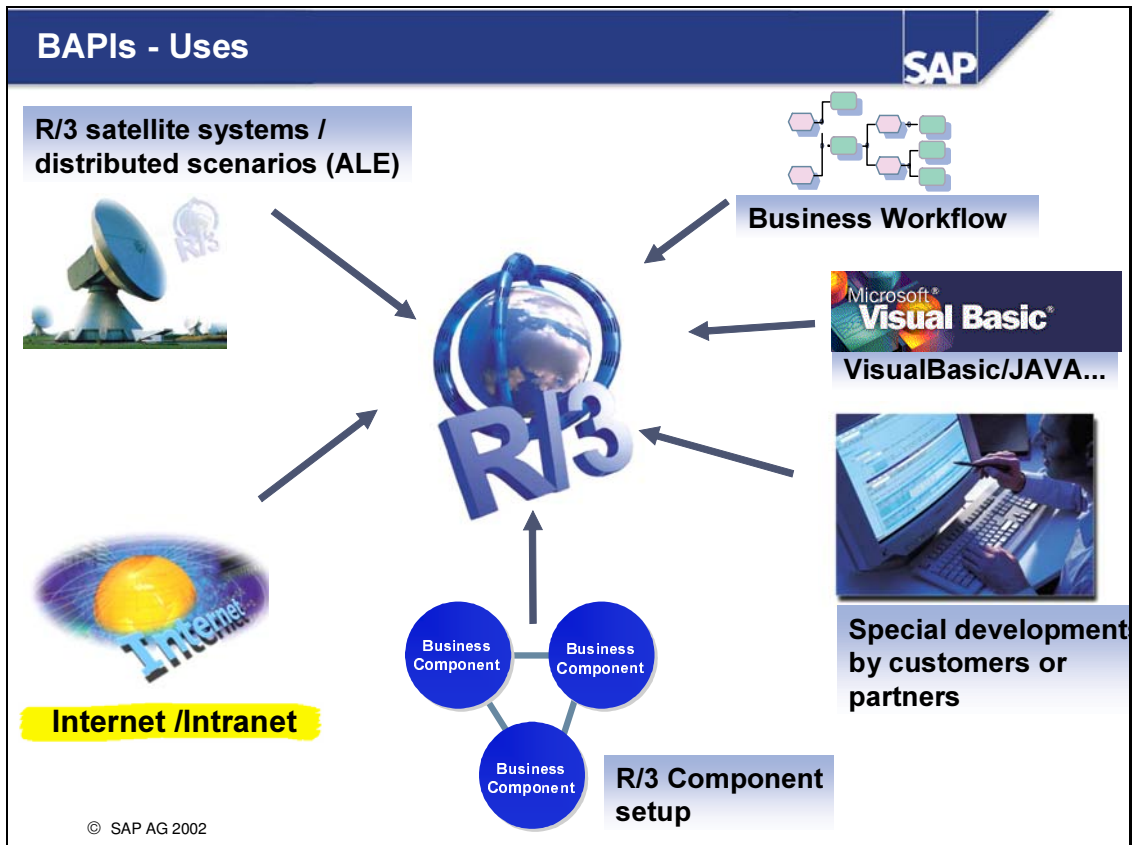
### Business Application Programming Interface

- **Well-defined interface for R/3 data and processes**
- **Implemented as a special remote-enabled method of business objects in the Business Object Repository**



© SAP AG 2002

- BAPIs are special, remote-enabled methods of a business object; they are implemented as RFC-enabled function modules.
- BAPIs can perform various functions such as:
  - Creating an object
  - Querying the attributes of an object
  - Changing the attributes of an object
- BAPIs are interfaces to R/3 data.



- You can use BAPIs for different applications. For example:
  - Internet application components: Where individual R/3 functions are implemented on the Internet or intranet for users with no R/3 experience.
  - Building R/3 components: Communication between the business objects from different R/3 components (applications).
  - VisualBasic / JAVA / C++ : External clients (for example, alternative GUIs) access R/3 data and processes through BAPIs.

- **Object-oriented**
  - Implemented as object methods from the BOR
- **Stable interface**
  - The interface of a BAPI cannot be changed
- **Can be used internally and externally**
  - BAPIs can be used both within the SAP R/3 System and externally
- **Do not have a presentation layer**
  - The caller is responsible for the external visualization of the results
- **Success and error confirmation through export parameter RETURN**
  - Structure or table (according to each BAPI)

© SAP AG 2002

- You can access BAPIs from external clients like methods access objects, which are themselves an instance in the client.
- SAP guarantees that no incompatible changes will be made to the interfaces of BAPIs.
- You can also use BAPIs within the R/3 System to access business data. This makes it easier for customers and partners to add to the functions offered by the R/3 System.
- Displaying data that is transmitted to a BAPI or that is returned from a BAPI must be done from the calling program.
- The return parameter RETURN contains success or error messages for the BAPI, and, depending on the SAP R/3 release, has one of these dictionary structures: BAPIRETURN, BAPIRETURN1, BAPIRET1, BAPIRET2 or BAPIRET2\_FIX. These structures commonly comprise the following relevant fields:
  - TYPE (message type: S(uccess), E(rror), W(arning), I(nformation) ) ;
  - ID (message class) ; NUMBER (message number) ; MESSAGE (message text) ;
  - MESSAGE\_V1, MESSAGE\_V2, MESSAGE\_V3, MESSAGE\_V4 (message variables)
- If the transmission is successful, RETURN is either completely empty (all the fields have their initial fields for their types), or only the TYPE field has the value S. Refer to the online documentation to find out which applies to the BAPI you are using.



- Database updates in BAPI always through update task
- No COMMIT WORK or ROLLBACK WORK in the BAPI
- BAPI-LUW is closed or rejected by corresponding BAPI call

Service object *BapiService*

BAPI *TransactionCommit*

BAPI *TransactionRollback*

© SAP AG 2002

- Database updates in BAPIs are always performed by an update task. The update type (synchronous or asynchronous) is specified when you call the commit BAPIs.
- In BAPIs containing database updates, a COMMIT WORK or ROLLBACK WORK is not used. Closing or rejecting a BAPI LUW occurs by calling a special BAPI from the service object BapiService (object type SAP0001).
  - BAPI *TransactionCommit* (relevant function module BAPI\_TRANSACTION\_COMMIT)
  - BAPI *TransactionRollback* (relevant function module BAPI\_TRANSACTION\_ROLLBACK)

### Standard BAPIs

- **GetList**
  - Returns the contents of the object's key fields
- **GetDetail**
  - Returns information (attributes) on the object concerned
- **CreateFromData**
  - Generates a new object in the R/3 System and returns information about it

### BAPI Explorer (Transaction BAPI)

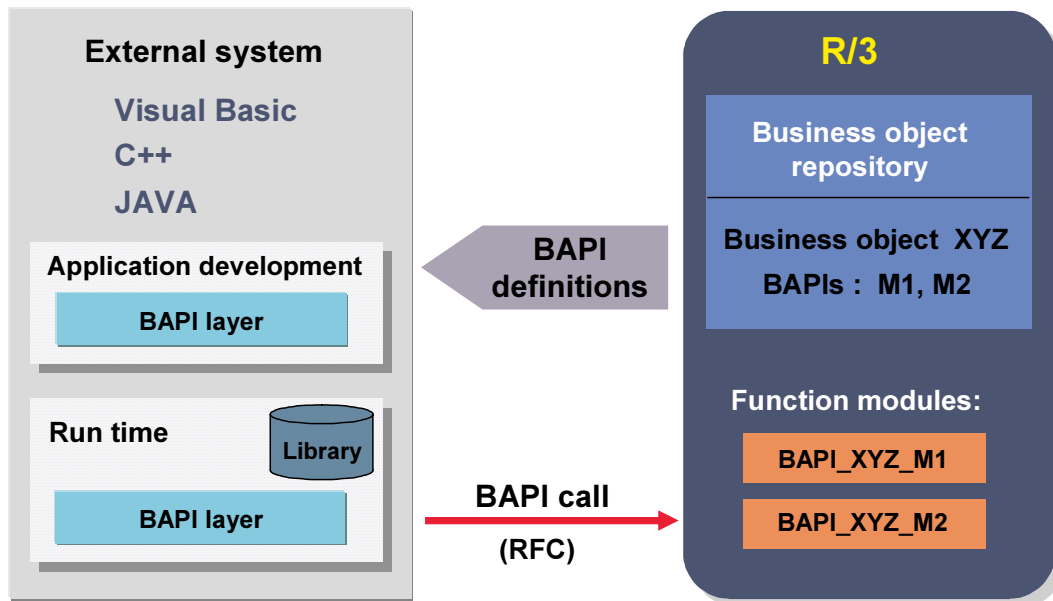
- **Display of all business objects with BAPIs grouped by application**

© SAP AG 2002

- The above BAPIs exist for many business objects.
- Depending on the object, you can use additional BAPIs (for example, CheckPassword for the business object *Customer*).
- The BAPI Explorer (transaction BAPI) displays all business objects with BAPIs, grouped by application.
- There is an example of how to create your own BAPIs in the appendix.
- For more detailed information and guidelines on BAPI programming, refer to the BAPI Explorer in the "Project" tab page.

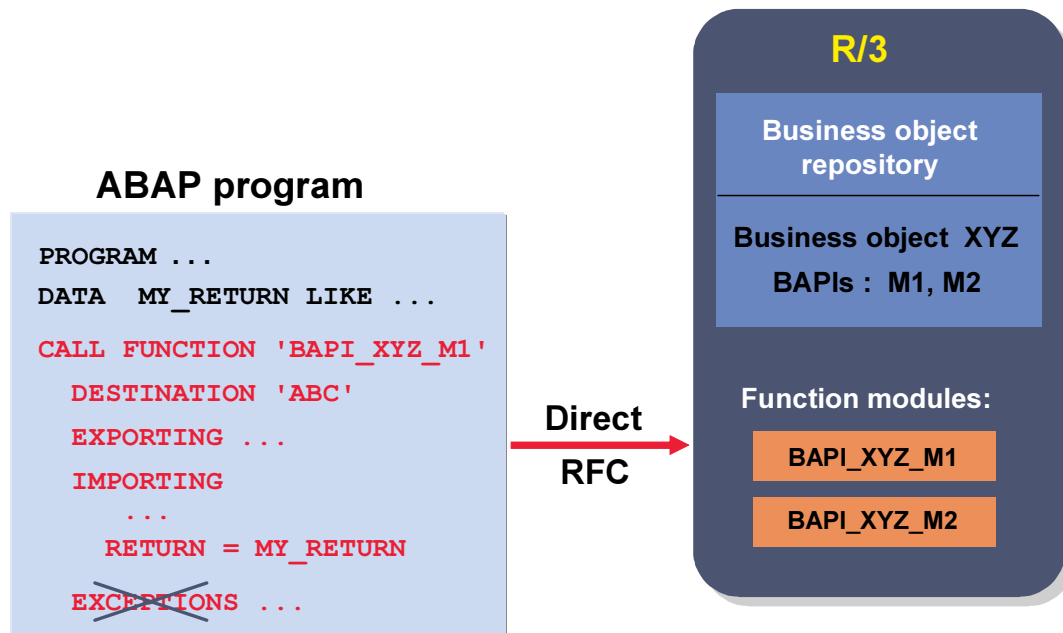
## Accessing BAPIs from External Systems

SAP



© SAP AG 2002

- The programmer needs to know which business objects are available with which BAPIs, and what the interface looks like.
- The BAPI Browser is a logical view of the BOR in which only those objects that have BAPIs are displayed. The complete interface of each BAPI is displayed.
- SAP delivers libraries for BAPI calls for various development environments (Visual Basic, C++, Java). These libraries include the technical details of the communication, enabling developers to call a BAPI from outside the system without having to worry about how BAPIs are actually implemented in the system.
- You can get more information on using BAPIs with Visual Basic, C++, and Java from the training courses CA925, CA926, and CA927.



© SAP AG 2002

- Currently, you can only access BAPIs from an ABAP program using the relevant direct function call (local or remote). SAP is planning a later release where all BOR business objects will be integrated in the class library of ABAP so that you will be able to access BAPIs object-oriented from ABAP using ABAP-OO.
- The naming convention for BAPI function modules is: BAPI\_<business\_object>\_<method>
- There are no exceptions for BAPI function modules.  
Success or error messages are returned by the RETURN export parameter, which, depending on the SAP R/3 release, has one of these dictionary structures: BAPIRETURN, BAPIRETURN1, BAPIRET1, BAPIRET2 or BAPIRET2\_FIX. These structures commonly comprise the following relevant fields:  
TYPE (message type: S(uccess), E(rror), W(arning), I(nformation) ) ;  
ID (message class); NUMBER (message number); MESSAGE (message text) ;  
MESSAGE\_V1, MESSAGE\_V2, MESSAGE\_V3, MESSAGE\_V4 (message variables)
- If the transmission is successful, RETURN is either completely empty (all the fields have their initial fields for their types), or only the TYPE field has the value S. Refer to the documentation to find out which applies to the BAPI you are using.



- **BAPIs are remote-enabled methods for objects in the BOR with an interface that cannot be changed.**
- **You can currently access BAPIs with object-oriented programming only from external systems (Visual Basic, C++, Java).**  
**The BAPI call can be used from ABAP as a direct (remote) function call only.**
- **BAPIs always run database updates through update tasks.**  
**Since BAPIs do not contain COMMIT WORK or ROLLBACK WORK, you can close or reject an LUW only by calling specific BAPIs.**
- **Using the BAPI Explorer, you can display BAPIs grouped by application.**

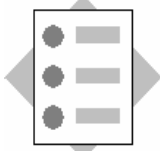
© SAP AG 2002



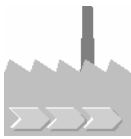


## Unit: BAPI - Business Application Programming Interface

### Topic: Finding, Displaying, Testing, and Calling BAPIs



- Searching for a BAPI in the BAPI Explorer
- Displaying and testing a BAPI
- Calling a BAPI from ABAP



You want to find existing BAPIs for a specific R/3 component, display them, test them locally, and call them from a remote R/3 System.

### BAPIs : Finding, Displaying, Testing, Calling

1-1 Find the business object *FlightConnection* in **system B**.

- 1-1-1 Which BAPIs does the business object contain?
- 1-1-2 Which BAPI do you use to get a list of flights?
- 1-1-3 Which function module is behind this BAPI?
- 1-1-4 Test the BAPI in the test environment.

1-2 Call the BAPI from **system A**.

- 1-2-1 To create the coding for the function module call, create in **system B** a dummy program with any name, and use the editor function *template* for **CALL FUNCTION** with the corresponding function module. Copy the coding you created to the Clipboard so you can enter it in the ABAP Editor of system A.

- 1-2-2 In **system A**, create a new **ZBC415\_##\_BAPI** program. Insert the function module call from the Clipboard and supplement it with additions and the necessary parameter values for an sRFC in system B. Get all the flights from FRANKFURT (DE) to NEW YORK (US) from the remote BAPI.

After a successful call (**SY-SUBRC = 0**) the field **TYPE** of the **RETURN** structure has the value "S". In this case, output the contents of the internal table (required flights) returned by the BAPI.

If an RFC communication error (**SY-SUBRC NE 0**) or a BAPI error (**RETURN-TYPE NE 'S'**) has occurred, output the relevant error message.





# Solutions



Unit: BAPI - Business Application Programming Interface

Topic: Find, Display, Test, and Call BAPIs



- Searching for a BAPI in the BAPI Explorer
- Displaying and testing a BAPI
- Calling a BAPI from ABAP

BAPI you searched for to list flights:

***GetList***

Function module belonging to BAPI:

**BAPI\_SFLIGHT\_GETLIST**

ABAP program with remote BAPI call:

see following pages

REPORT sapbc415\_bapi\_s1

PARAMETERS:

```
fromcoun TYPE bapisfdeta-countryfr OBLIGATORY DEFAULT 'DE',
fromcity TYPE bapisfdeta-cityfrom OBLIGATORY DEFAULT 'FRANKFURT',
tocoun TYPE bapisfdeta-countryto OBLIGATORY DEFAULT 'US',
tocty TYPE bapisfdeta-cityto OBLIGATORY DEFAULT 'NEW YORK',
carrier TYPE bapisfdeta-carrid DEFAULT 'LH',
afternon TYPE bapi_aux-afternoon,
maxread TYPE bapi_aux-maxread,
dest TYPE rfcdes-rfcdest DEFAULT 'NONE'.
```

DATA:

```
flightlist TYPE TABLE OF bapisflist,
flight LIKE LINE OF flightlist,
return TYPE bapiret2,
msg(80).
```

START-OF-SELECTION.

```
CALL FUNCTION 'BAPI_SFLIGHT_GETLIST'
  DESTINATION dest
  EXPORTING
    fromcountrykey = fromcoun
    fromcity = fromcity
    tocountrykey = tocoun
    tocity = tocity
    airlinecarrier = carrier
    afternoon = afternon
    maxread = maxread
  IMPORTING
    return = return
  TABLES
    flightlist = flightlist
  EXCEPTIONS
    system_failure = 1 MESSAGE msg
    communication_failure = 2 MESSAGE msg
    OTHERS = 3.
```

CASE sy-subrc.

WHEN 0.

```
IF return-type <> 'S'.
  WRITE: / text-001, return-message.
ELSE.
  PERFORM print_list.
ENDIF.

WHEN 1 OR 2.
  WRITE: / text-002, msg.

WHEN OTHERS.
  WRITE: / text-003.

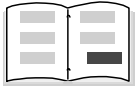
ENDCASE.

*&-----*
*&      Form  PRINT_LIST
*&-----*
FORM print_list.

  LOOP AT flightlist INTO flight.
    WRITE: / flight-carrid, flight-connid, flight-fldate,
            flight-airpfrom, flight-deptime,
            flight-seatsmax, flight-seatsocc.
  ENDLOOP.

ENDFORM.                " PRINT_LIST
```





**This appendix contains additional information, which is not a definite part of the standard course. Your instructor might not have enough time to cover this information during regular class time.**

© SAP AG 2002



**Maintaining an RFC Destination in R/2**

**Web RFC**

**The CPI-C Communication Interface**

**Asynchronous Data Transfer through Queue  
Application Programming Interface (QAPI)**

**Exercises and Solutions for CPI-C and Q-API;  
BAPI Workshop**

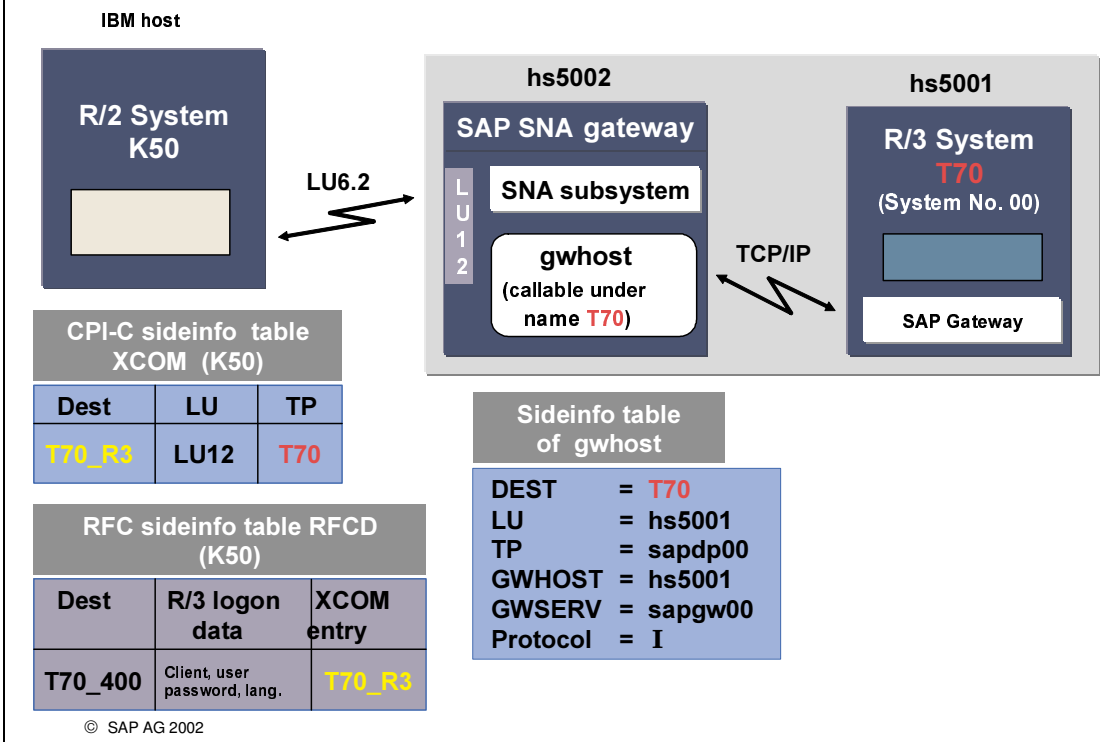
© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only

## Maintaining RFC Sideinfo: R/2 (IBM) - R/3

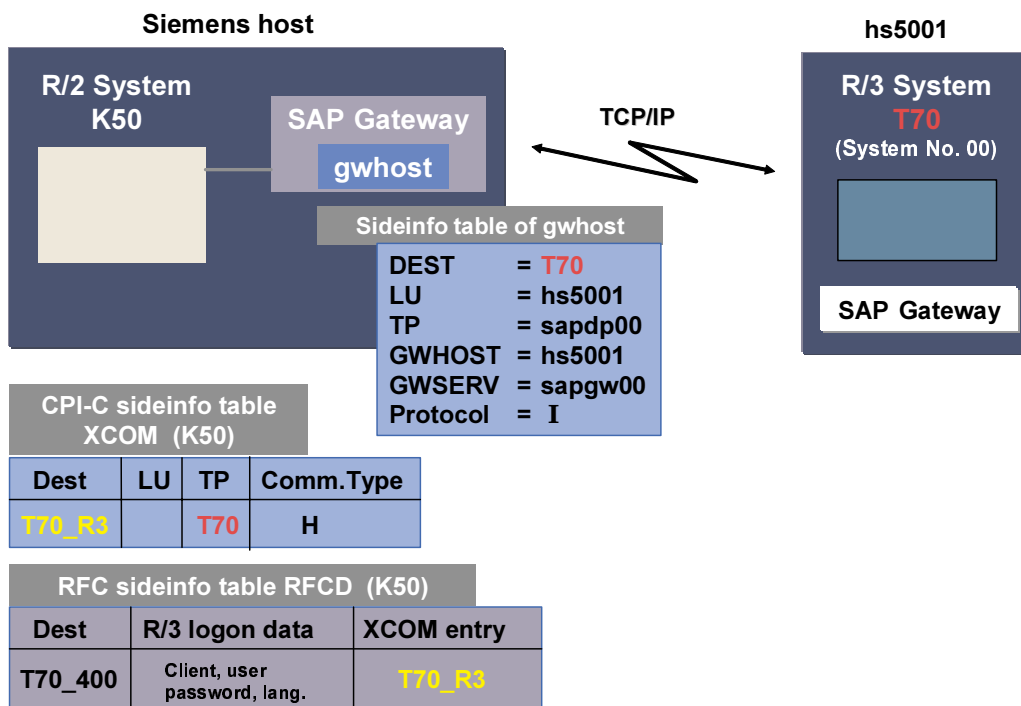
SAP



- The architecture of an RFC connection originating from an IBM R/2 System is identical to that of a CPI-C connection.
- For this type of communication, you must create an RFC destination in the RFC side information table RFCD within the IBM R/2 System using transaction TM31. You can choose any destination name you like. You must specify the R/3 logon data and the name of an existing XCOM entry as the communication parameters. The specified XCOM entry is used to build the connection when using this RFCdestination.

## Maintaining RFC Sideinfo: R/2 (Siemens) - R/3

SAP



© SAP AG 2002

- The architecture of an RFC connection originating from a Siemens R/2 System is identical to that of a CPI-C connection.
- For this type of communication, you must create an RFC destination in the RFC side information table RFCD in the Siemens R/2 System using transaction TM31. You can choose any destination name. You must specify the R/3 logon data and the name of an existing XCOM entry as the communication parameters. The specified XCOM entry is used to build the connection when using this RFC destination.



Maintaining an RFC Destination in R/2



**Web RFC**

The CPI-C Communication Interface

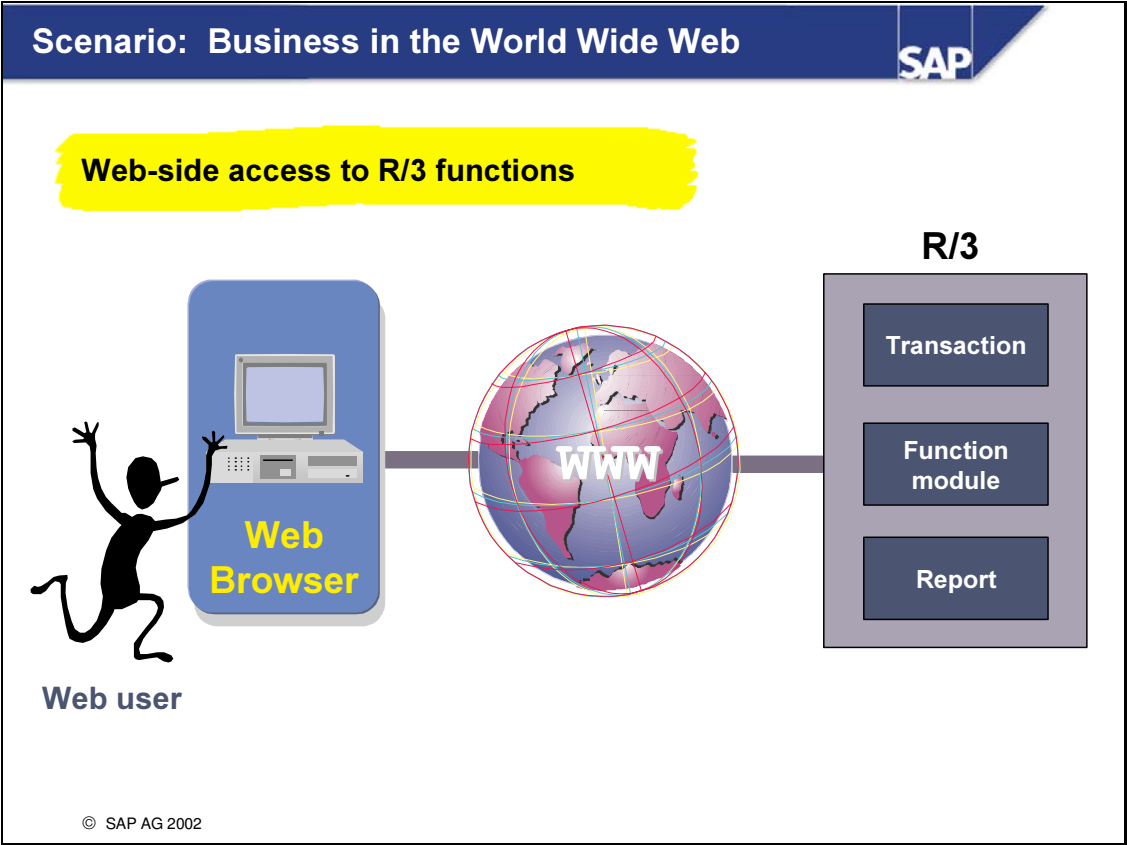
Asynchronous Data Transfer through QAPI

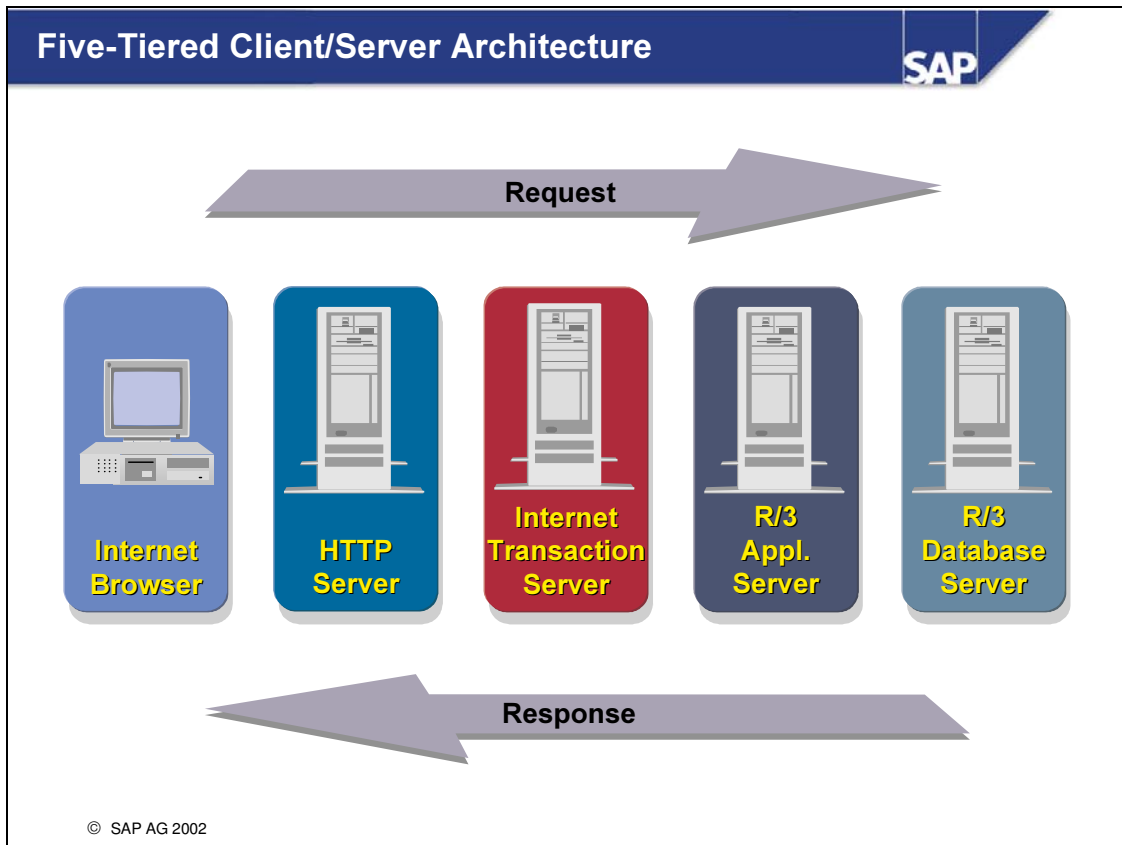
Exercises / Solutions for CPI-C and Q-API ;  
BAPI Workshop

© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only

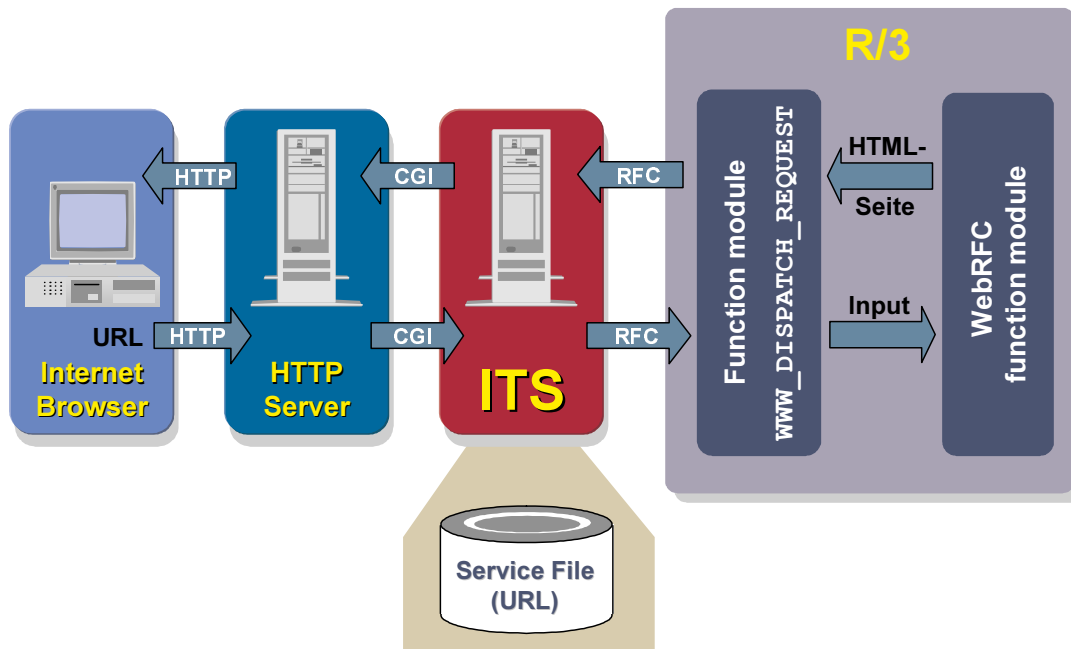




- The Internet Transaction Server connects the three-tiered client/server architecture of R/3 with the two-tiered HTTP client/server architecture.
- From the R/3 point of view, the ITS and HTTP servers form an intermediate layer between the application and presentation layers.
- From the Internet user's point of view, the ITS takes care of the interactive HTML pages (generated at runtime).

## Web RFC Architecture

SAP

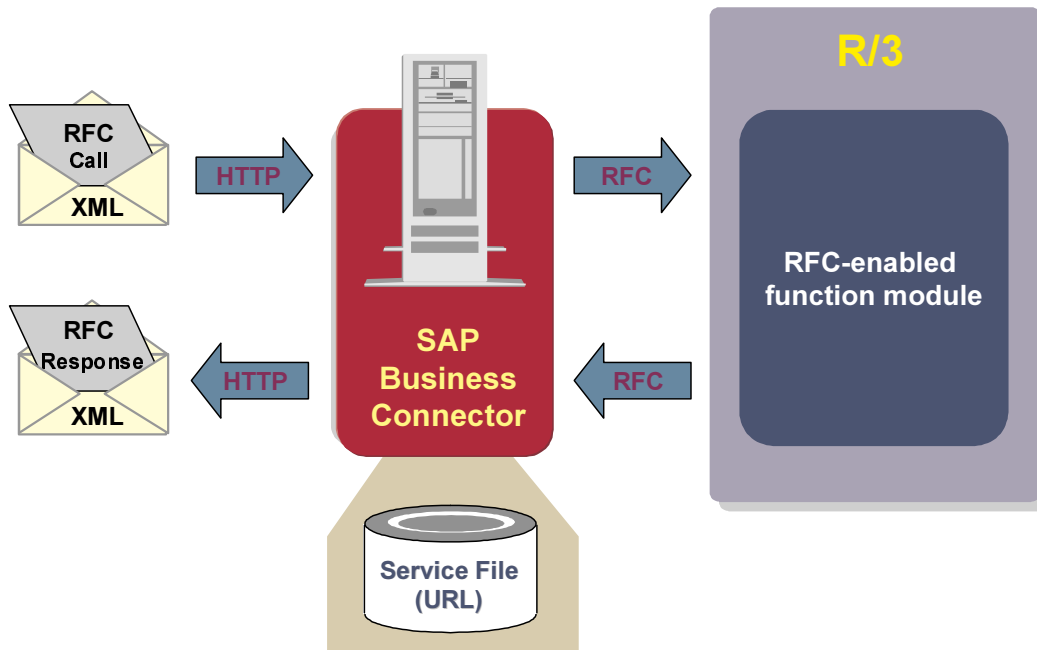


© SAP AG 2002

- WebRFC is the term used when you call special function modules in R/3 using the ITS. The service function module WWW\_DISPATCH\_REQUEST is always called as the agent that executes the call.
- Attributes of WebRFC-enabled function modules include:
  - Returns an HTML page or a MIME object as the output
  - Permits read or write access to the R/3 database
  - Allows business objects to be called from WebRFC-enabled function modules
  - Cannot be used by Internet users
- SAP delivers important, essential functions as WebRFC-enabled function modules.
- For more information, refer to training course BC440.

## RFC Using XML Files Through the Business Connector

SAP



© SAP AG 2002

- Use the SAP Business Connector to call an arbitrary RFC-enabled function module from the intranet or the Internet.
- On the SAP BC, you can set up a service that points to an RFC-enabled function module. If you call the service from the Internet using the corresponding URL, and an XML file is passed to the service, the application data will be transmitted through the SAP BC from the file to the function module interface in the SAP System. The response of the function module is converted by the SAP BC into an XML file, which is then returned to the caller.
- You will find examples of such XML files used for calling a function in the SAP Interface Repository, which is a collection of interfaces that can be viewed publicly (<http://ifr.sap.com>).
- The SAP BC is a middleware technology developed by SAP and can be used by SAP customers free of charge. For information on the SAP BC, refer to the service marketplace (<http://service.sap.com/connectors>).
- More details are available in the training course BIT530 *Introduction to the SAP Business Connector*.

## The CPI-C Communication Interface



Maintaining an RFC Destination in R/2

Web RFC



The CPI-C Communication Interface

Asynchronous Data Transfer through Q-API

Exercises / Solutions for CPI-C and Q-API ;  
BAPI Workshop

© SAP AG 2002

## The CPI-C Communication Interface: Topic Objectives



- In this chapter you will learn how to run program-to-program communication using the CPI-C interface.
- This includes:
  - Defining remote destinations
  - Using the CPI-C calls for communication
  - Understanding particular aspects of remotely calling an ABAP program
- You will also become familiar with the CPI-C test environment.

© SAP AG 2002

- **Introduction**
- **Remote Destinations**
- **Using CPI-C in Communications**
- **Calling an ABAP Program Remotely**
- **Development and Test Environment**

© SAP AG 2002





Introduction

Remote Destinations

Using CPI-C in Communications

Calling an ABAP Program Remotely

Development and Test Environment

© SAP AG 2002

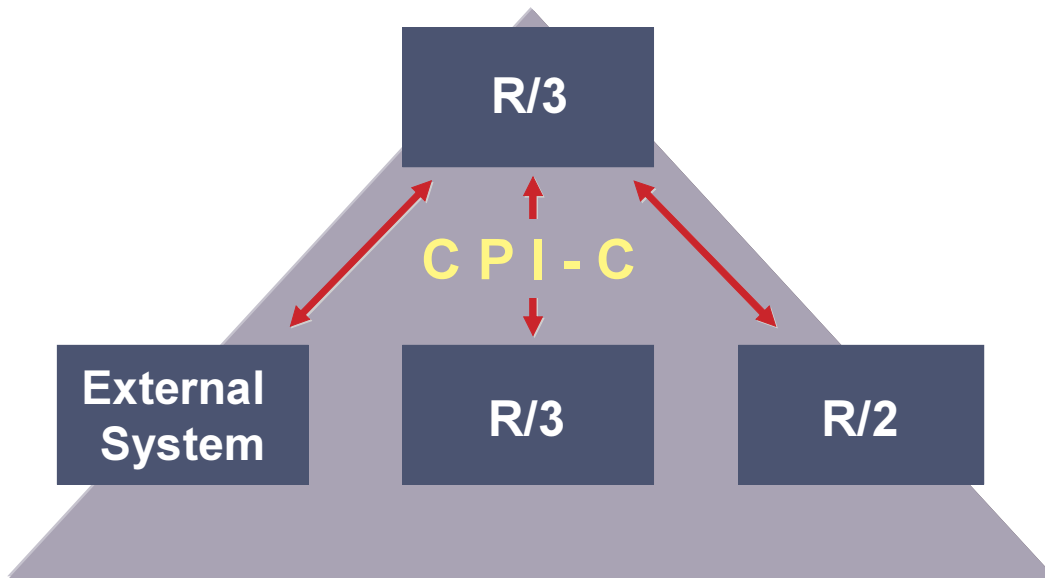
Internal Use SAP Partner Only

Internal Use SAP Partner Only



- In this topic you will learn about the general functions of CPI-C and the connection types.

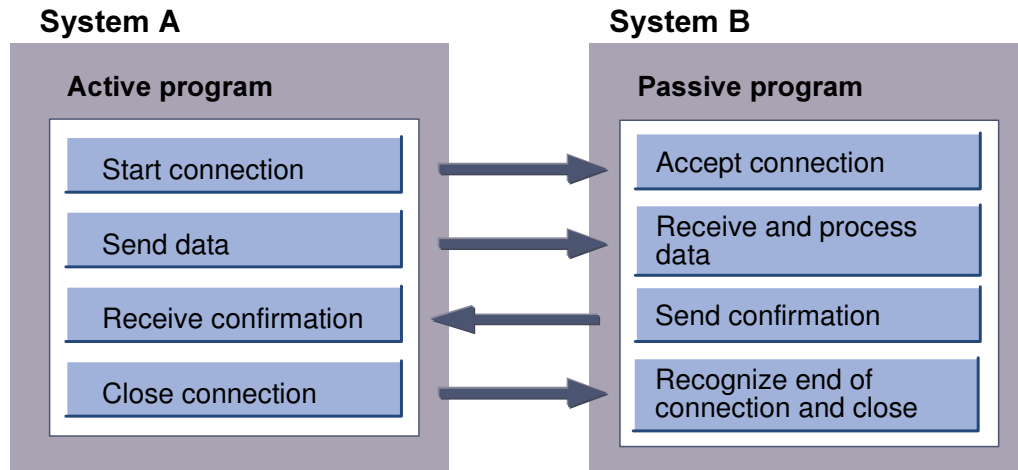
© SAP AG 2002



© SAP AG 2002

- The SAP CPI-C interface allows ABAP programs in an R/3 or an R/2 System and external C programs to communicate with each other and exchange data.
- Communication using CPI-C is an interesting option whenever RFC is not available.

## Program-to-program communication (CPI-C)



© SAP AG 2002

- CPI-C (Common Programming Interface for Communications) is a standardized program-to-program communication interface that was originally developed at IBM. It consists of a set of calls with which two programs can communicate with each other.
- SAP supports the CPI-C interface for ABAP and C programs.

Introduction



Remote Destinations

Using CPI-C in Communications

Calling an ABAP Program Remotely

Development and Test Environment

© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only



- In this topic, you will learn how to:  
Define the CPI-C destination (also known as a symbolic destination or side information entry) that is required to connect to the partner system.

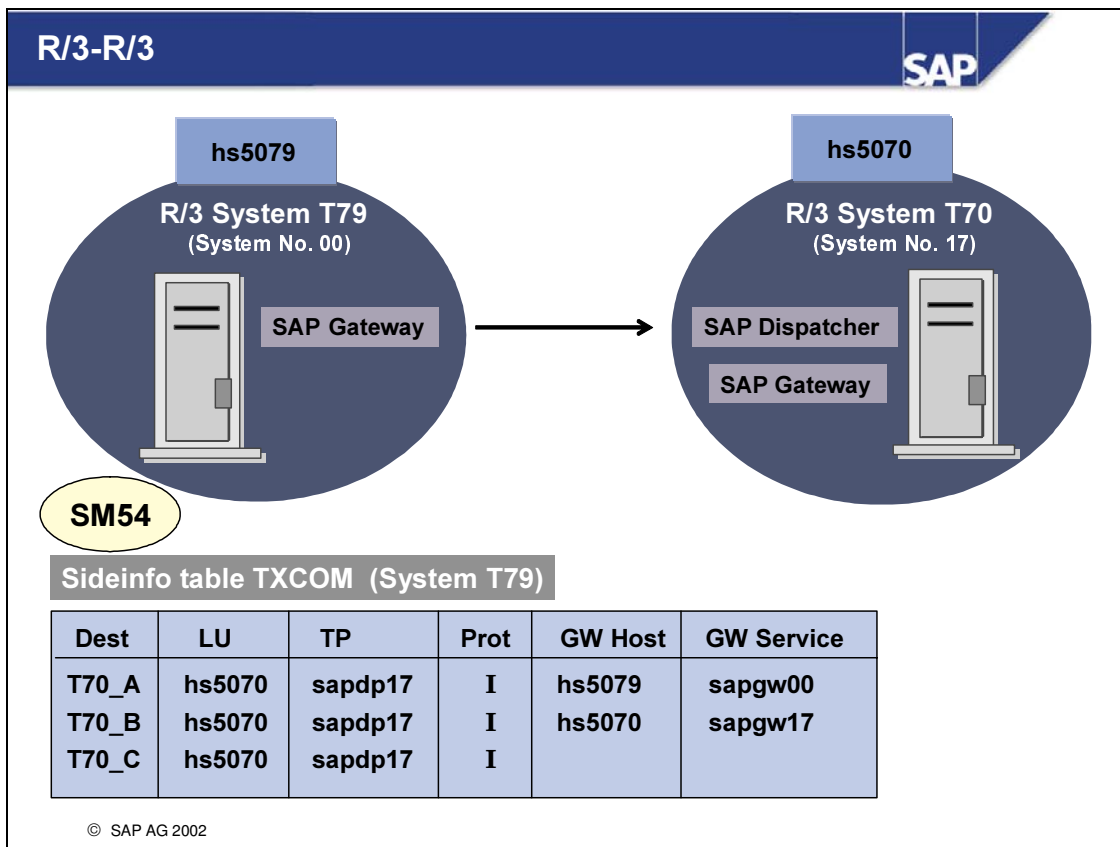
© SAP AG 2002

**Sideinfo Table in the Source System**

| Symbolic destination | Communication parameter |
|----------------------|-------------------------|
| T70                  | ...                     |
| K50                  | ...                     |
| EXTPGM               | ...                     |
| ...                  |                         |

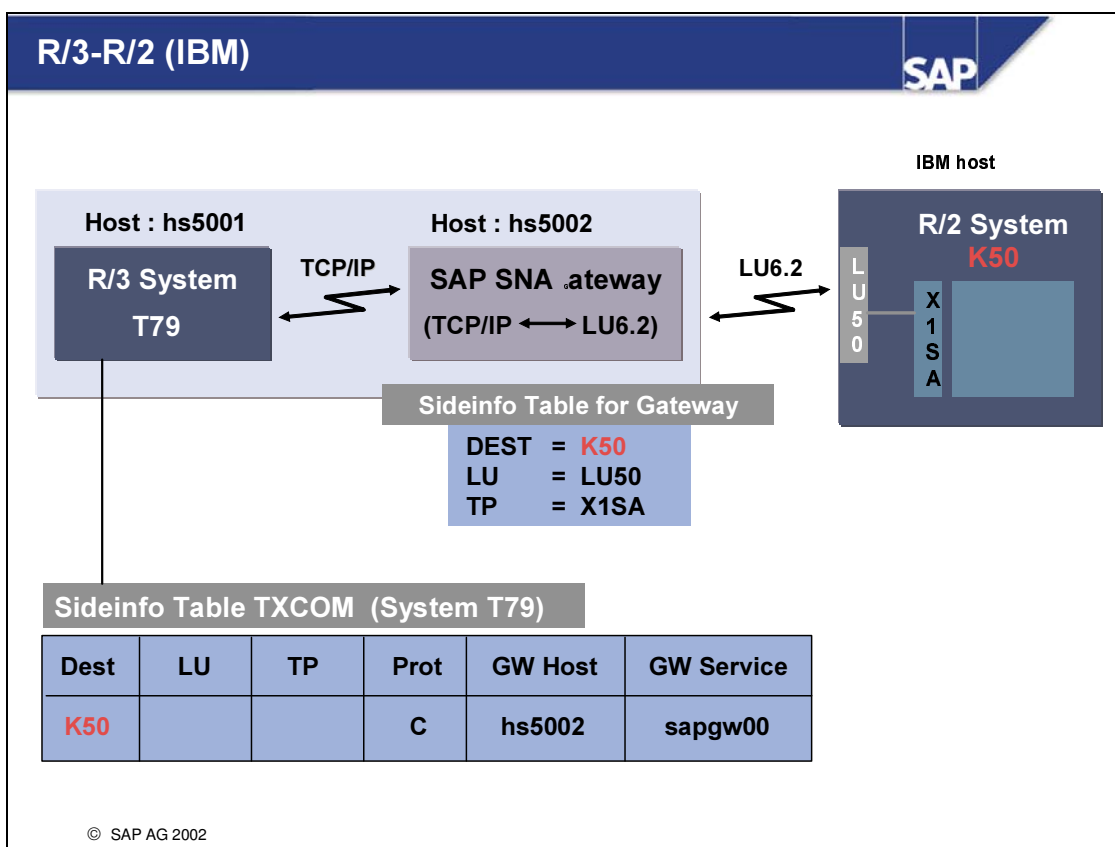
© SAP AG 2002

- To start communication with a partner using CPI-C, the active program has to have an entry in the side information table (sideinfo entry). This entry contains all of the necessary communication parameters.
- For each remote partner you want to call, you must have a corresponding sideinfo entry. These sideinfo entries are also known as **symbolic destinations**.
- The sideinfo table in the R/3 System is called **TXCOM**. You maintain it using transaction SM54.



- To start communication between two R/3 Systems, you need an entry in the sideinfo table TXCOM in the active (source) system.
- Specifications in the sideinfo entry:
  - Dest :** Any logical name (name of the sideinfo entry)
  - Prot:** 'I' (protocol type for R/3-R/3 connection)
  - GW host:** Host on which the requisite gateway is running
  - GW Service :** TCP/IP service for gateway to be used  
 (If the two fields *GW host* and *GW service* are not set with values, the default gateway of the current application server is used when the connection is set up.)
  - LU :** Application server of the appropriate R/3 system (target system)
  - TP :** TCP/IP service of the dispatcher of the passive R/3 System (target system)





- To communicate between an R/3 System and an IBM R/2 System, you need an **SNA gateway** (SAP Gateway with SNA functions) on which, in addition to TCP/IP, a suitable LU6.2 product is installed and configured. We recommend installing the SNA gateway as a dedicated standalone gateway that can be used by various R/3 application servers to communicate with SNA LU6.2 systems.
- To connect from an R/3 System to an IBM R/2 System, you need an entry in the sideinfo table **TXCOM** in the R/3 System, and an entry in the **sideinfo table of the SNA Gateway**.
 

**TXCOM entry:**

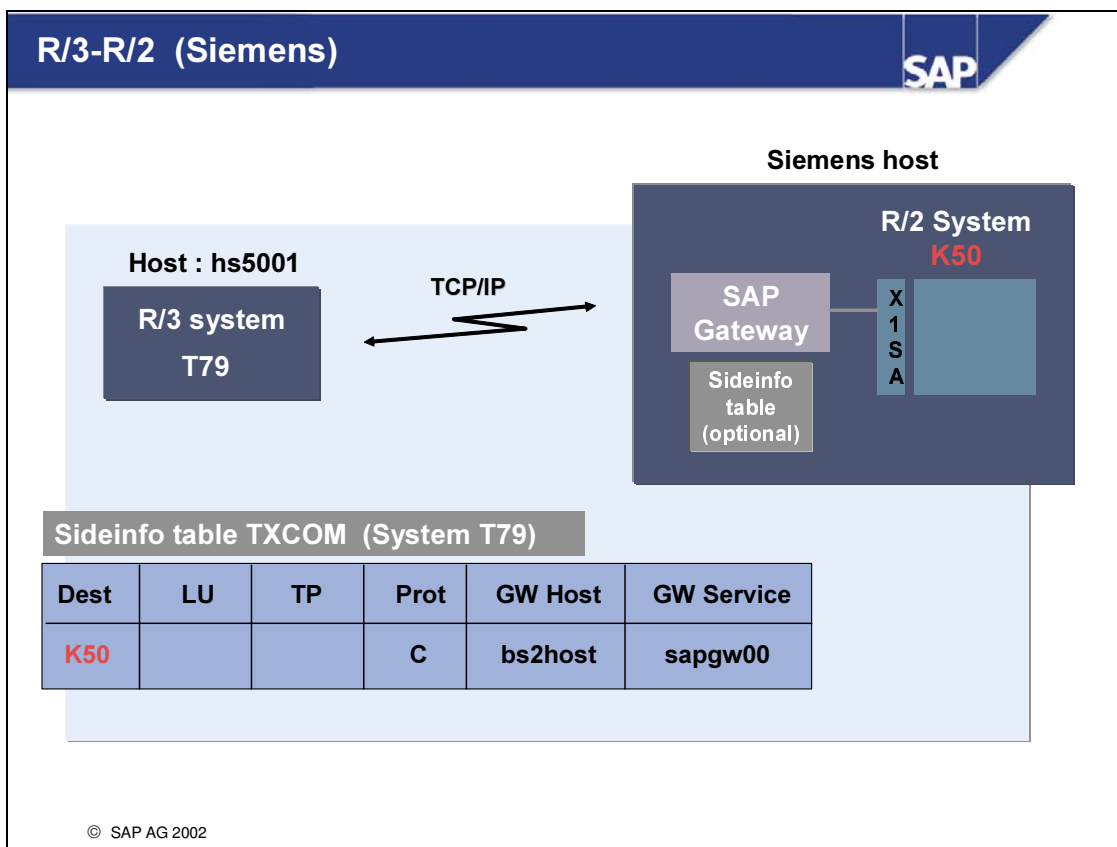
**Dest:** Name of sideinfo entry (must match name of corresponding gateway sideinfo entry)

**Prot:** C' (Protocol type for R/3-R/2 connection)

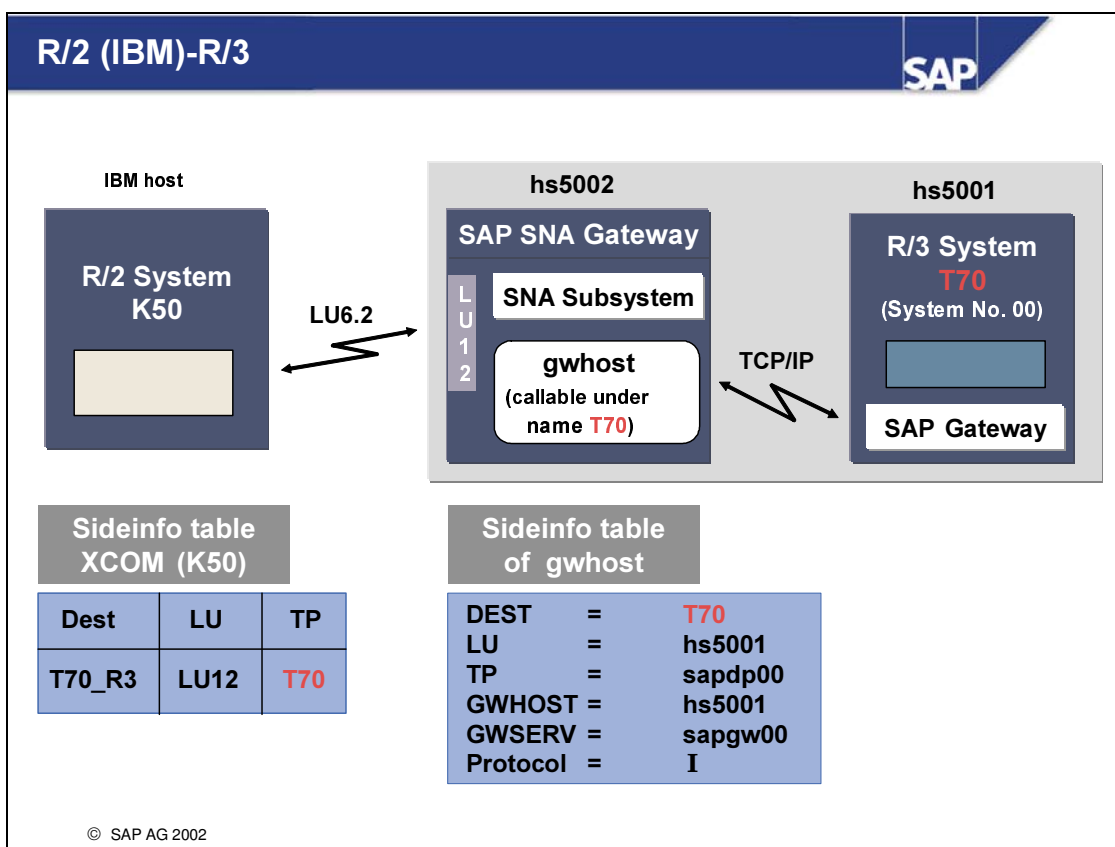
**GW host / GW service:** Host and TCP/IP service of the SNA gateway

**LU / TP:** Not important

**Sideinfo entry of gateway** (see graphic):  
Sideinfo on the gateway (see diagram): Contains the entry name (must match the corresponding TXCOM entry), SNA entries about the Logical Unit (LU) for the R/2 host (remotely called), and the R/2 Taskhandler (TP = X1SA). (See online documentation for maintenance of gateway sideinfo table)
- You can also install the SNA gateway on the host side. The advantage of running the SNA gateway on the IBM host is that you can use existing TCP/IP and LU6.2 software for the gateway on the host side.



- To communicate between an R/3 System and a Siemens R/2 System, you need an SAP gateway running on the Siemens host that can be addressed using TCP/IP.  
(TCP/IP functions on the host side are available starting with BS2000 version 10, DCAM version 11.)
- You can operate the host gateway with or without *sideinfo*.  
(Setting in host file SAPGW.RSPARAM, parameter *bs2/use\_sideinfo* [0,1] )  
You may have to maintain a corresponding sideinfo entry for the host gateway.  
Refer to the documentation for maintaining the gateway sideinfo table.
- To connect from an R/3 System to a Siemens R/2 System, you need an entry in the sideinfo table TXCOM in the R/3 System.  
**TXCOM entry:**  
**Dest :** Name of the DCAM or UTM application (SAP System) to be called by the gateway or name of the gateway side info entry.  
**Prot:** C (Protocol type for R/3-R/2 connection)  
**GW host / GW service:** Host name / TCP/IP service of the host gateway  
**LU / TP:** Not important



- To start a connection from an IBM R/2 System to an R/3 System, you need an entry in the **R/2 sideinfo table XCOM** and an entry in the **sideinfo table of the program gwhost** on the gateway.

Sideinfo entry in R/2 (table TXCOM)

**Dest** : Any logical name (name of the sideinfo entry)

**LU** : Logical unit of the SNA gateway

**TP** : Name under which the program *gwhost* can be called on the SNA gateway through the specified logical unit

**Sideinfo entry for gwhost on SNA gateway:**

**DEST** : Name of the sideinfo entry (must match TP value in XCOM)

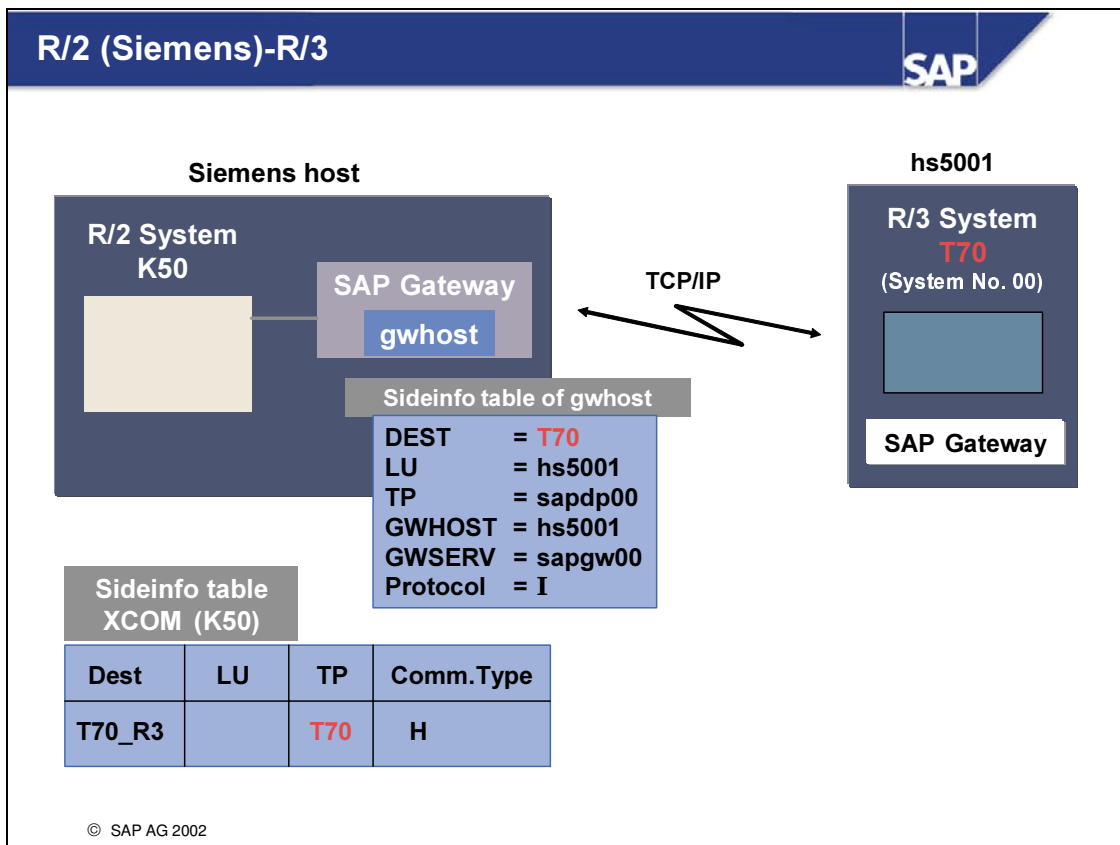
**Protocol** : I

**GWHOST / GWSERV** : Host and TCP/IP service of the required SAP gateway

**LU** : Application server of the R/3 System

**TP** : TCP/IP service of the R/3 dispatcher

- You can also operate the SNA gateway on the host side. The advantage of running the SNA gateway on the IBM host is that you can use existing TCP/IP and LU6.2 software for the gateway on the host side.



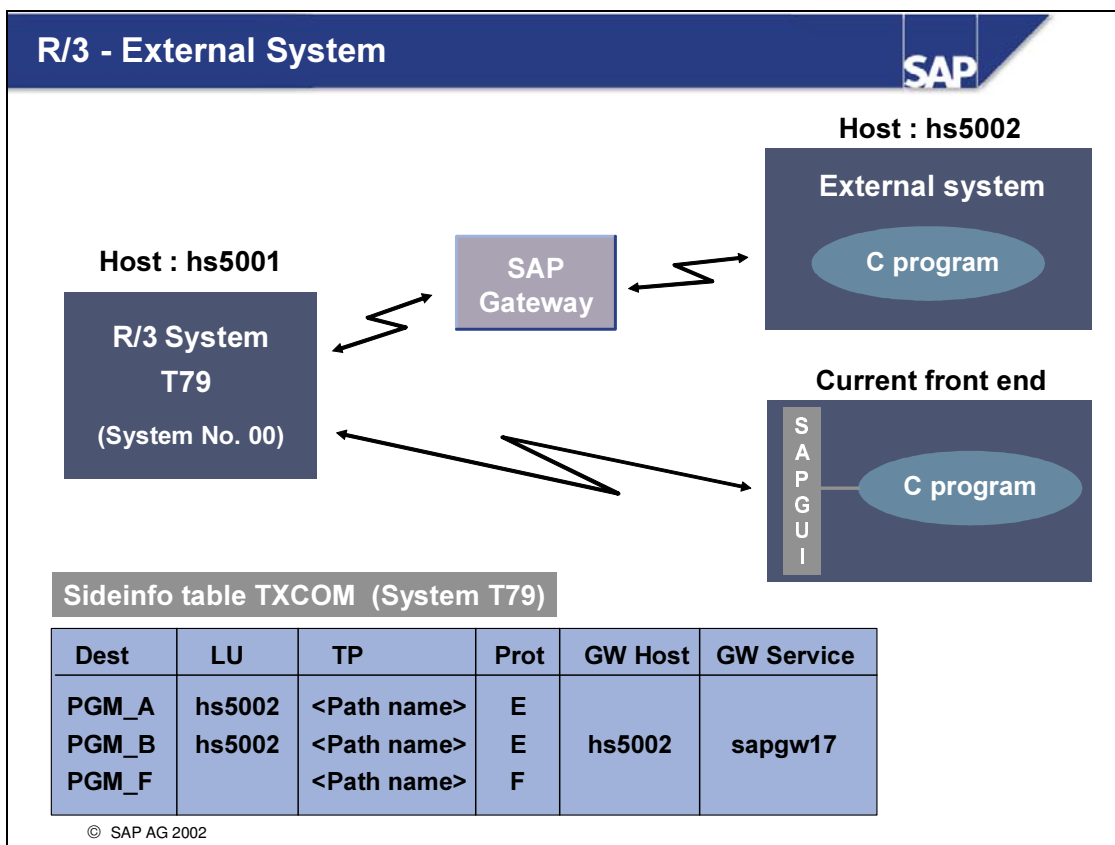
- For communication between Siemens R/2 and R/3, you need the host gateway that runs on the Siemens host.
- To start a connection from a Siemens R/2 System to an R/3 System, you use the program *gwhost* on the host gateway. For this, you need an entry in the sideinfo table of the R/2 System and one in the sideinfo table of gwhost.

#### Sideinfo entry in R/2 (table TXCOM)

**Dest** : Any logical name (name of the sideinfo entry)  
**LU** : Not important  
**TP** : Name of the corresponding gwhost sideinfo entry  
**Comm.Type** : H

#### Sideinfo entry for gwhost on host gateway:

**DEST** : Name of the sideinfo entry (must match TP value in XCOM)  
**Protocol** : I  
**GWHOST / GWSERV** : Host and TCP/IP service of the required SAP gateway  
**LU** : Application server of the R/3 System  
**TP** : TCP/IP service of the R/3 dispatcher



- **UNIX:** You call an external program over the SAP gateway using the user that started the gateway (for example, t79adm). This user must have the correct start authorization. If the SAP gateway does not run on the same host as the external program, it is started by *remote shell* from the gateway. The user who started the SAP gateway must also exist on the host of the external program. (The program is called using its name). The **.rhost** (remote shell authorization) on the remote machine must also have a corresponding entry.
- **OS/2 :** As with UNIX, but without user ID
- **NT :** As with OS/2, the SAP gateway must run on the same host as the external program because of the Windows NT remote shell function.
- **LU6.2 Systems:** You can also reach external LU6.2 systems using the SAP SNA gateway. For example, you can call a Cobol program running in CICS using CPI-C from an R/3 System. The corresponding sideinfo table is maintained in the same way as the R/3 -> R/2 (IBM) communication scenario that you see in the online documentation.
- **Windows 3.1 , WfW :**
  - No CPI-C
  - R C only at current front end

## Using CPI-C in Communications



Introduction

Remote Destinations



Using CPI-C in Communications

Calling an ABAP Program Remotely

Development and Test Environment

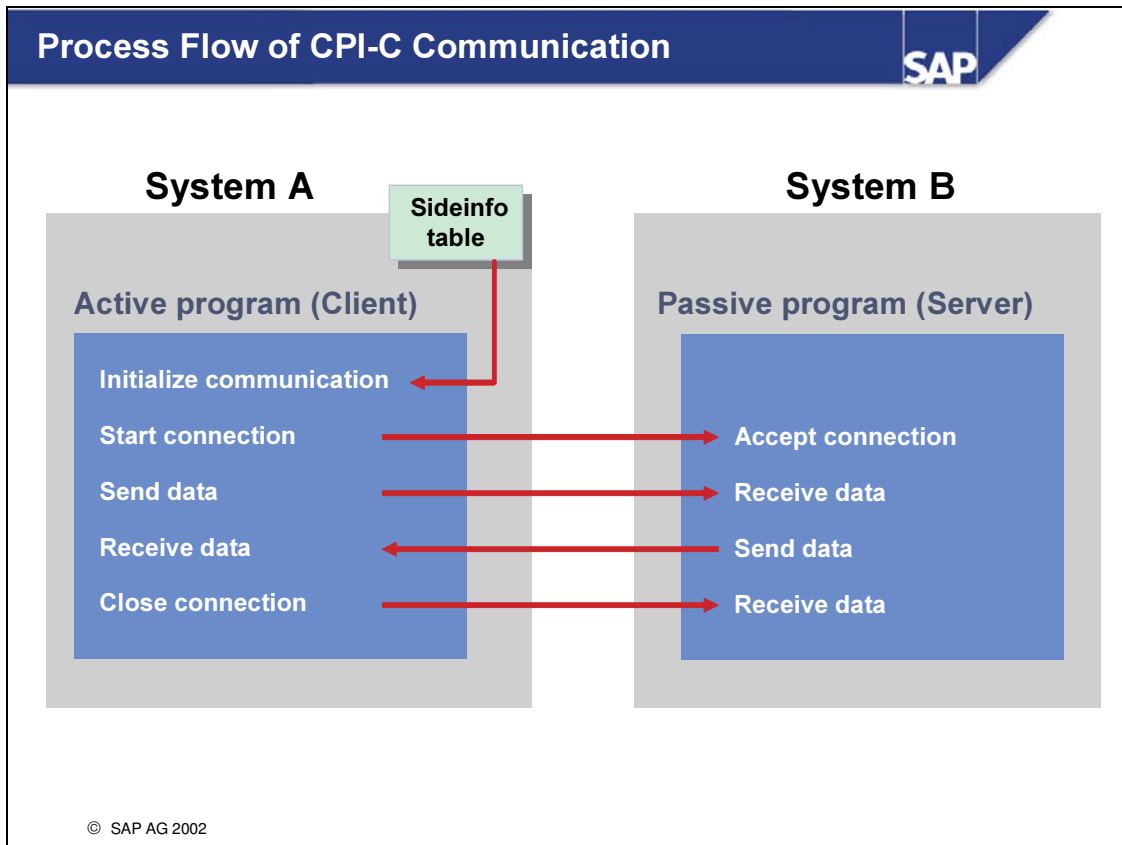
© SAP AG 2002



**At the conclusion of this topic, you will be able to:**

- **Learn the exact process flow of a CPI-C communication**
- **Identify the communication methods you can use and their advantages and disadvantages**
- **Implement communications in ABAP using CPI-C**
- **Use the commands for converting ASCII to EBCDIC**

© SAP AG 2002

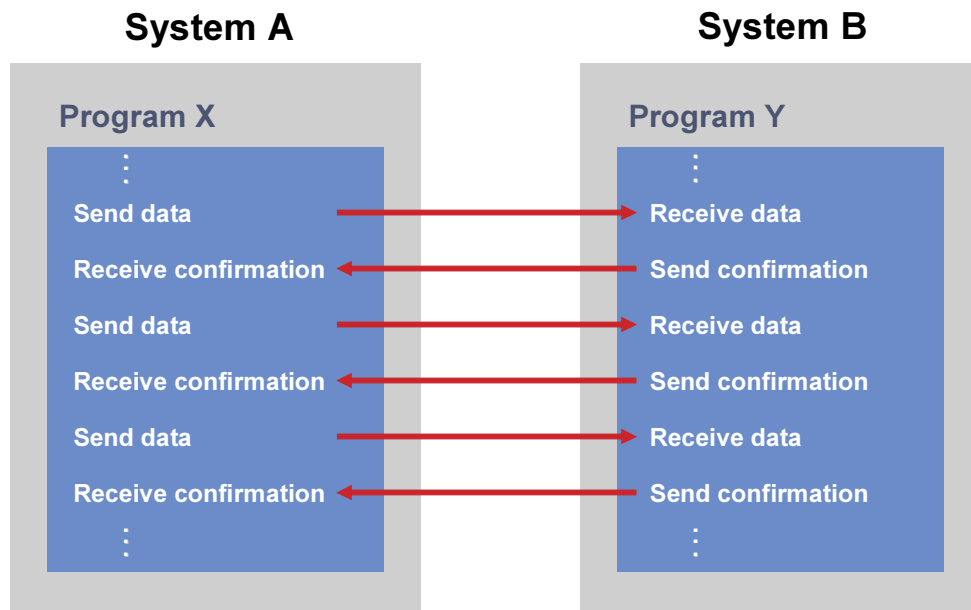


- The program that initiates the connection is called the active program or client. The program that is called is known as the passive program or server.
- To start a CPI-C connection, the active program must initialize it, that is, use a CPI-C call to read the necessary communication parameters from the corresponding sideinfo entry. These parameters are then stored under a conversation ID (output of the CPI-C call) and are used when the connection is made. After being accepted by the target program, the connection is available for data exchange.
- CPI-C is a semi-duplex communication method, that is, only one side may send data at a time. Once the passive program has accepted the communication, the active program may send data first, whereas the passive program must wait for data using a receive call.
- CPI-C transmission is asynchronous. In other words, the transmitting buffer is physically transmitted along with permission to send (!) only when the sending side submits its next receive command.
- The connection may only be closed by the program that is currently permitted to send data (otherwise data may be lost). The partner program waiting with the receive command then receives the corresponding return code (18).



## "Ping-Pong" Communication Method

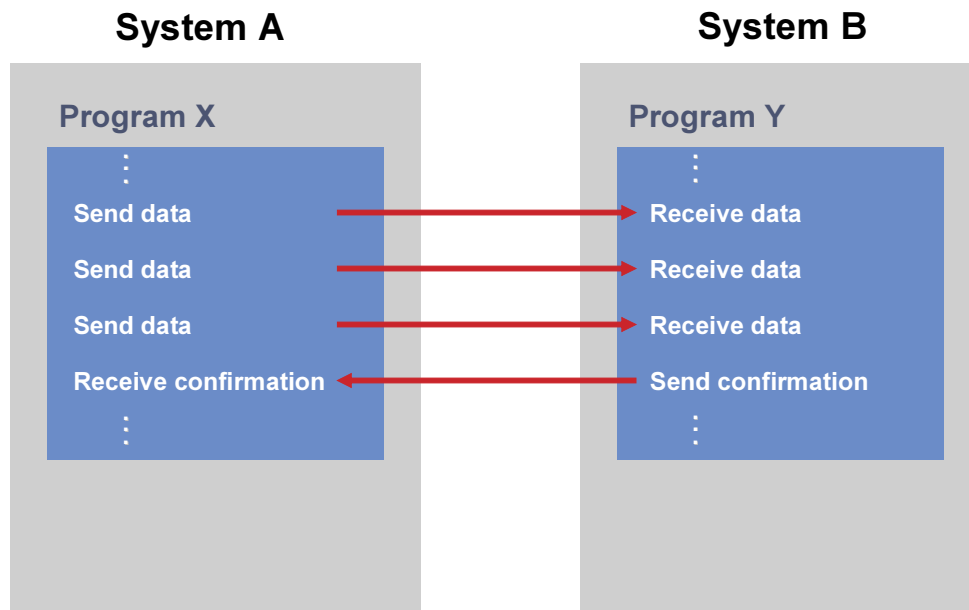
SAP



© SAP AG 2002

- The term *ping pong* describes the communication method illustrated above. With this method, a confirmation message is sent after each data record.

## "Multisend" Communication Method



© SAP AG 2002

- In contrast to the *ping-pong* method, *multisend* sends a series of data records before receiving a single confirmation at the end.
- **Comparison of the two communication methods:**  
 The *ping pong* method involves more work because you are dealing with single records and individual confirmations, whereas *multisend* uses a single asynchronous transmission, and only has to wait for one overall confirmation. However, *ping pong* provides you with better control of your data. If the communication is interrupted (communication error, or the receiving program crashes -> yielding return code 17), you always know, if you are using *ping pong*, which records have arrived in the target system, which have been processed, and where you need to resume processing. The *multisend* method, on the other hand, also returns code 17, but does not know which records have been received, and whether they have been fully or partially processed, or not processed at all.  
 You should, therefore, use the *ping pong* method where each data record contains the data for one transaction.

**COMMUNICATION INIT**  
**COMMUNICATION ALLOCATE**  
**COMMUNICATION ACCEPT**  
**COMMUNICATION SEND**  
**COMMUNICATION RECEIVE**  
**COMMUNICATION DEALLOCATE**

© SAP AG 2002

- You can execute the steps explained for CPI-C using the CPI-C calls specified previously:

**COMMUNICATION INIT** Initialize connection  
**COMMUNICATION ALLOCATE** Make (allocate) connection  
**COMMUNICATION ACCEPT** Accept connection  
**COMMUNICATION SEND** Send data  
**COMMUNICATION RECEIVE** Receive data  
**COMMUNICATION DEALLOCATE** Close connection

- External C programs use the same calls to communicate with ABAP programs using CPI-C. Although the syntax of the calls is different in C (CMINIT, CMALLC, CMACCP, CMSEND, CMRCV, CMDEAL), their functions are the same as the corresponding ABAP calls.

**ABAP include RSCPICDF**

```

DATA: INT2(2) TYPE X.
      .
      .
* return_code
DATA : CM_OK                                LIKE INT2  VALUE '0000',
      .
      CM_DEALLOCATED_ABEND                LIKE INT2  VALUE '0011',
      CM_DEALLOCATED_NORMAL                LIKE INT2  VALUE '0012',
      CM_PARAMETER_ERROR                   LIKE INT2  VALUE '0013',
      CM_PRODUCT_SPECIFIC_ERROR            LIKE INT2  VALUE '0014',
      .
      CM_PROGRAM_STATE_CHECK               LIKE INT2  VALUE '0019',
      .

```

© SAP AG 2002

- The return code of a CPI-C call in ABAP is always placed in the SY-SUBRC field. However, it can also be placed in the `RETURNCODE` parameter of the CPI-C call itself.
- In addition to the return code, some CPI-C calls also return special parameters containing extra information about data transfer and which side is entitled to send data.
- The return codes and return parameters of CPI-C calls are stored as hexadecimal code. When you query the codes, you should use meaningful constants instead of hexadecimal codes.  
For example, instead of `IF RC = 18` use `IF RC = CM_DEALLOCATED_NORMAL` to close the connection from the partner side.  
These CPI-C constants are contained in the standard ABAP include **RSCPICDF**.
- The most important return codes are:
  - 0000 ( `CM_OK` ) Connection OK
  - 0011 ( `CM_DEALLOCATED_ABEND` ) Connection terminated
  - 0012 ( `CM_DEALLOCATED_NORMAL` ) Connection from partner terminated normally
  - 0013 ( `CM_PARAMETER_ERROR` ) Invalid conversation ID in CPI-C call
  - 0014 ( `CM_PRODUCT_SPECIFIC_ERROR` ) Invalid sideinfo entry in CMINIT call
  - 0019 ( `CM_PROGRAM_STATE_CHECK` ) Send command rejected due to missing transmission permission

## COMMUNICATION INIT

```

        DESTINATION dest
        ID           convid
        [ RETURNCODE rc ].
    
```

© SAP AG 2002

- **COMMUNICATION INIT** is used to initialize the CPI-C connection.  
(Preparing the connection)

Entry:

**DESTINATION** Name of a sideinfo entry

Output:

**ID** Conversation ID: This indicates the communication partners for the specified sideinfo entry. It is used until the next connection is made and is, at the same time, the name of the connection.

**RETURNCODE** Return code of the call is always stored in SY-SUBRC; it can be used in additional fields.

This parameter is optional.

- The fields listed in the INIT call must have the following types:

dest C8

convid C8

rc X2

- Legend: Lowercase plus italics indicates ABAP field.  
Underlined indicates entry variable.  
[ ... ] indicates optional specification

⋮  
**COMMUNICATION ALLOCATE**

          ID          convid  
          [ RETURNCODE *rc* ].

⋮

© SAP AG 2002

- **COMMUNICATION ALLOCATE** builds the CPI-C connection and the corresponding partner program is started. You must specify the conversation ID returned by the INIT call.

### COMMUNICATION ACCEPT

```
          ID          convid  
[ RETURNCODE rc ].
```

```
...
```

© SAP AG 2002

- For a valid connection to be established, the called program must accept the connection using the **COMMUNICATION ACCEPT** statement. The ACCEPT call returns the conversation ID of the connection and a return code. This must be the first statement in the passive program.
- After the ACCEPT call from the side that was called, the active program has the required transmission permission. The passive program must first receive data.

**COMMUNICATION SEND**

**ID**                    *convid*  
**BUFFER**            *sendbuf*  
**[ LENGTH**            *sendlen* **]**  
**[ RETURNCODE** *rc* **]**.

© SAP AG 2002

- In the SEND command, you can transmit a data packet of up to 30000 bytes.

Parameters:

**ID**    Conversation ID**BUFFER**    Transmission buffer (max 30000 bytes)**LENGTH**    Length of the buffer you want to send (Optional: If entries are missing, the entire transmission buffer is sent.). The field must be type P.

Output:

RETURNCODE

- If a program sends data without having permission, the return code issued is 25 (hex 19).



⋮

## COMMUNICATION RECEIVE

|                   |                 |
|-------------------|-----------------|
| ID                | <u>convid</u>   |
| BUFFER            | rcvbuf          |
| [ LENGTH          | <u>rcvlen</u> ] |
| [ RECEIVED        | rcvdlen ]       |
| DATAINFO          | datainfo        |
| STATUSINFO        | statusinfo      |
| [ HOLD ]          |                 |
| [ RETURNCODE rc ] | .               |

⋮

© SAP AG 2002

- Your own previous SEND packages that are still in the buffer due to the asynchronous SEND, together with the transmission permission, are physically transmitted through the connection by the COMMUNICATION RECEIVE statement. The statement then waits for the data.

Parameters:

**ID** Conversation ID**Length** Use this parameter to specify which part of the specified receive buffer should be used. (Optional; if you do not specify a length, the system uses the whole receive buffer.) The field must be type P.**HOLD** This addition (without a field) causes the work process to be stopped (no rollout/rollin) while it waits from the RECEIVE. You would use this when using RECEIVE within a SELECT loop.

Output:

**BUFFER** Receiving buffer (contains data received)**RECEIVED** Number of character received. This field must be type X4.**DATAINFO** Information on whether the receive buffer was large enough for the incoming SEND packet. This field must be type X4.**STATUSINFO** Information on the transmission permission. This field must be type X4.**RETURNCODE**

## ABAP include RSCPICDF

```

DATA: INT4(4) TYPE X.
.
.
* data_received
DATA : CM_NO_DATA_RECEIVED           LIKE INT4 VALUE '00000000',
      CM_COMPLETE_DATA_RECEIVED      LIKE INT4 VALUE '00000002',
      CM_INCOMPLETE_DATA_RECEIVED    LIKE INT4 VALUE '00000003',
      .
      .
* status_received
DATA : CM_NO_STATUS_RECEIVED         LIKE INT4 VALUE '00000000',
      CM_SEND_RECEIVED              LIKE INT4 VALUE '00000001',
      .
      .

```

© SAP AG 2002

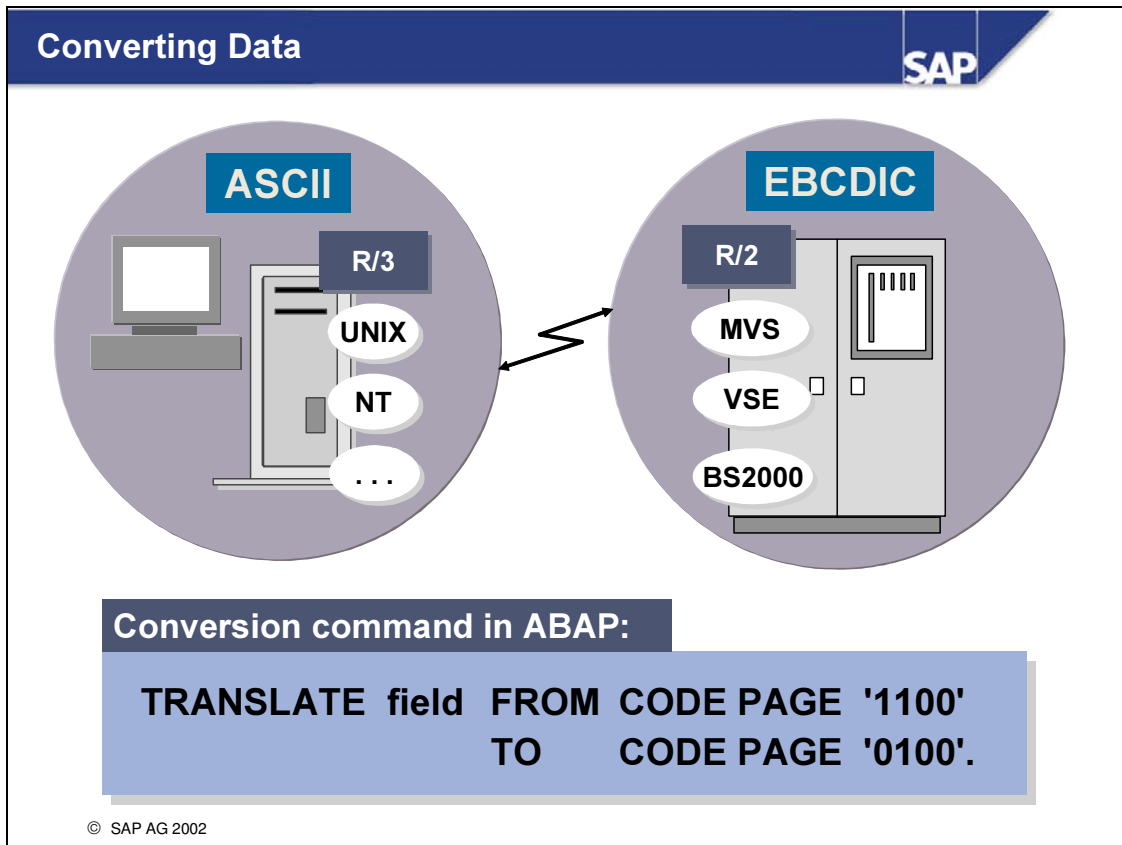
- The return parameter **DATAINFO** specifies whether data has been received and whether the receive buffer (or section of the buffer) used was large enough to receive the entire SEND packet. DATAINFO can have the following values:
  - CM\_NO\_DATA\_RECEIVED --> No data received
  - CM\_COMPLETE\_DATA\_RECEIVED --> Data received and RECEIVE buffer is large enough
  - CM\_INCOMPLETE\_DATA\_RECEIVED --> Data received but RECEIVE buffer too small to receive entire SEND packet. (In this case, the remaining segments must be transmitted as long as there are other RECEIVES to be received, until DATAINFO has the value CM\_COMPLETE\_DATA\_RECEIVED.)
- The return parameter **STATUSINFO** indicates whether transmission permission has also been received. STATUSINFO May have the following values:
  - CM\_NO\_STATUS\_RECEIVED --> No transmission permission received
  - CM\_SEND\_RECEIVED --> Transmission permission received

### COMMUNICATION DEALLOCATE

                  ID                  convid  
                  [ RETURNCODE rc ].

© SAP AG 2002

- You use this call to close the CPI-C connection. Any data in the local buffer that has not yet been sent is transmitted first and then the connection is closed.
- The connection should always be closed by the program that has transmission permission; otherwise, any data that has not yet been received is lost.



- Data conversion is necessary when two systems that use different character sets communicate with each other.
- To convert the character sets, you use the following ABAP command  
**TRANSLATE <field> FROM CODE PAGE '...' TO CODE PAGE '...'**  
 This command lets you convert character strings between different code pages. The contents of the field you specify is converted from the source code page representation to the target code representation.
- You can display and/or maintain the codepages in R/3 using transaction **SPAD**.

## Calling an ABAP Program Remotely



Introduction

Remote Destinations

Using CPI-C in Communications



Calling an ABAP Program Remotely

Development and Test Environment

© SAP AG 2002

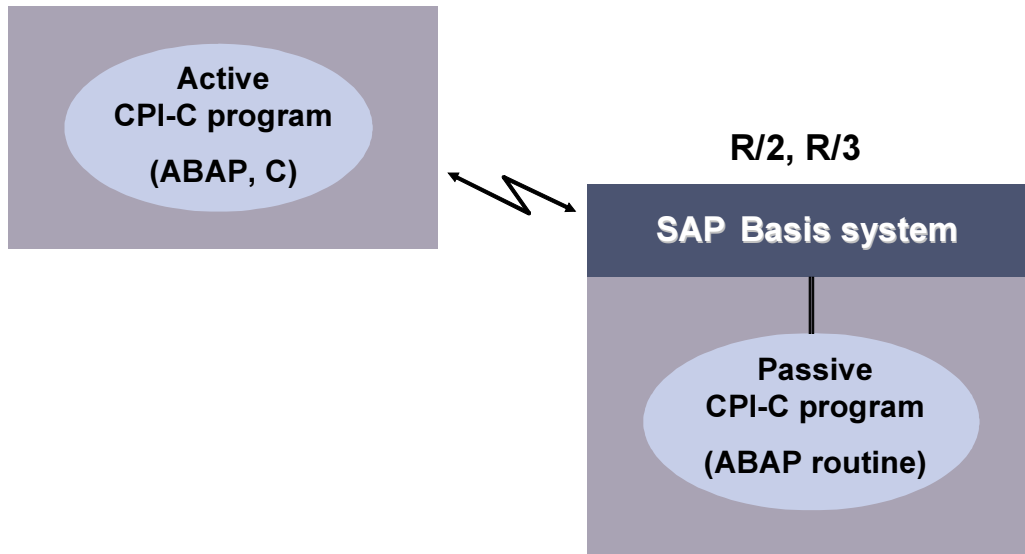
## Calling an ABAP Program Remotely: Topic Objectives



- At the conclusion of this topic, you will be able to call a passive ABAP program remotely.

© SAP AG 2002

### R/2, R/3, external system



© SAP AG 2002

- You can call ABAP programs (or more precisely, ABAP form routines) in an R/2 or an R/3 System remotely using CPI-C. You cannot communicate with them directly, however. The connection must be made through the SAP Basis System of the passive program.
- In the R/2 System, the SAP Basis System is the task handler, called using X1SA. In the SAP R/3 System, you address the dispatcher using sapdp##.

- 1. Initialize connection with remote system (INIT)**
- 2. Establish connection with remote system (ALLOCATE)\***
- 3. Send connect string to remote system (SEND)**
- 4. Wait for response from remote system (RECEIVE)**
- 5. With affirmative answer, communicate with remote ABAP routine (send permission on active side)**

- \*) - Contains SAP logon data for the remote system, and the name of the remote ABAP routine**
- Must always be sent in EBCDIC**

© SAP AG 2002

- The above procedure for calling a remote ABAP routine is the same, regardless of whether the active program is an ABAP program in an R/2 or an R/3 System, or a C program in an external system.



## The Connect String



### Dictionary structure

| Field   | Type | Length |
|---------|------|--------|
| REQID   | C    | 4      |
| REQTYPE | C    | 4      |
| AMODE   | C    | 1      |
| IMODE   | C    | 1      |
| FILL1   | C    | 2      |
| MAND    | C    | 3      |
| NAME    | C    | 12     |
| CODE    | C    | 8      |
| LANG    | C    | 1      |
| CORR    | C    | 1      |
| PROG    | C    | 8      |
| MODN    | C    | 30     |

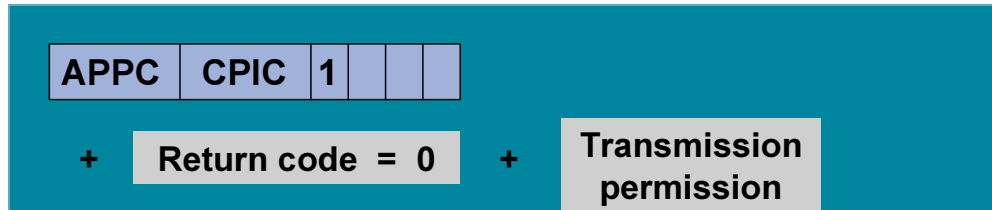
### Value

CONN  
 CPIC  
 1  
 < Not important >  
 < Not important >  
 < 3-character client >  
 < SAP user >  
 < Password >  
 < Logon language >  
 < Not important >  
 < ABAP program >  
 < Form routine >

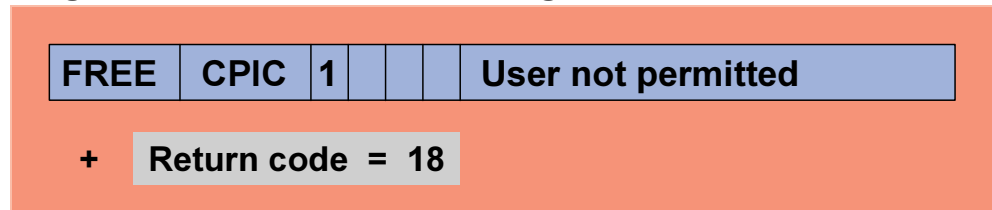
© SAP AG 2002

- The valid format for a connect string is contained in the ABAP Dictionary structure CPICCONN. By defining a structure using the latter and setting the fields with suitable values, you create your own connect string.  
If you are programming in C, use the call SAP\_CMLOGON to create a connect string in EBCDIC.
- All of the values in a connect string must be entered in uppercase.
- In the *NAME* field, enter a **user with type CPI-C** that is already defined in the relevant client of the remote system.
- The one-character language key is converted into the corresponding two-character key by the remote Basis system. This must be made known to the remote system.
- In the *PROG* field, enter the main program of the remote routine. The syntax must be correct. The name of this main program can have a maximum of 8 characters.
- The form routine that you enter in the *MODN* field must exist within the specified main program.

### Positive Confirmation Message



### Negative Confirmation Message



© SAP AG 2002

- If the remote Basis system's check on the connect string is positive (valid logon data, main program and form routine are syntactically correct and exist in the target system), the active program receives the 12-byte EBCDIC string APPCCPIC1 with return code 0 and the transmission permission. As of this point, the system can communicate directly with the called ABAP routine. The data format is completely open.
- If the checks fail, the remote Basis system sends a corresponding error message with the 12-byte header FREECPIC1 in EBCDIC and terminates the connection.

```
REPORT XYZ .
INCLUDE RSCPICDF.
TABLES CPICCONN.
  ⋮
CPICCONN-REQID = 'CONN' .
  ⋮
CPICCONN-PROG  = 'ZZZ' .
CPICCONN-MODN  = 'ABC' .
TRANSLATE CPICCONN FROM CODE PAGE '1100'
                  TO   CODE PAGE '0100' .

COMMUNICATION INIT ...
COMMUNICATION ALLOCATE ...
COMMUNICATION SEND  BUFFER CPICCONN ...
COMMUNICATION RECEIVE BUFFER RCVBUF ...
IF RETURNCODE NE CM_OK.  EXIT.  ENDIF.

COMMUNICATION SEND ...      " Send application data
COMMUNICATION RECEIVE ...    " Receive application data
```

© SAP AG 2002

- The above coding example shows an active ABAP program that calls an ABAP form routine remotely in another SAP system (R/2 or R/3) using CPI-C.

```
REPORT ZZZ .  
INCLUDE RSCPICDF.  
  :  
FORM ABC .  
  COMMUNICATION ACCEPT ...  
  COMMUNICATION RECEIVE ... " Receive application data  
  COMMUNICATION SEND ...   " Send application data  
  :  
ENDFORM .
```

© SAP AG 2002

- This coding example shows an ABAP program containing an ABAP routine that can be called remotely.
- The main program may contain more than one subroutine that can be called remotely. However, only the subroutine specified in the connect string is executed.

Introduction

Remote Destinations

Using CPI-C in Communications

Calling an ABAP Program Remotely



Development and Test Environment

© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only



**At the conclusion of this topic, you will be able to:**

- **Use the CPI-C test programs supplied by SAP in the standard R/3 System**
- **Analyze CPI-C programs and connections using the tools provided**

© SAP AG 2002

## The SAP CPI-C-Software Development Kit (SDK)



**R/3**

### ABAP language scope

```
COMMUNICATION INIT ...  
COMMUNICATION ALLOCATE ...  
COMMUNICATION ACCEPT ...  
COMMUNICATION SEND ...  
COMMUNICATION RECEIVE ...  
COMMUNICATION DEALLOCATE ...
```

**External  
system**

### CPI-C-SDK

```
CINIT (...)  
CMALLC (...)  
CMACCP (...)  
CMSEND (...)  
CMRCV (...)  
CMDEAL (...)  
...
```

**Sample  
programs**

© SAP AG 2002

- In R/3, the CPI-C calls are contained in the language scope of ABAP.
- However, on external platforms you must install the CPI-C library (CPI-C-SDK). For information on how to use the CPI-C calls, refer to the text files and sample programs delivered with the SDK.

## R/2 , R/3

## ABAP programs

**ACPICT1**

(active)

**ACPICT2**

(passive)

## External system

## C programs

**ccpict1t**

(active)

**ccpict2t**

(passive)

© SAP AG 2002

- The standard system contains the CPI-C test programs *ACPICT1* and *ACPICT2* (in R/2 as of release 5.0G). *ACPICT1* is an active CPI-C program, *ACPICT2* is a passive CPI-C program. You can use them to test R/3-R/3 and R/3-R/2 CPI-C connections.
- SAP also supplies the SAP CPI-C SDK (Software Development Kit) with every SAP R/3 System. You can use this kit to develop CPI-C programs in C. The SDK contains the C programs *ccpict1t* (active) and *ccpict2t* (passive). You can use these programs to test connections between two external systems, but, above all, to test connections between external systems and an R/3 System. *ccpict1t* can call not only *ccpict2t* remotely, but also the passive ABAP program *ACPICT2*. Likewise, the active ABAP program *ACPICT1* can call the passive C program *ccpict2t*.
- When you run *ACPICT1*, you must specify:
  - If *ACPICT2* or *ccpict2t* should be called remotely (parameter 'ABAP') ,
  - If test data should be converted from ASCII to EBCDIC (parameter 'CONVERT').
 If you call *ACPICT2*, you must also enter the logon data for the connect string on the selection screen of *ACPICT1*.



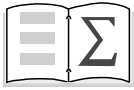
**ABAP Debugger**

**Gateway Monitor (SMGW)**

**System Log (SM21)**

© SAP AG 2002

- You can use the Debugger to analyze an active ABAP CPIC-C program in R/3. This does not apply in R/2 since screen changes are not allowed in R/2 during a CPI-C connection (Debugging involves screen changes). You cannot analyze passive programs in the Debugger.
- You can use the Gateway monitor (transaction SMGW) to monitor SAP Gateway activity (existing CPI-C connections) and generate and display Gateway traces.
- One of the tasks of the system log is to record failed CPI-C logons that have been sent from remote systems to the current system. You can display the system log using transaction SM21.



- The CPI-C interface allows programs in R/3, R/2, and external systems to communicate and exchange data with one another.
- The communication always runs using an SAP Gateway.
- To start a connection, the active program requires a destination (entry in the side information table).
- If the target system is an SAP system, the active program must send the required logon data and the name of the form routine it wants to call as a connect string.
- There are many CPI-C test programs and analysis tools available.

© SAP AG 2002

Maintaining an RFC Destination in R/2

Web RFC

The CPI-C Communication Interface



Asynchronous Data Transfer through Q-API)

Exercises / Solutions for CPI-C and Q-API ;  
BAPI Workshop

© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only

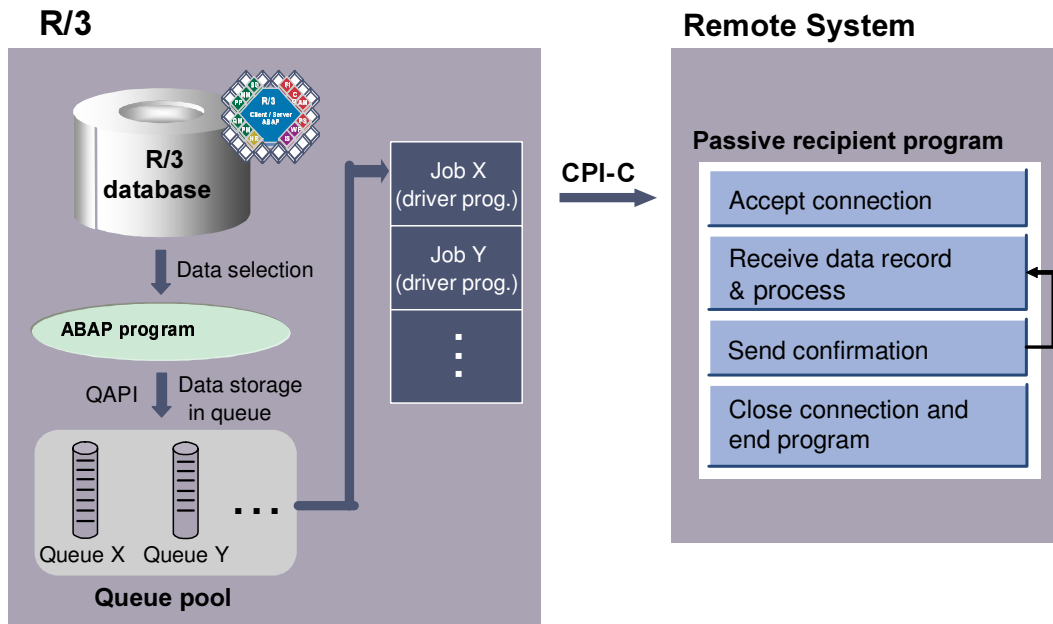
## Asynchronous Data Transfer Through QAPI: Topic Objectives



**At the conclusion of this unit, you will be able to**

- **Transmit (buffered) data asynchronously**
- **Generate a queue (locally deposited data that is to be sent asynchronously)**
- **Manage queues**

© SAP AG 2002



© SAP AG 2002

- One of the prerequisites of CPI-C communication is that the recipient system be available. Asynchronous data transmission (or buffered data transmission) is useful for cases where the recipient system is unavailable or overloaded (and so unable to receive and process data).
- The data that is to be sent is first placed in a queue. Then a job is automatically scheduled that starts an active CPI-C send program (driver program) at the set time to read and send the queue data. You specify the recipient passive program in the queue attributes, which is then called at the appropriate time to receive and process the data.

## Automatic Scheduling of a Driver Program

**R/3**

**Queue pool**

Queue X

Automatic scheduling of the driver program when a queue is created

**Job X**

Start time : 20:15:00  
Start date : 24.12.98  
Step : RSQAPI20

**Queue attributes**

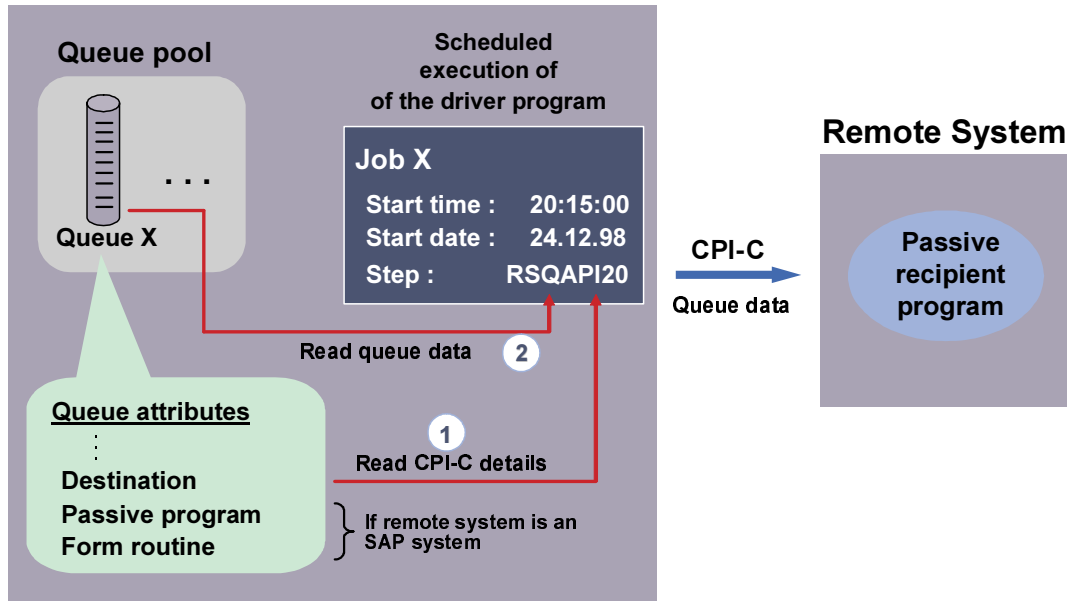
Start mode 'A'  
Send time '201500'  
Send date '19981224'  
Driver prog. 'RSQAPI20'

© SAP AG 2002

- The system only schedules the transmission job for queues with the value A (automatic transmission) in the *Start mode* attribute.

The job has the same name as the queue, and is scheduled for the time and date specified in the 'Transmission date/time' attribute. The only step in the job is to call the active CPI-C program specified in the *Driver program* attribute of the queue.

## R/3



© SAP AG 2002

- The specified driver program is started automatically at the chosen start time. This active CPI-C program calls the relevant recipient program and sends the queue to it. The driver program reads the relevant parameters for the CPI-C communication from the corresponding queue attributes.
- The SAP standard driver program **RSQAPI20** has the following properties:
  - If the connection to the remote system cannot be established, the driver program schedules itself again (-> new job). You can set the interval between attempts using the constant **DELAYTIME** in the driver program (300 sec.).
  - If the **CLIENT** attribute (client in remote SAP system) is set; the corresponding CPI-C connect string is constructed and sent automatically after establishing the connection with the remote system.
  - The queue records are then read and sent in *ping pong* mode.
  - LUWs that have been sent successfully are deleted from the queue (LUW = records that belong together).
  - If the connection is terminated when the data is transmitted, the current LUW in the queue is rolled back, the queue status is set to E, and an express e-mail is sent to the queue creator.
  - (In this error case, the recipient program must roll back the current LUW in the remote system itself.).
  - If required, you can adapt the driver program to your own needs, or write your own driver program for use in queue transmission.
- You must create a passive recipient program in the remote system that can receive and process the queue data.

## Creating a Queue



- 1 Open queue (generate queue header)**
- 2 Place records of first LUW into queue one by one  
+ COMMIT WORK (fixes LUW in queue)**  
  
**Place records of second LUW into queue one by one  
+ COMMIT WORK (fixes LUW in queue)**  
  
⋮
- 3 Close queue followed by COMMIT WORK (fixes queue)**

© SAP AG 2002

- The above is the procedure for creating a queue.



## Function module QUEUE\_OPEN

### Import parameters

**NAME**  
**OPENMODE**  
**ERASE**  
**TYPE**  
**START**  
**DATE**  
**TIME**  
**:**  
**:**

### Values

< Queue name >  
 'W' , 'R'  
 'X' , SPACE  
 'A' , 'U'  
 'A' , 'M' , 'E' ( start mode )  
 < Send date >  
 < Send time >

© SAP AG 2002

- When you open a queue using the function module QUEUE\_OPEN, you must supply the following parameters:
  - NAME** Name of queue to be opened  
This parameter is not specified when opening a new queue. The queue, therefore, receives the current time stamp as its name, which you receive from the export parameter NAME of the function module.
  - OPENMODE** W Write ; R: Read
  - ERASE** X Deletes the queue shell when the queue is empty
  - TYPE** U : Unique (queue can only be filled with data once; it cannot be extended subsequently)
  - Appendable (queue may be extended, even during transmission)
  - START** A Automatic (queue data is automatically sent through an internally scheduled job at the specified send time (DATE / TIME).
  - M** Manual (queue must be sent manually using transaction SM38) -> internal scheduling of an immediate job)
  - E** Event-controlled (used to start transmission immediately once the queue has been created.) To do this, you must manually create an appendable queue first from start mode E using transaction SM38. When the queue is activated, a job is created internally that is executed at the event SAP-
  - QEVENT** / parameter <queue name>. If you place data from this program into this queue, the accompanying event is triggered automatically at QUEUE\_CLOSE. The scheduled job for sending data commences.
  - DATE/TIME** Transmission start time and date (only applies to start type A).

## Function module QUEUE\_OPEN

### Import parameters

...

**DATATYPE**  
**DRIVER**  
**DESTINATION**  
**CLIENT**  
**USERID**  
**PASSWORD**  
**PROGRAM**  
**FORM**

### Values

**SPACE , ' CPIC ' , ' RODC '**  
 < Driver program >  
 < Side info entry for target system >  
 < Client in target system >  
 < User in target system >  
 < Password of remote user >  
 < Passive recipient program >  
 < Form routine in recipient program>

© SAP AG 2002

- Further import parameters of function module QUEUE\_OPEN:
  - DATATYPE: This queue attribute is interpreted by the driver program RSQAPI20.
  - SPACE : Queue data is transmitted unconverted.
  - CPIC: If the sending system uses ASCII, the queue data is converted from ASCII to EBCDIC before it is sent.
  - RODC: Queue data is sent by CPI-C using RODC protocol (only when sending to R/2).
  - DRIVER Name of driver program (for example RSQAPI20)
  - DESTINATION TXCOM entry (for making connection to receiving system by driver)

Only enter the following parameters if the receiving system is an SAP System:

- CLIENT Client in receiving system (with CLIENT values, the driver RSQAPI20 automatically sends a connect string.)
  - USERID / PASSWORD Logon data for the receiving system
  - PROGRAM / FORM Receiving program or form routine called remotely from the driver program
- The input parameters of the function module QUEUE\_OPEN are stored in the queue header as queue attributes.
- The parameters of the QUEUE\_\* function modules are all defined with reference to ABAP Dictionary fields. Therefore, the corresponding actual parameters must all have the same types.

## Function module QUEUE\_PUT

### Import parameters

**NAME**

**STATE**

**BUFFER**

**LENGTH**

### Values

< Queue name >

'S' ( single record of LUW )

'F' ( First record of LUW )

SPACE ( Middle record of LUW )

'L' ( Last record of LUW )

< Record contents

> Record length >

© SAP AG 2002

- The function module QUEUE\_PUT requires the following input parameters:
  - NAME Name of the queue into which you want to place the record.
  - STATE Position of the current record within its LUW:
    - S: Current record is the only record in the LUW
    - F Current record is the first (but not only) record in the LUW
    - SPACE Current record is neither the first nor the last record in the LUW
    - L Current record is the last of several records in the LUW
  - BUFFER Current record
  - LENGTH Length of current record

### Function module QUEUE\_CLOSE

#### Import parameters

**NAME**

**OPENMODE**

#### Values

< Queue name >

< Open mode used to  
open the queue ( 'W' / 'R' ) >

© SAP AG 2002

- The function module QUEUE\_CLOSE requires the following input parameters:
  - NAME Name queue to be closed
  - OPENMODE Open mode used when queue was opened
- If you created the queue with start mode A (automatic transmission), the corresponding job is scheduled automatically when you call QUEUE\_CLOSE. The job name is the same as the queue name.

### Function modules

- **QUEUE\_GET**
- **QUEUE\_DELETE**
- **QUEUE\_ERASE**

### Function

Reads a record from a queue

Deletes an LUW from the queue

Delete queue  
(and any data in it)

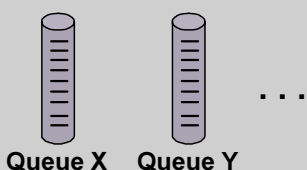
© SAP AG 2002

- If you modify driver program RSQAPI20 or write your own driver program, you can use the above function modules to read or delete queues. For further information about their use, refer to their documentation in the Function Builder.

R/3

SM38

Queue pool

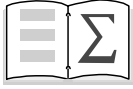


- List queues
- Display/change queue attributes
- Display queue data
- Start queue transmission
- Display transmission log
- Delete queue
- Create queues with no data

© SAP AG 2002

- You can manage queues in the queue pool using transaction SM38.
- The standard R/3 System contains the following demonstration programs, which show how to create a queue and send and received data:
  - Creating a queue: **SAPBC415D\_CREATE\_QUEUE** or **RSQAPI10**
  - Data transmission: **RSQAPI20** (Standard driver program)
  - Data transmission: **RCVQDATP** (Passive recipient program in the remote system)
- You can also use asynchronous (buffered) data transmission from an R/2 System. For further information, refer to the online documentation for *CPI-C / Asynchronous Data Transmission*.

## Asynchronous Data Transfer Through QAPI: Topic Summary



- Asynchronous data transmission allows you to prepare and send data separately.
- The data is first placed in a queue that is sent by a driver program to the appropriate recipient program at the scheduled time.
- You use the function modules QUEUE\_OPEN, QUEUE\_PUT, and QUEUE\_CLOSE to create a queue.
- The standard system contains driver program RSQAPI20. This program contains many useful functions, and can serve as a template for your own driver programs.
- You must create a passive recipient program in the remote system to correspond to your driver program.
- Transaction SM38 contains functions that allow you to manage your queues.

© SAP AG 2002

Internal Use SAP Partner Only

Internal Use SAP Partner Only

## Exercises and Solutions for CPI-C and Q-API; BAPI Workshop



Maintaining an RFC Destination in R/2

Web RFC

The CPI-C Communication Interface

Asynchronous Data Transfer through Q-API

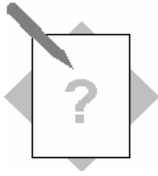


Exercises and Solutions for CPI-C and Q-API;  
BAPI Workshop

© SAP AG 2002

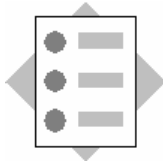


# Exercises

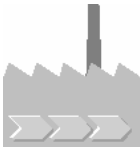


## Unit: The CPI-C Communication Interface

### Topic: CPI-C Communication Between Two ABAP Programs



- Creating a remote destination in the active system
- Developing active and passive ABAP programs



You have two R/3 Systems and want to use CPI-C communication to transmit data from one to the other.

- 1-1 Use CPI-C to link two R/3 Systems and transfer data from the passive system to the active system.  
(Your instructor will tell you the names of the active and passive systems).
- 1-1-1 In the active system, create a remote destination using transaction SM54. Destination name: **DEST##** ( where ## is your group number).
- 1-1-2 In the active system, copy the main program **SAPBC415T\_ACTIVE\_CPIC\_PGM** to program name **Z415\_A##** ( where ## is your group number), and convert it into an active CPI-C program that:
  - Prompts the user to enter a destination (DEST##) and an airline code (such as 'AA')
  - Calls the SEND form routine of program Z415\_P## remotely  
(## = group number. You must create the program in the passive system yourself)
  - Sends the airline code to the partner program
  - Receives the relevant SPFLI entries in *ping-pong* mode, stores them, and then displays them.
- 1-3 In the passive system, copy the main program **SAPBC415T\_PASSIVE\_CPIC\_PGM** to your program **Z415\_P##** (where ## is your group number), and extend the form routine SEND so that it is a passive CPI-C partner. When it is called, the routine should:
  - Receive an airline code
  - Read the corresponding SPFLI entries
  - Send the SPFLI entries to the caller in *ping-pong* mode
  - Terminate the CPI-C connection when finished.

*Tips & Tricks*

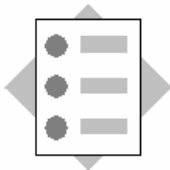


Find out the information required to create the sideinfo entry in the active system, use transaction SM51 (in the passive system).

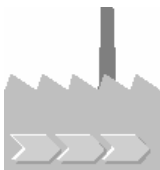


## Unit: The CPI-C Communication Interface

### Topic: CPI-C Communication Between Two ABAP Programs



- Creating a remote destination in the active system
- Developing active and passive ABAP programs



You have two R/3 Systems and want to use CPI-C communication to transmit data from one to the other.

```

*****
*                                                                 *
*      Unit: The CPI-C Communication Interface                    *
*                                                                 *
*      Model solution for exercise 1                             *
*      ( Active program )                                        *
*                                                                 *
*****

REPORT SAPBC415S_ACTIVE_CPIC_PGM.

TABLES SPFLI.           " Receive buffer for an SPFLI record
TABLES CPICCONN.        " Connect string
INCLUDE RSCPICDF.        " CPI-C constants

* User details (CPI-C destination, required SPFLI records)
PARAMETERS: DEST(8) DEFAULT 'DEST00',
            CARRID LIKE SPFLI-CARRID DEFAULT 'LH'.

```

```

* CPI-C variables
DATA: CONVID(8),           " Conversation ID
      RECEIPT(40),         " Receipt: Record received
      RBUFFER(255),        " Receive buffer for logon response
      RL(4) TYPE X,         " Length of data received
      DI(4) TYPE X,         " Is the received data complete
      SI(4) TYPE X,         " Info about send permission
      CPIC_RC LIKE SY-SUBRC, " Return code for CPI-C calls
      COUNTER(2) TYPE P.    " Counter for SPFLI records received

```

```

* Internal table for storing SPFLI records received
DATA: ITAB LIKE SPFLI OCCURS 0.

```

```

* Fill the connect string
CPICCONN-REQID   = 'CONN'.    " Request ID
CPICCONN-REQTYPE = 'CPIC'.    " Connection type
CPICCONN-AMODE   = '1'.       " Mode
CPICCONN-IMODE   = ' '.       " Unused
CPICCONN-FILL1   = ' '.       " Unused
CPICCONN-MAND    = '401'.     " Client
CPICCONN-NAME    = 'SAPCPIC'. " CPI-C user in remote system
CPICCONN-CODE    = 'ADMIN'.   " Password of CPI-C user
CPICCONN-LANG    = SY-LANGU.  " Logon language
CPICCONN-KORR    = ' '.       " Unused
CPICCONN-PROG    = 'CPICS33P'. " Partner ABAP program
CPICCONN-MODN    = 'SEND'.    " FORM routine

```

```

* Initialize communication
SKIP.  FORMAT COLOR 1.
WRITE: / 'COMMUNICATION INIT :  SYMDEST =', DEST.

```

```

COMMUNICATION INIT
      DESTINATION DEST
      ID           CONVID
      RETURNCODE CPIC_RC.

```

```

FORMAT COLOR 2.
WRITE: / 'RC =', CPIC_RC.

```

```

IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.      " Simple error handling
WRITE: / ' ==>  CONVID =', CONVID.

```

```

* Start communication
SKIP.  FORMAT COLOR 1.
WRITE: / 'COMMUNICATION ALLOCATE :  CONVID =', CONVID.

```

```

COMMUNICATION ALLOCATE
          ID          CONVID
          RETURNCODE CPIC_RC.

```

```

FORMAT COLOR 2.
WRITE: / 'RC =', CPIC_RC.
IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.      " Simple error handling

```

```

* Send logon data
SKIP.  FORMAT COLOR 1.
WRITE: / 'COMMUNICATION SEND (Logon-Data) : '.
FORMAT COLOR 2.  WRITE CPICCONN.

```

```

TRANSLATE CPICCONN FROM CODE PAGE '1100'
          TO CODE PAGE '0100'.

```

```

COMMUNICATION SEND
          ID          CONVID
          BUFFER      CPICCONN
          RETURNCODE CPIC_RC.

```

```

WRITE: / 'RC =', CPIC_RC.
IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.      " Simple error handling

```

```

* Logon response received
SKIP.  FORMAT COLOR 1.
WRITE: / 'COMMUNICATION RECEIVE (Logon response) :'.

```

```

COMMUNICATION  RECEIVE
          ID          CONVID
          BUFFER      RBUFFER

```

```

RECEIVED    RL
DATAINFO    DI
STATUSINFO  SI
RETURNCODE  CPIC_RC.

```

```

TRANSLATE RBUFFER FROM CODE PAGE '0100'
          TO CODE PAGE '1100'.

```

```

FORMAT COLOR 2.

```

```

WRITE: /    RBUFFER,
        /2 'RL =' , RL,
        /2 'SI =' , SI,
        /2 'DI =' , DI,
        /2 'RC =' , CPIC_RC.

```

```

IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.      " Simple error handling

```

```

* Send CARRID

```

```

SKIP.  FORMAT COLOR 1.

```

```

WRITE: / 'COMMUNICATION SEND (CARRID) :'.

```

```

COMMUNICATION SEND

```

```

          ID          CONVID
          BUFFER      CARRID
          RETURNCODE  CPIC_RC.

```

```

FORMAT COLOR 2.

```

```

WRITE: /    CARRID,
        /2 'RC =' , CPIC_RC.

```

```

IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.      " Simple error handling

```

```

* Loop for receiving and processing data and sending acknowledgement
* in ping pong mode:
*   In each loop pass
*     - One SPFLI record is received
*     - appended to ITAB (processing) and
*     - a receipt is sent to the sender.

```

```

DO.

```

```

* Receive SPFLI record
SKIP.  FORMAT COLOR 1.
WRITE: / 'COMMUNICATION RECEIVE (SPFLI-Entry) : ...'.

COMMUNICATION  RECEIVE
                ID          CONVID
                BUFFER      SPFLI
                RECEIVED    RL
                DATAINFO   DI
                STATUSINFO  SI
                RETURNCODE  CPIC_RC.

IF CPIC_RC NE CM_OK.                                " If connection is
    FORMAT COLOR 2.                                " terminated by
    WRITE / '->  CPI-C connection deallocated.'.    " partner exit from
    EXIT.                                           " DO loop
ENDIF.                                              " Loop

* Append SPFLI record to itab
APPEND SPFLI TO ITAB.
ADD 1 TO COUNTER.

FORMAT COLOR 2.
WRITE: / '->',
        3 COUNTER,
        6 '. SPFLI-entry received and processed.',
        /2 'RL =' , RL,
        /2 'SI =' , SI,
        /2 'DI =' , DI,
        /2 'RC =' , CPIC_RC.

* Send receipt
SKIP.  FORMAT COLOR 1.
WRITE: / 'COMMUNICATION SEND (Receipt)'.

RECEIPT = '      . entry received and processed.'.
MOVE COUNTER TO RECEIPT(4).

COMMUNICATION SEND
                ID          CONVID

```

```

        BUFFER      RECEIPT
        RETURNCODE CPIC_RC.

```

```

        FORMAT COLOR 2.
        WRITE: /2 'RC =', CPIC_RC.
        IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.      " Simple error handling

ENDDO.

```

```

* Display itab
SKIP 2.  FORMAT COLOR 3.
WRITE / 'Received SPFLI entries :'.
FORMAT COLOR 5.

LOOP AT ITAB INTO SPFLI.
    WRITE: /3 SPFLI-CARRID, SPFLI-CONNID, SPFLI-CITYFROM, SPFLI-CITYTO.
ENDLOOP.

```

```

*****
*                                                                 *
*      Unit: The CPI-C Communication Interface                    *
*                                                                 *
*      Model solution for exercise 1                              *
*      ( Passive program )                                        *
*                                                                 *
*****

```

```

REPORT CPICS33P .

```

```

TABLES SPFLI.

```

```

INCLUDE RSCPICDF.

```

```

DATA: CONVID(8),          " Conversation ID
      RL(4) TYPE X,        " Length of data received
      DI(4) TYPE X,        " Is the received data complete
      SI(4) TYPE X,        " Info about send permission
      CPIC_RC LIKE SY-SUBRC, " Return code for CPI-C calls

```



CARRID LIKE SPFLI-CARRID, " Receive field for SPFLI-CARRID  
RECEIPT(40). " Receive field for receipt

DATA: ITAB LIKE SPFLI OCCURS 0.

FORM SEND.

COMMUNICATION ACCEPT  
ID CONVID  
RETURNCODE CPIC\_RC.

IF CPIC\_RC NE CM\_OK. EXIT. ENDIF. " Simple error handling

\* Receive SPFLI-CARRID from active program

COMMUNICATION RECEIVE  
ID CONVID  
BUFFER CARRID  
RECEIVED RL  
DATAINFO DI  
STATUSINFO SI  
RETURNCODE CPIC\_RC.

IF CPIC\_RC NE CM\_OK. EXIT. ENDIF. " Simple error handling

\* Read SPFLI records into itab using array fetch

SELECT \* FROM SPFLI  
INTO TABLE ITAB  
WHERE CARRID = CARRID.

\* send contents of itab record by record in ping pong mode

LOOP AT ITAB INTO SPFLI.

COMMUNICATION SEND  
ID CONVID

```
        BUFFER      SPFLI
        RETURNCODE CPIC_RC.

IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.    " Simple error handling

COMMUNICATION RECEIVE

        ID          CONVID
        BUFFER      RECEIPT
        RECEIVED    RL
        DATAINFO   DI
        STATUSINFO  SI
        RETURNCODE CPIC_RC.

IF CPIC_RC NE CM_OK.  EXIT.  ENDIF.    " Simple error handling

ENDLOOP.

* Close connection

COMMUNICATION DEALLOCATE

        ID          CONVID
        RETURNCODE CPIC_RC.

ENDFORM.
```

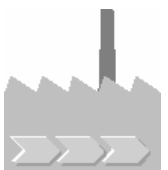


## Unit: Asynchronous Data Transfer (Q-API)

### Topic: Asynchronous Data Transfer between R/3 Systems



- Creating a queue for automatic transmission in the source system
- Queue administration
- Checking the processing results



You want to send data from one system to another system, but asynchronously, because the target system is currently not available or overloaded.

- 1-1 In this exercise, you will pass data between two R/3 Systems using a buffer in the source system that is then transmitted to the target system.  
(Your instructor will tell you the names of the source and target systems).
- 1-1-1 In the source system, copy the main program **SAPBC415T\_CREATE\_QUEUE** to your program **ZBC415\_##\_CREATE\_QUEUE** (## = your group number), and convert it into a program that generates a queue.

The queue should have the following attributes:

- Name **Q##** (## = group number)
- Extendable
- Automatic transmission at your chosen time using the driver program **RSQAPI21** (a slightly modified version of **RSQAPI20**, with the same functions as the original version).
- Recipient program and form routine **RCVQDATP / RECEIVE** (already exist in target system).
- CPI-C destination and remote logon data as in the CPI-C exercise
- Data: two LUWs, each with three records.

Record contents:

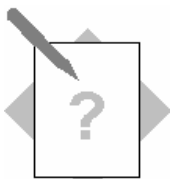
'GR##\_LUW1\_REC1', 'GR##\_LUW1\_REC2', 'GR##\_LUW1\_REC3',  
'GR##\_LUW2\_REC1', 'GR##\_LUW2\_REC2', 'GR##\_LUW2\_REC3'  
(## = your group number)

**Notes :**

- 1 You can ignore the parameters for the function modules you are calling that are not documented in the exercises.
  - 2 You can use OTHERS for the exceptions to the function modules you want to call. Refer to the relevant function module documentation for their meanings.
- 1-2 Use transaction **SM38 (Queue Administration)** to ensure that your queue has been created successfully. Display its attributes. You can display the corresponding job using transaction **SM37 (Job Administration)**.
- 1-3 Log onto the target system and look at the coding of receiving program **RCVQDATP**. This program receives the queue data in ping-pong mode and stores it in table **QDAT** (data processing). Use transaction **SE16 (Data Browser)** to ensure that your queue data has been successfully received and processed (stored in table QDAT).

*Tips & Tricks*

If you specify a **transmission time in the past** when you create your queue, the system will be able to create the queue successfully, but the corresponding job generated when you use `QUEUE_CLOSE` will be scheduled **without a start time**.

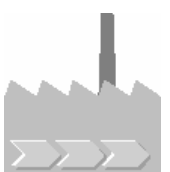


## Unit: Asynchronous Data Transfer (Q-API)

### Topic: Asynchronous Data Transfer between R/3 Systems



- Creating a queue for automatic transmission in the source system
- Queue administration
- Checking the processing results



You want to send data asynchronously from one system to another system because the target system is currently not available or overloaded.

```
REPORT SAPBC415S_CREATE_QUEUE MESSAGE-ID BC415.
```

```
INCLUDE RSQAPIDF.
```

PARAMETERS:

|          |                     |                     |                     |
|----------|---------------------|---------------------|---------------------|
| QID      | LIKE APQI-QID       | DEFAULT 'Q00',      |                     |
| QTYPE    | LIKE APQI-QATTRIB   | DEFAULT 'A',        | " unique/appendable |
| STRTMODE | LIKE APQI-STARTMODE | DEFAULT 'A',        | " manual/automatic  |
| STRTDAT  | LIKE APQI-STARTDATE | DEFAULT SY-DATUM,   |                     |
| STRTTIME | LIKE APQI-STARTTIME | DEFAULT SY-UZEIT,   |                     |
| DRIVER   | LIKE APQI-STARTPGID | DEFAULT 'RSQAPI21', | " data sending pgm  |
| DEST     | LIKE APQI-DESTSYS   | DEFAULT 'DEST00',   | " Sideinfo-entry    |
| CLIENT   | LIKE APQI-MANDANT   | DEFAULT '000',      | " logon-data        |
| USERID   | LIKE APQI-USERID    | DEFAULT 'SAPCPIC',  | " for               |
| PASSWD   | LIKE APQI-PASSWD    | DEFAULT 'ADMIN',    | " remote system     |
| PROG     | LIKE APQI-PROGID    | DEFAULT 'RCVQDATP', | " data receiving    |
| FORM     | LIKE APQI-FORMID    | DEFAULT 'RECEIVE',  | " program           |
| LUWS     | TYPE N              | DEFAULT '2'.        |                     |

DATA:

|                           |                          |
|---------------------------|--------------------------|
| RCDPOS,                   | " record position in LUW |
| BUFFER LIKE APQD-VARDATA, | " input data for queue   |
| LENGTH TYPE I,            | " input length           |

```

        LUWCOUNTER TYPE N.                " counter for LUWs

*-----*
*  Open queue                             *
*-----*

CALL FUNCTION 'QUEUE_OPEN'
  EXPORTING
    NAME           = QID
    OPENMODE       = 'W'
    ERASE          = 'X'                " automatically erased when empty
    TYPE           = QTYPE              " unique/appendable
    START          = STRTMODE           " manual/automatic start
    DATE           = STRTDATE
    TIME           = STRTTIME
    DATATYPE       = ' '                " no data conversion ASC_TO_EBC
    DRIVER         = DRIVER             " active data sending program
    DESTINATION    = DEST               " Sideinfo-entry
    CLIENT         = CLIENT             " logon-data
    USERID         = USERID            "   for
    PASSWORD       = PASSWD            "   remote system
    PROGRAM        = PROG               " passive data receiving program
    FORM           = FORM               "   and subroutine
  EXCEPTIONS
    INVALID_PARAMETER = 1              " invalid value in parameter
    OTHERS            = 2.

CASE SY-SUBRC.
  WHEN 1.
    MESSAGE E184 WITH 'Invalid value in QUEUE_OPEN-parameter !'.
  WHEN 2.
    MESSAGE E184 WITH 'Other error in QUEUE_OPEN !'.
ENDCASE.

*-----*
*  Fill queue                             *
*-----*

DO LUWS TIMES.

    RCDPOS = Q_FIRST.                  " first record of LUW

```

```

BUFFER = 'GR00_LUW _REC1'.
BUFFER+8(1) = LUWCOUNTER = SY-INDEX.
LENGTH = STRLEN( BUFFER ).
PERFORM INSERT_RECORD_TO_QUEUE.

RCDPOS = Q_MIDDLE.                " second record of LUW
BUFFER = 'GR00_LUW _REC2'.
BUFFER+8(1) = LUWCOUNTER = SY-INDEX.
LENGTH = STRLEN( BUFFER ).
PERFORM INSERT_RECORD_TO_QUEUE.

RCDPOS = Q_LAST.                  " third (last) record of LUW
BUFFER = 'GR00_LUW _REC3'.
BUFFER+8(1) = LUWCOUNTER = SY-INDEX.
LENGTH = STRLEN( BUFFER ).
PERFORM INSERT_RECORD_TO_QUEUE.

ENDDO.

*-----*
*  Close queue                                *
*-----*

CALL FUNCTION 'QUEUE_CLOSE'
  EXPORTING
    NAME           = QID
    OPENMODE       = 'W'          " openmode used in QUEUE_OPEN
  EXCEPTIONS
    INVALID_PARAMETER = 1
    OTHERS           = 2.

CASE SY-SUBRC.
  WHEN 0.          " queue sucessfully closed
    COMMIT WORK.
    SKIP 2.
    WRITE: 6 'Created queue sucessfully !'.
    SKIP.
    WRITE: 6 'Queue-Id : ', QID.
  WHEN 1.
    MESSAGE E184 WITH 'Invalid value in QUEUE_CLOSE-parameters !'.
  WHEN 2.

```

```

        MESSAGE E184 WITH 'Other error in QUEUE_CLOSE !'.
    ENDCASE.

```

```

*-----*
*  FORM  insert_record_to_queue                                *
*-----*

```

```

FORM INSERT_RECORD_TO_QUEUE.

```

```

    CALL FUNCTION 'QUEUE_PUT'
      EXPORTING
        NAME      = QID
        STATE     = RCDPOS           " record position in LUW
        BUFFER    = BUFFER          " record data
        LENGTH    = LENGTH          " record length
      EXCEPTIONS
        INVALID_PARAMETER = 1       " invalid value in parameter
        OTHERS           = 2.

```

```

CASE SY-SUBRC.

```

```

    WHEN 1.
        " invalid value in QUEUE_PUT-parameters
        " -> message + LUW-rollback

        MESSAGE E184
          WITH 'Invalid value in QUEUE_PUT-parameters -> LUW-rollback'.

```

```

    WHEN 2.
        " other error in QUEUE_PUT !
        " -> message + LUW-rollback

        MESSAGE E184
          WITH 'Other error in QUEUE_PUT -> LUW-rollback'.

```

```

ENDCASE.

```

```

IF RCDPOS = Q_LAST.
    " complete LUW in queue
    COMMIT WORK.
ENDIF.

```

```

ENDFORM.

```





# BAPI Workshop

## Description:

The following pages provide guidelines for creating BAPIs. We will start with the business object *FlightCustomer* (object type *SCUSTOM*). This business object is available in all SAP training systems, but not in the standard delivery. If you want to reconstruct the example described in the guide in your own system, you must first import the 4.6 training add-on. This add-on contains all the objects used in ABAP training courses and can be downloaded from the SAP service homepage:

**<http://service.sap.com/ocs-download>**

→ Logon with OSS user or User "walldorf" (Password "training")

### In the navigation area:

- Download support packages
- Add-on support packages / CRTs
- Further add-on components
- Training
- Training 46C

### In the contents area:

- All objects (full task)
- Download

### Note:

Data for flight tables can be generated automatically by calling the report SAPBC\_DATA\_GENERATOR.

**Scenario:**

In the scenario to be implemented here, you require a BAPI to read data about a flight customer. First, you need to display a list of flight customers. From this list you can select a flight customer, and then, using another BAPI, display specific details about this customer.

The relevant **SAP Business Object** for this scenario is *FlightCustomer*.

To read a flight customer's details as described in the above scenario, two BAPIs are required: One to display a list of flight customers (*GetList*) and another to display details of a specific flight customer (*GetDetail*).

The interdependency of these two BAPIs is evident because, first of all, the list of flight customers is displayed to obtain the ID of the specific flight customer sought. From this ID, you can display details of the flight customer.

However, the two BAPIs remain functionally independent of each other because, if the flight customer ID is known, the BAPI to display the details of a specific flight customer can be used without calling the *GetList* BAPI.

In this workshop, you will create the two above-mentioned BAPIs.

You will create the *GetList* BAPI first. It requires that you create a structure, a function module, and a form routine. To save time, the ABAP code is attached to the end of this workshop. After creating the ABAP programs, you will then use the BOR/BAPI Wizard to create the BAPI and attach it to the *FlightCustomer* object.

The second BAPI you will create is the *GetDetail* BAPI. You will have to create a function module to get the detail about a flight customer and then create the BAPI using the BOR/BAPI wizard. To save time, the ABAP code is attached to the end of this workshop.

## Section 1: Programming the *GetList* BAPI

### Background

The *GetList* BAPI requires a structure to hold the return data for the BAPI method call. The BAPI will return the *Customer ID* and the *Customer Name* fields in a structure. The structure we will create is based upon the SCUSTOM table.

- 1 Go to the *Data Dictionary* and define a structure for the BAPI.  
Use the Menu Path *Tools* → *ABAP WORKBENCH* and select the *Dictionary* command button in the toolbar or use transaction SE11 to get into the *Data Dictionary*.
- 2 Select the *structures* option button and enter the name as ZBAPICUSTxx (xx is your group number). Press the *Enter* key to display the detailed structure screen.
- 3 In the *Short Text* field, enter the structure for the *GetList* BAPI.
- 4 Complete the remaining entries for the two fields in the structure as follows:  
  
Field Name: ID  
  
Type Name: S\_CUSTOMER  
  
Press the *Enter* key, and the remaining entries for ID will be filled in from the S\_CUSTOMER definition.  
  
Field Name: NAME  
  
Type Name: S\_CUSTNAME  
  
Press the *Enter* key, and the remaining entries for NAME will be filled in from the S\_CUSTNAME definition.
- 5 Save and activate the structure.

## Section 2: Building the Function Module for the *GetList* BAPI

- 1 Go to the *Function Builder* and create a function group. Use the menu path *Tools* → *ABAP Workbench* → *Function Builder*.
- 2 Name the function group ZGxx (xx is your group number).

- 3 Create a function module.  
Name the module ZBAPI\_GET\_CUSTOMER\_LISTxx (xx is your group number).

- 4 Define the interface for your function module.

On the *Admin* tab :

- Enter the short text (your choice).
- Select *Remote Function Call* supported in the processing type

On the *Export* tab:

- Enter *Return* for the export parameters and BAPIRETURN for the *Ref.field / structure*.

On the *Tables* tab:

- Enter SCUST\_LIST for the table parameters and ZBAPICUSTxx for the *Ref.Structure* (xx is your group number).

On the *Documentation* tab: Entries are made for you.

On the *Import, Changing* and *Exceptions* tabs: No entries required.

- 5 You will need to add global data for the function group. Specifically, you will need to add declarations for the table work area *scustom*, the internal table, and a message structure. See the attached program code for the function group global declarations at the end of this workshop. Since these are global for your function group, you will not need to repeat these declarations for the *GetDetail* BAPI. To access the *Global Data* area from the source code editor, use the menu path *GoTo → Global Data*.
- 6 Code the function module. The code for your function module is attached at the end of this workshop.  
PLEASE NOTE: YOU NEED TO CHANGE ANY REFERENCES TO TABLES AND PROGRAMS BASED UPON THE NAMES YOU HAVE BEEN ASSIGNED. THE SAMPLE CODE USES A STRUCTURE AND FUNCTION NAME DIFFERENT FROM YOURS.

- 7 You will need to create a form routine that handles messages. You can find the ABAP code at the end of this workshop. It should not be changed from the printed copy. You can just enter it as is.

To access the form routine (subroutine) area, double click the name of the form routine being called. For our workshop, the form routine is `set_return_message`. This routine handles the messaging return structure (of type `BAPIRETURN`), which is used to inform the external program calling the BAPI whether the BAPI was successful or not.



## Section 3 : Defining Methods in the BOR using the BOR / BAPI wizard

- 1 Go to the *Business Object Builder*. You can use the path from the main menu *Tools → Business Framework → BAPI development → Business Object Builder*. Or you can use transaction SWO1 (alpha O).
- 2 In the *Business Object Builder* initial screen, enter SCUSTOMER as the object type. Then select the *Change* command button.  
Note: If you are prompted for a change request, enter one through the dialogs using a local change request path.
- 3 Once you have displayed the *Change Object Type* screen, go to the menu path *Utilities → API methods → Add method*. This will display a dialog to enter the function module name you created in the previous steps (for example, ZBAPI\_GET\_CUSTOMER\_LIST).  
Select the green checkmark, and the *Create API method: Method features* dialog will be displayed.
- 4 Enter the method name. Use the name *GetCustomerListxx* (xx is your group number). In the text boxes, type an appropriate name and description for documentation purposes. In the *Characteristics* section, make sure *Synchronous* and *Instance-independent* are checked.
- 5 Select the next button, which is the second button of three in the lower left corner of the dialog box. This will take you to the next step in the BOR/BAPI wizard. The next step is the *Create parameter* dialog. In the *Create parameter* step, a list of parameters and default names is displayed. You may need to edit this, as required. Modify the parameter names as follows:
  - Each new word in the parameter name must start with a capital letter, for example, ScustList or CustDetail.
  - Make sure that the parameter names of the method in the BOR are identical to the parameter names in the function module, except for the upper/lower case letters.
  - Specify whether the individual table parameters are used for data import or data export. Table parameters are marked with a check in column *Mline* (multiple lines).



NOTE: You do not need to make any changes in this step for our workshop. Select the next button to bring you to the next step in the BOR/BAPI wizard.

- 6 The next step is the *Extend Program* step. The dialog warns you that the BAPI is not yet implemented. Select *Yes* to generate the template that will add the BAPI as a method of the SCUSTOMER object type. This step adds “wrapper object-oriented” code to your function module, converting it to a method of an SAP Business Object.
- 7 The next step is to generate the BAPI entry. You should now be back on the *Change Object Type* screen. Select the plus sign next to the *Methods* to expand and display all the methods of SCUSTOMER. Your BAPI should now be listed with a green circle icon (green light). Select your BAPI once and then select the *Generate* icon in the toolbar (fourth icon from the left).
- 8 Test your BAPI by selecting your BAPI once and then select the test tool in the toolbar (the sixth button from the left) or you can press the F8 key.

## Section 4: Programming the *GetDetail* BAPI

### Background

The *GetDetail* BAPI requires several fields to hold the return data for the BAPI method call. The BAPI will return the following fields :

Customer ID  
Customer Name  
Postal Code  
City  
Customer Type  
Discount

This BAPI will be instance-dependent and therefore require that a keyfield be passed to create the object.

What is the key field for the business object *FlightCustomer* ?



## Section 5: Building the Function Module for the *GetDetail* BAPI

- 1 Create a function module in your Function Group ZGxx.  
Name the module ZBAPI\_GET\_CUSTOMER\_DETAILxx (xx is your group number). The global data for this function module was already created in the function group in the previous steps.

- 2 Define the interface for your function module.

On the *Admin* tab :

- Enter the short text (your choice).
- Select *Remote Function Call* supported in the processing type.

On the *Import* tab:

- Enter *ID* as the import parameter and ZBAPICUSTxx-ID as the *Ref.field / structure*.

On the *Export* tab, enter the following:

| Export Parameters | Ref.field / structure |
|-------------------|-----------------------|
| SCUST_ID          | SCUSTOM-ID            |
| SCUST_NAME        | SCUSTOM-NAME          |
| SCUST_POSTCODE    | SCUSTOM-POSTCODE      |
| SCUST_CITY        | SCUSTOM-CITY          |
| SCUST_CUSTTYPE    | SCUSTOM-CUSTTYPE      |
| SCUST_DISCOUNT    | SCUSTOM-DISCOUNT      |
| RETURN            | BAPIRETURN            |

On the *Changing, Tables and Exceptions* tab: no entries.

On the *Documentation* tab: Entries are made for you.

- 3 The code for your function module is attached to the end of this workshop.  
PLEASE NOTE: YOU NEED TO CHANGE ANY REFERENCES TO TABLES AND PROGRAMS BASED UPON THE NAMES YOU HAVE BEEN ASSIGNED. THE SAMPLE CODE USES A STRUCTURE AND FUNCTION NAME DIFFERENT FROM YOURS.

- 4 The form routine your function module needs to handle messages was already created in the function group ZGxx in one of the last sections, so there is no need to create it again.

## SECTION 6: Defining Methods in the BOR using the BOR / BAPI wizard

- 1 Go to the *Business Object Builder*. You can use the path from the main menu *Tools → Business Framework → BAPI development → Business Object Builder*. Or you can use transaction SWO1 (alpha O).
- 2 In the *Business Object Builder* initial screen, enter SCUSTOMER as the object type. Then select the *Change command* button.  
Note: If you are prompted for a change request, enter one through the dialogs using a local change request path.
- 3 Once you have displayed the *Change Object Type* screen, go to the menu path *Utilities → API methods → Add method*. This will display a dialog to enter the function module name you created in the previous steps (for example, ZBAPI\_GET\_CUSTOMER\_DETAILxx).  
Select the green checkmark, and the *Create API method : Method features* dialog will be displayed.
- 4 Enter the *Method* name. Use the name *GetCustomerDetailxx* (xx is your group number). In the text boxes, type an appropriate name and description for documentation purposes. In the *Characteristics* section, make sure *Synchronous* is checked. *Instance-independent* will also be checked, but you will change this in a later step.
- 5 Select the next button, which is the second button of three in the lower left corner of the dialog box. This will take you to the next step in the BOR/BAPI wizard. The next step is the *Create parameter* dialog. In the *Create parameter* step, a list of parameters and default names is displayed. You may need to edit this, as required. Modify the parameter names as follows:
  - Each new word in the parameter name must start with a capital letter, for example, ScustList or CustDetail.
  - Make sure that the parameter names of the method in the BOR are identical to the parameter names in the function module, except for the upper/lower case letters.

- Specify whether the individual table parameters are used for data import or data export. Table parameters are marked with a check in the column *Mline* (multiple lines).

NOTE: You do not need to make any changes in this step for our workshop. Select the next button to bring you to the next step in the BOR/BAPI wizard.

- 6 The next step is the *Extend Program* step. The dialog warns you that the BAPI is not yet implemented. Select *Yes* to generate the template, which will add the BAPI as a method of the SCUSTOMER object type.
- 7 The next step is to generate the BAPI entry. You should now be back on the *Change Object Type* screen. Select the plus sign next to the *Methods* to expand and display all the methods of SCUSTOMER. Your BAPI should now be listed with a green circle icon (green light). Select your BAPI once and then select the *Generate* icon in the toolbar (fourth icon from the left).
- 8 After generating the BAPI, double click the *GetCustomerDetailxx* BAPI. This will display the *Method* attributes dialog with 3 tabs. On the *General* tab, deselect the *Instance-independent* checkbox. Why do you deselect this checkbox? If you are not sure, have a look at the BAPI Programming Guide. It is very important to BAPI development that you know the difference between instance-dependent and instance-independent BAPIs.
- 9 Test your BAPI by selecting your BAPI once and then select the test tool in the toolbar (the sixth button from the left), or press the F8 key.

# ABAP Codes

## Global Function Group Data Declaration

```
FUNCTION-POOL ZG00.                                "MESSAGE-ID ..

TABLES SCUSTOM.

DATA: ITAB LIKE ZBAPICUST00 OCCURS 0 WITH HEADER LINE.

DATA: BEGIN OF MESSAGE,
      MSGTY LIKE SY-MSGTY,
      MSGID LIKE SY-MSGID,
      MSGNO LIKE SY-MSGNO,
      MSGV1 LIKE SY-MSGV1,
      MSGV2 LIKE SY-MSGV2,
      MSGV3 LIKE SY-MSGV3,
      MSGV4 LIKE SY-MSGV4,
END OF MESSAGE.
```

Internal Use SAP Partner Only

Internal Use SAP Partner Only



## BAPI Function Module ZBAPI\_GET\_CUSTOMER\_LIST00

```
FUNCTION ZBAPI_GET_CUSTOMER_LIST00.

*"-----
*""Local interface:
*"  EXPORTING
*"    VALUE(RETURN) LIKE BAPIRETURN STRUCTURE BAPIRETURN
*"  TABLES
*"    SCUST_LIST      STRUCTURE ZBAPICUST00
*"-----

  CLEAR SCUST_LIST.
  REFRESH SCUST_LIST.

  CLEAR ITAB.
  REFRESH ITAB.

  CLEAR RETURN.
  CLEAR SCUSTOM.

  SELECT * FROM SCUSTOM
    INTO CORRESPONDING FIELDS OF TABLE ITAB.

  IF SY-SUBRC NE 0.
    CLEAR MESSAGE.
    MESSAGE-MSGTY = 'E'.
    MESSAGE-MSGID = 'FN'.
    MESSAGE-MSGNO = 022.
    PERFORM SET_RETURN_MESSAGE
      USING      MESSAGE
      CHANGING RETURN.
  ENDIF.
```

CHECK RETURN IS INITIAL.

SCUST\_LIST[] = ITAB[].

ENDFUNCTION.



## BAPI Function Module

### ZBAPI\_GET\_CUSTOMER\_DETAIL00

```

FUNCTION ZBAPI_GET_CUSTOMER_DETAIL00.

*"-----
*""Local interface:
*"  IMPORTING
*"    VALUE(ID)                LIKE ZBAPICUST-ID
*"  EXPORTING
*"    VALUE(SCUST_ID)          LIKE SCUSTOM-ID
*"    VALUE(SCUST_NAME)        LIKE SCUSTOM-NAME
*"    VALUE(SCUST_POSTCODE)    LIKE SCUSTOM-POSTCODE
*"    VALUE(SCUST_CITY)        LIKE SCUSTOM-CITY
*"    VALUE(SCUST_CUSTTYPE)    LIKE SCUSTOM-CUSTTYPE
*"    VALUE(SCUST_DISCOUNT)   LIKE SCUSTOM-DISCOUNT
*"    VALUE(RETURN) LIKE BAPIRETURN STRUCTURE BAPIRETURN
*"-----

      CLEAR: SCUST_ID, SCUST_NAME, SCUST_POSTCODE,
              SCUST_CITY, SCUST_CUSTTYPE, SCUST_DISCOUNT.

      CLEAR: SCUSTOM, RETURN.

      SELECT SINGLE * FROM SCUSTOM WHERE ID = ID.

      IF SY-SUBRC NE 0
        CLEAR MESSAGE.
        MESSAGE-MSGTY = 'E'.
        MESSAGE-MSGID = 'FN'.
        MESSAGE-MSGNO = 020.
        MESSAGE-MSGV1 = ID.
        PERFORM SET_RETURN_MESSAGE

```

INTERNAL USE SAP PARTNER ONLY

```
        USING      MESSAGE
        CHANGING RETURN.

ENDIF.
```

```
CHECK RETURN IS INITIAL.
```

```
SCUST_ID      = SCUSTOM-ID.
SCUST_NAME    = SCUSTOM-NAME.
SCUST_POSTCODE = SCUSTOM-POSTCODE.
SCUST_CITY    = SCUSTOM-CITY.
SCUST_CUSTTYPE = SCUSTOM-CUSTTYPE.
SCUST_DISCOUNT = SCUSTOM-DISCOUNT.
```

```
ENDFUNCTION.
```

INTERNAL USE SAP PARTNER ONLY

## Function Group Form Routine SET\_RETURN\_MESSAGE

```

*-----*
***INCLUDE LZG00F01 .
*-----*
*&-----*
*&      Form  SET_RETURN_MESSAGE
*&-----*
*      text
*-----*
*      -->P_MESSAGE  text
*      <--P_RETURN   text
*-----*

FORM SET_RETURN_MESSAGE
      USING      P_MESSAGE LIKE MESSAGE
      CHANGING P_RETURN  LIKE BAPIRETURN.

CHECK NOT MESSAGE IS INITIAL.

CALL FUNCTION 'BALW_BAPIRETURN_GET'
      EXPORTING
          TYPE      = P_MESSAGE-MSGTY
          CL        = P_MESSAGE-MSGID
          NUMBER    = P_MESSAGE-MSGNO
          PAR1      = P_MESSAGE-MSGV1
          PAR2      = P_MESSAGE-MSGV2
          PAR3      = P_MESSAGE-MSGV3
          PAR4      = P_MESSAGE-MSGV4
      IMPORTING
          BAPIRETURN = P_RETURN

```

EXCEPTIONS

OTHERS = 1.

ENDFORM.

" SET\_RETURN\_MESSAGE

Internal Use SAP Partner Only

Internal Use SAP Partner Only