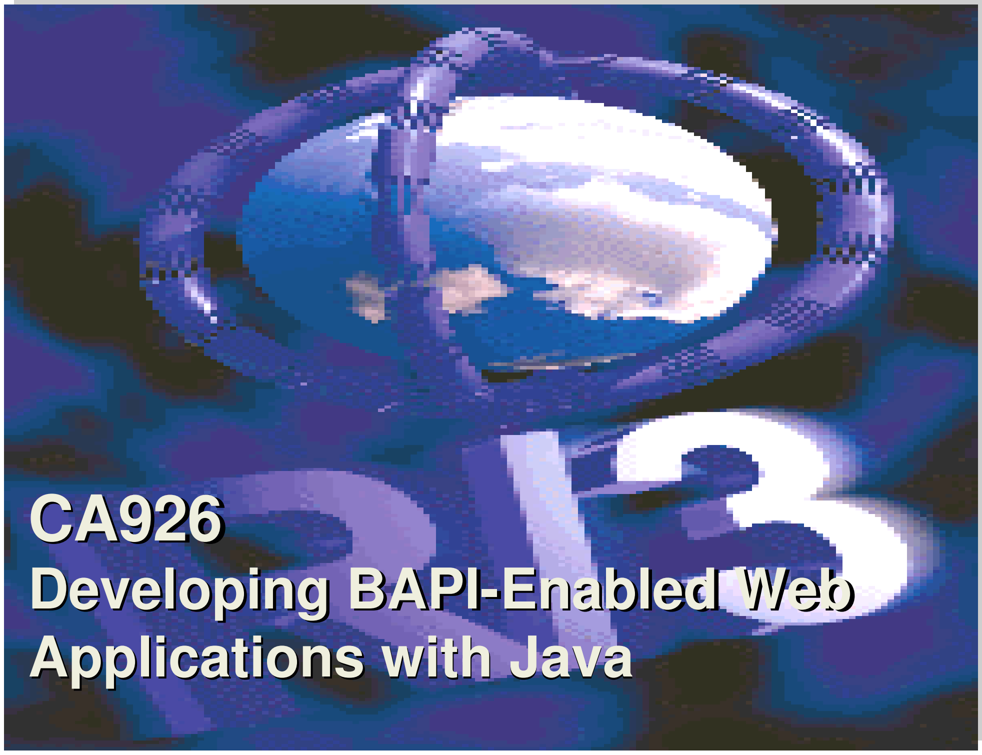
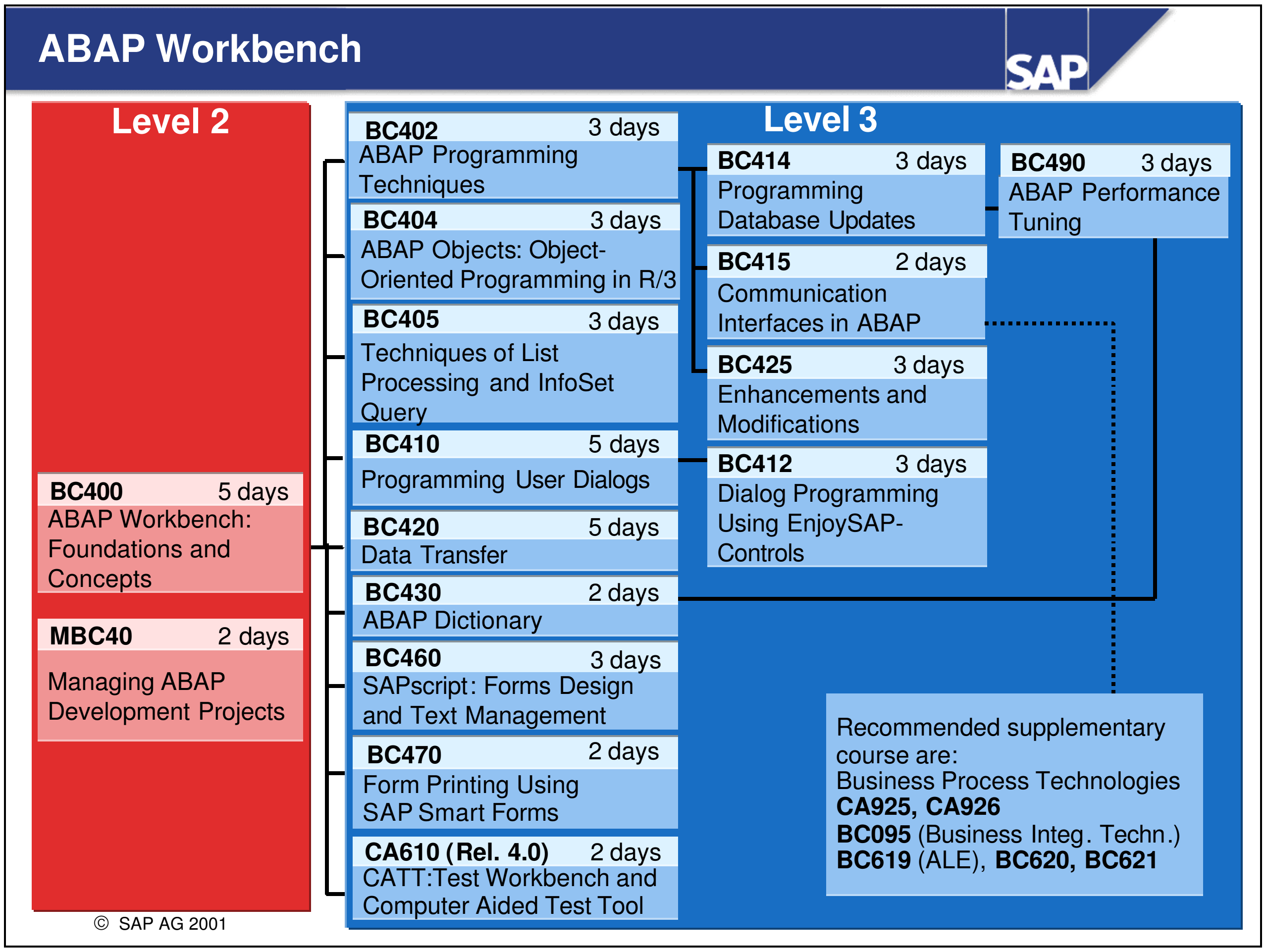


CA926
Developing BAPI-Enabled Web Applications with Java

- R/3 System
- Release 46C
- June 2001
- Material Number: 50044886

Copyright

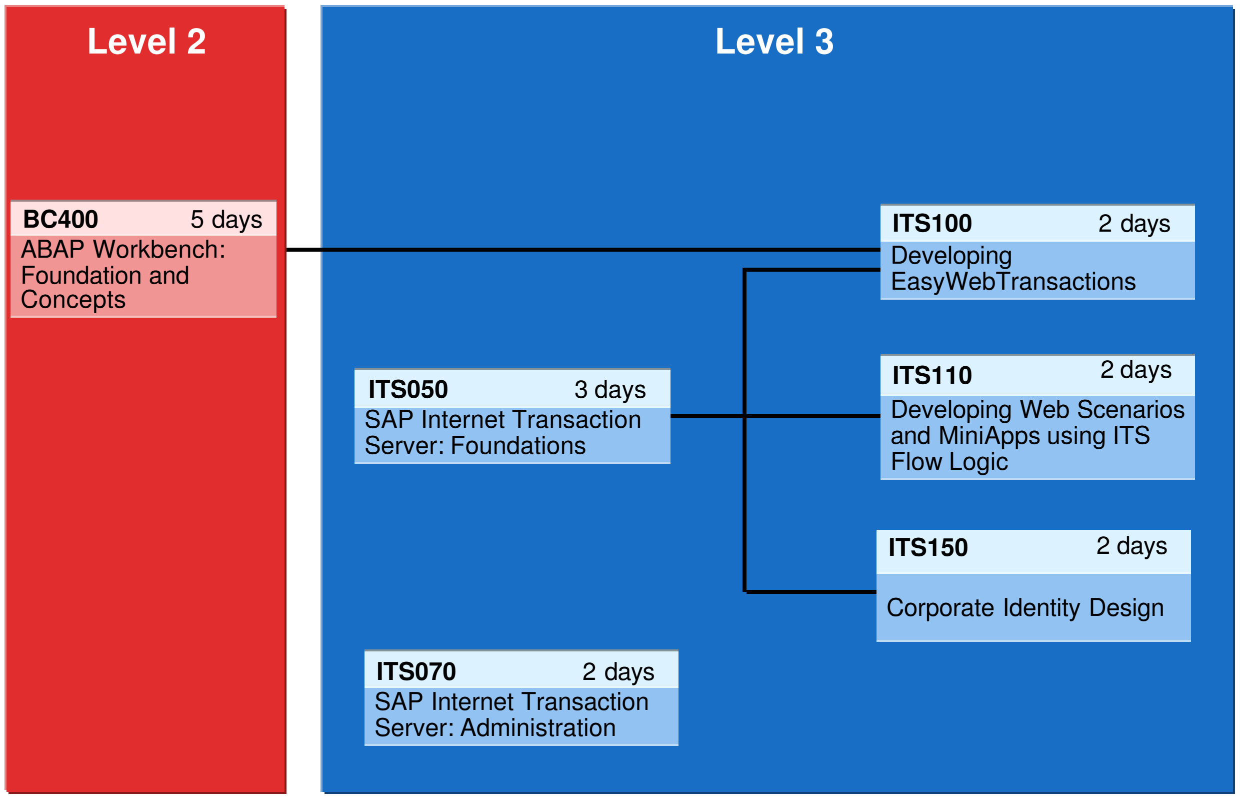
ABAP Workbench



Level 2
BC400 5 days
ABAP Workbench: Foundations and Concepts
MBC40 2 days
Managing ABAP Development Projects
Level 3
BC402 3 days
ABAP Programming Techniques
BC404 3 days
ABAP Objects: Object-Oriented Programming in R/3
BC405 3 days
Techniques of List Processing and InfoSet Query
BC410 5 days
Programming User Dialogs
BC420 5 days
Data Transfer
BC430 2 days
ABAP Dictionary
BC460 3 days
SAPscript: Forms Design and Text Management
BC470 2 days
Form Printing Using SAP Smart Forms
CA610 (Rel. 4.0) 2 days
CATT:Test Workbench and Computer Aided Test Tool
BC414 3 days
Programming Database Updates
BC415 2 days
Communication Interfaces in ABAP
BC425 3 days
Enhancements and Modifications
BC412 3 days
Dialog Programming Using EnjoySAP-Controls
BC490 3 days
ABAP Performance Tuning

Recommended supplementary course are:
Business Process Technologies
CA925, CA926
BC095 (Business Integ. Techn.)
BC619 (ALE), **BC620, BC621**

ITS



Level 2

BC400 5 days
ABAP Workbench: Foundation and Concepts

Level 3

ITS050 3 days
SAP Internet Transaction Server: Foundations

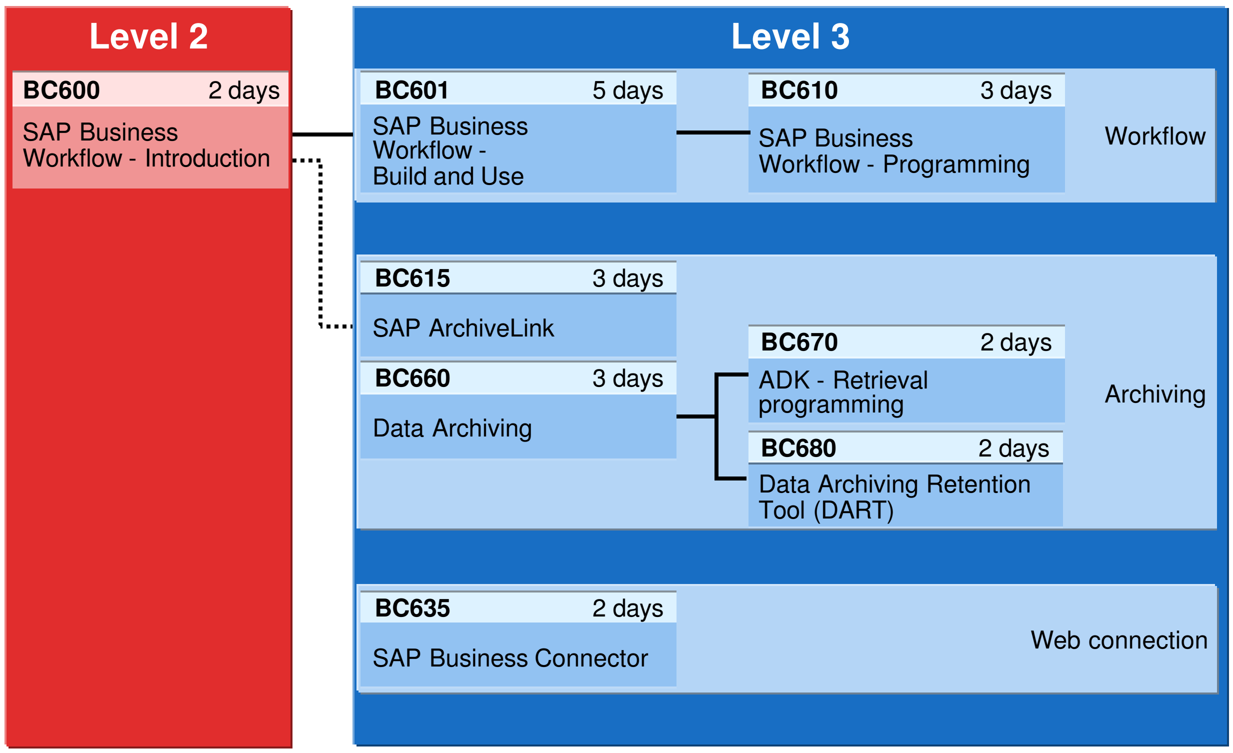
ITS070 2 days
SAP Internet Transaction Server: Administration

ITS100 2 days
Developing EasyWebTransactions

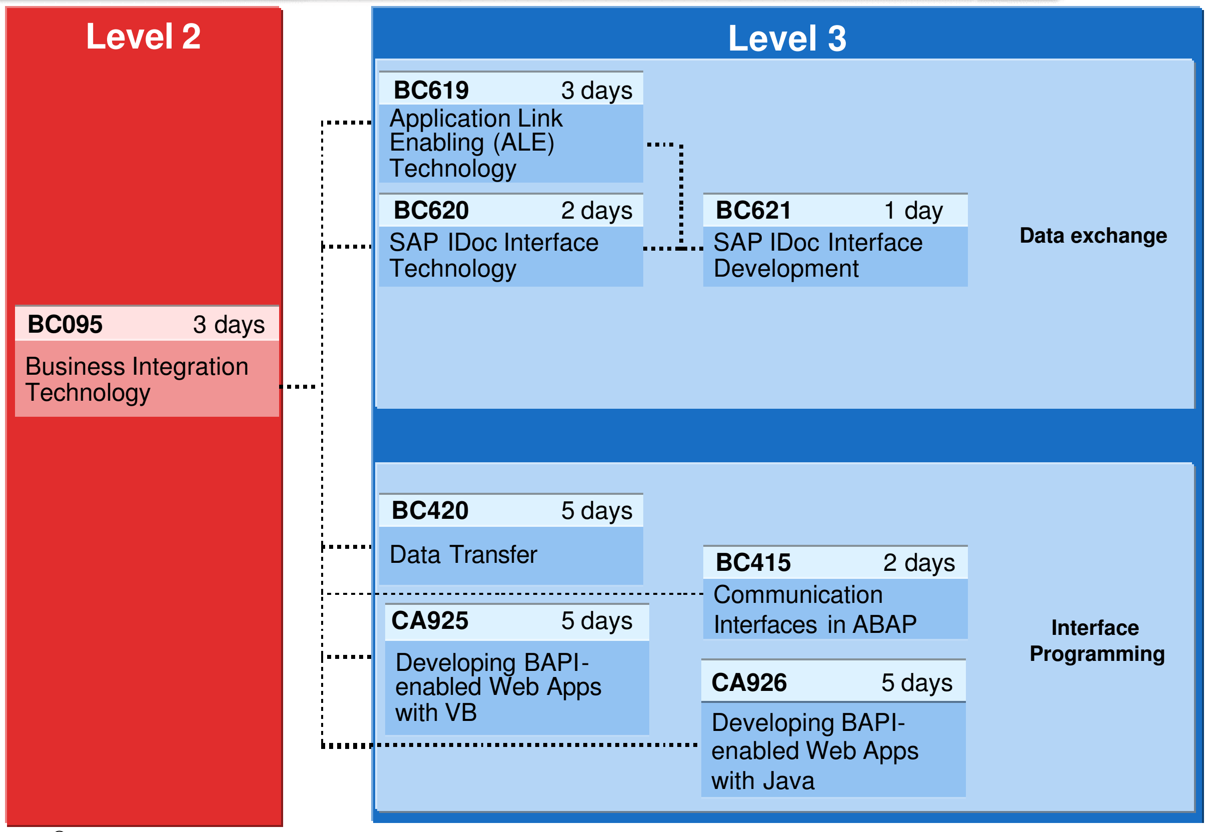
ITS110 2 days
Developing Web Scenarios and MiniApps using ITS Flow Logic

ITS150 2 days
Corporate Identity Design

Business Integration Technologies I



Business Integration Technologies II



Level 2

BC095 3 days
Business Integration Technology

Level 3

Data exchange

BC619 3 days
Application Link Enabling (ALE) Technology

BC620 2 days
SAP IDoc Interface Technology

BC621 1 day
SAP IDoc Interface Development

Interface Programming

BC420 5 days
Data Transfer

BC415 2 days
Communication Interfaces in ABAP

CA925 5 days
Developing BAPI-enabled Web Apps with VB

CA926 5 days
Developing BAPI-enabled Web Apps with Java

Course Prerequisites

- "At least one year of real-world Java programming experience is required. Attending this class without this makes no sense whatsoever. This is a hands-on class with extensive exercises, not a management overview!"

Target Group

- **Audience:**
  - **Experienced Java programmers who want to use BAPIs, and/or IDocs to build Internet, Intranet, or desktop applications communicating with R/3 and other SAP components.**
- **Duration: 5 days**

Course Overview

Contents:

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

Course Goals

This course will prepare you to:

- Build applications and components in Java that communicate with R/3 and other SAP components using Business Application Programming Interfaces (BAPIs), other RFC-enabled Function Modules (RFMs), and ALE Intermediate Documents (IDocs). Both inbound and outbound calls are covered.

Course Objectives

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

- Call BAPIs.
- Call other RFMs.
- Send and receive IDocs.
- Build SAP-enabled Internet/Intranet/Extranet applications.

Course Content

Integration Scenarios: Contents

- Communication types
- Roles of the external application
- Types of applications

Integration Scenarios: Unit Objectives

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

- Understand the various integration scenarios between SAP and non-SAP components and applications.

Communication Types

- Synchronous
- Asynchronous

Roles of the External Application

- Client
- Server

Types of Applications

- Alternative Front-Ends
  - Internet
  - Intranet
  - Extranet
  - SAP Workplace miniApps
  - Desktop
  - Plug-ins
  - non-PC devices
- Data synchronization
- Application coupling
  - Inside your company
  - Business-to-business
- Message-based middleware
- Enabling components

Alternative Front-Ends: Examples

- Custom-written, high-speed sales order entry transaction
- Sales Force Automation (SFA) on a laptop
- Employee Self-Service (ESS) applications
- Mail merge with word processing tools
- Sales order entry on the Internet
- Information for suppliers/distributors (Extranet)
- Reporting on WAP-enabled phones

- AFEs can be completely customized for the company.

Integration Scenarios: Unit Summary

- Many different scenarios exist for the integration between SAP and non-SAP components and applications.

SAP Overview: Contents

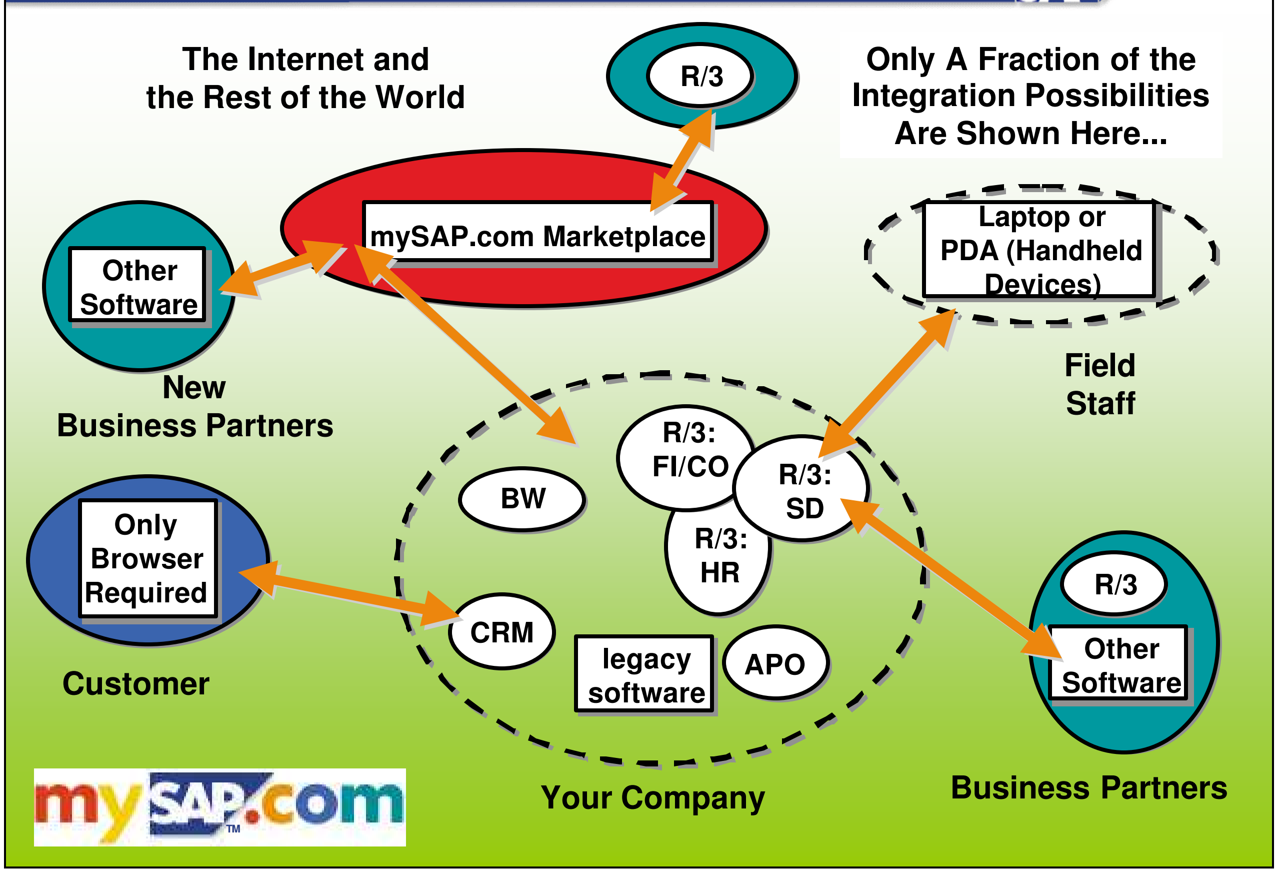
- mySAP.com Overview
- SAP Architecture
- SAP Open Interfaces
- mySAP.com Components

SAP Overview: Unit Objectives

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

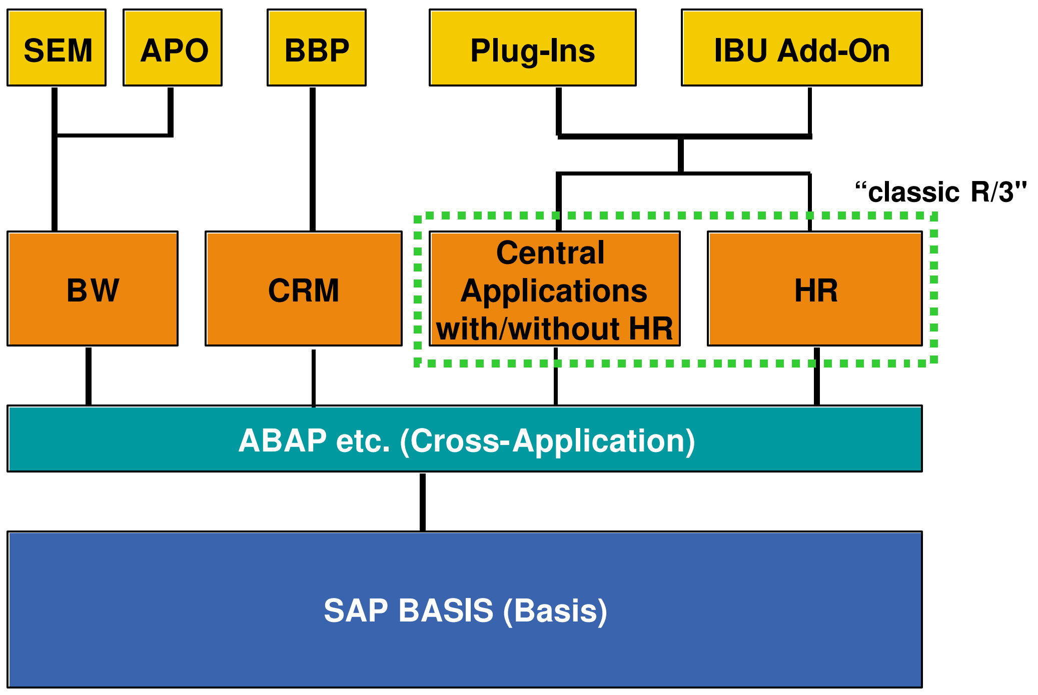
- Understand the SAP architecture.
- List the most important interface technologies.

What is mySAP.com?



- SAP's mySAP.com strategy offers SAP customers a tailored, highly-integrated, and open software landscape.
  - Tailored: SAP offers many different mySAP.com components that cover the range of modern business functions
  - Highly-integrated: Data from a mySAP.com environment is consistent beyond system borders and and can be called using various standard interfaces.
  - Open: The interfaces used in the mySAP.com environment are either industry or technical standards (such as RosettaNet or XML) or SAP standard interfaces like, for example, Business Application Programming Interfaces (BAPIs). Using these interfaces, you can transfer data in many different ways. See, for example, *http://www.rosettanet.org*.
- Only a small selection of the integrations possibilities are shown in the graphic above, but the basic principal should be clear: Using the available interfaces, it is possible not only to link the applications within your company, but also to communicate directly with business partners, marketplaces, field sales representatives, and customers.
- Another basic element is the mySAP.com Workplace, which offers every employee an individual and easy-to-use view of all of the functions that are required for that employee's job role. The current browser front end can offer access to several different systems, without the end user ever having to be aware that more than one system is being used. This is made possible by a *Single Sign On* mechanism.

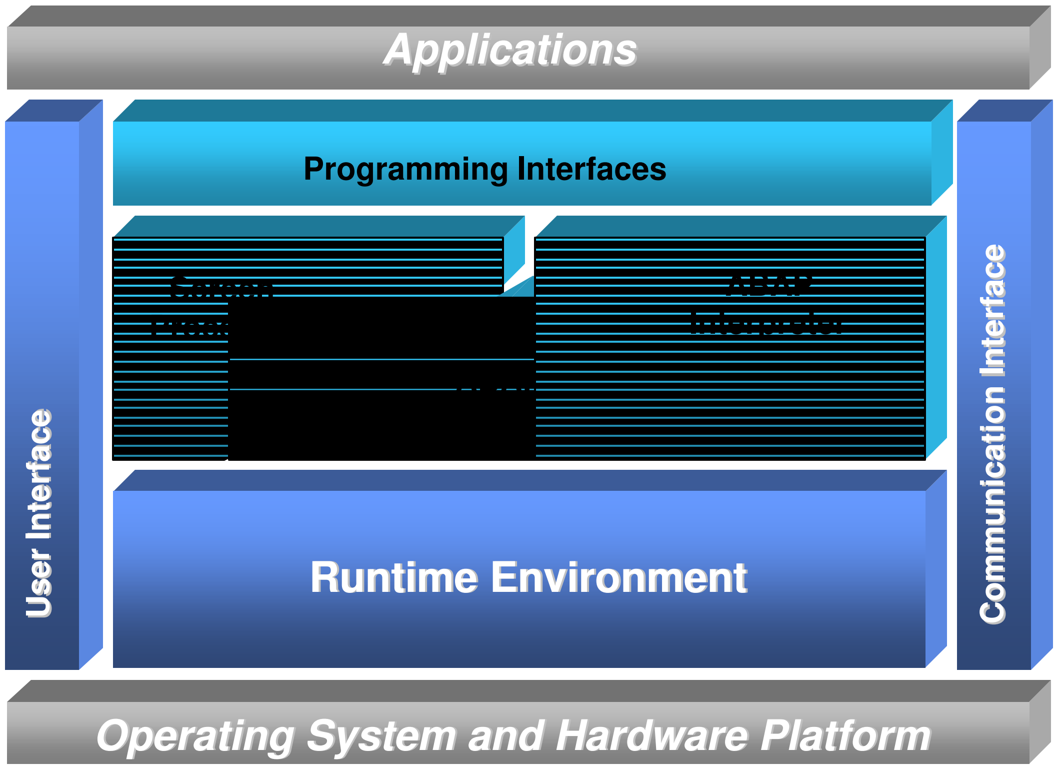
Common mySAP.com Technology


SEM
APO
BBP
Plug-Ins
IBU Add-On
"classic R/3"
BW
CRM
Central Applications with/without HR
HR
ABAP etc. (Cross-Application)
SAP BASIS (Basis)

A common Basis for all mySAP.com Products

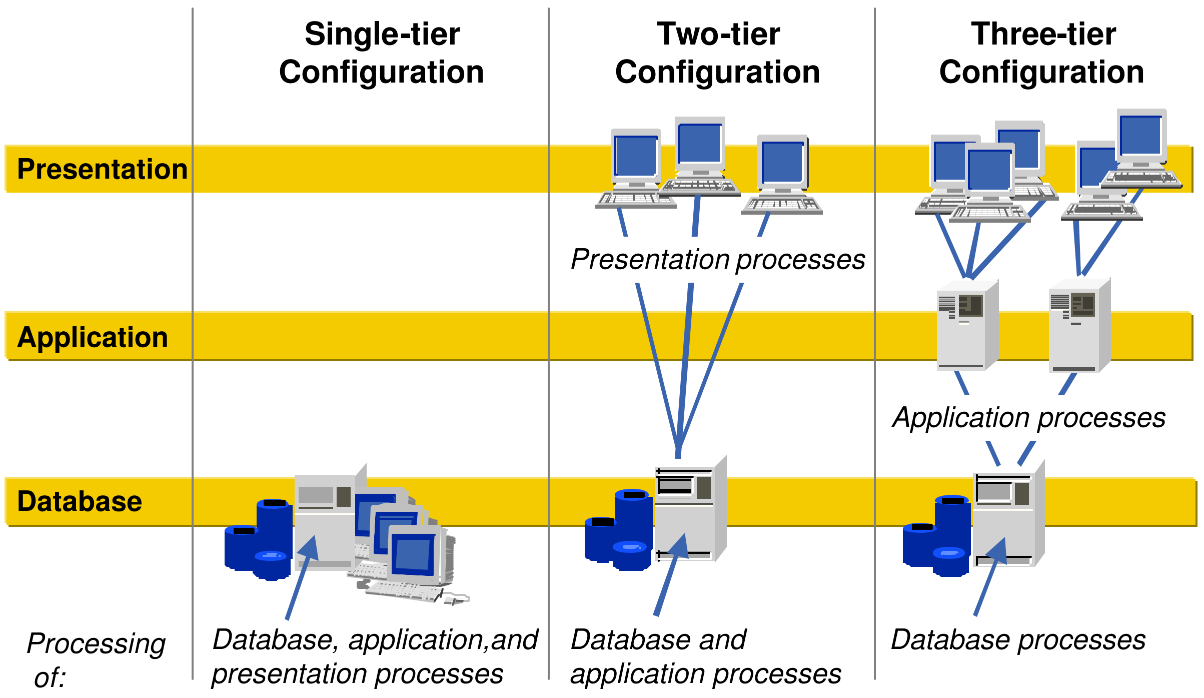
- mySAP.com Technology (SAP Basis) forms the technical foundation for almost all of the current mySAP.com components. The abbreviations used above represent the following components:
  - SEM: Strategic Enterprise Management
  - APO: Advanced Planner and Optimizer
  - BBP: Business to Business Procurement
  - IBU: Industry Business Unit
  - BW: Business Information Warehouse
  - CRM: Customer Relationship Management
  - HR: Human Resources
- More detail is provided about many of the products mentioned above in the mySAP.com Components unit, with the exception of SEM, which is not considered here because it is such a specialized component. SAP Knowledge Warehouse (KW) is also not included in the illustration above, due to its very unique structure. It is, however, also described in the mySAP.com Components unit.
- You can find more information about these products from the individual training courses for the components (see the *Knowledge & Training* section in the SAP Service Marketplace) and using their names as aliases in the SAP Service Marketplace, such as http://service.sap.com/BW.

Overview of the SAP Basis System



- To guarantee portability of mySAP.com applications, the system software interfaces are combined on an isolated level. Above this level, the functions of all SAP products are completely independent of the hardware and software environment.
- The runtime environment controls services such as scheduling or memory administration that could partly be left to the surrounding operating system software, but which are executed within the Basis System for reasons of portability and performance.
- The user interface provides the application presentation options.
- The communication interface defines the channels for exchanging information electronically, for transferring external data, for example, or for program-to-program communication according to the Remote Function Call (RFC) protocol and for the standard exchange of application data using Application Link Enabling (ALE).
- All application programs in SAP business applications are created in Advanced Business Application Programming (ABAP), SAP's own, interprative language. The controlling components for the screen sequence are DYNPROS (= dynamic programs). The interaction between the screen processor and the ABAP interpreter forms the technological basis of the mySAP.com applications. Both interpreters use the overall view of the SAPda ta that is stored in the ABAP Dictionary.

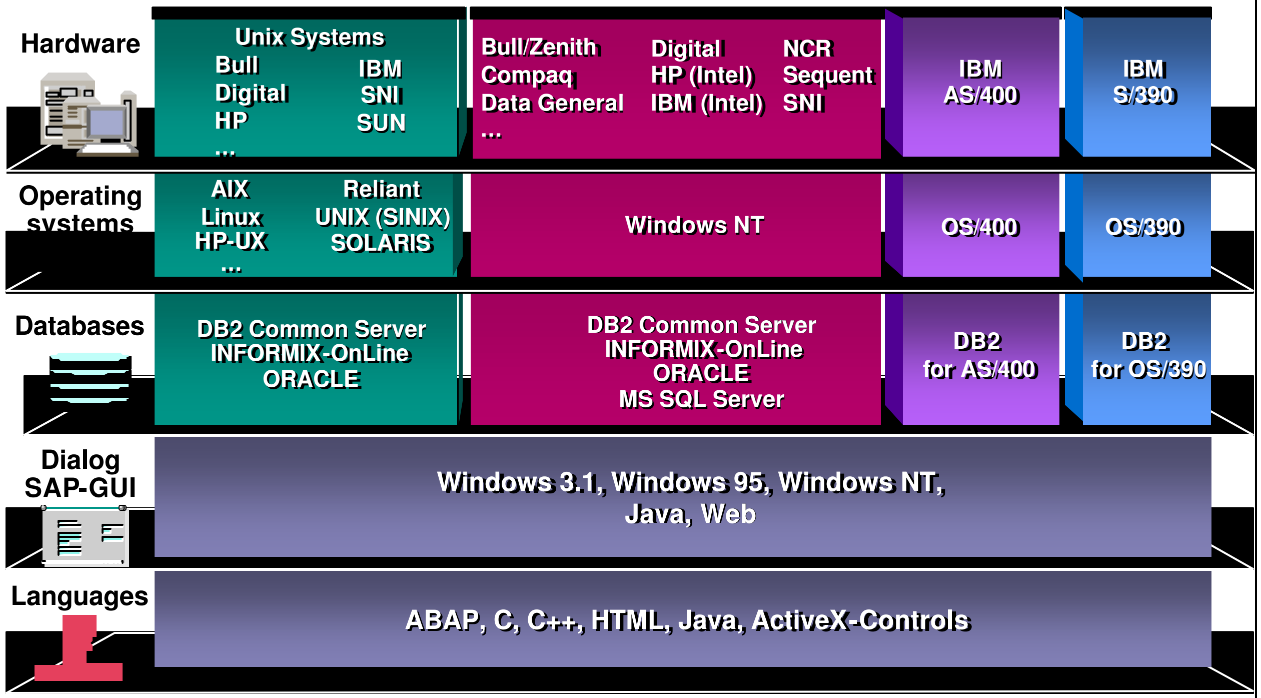
Simple Client/Server Configurations



- The fundamental services in a business application system are presentation services, application services and database services.
- In the following, the SAP R/3 System is used as an example of business application software.
- In a **single-tier** SAP R/3 System configuration, all processing tasks are performed by one host. This is classic mainframe processing.
- **Two-tier** SAP R/3 System configurations are usually implemented using special presentation servers that are responsible solely for formatting the graphical interface. Many R/3 System users use Windows PCs, for example, as presentation servers.

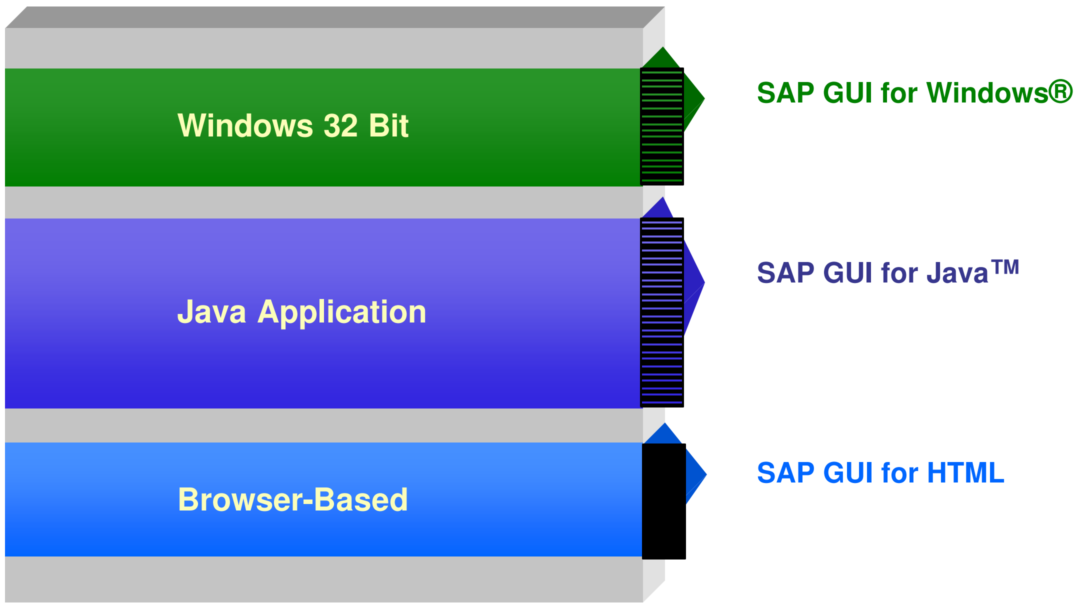
  An alternative two-tier configuration (not shown) is to install powerful desktop systems and to use these for presentation and applications (two-tier client/server). These configurations are especially suited to applications with high processor demands (for example simulations or for software developers), but are not implemented in the R/3 environment, other than for test purposes, due to the the additional administrative costs.
- In a **three-tier** configuration, each tier is on a separate host. Several different application servers can operate at the same time, using data from the database server. To ensure that the load on individual servers is as even as possible and to achieve optimal performance, you can use special application servers for individual application areas such as sales and distribution or financial accounting (Logon and Load Balancing).
- In the mySAP.com environment, more complex client/server configurations of more than three tiers are possible and implemented. Additional tiers could be: Internet Transaction Server (ITS), web server, mySAP.com Workplace Server, and others.

Technology Environment



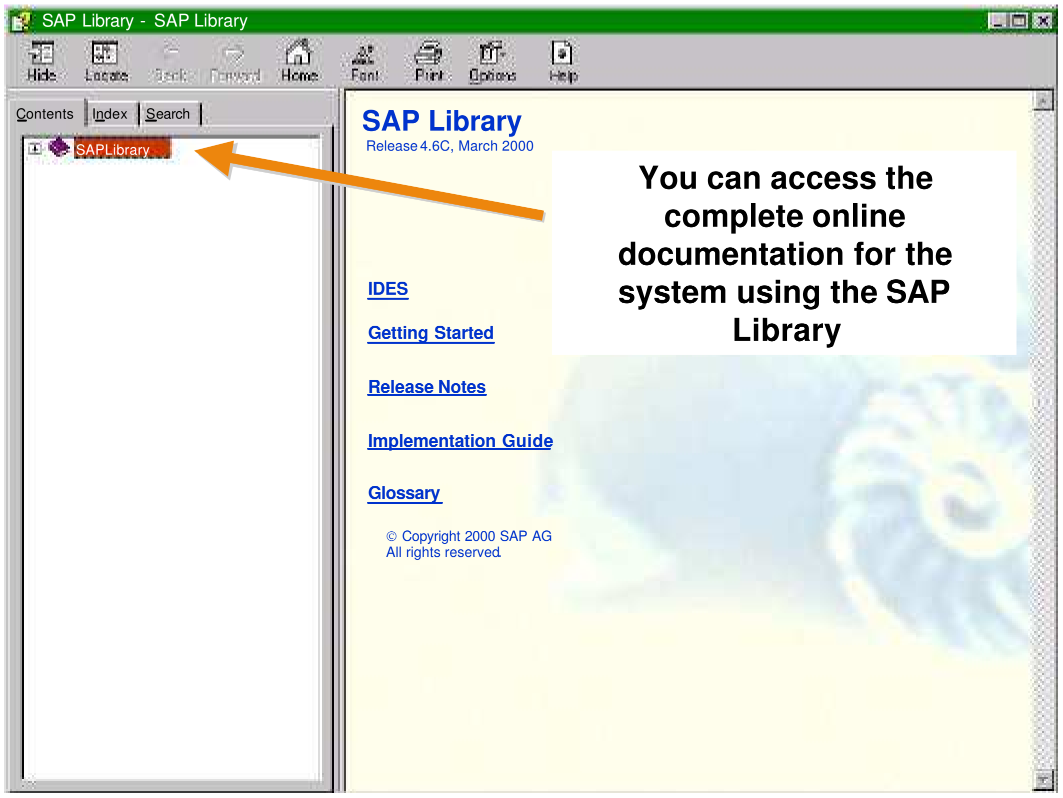
Hardware	Unix Systems Bull IBM Digital SNI HP SUN ...	Bull/Zenith Digital NCR Compaq HP (Intel) Sequent Data General IBM (Intel) SNI ...	IBM AS/400	IBM S/390
Operating systems	AIX Reliant Linux UNIX (SINIX) HP-UX SOLARIS ...	Windows NT	OS/400	OS/390
Databases	DB2 Common Server INFORMIX-OnLine ORACLE	DB2 Common Server INFORMIX-OnLine ORACLE MS SQL Server	DB2 for AS/400	DB2 for OS/390
Dialog SAP-GUI	Windows 3.1, Windows 95, Windows NT, Java, Web			
Languages	ABAP, C, C++, HTML, Java, ActiveX-Controls			

SAP GUI Types



- The presentation interface **SAPGUI** (GUI = **Graphical User Interface**) implements the platform-specific input and output functions of an SAP business application. The SAP GUI is primarily based on the Windows Style Guide and is available for several platforms providing the same functions for each. If you have learned to use the SAP GUI on one platform, with the exception of a few small platform-specific GUI attributes, you can use the system on another platform exactly the same as before.
- The presentation software implements the graphical user interface using the functions provided by the relevant presentation environment.
- As of SAP R/3 Release 4.6B you have a choice between the “classic” SAP GUI and a number of alternative access possibilities:
  - *SAP GUI for HTML*
  - *SAP GUI for Java*™

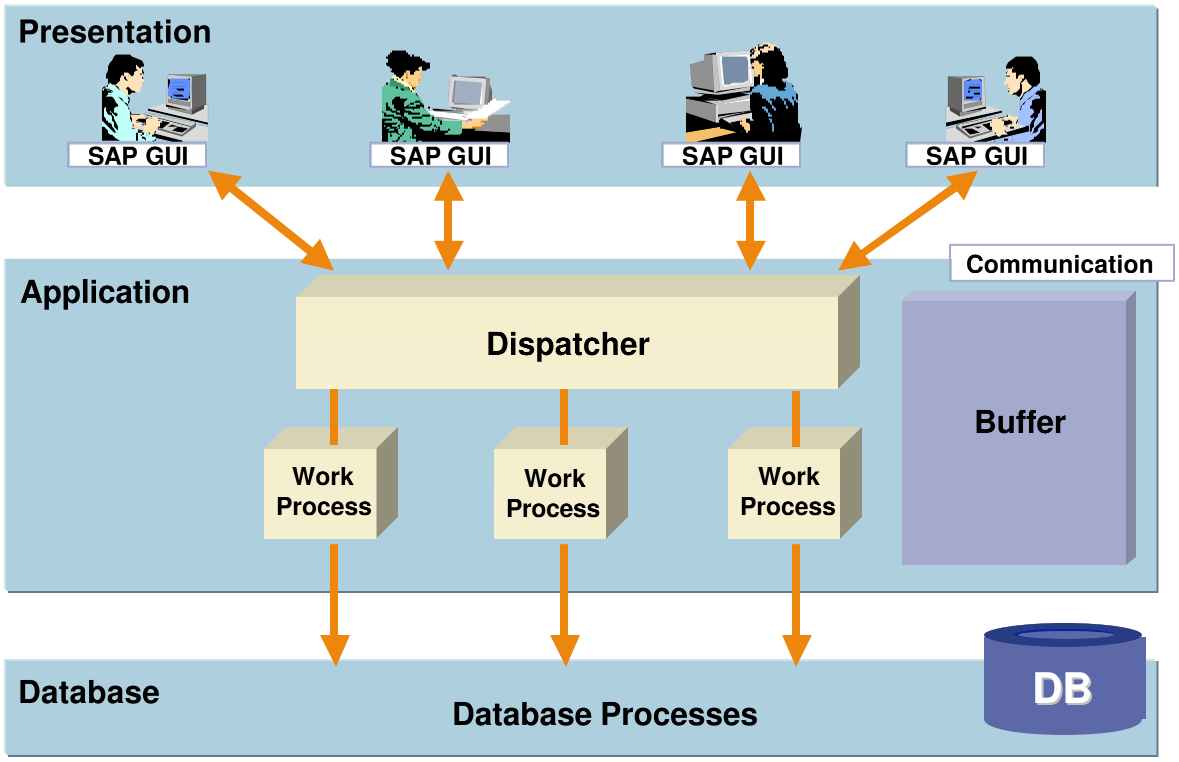
SAP Library



You can access the complete online documentation for the system using the SAP Library

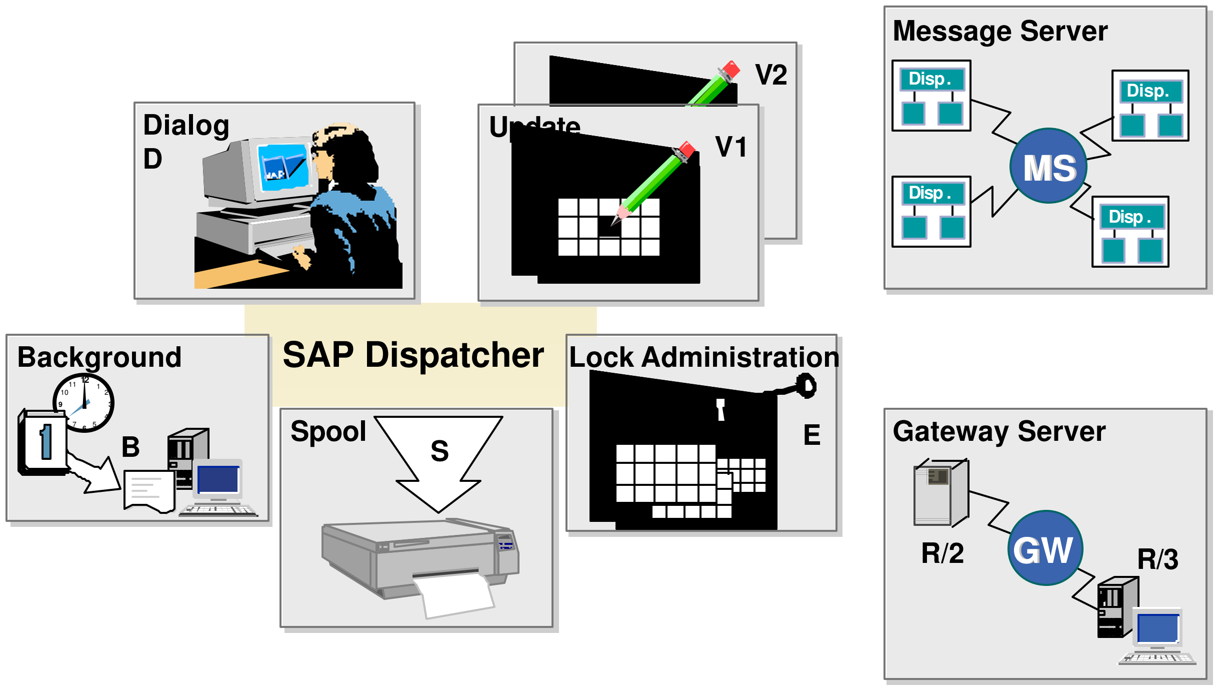
- SAP R/3 Systems provide comprehensive online help. You can display the help from any screen in the system. You can always request help using the *Help* menu or using the relevant icon (the yellow question mark).
- You can access the SAP Library quickly and comfortably by using the SAP Service Marketplace. There you can find the *SAP Help Portal* under *Knowledge and Training,* where you can not only access Help in HTML format, but can also perform efficient full-text searches in the SAP Library. If you have the SAP Library installed, you also have, of course, these opportunities within your company.
- You can access the Help Portal directly at *http://help.sap.com*

Processing User Requests



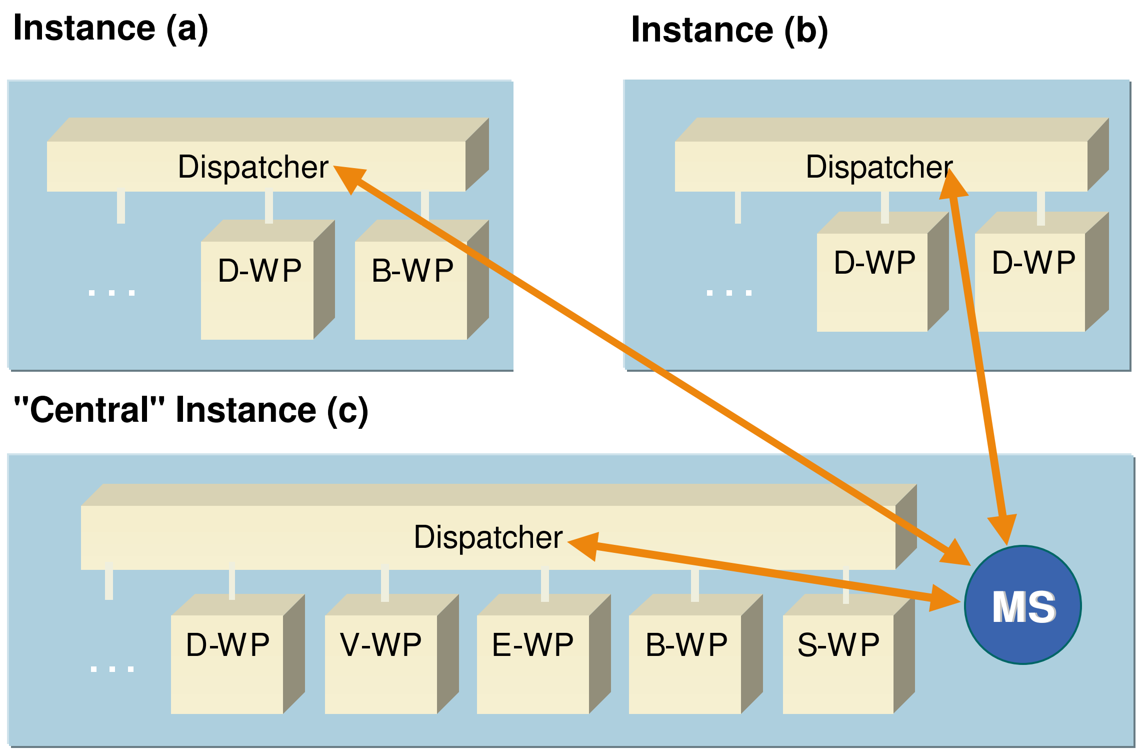
- The dispatcher is the central process of the application server. The dispatcher manages, in association with the operating system, the resources for the applications written in ABAP. The main tasks of the dispatcher include distributing transaction load to the work processes, connecting to the presentation level, and organizing communication.
- User input is received by the SAP presentation program SAPGUI, converted into its own format and then sent to the dispatcher. The processing requests are then saved by the dispatcher in request queues and processed according to a *first in-first-out* principle.
- The dispatcher distributes the requests one after the other to available work processes. Data is actually processed in the work process, although the user, who creates the request using the SAP GUI, is not always assigned the same work process. There is no fixed assignment of work processes to users.
- Once the data has been processed, the processing result from the work process is sent via the dispatcher back to the SAP GUI. The SAP GUI interprets the received data and generates the output screen for the user with the help of the operating system on the front end computer.
- During initialization of the mySAP.com component system, the dispatcher executes the following actions among others: It reads the system profile parameters, starts work processes, and logs on to the message server (this service will be explained later).

mySAP.com Processes



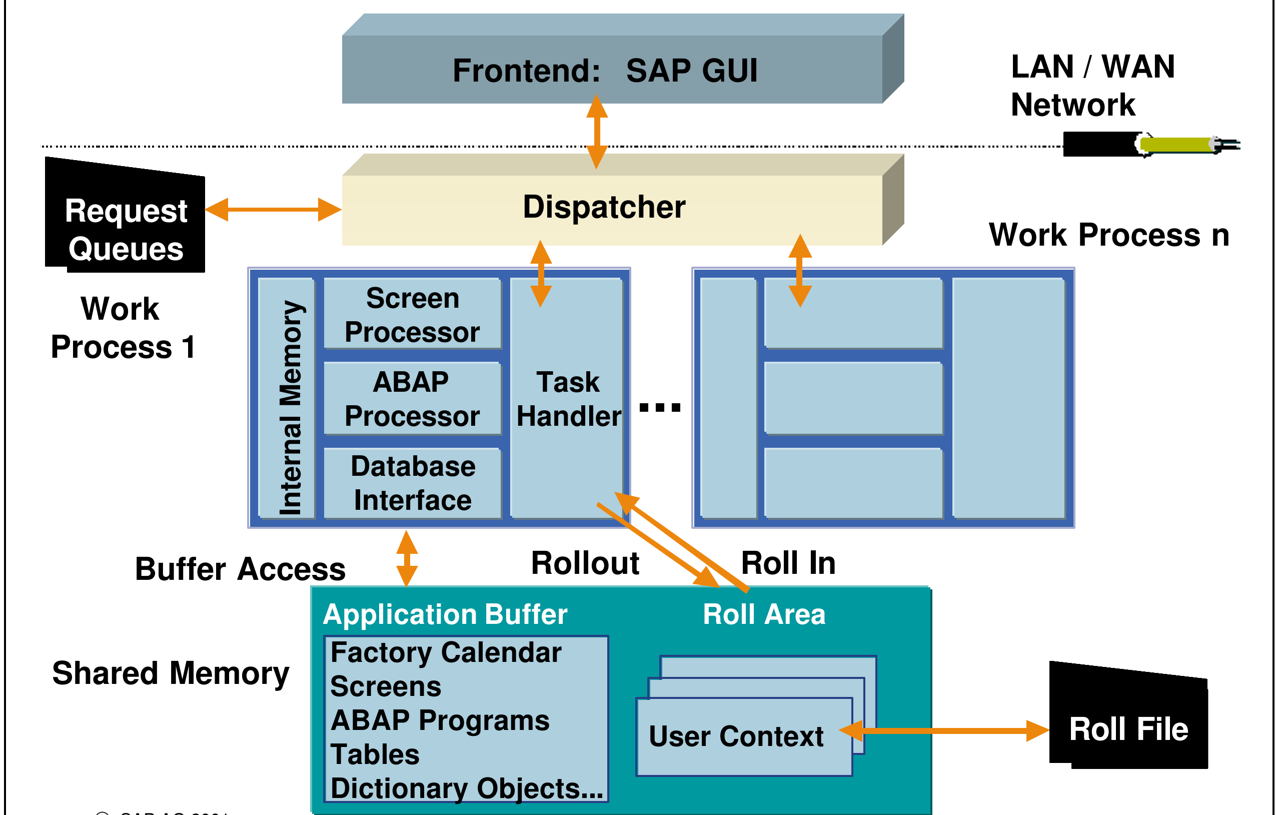
- The operating system views the SAP runtime system as a group of parallel, cooperating processes. On each application server these processes include the dispatcher as well as work processes; the number of work processes depends on the available resources. Special work processes may be installed for dialog processing, update, background processing and spooling.
- In addition to these work process types (dialog processing (D), update (V: for the German "Verbuchung"), lock management (E), background processing (B), spool (S), the runtime system provides two additional services for internal and external communication (below are the restrictions on the number of work processes):
  - The message server (MS or M) communicates between the distributed dispatchers within a mySAP.com System and is therefore the prerequisite for scalability using several parallel-processing application servers.
  - The gateway server (GW or G) allows communication between mySAP.com components such as SAP R/3 and SAP R/2 and external application systems. There is only one gateway process for each dispatcher process.
  - Dialog: Every dispatcher requires at least two dialog work processes
  - Spool: At least one for each mySAP.com System (more than one allowed for each dispatcher)
  - Update: At least one for each mySAP.com System (more than one allowed for each dispatcher)
  - Background processing: At least two for each mySAP.com System (more than one allowed for each dispatcher)
  - Enqueue: Only one enqueue work process is needed for each system

The Instance Concept



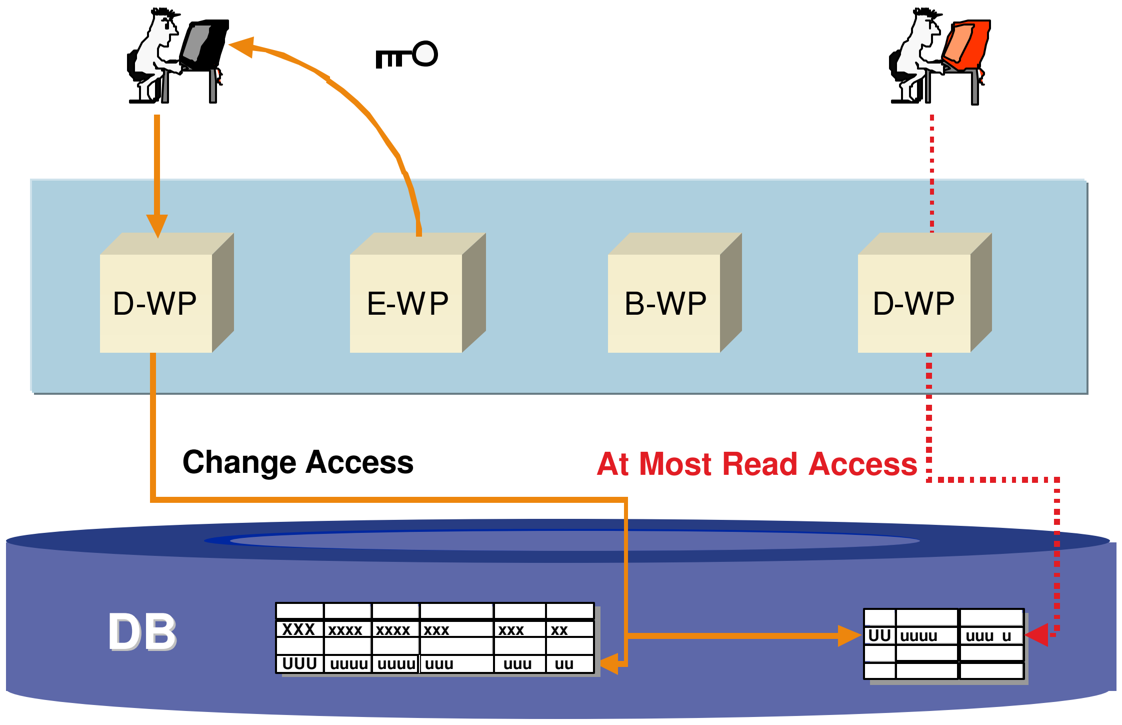
- An instance is an administrative unit that combines mySAP.com system components providing one or more services. The services offered by an instance are started or stopped together. You use a common instance profile to set parameters for all of the components of an instance.
- A central mySAP.com System consists of a single instance that provides all of the necessary services.
- Each instance has its own SAP buffer areas.
- The example illustrates how a background processing server (a) and dialog server (b) are set up. These instances, which provide specific services, generally run on separate servers, but can also run on the same server, if needed.
- The message server provides the application servers with a central message service for internal communication (for example: trigger update, request and remove locks, or trigger background requests).
- The dispatchers for the individual application servers communicate through the message server that is installed once in each mySAP.com System (it is configured in the system profile files).
- Presentation servers can also log on to an application server through the message server. This means that you can use the message server performance database for automatic load distribution (logon load balancing).

The Dialog Work Process



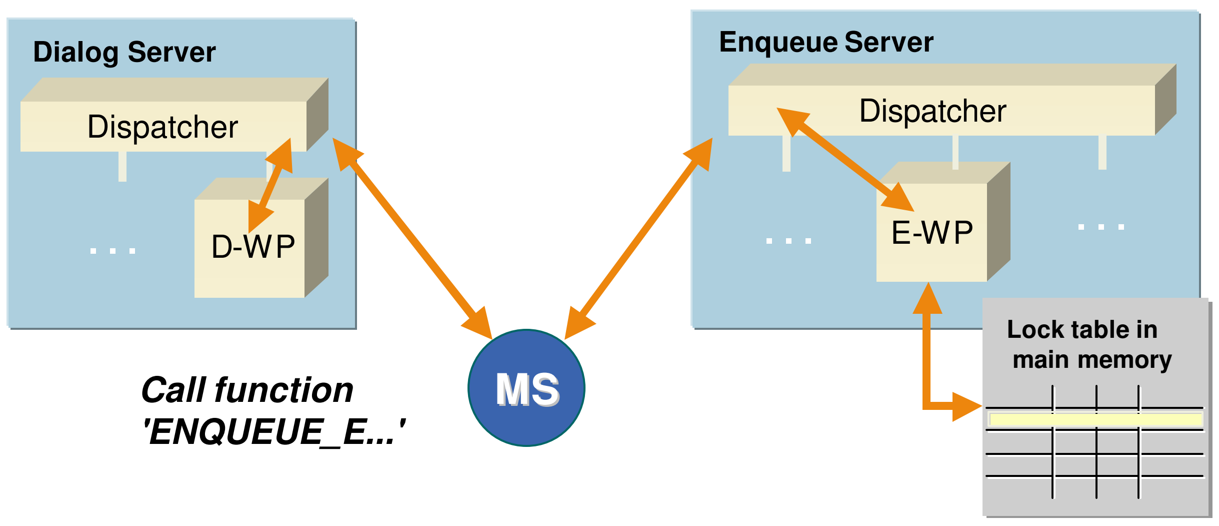
- The following components on the application level are involved in processing a dialog request:
  - The dispatcher as central control process
  - Work process queues (administered by the dispatcher) for incoming requests.
  - One of the diaog work processes
  - Buffers in shared memory and also possibly the roll file
- The **task handler** coordinates activity within a dialog work process. It activates the screen processor or the the ABAP processor (which control the screen flow logic and process ABAP statements, respectively) and executes the roll-in and roll-out of the user context.
- The memory management system differentiates between main memory areas that are available exclusively to a work process, and memory areas that can be used by all work processes. The memory space used exclusively by a work process stores session-specific data that must be kept longer than the duration of a work step. This data is automatically made available to the process at the start of a dialog step (rolled-in) and saved at the end of the dialog step (rolled-out). This data characterizes users (user context), such as their authorizations, administration information and additional data for the ABAP and dialog processor. It also contains data collected by the system in the preceding dialog steps in the running transaction (see slide *Work Process Multiplexing and SAP Transactions*).
  There are also additional memory areas for all processes in the shared memory for the factory calendar, screen, table, program, and other buffers.

Lock Administration in mySAP.com Systems



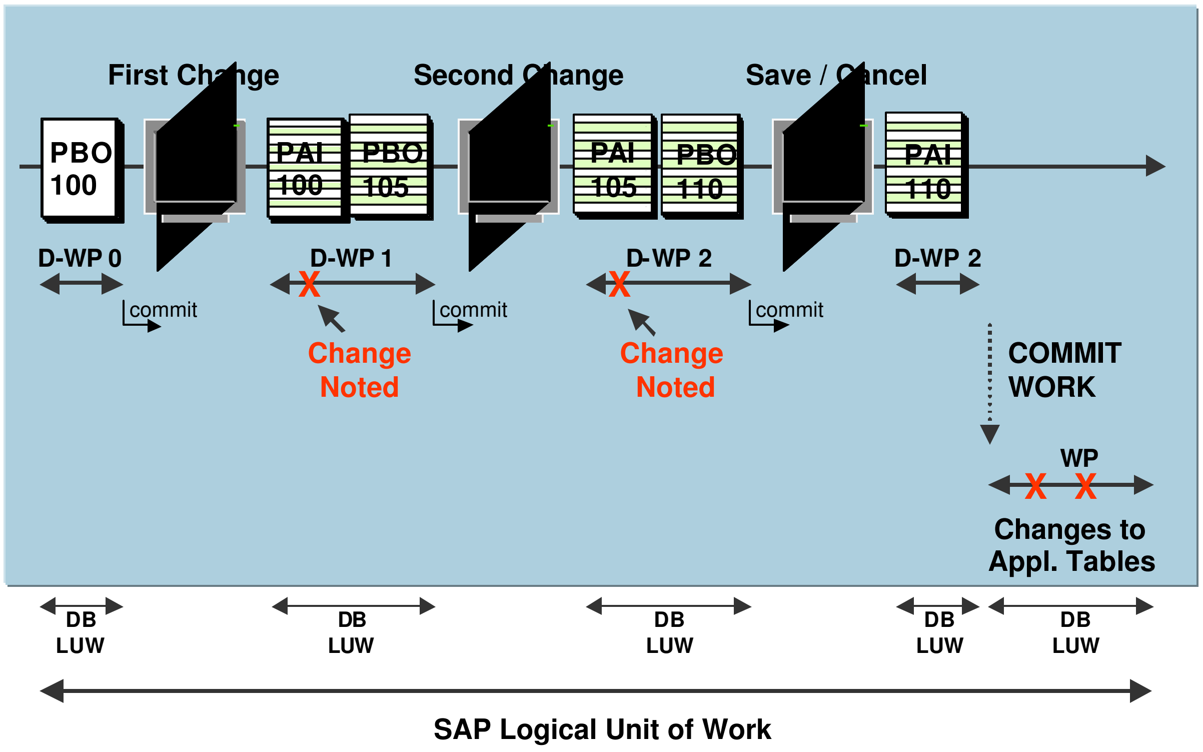
- The lock mechanisms present in the current relational database systems are not generally sufficient to handle business data objects (such as sales orders) that may affect several tables in the database. To coordinate several applications simultaneously accessing the same business object, the mySAP.com System provides its own lock management, controlled by the enqueue work process.
- In order for the system to execute lock requests, you must first define a lock object in the ABAP Dictionary. The lock object contains tables whose entries are to be locked. You can also have additional secondary tables using foreign key relationships (the name of a user-defined lock object must begin with "EY" or "EZ").
- You can specify the lock mode ("S": shared lock or "E": exclusive lock) for a lock object. An exclusive lock (mode "E") can only be set if no other user has set a lock ("E" or "S") on the data record. The same user can request additional "E" or "S" locks within a transaction.
- If a lock object is activated, the system generates an ENQUEUE and a DEQUEUE function module. These function modules have the names ENQUEUE_<object_name> and DEQUEUE_<object_name>, and are used in ABAP coding to lock and unlock data.

Requesting a Lock from the Enqueue WP



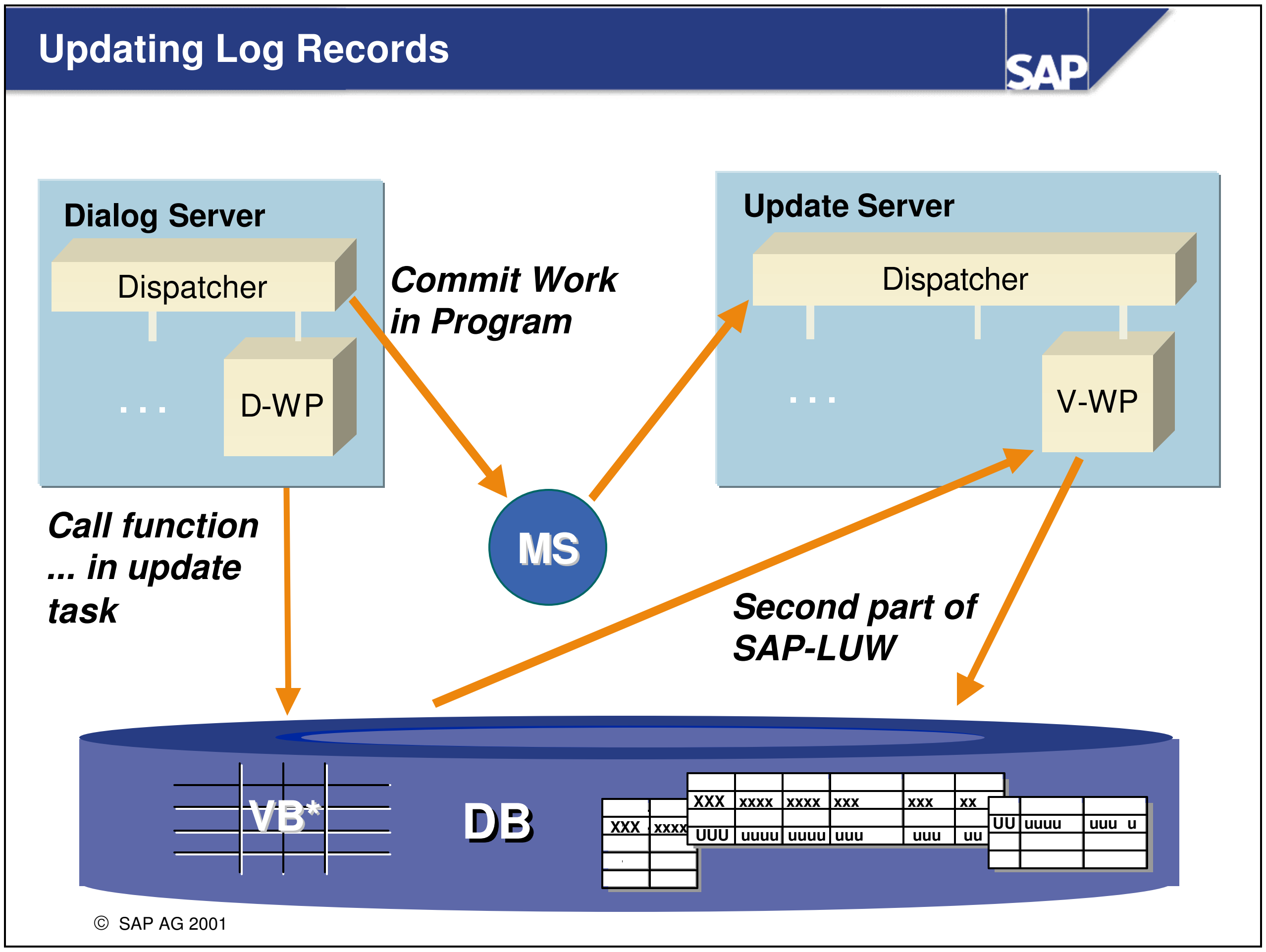
- When a lock is requested, the system checks to determine whether the requested lock conflicts with any entries in the lock table. If there are conflicts, the lock request is rejected. The application program can then inform the user that the requested operation cannot currently be executed.
- The locks (enqueues) are administered by the enqueue work process using the lock table. The lock table is stored in the main memory of the server where the enqueue work process is running. In the example shown above, the dialog work process and the enqueue work processes are not located on the same application server, they communicate through the message server.
- Locks set by an application program are either reset by the application program itself, or using a special update program (second part of the SAP Logical Unit of Work). Locks, which are inherited in this way by an update work process, are also written to a file at operating system level. In transaction SM12, the locks held by the update program are colored blue, while the locks held in the dialog work process are colored black.

Asynchronous Update

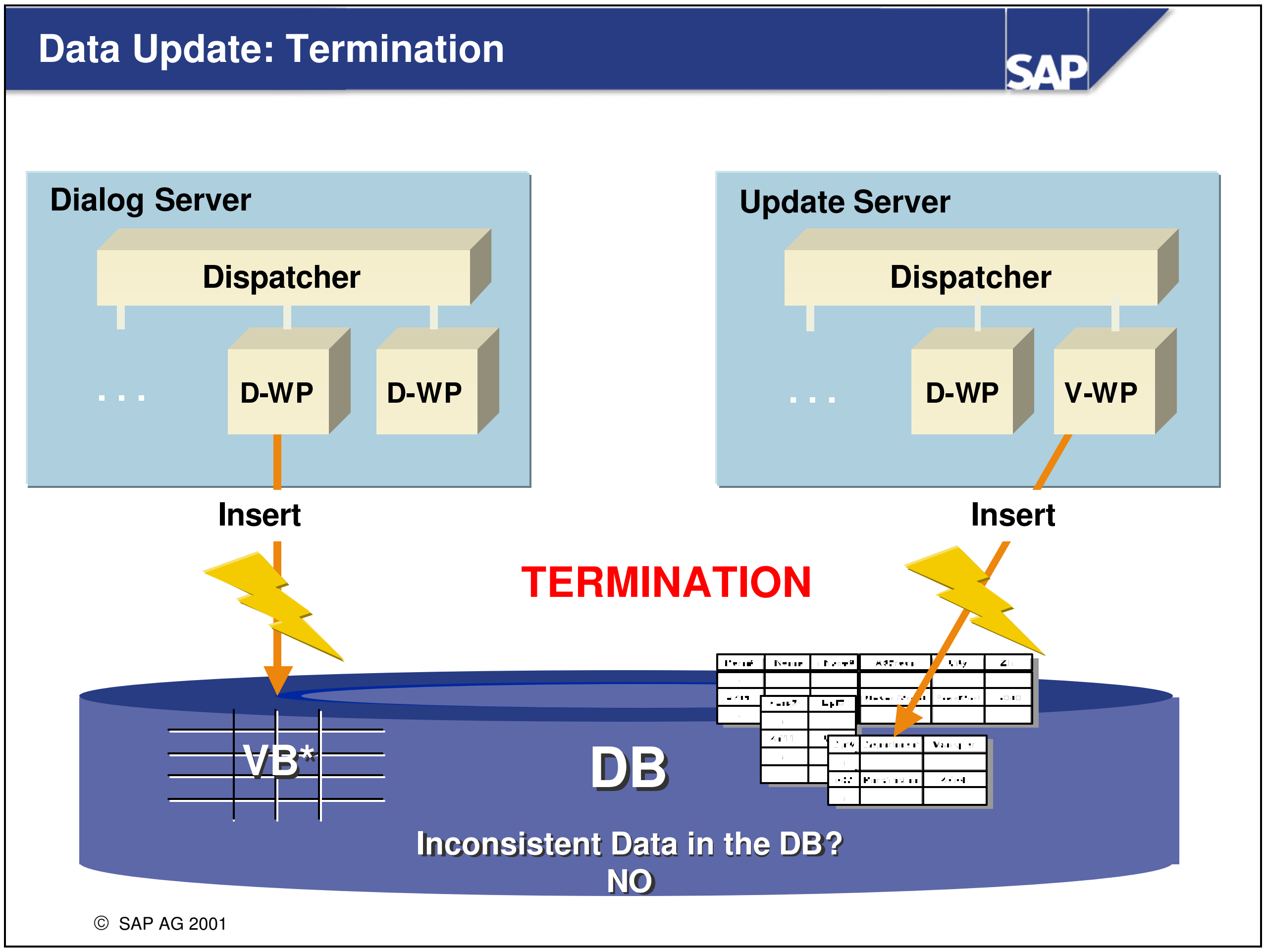


- The transaction concept, in as far as it has been previously explained, corresponds to the term **Logical Unit of Work** (LUW)
- However, as today's database systems do not support cross-process transaction flow, we must differentiate between the elementary processing steps (LUWs) in the SAP R/3 System and those in the database system (SAP-LUW / DB-LUW). A DB-LUW is fully executed, or not at all. The DB - LUW moves the database from one consistent state to the next. This means that the data is logical and correct before as well as after the LUW; this applies to both DB - LUW and SAP - LUW.
- The start of an SAP transaction is also the start of an SAP-LUW. SAP-LUWs are completed by a "COMMIT WORK" statement in the ABAP coding or by the end of the corresponding asynchronous update (second part of the SAP-LUW). As explained previously, each dialog step in an SAP - LUW is processed by one work process, as is the case for the DB - LUW. Each database change is executed in its own DB-LUW.
- The asynchronous updating usually used in an SAP - LUW allows the system to temporarily collect changes made by users and then, at the end of the dialog phase (in the second part of the SAP - LUW), make the necessary changes to the database in a separate update work process To ensure data consistency, the resulting database change (like every "dialog step change") is executed in only one DB - LUW.

Updating Log Records



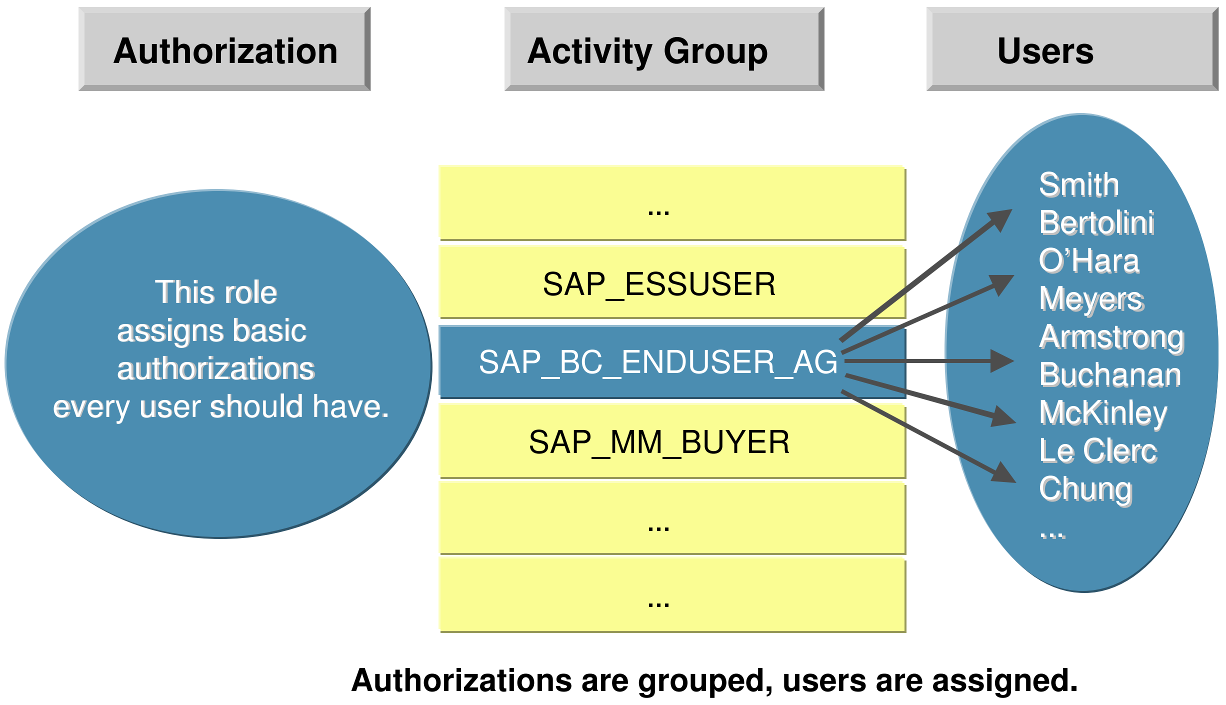
- When the ABAP keyword CALL FUNCTION '…' IN UPDATE TASK is processed during asynchronous update the data changes are stored as log records in temporary tables VB*. These system tables store data changes made by a user within an SAP transaction. The log record contains the names of the update routines to be executed, and all the data required to make the changes to the database.
- The update itself is triggered by the ABAP statement COMMIT WORK specified in the last dialog step of an SAP transaction. The locks set by the application program using the enqueue work process are passed to the update task. If the user cancels the SAP transaction during the dialog phase, or if the transaction terminates for another reason, the changes to be made to the database are discarded. In the second part of the SAP-LUW, the update work process reads the log records from the VB* tables and updates the corresponding application tables in the database according to the changes buffered in the VB* tables.
- During the update, errors cannot be corrected interactively by the user. Instead, the system terminates processing of the current update components. Users are automatically notified by express mail when an update terminates. The system has to handle the incorrect update that terminated (see next slide).



Data Update: Termination

- If a dialog work process terminates when writing data to VBLOG, VBLOG will contain incomplete data that will not be updated. These entries can be automatically deleted the next time you start the system. The application tables themselves remain unchanged.
- An asynchronous update may terminate for a number of reasons. Report VBTST300 is an example. If you execute this report using option "I" instead of the default "U", then an insert is transmitted to a table. If this insert is then repeated in the same table, this triggers the exception condition "Duplicate Key" in the coding because an entry already exists in the table under this key.
- When an update terminates, the system sends an express mail to the user who triggered the update. Any further steps to be taken must be carried out by the system administrator.
  - Transaction SM13 provides system administrators with analysis tools to handle terminated updates. Once the error that caused the termination has been corrected (for example, hardware damage repaired) the end user should repeat the update.
  - In the case of terminated V2 updates, you can update their log records later. You cannot do this for V1 updates, however, as this may create inconsistent data.

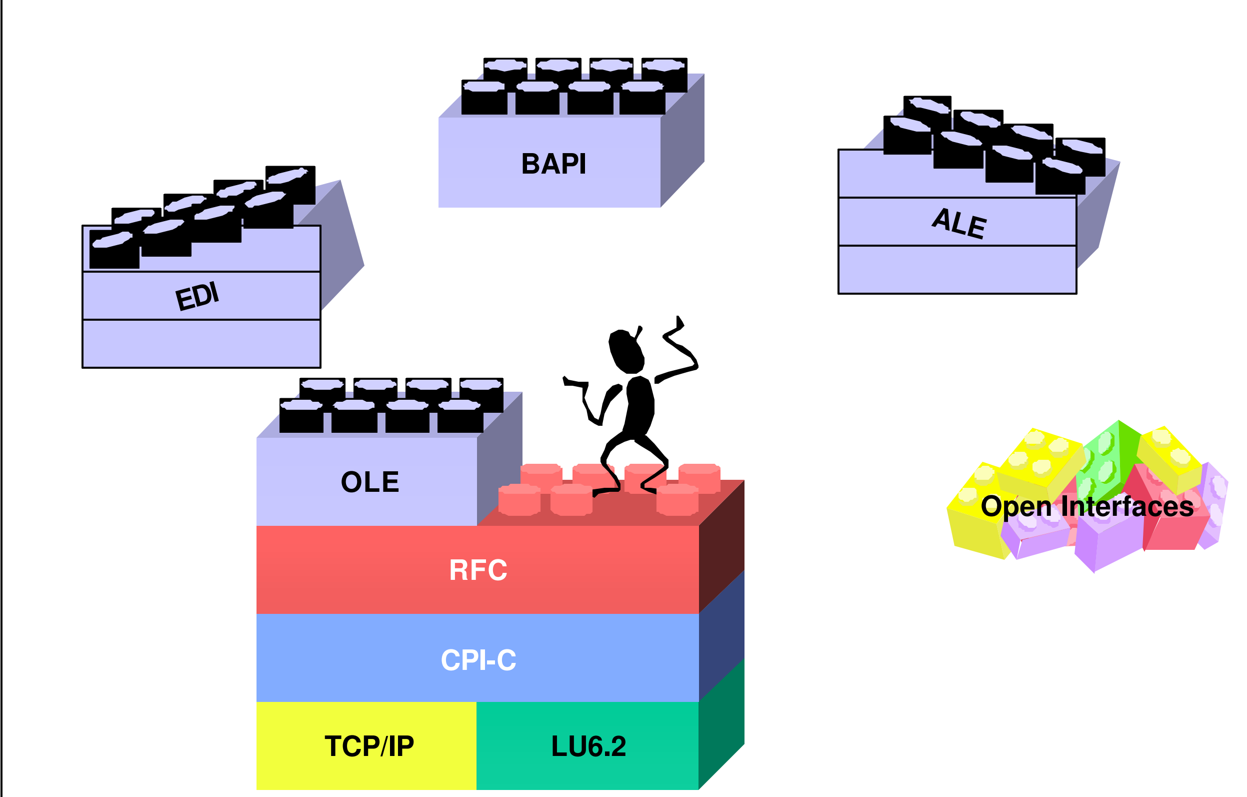
The Authorization Concept in SAP Business Applications



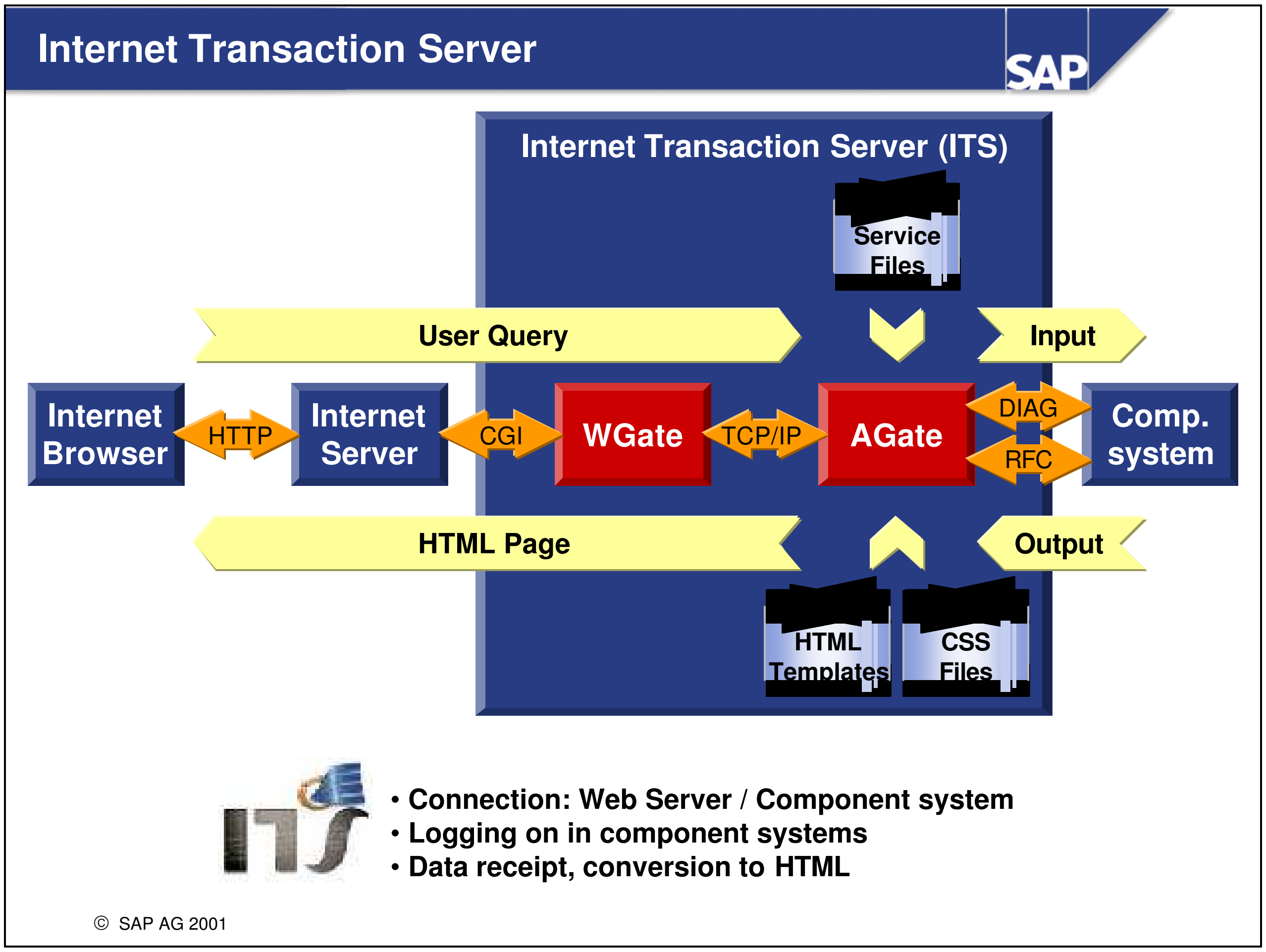
Authorizations are grouped, users are assigned.

- The SAP authorization concept recognizes a large number of different authorizations. These are all managed centrally in the user master record for every user.
- Authorizations are not directly assigned to users, but stored in work center descriptions (profiles).
- These profiles are generated using the Profile Generator, which administers the profiles as roles. Roles have additional attributes compared to profiles, such as validity period.
- Users can be assigned one or more roles and are then assigned the authorizations associated with those roles.
- SAP R/3 Release 4.6 contains a large number of pre-defined roles. You can use these as is or copy and tailor them to your specific needs.
- Since SAP R/3 Release 4.5, SAP R/3 Systems have also included the Central User Administration (CUA). Using this, it is possible to centrally administer users and their authorizations beyond system borders. This means that once CUA is installed, users and their role assignments are only maintained in one system, and these settings are transferred to other systems using ALE. Roles and profiles continue to be administered on the relevant component systems.

SAP Applications and Open Interfaces



- mySAP.com is an **open system**. It supports a variety of network communication protocols. Information can be exchanged between component systems and non-SAP applications.
- SAP supports the Transmission Control Protocol / Internet Protocol (TCP/IP) and System Network Architecture: Logical Unit 6.2 (SNA LU6.2) protocols. Communication within mySAP.com uses the standard protocol TCP/IP. LU6.2 was developed by IBM and is used to communicate with mainframe-based SAP R/2 Systems.
- SAP R/3 application programming supports the following communication interfaces: common programming interface communication (CPI-C), remote function call (RFC), and object linking and embedding (OLE) automation. Other interfaces are based on these technologies, for example communication using Business Application Programming Interfaces (BAPIs), Electronic Data Interchange (EDI) or according to an Application Link Enabling (ALE) distribution model.
- For more information about communication, see the online documentation. You can also order a free "Interface Adviser" Knowledge CD from SAP that uses many practical examples to explain communication in mySAP.com Systems. See also the SAP Service Marketplace, which contains additional information, such as under the alias */int-adviser*.



Internet Transaction Server

- Connection: Web Server / Component system
- Logging on in component systems
- Data receipt, conversion to HTML

- The web server and component system is connected through SAP's **Internet Transaction Server** (ITS). The ITS is used for the following tasks:
  - Data exchange (for example using protocol conversion);
  - Logging on in a component system;
  - Forwarding of data to component systems, conversion of output into HTML;
  - Status administration.
- The ITS consists of two components:
  - **Wgate**: The interface to the Web server. The WGate and the Web server must be running on the same hardware. As well as the Common Gateway Interface (CGI), SAP supports the secure interfaces ISAPI (Microsoft) and NSAPI (Netscape).
    If you are using Microsoft's IIS Web servers, you can find the WGate as a DLL file in the *Scripts* folder.
  - **Agate**: Exchanges data with the component systems. It can be run on the same hardware as the WGate (single host) or on its own server (dual host). Several files that are stored on the AGate configure the ITS. Among these are:
    - Service files: Specify the component system and logon data (client, user, and password)
    - HTML Business Templates: HTML with additional commands, placeholders for display fields
    - CSS files: Describe the optical and functional layout of the generated HTML pages

SAP Web Application Server - Goals

- Bring business applications and the Web together.
- Support important standards.
- Allow ABAP developers to build complete, flexible Web applications.
- Drastically reduce the cost of building Web applications.
- Allow the development of new Web applications (inside-out) as well as Web front-ends to existing SAP components (outside-in).
- Manage all aspects of your Web applications in one system.
- Provide massive scalability based on the proven SAP Application Server ("Basis System").

SAP Web Application Server - Features

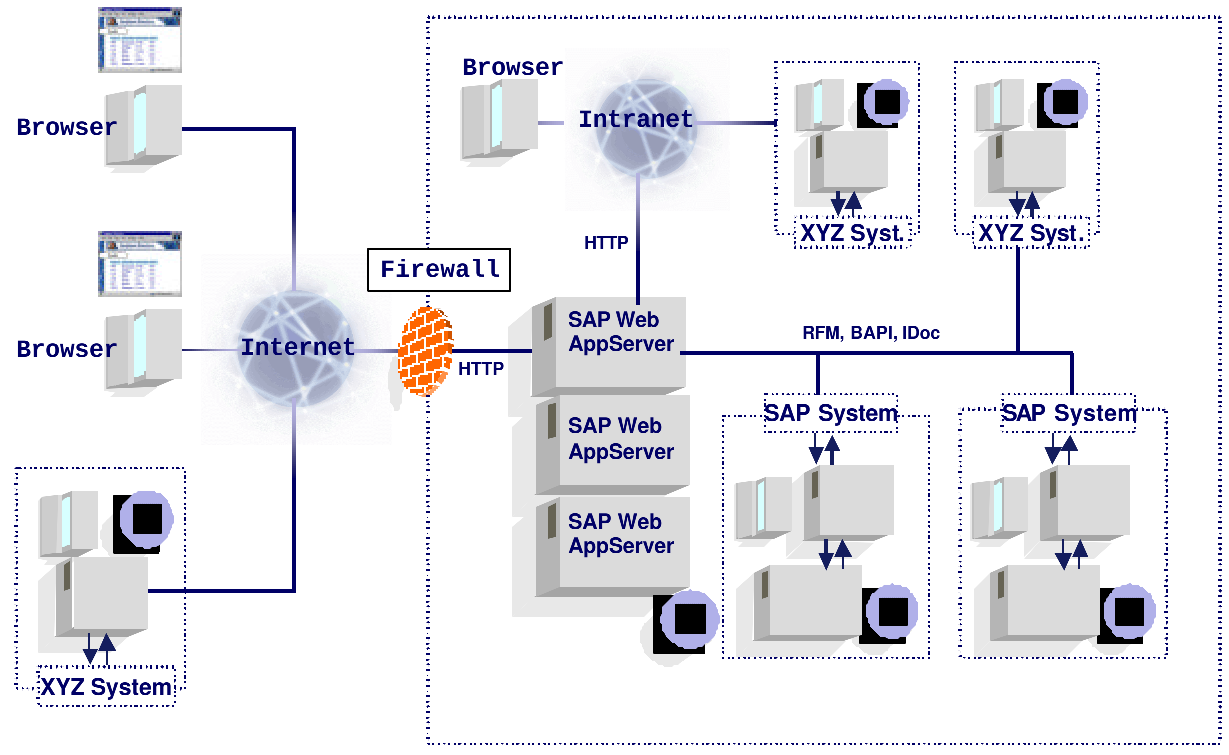
- Supports HTTP, HTTPS, SOAP, SMTP, WEBDAV, XML, XSLT.
- Supports stateless and stateful applications.
- Business Server Pages
- Web Control Framework
- Support for ABAP and JavaScript.
- Supports Unicode.
- Complete Life Cycle Management of your Web applications.
- "Inherits" all the advantages of the SAP Application Server.
  - Complete, integrated, multi-user development environment.
  - Stable, massively scalable runtime system with monitoring and debugging tools.

- Web AS is not a complete Web server in the sense that it runs your Perl scripts etc., but it can easily be used together with a traditional, non-SAP Web server.

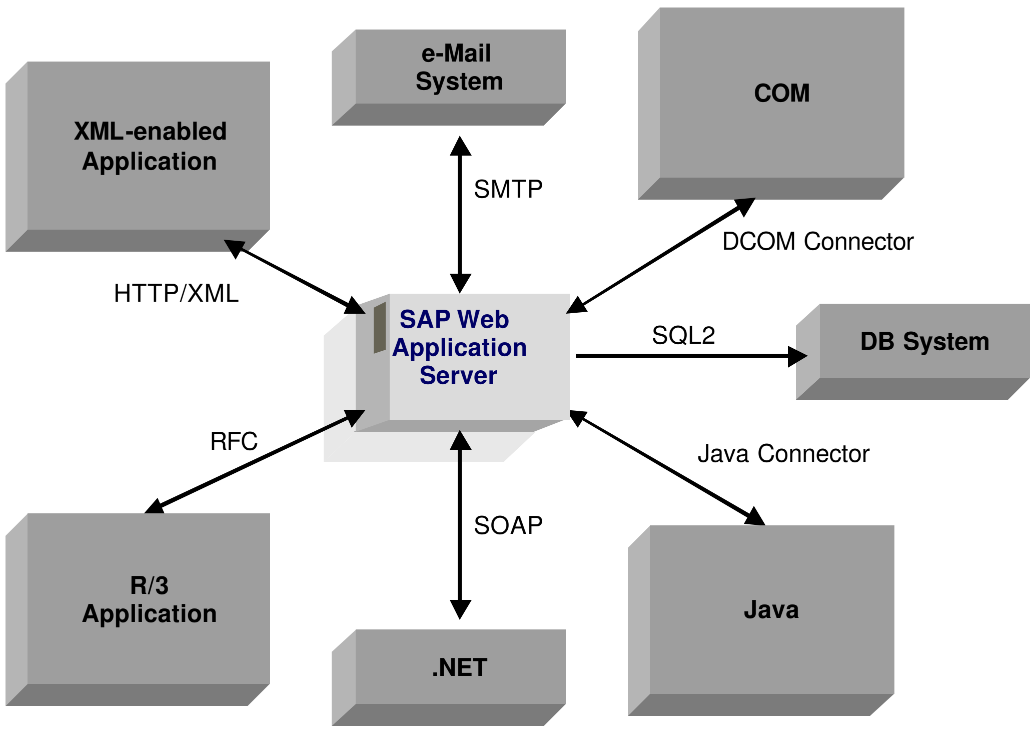
SAP Web AS Release History

- 5.0 FCS shipped in November 2000.
  - Not full-function, might be called a beta release.
  - 1st productive customer in April 2001.
  - Linux version available in the SAP Shop and at ftp://ftp.sap.com/pub/linuxlab/test_drive/WAS_testdrive.iso
- 6.10 GA approx. end of June 2001.
  - The next releases of CRM and other products will be based on that release.
- R/3 4.7 (R/3 Enterprise) will include the 6.10 kernel.
  - Will allow you to use inside-out SAP Web enabling for the R/3 applications.
  - FCS approx. end of 2001.

Architecture



Connectivity to Other Systems



Business Server Page (BSP)

Layout
- Static formatting directives (HTML, XML, WML, etc.)
- Client-side script code

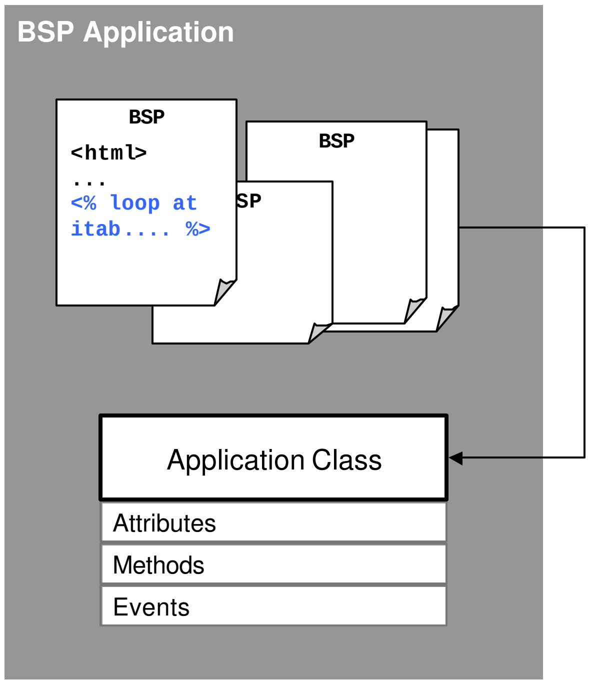
Server-side script code
- ABAP
- JavaScript

Predefined events to allow user-defined routines for
- Processing input data
- Data retrieval
- Input validation
- Error handling
- Navigation to next page

```
<%@page info="who is who" language="javascript"%>
<html>
<body>
<form method="post">
  <input type=text name="lastname"
         value="<%= ds_lastname %>">
  <input type=submit name="onInputProcessing(a)"
         value="New Search">
  ...
</form>
<table>
  <tr>
    <td> Lastname </td> <td> Firstname </td>
    <td> Phone Number</td>
  </tr>
<% for (i = 0; i < ds_persons.length; i++) { %>
  <tr>
    <td> <%= ds_persons[i].lastname  %> </td>
    <td> <%= ds_persons[i].firstname %> </td>
    <td> <%= ds_persons[i].phone    %> </td>
  </tr>
<% } %>
</table>
</body>
```

BSP Application

- Business Server Pages
- MIME objects (images, style sheets, ...)
- Application class that acts as container for the business logic



BSP Layout

Business Server Page

onInitialization

```
call method application->retrieve_persons
  exporting criteria = ds_lastname
  importing result   = ds_persons.
```

Layout

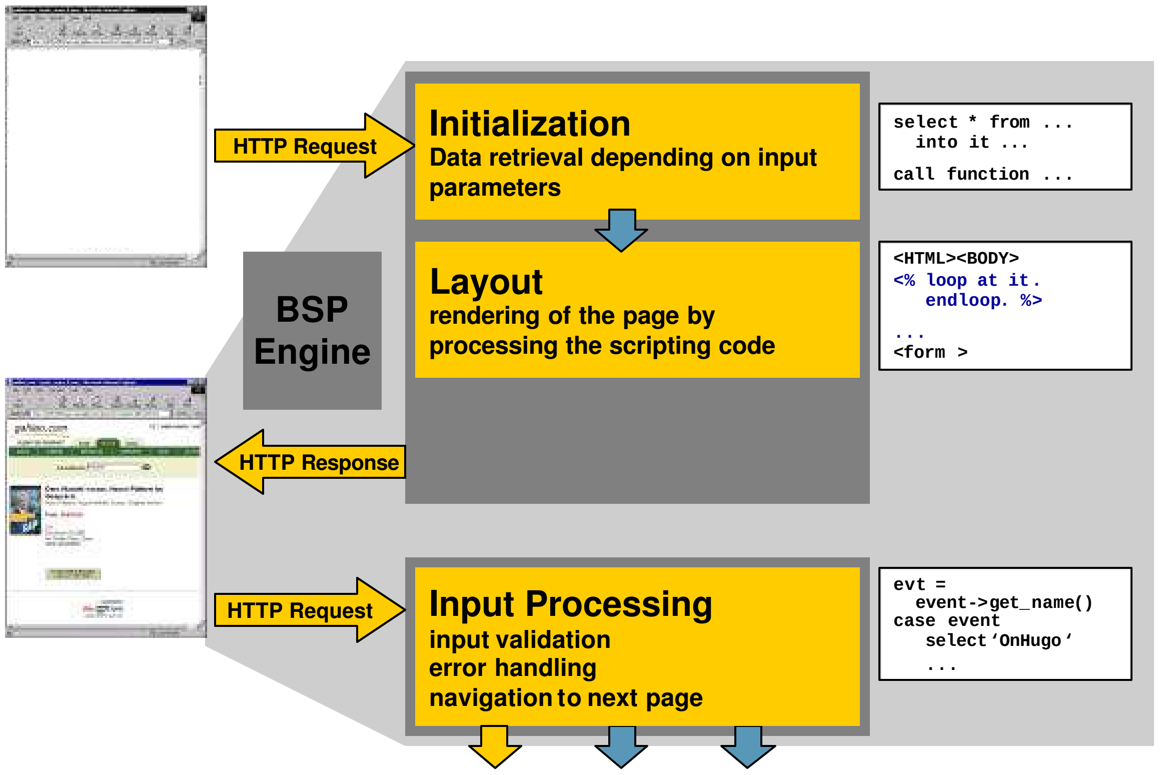
```
<%@page info="who is who" language="abap"%>
<html>
<body>
<form method="post">
  <input type=text name="lastname" value="<%= ds_lastname %>">
  <input type=submit name=”onInputProcessing(a)" value="New Search">
</form>
<table>
  <tr>
    <td> Lastname </td> <td> Firstname </td> <td> Phone Number </td>
  </tr>
<% data: wa like line of ds_persons.
   loop at ds_persons into wa %>
  <tr>
    <td> <%= wa-lastname  %> </td>
    <td> <%= wa-firstname %> </td>
    <td> <%= wa-phone     %> </td>
  </tr>
<% endloop. %>
</table></body></html>
```

...

onInputProcessing

The layout part contains script code mixed with static formatting directives (e.g. HTML).
The data sources provided by the initialization step are directly accessible in the layout step.

Business Server Pages - Control Flow



```
select * from ...
  into it ...

call function ...
```

```
<HTML><BODY>
<% loop at it .
   endloop. %>

...
<form >
```

```
evt =
  event->get_name()
case event
   select ‘OnHugo‘
   ...
```

Advantages of the Web Control Framework

- Additional abstraction layer allows
  - Browser-independent application programming
  - Easier customizing (important for customers' corporate branding)
  - WYSIWYG design environment
- The Web controls are implemented in the kernel and can be addressed from
  - ABAP
  - JavaScript
  - Java

Web Control Framework: Components

- A set of server-side controls
- Rendering services - both generic and browser-specific
- Services for
  - Persisting the view state of controls
  - Processing requests
  - Processing responses

WCF and BSP Applications

- SAP Web Application Server runs Business Server Pages applications - page-based approach
- It is possible to write a BSP using native HTML...

```
<table border=1>
   <th>Flight Number</th><th>Date</th>
   <th>Capacity</th><th>Occupied seats</th>
   <% data wa type sflight.
      loop at flight_tab into wa. %>
   <tr><td><%= wa-connid %></td>
     <td><%= wa-fldate %></td>
     <td><%= wa-seatsmax %></td>
     <td><%= wa-seatsocc %></td></tr>
   <% endloop. %>
</table>
```

WCF and BSP Applications

```
<bsp:TableView title="Flight Information"
               dataSrc="<%#flight_tab%>" />
```

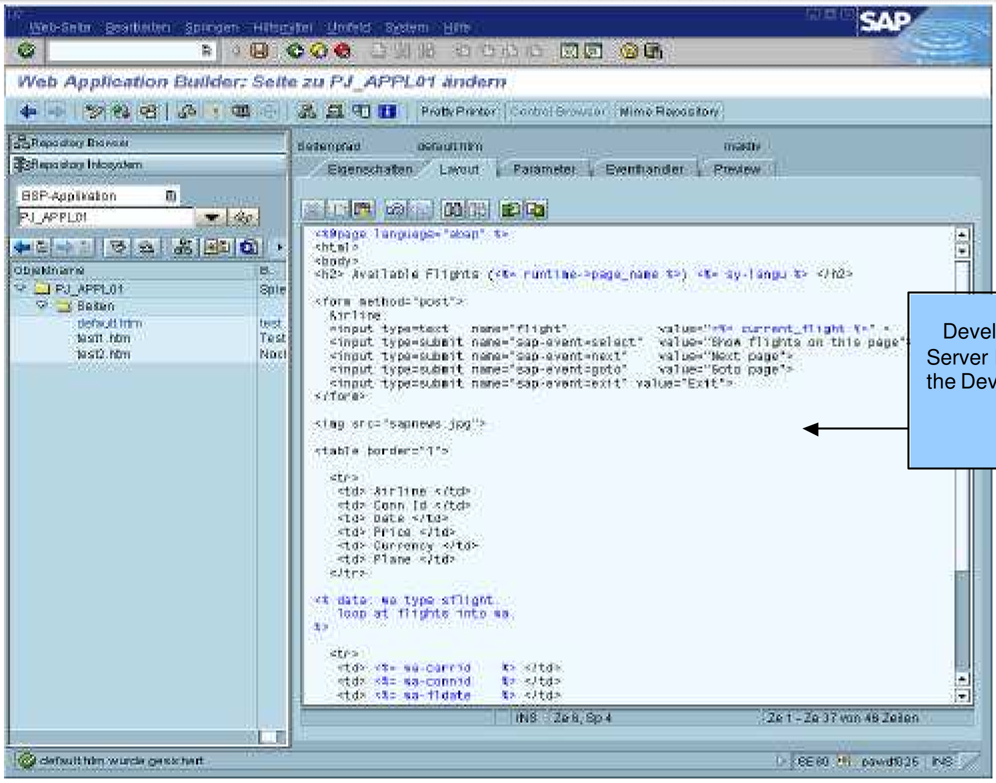
... But web controls are more convenient!

Easier to use (no HTML, JavaScript, CSS...)
Faster
Services automatically provide
- Data conversion
- Layout

Control Classes

- Button
- Checkbox
- Checkbox Group
- Document Head
- Document Head Include
- Dropdown List Box
- Image
- Input Field
- List Box
- Radio Button Group
- Table View
- Text View
- Tree View

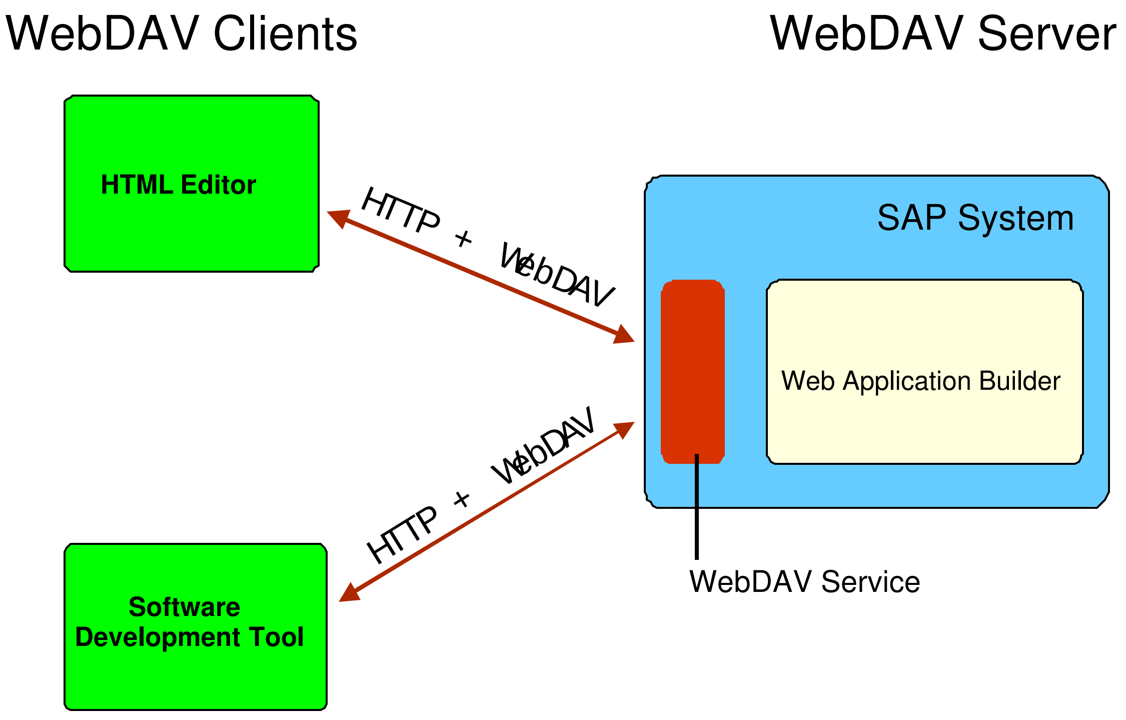
BSP Development Workbench



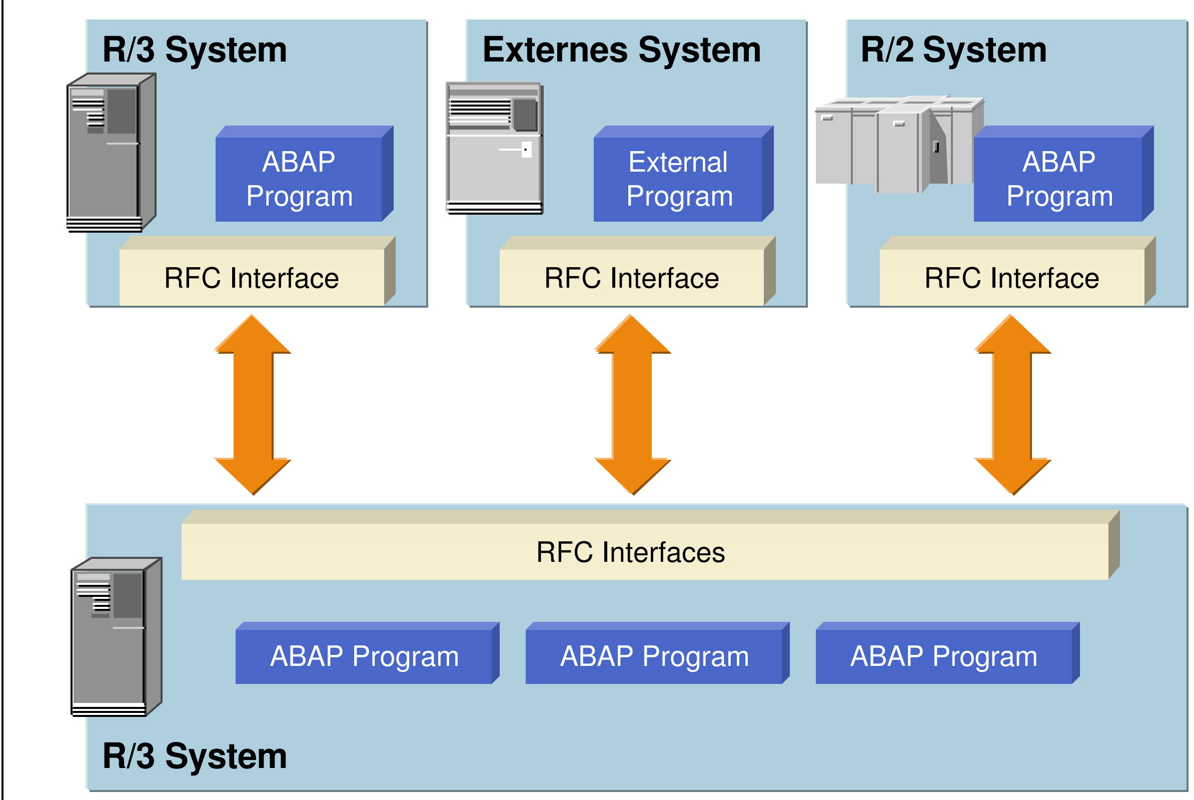
```
<%@page language="abap" %>
<html>
<body>
<h2> Available Flights (<%= runtime->page_name %>) <%= sy-langu %> </h2>

<form method="post">
  Airline
  <input type=text   name="flight"              value="<%= current_flight %>" >
  <input type=submit name="sap-event=select"  value="Show flights on this page">
  <input type=submit name="sap-event=next"    value="Next page">
  <input type=submit name="sap-event=goto"    value="Goto page">
  <input type=submit name="sap-event=exit" value="Exit">
</form>

<img src="sapnews.jpg">

<table border="1">

  <tr>
   <td> Airline </td>
   <td> Conn Id </td>
   <td> Date </td>
   <td> Price </td>
   <td> Currency </td>
   <td> Plane </td>
  </tr>

<% data: wa type sflight.
   loop at flights into wa.
%>

  <tr>
   <td> <%= wa-carrid     %> </td>
   <td> <%= wa-connid     %> </td>
   <td> <%= wa-fldate     %> </td>
```

Development of Business Server Pages is integrated in the Development Workbench (SE80)

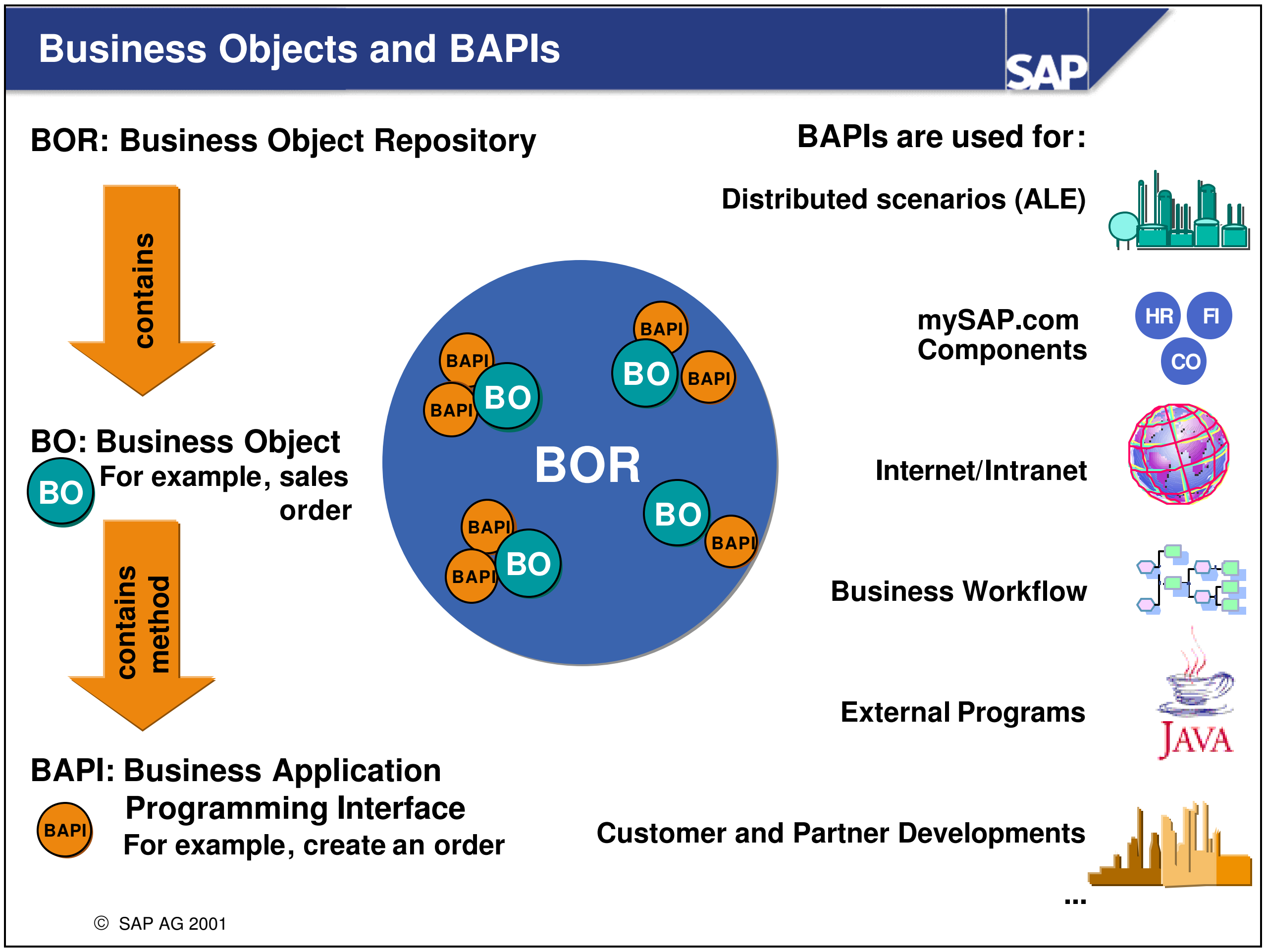
BSP Development and WebDAV



Remote Function Call



- Remote Function Call (RFC) is a communications interface based on CPI-C, but with more functions and easier for application programmers to use. You can use SAP R/3 and SAP R/2 Systems as well as external applications as RFC communication partners. For more information, see SAP Notes 13903 and 116051.
- RFC is the protocol for calling special subroutines (function modules) over the network. Function modules are comparable with C functions or PASCAL procedures. They have a defined interface through which data, tables and return codes can be exchanged. Function modules are managed in the SAP R/3 System in their own function library, called the Function Builder (transaction SE37).
- The Function Builder provides application programmers with a useful environment for programming, documenting and testing function modules that can be called locally as well as remotely.
- You maintain the parameters for RFC connections using transaction SM59. The SAP R/3 System is also delivered with an RFC-SDK (Software Development Kit) that uses extensive C libraries to allow external programs to be connected to the SAP R/3 System.



Business Objects and BAPIs

BOR: Business Object Repository

contains

BO: Business Object
For example, sales order

contains method

BAPI: Business Application Programming Interface
For example, create an order

BAPIs are used for:

Distributed scenarios (ALE)

mySAP.com Components

Internet/Intranet

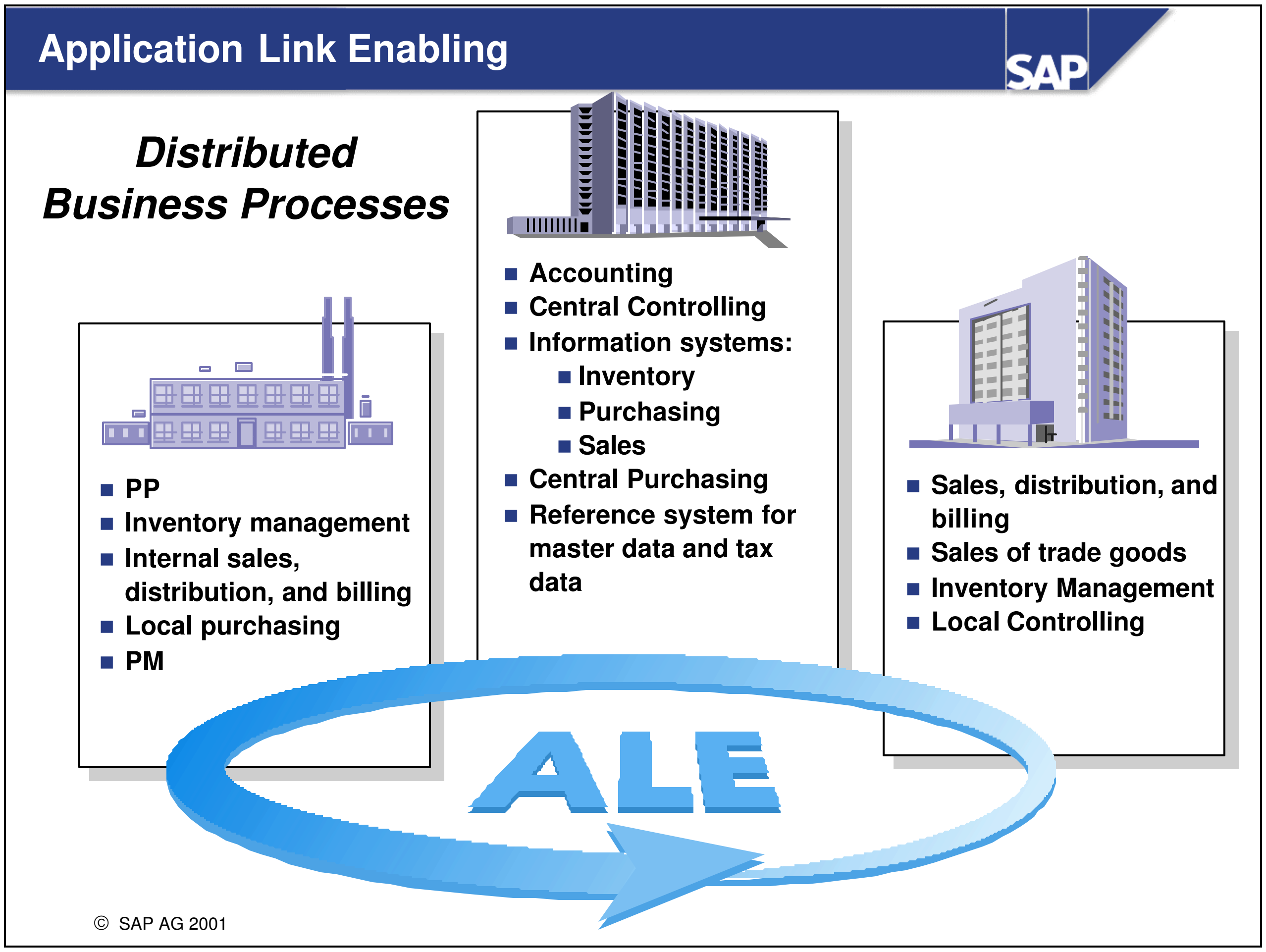
Business Workflow

External Programs

Customer and Partner Developments

...

- **Business objects** form the basis for communicating on high (user-friendly) network layers. For example, they enable the SAP R/3 System to support the Internet and desktop programs to be connected. The goal of SAP's object-oriented strategy is to integrate objects at a business level rather than on a purely technical level. You can create and manage business objects using transaction SWO1.
- Business objects:
  - Form the basis of well-defined communication between client / server systems.
  - Are business-oriented: there are objects such as "customer", "order" or "employee".
  - Provide business functions (methods). For a "customer" object, for example, there are "Create customer" and "View customer" methods. These names support clear and therefore error-free programming.
  - Are managed centrally in the SAP R/3 System in the Business Object Repository (BOR).
- Business Application Programming Interfaces (**BAPIs**) are functional interfaces. They use the business methods from the business objects. BAPIs may be addresses within or outside the SAP R/3 System. You can create and manage BAPIs using the transaction BAPI.
- For specifications and more information about BAPIs, see the alias "*bapi*" in the SAP Service Marketplace.



Application Link Enabling

Distributed Business Processes

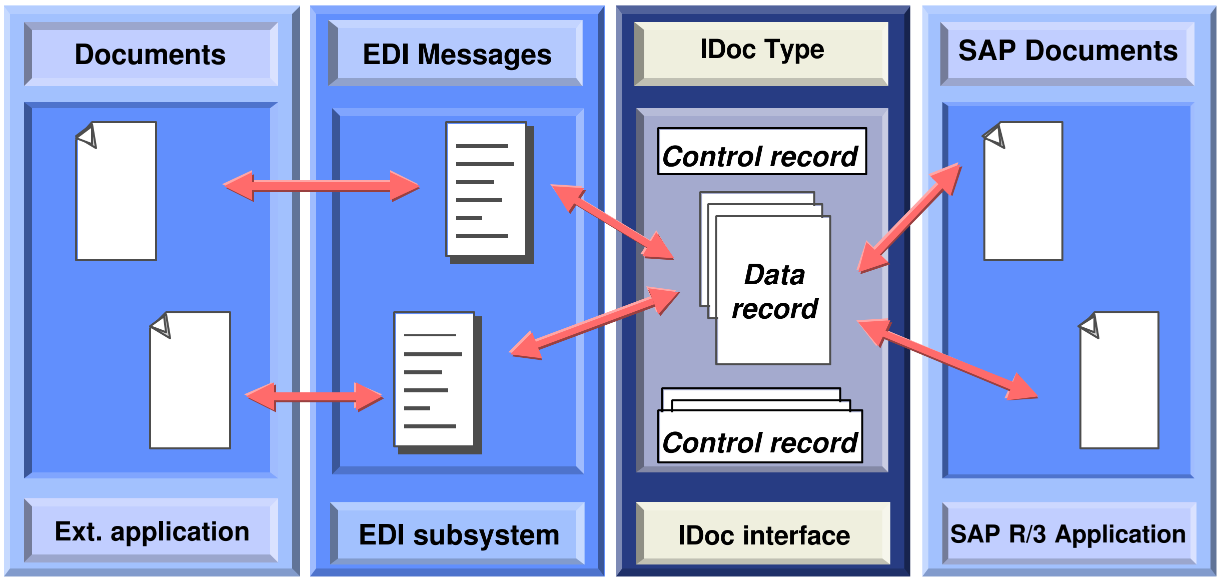
- PP
- Inventory management
- Internal sales, distribution, and billing
- Local purchasing
- PM

- Accounting
- Central Controlling
- Information systems:
  - Inventory
  - Purchasing
  - Sales
- Central Purchasing
- Reference system for master data and tax data

- Sales, distribution, and billing
- Sales of trade goods
- Inventory Management
- Local Controlling

- ALE is the exchange of messages controlled from a business point of view
  - By synchronous and asynchronous communication
  - Using BAPI interfaces and IDoc Data containers
  - With consistent data storage
    - The applications do not use a central database
    - They are supplied by regional, closed databases with their own data basis
  - Between loosely linked R/3 applications.
- An ALE business process is an integrated, cross-system business process. ALE business processes run between:
  - Several SAP R/3 Systems
    For example, Logistics, Finances and HR on separate systems
  - R/3 and non-SAP R/3 systems
    For example, in SAP R/3, external warehouse control system
  - R/3 and Web/Desktop systems
- Possible linkages of ALE business processes:
  - Loose linkage (asynchronous, message-based)
  - Tight linkage (synchronous)
  - A combination of both

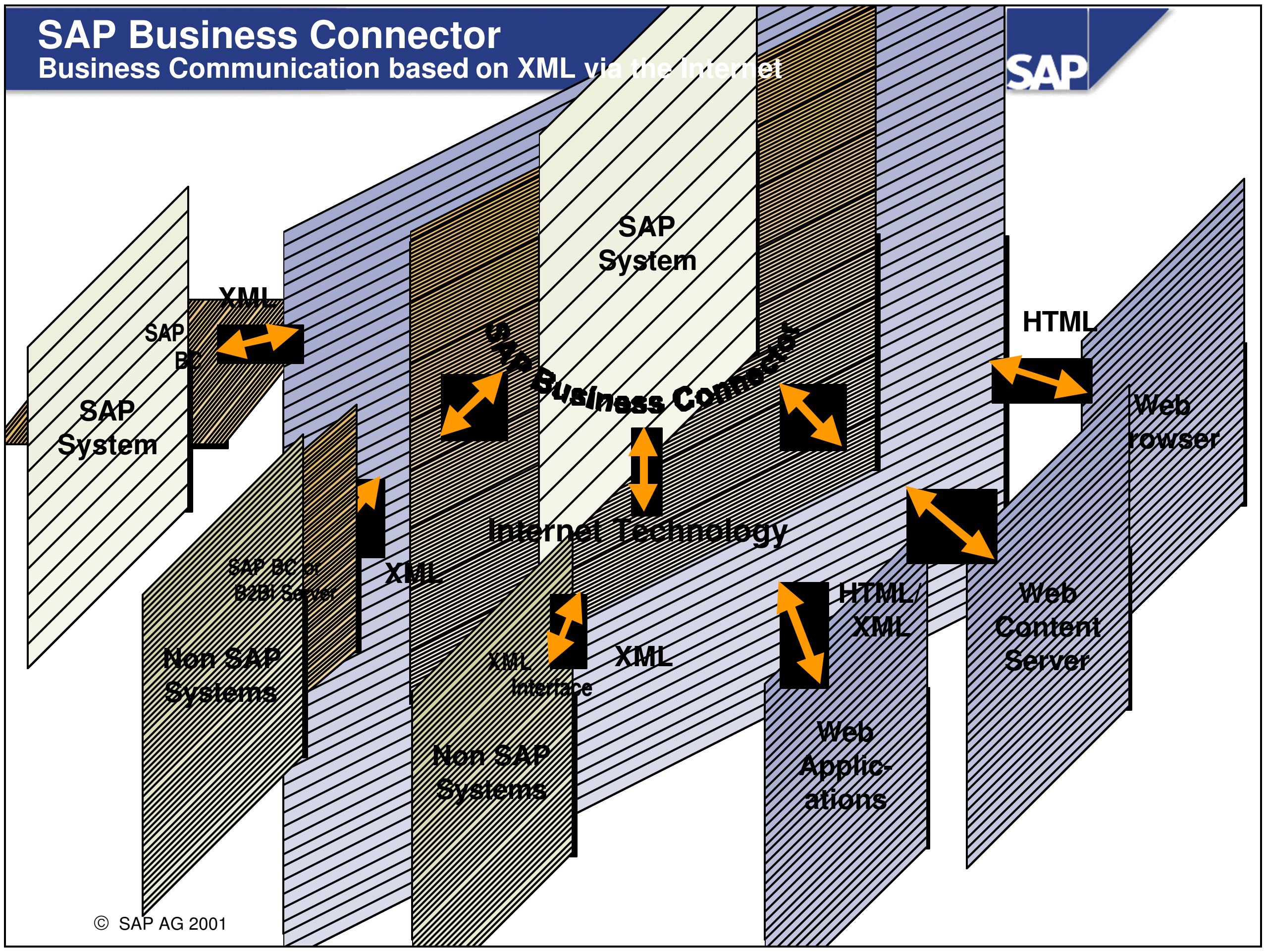
EDI Architecture



- Electronic Data Interchange (EDI) describes the electronic exchange of structured business data between applications.
- EDI architecture consists of:
  - EDI-enabled applications; they allow business transactions to be processed automatically
  - The IDoc interface; this was designed as an open interface and consists of the intermediate documents (IDocs) and the corresponding function modules, which create the interface to the application.
  - The EDI subsystem; this converts the intermediate documents into EDI messages and back. SAP does not supply this element of the EDI architecture.
- The main component of the IDoc interface is the IDoc type. An IDoc is an SAP standard that specifies the structure and format of the data to be transferred electronically. It was developed to support the EDIFACT and ANSI X12 standards. IDocs are identified uniquely using a control record. The application data records form the core. The status records log the status of an IDoc as it is passed from the application to the trading partner and back.
- XML will most likely be used more and more for transferring business information between enterprises.

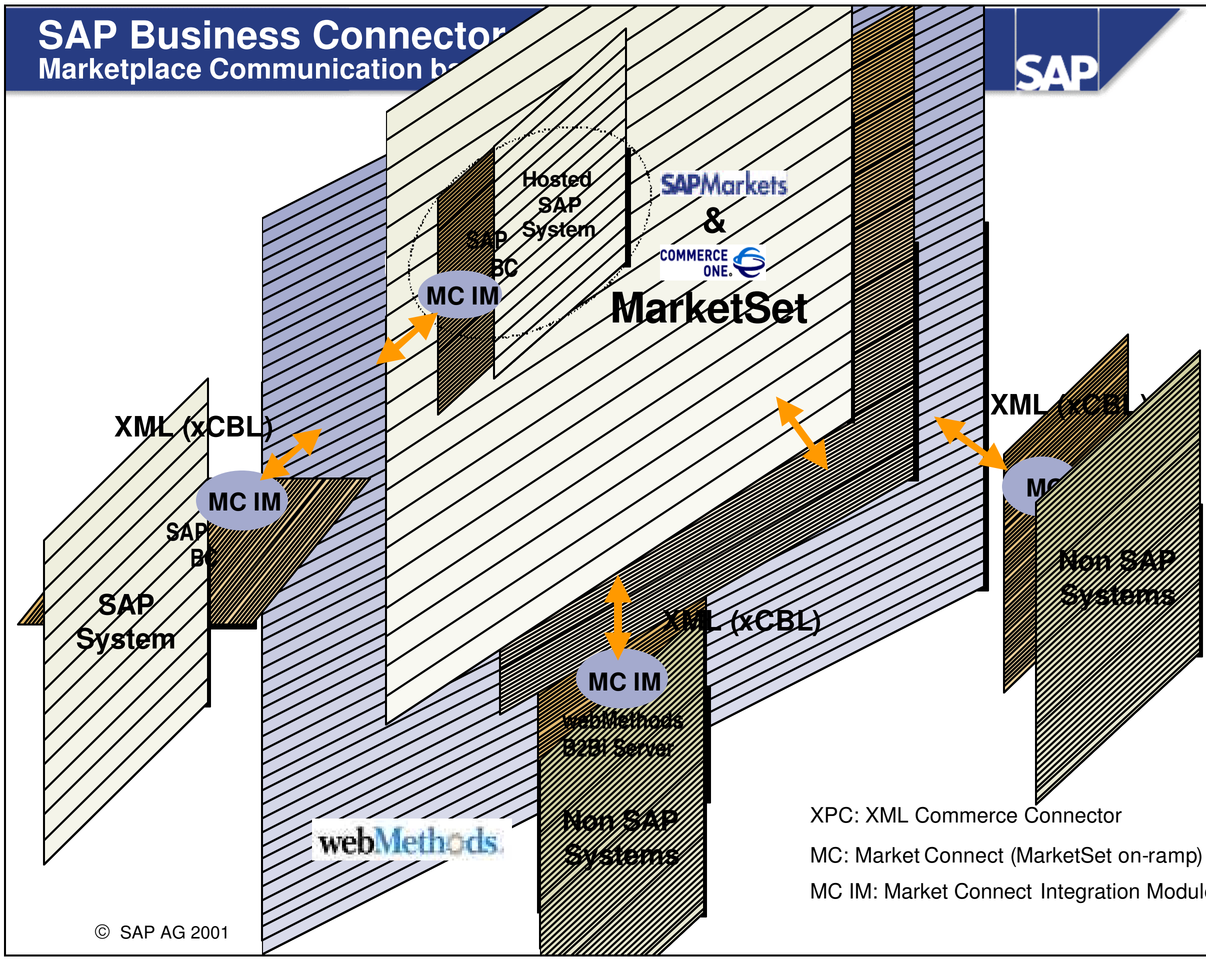
SAP Business Connector

Business Communication based on XML via the Internet



SAP Business Connector

Marketplace Communication b



XPC: XML Commerce Connector

MC: Market Connect (MarketSet on-ramp)

MC IM: Market Connect Integration Module

SAP BC - Benefits for SAP Customers

- Seamless Integration of different IT architectures
  - Reduce cycle times and eliminate supply-chain inefficiencies
  - Automation of Business Processes
  - Build tighter linkages to customers, partners, suppliers

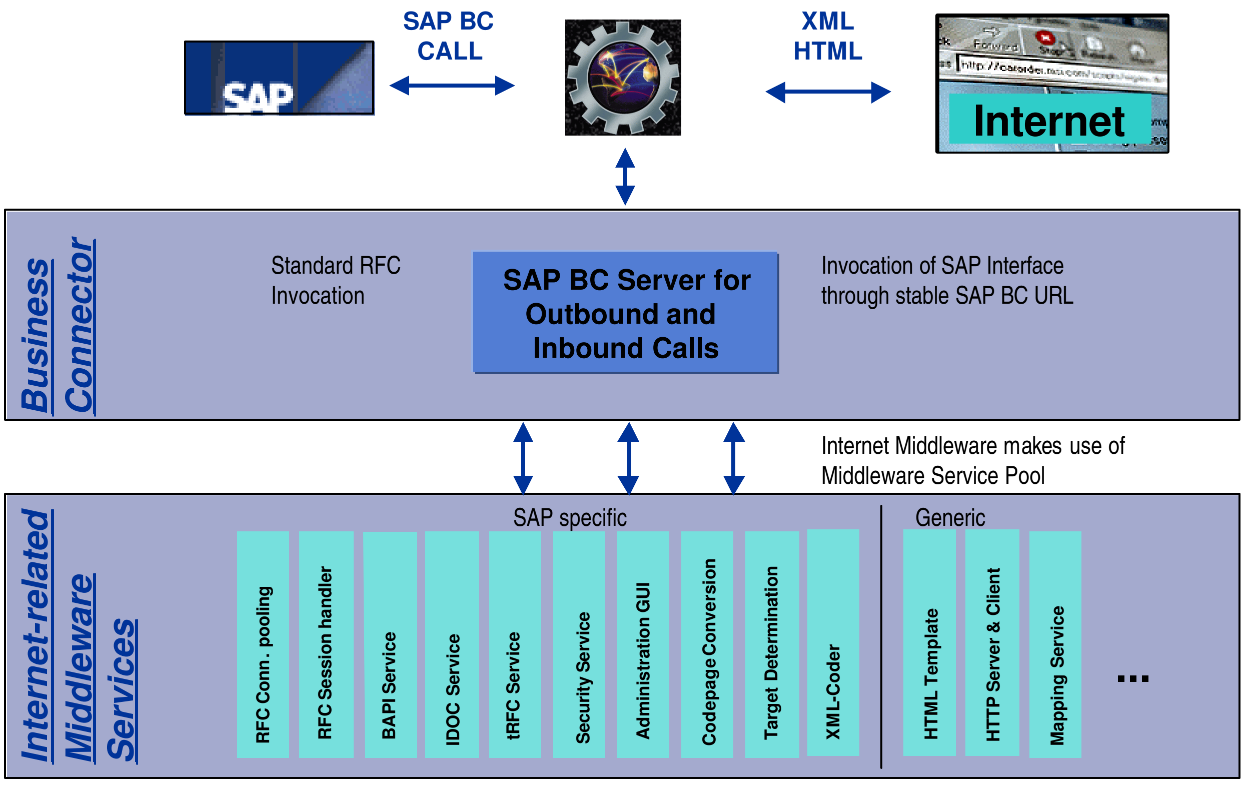
- Extend traditional EDI infrastructure
  - Add realtime integration
  - Reach non-EDI partners
  - Leverage partner investment in web-based systems

SAP BC - Benefits for SAP Customers

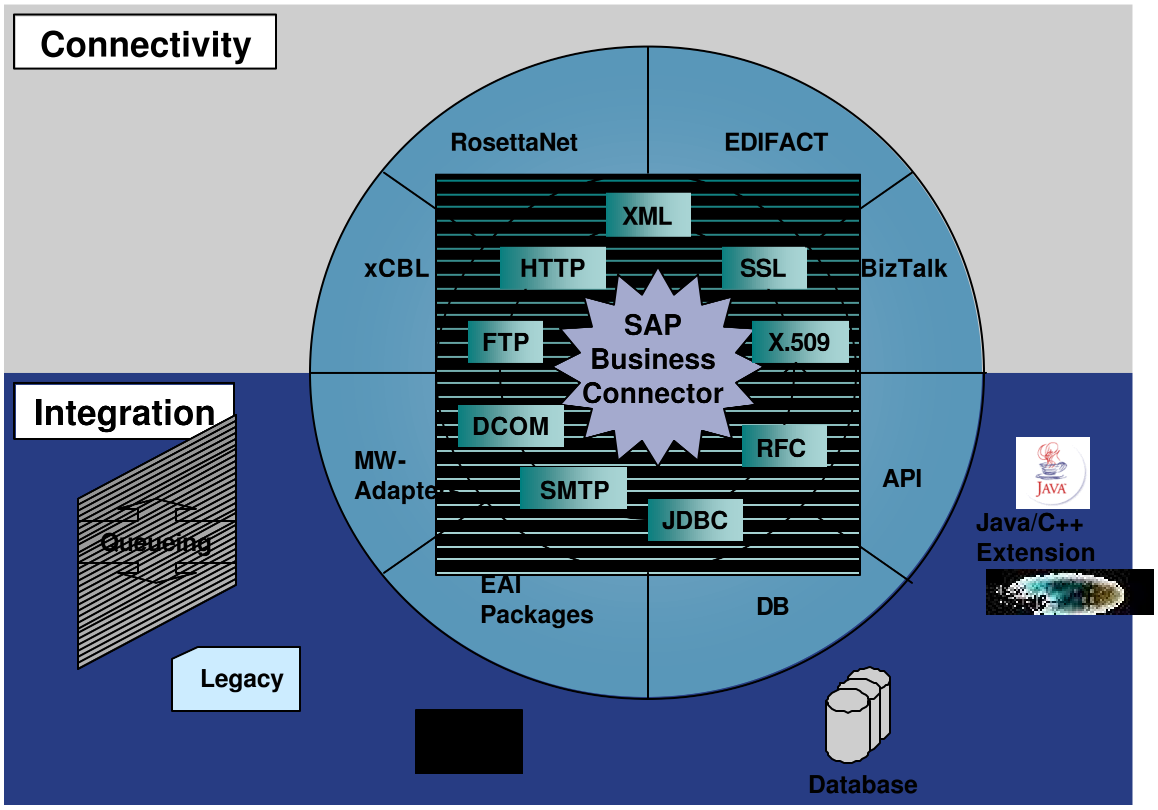
- XML-enabling of SAP solutions
  - Easy to understand, deploy and customize
  - Highly flexible; lends itself to real-time data exchanges between business partners
  - Rapid and Flexible adaption to evolving business document standards
  - The platform (Web) is already in place

- Tight integration with traditional SAP Components and mySAP.com
  - Integrated with SAP's BAPI and ALE technologies
  - Supports R/3 releases from 3.1G upward
  - Integration within SAP Marketplaces

SAP BC - A Pool of services



SAP BC - Architecture



SAP BC - Key Features (1)

The SAP Business Connector Server supports...

- Bi-directional communication
  - Inbound calls to SAP System and outbound calls to external servers
  - BC Gateway Manager handles routing
- Synchronous and asynchronous communication
  - Transparent processing of RFC or tRFC calls
- Simultaneous protocols
  - HTTP(S), FTP, SMTP
  - Guaranteed delivery handles transient failures

SAP BC - Key Features (2)

The SAP Business Connector Server provides...

- Generic IDoc service
  - Binary IDoc segments are resolved into single fields
  - ALE message status tracking (extends ALE monitoring)
- Generic BAPI service
  - Full support of XML interface descriptions published in SAP Interface Repository
  - Unifies different interface types (BAPIs, RFCs, IDocs) on XML level (Standard Biztalk XML envelopes , unified errorhandling)
  - Built-In BAPI-browser

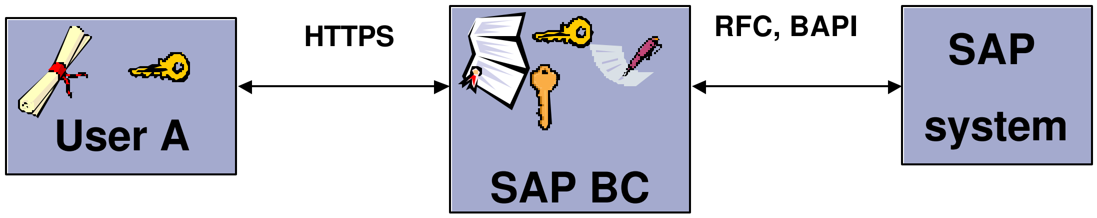
SAP BC - Key Features (3)

The SAP Business Connector Server supports...

- Platform independence
  - Business Connector is implemented in Java
    - runs on any platform which has a Virtual Machine (VM)
  - SAP Interface Module requires multi-threaded shared RFC library (librfc) which is platform-dependent
    - currently only available for MS Windows NT 4.0 and Windows 2000
    - Linux available with Release 3.5.1
    - Other operating systems available on demand
- Built in Web-Server
  - HTTP server implemented in SAP BC

SAP BC - Key Features (4)

The SAP Business Connector Server provides secure communication...



- Security Issue
  - SSL for strong encryption of internet communication
  - X.509 (client and server) certificates
  - Digitale Signature
  - Only functionality exported for SAP BC can be accessed
  - All services are secured by access-control lists

SAP BC - Key Features (5)

The SAP Business Connector Server provides high available communication...

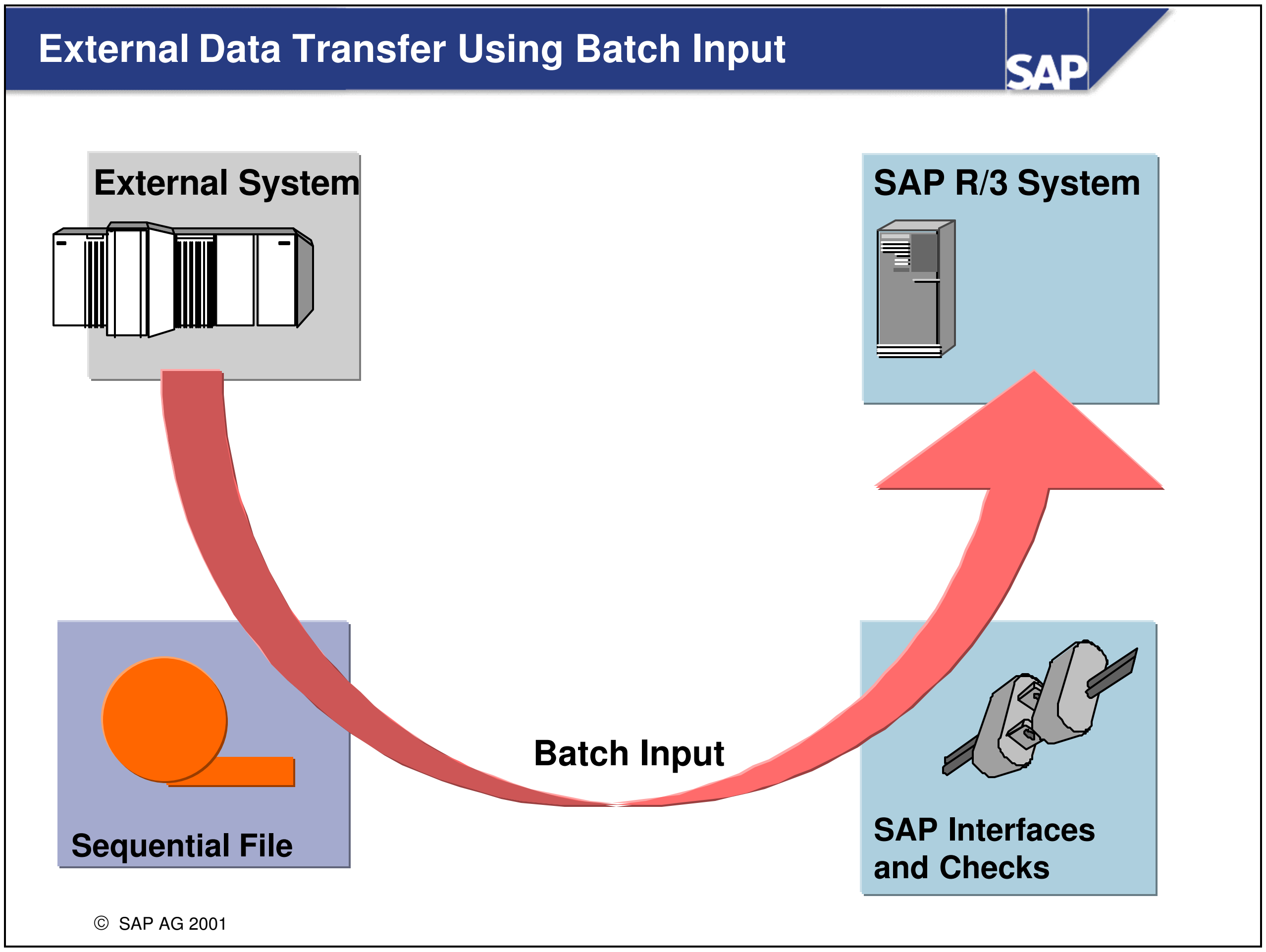
- **Clustering (with Release 3.5.1)**
  - **Load Balancing – workload can be spreaded over a number of servers**
  - **Failover Support – requests are automatically redirected to another server**
  - **Scalability – increase capacity is easily possible**

SAP BC - Key Features (6)

SAP BC Developer supports...

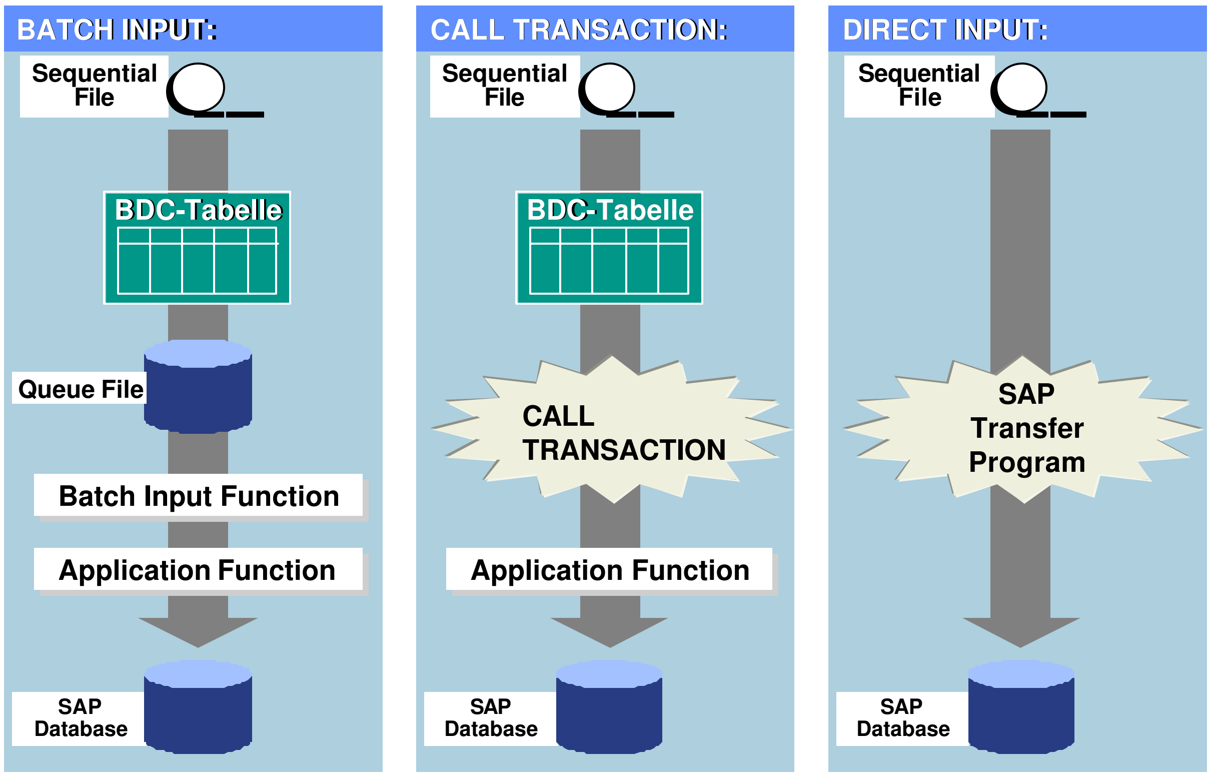
- Graphically designed flow
- Graphical mapping for name-, structural-, value transformations
- Web-Automation
- Flexible architecture
  - Plug-In services can either be built with
    - Flow Language or
    - Programming languages such as Java, C/C++ or Visual Basic
- Client and server code generation

External Data Transfer Using Batch Input



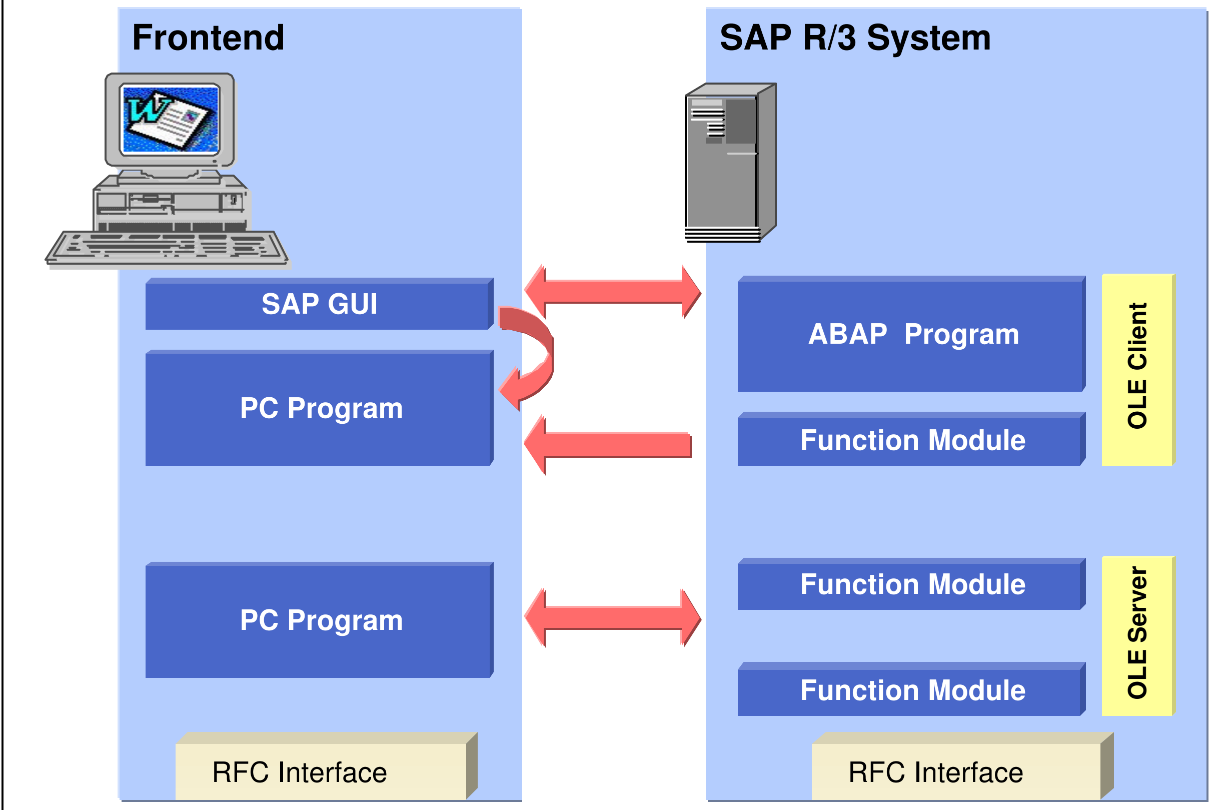
- When you transfer data from one SAP R/3 System to another, or from an external system to an SAP R/3 System, you must ensure the integrity of the data that is transferred. This means the external data must be subjected to the same checks and controls as data that was entered manually online.
- As the online checks in transactions are very comprehensive and are partly used across applications, it is very difficult to program them yourself.
- The best and easiest way is to use the online checks in SAP transactions, which also includes using SAP transactions for data transfer.
- The methods used to transfer external data are known as "batch input" methods.
- For many areas, SAP provides standardized external data transfer methods. These methods use **batch input, call transaction** and **direct input** programming methods. The SAP standard direct input methods are controlled using the Data Transfer Workbench (transaction SXDA)**.** If no SAP standard transfer method is available, you can program transfers using batch input or call transaction. For more information, see the next slide.

Transferring External Data Using Batch Input



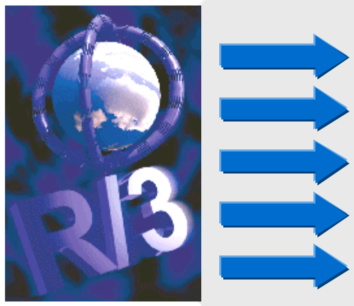
- With **batch input**, transfer data is buffered as a Batch Data Communication (BDC) table in a queue file (batch input session). In the next step, the system processes this session, that is, the data is transferred to the application transaction within the batch input environment and therefore entered in the database.
- SAP R/3 Systems enable you to record a transaction in dialog mode. You can use this recording to generate a batch input session and a transfer program.
- Alternatively to batch input, you can also call the transaction directly (**Call Transaction**) after a data record is taken from the sequential file and placed in the BDC table. The read and write process for the queue file does not apply here.
- The batch input and call transaction methods access application transactions within the component system. The data is therefore subjected to the same consistency checks before it is transferred as data in the dialog mode.
- **Direct input** programs execute the consistency checks and update the data in the database, without having to access the application transactions. Direct input methods are used in a few SAP standard transfer methods such as transferring bills of material and documents). These processes are only programmed by SAP developers.

Office Integration Using OLE



- Object linking and embedding (OLE) is an object-oriented method for program-to-program communication. You can connect **office applications** that support OLE2 automation (for example, Word and Excel) to mySAP.com components systems. In this way, users can use some SAP R/3 functions within their usual desktop environment.
- The office programs' OLE functions are specified in the type information. This information contains a description of the methods, attributes and parameters. Type information can be language-independent.
- When using OLE, the component system can play two separate roles:
  - If the component system is acting as an **OLE client**, then the user calls the desktop program from the ABAP application. OLE commands are transferred from the ABAP code as remote function calls (RFCs) through the SAP GUI to the PC. The SAP GUI maps RFC calls to OLE commands for the PC application.
  - If the component system is acting as an **OLE server**, functions from that component system can be called from the desktop application. In the component system, function calls and BAPIs are triggered by business objects. After the data is processed successfully, the business object sends the data back to the desktop program.
- There are now pushbuttons in many standard transactions, which can be used to download data to a Frontend program using OLE.

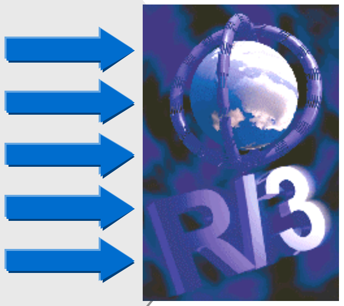
Other Interface Technologies (Outbound)



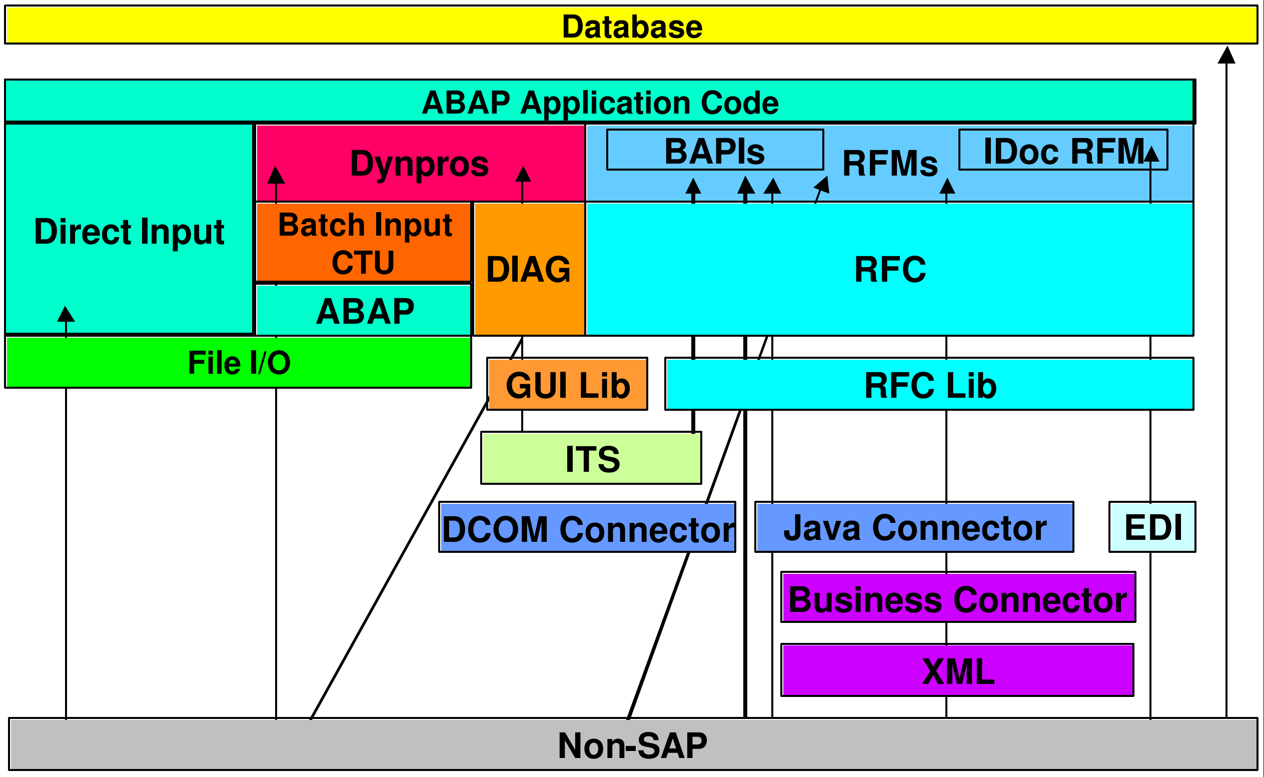
- SAP technologies
  - ABAP export program
  - SAP Automation GUI
  - SAP-GUI download
- Non-SAP technologies
  - Direct database access (SQL, ODBC)
  - Database replication

Other Interface Technologies (Inbound)

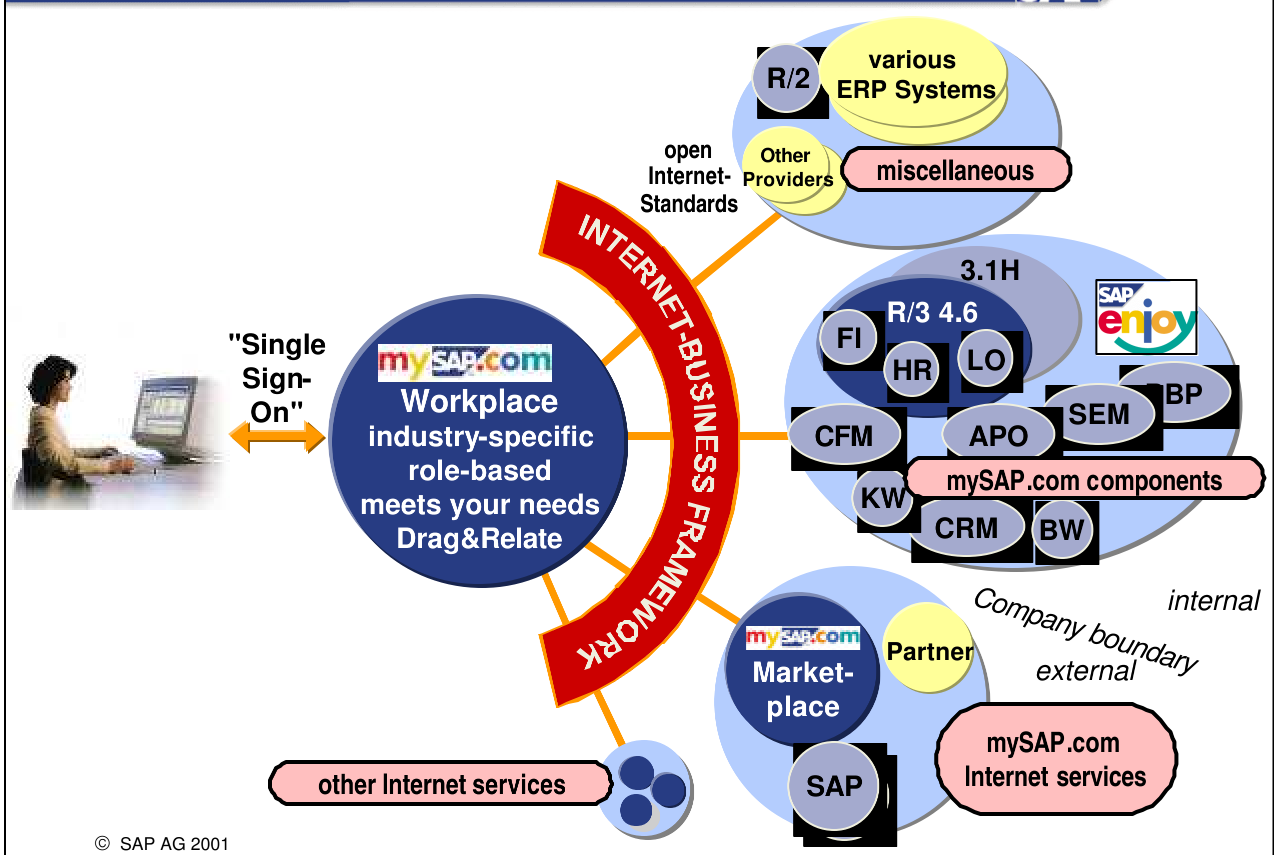
- SAP technologies
  - SAP Automation GUI
- Non-SAP technologies
  - (((Database access)))



A Simplified (!) Depiction of Some SAP Interfaces

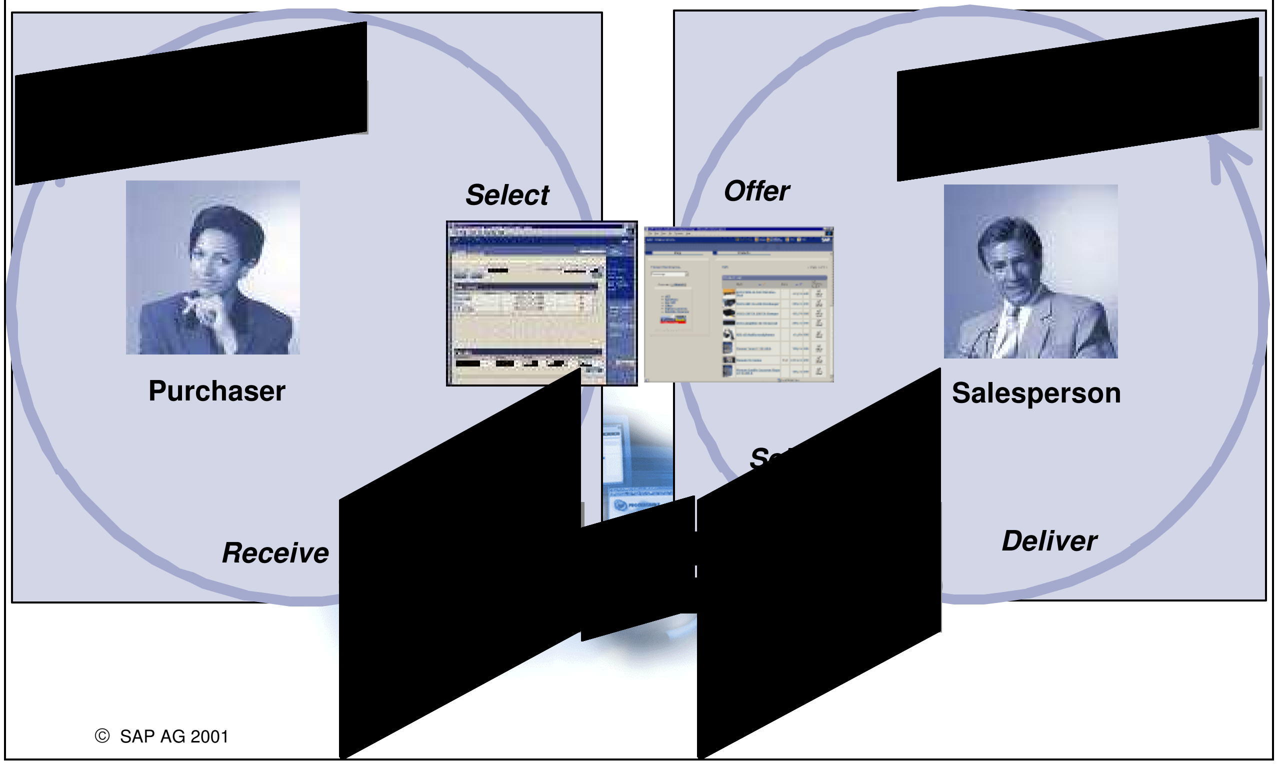


mySAP.com Workplace



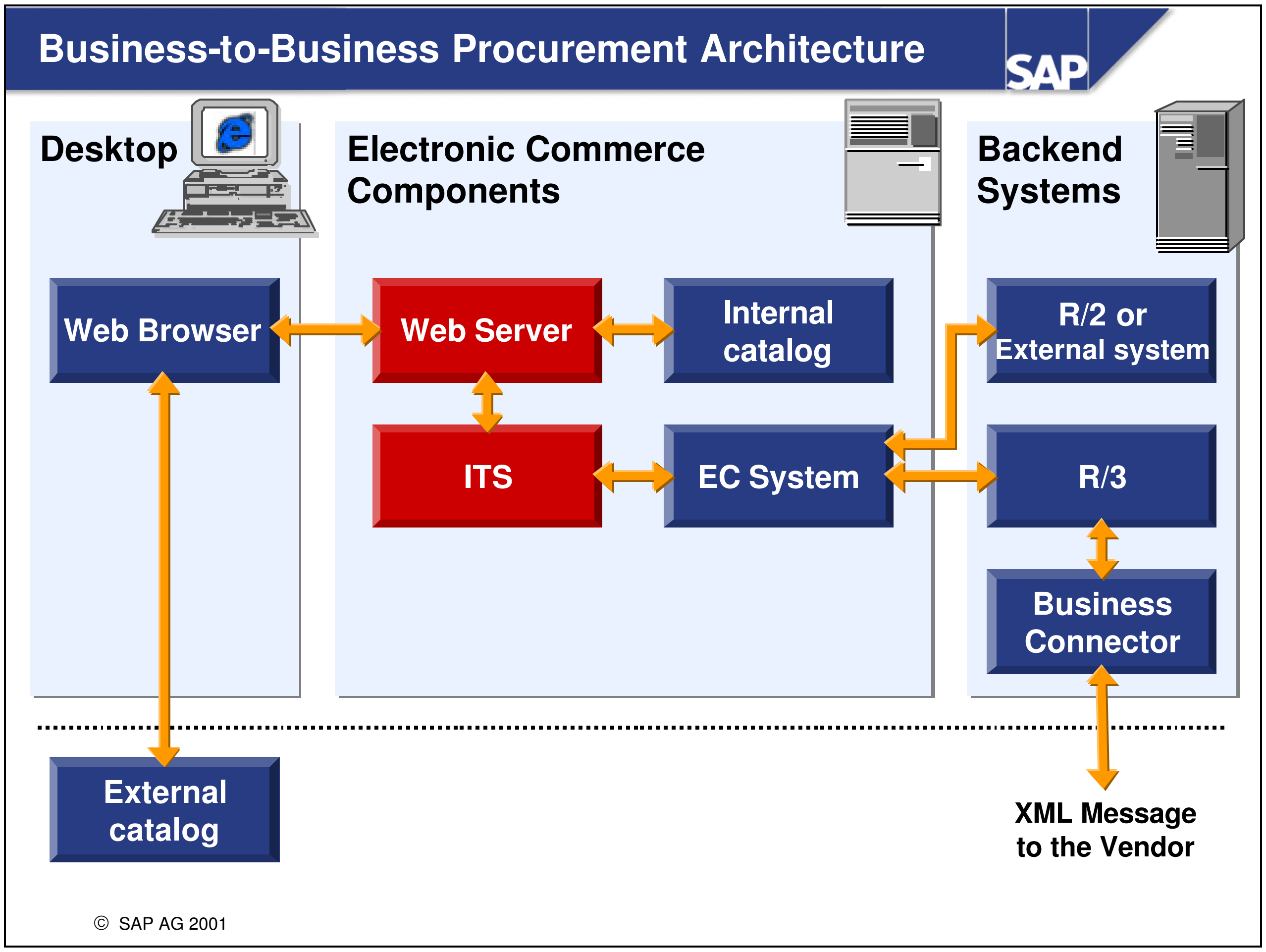
- The mySAP.com **Workplace** is the user interface and entry point for activity-related content.
- Access to the Workplace and all of the applications accessed through it is possible by logging on only once (**Single Sign-On**).
- The user can individually configure the Workplace. The Workplace is also delivered in a number of different industry- and role- specific versions according to the requirements of the customer. By adding frequently used links and transactions, the Workplace can be adjusted to the needs of the individual users. The *Drag&Relate* function is implemented throughout.
- The Workplace allows access to a range of mySAP.com and external components. These are called over the Internet or an internal network (intranet) using Single Sign-On.
- All component systems are accessed from the Workplace screen. SAP R/3 Systems are Internet- and intranet-enabled as of SAP R/3 Release 3.1. Important mySAP.com components are Knowledge Warehouse (KW), Business Information Warehouse (BW), Customer Relationship Management (CRM), Advanced Planner & Optimizer (APO), Strategic Enterprise Management (SEM), Corporate Finance Management (CFM), and Business-to-Business Procurement (BBP).

Business-to-Business Procurement (BBP)



- **SAP Business-to-Business Procurement** (SAP BBP) is an e-Commerce component for business purchases. It creates a seamless connection between purchasers of production-independent goods and services and suppliers, and can be directly linked to the SAP ERP System. SAP Business-to-Business Procurement supports all procurement processes, from the requirement coverage request, through the goods receipt to the invoice payment.
- SAP BBP offers:
  - Involvement of employees in procurement using user-friendly Self Service functions
  - Higher productivity and quicker purchasing management using new business scenarios
  - Access to product information through the integration of almost any online catalog
  - Simple, direct links to e-Commerce Marketplaces
  - Quick communication with suppliers
  - Flexible implementation scenarios
  - Support for several backend systems
  - Efficient processing of invoices and payments
  - Secure data transfer

Business-to-Business Procurement Architecture

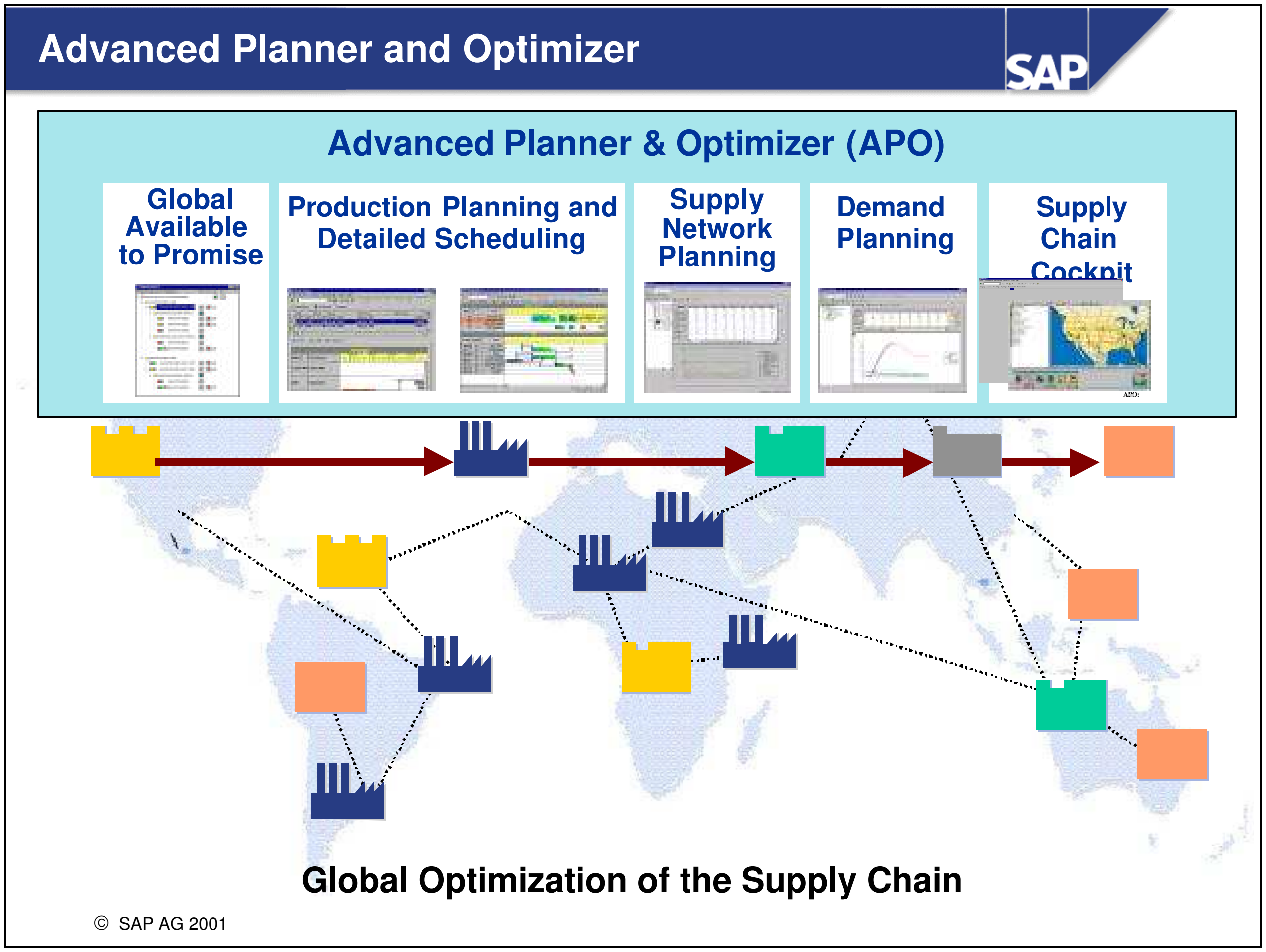


- The employee only requires a **Web browser** that communicates with a **Web server**.
- The employee chooses the products to be ordered from an electronic catalog. As well as having access to the (empty) catalog provided by SAP, you can include **external catalog systems,** as long as they fulfill the specifications for the Business-to-Business Procurement Open Catalog (BBP-OCI) Interface.
- The **Internet Transaction Server** (ITS) connects the Web server and the **Electronic Commerce** (EC) system. The management of the shopping basket, the authorization process, and later also the receipt confirmation all run in the EC system.
- SAP BBP 2.0 supports various scenarios for connecting to **Backend Systems** :
  - *classic*: All materials management runs in one or more external systems (SAP R/3, SAP R/2 or non-SAP ERP Software).
  - *linked*: The requirement coverage request for non-production-related goods and services are processed directly by the EC system and the orders and all follow-on documents are created. You continue to order production-related materials and services using the materials management in your ERP system.
  - *stand alone*: You have no materials management in your ERP system and use the materials management functions in SAP BBP for all non-production-related procurement.
- The order is sent to the vendor in XML format using the **Business Connector**.

Business Information Warehouse (BW)

- The **SAP Business Information Warehouse (BW)** is the mySAP.com business component with which data can be extracted from operational business applications (Online Transaction Processing Systems (OLTP) and analyzed. As well as OLTP Systems such as R/3 and SAP BBP, external data sources such as databases or online services can be connected.
- SAP Business Information Warehouse supports Online Analytical Processing (OLAP) and is especially suited to processing large volumes of operational and historical data.
- SAP BW contains all the necessary metadata for fast-moving business processes. These include InfoSources, InfoObjects, InfoCubes and Standard reports, transfer structures for all supported releases, and communications structures and update rules for each InfoCube. These elements are part of a “ready-to-go” strategy that supports automatic data transfer and immediate analysis after installation of the system and designation of the source system.
- SAP BW requests the application data at regular intervals from the assigned source systems (pull mechanism). For this purpose, the backend systems include extractors that collect data and deliver it to the SAP Business Information Warehouse.

Advanced Planner and Optimizer


SAP
Advanced Planner & Optimizer (APO)
Global Available to Promise
Production Planning and Detailed Scheduling
Supply Network Planning
Demand Planning
Supply Chain Cockpit

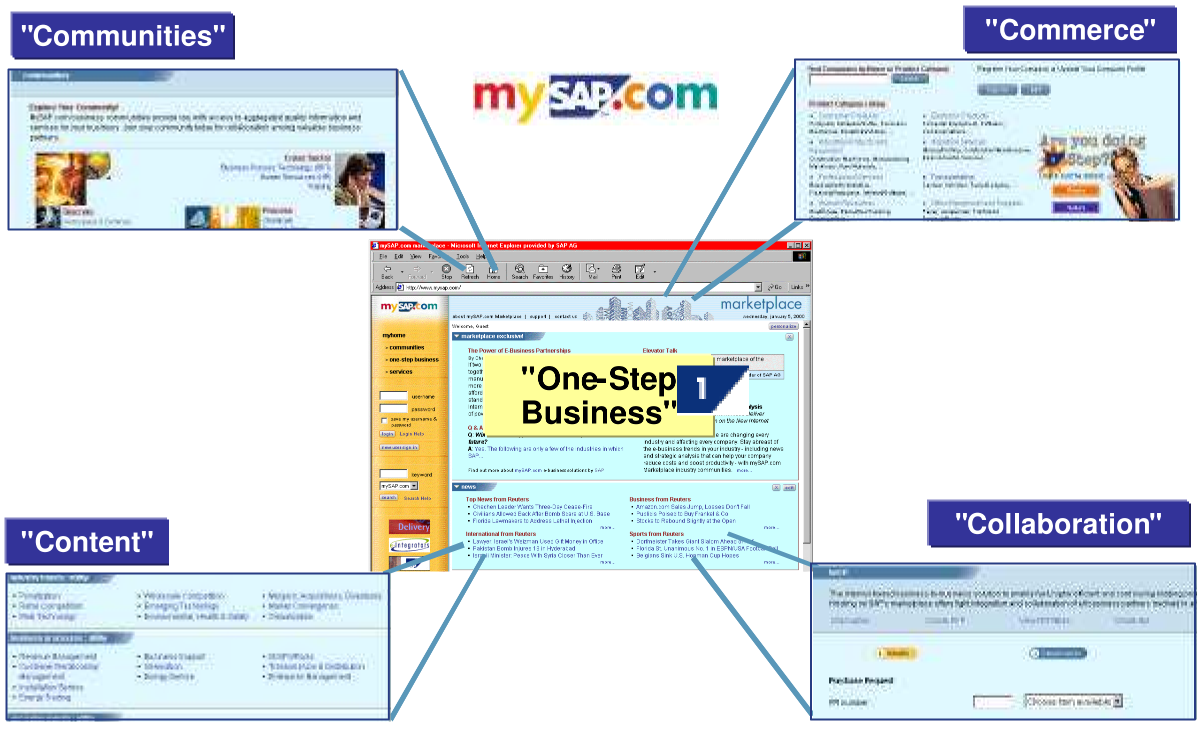
Global Optimization of the Supply Chain

- The **Advanced Planner and Optimizer (APO)** provides a complete solution for the supply chain. Every task in the process, from planning to optimization and checks, can be performed using APO.
  - APO is a dedicated solution with its own release cycle. Every component can be implemented as an stand-alone system.
  - **Global Available-To-Promise (Global ATP)** – Coordinates supply and demand worldwide, and allows authoritative delivery confirmations to customers through the use of real-time checks and sophisticated simulated processes.
  - **Production Planning and Detailed Scheduling** – Optimizes the use of resources and creates production plans that help to make production cycles shorter and to react to changes in demand quickly.
  - **Supply Network Planning** – Coordinates procurement, production, and transport processes with demand and allows clearing and optimization of your entire supply chain.
  - **Demand Planning** – Identifies and analyzes demand patterns and fluctuations and creates exact, dynamic demand predictions.
  - **Supply Chain Cockpit** – Models, monitors, and controls your supply chain using a specially-designed graphic interface. The Supply Chain Cockpit provides the user with a detailed view of all activities and applications.

Customer Relationship Management

- **SAP Customer Relationship Management (CRM)** provides solutions that allow companies to effectively manage their customer relationships throughout their life cycle. CRM assists the company in understanding and anticipating the demands of its customers and its potential customers. SAP CRM can be broadly divided into four main business scenarios:
  - **Mobile Sales Scenario**: Allows field staff complete access to all of the information necessary for their work. The data is stored in a database on the laptop and is regularly reconciled with the CRM System.
  - **Contact Center Scenario**: Forms one of the key components of SAP CRM, along with the Customer Interaction Center (CIC). CIC allows the processing of incoming and outgoin g telephone calls using the SAPphone Computer Telephony Integration (CTI) interface , the processing of incoming and outgoing e-mails, and activity management (recording in the Contact Center all contacts that are made). The CIC can be used for Service (Service Interaction Center SIC), in Distribution, or in Marketing (Telesales or Telemarketing).
  - **Internet Sales Scenario**: Makes electronic business activities possible using the World Wide Web: between companies (B2B and B2R; for more information, see SAP BBP below) and between companies and end customers (B2C).
  - **Business-to-Business Procurement** (SAP BBP): Is the SAP solution for business activities between companies using the World Wide Web (for more information, see the previous slides on SAP BBP).

mySAP.com Marketplace



http://marketplace.mysap.com

- The mySAP.com Marketplace is divided into four main areas:
  - In the **Communities** area, companies can create a **virtual industry community**. Members of these communities exchange information in discussion forums. Cross-site cooperation is also encouraged by the Communities.
  - Constantly updated content is also available on the Marketplace. Included in this is information about industry conferences, industry news, and information about trends. The user can additionally display, for example, current market prices for a stock portfolio that he or she has selected.
  - The **Commerce** area has its own industry directory. Companies that have entered the Marketplace offer their products and services. Using integrated exchange of business documents, One-Step Business is possible. You can subscribe to various Online Services at the mySAP.com Homepage.
  - **Internet-based Collaboration** between companies is also made easier using the mySAP.com Marketplace.
- The Marketplace makes **One-Step Business** possible**:** Goods and services can be bought and sold in a single operation. Two processing systems in the background of each Marketplace ensure correct financial transactions.

SAP Overview: Unit Summary

- The most important technologies for interfacing SAP and non-SAP components are
  - Remote Function Call
  - ALE/IDoc
  - Business Connector

Unit: SAP Overview
Topic: SAP Architecture

At the conclusion of these exercises you will be able to:

Understand the SAP architecture.

Understand BAPI commit handling.

1-1 Answer the following questions:

1-1-1 Which layer in the SAP architecture do BAPI-enabled programs talk to?

1-1-2 Can there be multiple update tasks in one SAP system?

1-1-3 If a BAPI contains a COMMIT WORK statement, it commits

a) only its own changes

b) all changes in this session since the last commit or rollback.

Unit: SAP Overview
Topic: SAP Architecture

1-1 Answer the following questions:

1-1-1 Which layer in the R/3 architecture do BAPI-enabled programs talk to?

The application layer

1-1-2 Can there be multiple update tasks in one R/3 system?

Yes

1-1-3 If a BAPI contains a COMMIT WORK statement, it commits

b) all changes in this session since the last commit or rollback.

SAP Remote Function Call (RFC): Contents

- Overview
- Middleware Support
- RFC Flavors

SAP Remote Function Call (RFC): Unit Objectives

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

- **Understand the main features of RFC.**
- **List the available middleware choices.**
- **Understand the different RFC flavors.**

RFC Overview

- Standard, bi-directional interface technology between SAP and other SAP or non-SAP components
- Different flavors
  - Synchronous ((s)RFC)
  - Transactional (tRFC)
  - Queued (qRFC)

SAP Prerequisites for Inbound Calls

- The Function Module to be called must be flagged as RFC-enabled
  - RFM = RFC-Enabled Function Module
  - A BAPI is an RFM which has been defined as an object method in the SAP Business Object Repository (BOR).

Non-SAP Client/Server Prerequisites

- RFC Library, available for
  - Win32 (librfc32.dll)
  - Linux
  - Other SAP-supported Unix flavors
  - OS/390
  - OS/400
  - OS/2
  - Macintosh (client only)

RFC Main Features

- Encapsulates
  - Network communication
  - Data conversion
  - Code page handling
- Thread-safe
  - Win32 since 3.1H
  - Unix since 4.6D
  - But no concurrent calls on the same connection!
- Easy to use in ABAP
- External (non-SAP) clients/servers can be built based on the (low level) RFC Library directly or using higher level middleware.

Middleware Solutions for RFC

- Java
  - SAP Java Connector (JCO)
  - Obsolete: SAP JRFC
- Windows
  - SAP DCOM Connector (inbound)
  - SAP COM4ABAP (outbound)
    - Since 4.6D
  - SAP ActiveX Controls (inbound)

  All the above are covered in CA925
- Various non-SAP products

Programming Languages

- Java
- C
- Any language that supports ActiveX
  - Visual Basic
  - Visual C++
  - many others

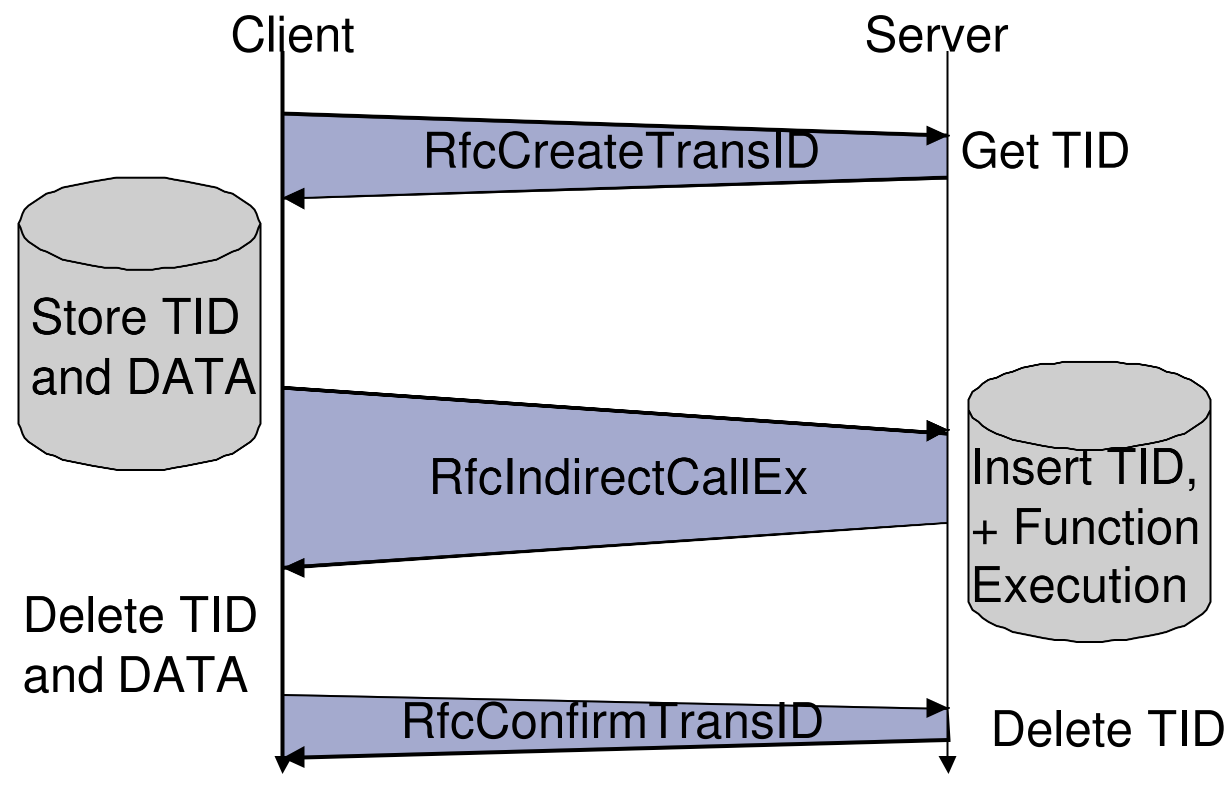
Synchronous RFC

- Client waits until the processing is complete.
- sRFC supports input AND output parameters.
- Error handling easy
- Problem: How to avoid duplicate invocations

Transactional RFC (tRFC)

- Guaranteed to execute only once.
- External tRFC is always synchronous.
- ABAP tRFC is asynchronous.
- Only supports input parameters.
- External program must provide
  - Transaction ID management
  - Scheduling for retry after errors
- A particular order for execution cannot be guaranteed.
- Error handling must happen in target system.
- SAP JCO supports tRFC.

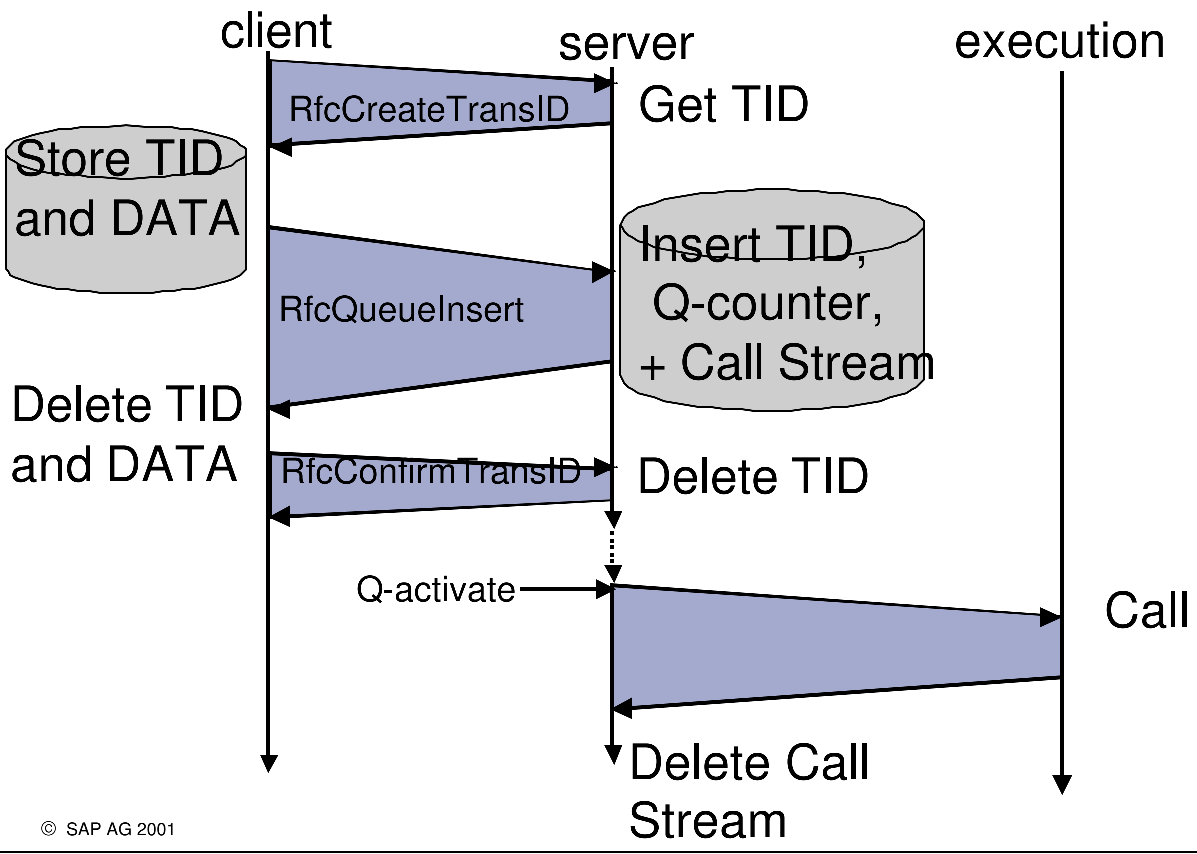
tRFC Execution Steps



Queued RFC (qRFC)

- Guaranteed to execute only once and in the specified sequence
- Execution is asynchronous
- Not integrated in the ABAP runtime. qRFC is controlled by calls to system function module
- In ABAP, you can choose between
  - queuing on the caller system: outbound queue (APO)
  - queuing on the called system: inbound queue (CRM)
- The RFC Library offers support for queues in the SAP system:
  - inbound queues for external clients,
  - outbound queues for external servers.
  - SAP JCO supports outbound qRFC.
- However, external programs can also perform their own external queuing (MQLink, MSMQ,BizTalk....)
- External program must ensure
  - Transaction ID management
  - Scheduling for renewed submission after an error
- “Someone” must activate the queue
- SAP System may schedule a retry after errors for the execution

Review of RFC Quality of Service: qRFC



SAP Remote Function Call (RFC): Unit Summary

- RFC is the main interface technology between SAP and other SAP or non-SAP components.
- Three flavors of RFC exist
  - sRFC
  - tRFC
  - qRFC
- External clients/servers can be developed using different programming languages (like Java, C(++), and Visual Basic), based on the RFC Library directly, or using RFC-enabled middleware (like SAP JCO).

RFMs, BAPIs, and Tools: Contents

- RFC-enabled Function Modules (RFMs)
- BAPI Overview
- Business Object Repository (BOR)
- BAPI Object Model

RFMs, BAPIs, and Tools: Unit Objectives

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

- Understand RFMs.
- Describe the attributes of BAPIs.
- Describe the Business Object Repository.
- Use the appropriate SAP tools for RFMs and BAPIs.

RFM Object Model

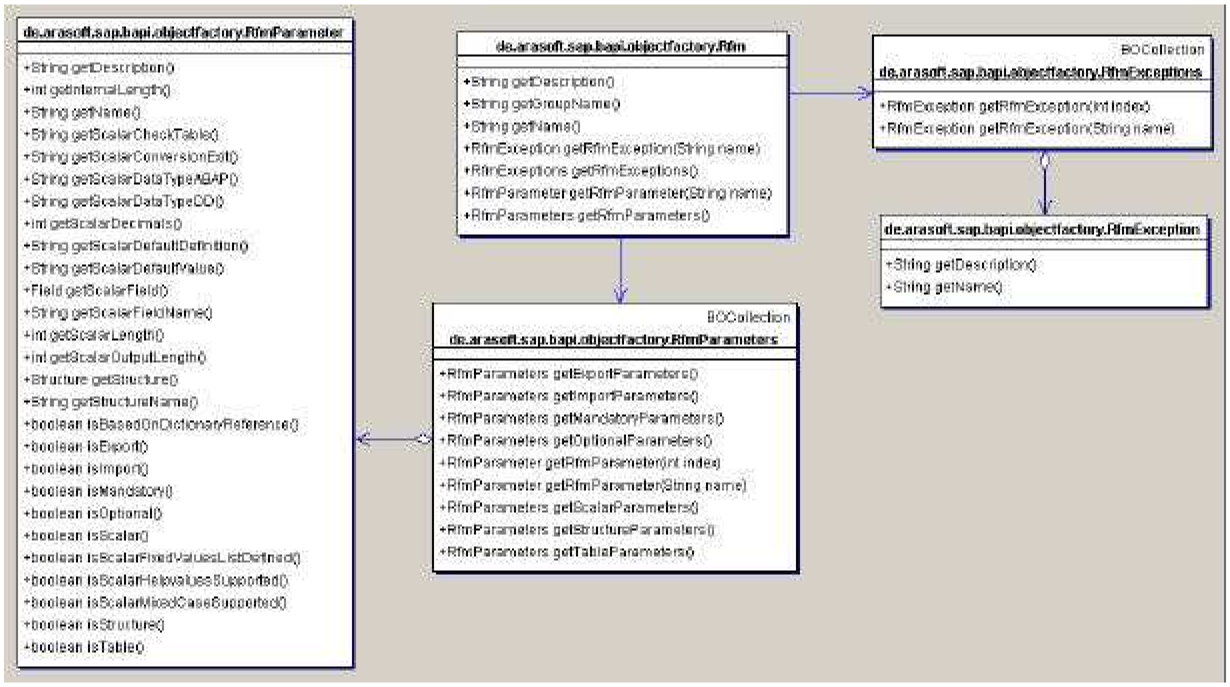
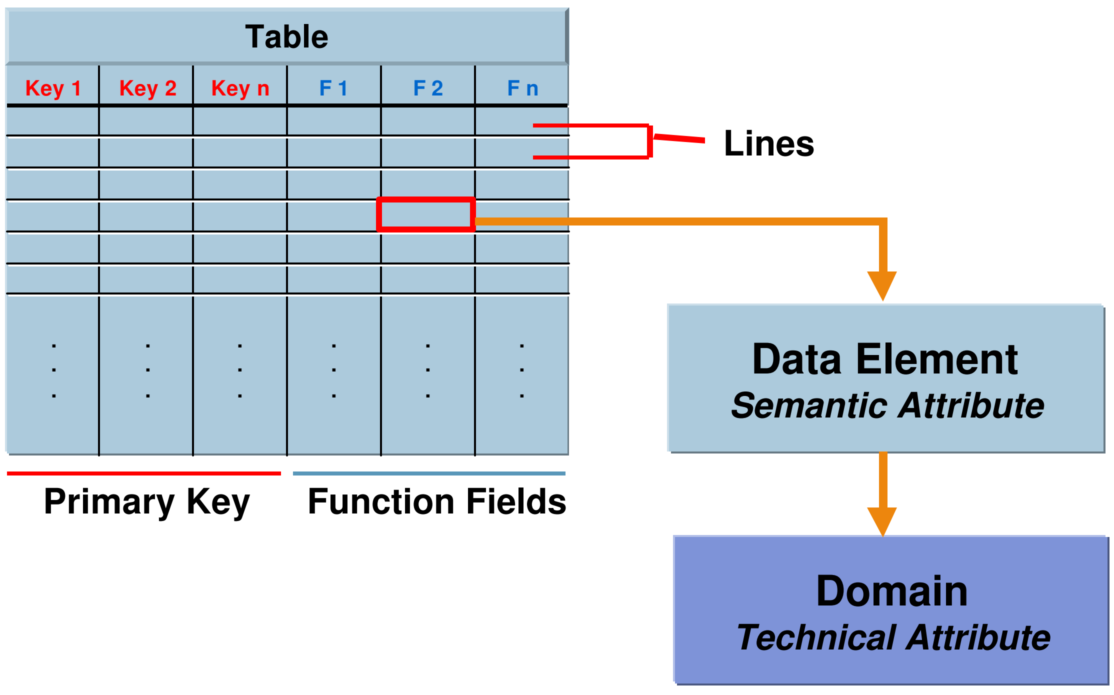
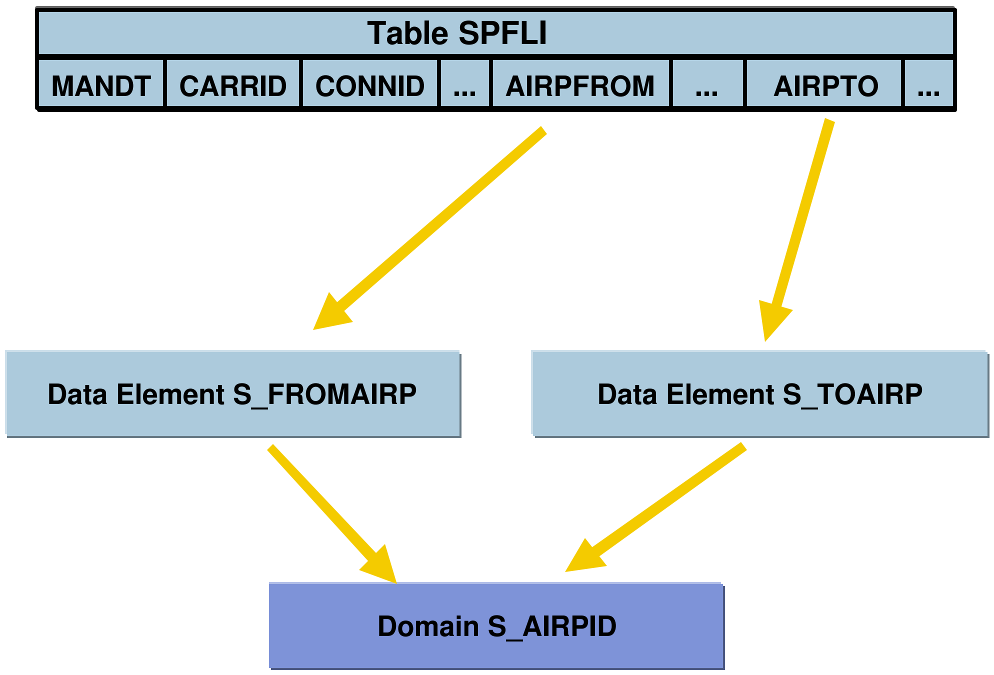

de.arasoft.sap.bapi.objectfactory.RfmParameter
+String getDescription()
+int getInternalLength()
+String getName()
+String getScalarCheckTable()
+String getScalarConversionExit()
+String getScalarDataTypeABAP()
+String getScalarDataTypeDD()
+int getScalarDecimals()
+String getScalarDefaultDefinition()
+String getScalarDefaultValue()
+Field getScalarField()
+String getScalarFieldName()
+int getScalarLength()
+int getScalarOutputLength()
+Structure getStructure()
+String getStructureName()
+boolean isBasedOnDictionaryReference()
+boolean isExport()
+boolean isImport()
+boolean isMandatory()
+boolean isOptional()
+boolean isScalar()
+boolean isScalarFixedValuesListDefined()
+boolean isScalarHelpvaluesSupported()
+boolean isScalarMixedCaseSupported()
+boolean isStructure()
+boolean isTable()
de.arasoft.sap.bapi.objectfactory.Rfm
+String getDescription()
+String getGroupName()
+String getName()
+RfmException getRfmException(String name)
+RfmExceptions getRfmExceptions()
+RfmParameter getRfmParameter(String name)
+RfmParameters getRfmParameters()
BOCollection
de.arasoft.sap.bapi.objectfactory.RfmExceptions
+RfmException getRfmException(int index)
+RfmException getRfmException(String name)
de.arasoft.sap.bapi.objectfactory.RfmException
+String getDescription()
+String getName()
BOCollection
de.arasoft.sap.bapi.objectfactory.RfmParameters
+RfmParameters getExportParameters()
+RfmParameters getImportParameters()
+RfmParameters getMandatoryParameters()
+RfmParameters getOptionalParameters()
+RfmParameter getRfmParameter(int index)
+RfmParameter getRfmParameter(String name)
+RfmParameters getScalarParameters()
+RfmParameters getStructureParameters()
+RfmParameters getTableParameters()

Table Definition

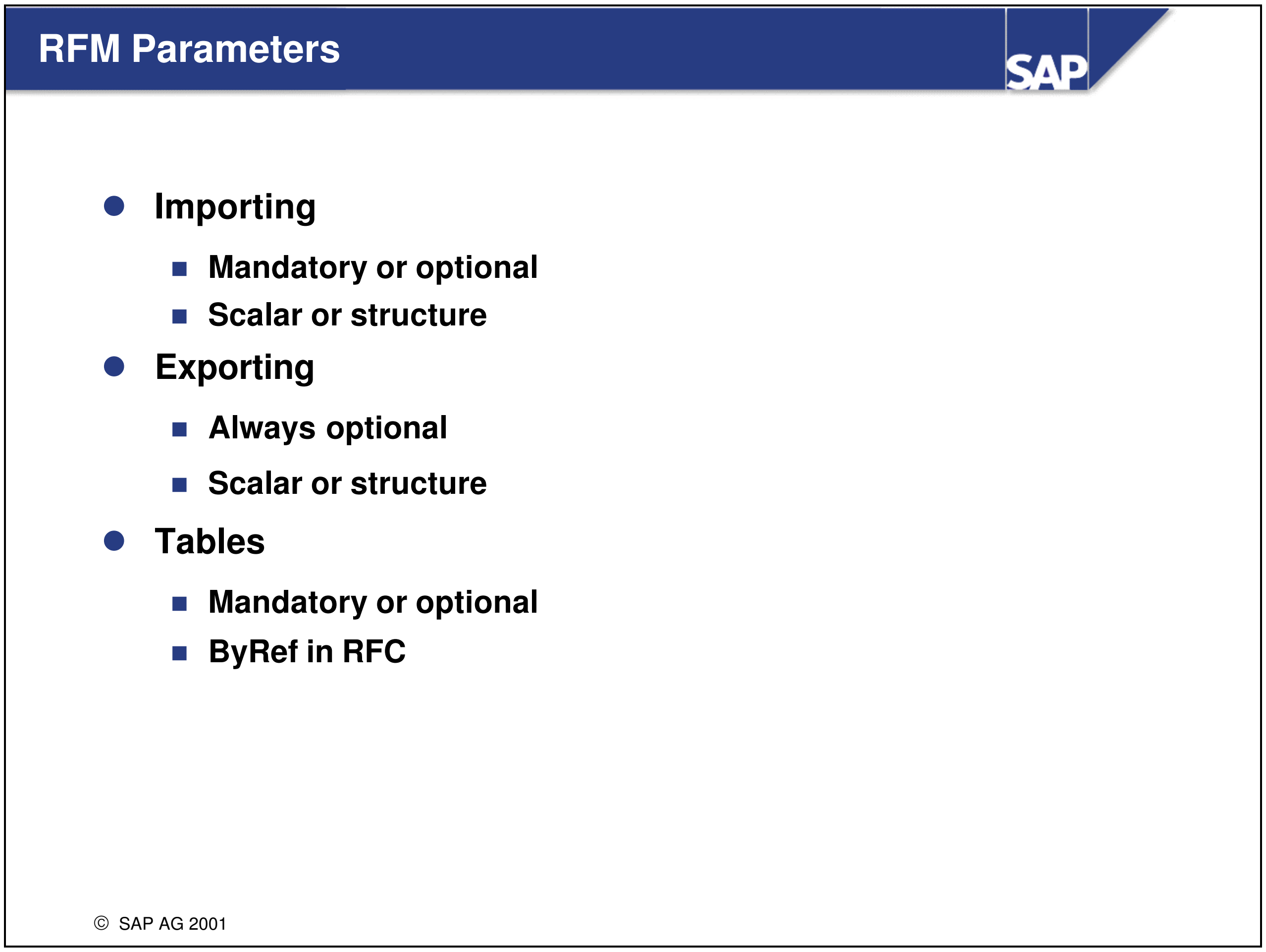


- A table is a two-dimensional matrix. It has a name and attributes, such as the table type. Every SAP table has a primary key consisting of a combination of columns that uniquely identify a table record. This means a table cannot contain two records with the same primary key.
- A field has a name and attributes. For example, it can be a primary key field. A field is dependent on a table and is, therefore, not an independent object and can only be maintained within the table. A table field is defined using domains and data elements.
- A domain is used to define the technical attributes of a table field and contains technical attributes of the table field, such as field length, field type, output attributes, and any value restrictions based on default values.
- A data element is the semantic definition of a table field and can contain a short description of the table field, for example, displayed when the user chooses *F1*. As of SAP R/3 Release 4.6, the technical attributes of a field can be defined in the data element, without having to use a domain.
- Tables, data elements, and domains are maintained centrally in the ABAP Dictionary. When a table is activated, it is created in the database using the same name as in the ABAP Dictionary.

Two-Level Domain Concept



- The two-level domain concept allows you to define and maintain technical field attributes at domain level. A domain can pass its field attributes to any number of fields, whereby only the domain, but not the individual fields, must be explicitly changed when the defined field attributes are modified. This also ensures that if domains are identical, field values can be compared without requiring conversion.
- The data element describes the semantic attributes of a field in the context of the table. These are attributes that are only important at that location and not globally (like the technical attributes).
- The example here shows the table SPFLI from the ABAP flight booking model. The table is a central store of flights, such as Lufthansa flight XY from Frankfurt to New York. The table contains the departure airport (*AIRPFROM*) and arrival airport (*AIRPTO*) fields. From a business viewpoint, the departure airport and the arrival airport are two separate entities, which is why two data elements, S_FROMAIRP and S_TOAIRP, are defined. As both columns contain names of airports, both data elements are related to the same domain, S_AIRPID, which is defined as being of type CHAR and length 3.



RFM Parameters

- Importing
  - Mandatory or optional
  - Scalar or structure
- Exporting
  - Always optional
  - Scalar or structure
- Tables
  - Mandatory or optional
  - ByRef in RFC

- Export parameters are values/messages coming from SAP to the calling application.
- Import parameters are values/messages supplied by the calling application to SAP.

RFM-related Tools

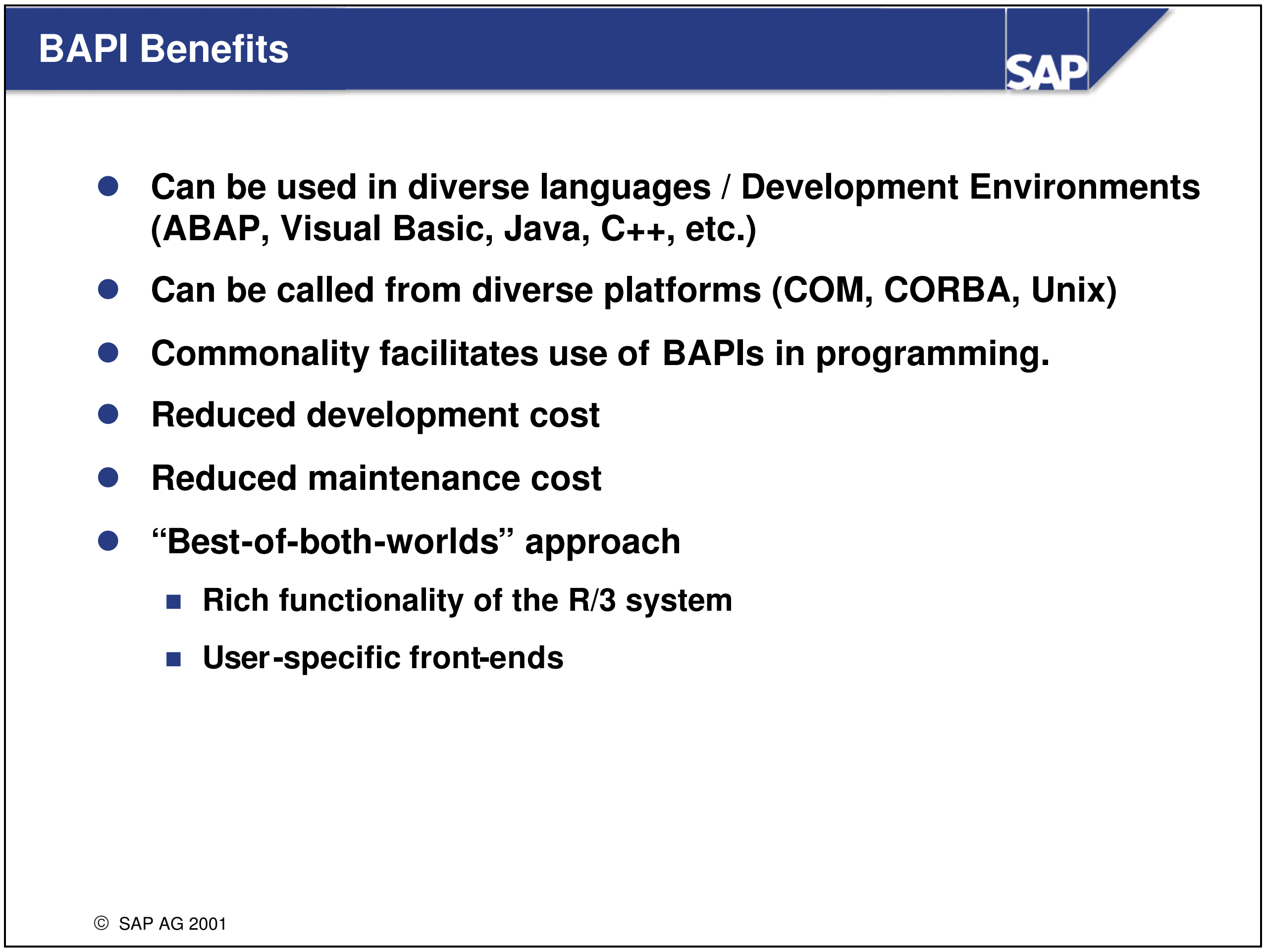
- Function Builder (SE37)
  - Display
  - Change
  - Create
  - Test
- Data Dictionary (SE11)

BAPI Overview (1)

- Official, supported, upward-compatible interfaces to the R/3 system.
- Access to SAP data and functionality.
- Support for read, create, delete, update.
- Support for transactions that transcend one BAPI call.
- Object-oriented (methods of Business Objects)
- Defined in the Business Object Repository (BOR).
- Currently implemented as RFC-enabled function modules.

BAPI Overview (2)

- Defined independently of middleware.
- Used by SAP and external applications.
- Based on development standards (e. g. naming conventions).
- Developed by
  - SAP
  - Partners
  - Customers



BAPI Benefits

- Can be used in diverse languages / Development Environments (ABAP, Visual Basic, Java, C++, etc.)
- Can be called from diverse platforms (COM, CORBA, Unix)
- Commonality facilitates use of BAPIs in programming.
- Reduced development cost
- Reduced maintenance cost
- "Best-of-both-worlds" approach
  - Rich functionality of the R/3 system
  - User-specific front-ends

- SAP BAPIs provide developers with stable, reliable interfaces for accessing the functionality offered by SAP business objects.
- BAPIs can reduce development cost by providing built in functionality through predefined methods of the SAP business objects. These methods can reduce programming time in development projects.
- Maintenance costs can be reduced because the BAPIs are stable and documented structures representing SAP's investment in the R/3 business object architecture.

Open BAPI Network

- BAPI section
  - Documentation
  - Mailing list
- COM section
  - Downloads
- Java section
  - Downloads
- http://sapnet.sap.com/bapi
  http://www.sap.com/bapi

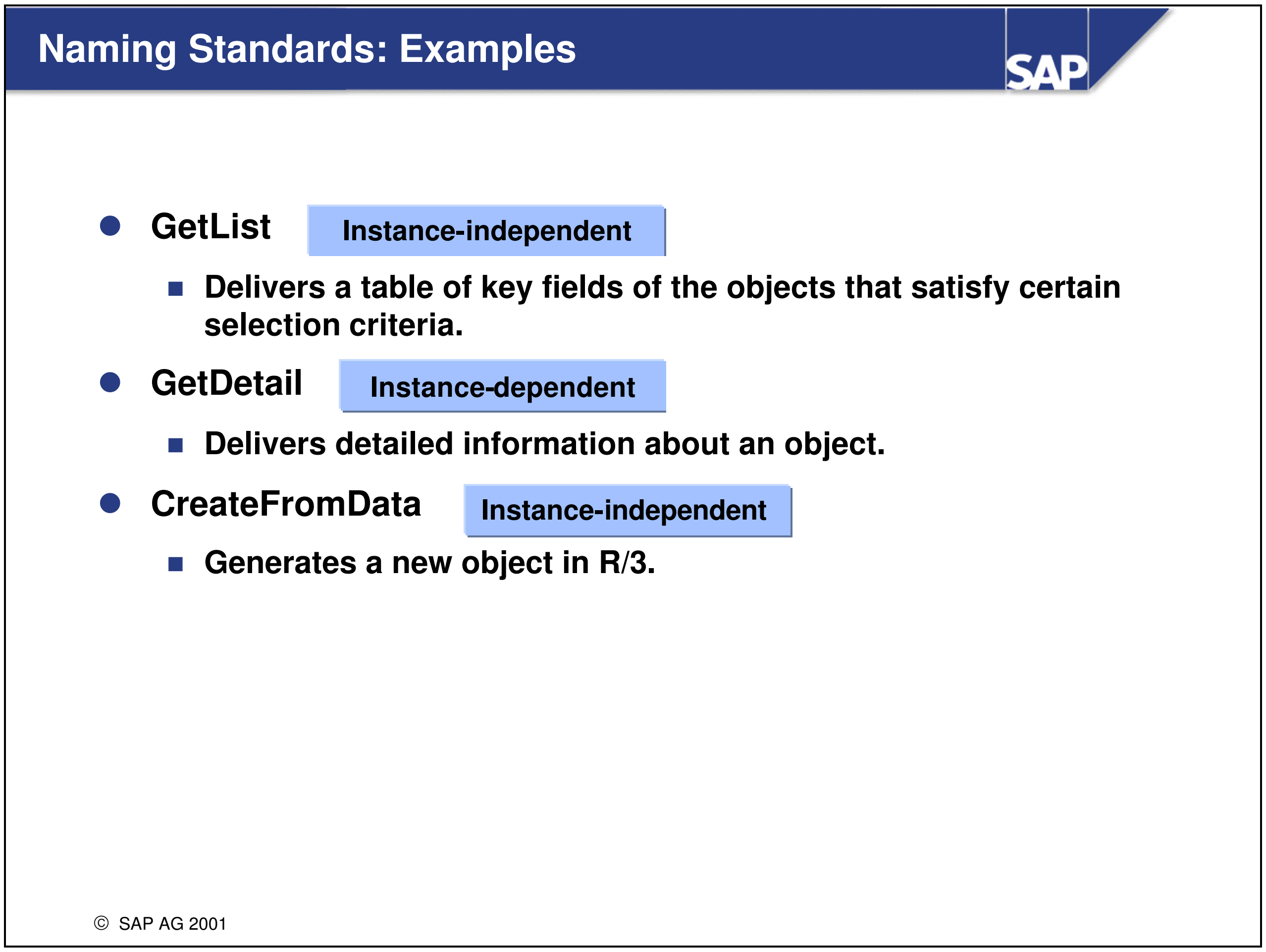
- Access via sapnet requires an Online Service System (OSS) user ID

Interface Repository

- Lists interfaces for many SAP components and releases
  - IDoc message types
  - RFMs
  - Business object types and BAPIs
- http://ifr.sap.com

Software Partner Program

- Ready-to-use solutions
- Reduced implementation time & costs
- Wide choice of integrated solutions
- http://sapnet.sap.com/spp
  http://www.sap.com/spp



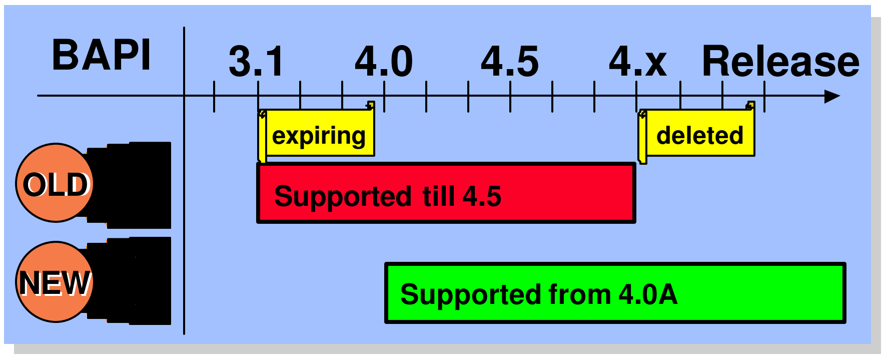
Naming Standards: Examples

- **GetList** — **Instance-independent**
  - **Delivers a table of key fields of the objects that satisfy certain selection criteria.**
- **GetDetail** — **Instance-dependent**
  - **Delivers detailed information about an object.**
- **CreateFromData** — **Instance-independent**
  - **Generates a new object in R/3.**

- Instance-independent BAPIs can be used without explicitly creating an instance
- of the business object.
- Instance-dependent BAPIs require explicit creation of the business object instance.

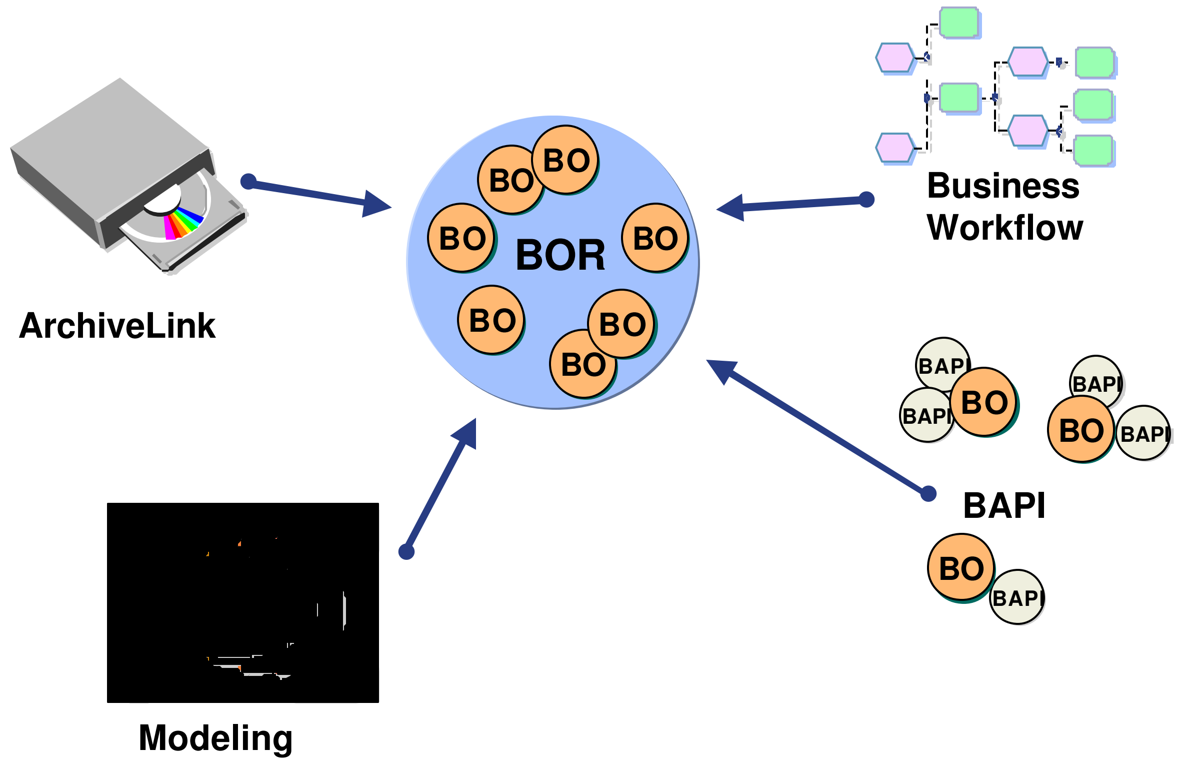
BAPI Life Cycle: Change Management

- **BAPIs can only be changed compatibly (i.e, enhanced with optional parameters)**
- **In case of unavoidable incompatible changes:**
  - **An additional BAPI has to be created, e.g. ExchangeRate.Create1**
  - **The existing BAPI is flagged as expiring and continues to be supported for at least two subsequent SAP functional releases (roughly two years).**



- On the basis of SAP Release strategy and the strict rules for enhancing existing BAPIs, application developers can rely on the stability of BAPI interfaces.
- Each enhancement to a released BAPI is carried out ensuring that the syntax and contents are downwardly compatible. This means that applications programmed with BAPIs in a particular R/3 Release are not affected by changes to this BAPI in later releases.
- If changes that are not downwards compatible need to be made to an existing BAPI, a new interface, that is, a new additional BAPI is created. This BAPI has the same name of the existing BAPI to be replaced but with a number added at the end, for example, CustomerCode.GetDetail1.
- The BAPI to be replaced is marked as expiring (obsolete) in the Release in which the successor BAPI is implemented, but will continue to be supported in the next two functional Releases. BAPIs that are set to “obsolete” in an R/3 Release are listed in Note 0107644, “Collective Note for Obsolete BAPIs as of Release 4.5A” in the Online Software Service (OSS)

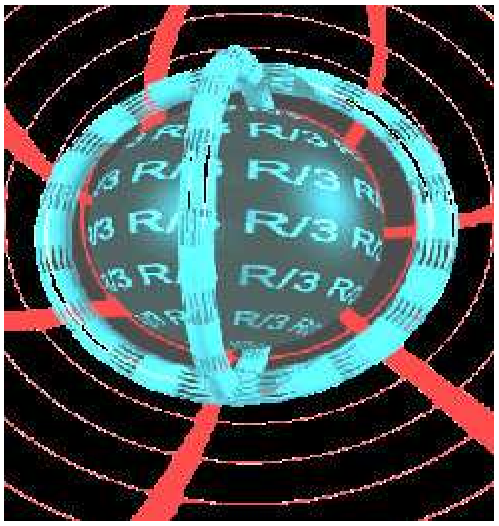
Business Object Repository (BOR)



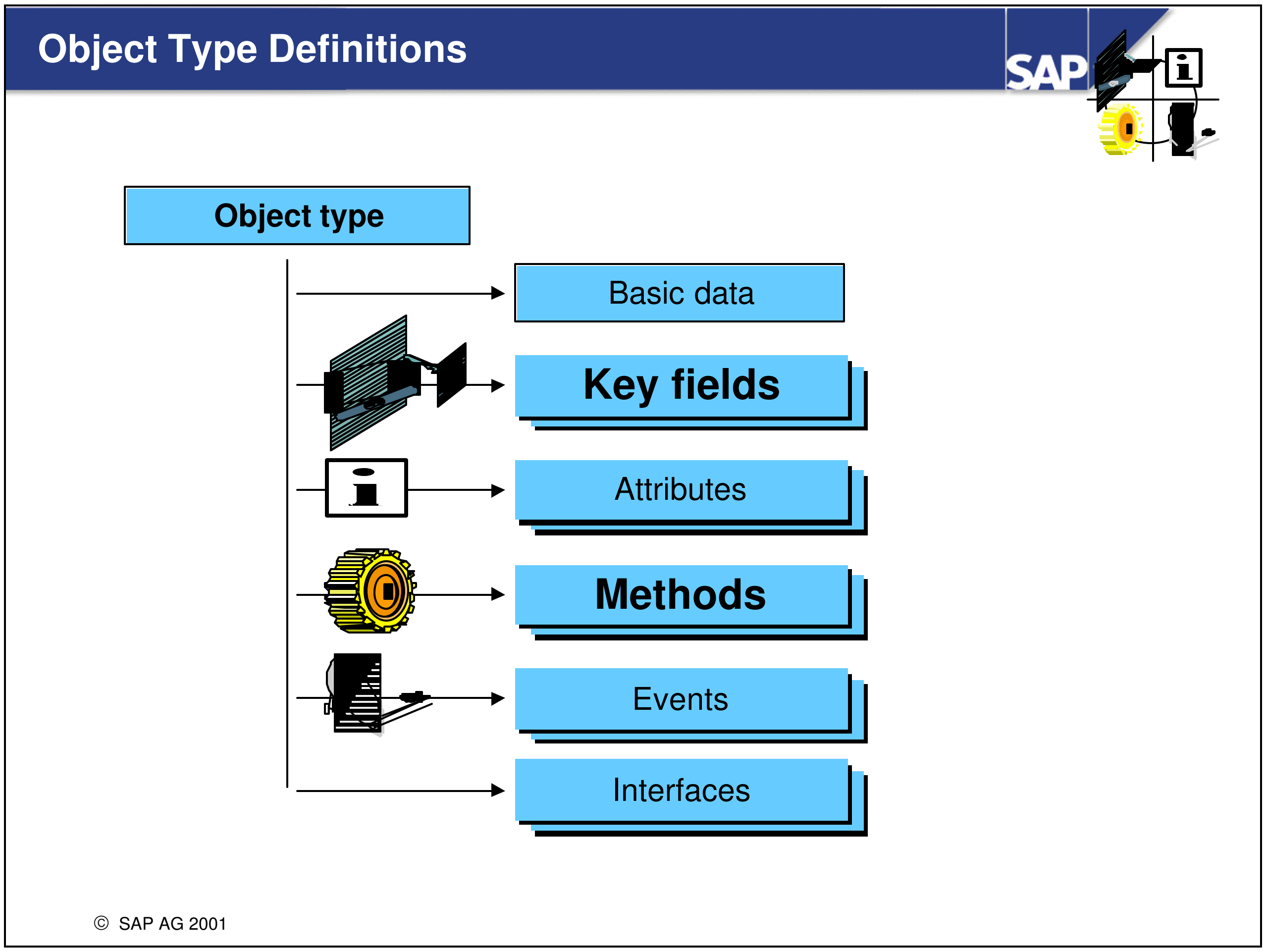
- The BOR is a complete development and runtime environment able to handle the following object types:
  - Business objects: Business objects include objects such as "Customer", "Material", and "CompanyCode". They provide both a high-level business-oriented view of, and a programming interface to, the R/3 System.
  - Technical objects: Technical objects include texts, notes, work items and archived documents, as well as desktop objects like texts, graphics and spreadsheets. These desktop objects can be described in condensed form in the BOR.
  - Metaobjects: Metaobjects document object types, methods, attributes and events. Each object has an attribute "ObjectType" which refers to the metaobject to which it is assigned. The methods, attributes and events available for a particular object can be retrieved from its "ObjectType".
- Business Objects are modeled, abstracted, and stored in the Business Object Repository.

Business Object Repository

- Application hierarchy
- Support for inheritance
- Will be merged with the ABAP Class Library in the future.



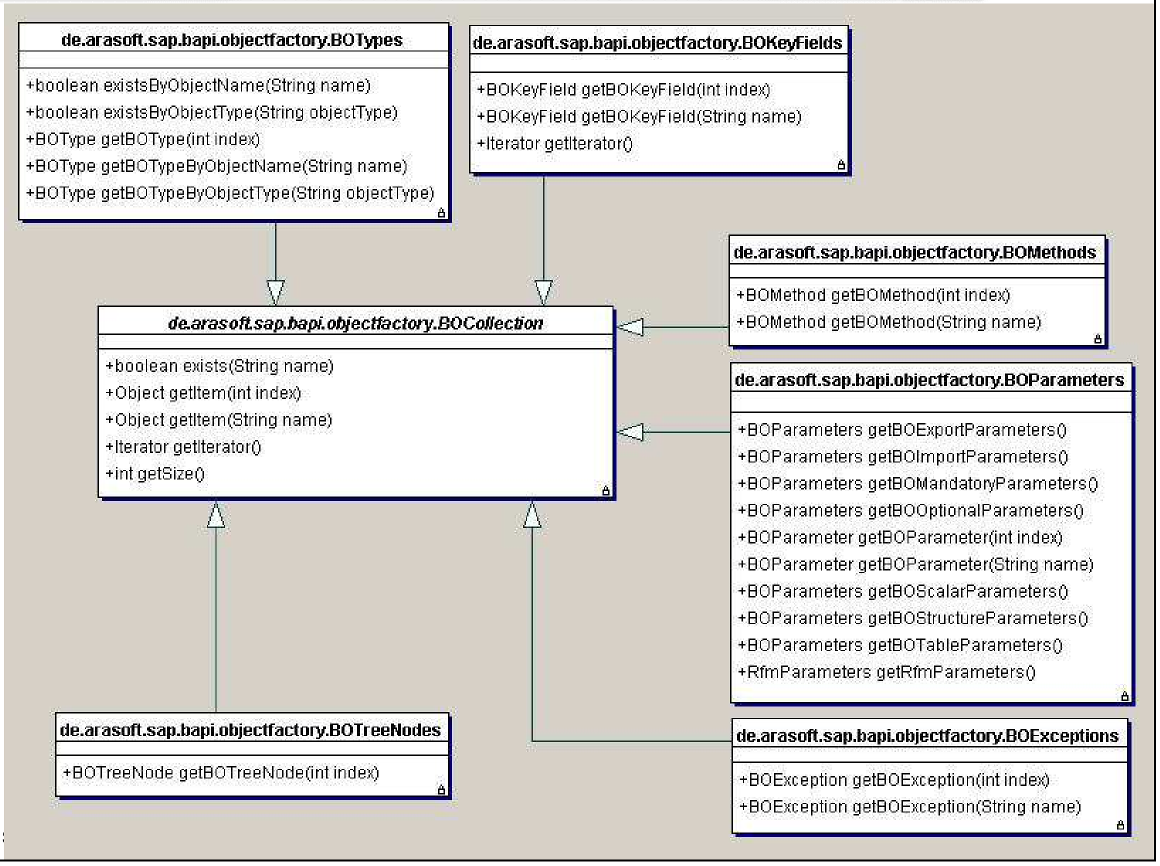
- The BOR organizes SAP objects in a hierarchical scheme within applications.
- The object-oriented principle of inheritance is supported enabling hierarchies and subclassing.



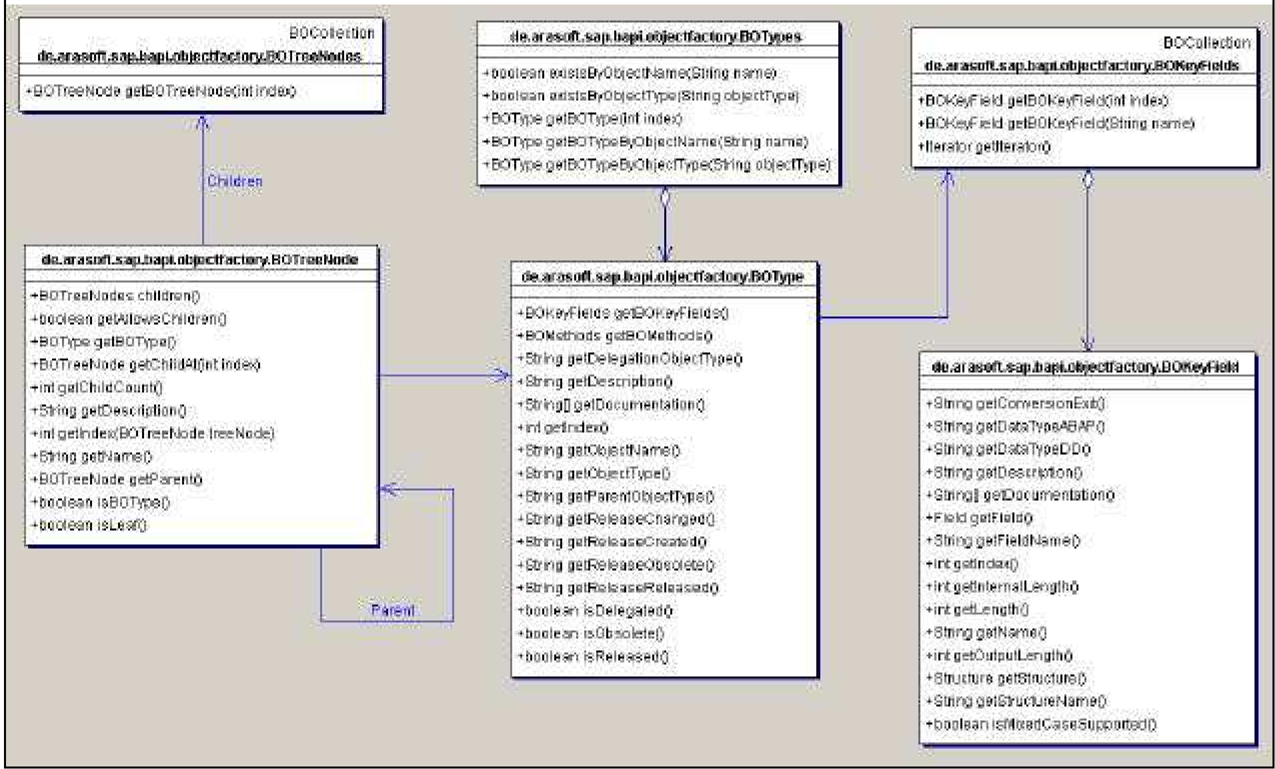
Object Type Definitions

- Only key fields and methods are relevant for BAPI programming.
- Key fields are essential in defining object types. They define field instances in the data dictionary.
- Methods encapsulate and define an object type’s exposed functionality

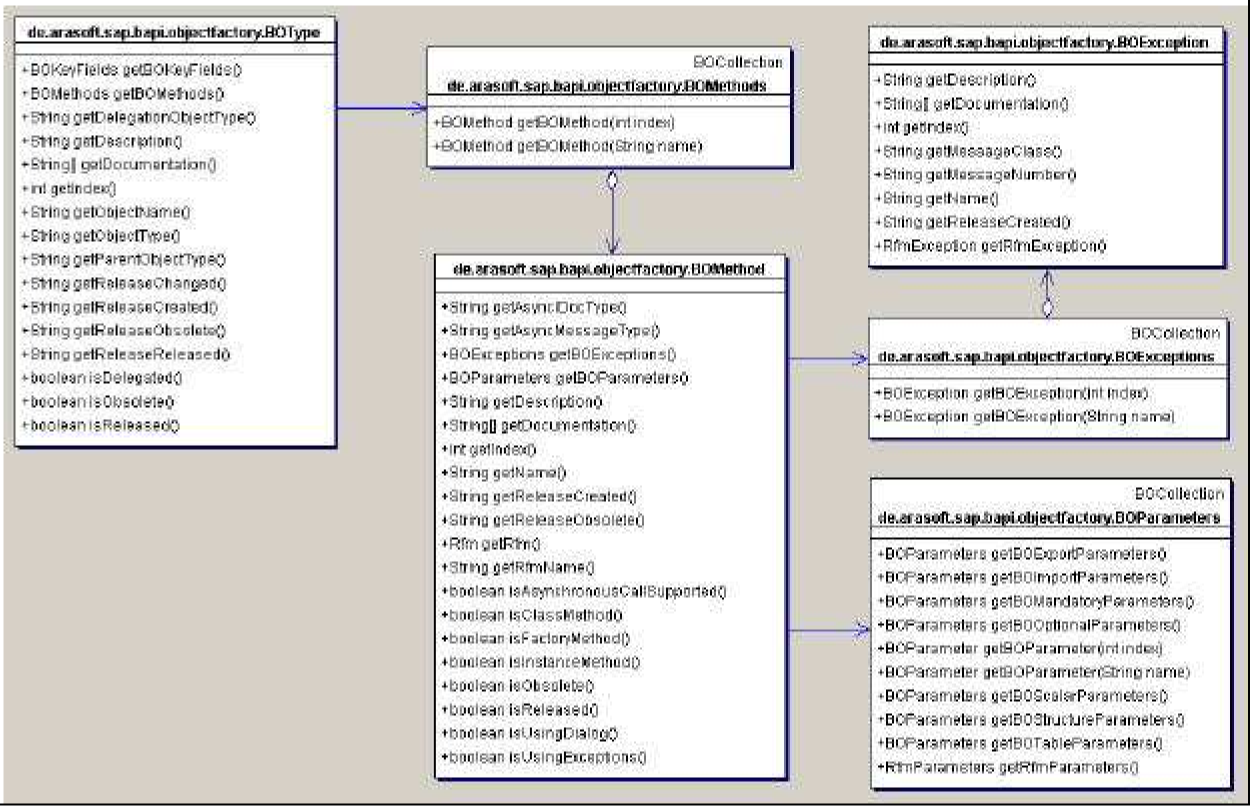
BOR Object Model I


de.arasoft.sap.bapi.objectfactory.BOTypes
+boolean existsByObjectName(String name)
+boolean existsByObjectType(String objectType)
+BOType getBOType(int index)
+BOType getBOTypeByObjectName(String name)
+BOType getBOTypeByObjectType(String objectType)
de.arasoft.sap.bapi.objectfactory.BOKeyFields
+BOKeyField getBOKeyField(int index)
+BOKeyField getBOKeyField(String name)
+Iterator getIterator()
de.arasoft.sap.bapi.objectfactory.BOMethods
+BOMethod getBOMethod(int index)
+BOMethod getBOMethod(String name)
de.arasoft.sap.bapi.objectfactory.BOCollection
+boolean exists(String name)
+Object getItem(int index)
+Object getItem(String name)
+Iterator getIterator()
+int getSize()
de.arasoft.sap.bapi.objectfactory.BOParameters
+BOParameters getBOExportParameters()
+BOParameters getBOImportParameters()
+BOParameters getBOMandatoryParameters()
+BOParameters getBOOptionalParameters()
+BOParameter getBOParameter(int index)
+BOParameter getBOParameter(String name)
+BOParameters getBOScalarParameters()
+BOParameters getBOStructureParameters()
+BOParameters getBOTableParameters()
+RfmParameters getRfmParameters()
de.arasoft.sap.bapi.objectfactory.BOTreeNodes
+BOTreeNode getBOTreeNode(int index)
de.arasoft.sap.bapi.objectfactory.BOExceptions
+BOException getBOException(int index)
+BOException getBOException(String name)

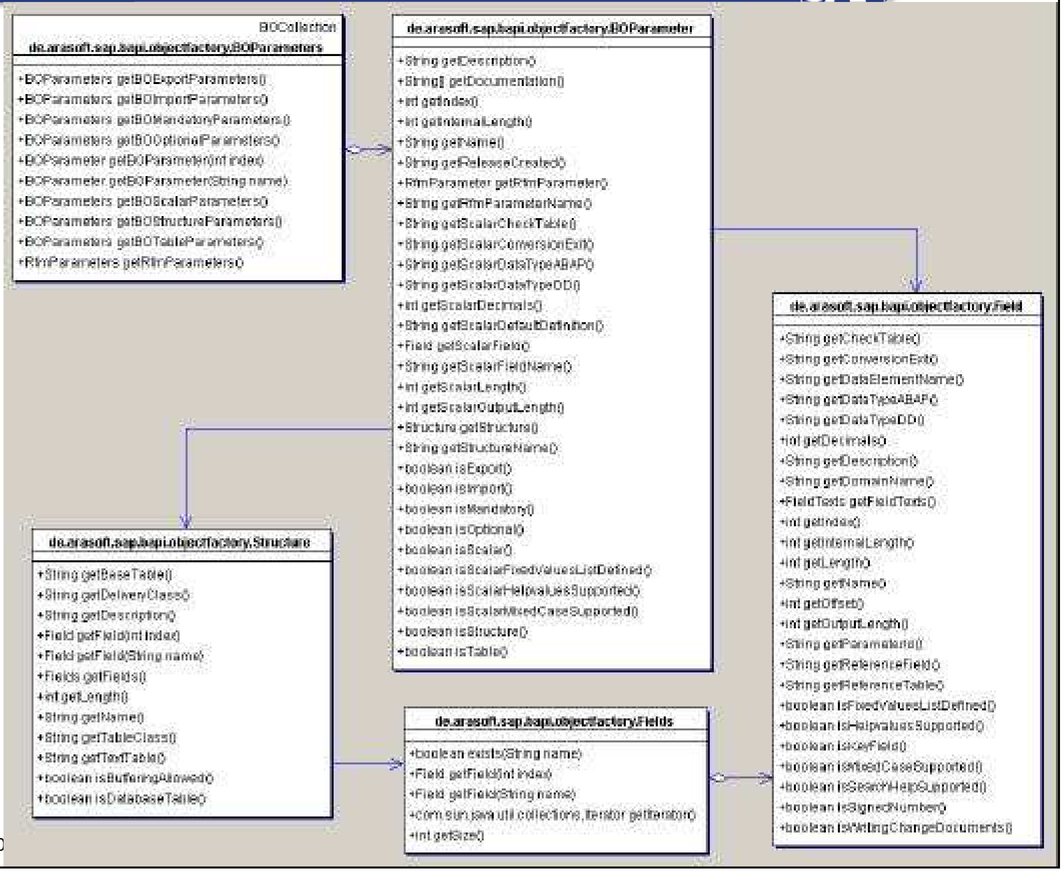
BOR Object Model II


BOCollection
de.arasoft.sap.bapi.objectfactory.BOTreeNodes
+BOTreeNode getBOTreeNode(int index)
Children
de.arasoft.sap.bapi.objectfactory.BOTreeNode
+BOTreeNodes children()
+boolean getAllowsChildren()
+BOType getBOType()
+BOTreeNode getChildAt(int index)
+int getChildCount()
+String getDescription()
+int getIndex(BOTreeNode treeNode)
+String getName()
+BOTreeNode getParent()
+boolean isBOType()
+boolean isLeaf()
Parent
de.arasoft.sap.bapi.objectfactory.BOTypes
+boolean existsByObjectName(String name)
+boolean existsByObjectType(String objectType)
+BOType getBOType(int index)
+BOType getBOTypeByObjectName(String name)
+BOType getBOTypeByObjectType(String objectType)
de.arasoft.sap.bapi.objectfactory.BOType
+BOKeyFields getBOKeyFields()
+BOMethods getBOMethods()
+String getDelegationObjectType()
+String getDescription()
+String[] getDocumentation()
+int getIndex()
+String getObjectName()
+String getObjectType()
+String getParentObjectType()
+String getReleaseChanged()
+String getReleaseCreated()
+String getReleaseObsolete()
+String getReleaseReleased()
+boolean isDelegated()
+boolean isObsolete()
+boolean isReleased()
BOCollection
de.arasoft.sap.bapi.objectfactory.BOKeyFields
+BOKeyField getBOKeyField(int index)
+BOKeyField getBOKeyField(String name)
+Iterator getIterator()
de.arasoft.sap.bapi.objectfactory.BOKeyField
+String getConversionExit()
+String getDataTypeABAP()
+String getDataTypeDD()
+String getDescription()
+String[] getDocumentation()
+Field getField()
+String getFieldName()
+int getIndex()
+int getInternalLength()
+int getLength()
+String getName()
+int getOutputLength()
+Structure getStructure()
+String getStructureName()
+boolean isMixedCaseSupported()

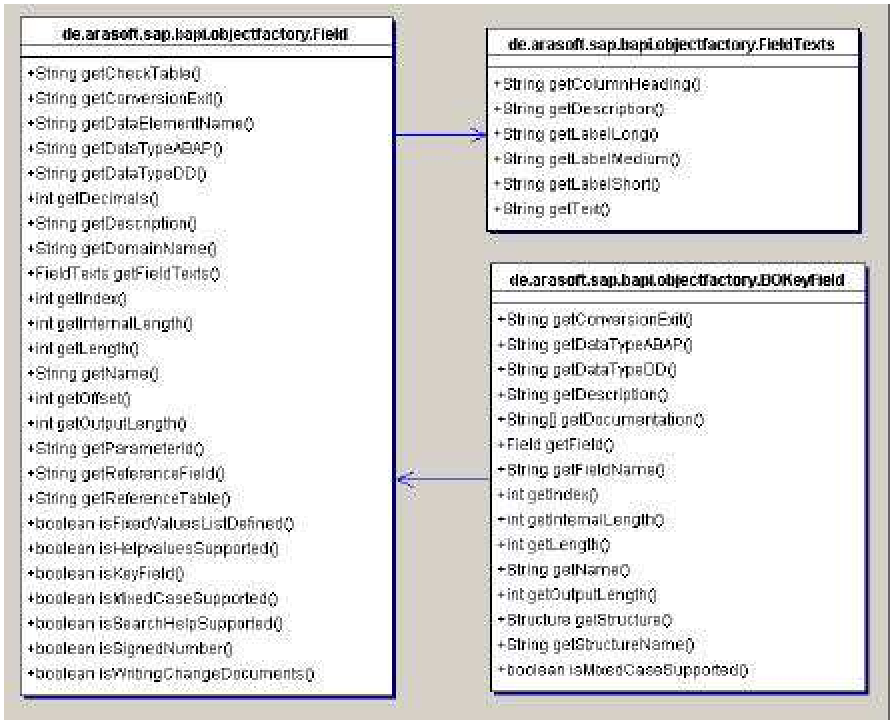
BOR Object Model III


de.arasoft.sap.bapi.objectfactory.BOType
+BOKeyFields getBOKeyFields()
+BOMethods getBOMethods()
+String getDelegationObjectType()
+String getDescription()
+String[] getDocumentation()
+int getIndex()
+String getObjectName()
+String getObjectType()
+String getParentObjectType()
+String getReleaseChanged()
+String getReleaseCreated()
+String getReleaseObsolete()
+String getReleaseReleased()
+boolean isDelegated()
+boolean isObsolete()
+boolean isReleased()
BOCollection
de.arasoft.sap.bapi.objectfactory.BOMethods
+BOMethod getBOMethod(int index)
+BOMethod getBOMethod(String name)
de.arasoft.sap.bapi.objectfactory.BOMethod
+String getAsyncDocType()
+String getAsyncMessageType()
+BOExceptions getBOExceptions()
+BOParameters getBOParameters()
+String getDescription()
+String[] getDocumentation()
+int getIndex()
+String getName()
+String getReleaseCreated()
+String getReleaseObsolete()
+Rfm getRfm()
+String getRfmName()
+boolean isAsynchronousCallSupported()
+boolean isClassMethod()
+boolean isFactoryMethod()
+boolean isInstanceMethod()
+boolean isObsolete()
+boolean isReleased()
+boolean isUsingDialog()
+boolean isUsingExceptions()
de.arasoft.sap.bapi.objectfactory.BOException
+String getDescription()
+String[] getDocumentation()
+int getIndex()
+String getMessageClass()
+String getMessageNumber()
+String getName()
+String getReleaseCreated()
+RfmException getRfmException()
BOCollection
de.arasoft.sap.bapi.objectfactory.BOExceptions
+BOException getBOException(int index)
+BOException getBOException(String name)
BOCollection
de.arasoft.sap.bapi.objectfactory.BOParameters
+BOParameters getBOExportParameters()
+BOParameters getBOImportParameters()
+BOParameters getBOMandatoryParameters()
+BOParameters getBOOptionalParameters()
+BOParameter getBOParameter(int index)
+BOParameter getBOParameter(String name)
+BOParameters getBOScalarParameters()
+BOParameters getBOStructureParameters()
+BOParameters getBOTableParameters()
+RfmParameters getRfmParameters()

BOR Object Model IV


BOCollection
de.arasoft.sap.bapi.objectfactory.BOParameters
+BOParameters getBOExportParameters()
+BOParameters getBOImportParameters()
+BOParameters getBOMandatoryParameters()
+BOParameters getBOOptionalParameters()
+BOParameter getBOParameter(int index)
+BOParameter getBOParameter(String name)
+BOParameters getBOScalarParameters()
+BOParameters getBOStructureParameters()
+BOParameters getBOTableParameters()
+RfmParameters getRfmParameters()
de.arasoft.sap.bapi.objectfactory.BOParameter
+String getDescription()
+String[] getDocumentation()
+int getIndex()
+int getInternalLength()
+String getName()
+String getReleaseCreated()
+RfmParameter getRfmParameter()
+String getRfmParameterName()
+String getScalarCheckTable()
+String getScalarConversionExit()
+String getScalarDataTypeABAP()
+String getScalarDataTypeDD()
+int getScalarDecimals()
+String getScalarDefaultDefinition()
+Field getScalarField()
+String getScalarFieldName()
+int getScalarLength()
+int getScalarOutputLength()
+Structure getStructure()
+String getStructureName()
+boolean isExport()
+boolean isImport()
+boolean isMandatory()
+boolean isOptional()
+boolean isScalar()
+boolean isScalarFixedValuesListDefined()
+boolean isScalarHelpvaluesSupported()
+boolean isScalarMixedCaseSupported()
+boolean isStructure()
+boolean isTable()
de.arasoft.sap.bapi.objectfactory.Field
+String getCheckTable()
+String getConversionExit()
+String getDataElementName()
+String getDataTypeABAP()
+String getDataTypeDD()
+int getDecimals()
+String getDescription()
+String getDomainName()
+FieldTexts getFieldTexts()
+int getIndex()
+int getInternalLength()
+int getLength()
+String getName()
+int getOffset()
+int getOutputLength()
+String getParameterId()
+String getReferenceField()
+String getReferenceTable()
+boolean isFixedValuesListDefined()
+boolean isHelpvaluesSupported()
+boolean isKeyField()
+boolean isMixedCaseSupported()
+boolean isSearchHelpSupported()
+boolean isSignedNumber()
+boolean isWritingChangeDocuments()
de.arasoft.sap.bapi.objectfactory.Structure
+String getBaseTable()
+String getDeliveryClass()
+String getDescription()
+Field getField(int index)
+Field getField(String name)
+Fields getFields()
+int getLength()
+String getName()
+String getTableClass()
+String getTextTable()
+boolean isBufferingAllowed()
+boolean isDatabaseTable()
de.arasoft.sap.bapi.objectfactory.Fields
+boolean exists(String name)
+Field getField(int index)
+Field getField(String name)
+com.sun.java.util.collections.Iterator getIterator()
+int getSize()

BOR Object Model V



```
de.arasoft.sap.bapi.objectfactory.Field
+String getCheckTable()
+String getConversionExit()
+String getDataElementName()
+String getDataTypeABAP()
+String getDataTypeDD()
+int getDecimals()
+String getDescription()
+String getDomainName()
+FieldTexts getFieldTexts()
+int getIndex()
+int getInternalLength()
+int getLength()
+String getName()
+int getOffset()
+int getOutputLength()
+String getParameterId()
+String getReferenceField()
+String getReferenceTable()
+boolean isFixedValuesListDefined()
+boolean isHelpvaluesSupported()
+boolean isKeyField()
+boolean isMixedCaseSupported()
+boolean isSearchHelpSupported()
+boolean isSignedNumber()
+boolean isWritingChangeDocuments()
```

```
de.arasoft.sap.bapi.objectfactory.FieldTexts
+String getColumnHeading()
+String getDescription()
+String getLabelLong()
+String getLabelMedium()
+String getLabelShort()
+String getText()
```

```
de.arasoft.sap.bapi.objectfactory.BOKeyField
+String getConversionExit()
+String getDataTypeABAP()
+String getDataTypeDD()
+String getDescription()
+String[] getDocumentation()
+Field getField()
+String getFieldName()
+int getIndex()
+int getInternalLength()
+int getLength()
+String getName()
+int getOutputLength()
+Structure getStructure()
+String getStructureName()
+boolean isMixedCaseSupported()
```

BAPI Parameters

- Table parameters in the BOR can be defined as
  - Import only
  - Export only
  - Import and Export

- Export parameters are values/messages coming from R/3 to the calling application.
- Import parameters are values/messages supplied by the calling application to R/3.

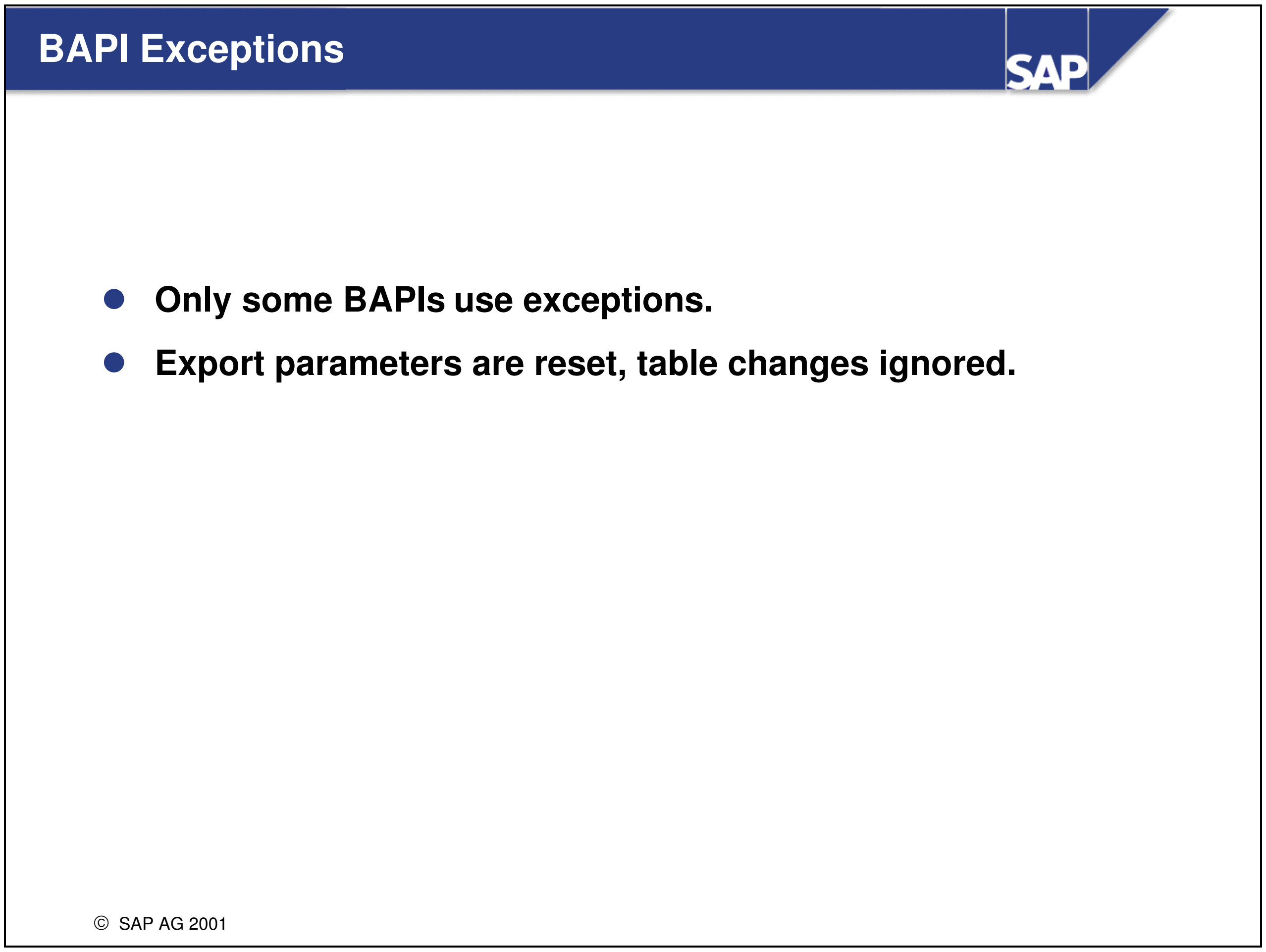
Return Code Information

- Usually a structure, sometimes a table
- Data dictionary structures used
  - BAPIRETURN
  - BAPIRETURN1
  - BAPIRET1
  - BAPIRET2
- Article on BAPI error handling:
  http://www.intelligenterp.com/columns/archive/bapi4.shtml

BapiReturn Structure

- Type — Message type
  - blank or "S"=Success, "E"=Error, "W"=Warning, "I"=Information, "A"=Abort

Type	Message type
Code	Message number
Message	Message text
Log_No	Application Log Number
Log_Msg_No	Application Log Message Serial Number
Message_V1 - V4	Message variables



BAPI Exceptions

- Only some BAPIs use exceptions.
- Export parameters are reset, table changes ignored.

- When exceptions occur, any changes to R/3 tables are ignored.

BOR Delegation

- If you are using proxies, you need to regenerate them after making changes in SAP.
- If you are calling the RFMs directly, you need to change your code after making changes in SAP.

BOR Tools

- SAP transactions (using SAP-GUI)

BAPI	Business Object Browser (BAPIs only)
SWO1	Business Object Builder (all objects)
SWO2	Business Object Browser (all objects)
SE11	Data Dictionary
SE37	Function Builder

Writing your own BAPIs

- Read BAPI Programming Guide.
- Subclass an SAP object type (or add a new object type).
- Define and implement an RFC-enabled function module.
- Use wizard to define your BAPI in the BOR inside R/3.

- The BAPI programming guide is available from the Open BAPI Network.
- Subclasses of existing SAP object types can be developed to take advantage of existing functionality.

Additional Information

- www.sapinsider.com
- www.sappro.com

RFMs, BAPIs, and Tools: Unit Summary

- RFMs are ABAP function modules that can be invoked from the outside.
- BAPIs are RFMs that follow additional rules and are defined as object methods in the BOR.

RFMs, BAPIs, and Tools Exercises

Unit: RFMs, BAPIs, and Tools

Topic: Tools

At the conclusion of these exercises you will be able to:

Find required metadata of the BAPIs.

1-1 Starting in transaction BAPI, answer the following questions:

1-1-1 Which object type is SalesOrder delegated to in this system?

1-1-2 List the BOR name, data type and length for each key field of SalesOrder.

1-1-3 List the instance methods of SalesOrder.

1-1-4 List the mandatory table parameters for the SalesOrder.CreateFromDat1 method.

1-1-5 List the obsolete BAPIs of the SalesOrder object type.

1-1-6 What is the name of the RFC-enabled function module (RFM) that implements SalesOrder.CreateFromDat1?

1-1-7 What is the name of the OrderItemsIn parameter of the SalesOrder.CreateFromDat1 BAPI on the RFC level?

1-1-8 What is the name of the check table of the COUNTRY field in the OrderPartners parameter of the SalesOrder.CreateFromDat1 BAPI?

1-1-9 Which authorization objects are checked in the PurchaseRequisition.CreateFromData method? (Refer to the documentation.)

1-1-10 Execute CompanyCode.GetList in the Function Builder. How many rows are returned?

1-1-11 Execute CompanyCode.GetDetail for the third company code returned by the previous question. What is the ISO language code used?

RFMs, BAPIs, and Tools Solutions

Unit: RFMs, BAPIs, and Tools

Topic: Tools

1-1 Starting in transaction BAPI, answer the following questions:

1-1-1 Which object type is SalesOrder delegated to in this system?

CustomerOrder

1-1-2 List the BOR name, data type and length for each key field of SalesOrder.

SalesDocument Char 10

1-1-3 List the instance methods of SalesOrder.

ChangeFromData

GetStatus

1-1-4 List the mandatory table parameters for the SalesOrder.CreateFromDat1 method.

OrderItemsIn

OrderPartners

1-1-5 List the obsolete BAPIs of the SalesOrder object type.

CreateFromData

1-1-6 What is the name of the RFC-enabled function module (RFM) that implements SalesOrder.CreateFromDat1?

BAPI_SALESORDER_CREATEFROMDAT1

1-1-7 What is the name of the OrderItemsIn parameter of the SalesOrder.CreateFromDat1 BAPI on the RFC level?

ORDER_ITEMS_IN

1-1-8 What is the name of the check table of the COUNTRY field in the OrderPartners parameter of the SalesOrder.CreateFromDat1 BAPI?

T005

1-1-9 Which authorization objects are checked in the PurchaseRequisition.CreateFromData method? (Refer to the documentation.)

M_BANF_BSA

M_BANF_EKG

M_BANF_EKO

M_BANF_WRK

1-1-10 Execute CompanyCode.GetList in the Function Builder. How many rows are returned?

(Answer depends on the system.)

1-1-11 Execute CompanyCode.GetDetail for the third company code returned by the previous question. What is the ISO language code used?

(Answer depends on the system.)

JCO Client Programming: Contents

JCO Client Programming: Unit Objectives

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

- **Understand the JCO Middleware and its important classes and methods.**

JCO Overview

- High-performance JNI-based middleware
- Support R/3 3.1H and higher.
- Supports inbound and outbound calls.
- Supports client pooling.
- Supports desktop and web/application server applications.
- Multi-platform
  - First release: Windows 32 only
- Complete and correct code page handling
- Easy to install and deploy

Installation and Deployment

- Required files in \WINNT\system32:

librfc32.dll	(at least 46D, build 263)
jRFC11.dll	(JDK 1.1)
jRFC12.dll	(JDK 1.2 and 1.3)

- Required files in Java classpath:
  - jCO.jar

Import Statements

- **One import statement for JCO**
  - **Sample code:**

```
import com.sap.mw.jco.*;
```

JCO.Client

- An object of type JCO.Client represents a session with SAP.
- Instantiated via a call to JCO.createClient(…) or reserved in a client pool.

JCO.Client User Properties

- Client
  - SAP client

```
String getClient()
```

- User
  - Userid

```
String getUser()
```

- Language
  - User language (2-byte ISO code)

```
String getLanguage()
```

JCO.Client Application Server Properties

- **ASHost**
  - **Application server host name**

```
String getASHost()
```

- **SystemNumber**
  - **SAP system number**

```
String getSystemNumber()
```

JCO.Client Group Properties

- Group
  - Load balancing group

```
String getGroup()
```

- MSHost
  - Message server host name

```
String getMSHost()
```

- SystemID
  - SAP system name

```
String getSystemID()
```

JCO.Client Property Setting

- **Setting a property**

```
void setProperty(String key, String value)
```

  - **Sample code:**

```
mConnection.setProperty("jco.client.user", "C3026902");
```

jco.client.client	
jco.client.user	
jco.client.passwd	
jco.client.lang	
jco.client.ashost	
jco.client.sysnr	SAP system number
jco.client.group	
jco.client.mshost	
jco.client.r3name	SAP system name
jco.client.trace	("0" or "1")

JCO.Client createClient I

- JCO.createClient(…) for a specific application server

```
JCO.Client JCO.createClient(...)
```

  - Sample code:

```
JCO.Client mConnection = null;
mConnection =
JCO.createClient("400",           // SAP client
                 "c3026902",      // userid
                 "*********",     // password
                 "EN",            // language
                 "iwdf5020",      // application server host name
                 "00");           // SAP system number
```

  - Language can be null; system default language used.

JCO.Client createClient II

- JCO.createClient(...) for a load balancing group

```
JCO.Client JCO.createClient(...)
```

- Sample code:

```
JCO.Client mConnection = null;
mConnection =
JCO.createClient("400",             // SAP client
                 "c3026902",        // userid
                 "********",        // password
                 "EN",              // language
                 "iwdf5020",        // message server host name
                 "T20",             // SAP system name
                 "PUBLIC");         // group name
```

- Language can be null; system default language used.

JCO.Client createClient III

- **JCO.createClient(…) with a properties file**

```
JCO.Client JCO.createClient(Properties properties)
```

  - **Sample code:**

```
JCO.Client mConnection = null;
OrderedProperties logonProperties =
  OrderedProperties.load("/logon.properties");
mConnection =
  JCO.createClient(logonProperties);
```

  - **logon.properties:**

```
jco.client.client=400
jco.client.user=c3026902
jco.client.passwd=********
jco.client.ashost=iwdf5020
jco.client.sysnr=00
```

JCO.Client Methods I

- Connecting to SAP

```
void connect()
```

- Disconnecting from SAP

```
void disconnect()
```

- Executing an RFM

```
void execute(JCO.Function function)
```

- Retrieving attributes

```
JCO.Attributes getAttributes()
```

- Enable SAPGUI

```
void setSapGui(int use_sapgui)
```

  - 0 without SAPGUI (default)
  - 1 with SAPGUI
  - 2 with invisible SAPGUI

JCO.Client Methods II

- Check the connection

```
boolean isAlive()
```

- Check the state

```
public byte getState()
```

  - State bits

`JCO.STATE_DISCONNECTED`	The client is disconnected
`JCO.STATE_CONNECTED`	The client is connected to the remote server
`JCO.STATE_BUSY`	The client is currently executing a remote function call
`JCO.STATE_USED`	A pooled connection has been allocated by a program

JCO.Client Trace/Debug Methods

- Is trace enabled?

```
boolean getTrace()
```

- Enable trace

```
void setTrace(boolean trace)
```

  - The trace must be turned on *before* connect().

- Is ABAP debugging enabled?

```
boolean getAbapDebug()
```

- Enable ABAP debugging

```
void setAbapDebug(boolean debug)
```

JCO.Attributes I

- SAP client (CLIENT)

```
String getClient()
```

- Userid (USER)

```
String getUser()
```

- SAP language code (1 byte) (LANGUAGE)

```
String getLanguage()
```

  - Only available if language has been explicitly set.

- ISO language code (2 byte) (ISO_LANGUAGE)

```
String getISOLanguage()
```

  - Only available if language has been explicitly set.

- SAP host name (PARTNER_HOST)

```
String getPartnerHost()
```

JCO.Attributes II

- SAP system name (SYSID)

```
String getSystemID()
```

- SAP system number (SYSTNR)

```
String getSystemNumber()
```

- SAP host release (PARTNER_REL)

```
String getPartnerRelease()
```

- Client host name (OWN_HOST)

```
String getHost()
```

- RFC library release (OWN_REL)

```
String getRelease()
```

- Trace setting (TRACE)

```
boolean getTrace()
```

Client Pooling

- Each client pool has a name.
- Each client pool is global within one JVM.
- Each client pool uses a specific userid within a specific SAP client in a specific SAP system.
- Only useful for anonymous users with the same (limited) authorizations.
- A session exists from the time the JCO.Client is reserved until it is released.
  - Release Client objects as soon as possible.
- All pools managed by JCO.PoolManager.
- Reusing a connection with reset supported since 4.0A.
- Do not use Client objects reserved in one thread in another thread.

Creating And Removing A Client Pool

- Creating a new client pool

```
void JCO.addClientPool(...)
```

  - Sample code:

```
static final String SID = "Pool";
JCO.addClientPool(SID,                    // pool name
                  10,                     // maximum number of connections
                  "400",                  // SAP client
                  "c3026902",             // userid
                  "********",             // password
                  "EN",                   // language
                  "iwdf5020",             // host name
                  "00");                  // system number
```

- Overloaded versions exist for load balancing and the use of a properties file.
- Removing a client pool

```
JCO.removeClientPool(String key);
```

Using A Client In A Client Pool I

- **Reserving a JCO.Client**

```
JCO.Client JCO.getClient(String key)
JCO.Client JCO.getClient(String key, boolean reset)
```

  - **Sample code I:**

```
JCO.Client mConnection = null;
mConnection = JCO.getClient(SID);
```

  - **No reset !!!**
  - **getClient(...) will fail if all clients in the pool are in use.**
  - **You have to program your own wait/retry logic.**
  - **Sample code II:**

```
JCO.Client mConnection = null;
mConnection = JCO.getClient(SID, true);
```

  - **Resets the session (/n in SAPGUI).**
  - **Dis- and reconnects before SAP 4.0A.**

Using A Client In A Client Pool II

- **Releasing a JCO.Client**

```
void JCO.releaseClient(JCO.Client client)
```

  - **Make sure you call `releaseClient()` also when an exception happens (`finally`...).**

JCO.PoolManager

- Accessing the pool manager

```
JCO.PoolManager JCO.getClientPoolManager()
```

- Accessing a pool

```
JCO.Pool getPool(String pool_name)
```

JCO.Pool

- **Retrieving the maximum number of clients in the pool**

```
int getMaxPoolSize()
```

JCO.PoolChangedListener Interface

- Listen to changes in pools.
- Method to be implemented

```
public void poolChanged(JCO.Pool pool)
```

- Add listener

```
myPoolManager.addPoolChangedListener(this);
```

JCO.Repository

- Represents SAP metadata.
- Created via a call to the constructor.

```
JCO.Repository (String name, JCO.Client client)
JCO.Repository (String name, String pool_name)
```

- The first parameter is an arbitrary name.
  - Sample code:

```
JCO.Repository mRepository;
  mRepository = new JCO.Repository("CA926",
   mConnection);
```

JCO.Repository Methods

- Create a function template

```
IFunctionTemplate getFunctionTemplate(String function_name)
```

- Returns null if the RFM does not exist.
- A function template represents the metadata for an RFM, but has no data; a function object has parameter data. This is similar to the class/object relationship in Java.

IFunctionTemplate Methods

- **Create a JCO.Function object.**

```
JCO.Function getFunction()
```

  - **Sample code:**

```
JCO.Repository mRepository;
  mRepository = new JCO.Repository("CA926", mConnection);
public JCO.Function createFunction(String name) {
  try {
    return
      mRepository.
        getFunctionTemplate(name.toUpperCase()).getFunction();
  }
  catch (Exception ex) {}
  return null;
}
```

JCO.Function

- Represents an RFM.
- Contains all metadata required for using the function.
  - Sample code:

```
JCO.Function function =
  createFunction("BAPI_COMPANYCODE_GETLIST");
if (function != null) {
  mConnection.execute(function);
}
```

JCO.Function Methods

- **Accessing the parameters**

```
JCO.ParameterList getExportParameterList()
JCO.ParameterList getImportParameterList()
JCO.ParameterList getTableParameterList()
```

  - **All three methods return null if no parameters of the specific type exist for the function.**

Accessing the exceptions

```
JCO.AbapException[ ] getExceptionList()
```

  - **Returns null if no exceptions exist for the function.**

JCO.AbapException Methods

- Accessing the exception definition

`String getKey()`

  - The ABAP exception code

`String getMessage()`

  - The ABAP exception description

Mapping Between ABAP And Java Data Types

ABAP		Java	
b	1-byte integer	int	JCO.TYPE_INT1
s	2-byte integer	int	JCO.TYPE_INT2
I	4-byte integer	int	JCO.TYPE_INT
C	Character	String	JCO.TYPE_CHAR
N	Numerical Character	String	JCO.TYPE_NUM
P	Binary Coded Decimal	BigDecimal	JCO.TYPE_BCD
D	Date	Date	JCO.TYPE_DATE
T	Time	Date	JCO.TYPE_TIME
F	Float	double	JCO.TYPE_FLOAT
X	Raw data	byte[ ]	JCO.TYPE_BYTE
g	String (variable-length)	String	JCO.TYPE_STRING
y	Raw data (variable-length)	byte[ ]	JCO.TYPE_XSTRING

JCO.ParameterList Methods I

- **Accessing generic metadata**

```
int getFieldCount()
String getName(int index)
boolean isExport(String name/int index)
boolean isImport(String name/int index)
boolean isOptional(String name/int index)
boolean isStructure(String name/int index)
boolean isTable(String name/int index)
```

- **Accessing metadata for scalars**

```
int getDecimals(String name/int index)
String getDefault(String name/int index)
int getLength(String name/int index)
int getType(String name/int index)
```

  - **Cf. mapping table.**

Generic Parameter Metadata

- **Sample code:**

```
JCO.ParameterList params = function.getImportParameterList();
if (params != null) {
  for (int i = 0; i < params.getFieldCount(); i++) {
    String s = params.isStructure(i) ? "structure" : "scalar";
    System.out.println(params.getName(i) + " is a " + s +
                       " import parameter.");
  }
}
```

JCO.ParameterList Methods II

- **Accessing scalars**

```
JCO.Field getField(String name/int index)
BigDecimal getBigDecimal(String name/int index)
byte[] getByteArray(String name/int index)
Date getDate(String name/int index)
double getDouble(String name/int index)
int getInt(String name/int index)
String getString(String name/int index)
Date getTime(String name/int index)
Object getValue(String name/int index)
void setValue(..., String name/int index)
```

  - **Sample code:**

```
function.getImportParameterList().setValue("0001", "COMPANYCODEID");
```

JCO.Field Methods

- **Retrieving a value**

```
BigDecimal getBigDecimal()
byte[] getByteArray()
Date getDate()
double getDouble()
int getInt()
String getString()
Date getTime()
Object getValue()
```

- **Setting a value**

```
void setValue(...)
```

- **If you want to use the default defined for an optional scalar import parameter, do not assign any value or assign an empty string.**

JCO.Field Metadata Access

```
String getDefault()
int getLength()
int getType()
```

- Cf. mapping table.

```
int getDecimals()
```

is currently not available, but instead you can use

```
function.getExportParameterList().getDecimals
         (String name/int index))
```

JCO.ParameterList Methods III

- **Accessing structures/tables**

```
JCO.Structure getStructure(String name/int index)
JCO.Table getTable(String name/int index)
```

  - **Sample code:**

```
JCO.Structure returnStructure =
  function.getExportParameterList().getStructure("RETURN");
```

JCO.Structure

- **Accessing fields in the structure**

```
JCO.Field getField(String name/int index)
BigDecimal getBigDecimal(String name/int index)
byte[] getByteArray(String name/int index)
Date getDate(String name/int index)
double getDouble(String name/int index)
int getInt(String name/int index)
String getString(String name/int index)
Date getTime(String name/int index)
Object getValue(String name/int index)
void setValue(..., String name/int index)
```

- **Accessing structure metadata**

```
int getFieldCount()
String getName()
```

  - Name of the SAP Data Dictionary structure

Accessing Fields In A Structure

- **Sample code:**

```
JCO.Structure returnStructure =
  function.getExportParameterList().getStructure("RETURN");
if (! (returnStructure.getString("TYPE").equals("") ||
       returnStructure.getString("TYPE").equals("S")) ) {
  System.out.println(returnStructure.getString("MESSAGE"));
}
```

JCO.Table I

- Based on the recordset paradigm.
- Supports all methods found in JCO.Structure.
- Navigating a table

```
int getNumRows()
void setRow(int pos)
int getRow()
```

  - Current position

```
void firstRow()
void lastRow()
boolean nextRow()
```

  - False if at end of table

```
boolean previousRow()
```

  - False if at beginning of table

JCO.Table II

- **Manipulating a table**

`void appendRow()`

  - Current row is the new row.

`void appendRow(int num_rows)`

  - Current row is the first new row.

`void deleteAllRows()`

`void deleteRow()`

  - Deletes the current row.
    - if (current_row < last_row) current_row = current_row;
      else current_row--;

`void deleteRow(int pos)`

    - if (current_row < last_row) current_row = current_row;
      else current_row--;

`void insertRow(int pos)`

  - Current row is the new row.

Browsing A Table

- **Sample code I:**

```
JCO.Table codes =
  function.getTableParameterList().getTable("COMPANYCODE_LIST");
for (int i = 0; i < codes.getNumRows(); i++) {
  codes.setRow(i);
  System.out.println(codes.getString("COMP_CODE") + '\t' +
                     codes.getString("COMP_NAME"));
}
```

- **Sample code II:**

```
JCO.Table codes =
  function.getTableParameterList().getTable("COMPANYCODE_LIST");
if (codes.getNumRows() > 0) {
  do {
    System.out.println(codes.getString("COMP_CODE") + '\t' +
                       codes.getString("COMP_NAME"));
  } while (codes.nextRow());
}
```

Adding A New Table Row

- **Sample code:**

```
JCO.Table codes =
  function.getTableParameterList().getTable("COMPANYCODE_LIST");
codes.appendRow();
codes.setValue("XXXX", "COMP_CODE");
codes.setValue("Does not exist", "COMP_NAME");
```

Adding Multiple Table Rows

- **Sample code:**

```
table.appendRows(10);
do {
  table.setValue("Value1","FIELD1");
// ...
} while (table.nextRow());
```

  - **This is much faster than individual `appendRow()` calls.**

Optional Tables

- You can deactivate optional tables that you do not need in your application.
  - Sample code:

```
function.getTableParameterList().setActive(false,
                                           "COMPANYCODE_LIST");
```

  - Results in better performance!

Exception Handling

- JCO throws three types of exceptions:
  - JCO.Exception
  - JCO.ConversionException
  - JCO.AbapException

JCO.Exception

- Generic exception

```
int getGroup()
```

  - e.g.

```
        JCO_ERROR_COMMUNICATION
        JCO_ERROR_LOGON_FAILURE
        JCO_ERROR_SYSTEM_FAILURE
String getKey()
String getMessage()
```

JCO.ConversionException

- Extends JCO.Exception.
- Thrown if a type conversion fails.

JCO.AbapException

- Extends JCO.Exception.
- Thrown if a BAPI/RFM throws an ABAP exception.

```
String getKey()
```

  - Contains the ABAP Exception code.

Exception Handling Sample Code

```
JCO.Function function = this.createFunction("DDIF_FIELDINFO_GET");
try {
  function.getImportParameterList().setValue("MARA", "TABNAME");
  mConnection.execute(function);
}
catch (JCO.AbapException ex) {
  if (ex.getKey().equalsIgnoreCase("NOT_FOUND")) {
    System.out.println("Dictionary structure/table not found.");
    System.exit(1);
  }
  else {
    System.out.println(ex.getMessage());
    System.exit(1);
  }
}
catch (JCO.Exception ex) {
// Handle the exception
}
catch (Exception ex) {
// Handle the exception
}
```

Synchronization in JCO

- Access to Pool and Repository objects is synchronized, everything else is not.
  - Sample code:

```
synchronized (table) {
  table.appendRows(10);
  do {
    table.setValue("Value1","FIELD1");
  // ...
  } while (table.nextRow());
}
```

Call Sequence for tRFC Inbound

- Read function data from DB.
- Retrieve TID (`createTID()`).
- Store TID and function data key in DB.
- DB Commit
- Execute application function.
- If successful
  - Delete TID and function data key
  - Delete function data
  - DB Commit
  - Call `confirmTID(tid)`

  If unsuccessful
  - Retry later, starting with step 5.

JCO.Client Methods for tRFC

- Creating a TID

```
String createTID()
```

- Executing an RFM via tRFC

```
void execute(JCO.Function function, String tid)
```

- Confirming a TID

```
void confirmTID(String tid)
```

JCO Client Programming: Unit Summary

- You now know how to use the JCO middleware to call BAPIs and other RFMs.

JCO Client Programming Exercises

Unit: JCO Client Programming

Topic: Connection Attributes

At the conclusion of these exercises you will be able to:

Connect to SAP and display connection attributes.

1-1 Install the Java SDK.

1-2 Install JCO (unzip to a directory called JCO, keep the directory structure). Copy librfc32.dll, jRFC11.dll, and jRFC12.dll to c:\winnt\system32.

1-3 Install Forte for Java.

1-4 Create a directory CA926.

1-5 Start Forte and create a new project (CA926). Mount the CA926 directory, add jCO.jar to the file systems.

1-6 Create a class Client1 that does the following:

1-6-1 Connect to SAP using a connection created with createClient().

1-6-2 List the connection attributes using System.out.println.

1-6-3 Disconnect.

Exercises

Unit: JCO Client Programming

Topic: Client Pooling

At the conclusion of these exercises you will be able to:

Connect to SAP using a client pool.

2-1 Create a class Client2 that does the same as Client1, but uses a client pool.

Exercises

Unit: JCO Client Programming

Topic: Connection Properties

At the conclusion of these exercises you will be able to:

Connect to SAP using a properties file.

3-1 Copy OrderedProperties.java and logon.properties to your CA926 directory.

3-2 Compile OrderedProperties.java.

3-3 Modify logon.properties.

3-4 Create a class Client3 that does the same as Client2, but uses the properties file rather than hardcoded information.

3-4-1 Use the following code to load the properties:

```
OrderedProperties logonProperties =
OrderedProperties.load("/logon.properties");
```

Exercises

Unit: JCO Client Programming

Topic: RFM Calls with Scalar Parameters

At the conclusion of these exercises you will be able to:

Call RFMs with scalar parameters.

4-1 Create a class SAPServices with a static method using the following signature:
public static String getLanguage(
  JCO.Client connection,
  IRepository repository) throws Exception

4-2 Implement this method to call RFM RFC_GET_SAP_SYSTEM_PARAMETERS and return the user's logon language (internal one-byte code).

4-3 Create a class TestLanguage that creates a Repository object, calls getLanguage() in SAPServices, and displays the user language.

Exercises

Unit: JCO Client Programming

Topic: RFM Calls with Structure Parameters

At the conclusion of these exercises you will be able to:

Call RFMs with structure parameters.

5-1 Create a class SAPServices2 that extends SAPServices with a static method using the following signature:

```
public static String getOpSys(JCO.Client connection,
IRepository repository) throws Exception
```

5-2 Implement this method to call RFM RFC_SYSTEM_INFO and return the operating system for the application server.

5-3 Create a class TestOpSys that creates a Repository object, calls getOpSys() in SAPServices2, and displays the operating system.

Exercises

Unit: JCO Client Programming

Topic: ABAP Exceptions and Tables

At the conclusion of these exercises you will be able to:

Call RFMs with table parameters and handle ABAP exceptions.

6-1 Create a class SAPServices3 that extends SAPServices2 with a static method using the following signature:

```
public static String getFieldText(
  JCO.Client connection,
  IRepository repository,
  String tableName,
  String fieldName) throws Exception
```

6-2 Implement this method to call RFM DDIF_FIELDINFO_GET and return the description for a field in a table or structure. Handle the ABAP exception NOT_FOUND and return an empty string if this exception is returned from SAP.

6-3 Create a class TestFieldText that calls getFieldText() in SAPServices3, and displays the description for field KNA1-KUNNR.

Exercises

Unit: JCO Client Programming

Topic: BAPIs

At the conclusion of these exercises you will be able to:

Call class method BAPIs

7-1 Create a class SAPCompanyCode.

7-2 Add and implement the following constructor:
public SAPCompanyCode(String code, String name)

7-3 Add and implement the following methods:
public String getCode()
public String getName()

7-4 Add the following static method:
public static SAPCompanyCode[] getList(JCO.Client connection, IRepository repository) throws Exception

7-5 Implement this method to call the CompanyCode.GetList BAPI. Check the BAPI return code, throw an exception if it is bad. Otherwise return an array of suitably initialized SAPCompanyCode objects.

7-6 Create a class TestCompanyCode that calls SAPCompanyCode.getList() and display all returned information.

Exercises

Unit: JCO Client Programming

Topic: BAPIs

At the conclusion of these exercises you will be able to:

Call instance method BAPIs

8-1 Add the following method to SAPCompanyCode:
public String getCity(JCO.Client connection, IRepository repository) throws Exception

8-2 Implement this method to call the CompanyCode.GetDetail BAPI. Check the BAPI return code, throw an exception if it is bad. Otherwise return the city for this company code.

8-3 Modify class TestCompanyCode to display the city in addition to the code and the name.

Exercises

Unit: JCO Client Programming

Topic: tRFC Clients

At the conclusion of these exercises you will be able to:

Call RFMs via tRFC.

9-1 Create a class TRfcClient.

9-2 Call RFM STFC_WRITE_TO_TCPIC via tRFC using data that includes your group number.

- 9-2-1 Create a TID.
- 9-2-2 Call the RFM with the TID.
- 9-2-3 Confirm the TID.
- 9-2-4 Print the TID.

9-3 Check table TCPIC in SAP using SE16 to verify that your data was added.

Debugging and Tracing: Contents

- Java Program
- SAP

Debugging and Tracing: Unit Objectives

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

- Use debugging and tracing.

Debugging Client

- RFC error log
  - Created automatically
  - dev_trc.trc
- RFC trace
  - Environment variable RFC_TRACE = 0 or 1
  - **rfc**nnnnn_nnnnn**.trc**
- RFC trace directory
  - Environment variable RFC_TRACE_DIR
- Complete table trace (uncompressed)
  - Environment variable RFC_TRACE_DUMP = 0 or 1

Debugging Server

- RFC error log and trace: ST11
- Debug ABAP code
  - Requires SAP-GUI and some ABAP Workbench knowledge

JCO.Client Trace/Debug Methods

- Is trace enabled?

```
boolean getTrace()
```

- Enable trace

```
void setTrace(boolean trace)
```

  - The trace must be turned on *before* connect().

- Is ABAP debugging enabled?

```
boolean getAbapDebug()
```

- Enable ABAP debugging

```
void setAbapDebug(boolean debug)
```

Dump Data to HTML

- JCO.Record method
  - `void writeHTML(java.lang.String html_filename)`
- Can be used for parameter lists, structures, and tables.
  - Big tables are not shown completely.

Debugging and Tracing: Unit Summary

- You can use traces and debugging both on the client and the server side.

Helpvalues And BapiService: Contents

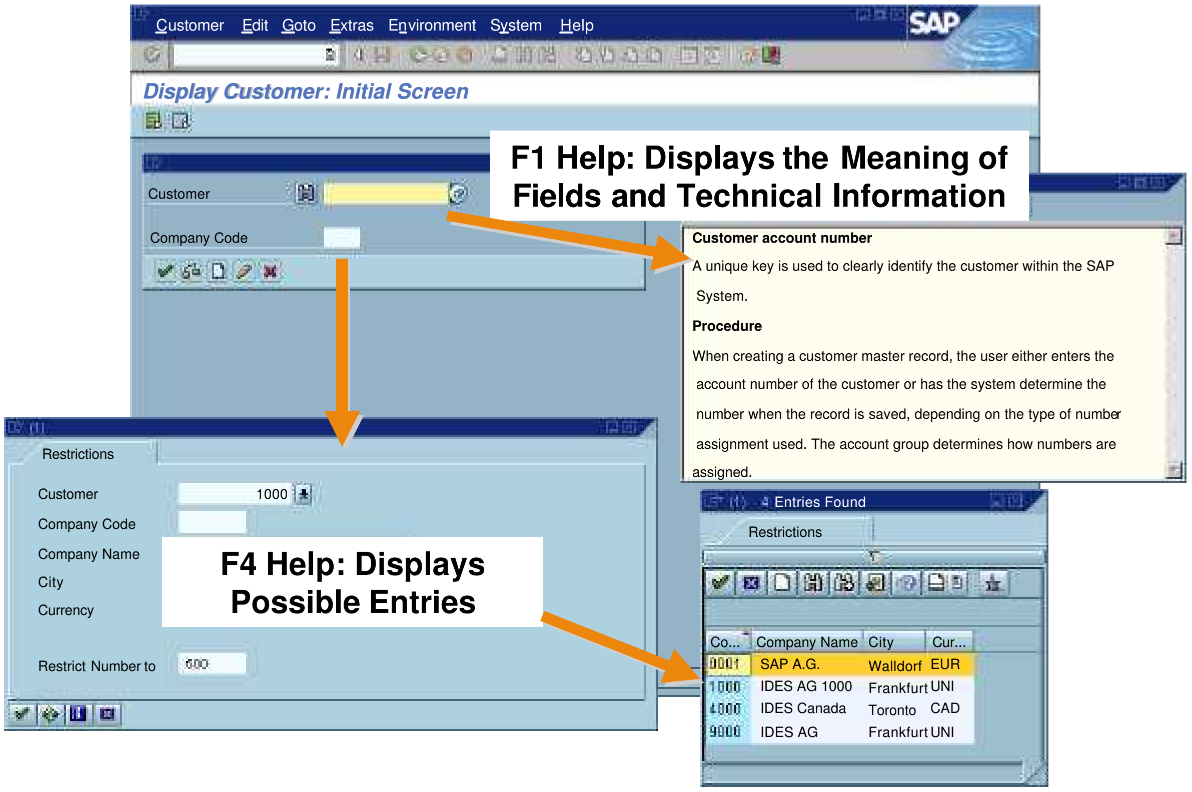
- Introduction to Helpvalues
- Helpvalues API
- Transaction Handling
- Introduction to BapiService
- BapiService API

Helpvalues And BapiService: Unit Objectives

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

- Use the Helpvalues and BapiService object types.

Field Help: F1 and F4


Customer Edit Goto Extras Environment System Help
Display Customer: Initial Screen
Customer
Company Code
F1 Help: Displays the Meaning of Fields and Technical Information
F4 Help: Displays Possible Entries
Restrictions
4 Entries Found

Customer account number

A unique key is used to clearly identify the customer within the SAP System.

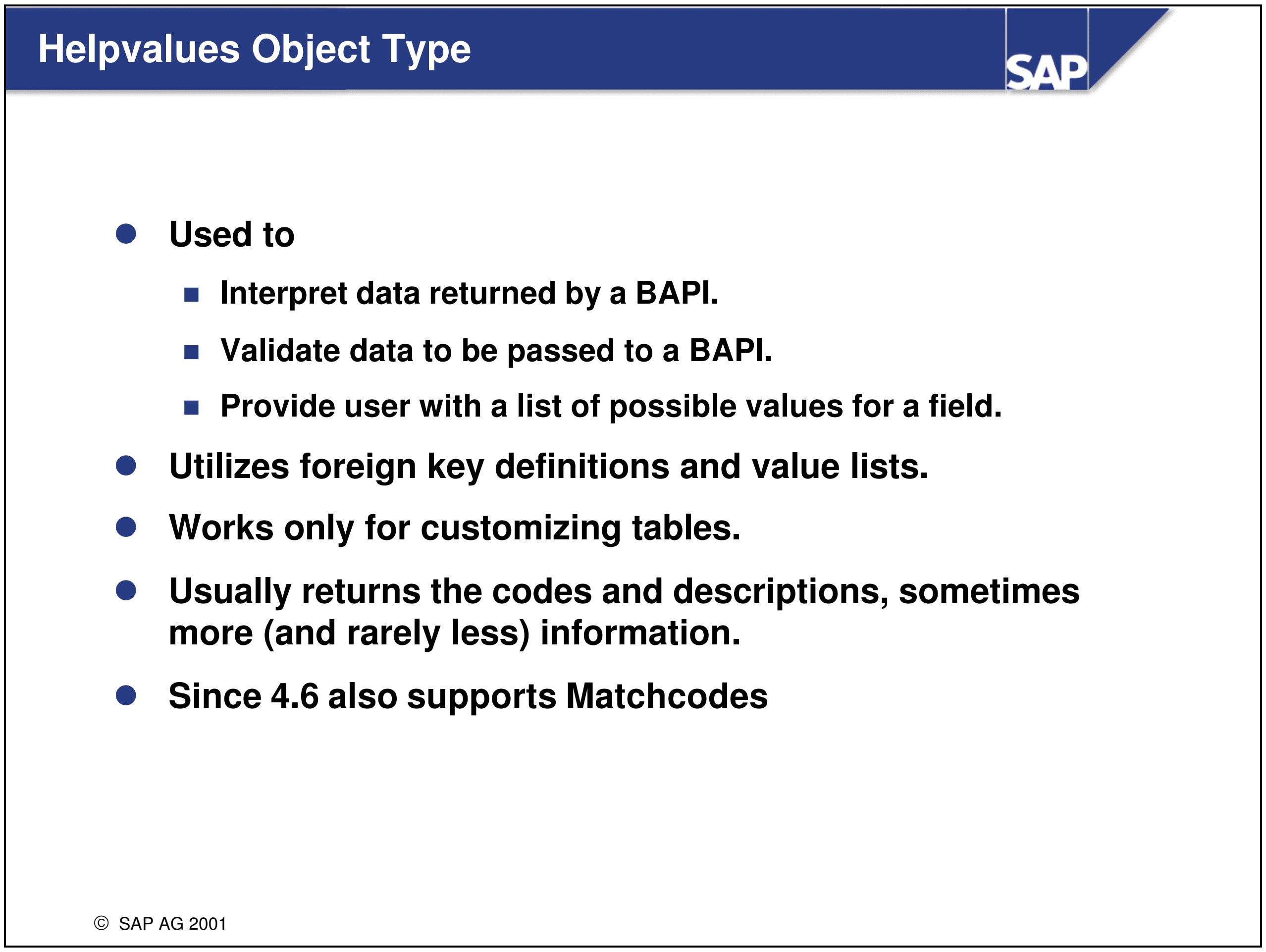
Procedure

When creating a customer master record, the user either enters the account number of the customer or has the system determine the number when the record is saved, depending on the type of number assignment used. The account group determines how numbers are assigned.

Customer	1000
Company Code	
Company Name	
City	
Currency	
Restrict Number to	500

Co...	Company Name	City	Cur...
0001	SAP A.G.	Walldorf	EUR
1000	IDES AG 1000	Frankfurt	UNI
4000	IDES Canada	Toronto	CAD
9000	IDES AG	Frankfurt	UNI

- For help on fields, menus, functions, and messages, use **F1**.
- F1 help also provides technical information on the relevant field. This includes, for example, the parameter ID, which you can use to assign values for your user to input fields , which have to refer to these parameter IDs.
- For information on what values you can enter, use **F4**. You can also access F4 help for a selected field using the button immediately to the right of that field.
- If input fields are marked with a small icon with a checkmark, then you can only continue in that application by entering a permitted value. You can mark many fields in an application as either required entry fields or optional entry fields. You can also hide fields and preassign values using transaction or screen variants or Customizing.



Helpvalues Object Type

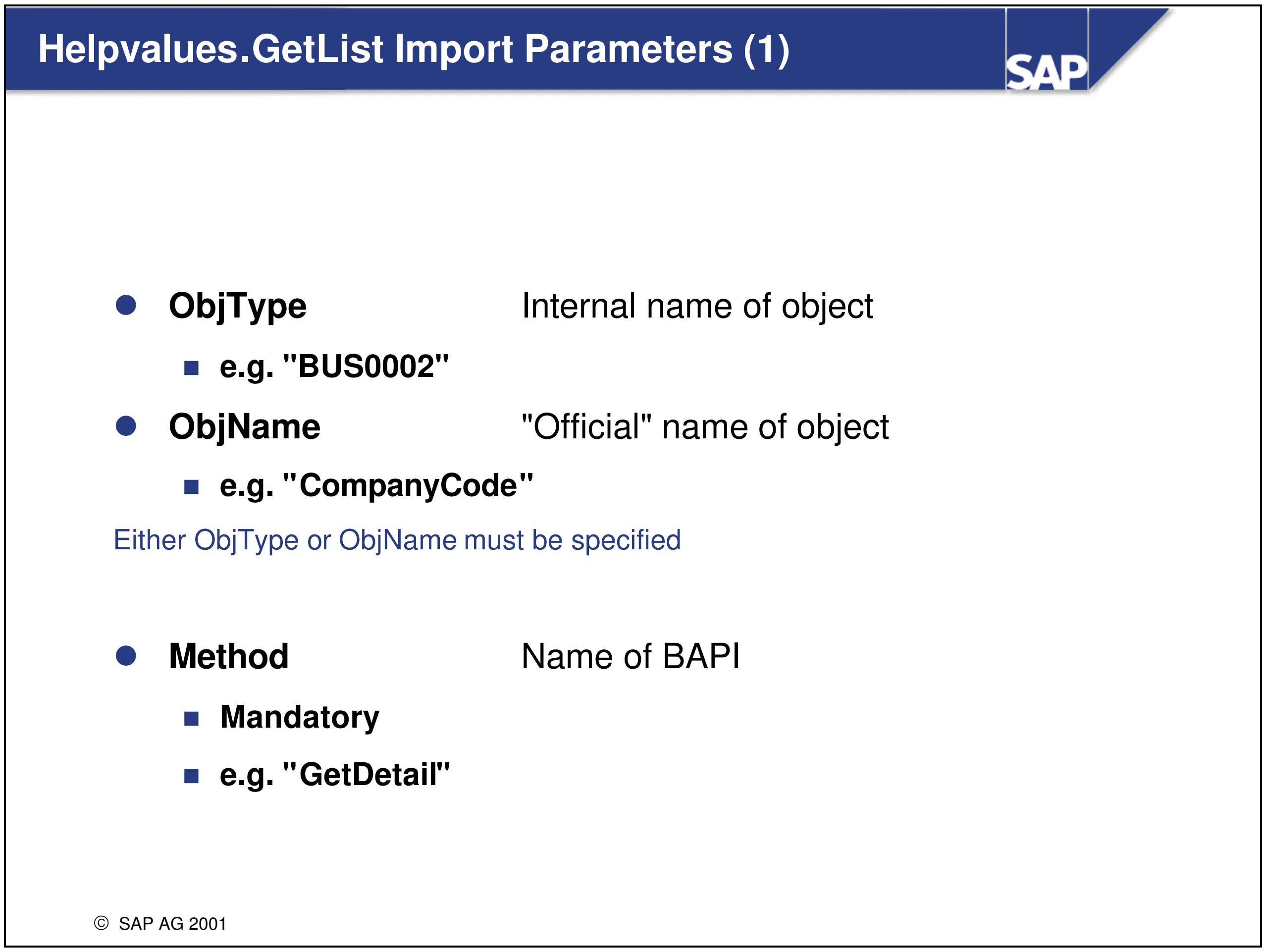
- Used to
  - Interpret data returned by a BAPI.
  - Validate data to be passed to a BAPI.
  - Provide user with a list of possible values for a field.
- Utilizes foreign key definitions and value lists.
- Works only for customizing tables.
- Usually returns the codes and descriptions, sometimes more (and rarely less) information.
- Since 4.6 also supports Matchcodes

- This object can be used to provide lookups via comboboxes or list lookup controls.
- Helpvalues is a service object that enables the display of possible entries to assist data entry and lookups.
- The Helpvalues object is used to interpret or validate data passed to a BAPI or returned by a get/read BAPI.
- The Helpvalues object can be found in the BOR in the cross application and Business Framework Architecture hierarchy.
- Helpvalues uses metadata and retrieves key values, for example a list of material numbers.

Helpvalues: GetList Method

- Retrieves all or selected values for an entity.
- Dynamically determines view structure.
- Some entities depend on other entities
  - e.g. Distribution Channel depends on Sales Organization.

- The GetList BAPI determines the allowed input values (the F4 key in R/3). For fields in BAPI parameters. This method returns the permitted input values for a field which you pass in a BAPI call.
- The input values are stored in corresponding check tables in the R/3 system. Check tables are used in R/3 for consistency and lookup activites.
- This method can be used to narrow down choices based on selection criteria at runtime.



Helpvalues.GetList Import Parameters (1)

● **ObjType**	Internal name of object
■ **e.g. "BUS0002"**	
● **ObjName**	"Official" name of object
■ **e.g. "CompanyCode"**	

Either ObjType or ObjName must be specified

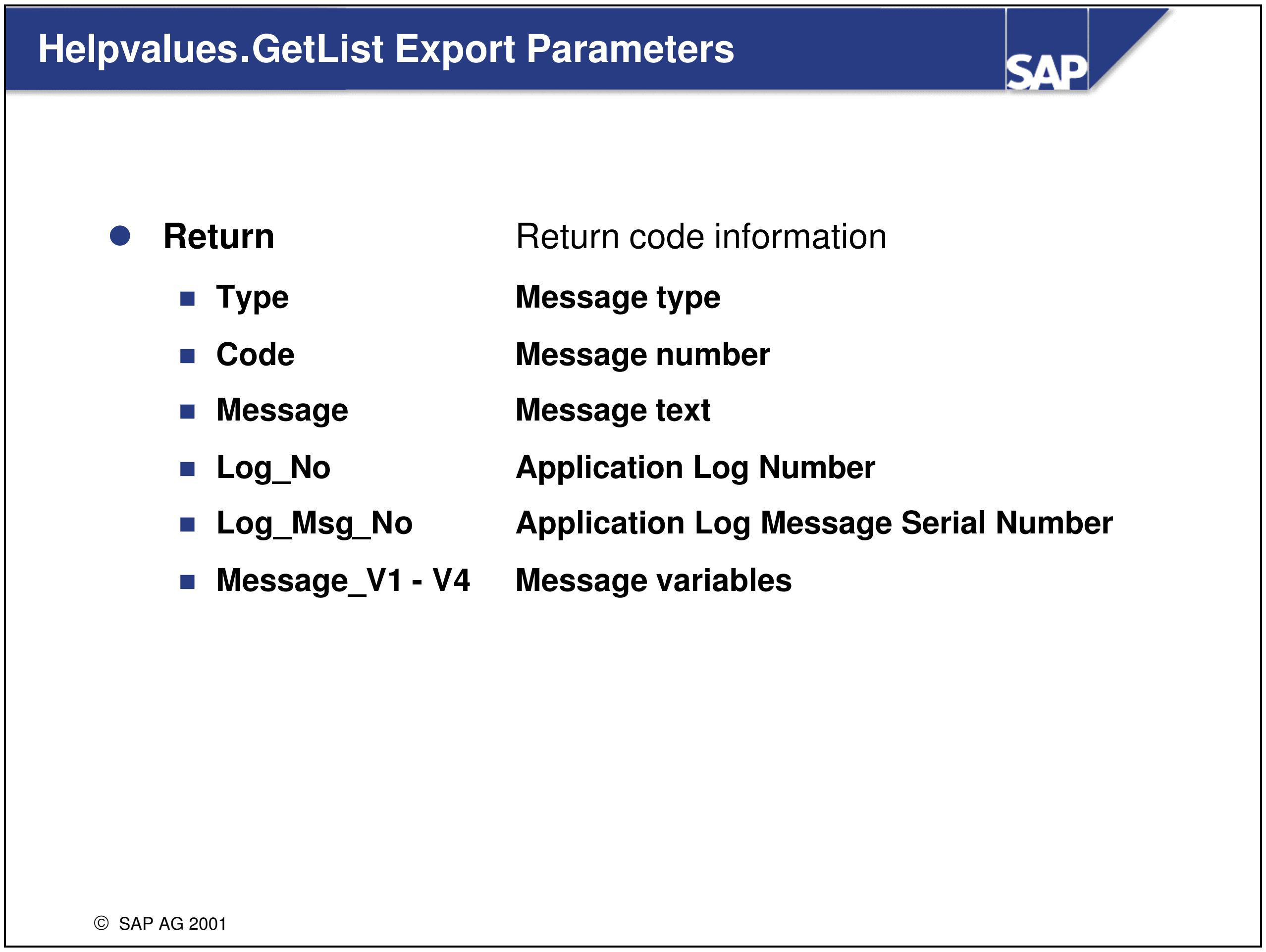
● **Method**	Name of BAPI
■ **Mandatory**	
■ **e.g. "GetDetail"**	

- The GetList BAPI can take either the Object’s internal R/3 name, ObjType or the Object’s externally used name, ObjName. The name must be exact or the method invocation will not work.

Helpvalues.GetList Import Parameters (2)

● **Parameter**	Name of parameter
■ **Mandatory**	
■ **e.g. "CompanyCodeDetail"**	
● **Field**	Field name in structure or table
■ **Not used for scalar parameters, mandatory otherwise.**	
■ **e.g. "Country"**	
● **MaxOfRows**	Maximum no. of rows returned
■ **Default: All rows**	
● **DescriptionOnly**	No values, just metadata

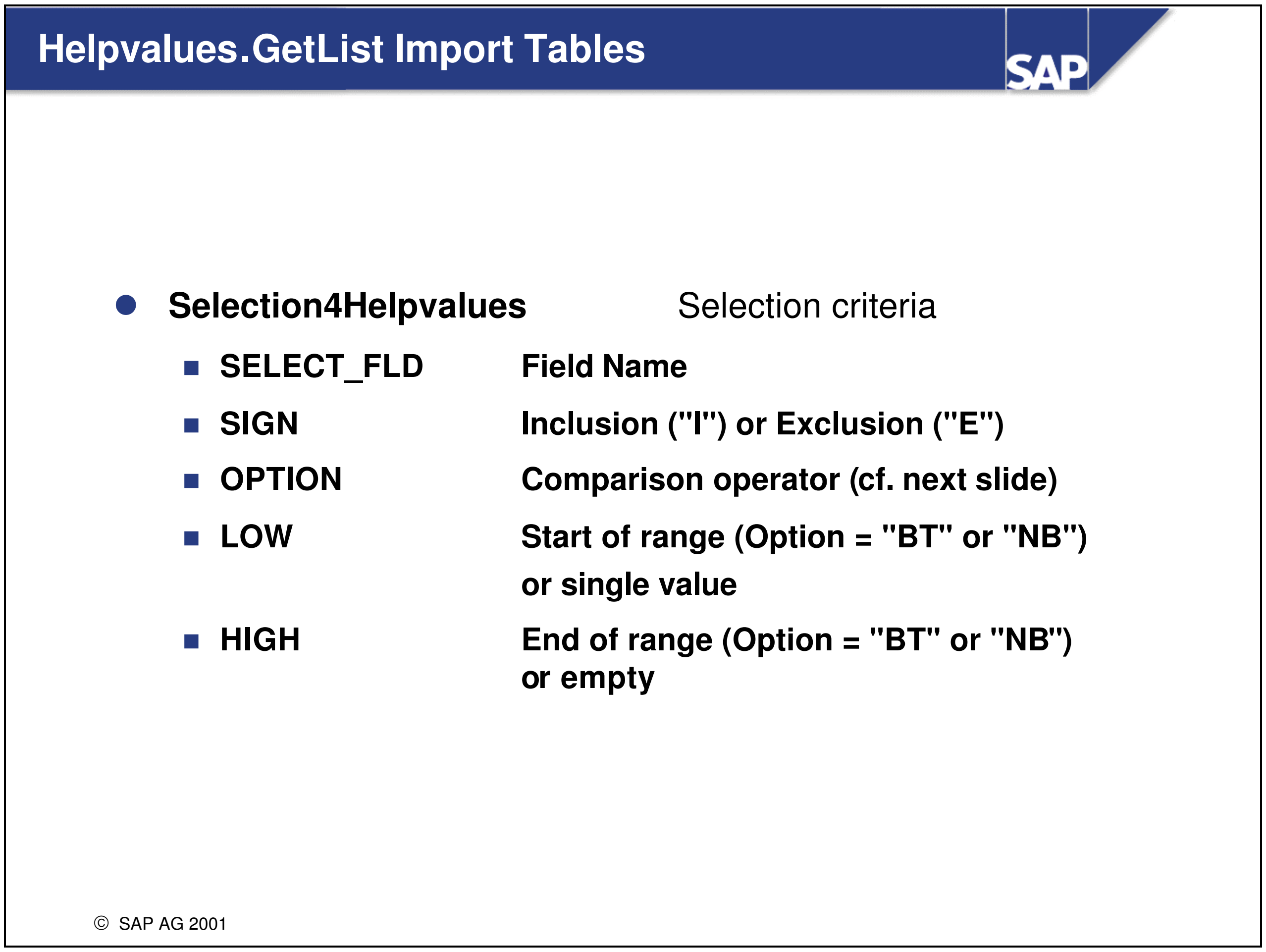
- The Helpvalues GetList BAPI requires that the name of the parameter be passed. If the parameter is a structure or table, the field must be specified.



Helpvalues.GetList Export Parameters

● **Return**	Return code information
■ **Type**	**Message type**
■ **Code**	**Message number**
■ **Message**	**Message text**
■ **Log_No**	**Application Log Number**
■ **Log_Msg_No**	**Application Log Message Serial Number**
■ **Message_V1 - V4**	**Message variables**

■ A nonblank Return Code indicates an exception has occurred.



Helpvalues.GetList Import Tables

- **Selection4Helpvalues** Selection criteria

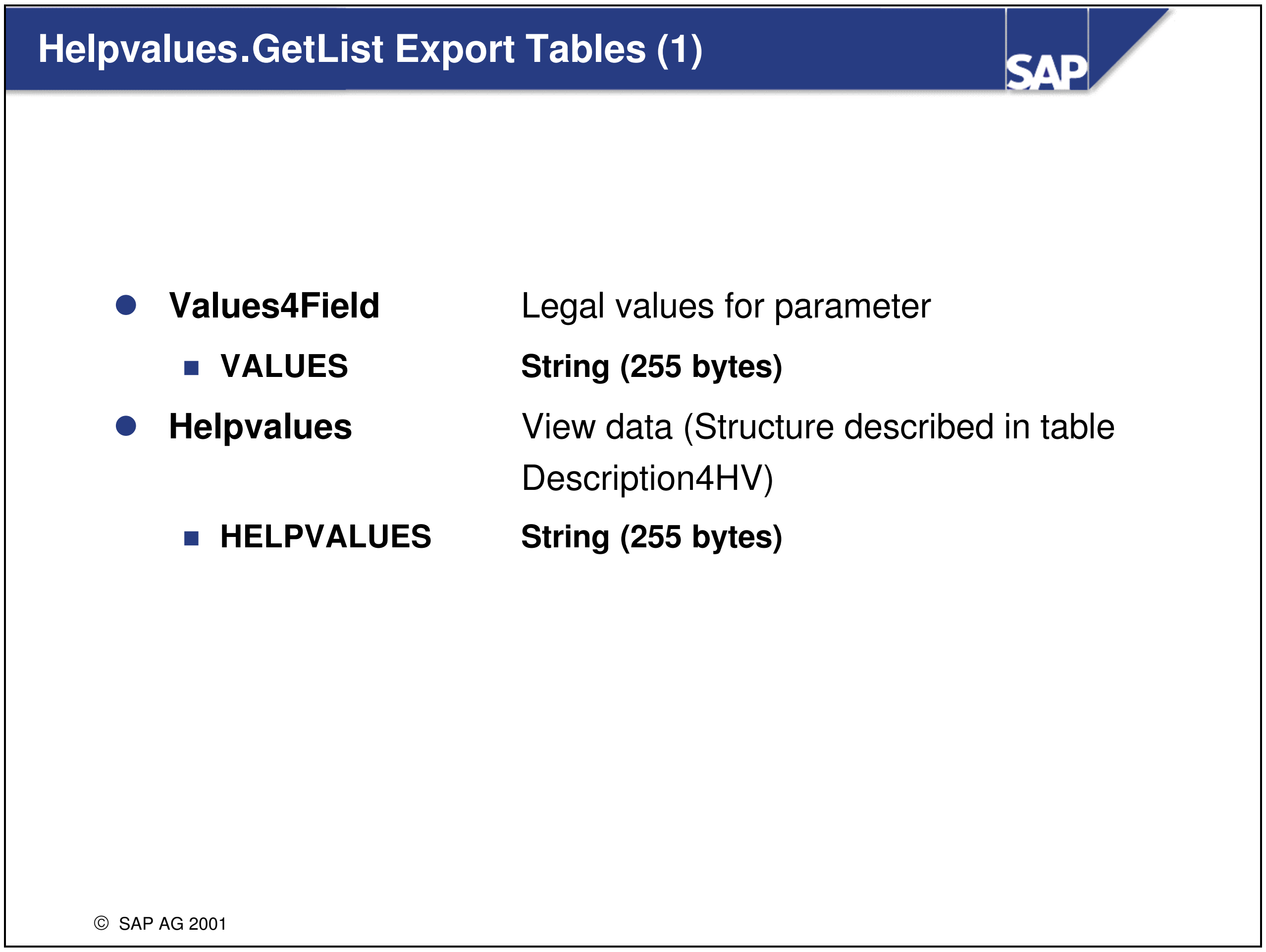
SELECT_FLD	**Field Name**
SIGN	**Inclusion ("I") or Exclusion ("E")**
OPTION	**Comparison operator (cf. next slide)**
LOW	**Start of range (Option = "BT" or "NB") or single value**
HIGH	**End of range (Option = "BT" or "NB") or empty**

- The Selection4Helpvalues parameter of the GetList method is similar to a SQL query except the SQL is being performed as part of a BAPI method invocation.

Comparison Operators

EQ	Equal
NE	Not equal
GT	Greater than
LT	Less than
LE	Less or equal
CP	Containing pattern
NP	Not containing pattern
BT	Between (for ranges)
NB	Not between (for ranges)

- These comparison operators can be used as valid entries in the Option field in the Selection4HelpValues table parameter.



Helpvalues.GetList Export Tables (1)

- **Values4Field** — Legal values for parameter
  - **VALUES** — **String (255 bytes)**
- **Helpvalues** — View data (Structure described in table Description4HV)
  - **HELPVALUES** — **String (255 bytes)**

- The Values4Field contains the valid values for the specified field (for example all the Country Codes). The values are exported from R/3 in a String structure (255 bytes). The string must be parsed.
- To assist in identifying the values, the HelpValues parameter provides information about the structure of the values.

Helpvalues.GetList Export Tables (2)

● **Description4HV**	Defines structure of table HelpValues
▪ **TABNAME**	**Table name**
▪ **FIELDNAME**	**Field name**
▪ **LANGU**	**Language code**
▪ **POSITION**	**Relative position of field**
▪ **OFFSET**	**Field offset in table HelpValues**
▪ **LENG**	**Field length**
▪ **FIELDTEXT**	**Description text for field**
▪ **REPTEXT**	**Report header**
▪ **SCRTEXT_S**	**Short screen keyword**
▪ **SCRTEXT_M**	**Medium screen keyword**
▪ **SCRTEXT_L**	**Long screen keyword**

- The Description4HV export table is returned by R/3 with the structure definition of the table HelpValues.

Helpvalues: Programming Hints

- The object name is "Helpvalues", not "HelpValues"
- Field names can only be determined by calling the method/function
- Retrieve data for each entity only once
- Consider local storage
- Encapsulate as a JavaBeans component
- Build Swing components
- More information in the SAP Technical Journal, Volume 1, Number 1: http://www.intelligenterp.com/columns/archive/bapi1.shtml

Matchcode Support in 4.6

- Helpvalues now has two BAPIs
  - GetList (since 3.1)
  - GetSearchhelp (new in 4.6)
- Helpvalues now supports Matchcodes
  - Only entities with authorization handling supported

Helpvalues: Matchcode Support

- Authorization checks as defined in table BAPIF4T
  - Column FNAM specifies which function is called for the authority check
  - Customers, Suppliers, and Partners supported
  - Table can be expanded

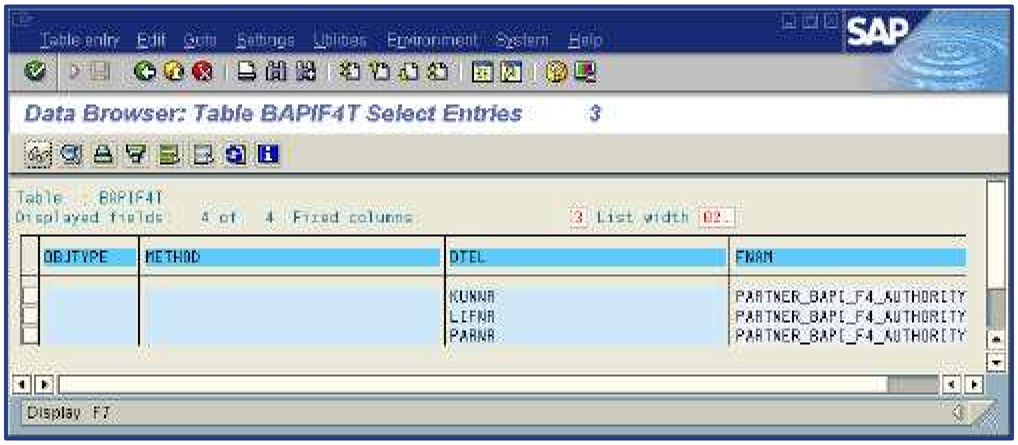


Table entry Edit Goto Settings Utilities Environment System Help
SAP
Display F7

Data Browser: Table BAPIF4T Select Entries 3

```
Table           BAPIF4T
Displayed fields    4 of   4  Fixed columns              3  List width 02
```

OBJTYPE	METHOD	DTEL	FNAM
		KUNNR	PARTNER_BAPI_F4_AUTHORITY
		LIFNR	PARTNER_BAPI_F4_AUTHORITY
		PARNR	PARTNER_BAPI_F4_AUTHORITY

Helpvalues.GetSearchhelp

- **Parameters**
  - **Objname, Method, Parameter, Field**
    Used as in Helpvalues.GetList to identify the entity
  - **Return**
    Return structure (BAPIRETURN)
  - **ShlpForHelpvaluesGet**
    List of all Search Helps

SHLPNAME	Internal code
SHLPTYPE	Type (usually "SH")
TITLE	Description
REPTEXT	Report header text

Helpvalues.GetList

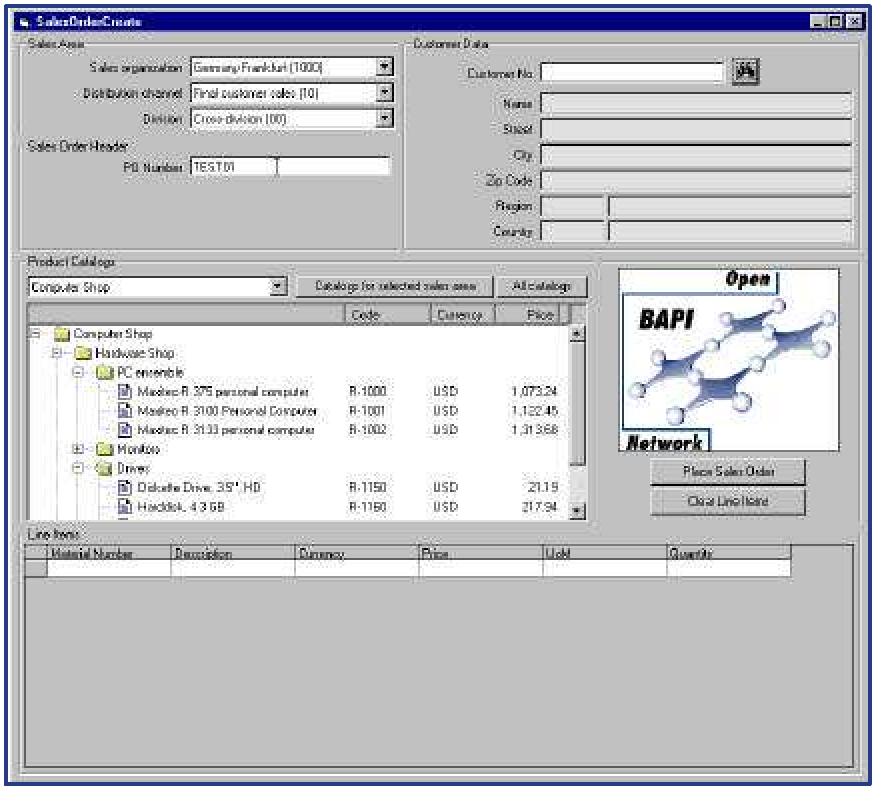
- **New parameter**
  - **ExplicitShlp**
    Allows one Search Help to be selected

◆ **SHLPNAME**	Internal code
◆ **SHLPTYPE**	Type (usually “SH”)
◆ **TITLE**	Description
◆ **REPTEXT**	Report header text

Matchcodes Step-by-Step

- Call Helpvalues.GetSearchhelp to retrieve a list of all Matchcodes for an entity.
- Display the list.
- Let the user select a Matchcode.
- Call Helpvalues.GetList for this Matchcode, specifying MaxOfRows=1.
- Build a selection screen with all the fields in table Description4HV.
- Let the user enter selection criteria.
- Call Helpvalues.GetList with all the selection criteria entered into table Selection4Helpvalues.
- Display the list of matching entries separated into columns according to Description4HV.
- Let the user select an entry.

Selection via Matchcodes



Sales Area

Sales organization	Germany Frankfurt (1000)
Distribution channel	Final customer sales (10)
Division	Cross-division (00)

Sales Order Header

PO Number	TEST01

Customer Data

Customer No.	
Name	
Street	
City	
Zip Code	
Region	
Country	

Product Catalogs

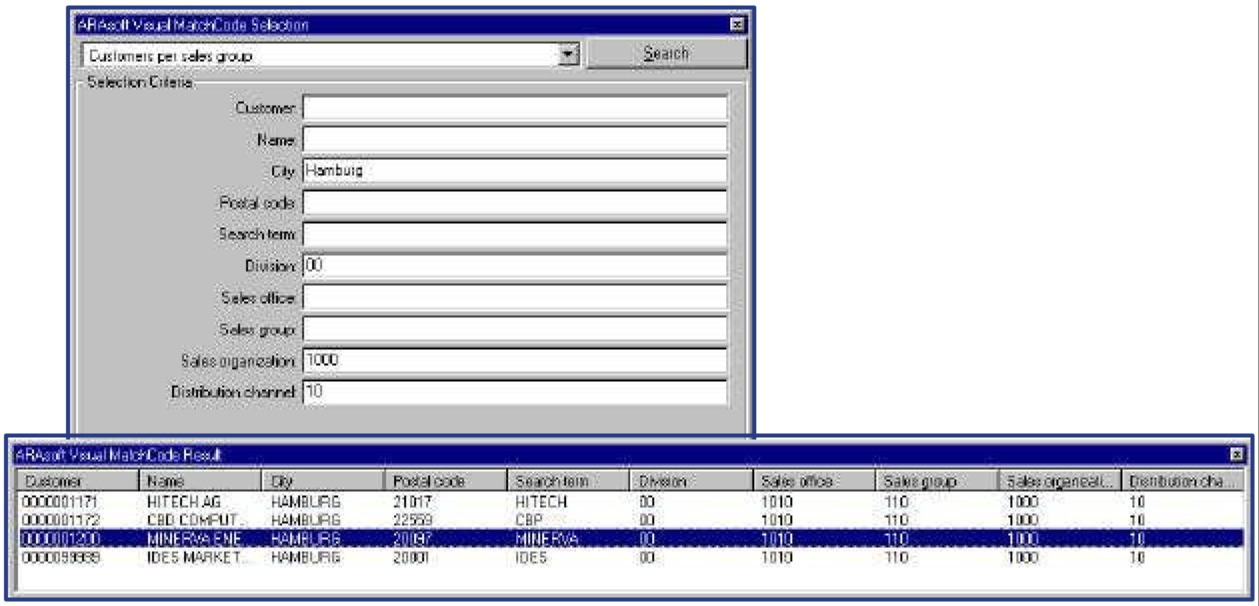
Computer Shop

	Code	Currency	Price
Computer Shop			
Hardware Shop			
PC ensemble			
Maxitec-R 375 personal computer	R-1000	USD	1,073.24
Maxitec-R 3100 Personal Computer	R-1001	USD	1,122.45
Maxitec R 3133 personal computer	R-1002	USD	1,313.68
Monitors			
Drives			
Diskette Drive, 3.5", HD	R-1150	USD	21.19
Harddisk, 4.3 GB	R-1160	USD	217.94

Line Items

Material Number	Description	Currency	Price	UoM	Quantity

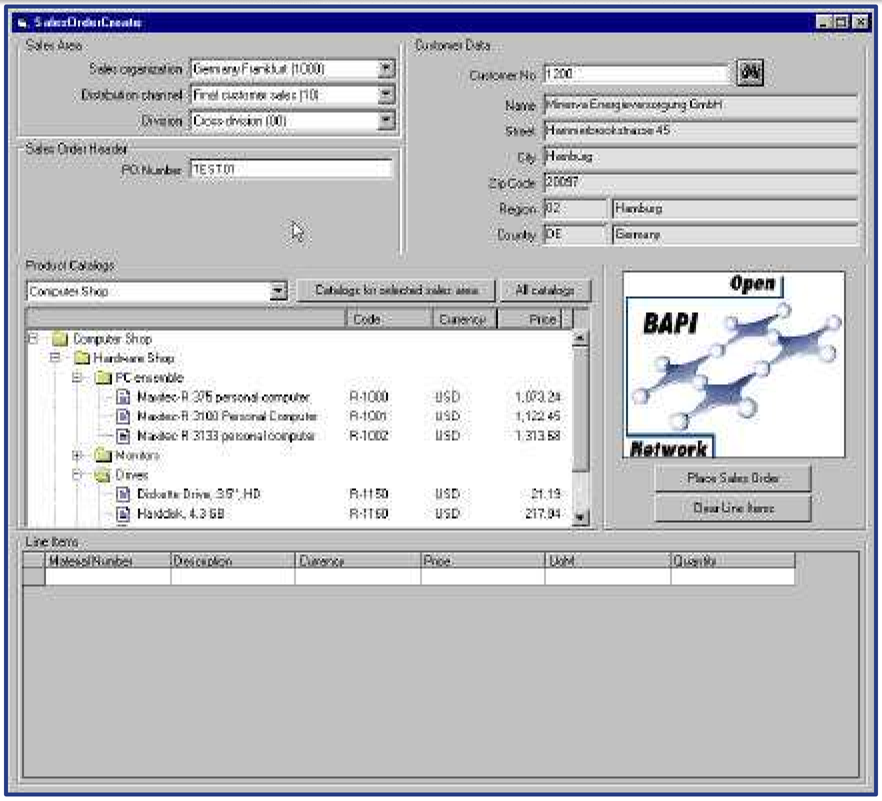
Selection via Matchcodes


ARAsoft Visual MatchCode Selection
Customers per sales group
Search
Selection Criteria
ARAsoft Visual MatchCode Result

Customer:	
Name:	
City:	Hamburg
Postal code:	
Search term:	
Division:	00
Sales office:	
Sales group:	
Sales organization:	1000
Distribution channel:	10

Customer	Name	City	Postal code	Search term	Division	Sales office	Sales group	Sales organizati...	Distribution cha...
0000001171	HITECH AG	HAMBURG	21017	HITECH	00	1010	110	1000	10
0000001172	CBD COMPUT...	HAMBURG	22559	CBP	00	1010	110	1000	10
0000001200	MINERVA ENE...	HAMBURG	20097	MINERVA	00	1010	110	1000	10
0000099999	IDES MARKET...	HAMBURG	20001	IDES	00	1010	110	1000	10

Selection via Matchcodes



SalesOrderCreate

Sales Area

Sales organization: Germany Frankfurt (1000)

Distribution channel: Final customer sales (10)

Division: Cross division (00)

Sales Order Header

PO Number: TEST01

Customer Data

Customer No: 1200

Name: Minerva Energieversorgung GmbH

Street: Hammerbrookstrasse 45

City: Hamburg

Zip Code: 20097

Region: 02 Hamburg

Country: DE Germany

Product Catalogs

Computer Shop

	Code	Currency	Price
Computer Shop			
Hardware Shop			
PC ensemble			
Maxitec-R 375 personal computer	R-1000	USD	1,073.24
Maxitec-R 3100 Personal Computer	R-1001	USD	1,122.45
Maxitec R 3133 personal computer	R-1002	USD	1,313.58
Monitors			
Drives			
Diskette Drive, 3.5", HD	R-1150	USD	21.19
Harddisk, 4.3 GB	R-1160	USD	217.94

Line Items

Material Number	Description	Currency	Price	UoM	Quantity

Matchcodes: Programming Hints

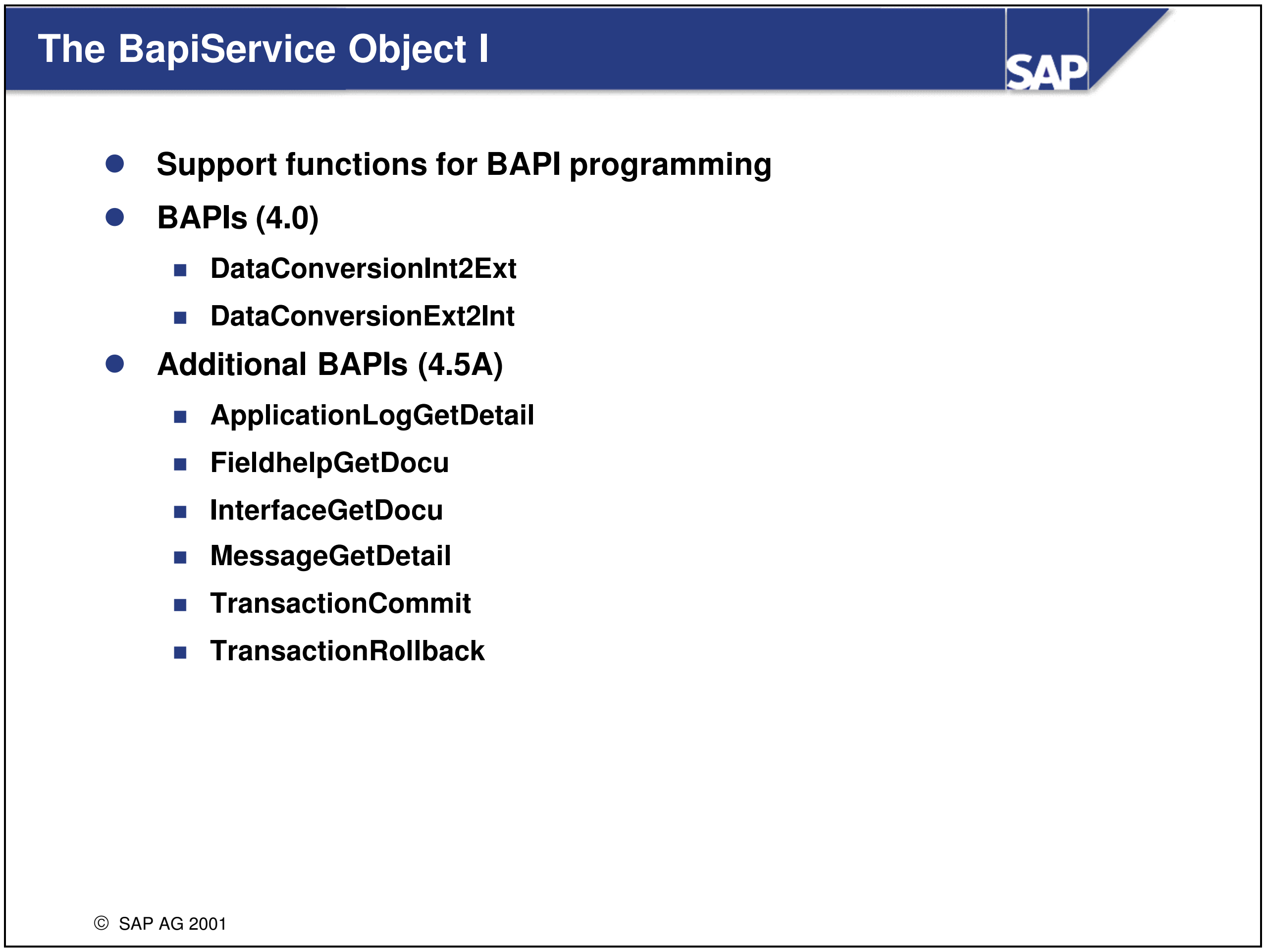
- Encapsulate all non-visual activities in a JavaBeans component.
- Encapsulate all user interface elements in an
  - Visual JavaBean

Transaction Handling

- All 3.1 update BAPIs execute their own COMMIT WORK.
- BAPIs since 4.0 are not supposed to do this.
- Double-check which category the BAPIs you want to use fall into.
  - There is no metadata attribute for this
  - Read the documentation
  - OSS Note 131838 should list all BAPIs that execute their own COMMIT WORK
  - If in doubt, test...
- Be careful when using BAPIs from both categories in the same session.
- Your own BAPIs must use the Update Task to support the new concept.

Commit/Rollback BAPIs

- Supported starting with 4.5 by the BapiService methods
  - TransactionCommit
    - WAIT parameter allows the client program to wait for the completion of the Update Task.
    - Return parameter (filled only when WAIT = "X")
  - TransactionRollback
- Supported starting with 4.0 by the RFC-enabled functions modules
  - BAPI_TRANSACTION_COMMIT
  - BAPI_TRANSACTION_ROLLBACK



The BapiService Object I

- Support functions for BAPI programming
- BAPIs (4.0)
  - DataConversionInt2Ext
  - DataConversionExt2Int
- Additional BAPIs (4.5A)
  - ApplicationLogGetDetail
  - FieldhelpGetDocu
  - InterfaceGetDocu
  - MessageGetDetail
  - TransactionCommit
  - TransactionRollback

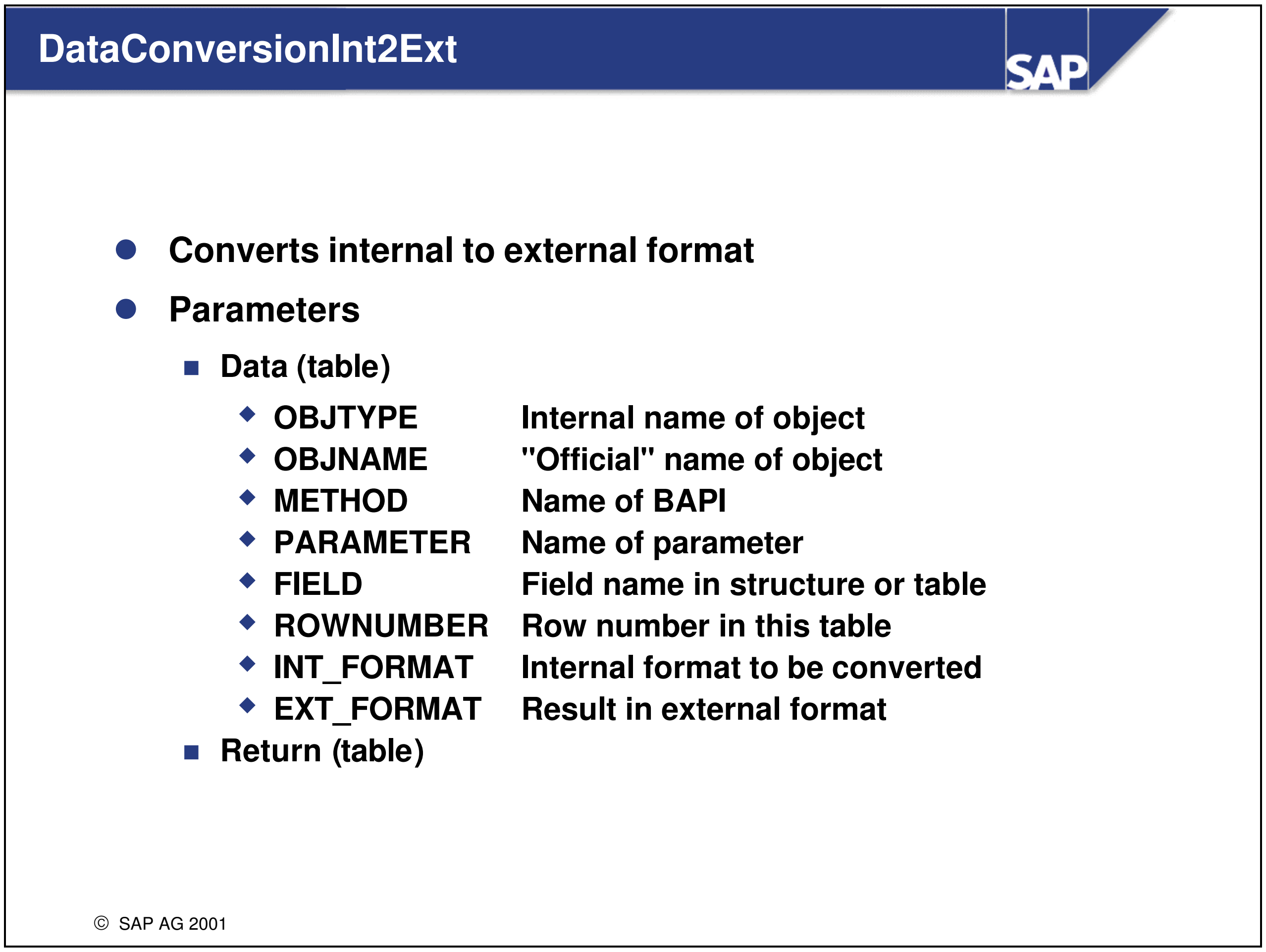
- This object is a service object whose methods provide basic BAPI services such as data conversion.
- The BapiService Object can be found in the BOR in the cross applications hierarchy within the Business Framework Architecture.

The BapiService Object II

- Additional BAPIs (4.5B)
  - DataConversionExt2int1 (sic!)
  - DataConversionInt2Ext1
- Additional BAPIs (4.6)
  - HyperLinkGetText
- Renamed BAPIs (4.6)
  - DataConversionExt2Int1 (sic!)

Conversion BAPIs

- BAPIs often deliver and require the internal SAP data format.
- Examples:
  - Customer numbers
  - Material numbers
  - Language-dependent codes
    - Units of measure etc.
- Check the conversion exit field of a field's domain to determine whether conversion is required.



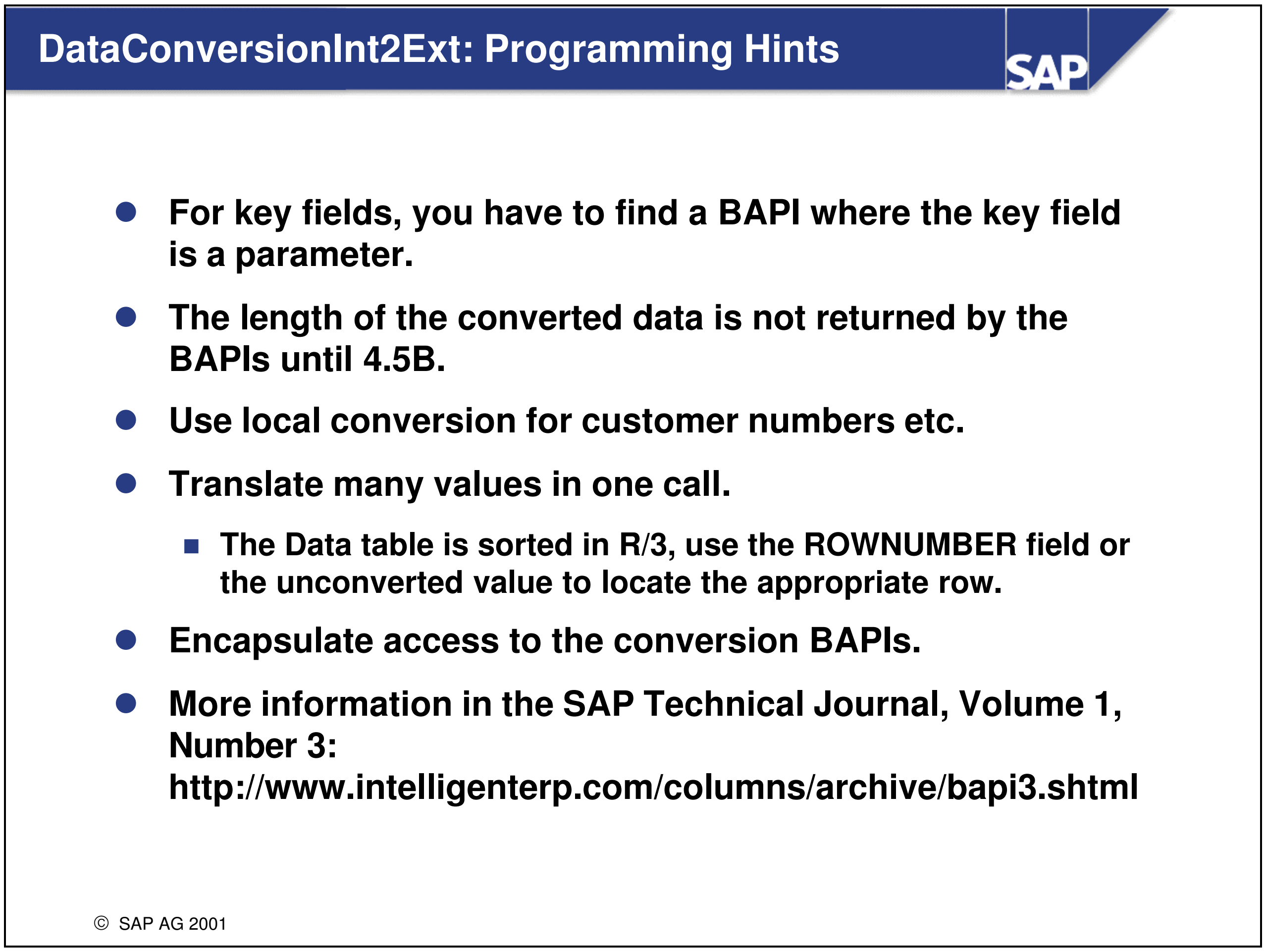
DataConversionInt2Ext

- Converts internal to external format
- Parameters
  - Data (table)

OBJTYPE	Internal name of object
OBJNAME	"Official" name of object
METHOD	Name of BAPI
PARAMETER	Name of parameter
FIELD	Field name in structure or table
ROWNUMBER	Row number in this table
INT_FORMAT	Internal format to be converted
EXT_FORMAT	Result in external format

  - Return (table)

- This BAPI could be used in an external application to convert the SAP employee number 00010018 to 10018 for a JAVA application. Some conversions of internal data are not intuitive and this BAPI can be useful when type mapping, field filling, and padding are not obvious.
- The Return values are only exported to the calling application if errors have occurred.



DataConversionInt2Ext: Programming Hints

- For key fields, you have to find a BAPI where the key field is a parameter.
- The length of the converted data is not returned by the BAPIs until 4.5B.
- Use local conversion for customer numbers etc.
- Translate many values in one call.
  - The Data table is sorted in R/3, use the ROWNUMBER field or the unconverted value to locate the appropriate row.
- Encapsulate access to the conversion BAPIs.
- More information in the SAP Technical Journal, Volume 1, Number 3: http://www.intelligenterp.com/columns/archive/bapi3.shtml

- This BAPI returns a 255 character structure so the structure must be trimmed.

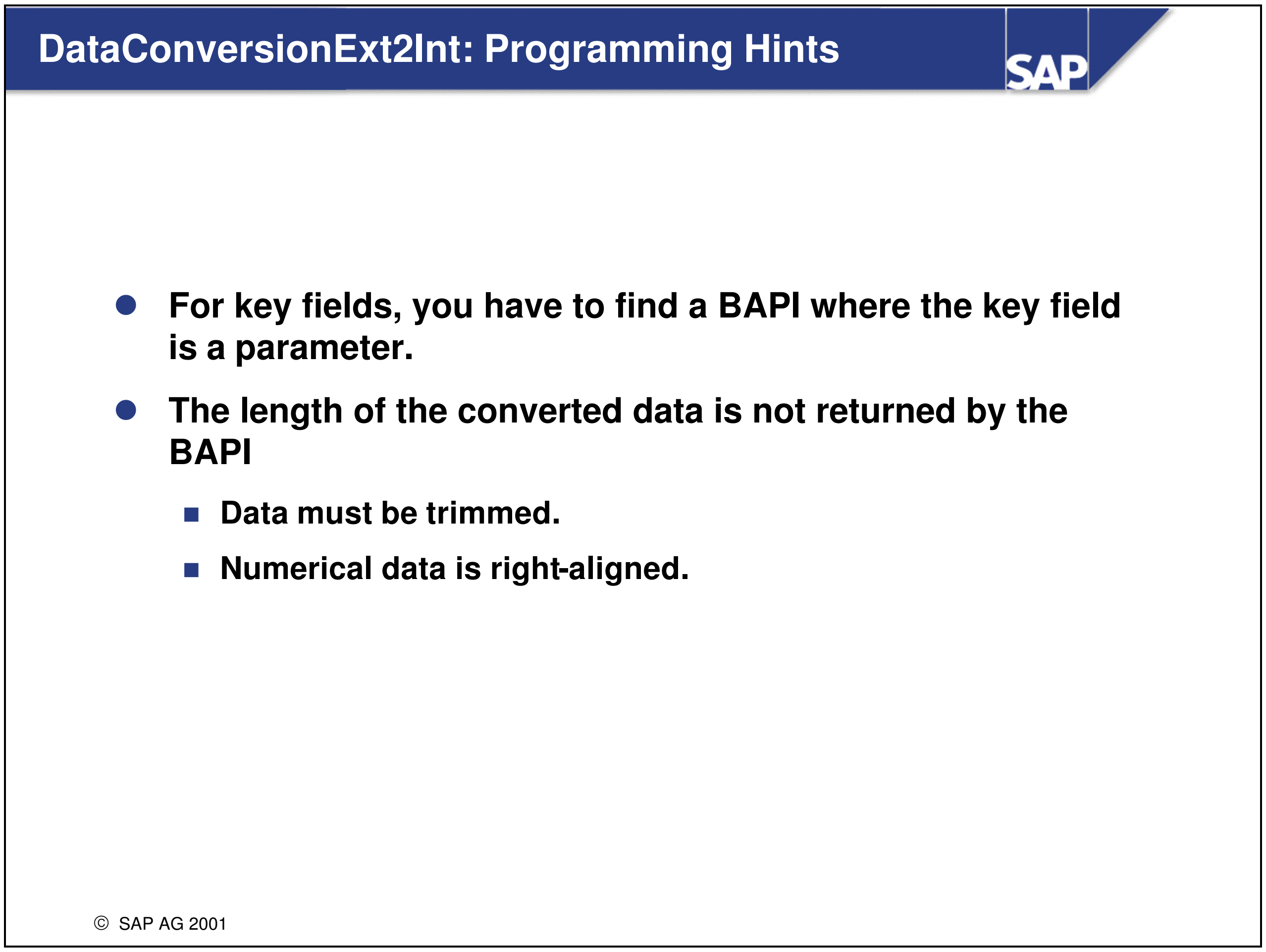
DataConversionExt2Int

- Converts external to internal format
- Parameters
  - Data (table)

OBJTYPE	Internal name of object
OBJNAME	"Official" name of object
METHOD	Name of BAPI
PARAMETER	Name of parameter
FIELD	Field name in structure or table
ROWNUMBER	Row number in this table
INT_FORMAT	Result in internal format
EXT_FORMAT	External format to be converted

  - Return (table)

- This BAPI performs a conversion from R/3 to an external format. For example Employee 10018 can be converted from a JAVA application to 00010018 for R/3.



DataConversionExt2Int: Programming Hints

- For key fields, you have to find a BAPI where the key field is a parameter.
- The length of the converted data is not returned by the BAPI
  - Data must be trimmed.
  - Numerical data is right-aligned.

- This BAPI returns a 255 character structure so the structure must be trimmed.

Helpvalues And BapiService: Unit Summary

- The Helpvalues and BapiService objects supply important support functions.

Helpvalues and BapiService Exercises

Unit: Helpvalues and BapiService

Topic: Helpvalues

At the conclusion of these exercises you will be able to:

Use Helpvalues.GetList.

1-1 Create a class SAPServices4 that extends SAPServices3 with a static method using the following signature:
```
public static SAPCountry[] getCountries(JCO.Client
connection, IRepository repository) throws Exception
```
(class SAPCountry is supplied by the instructor).

1-2 Implement this method to call Helpvalues.GetList for the country entity and return an array of all SAPCountry objects.

1-3 This step and the next one are optio nal. If you do not have enough time after completing the previous steps, simply "inherit" this method from the solution: Add a method to SAPServices4 using the following signature:
public static SAPCountry getCountry(String code, SAPCountry[] countries)

1-4 Implement this method to find a particular SAPCountry object (identified by its code) in the array. Return null if the specified code was incorrect.

1-5 Write a class called TestCountries to test your two methods.

Exercises

Unit: Helpvalues and BapiService

Topic: BapiService Conversions

At the conclusion of these exercises you will be able to:

Use the conversion BAPIs.

2-1 Create a class SAPServices5 that extends SAPServices4 with a static method using the following signature:
public static String getISOLanguageCode(JCO.Client connection, IRepository repository, String sapLanguageCode) throws Exception

2-2 Implement this method to call the appropriate conversion BAPI to convert a one-byte SAP language code to its two-byte ISO equivalent.

2-3 Write a class called TestLanguage2 to test your new method.

JCO Server Programming: Contents

- Outbound Calls in SAP
- JCO sRFC Servers
- JCO tRFC/qRFC Servers

JCO Server Programming: Unit Objectives

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

- Use JCO to build servers for outbound ABAP sRFC/tRFC/qRFC calls.

Design Goals in JCO

- Support outbound RFC calls from SAP (`CALL FUNCTION ... DESTINATION ... [ IN BACKGROUND TASK ]`)
  - Function interface may, but need not, be defined in the SAP Function Builder (SE37).
- Support multi-threading.

Prerequisites in SAP

- Create RFC destination (SM59)
  - Connection type: T (TCP/IP)
- Activation Type: Registration
  - Program ID: arbitrary name
    - Will be used on the server side.
      <same as in JCO.Server class constructor>
    - Case-sensitive
- Set Gateway options (via menu)
  - Gateway host: Host name
  - Gateway service: sapgw## [## = SAP system number]

Servers for Outbound Calls

- Multi-threading is supported by JCO.
- Implement the server as a subclass of JCO.Server.
  - Make sure your code is thread-safe.
- Override `handleRequest(JCO.Function function)`.
  - You can throw exceptions (whether they are defined in the function module interface or not).
- Start one or more instances of your server.

Server Sample Code

```
public class OutboundServer extends JCO.Server {
  public OutboundServer(String gwHost, String gwServer,
                        String programId, IRepository repository) {
    super(gwHost, gwServer, programId, repository);
  }
  protected void handleRequest(JCO.Function function) {
    if (function.getName().equals("BAPI_COMPANYCODE_GETLIST")) {
      JCO.Table codes =
        function.getTableParameterList().getTable("COMPANYCODE_LIST");
      codes.deleteAllRows();
      codes.appendRow();
      codes.setValue("XXXX", "COMP_CODE");
      codes.setValue("OutboundServer", "COMP_NAME");
    }
    else if (function.getName().equals("DDIF_FIELDINFO_GET")) {
      throw new JCO.AbapException("NOT_FOUND", "Structure not defined.");
    }
    else {
      throw new JCO.AbapException("FUNCTION_NOT_SUPPORTED",
                                  "This function is not supported.");
    }
  }
}
```

Starting A Server

- You need an instance of JCO.Repository in order to provide the metadata of all functions that you want to support.
  - Hard-code or create a Repository object with a connection to SAP.
- You can create as many server instances (threads) as you want.

Starting A Server: Sample Code

```
static final int MAX_SERVERS = 3;
IRepository repository;
OutboundServer servers[] = new OutboundServer[MAX_SERVERS];
public void startServers() {
  for (int i = 0; i < MAX_SERVERS; i++) {
    servers[i] = new OutboundServer("iwdf5020", "sapgw00", "CA926",
                                    repository);
    try {
      servers[i].start();
    }
    catch (Exception ex) {
      System.out.println("Could not start server !\n" + ex);
    }
  }
}
public void stopServers() {
  for (int i = 0; i < MAX_SERVERS; i++) {
    servers[i].stop();
  }
}
```

JCO.ServerExceptionListener Interface

- Listen to exceptions in your servers in the starting program.
- Method to be implemented

```
public void serverExceptionOccurred(JCO.Server server,
                                    Exception ex)
```

- Add listener

```
JCO.addServerExceptionListener(this);
```

JCO.ServerStateChangedListener Interface

- Listen to changes in the state of your servers in the starting program.
- Method to be implemented

```
void serverStateChangeOccurred(JCO.Server server,
                               int old_state, int new_state)
```

  - State bits

`JCO.STATE_STOPPED`	The server has been stopped
`JCO.STATE_STARTED`	The server has been started
`JCO.STATE_LISTENING`	The server is listening for incoming requests
`JCO.STATE_BUSY`	The server is currently processing an incoming request
`JCO.STATE_TRANSACTION`	The incoming request is a transaction

- Add listener

```
JCO.addServerStateChangedListener(this);
```

Programming tRFC/qRFC Servers 1

- No difference between tRFC and qRFC in the server.

Programming tRFC/qRFC Servers 2

- JCO.Server methods for TID management
  - `boolean onCheckTID(String tid)`
    - True = Okay
    - False = Not okay (already processed, `onConfirmTID` next)

    Throw an exception if you cannot store the TID.
  - `void onCommit(String tid)`
  - `void onRollback(String tid)`
  - `void onConfirmTID(String tid)`
- Normal call sequence
  - `onCheckTID`
  - Application function
  - `onCommit`
  - `onConfirmTID`

Programming tRFC/qRFC Servers 3

- `onCheckTID`
  - **Write TID to DB**
- Application function
  - **Process function call**
  - **Raise an exception if processing not successful.**
- `onCommit`
  - **DB Commit**
- `onRollback`
  - **DB Rollback**
- `onConfirmTID`
  - **Delete TID from DB, DB Commit**

t/qRFC Management Transactions

sm58	Transactional RFC Problems
smqr	QIN Scheduler
smq1	Outbound qRFC Monitor
smq2	Inbound qRFC Monitor

JCO Server Programming: Unit Summary

- The SAP Java Connector allows you to build server components for outbound ABAP calls.

JCO Server Programming Exercises

Unit: JCO Server Programming

Topic: sRFC Server Programming

At the conclusion of these exercises you will be able to:

Build a server for outbound sRFC calls.

1-1 Define a TCP/IP destination in SM59. Call it "CA926-xx" [xx = your group number], also use "CA926-xx" as the Program ID. Do not forget to set the gateway options.

1-2 Modify file logon.properties so that property sap.programid is set to the correct Program ID.

1-3 Create a class JCO.Server subclass OutboundServer that implements the BAPI BAPI_COMPANYCODE_GETLIST. Return at least one entry in the table.

1-4 Start your server using the supplied TestOutboundServer class.

1-5 Call the BAPI from SE37. Do not forget to specify the destination (all uppercase letters).

ALE/IDoc: Contents

- Intermediate Documents (IDocs)
- Application Link Enabling (ALE)

ALE/IDoc: Unit Objectives

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

- Understand the use of ALE/IDoc for asynchronous communication between SAP and other SAP or non-SAP applications.
- For more details, please sign up for the ALE training classes.

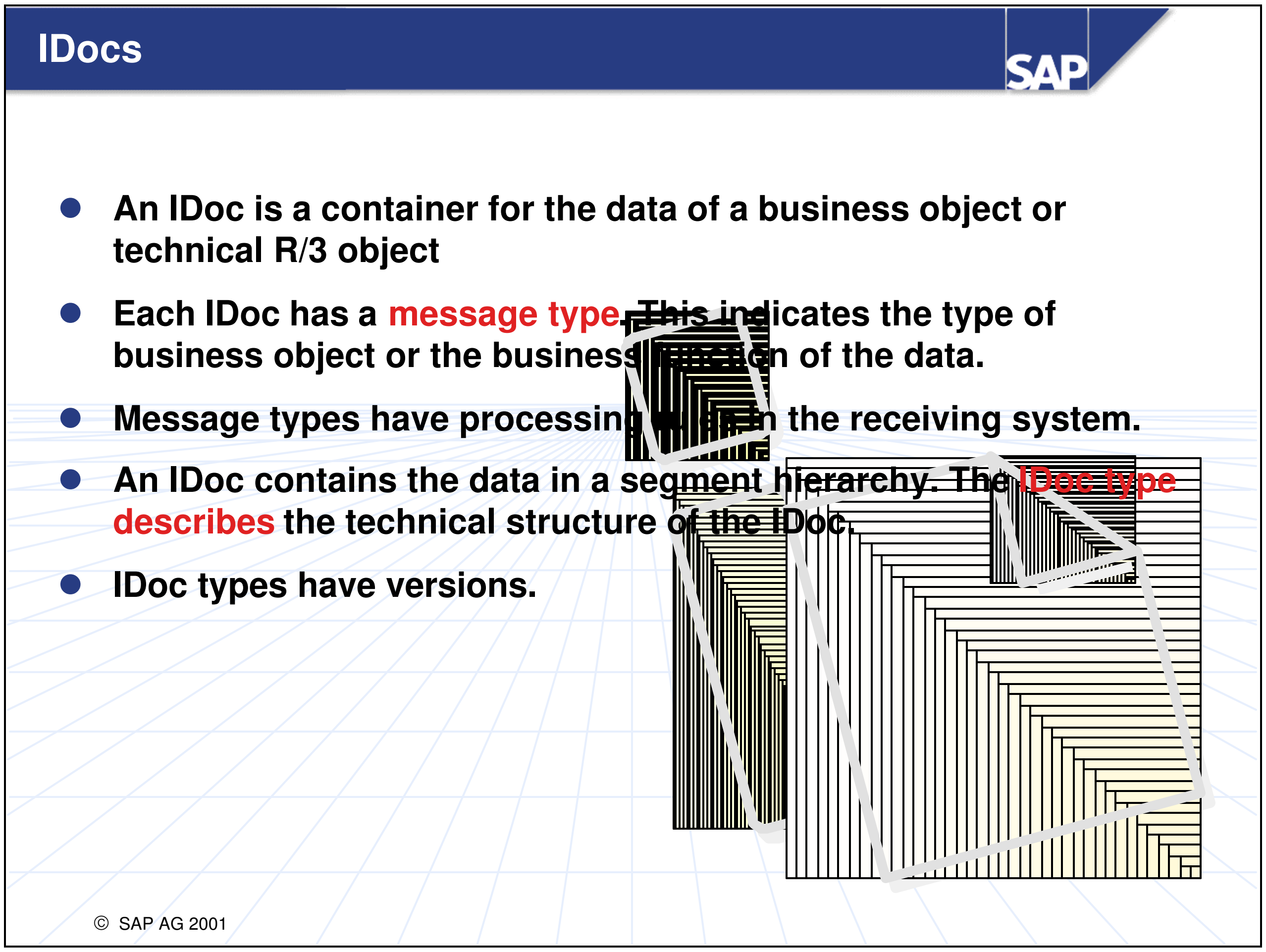
Distributed Business Processes

- Distributed applications require distributed business processes
- Available ALE Business Processes in standard
  - R/3 - R/3
  - R/3 - New Dimension Components
  - R/3 - Non-SAP
- If a scenario is not available in the standard, you can enhance existing scenarios or create new ones.
- Due to legal or technical reasons, some distributed scenarios might not be possible at all

- Information about the enhancement of existing or the creation of new scenarios are available in:
  - training class BC621
  - R/3 online documentation

    ALE Programming

    BAPI Programming
  - R/3 interface adviser
- Some scenarios are not possible to implement for legal or business reasons
  - e.g.: It makes no sense to link a logistic system to the financial system only by synchronous communication. This would disable most of the logistic transactions if the communication is down or there are problems within the financial system. The required postings within the general ledger could not be done. A scenario with asynchronous communication is required.

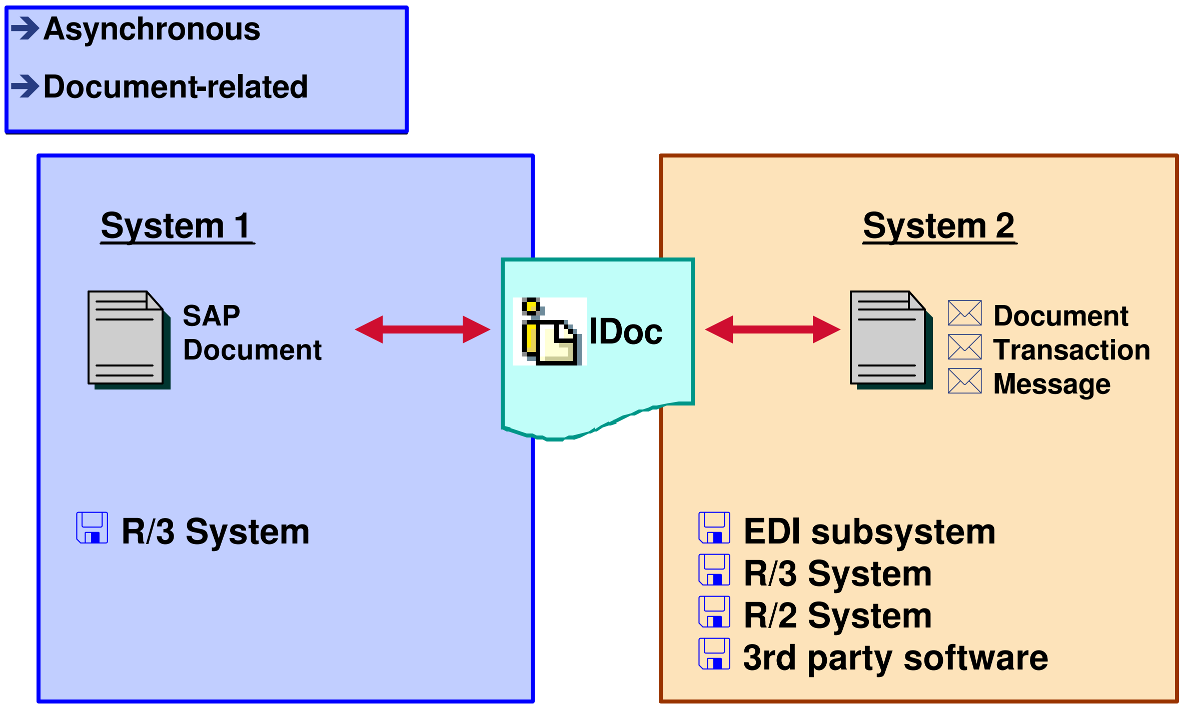


IDocs

- An IDoc is a container for the data of a business object or technical R/3 object
- Each IDoc has a message type. This indicates the type of business object or the business function of the data.
- Message types have processing rules in the receiving system.
- An IDoc contains the data in a segment hierarchy. The IDoc type describes the technical structure of the IDoc.
- IDoc types have versions.

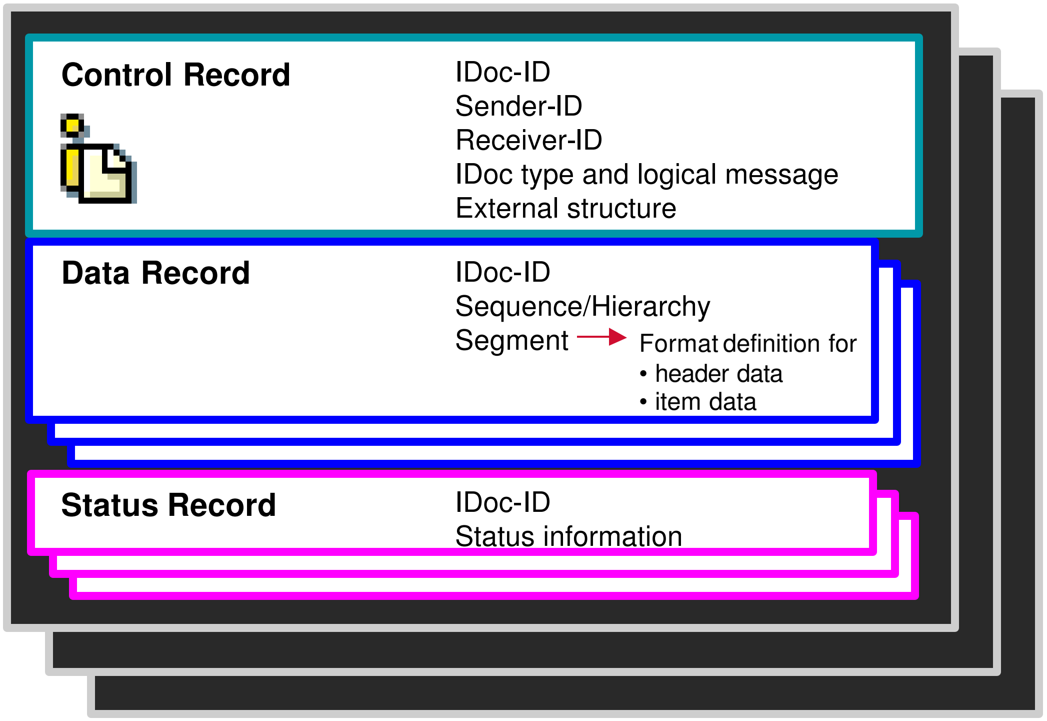
- Technical R/3 objects for IDocs with a message type exist, for example, audit data (ALEAUD) and distribution group (CONDA2)
- An IDoc has different characteristics: database table, text, HTML file, XML datastream, RFC call parameters.

IDoc Concept



- The key concepts of IDoc (i.e. *Intermediate Document*) are:
  - communication is asynchronous and
  - related to one document, i.e. transaction.
- Following these concepts IDoc facilitates distributed processes rather simply invoking functions.
- IDoc is the SAP standard interface to link application systems via messaging. Whereas one system is asumed to be SAP system R/3, the other system could be
  - EDI subsystem,
  - system R/3,
  - system R/2 or
  - any third-party application software.
- IDoc is available for system R/3 from release 2.1A on.
- IDoc is available for system R/2 from release 5.0F on.

IDoc Record Types



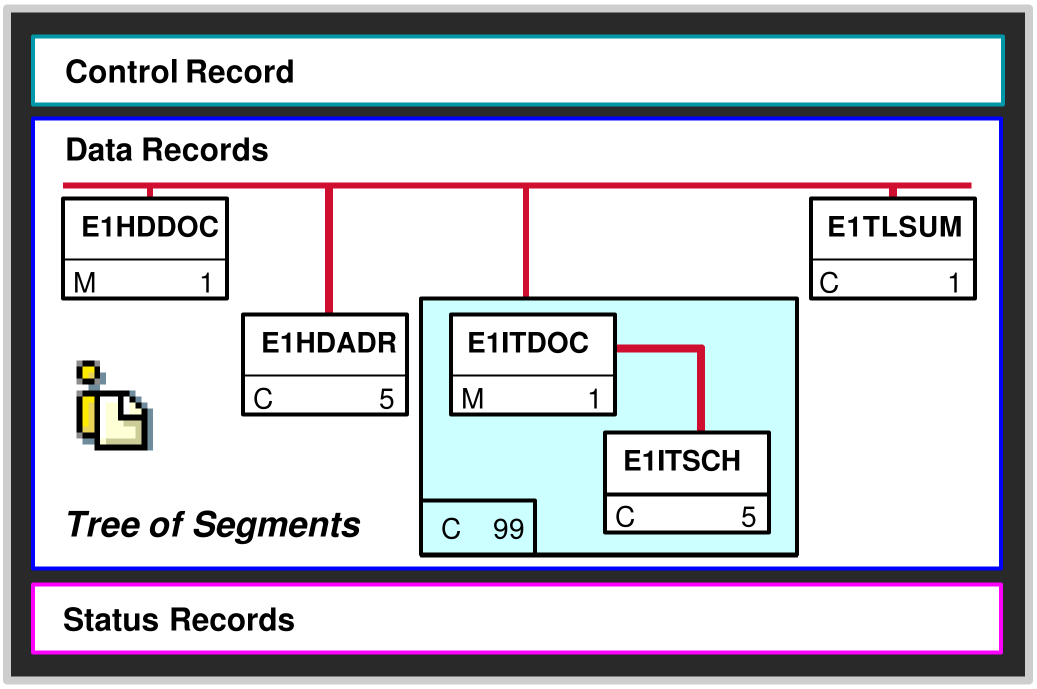
- Each IDoc in the SAP database consists of
  - exact one control record,
  - a number of data records, which span a tree of segments, and
  - a set of status records, though their number grows over time.
- The data to be exchanged between two systems via the IDoc interface carries in one transmission
  - IDocs (consisting only of control and data records), or
  - status information (consisting only of status records).
- Status information is expected to be sent to the IDoc interface to report on outbound IDocs. There is no status information sent for inbound IDocs by SAP in general.
- For individual status reporting in both directions and with all port types logical message STATUS is implemented by IDoc type SYSTAT01.
- Total length of records in version 2 (Releases 3.x):

• Control Record	464 bytes (plus 1 line delimiter)
• Data Record	55 bytes envelope, up-to 1,000 bytes segment (plus 1 line delimiter)
• Status Record	382 bytes (plus 1 line delimiter)

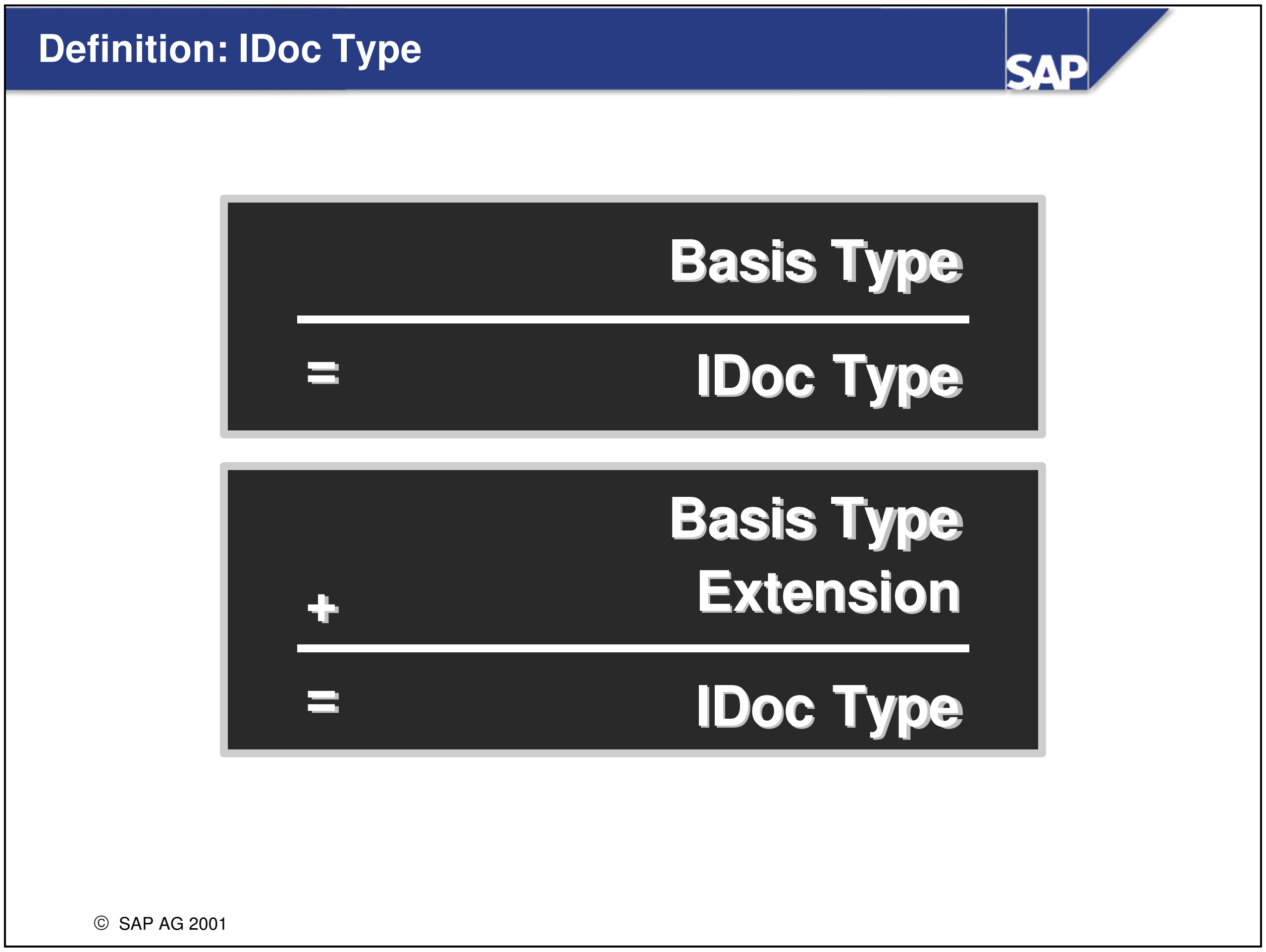
- Total length of records in version 3 (Releases 4.x):

• Control Record	524 bytes (plus 1 line delimiter)
• Data Record	63 bytes envelope, up-to 1,000 bytes segment (plus 1 line delimiter)
• Status Record	562 bytes (plus 1 line delimiter)

IDoc Types



- Each IDoc type is defined by its segments and their properties. These are segmenttype / segmentname, position / sequence and iteration / status.
- SAP segmenttypes (internal representation) begin with 'E1', all the corresponding segmentnames (external representation) begin with 'E2‘.
- Field length and data types are compared where applicable with data element directories of UN/EDIFACT, ANSI X12 and SAP‘s Data Repository.
- Codes for coded fields (currencies, countries, units of measure) are taken from ISO standards.
- The segments in the example above can be interpreted as follows
  - E1HDDOC Document Header
  - E1HDADR Header Addresses
  - E1ITDOC Item Details
  - E1ITSCH Item Schedule
  - E1TLSUM Trailor Section



Definition: IDoc Type

Basis Type

= IDoc Type

Basis Type

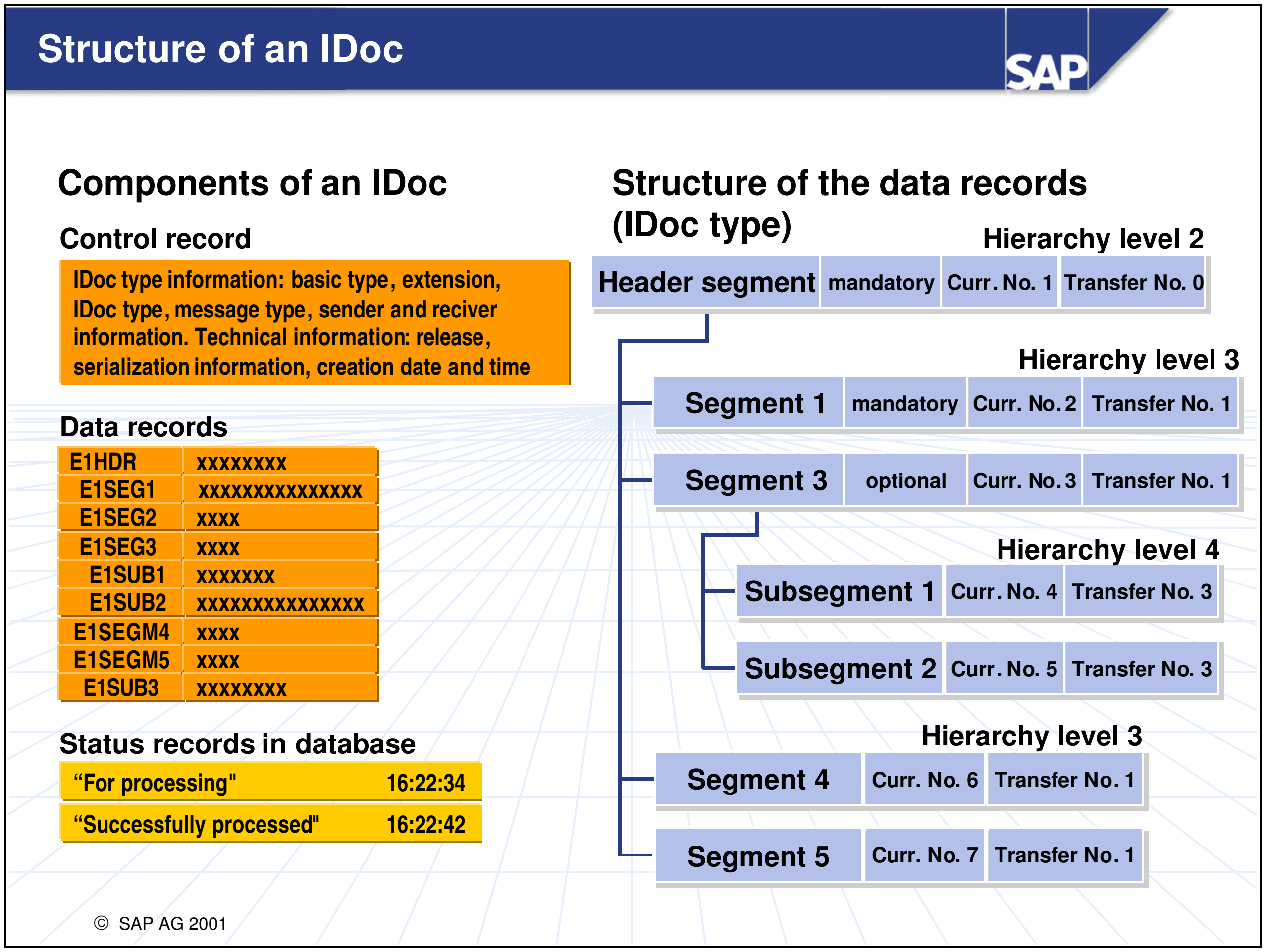
+ Extension

= IDoc Type

- IDoc = *I*ntermediate *Doc*ument !
  An IDoc type is a hierarichal, complex data structure constructed by segments, intended to carry a complete business document. Formally basis types and extensions are distinguished as IDoc types. The instance of such a document in „IDoc format“ is an IDoc of a well-defined IDoc type.
- An extension extends a basis type (SAP standard) with customer specific segments. All the customer segments are assigned in a direct or indirect way to segments of the basis type. The segments of the basis type are the „roots“ of the sub-trees constructed by the customer segments.
- In the control record the IDoc type constructed by an extension is identified by the fields

IDOCTYP	name of the basis type, and
CIMTYP	name of the extension.

- Examples:
  - IDoc type ORDERS01 as standard, i.e. without an extension:
    - field IDOCTYP has content ORDERS01
    - field CIMTYP is empty
  - IDoc type ORDERS01 as standard, but with an extension:
    - field IDOCTYP has content ORDERS01
    - field CIMTYP has the extension‘s name as content



Structure of an IDoc

Components of an IDoc

Control record

IDoc type information: basic type, extension, IDoc type, message type, sender and reciver information. Technical information: release, serialization information, creation date and time

Data records

E1HDR	xxxxxxxx
E1SEG1	xxxxxxxxxxxxxxx
E1SEG2	xxxx
E1SEG3	xxxx
E1SUB1	xxxxxxx
E1SUB2	xxxxxxxxxxxxxxx
E1SEGM4	xxxx
E1SEGM5	xxxx
E1SUB3	xxxxxxxx

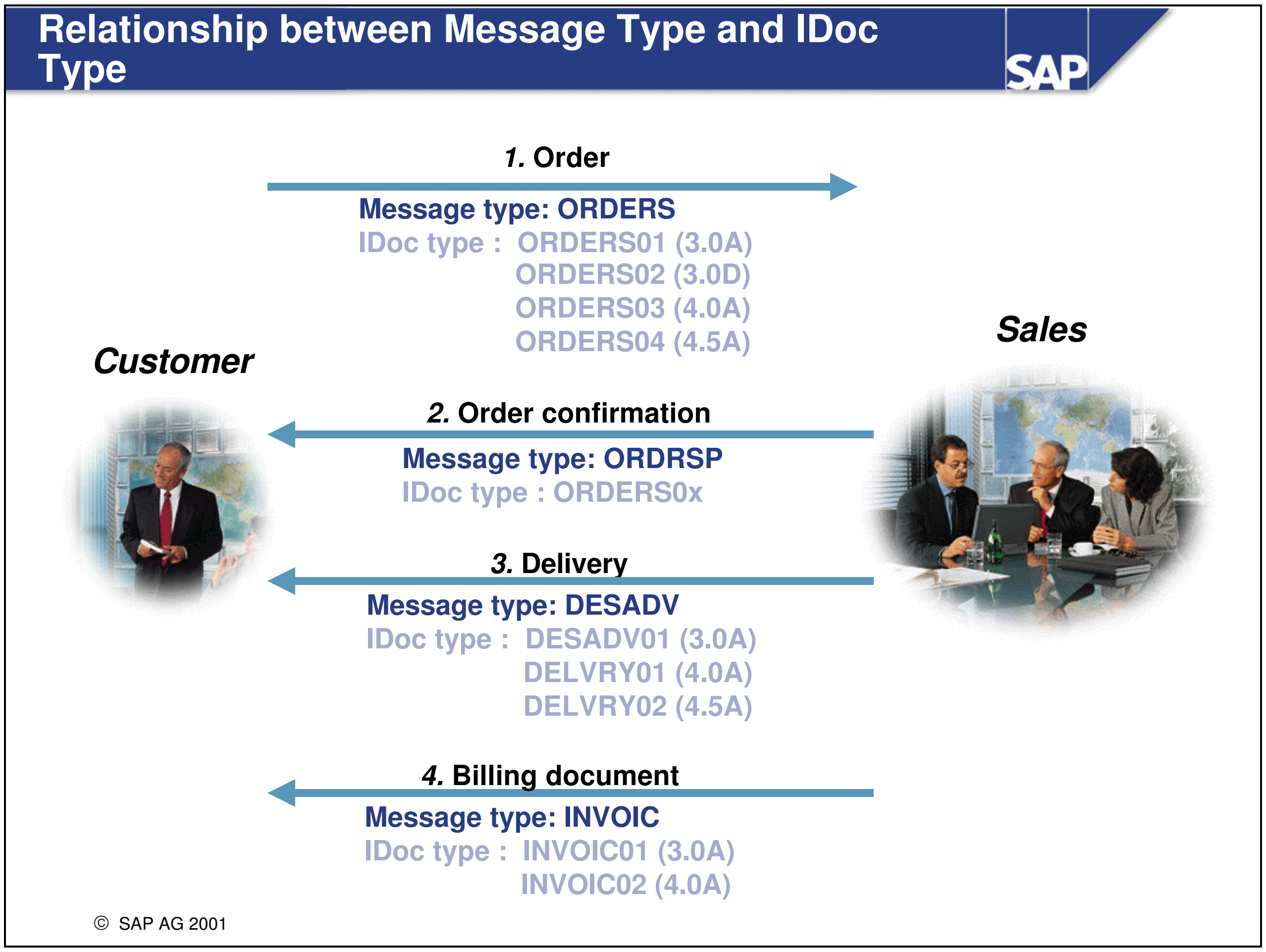
Status records in database

“For processing"	16:22:34
“Successfully processed"	16:22:42

Structure of the data records (IDoc type)

Hierarchy level 2
Header segment | mandatory | Curr. No. 1 | Transfer No. 0
Hierarchy level 3
Segment 1 | mandatory | Curr. No. 2 | Transfer No. 1
Segment 3 | optional | Curr. No. 3 | Transfer No. 1
Hierarchy level 4
Subsegment 1 | Curr. No. 4 | Transfer No. 3
Subsegment 2 | Curr. No. 5 | Transfer No. 3
Hierarchy level 3
Segment 4 | Curr. No. 6 | Transfer No. 1
Segment 5 | Curr. No. 7 | Transfer No. 1

- Status records are data records assigned locally to an IDoc - they not part of an IDoc . They are not sent together with the IDoc.



Relationship between Message Type and IDoc Type

Customer

Sales

1. Order

Message type: ORDERS

IDoc type : ORDERS01 (3.0A)
ORDERS02 (3.0D)
ORDERS03 (4.0A)
ORDERS04 (4.5A)

2. Order confirmation

Message type: ORDRSP

IDoc type : ORDERS0x

3. Delivery

Message type: DESADV

IDoc type : DESADV01 (3.0A)
DELVRY01 (4.0A)
DELVRY02 (4.5A)

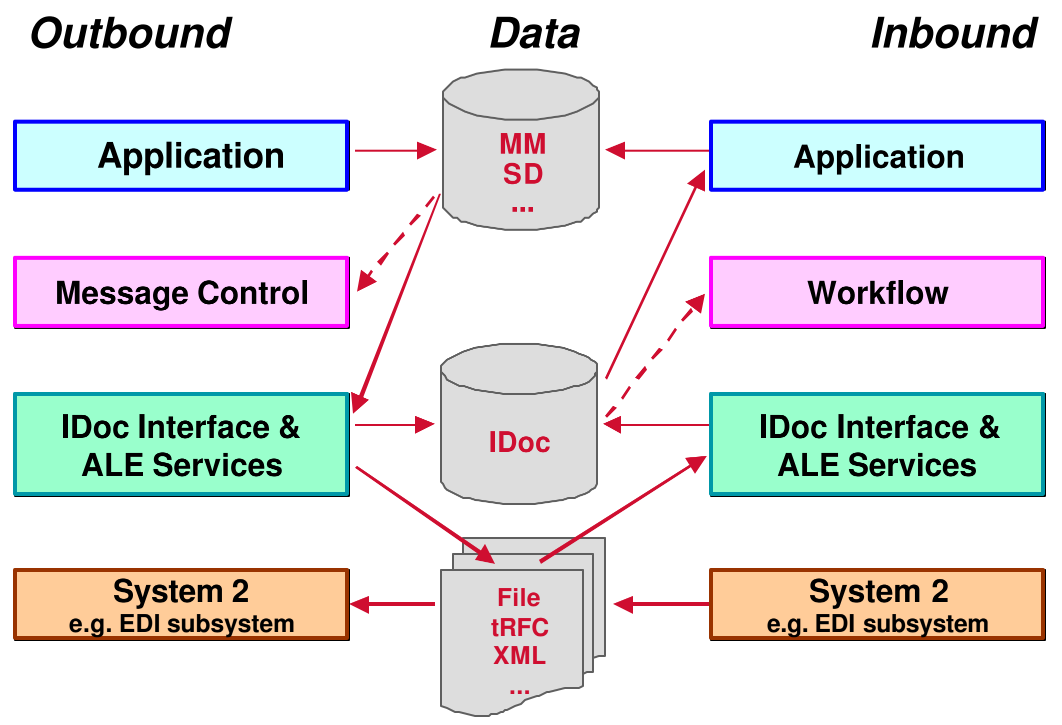
4. Billing document

Message type: INVOIC

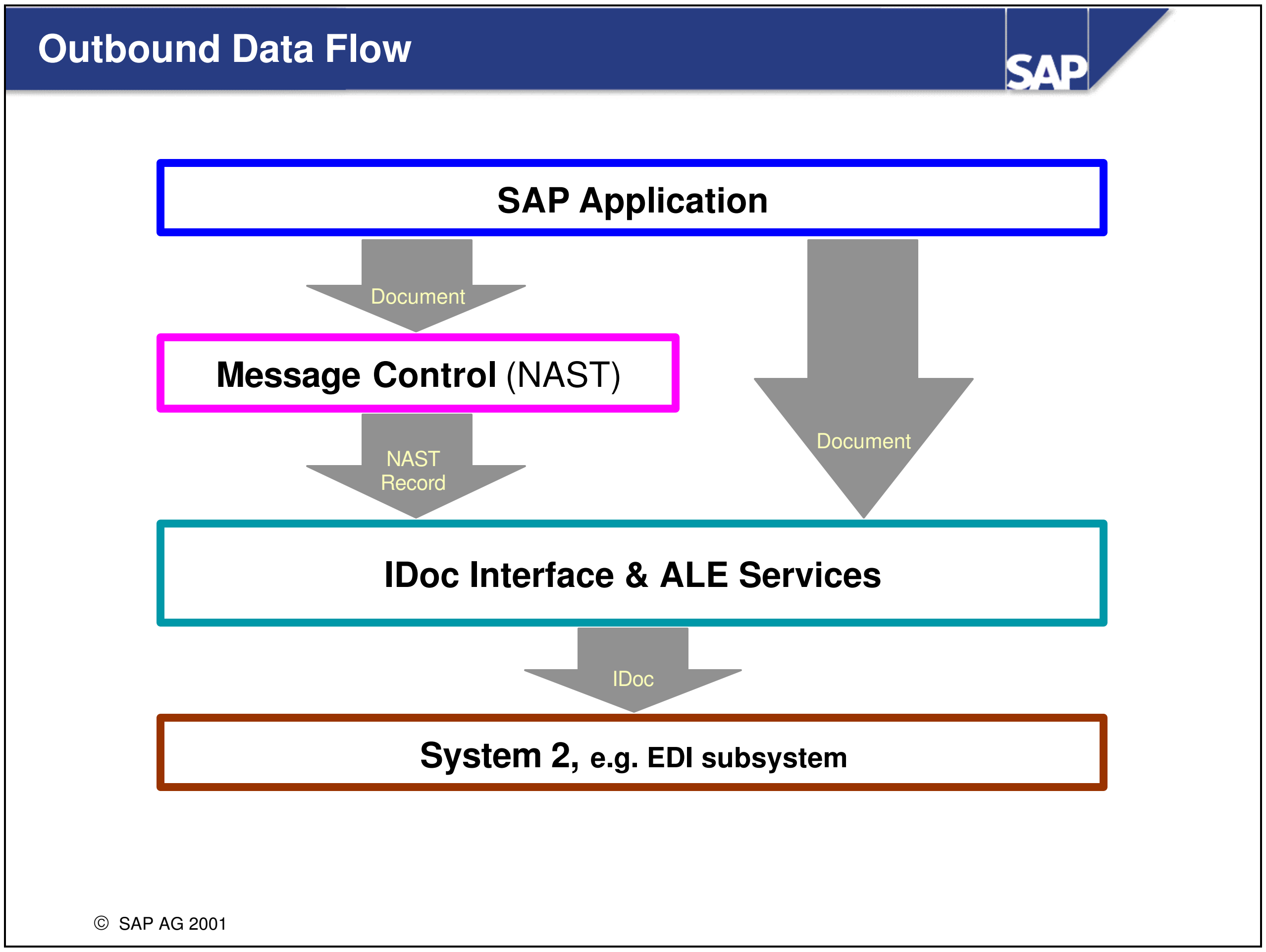
IDoc type : INVOIC01 (3.0A)
INVOIC02 (4.0A)

- Message types specify the semantics of application data. The message type is usually based on an EDIFACT message type.
- The syntax (structure information) of the data is described in the IDoc type.

IDoc Data Flow

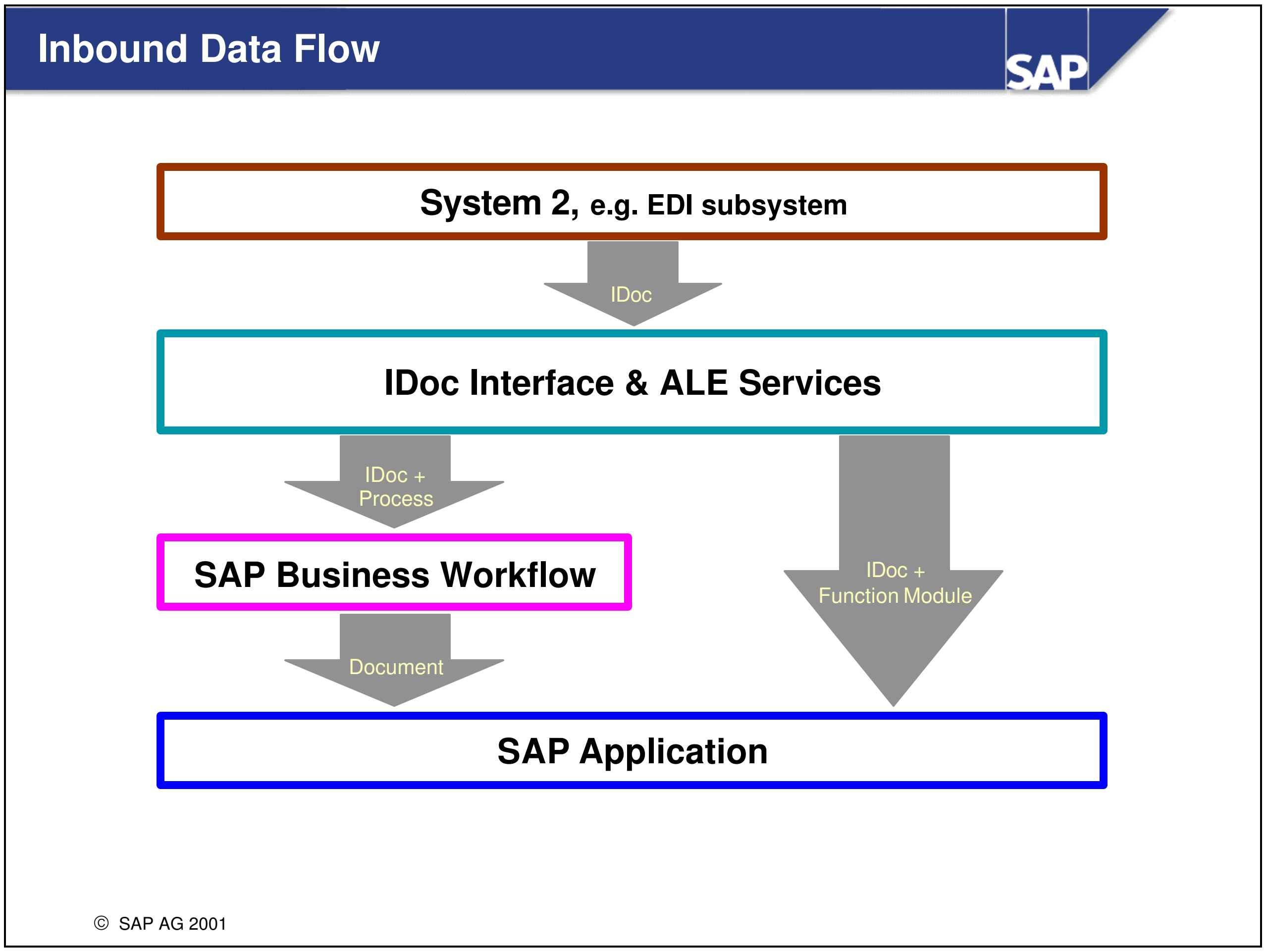


- Please read the left (outbound) from top and the right (inbound) from bottom.
- Message Control is in use with transactions of the supply chain, applications MM and SD, in outbound processing.
- Workflow is conditional with all inbound processing by all SAP applications.
- Nevertheless Workflow is in use with all SAP applications to report on exceptions in the implemented processes. For details please refer to the notification and monitoring section.



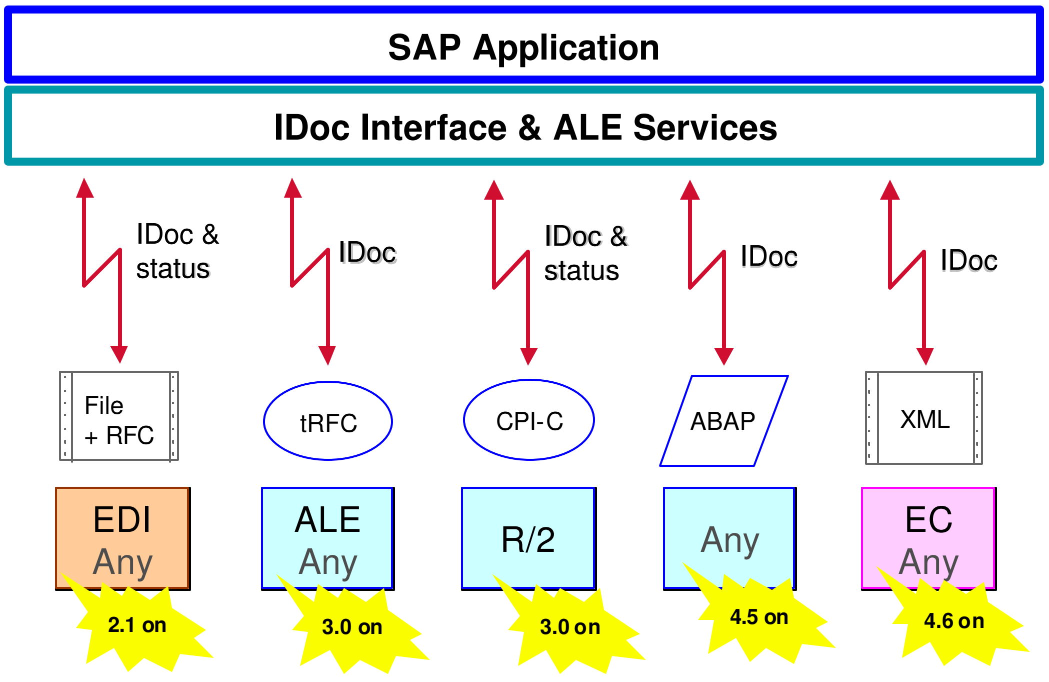
Outbound Data Flow

- SAP applications can create IDocs under control of Message Control or on their own.
- An IDoc can go from an application through Message Control into the IDoc interface. The IDoc interface is invoked by either form-routine EDI_PROCESSING or ALE_PROCESSING in program RSNASTED. In the IDoc interface the IDoc is passed to ALE services (conditional) and later on transferred to system 2 according to its port definition.
- Applications that do not use Message Control will create IDocs directly. They invoke function module MASTER_IDOC_DISTRIBUTE and pass the IDocs to ALE services, where the IDocs will be preprocessed according to ALE customizing.
- All IDocs are transferred according to the port definition,
  - most common between R/3 systems is the (asynchronous) transactional RFC interface (standard ALE scenario)
  - most common between R/3 system and EDI subsystem is the flat file interface (standard EDI scenario).


Inbound Data Flow

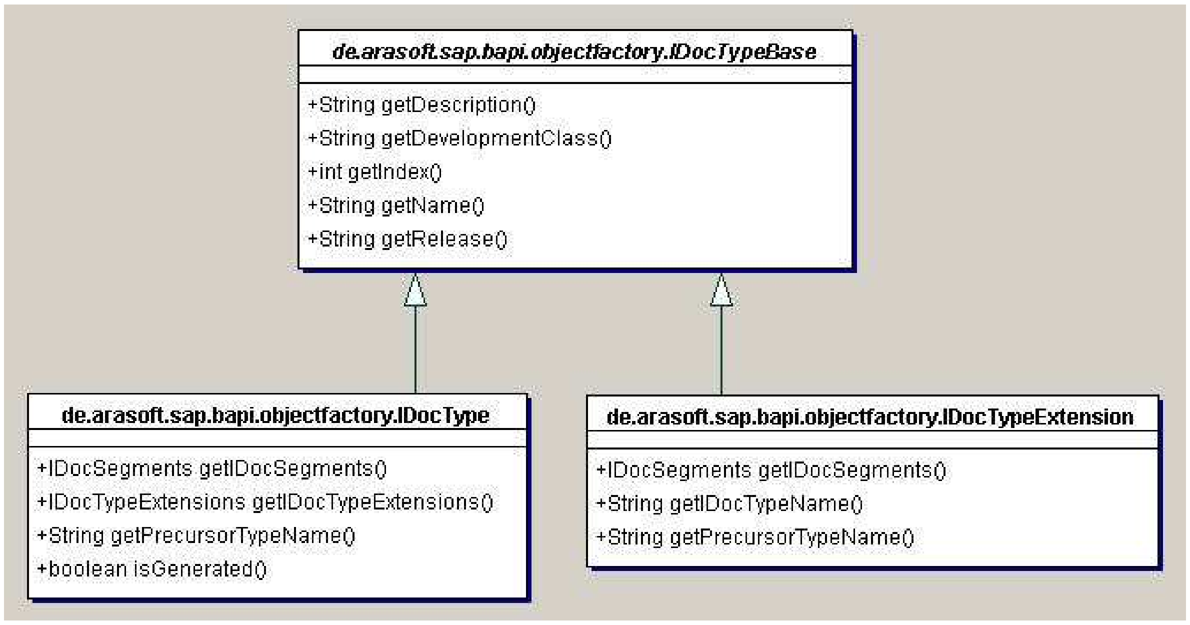
- System 2, e.g. the EDI subsystem, sends IDocs to the SAP system R/3.
- All IDocs are received via a port with its related port definition,
  - most common between R/3 systems is the (asynchronous) transactional RFC interface (standard ALE scenario)
  - most common between R/3 system and EDI subsystem is the flat file interface (standard EDI scenario).
- The IDocs enter the SAP system at the IDoc interface. The IDoc interface accepts the IDocs, passes the IDocs either through ALE services or to workflow management.
- The ALE services can be customized to post IDocs to the application directly via means of CALL FUNCTION or via means of workflow management.

Port Types of the IDoc Interface



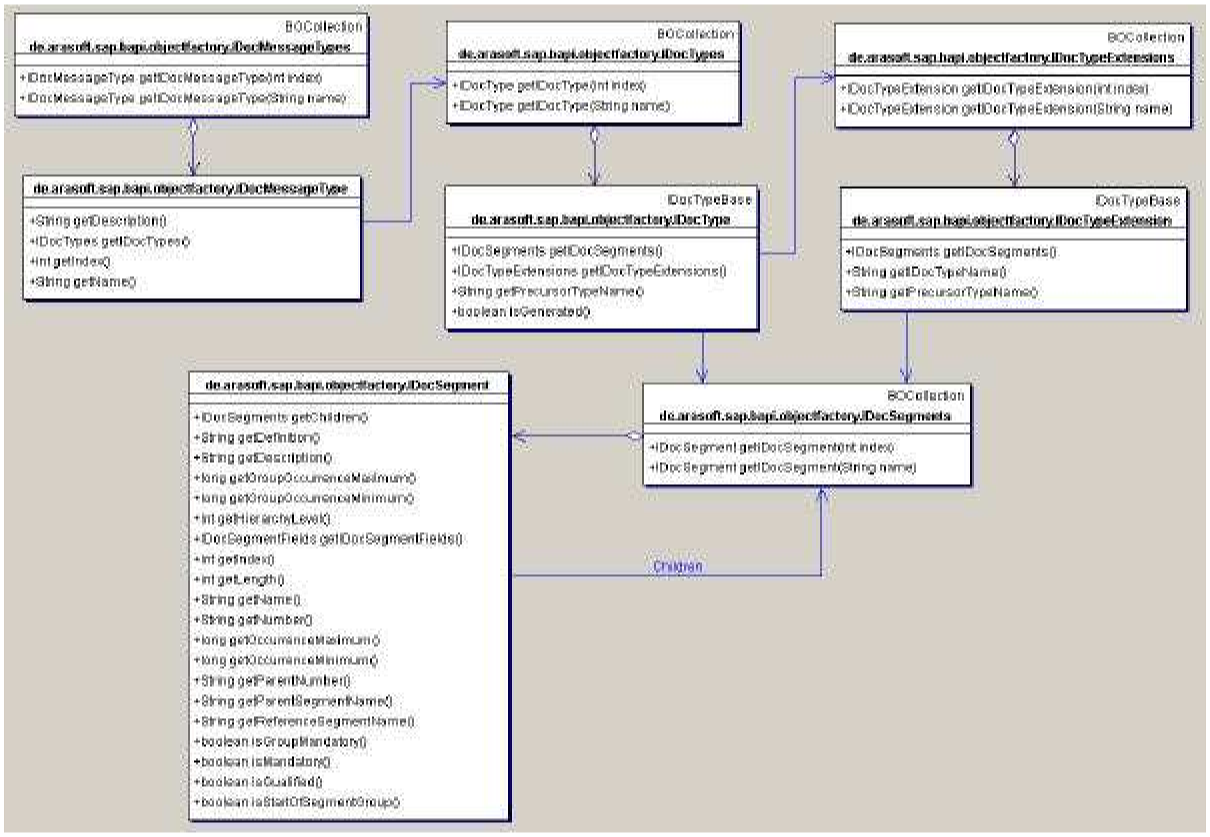
- The IDoc interface offers 5 different communication channels, defined by the appropriate port types, within system R/3.
- Port type „File“
  Transfer of information via files and synchronous RFC for triggering of the destination by the source.
  This allows transfer of IDoc data and status report.
- Port type „tRFC“
  Transfer of all information via (asynchronous) transactional RFC.
  This is the preferred method in ALE scenarios and allows transfer of IDoc data.
- Port type „CPI-C“
  Transfer of all information via CPI-C. This option is in use only with system R/2 couppling based on the R/2-IDoc interface as released in R/2, 5.0F.
  This allows transfer of IDoc data and status report.
- Port type „ABAP“
  Transfer of all information with customer specific program logic.
  This is thought to be a „Project-Exit“ for any unforseen communication channel.
- Port type „XML“
  Transfer of all information via XML compatible files, including DTD if requested. This port type is released as prototype, changes and enhancements are expected for future releases.
- The port type „Internet“ is not recommended for use anymore. If you plan for Web/EDI, please refer to an EDI subsystem. If you plan for mySAP.com, please use the strategic recommendation of SAP Business Connector with XML.
  Please refer also to OSS note #315083.

IDoc Object Model 1

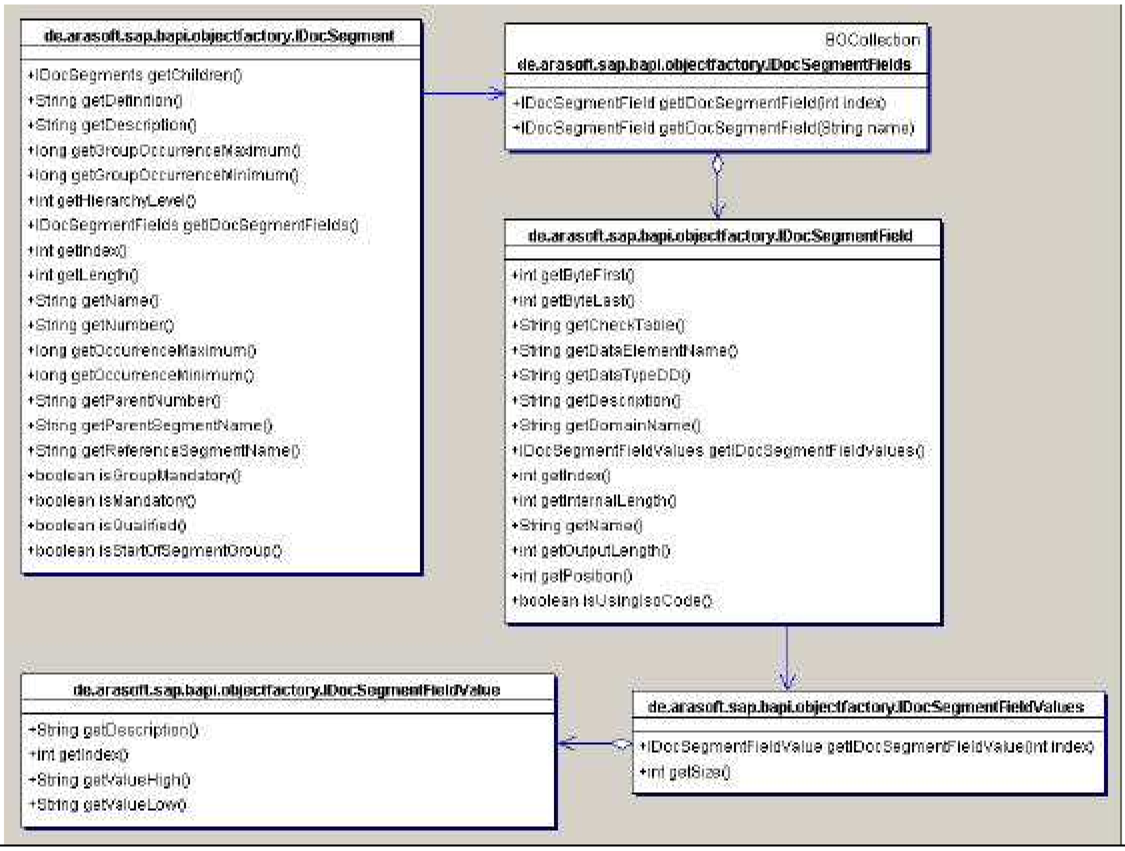


de.arasoft.sap.bapi.objectfactory.IDocTypeBase
+String getDescription()
+String getDevelopmentClass()
+int getIndex()
+String getName()
+String getRelease()
de.arasoft.sap.bapi.objectfactory.IDocType
+IDocSegments getIDocSegments()
+IDocTypeExtensions getIDocTypeExtensions()
+String getPrecursorTypeName()
+boolean isGenerated()
de.arasoft.sap.bapi.objectfactory.IDocTypeExtension
+IDocSegments getIDocSegments()
+String getIDocTypeName()
+String getPrecursorTypeName()

IDoc Object Model 2


BOCollection
de.arasoft.sap.bapi.objectfactory.IDocMessageTypes
+IDocMessageType getIDocMessageType(int index)
+IDocMessageType getIDocMessageType(String name)
de.arasoft.sap.bapi.objectfactory.IDocMessageType
+String getDescription()
+IDocTypes getIDocTypes()
+int getIndex()
+String getName()
BOCollection
de.arasoft.sap.bapi.objectfactory.IDocTypes
+IDocType getIDocType(int index)
+IDocType getIDocType(String name)
IDocTypeBase
de.arasoft.sap.bapi.objectfactory.IDocType
+IDocSegments getIDocSegments()
+IDocTypeExtensions getIDocTypeExtensions()
+String getPrecursorTypeName()
+boolean isGenerated()
BOCollection
de.arasoft.sap.bapi.objectfactory.IDocTypeExtensions
+IDocTypeExtension getIDocTypeExtension(int index)
+IDocTypeExtension getIDocTypeExtension(String name)
IDocTypeBase
de.arasoft.sap.bapi.objectfactory.IDocTypeExtension
+IDocSegments getIDocSegments()
+String getIDocTypeName()
+String getPrecursorTypeName()
de.arasoft.sap.bapi.objectfactory.IDocSegment
+IDocSegments getChildren()
+String getDefinition()
+String getDescription()
+long getGroupOccurrenceMaximum()
+long getGroupOccurrenceMinimum()
+int getHierarchyLevel()
+IDocSegmentFields getIDocSegmentFields()
+int getIndex()
+int getLength()
+String getName()
+String getNumber()
+long getOccurrenceMaximum()
+long getOccurrenceMinimum()
+String getParentNumber()
+String getParentSegmentName()
+String getReferenceSegmentName()
+boolean isGroupMandatory()
+boolean isMandatory()
+boolean isQualified()
+boolean isStartOfSegmentGroup()
BOCollection
de.arasoft.sap.bapi.objectfactory.IDocSegments
+IDocSegment getIDocSegment(int index)
+IDocSegment getIDocSegment(String name)
Children

IDoc Object Model 3



```
de.arasoft.sap.bapi.objectfactory.IDocSegment
+IDocSegments getChildren()
+String getDefinition()
+String getDescription()
+long getGroupOccurrenceMaximum()
+long getGroupOccurrenceMinimum()
+int getHierarchyLevel()
+IDocSegmentFields getIDocSegmentFields()
+int getIndex()
+int getLength()
+String getName()
+String getNumber()
+long getOccurrenceMaximum()
+long getOccurrenceMinimum()
+String getParentNumber()
+String getParentSegmentName()
+String getReferenceSegmentName()
+boolean isGroupMandatory()
+boolean isMandatory()
+boolean isQualified()
+boolean isStartOfSegmentGroup()
```

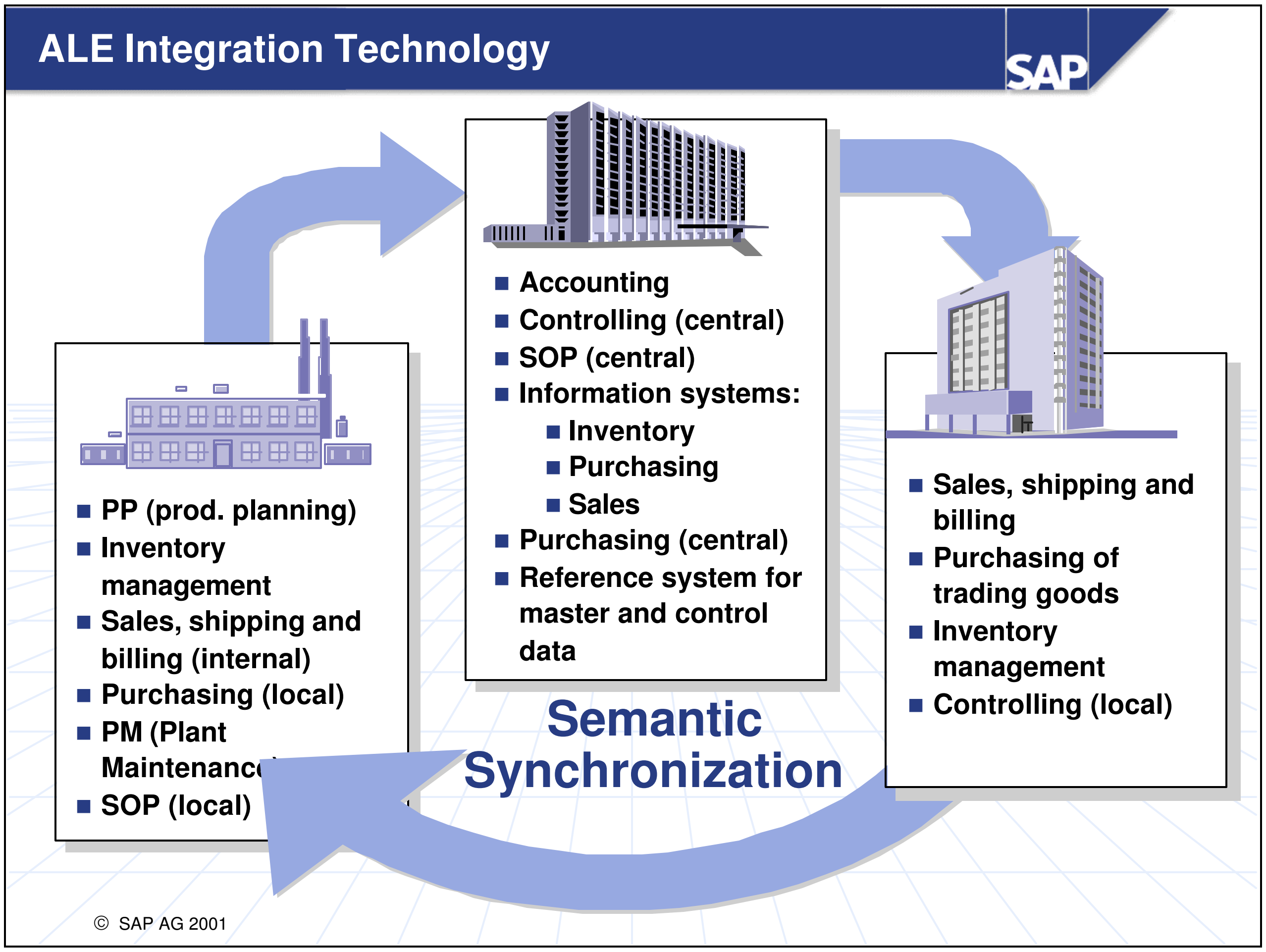
```
BOCollection
de.arasoft.sap.bapi.objectfactory.IDocSegmentFields
+IDocSegmentField getIDocSegmentField(int index)
+IDocSegmentField getIDocSegmentField(String name)
```

```
de.arasoft.sap.bapi.objectfactory.IDocSegmentField
+int getByteFirst()
+int getByteLast()
+String getCheckTable()
+String getDataElementName()
+String getDataTypeDD()
+String getDescription()
+String getDomainName()
+IDocSegmentFieldValues getIDocSegmentFieldValues()
+int getIndex()
+int getInternalLength()
+String getName()
+int getOutputLength()
+int getPosition()
+boolean isUsingIsoCode()
```

```
de.arasoft.sap.bapi.objectfactory.IDocSegmentFieldValue
+String getDescription()
+int getIndex()
+String getValueHigh()
+String getValueLow()
```

```
de.arasoft.sap.bapi.objectfactory.IDocSegmentFieldValues
+IDocSegmentFieldValue getIDocSegmentFieldValue(int index)
+int getSize()
```

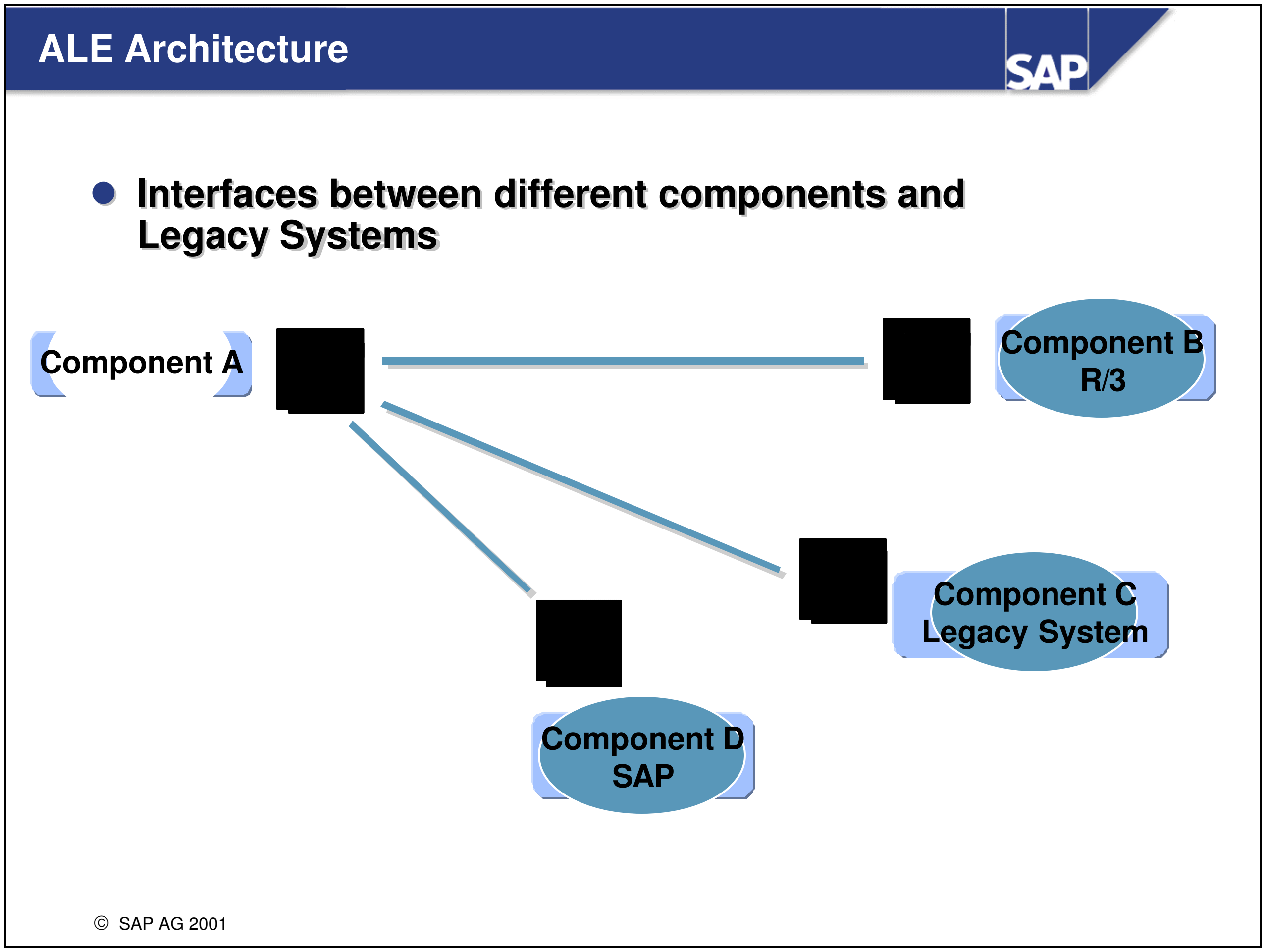
ALE Integration Technology



- ALE was originally used to distribute business processes by forwarding the required data and triggering a workflow in the external system.
- The focus was on a company's internal distribution scenarios: department-to-department
- Using the stable IDoc interface ALE enables business processes that go beyond company boundaries: business-to-business.
- ALE is independent of the communication layer and can implement business processes over the Internet.

Purpose of ALE

- Supports data consistency and data availability in distributed business processes
- Provides an infrastructure for coupling systems loosely through asynchronous messaging using IDocs or coupling systems narrowly through synchronous BAPI calls
- Enables distribution between systems with different release versions
- Enables R/3 to communicate with non-SAP systems and R/2
- Provides functions for administration, monitoring and development
- Predefined ALE business processes cover important business functions (Library of ALE Business Processes)

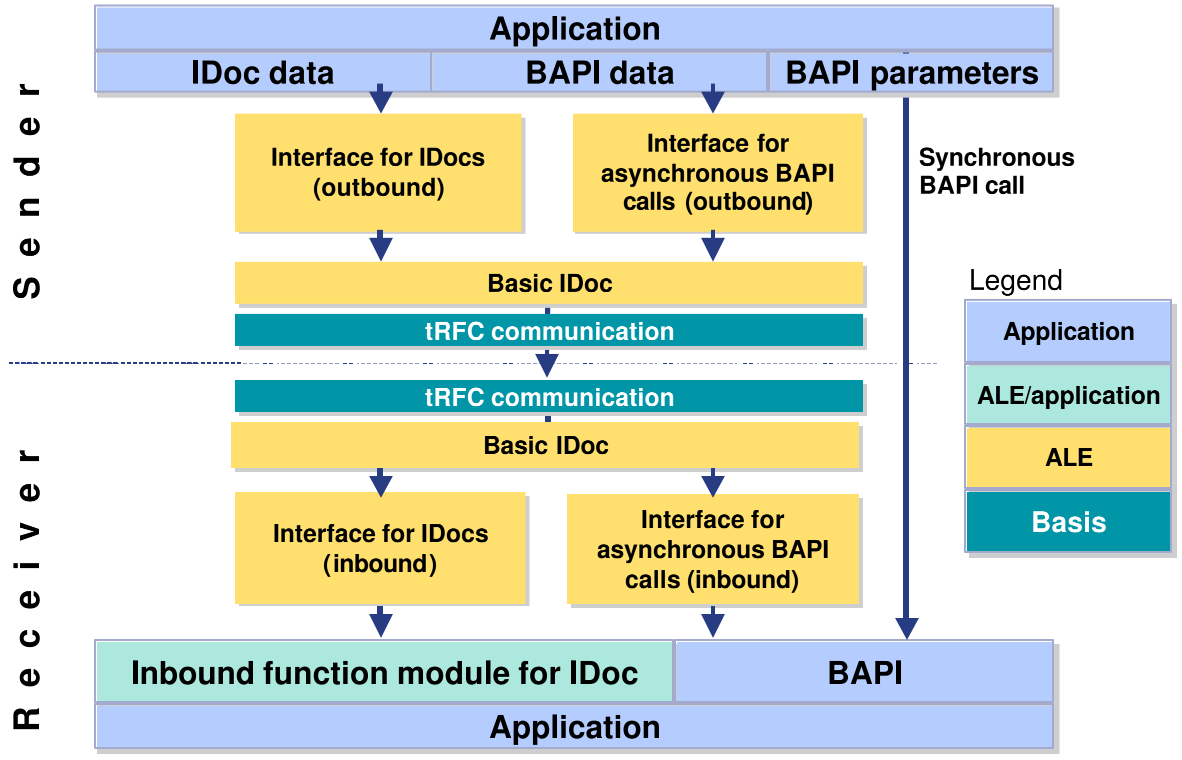


ALE Architecture

- Interfaces between different components and Legacy Systems

- ALE provides the functionality to distribute and receive data to and from:
  - R/3 Systems (>= 3.0)
  - different SAP components
  - R/2 Systems
  - Non-SAP-Systems
- It uses synchronous and asynchronous RFC's and standard communication protocols.
- The data can be processed on-line or in batch mode.

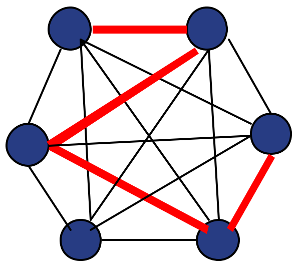
ALE Data Flow Between R/3 Systems



- Outbound function modules for asynchronous BAPI calls
  - ALE_<BO>_<METHOD>
- Inbound function modules for IDoc via tRFC
  - INBOUND_IDOC_PROCESS (3.x)
  - IDOC_INBOUND_ASYNCHRONOUS (since 4.x)
- Inbound function modules for updating asynchronous BAPIs
  - BAPI_<BO>_<METHOD>

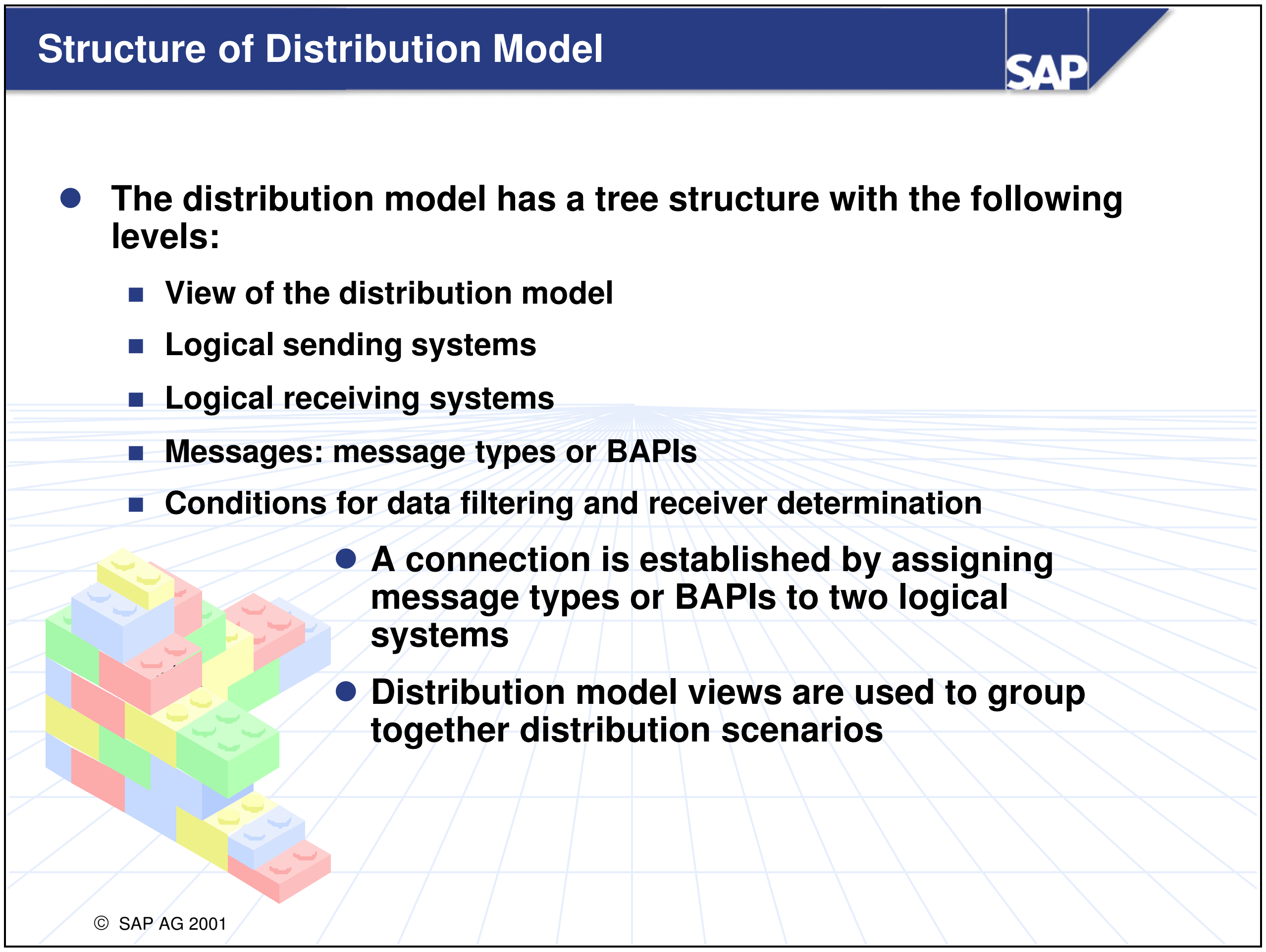
ALE Services

- Cross Application Processes require more than interfaces
  - Which system is the right server?
  - Harmonization of the sub-process
  - Semantic synchronization
  - Monitoring
  - Error handling



- => ALE provides special Services for this

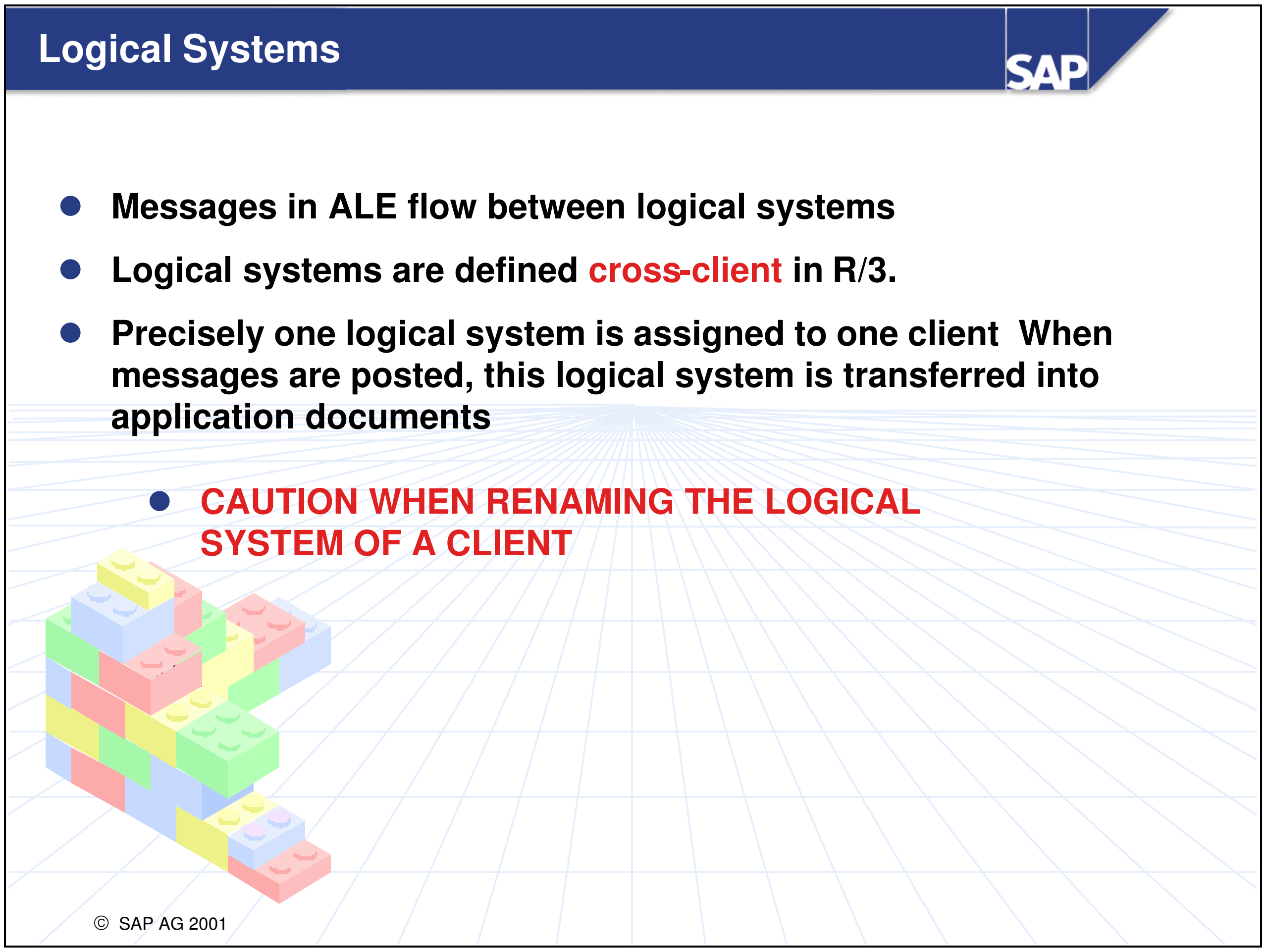
- The control of business processes in distributed environments is more complex than the process control on just one integrated system.
- Additional harmonization and synchronization of processes and data flows is required.
- More details of these services are explained on the next slides.



Structure of Distribution Model

- The distribution model has a tree structure with the following levels:
  - View of the distribution model
  - Logical sending systems
  - Logical receiving systems
  - Messages: message types or BAPIs
  - Conditions for data filtering and receiver determination
- A connection is established by assigning message types or BAPIs to two logical systems
- Distribution model views are used to group together distribution scenarios

- The distribution model is client-dependent and may contain one or more views.

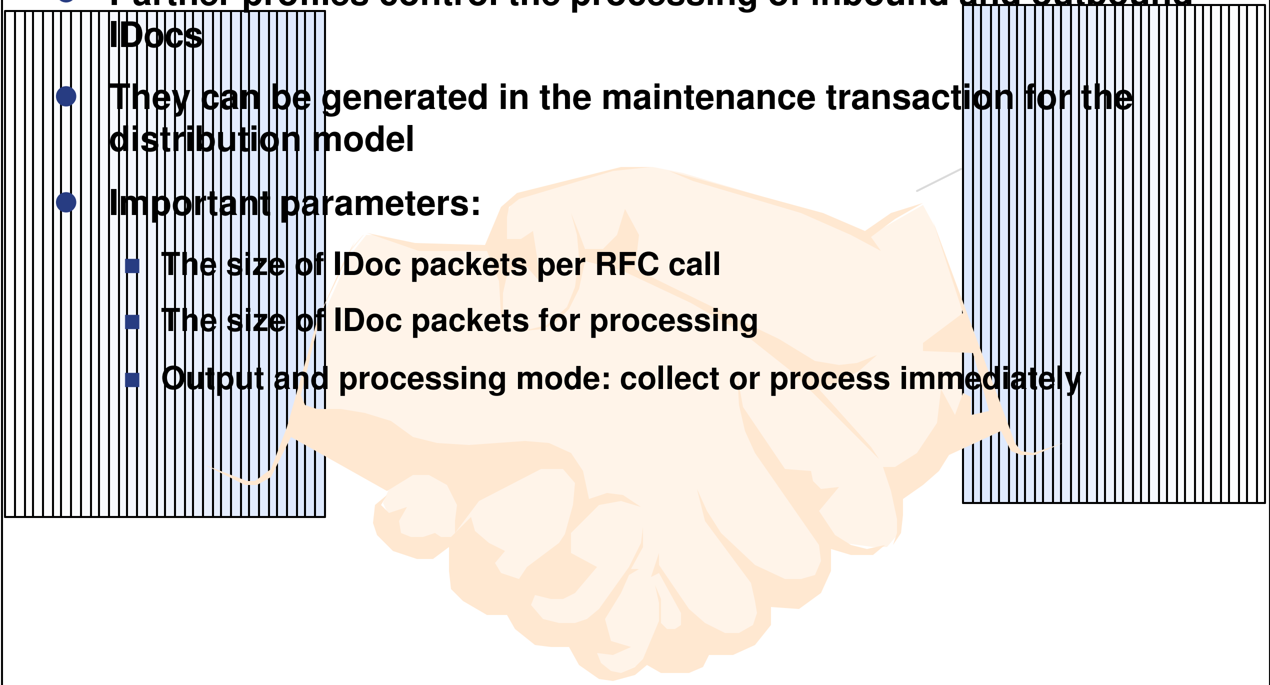


Logical Systems

- Messages in ALE flow between logical systems
- Logical systems are defined cross-client in R/3.
- Precisely one logical system is assigned to one client When messages are posted, this logical system is transferred into application documents
  - CAUTION WHEN RENAMING THE LOGICAL SYSTEM OF A CLIENT

- Other characteristics of logical systems
  - Logical systems have a technical name up to 10 characters long and an explanatory short text.
  - The maintenance of logical systems is an ALE Customizing activity
  - Application documents with an empty logical system or with the logical system of the current client are interpreted as local documents.
- Problems with the logical system name
  - If the logical system name of a client is changed, the modeling of the ALE message flow may become inconsistent.
  - If the logical system name of a client is changed, application documents may become invisible because the application classifies them as external documents.
  - As of R/3 Release 4.5A a logical system must be assigned to each client. For this reason, meaningless logical system names had often been assigned that had to later be renamed. When the logical systems were renamed, the problems with application documents described above occurred.
  - ALE provides a tool to rename logical system names in application tables. You can find more information in ALE Customizing or in a note.

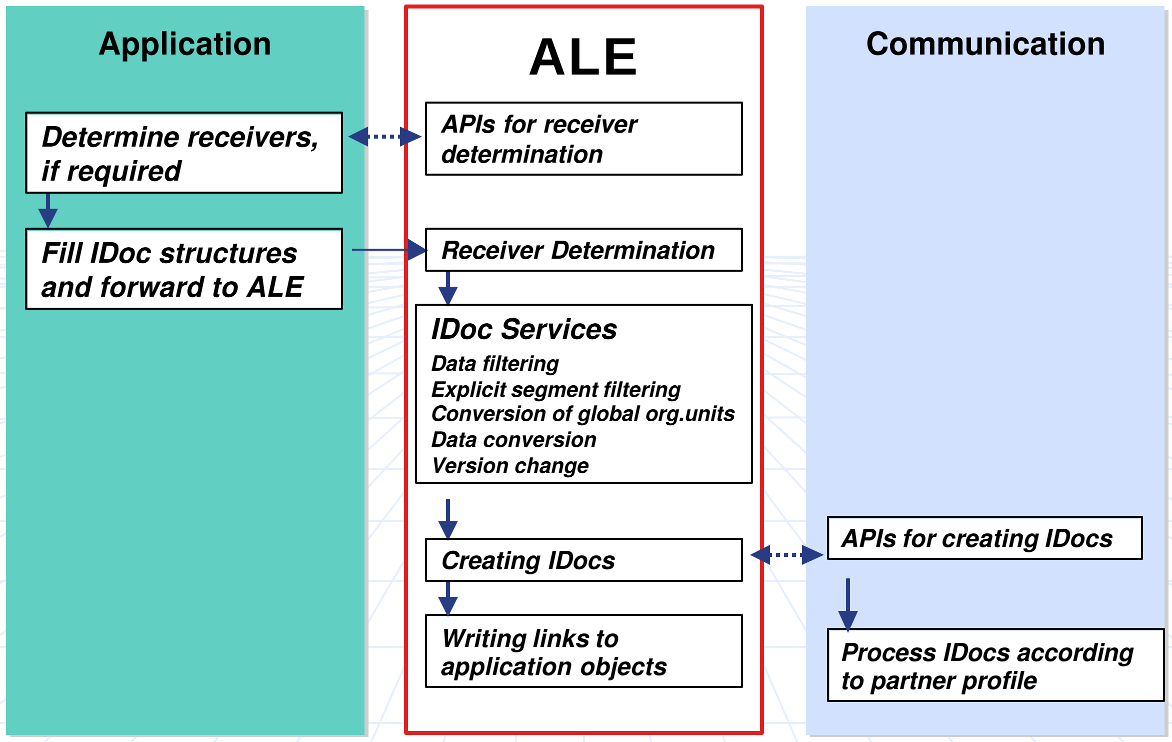
Partner Profiles



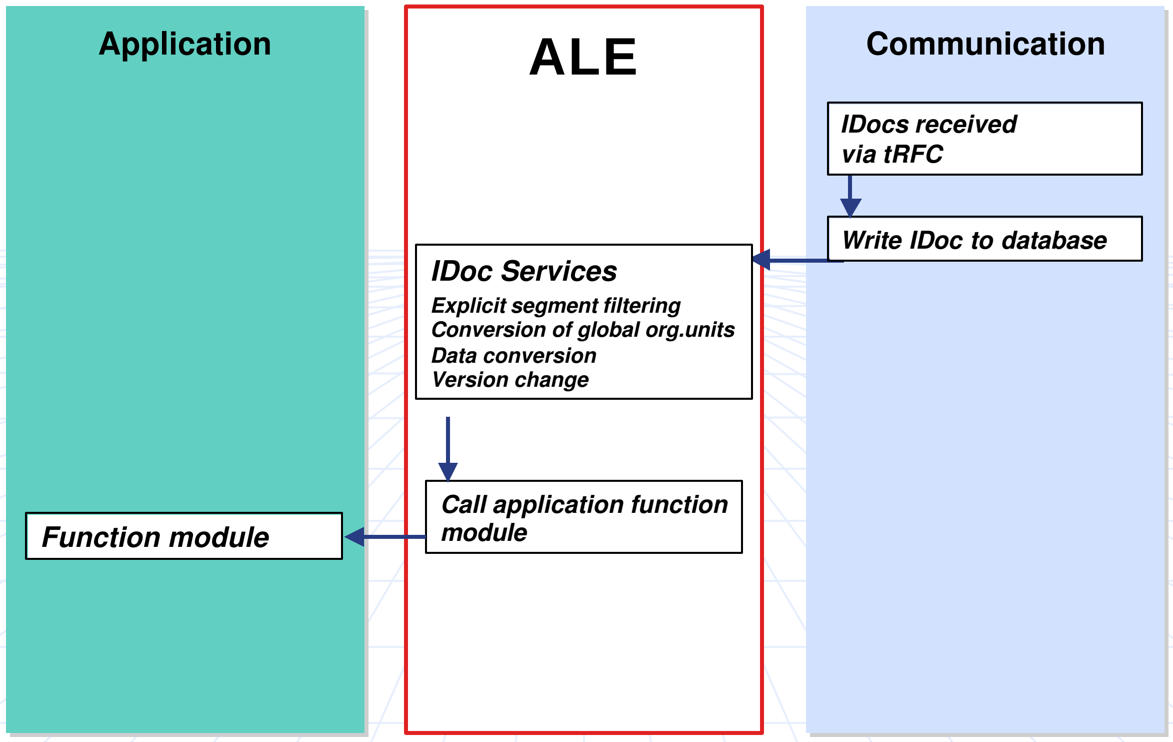
- Partner profiles control the processing of inbound and outbound IDocs
- They can be generated in the maintenance transaction for the distribution model
- Important parameters:
  - The size of IDoc packets per RFC call
  - The size of IDoc packets for processing
  - Output and processing mode: collect or process immediately

- Partner profiles are client-dependent.
- Partner profiles with the partner type logical system only exist for systems which the current client sends messages to or receives messages from.
- For this reason when the partner profiles are generated, ALE only considers connections that contain the logical system of the current client.

Flow Diagram
IDoc Outbound Processing


Application
Determine receivers, if required
Fill IDoc structures and forward to ALE
ALE
APIs for receiver determination
Receiver Determination
IDoc Services
Data filtering
Explicit segment filtering
Conversion of global org.units
Data conversion
Version change
Creating IDocs
Writing links to application objects
Communication
APIs for creating IDocs
Process IDocs according to partner profile

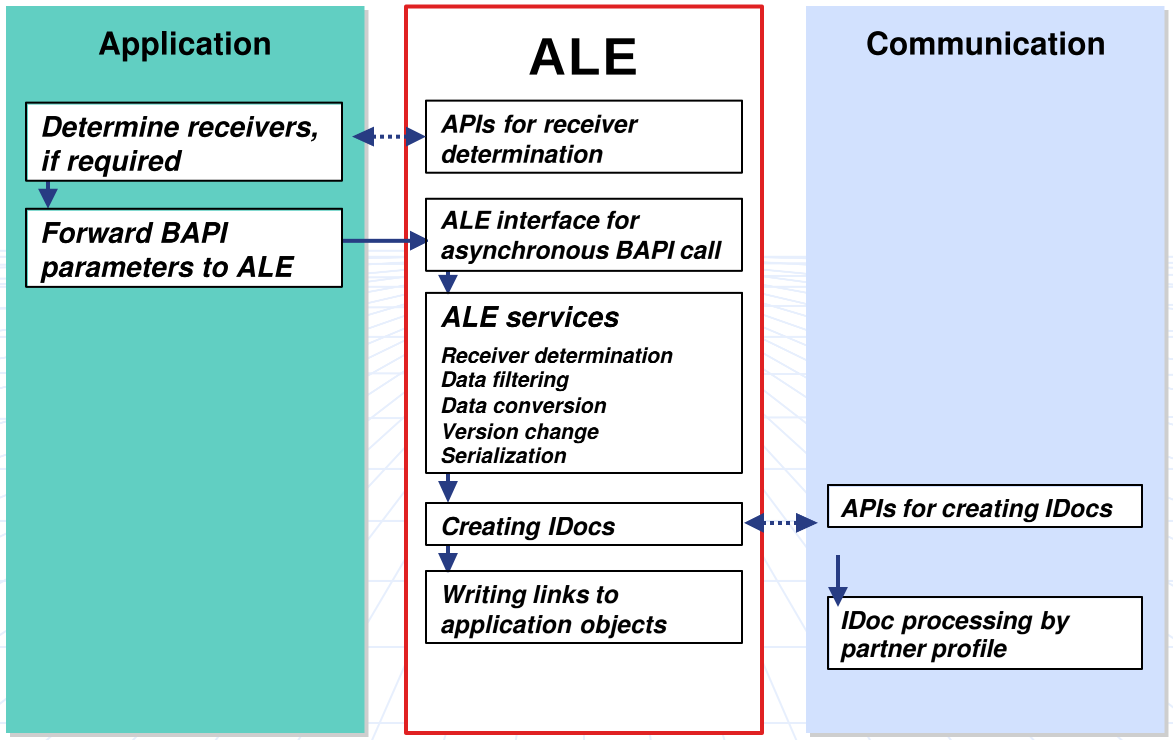
Flow Diagram IDoc Inbound Processing



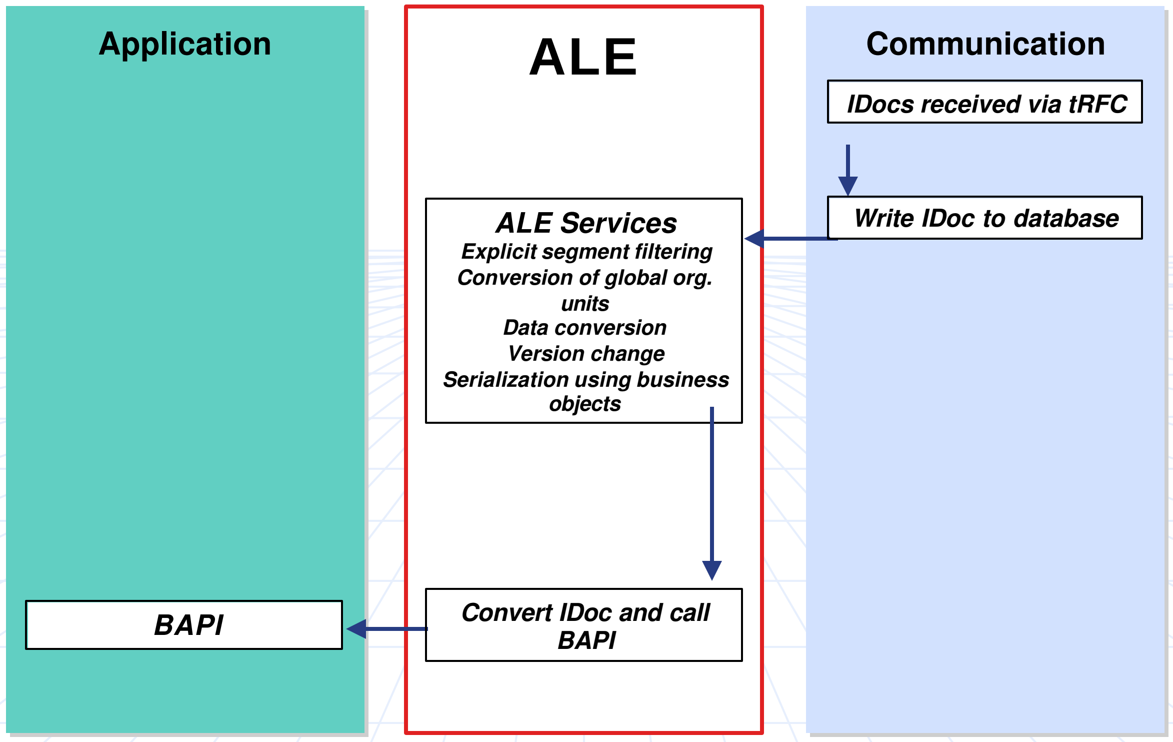
Application Interface for Asynchronous BAPIs

- Outbound processing for asynchronous BAPI calls is similar to IDoc processing
- In an interim step the ALE interface converts the BAPI parameters into an IDoc with the appropriate segment structure and message type
- For the ALE users the usage of a BAPI is transparent and equivalent to classical message types

Flow Diagram: Asynchronous BAPI Outbound Processing


Application
Determine receivers, if required
Forward BAPI parameters to ALE
ALE
APIs for receiver determination
ALE interface for asynchronous BAPI call
ALE services
Receiver determination
Data filtering
Data conversion
Version change
Serialization
Creating IDocs
Writing links to application objects
Communication
APIs for creating IDocs
IDoc processing by partner profile

Flow Diagram: Asynchronous BAPI Inbound Processing


Application
ALE
Communication
IDocs received via tRFC
Write IDoc to database
ALE Services
Explicit segment filtering
Conversion of global org. units
Data conversion
Version change
Serialization using business objects
Convert IDoc and call BAPI
BAPI

RFMs Used in IDoc Communication (tRFC)

- INBOUND_IDOC_PROCESS (3.x)
- IDOC_INBOUND_ASYNCHRONOUS (since 4.x)

- Use SAP JCO or SAP Business Connector for your own application scenarios.

Scenarios for External Applications

- **Inbound**
  - **Use IDocs as an alternative to update BAPIs if**
    - **No BAPI exists for your requirements**
    - **You need guaranteed (once and once only) delivery**
      - **BAPI call via tRFC also possible (remember: no outbound data)**
- **Outbound**
  - **Subscribe to IDoc message types in the ALE distribution model in order to keep an external database synchronized.**

ALE/IDoc: Unit Summary

- Application messages can be sent and received asynchronously via ALE/IDoc.

Unit: ALE/IDoc
Topic: Outbound tRFC/IDocs

At the conclusion of these exercises you will be able to:

Write servers for outbound tRFC calls.

1-1 In this exercise, you will set up a server component that receives IDocs via tRFC.

1-2 Starting in SALE, create a logical system CA926-xx.

1-3 Using BD64, define a Model View CA926-xx.
Add message type MATMAS (use the Sender name given by your instructor, CA926-xx as the receiver).

1-4 Using BD82, generate a partner profile for Model View CA926-xx, Partner System CA926-xx, set the Packet Size to 1.

1-5 Copy the template for class IDocReceiver ("IDocReceiver Template.java") to your PC and rename the file to IDocReceiver.java

1-6 Extend the handleRequest method by writing the IDOC_DATA_REC_40 and IDOC_CONTROL_REC_40 tables out to HTML, using the TID passed in onCheckTID as part of the file names.

1-7 Start your server (supplied class TestIDocReceiver.java).

1-8 Using BD10, send an IDoc for Material P-100 to your system. Check the new HTML files on your PC.

Browser-based Applications: Contents

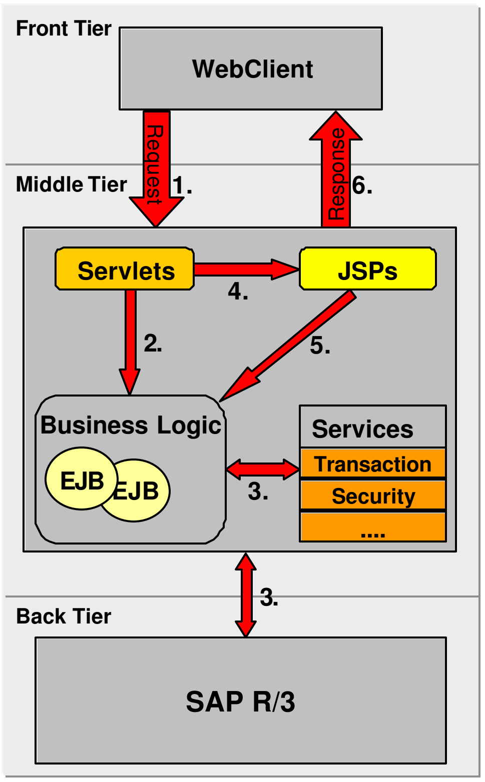
- The Common Programming Model
- Servlet Concepts
- Java Server Pages
- Tomcat

Browser-based Applications: Unit Objectives

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

- Describe the Common Programming Model used for Server-side Java Development
- Understand the Basic Concepts of Servlet Programming
- Create Java Server Pages
- Understand the Benefits of Components in BAPI Programming

Servlet Programming Model uses MVC Pattern



- 1. Client sends a request to the server.
- 2. Servlet (Controller) processes the request and calls the appropriate beans.
- 3. JavaBeans or Enterprise JavaBeans (Model) calculate the result accessing the services provided by the application server.
- 4. Servlet selects appropriate JSP to display the dynamic result.
- 5. JSP (View) retrieves result data from model and merges it dynamically into the HTML template.
- 6. Response is delivered to the client.

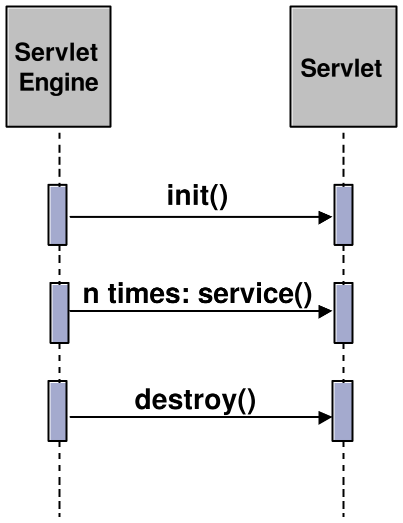
Servlets

- Serve as a controller for a web application.
- Parse parameters passed in from HTML clients.
- Process requests, e.g.
  - Invoke BAPIs.
  - Calculate intermediate results used for further BAPI calls.
- Return results
  - Provide the HTML pages directly to the web client.
  - Using a Java Server Page (JSP).

Servlets vs. Other Programming Models

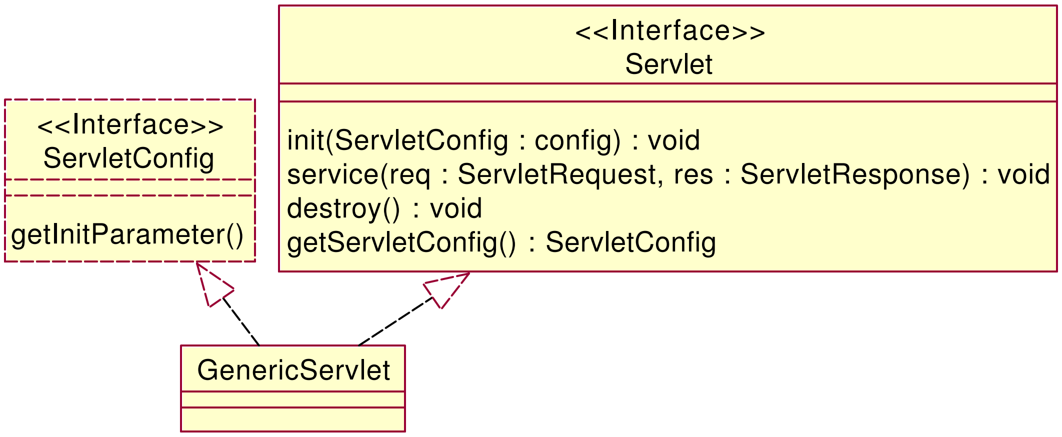
- **Advantages over CGI**
  - Does not run in separate process.
  - Stays in memory between requests.
  - Single instance for concurrent requests.
  - Runs in a secure environment (sandbox).
- **Advantages over NSAPI, ISAPI**
  - Portable (runs on every standard Java VM).
  - Scalability
    - Wide range of server implementations (NT -> OS390) available.
- **Java as programming language**
  - Object-oriented
  - Platform-independent

Architecture: Servlet Lifecycle



- Initialization
  - Server loads servlet and calls init().
  - Servlet accesses server configuration information for initialization.
- Handling of client requests
  - For every client request, the server calls the service() method.
  - Concurrent calls are handled in different threads (thread-safe code!).
- Destruction
  - Server calls destroy().

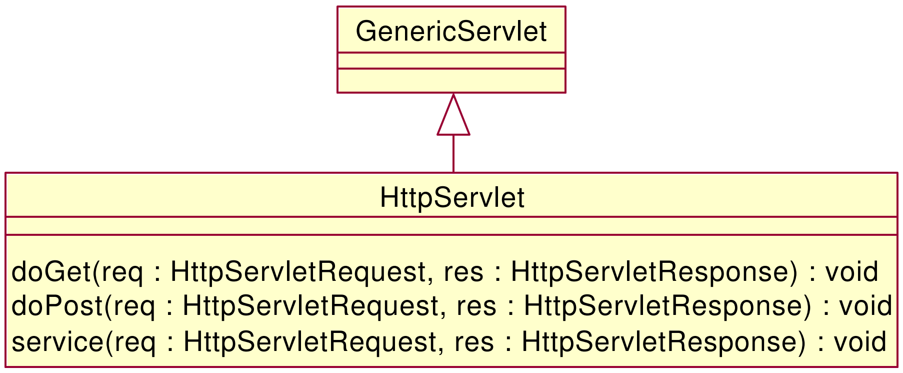
Architecture: GenericServlet



<<Interface>>
Servlet
init(ServletConfig : config) : void
service(req : ServletRequest, res : ServletResponse) : void
destroy() : void
getServletConfig() : ServletConfig
<<Interface>>
ServletConfig
getInitParameter()
GenericServlet

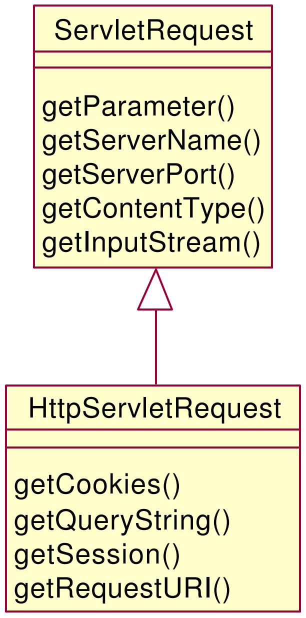
- **Default implementation of ServletConfig and Servlet interfaces**
  - **Provides life-cycle methods.**
  - **Provides access to configuration information defined by application server.**
- **Base-class for HttpServlet**

Architecture: HttpServlet


GenericServlet
HttpServlet
doGet(req : HttpServletRequest, res : HttpServletResponse) : void
doPost(req : HttpServletRequest, res : HttpServletResponse) : void
service(req : HttpServletRequest, res : HttpServletResponse) : void

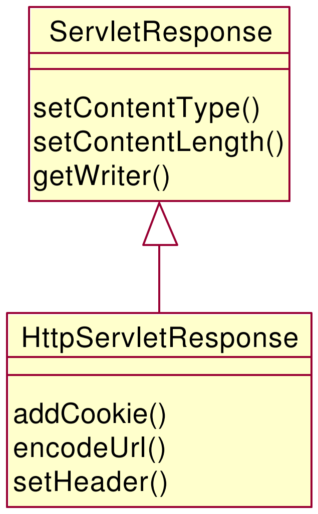
- **Base class for own HTTP specific servlets.**
- **Service() analyzes the HTTP request and dispatches to the specialized Java methods**
  - **doGet() handles client requests.**
  - **doPost() handles server respone.**

Architecture: ServletRequest



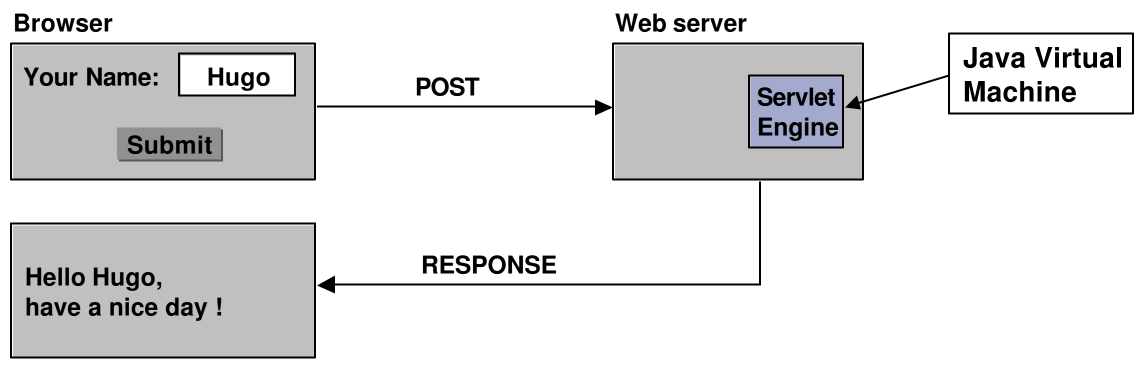
- **Encapsulates the client request.**
- **HttpServletRequest**
  - **Specialized object for servlets in HTTP environments**
    - **Query string**
    - **Cookies**
    - **HTTP session**
    - **...**

Architecture: ServletResponse



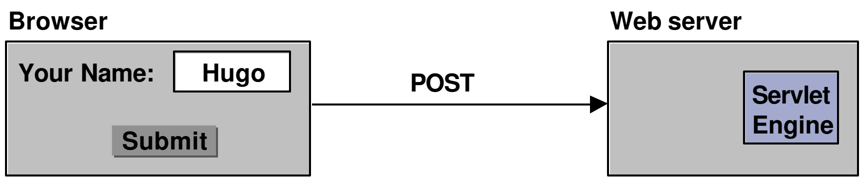
- Encapsulates the server response.
- HttpServletResponse
  - Specialized object for servlets in HTTP environments
    - Cookies
    - URLs
    - ...

Simple Servlet Scenario (1)



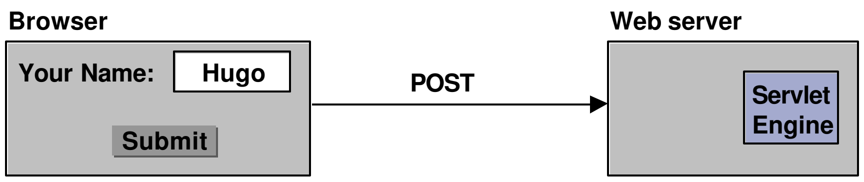
- Entering data into HTML form.
- Posting data to the Web server.
- Server processes the request using a servlet and sends a response.
- Browser receives and displays the HTML response.

Simple Servlet Scenario (2)



- The Browser sends a POST request
  - URL contains name of the servlet
    - e.g. http://demo/servlet/Hello
  - Body contains data entered by the user as name/value pairs
    - e.g. name of user: NAME=Hugo
- The servlet engine loads the class of corresponding servlet and ...
  - Creates instance calling the default constructor.
  - Calls the init() method.

Simple Servlet Scenario (3)



- The ServletEngine calls the service() method.
- service() analyzes the HTTP request and routes to the doPost() method.

```
 public void doPost(
HttpServletRequest req,
HttpServletResponse res)throws
ServletException, IOException
 {...}
```

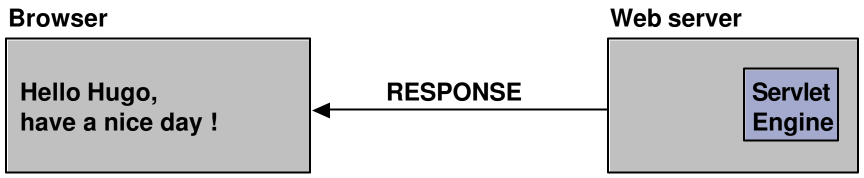
Simple Servlet Scenario (4)

Browser
Your Name:
Hugo
Submit
Web server
Servlet Engine

- The HttpServletRequest object provides information on the client request
  - getParameter(String) gets the value of the NAME parameter.

```
{[…]
 String name =
req.getParameter("NAME")
[…]}
```

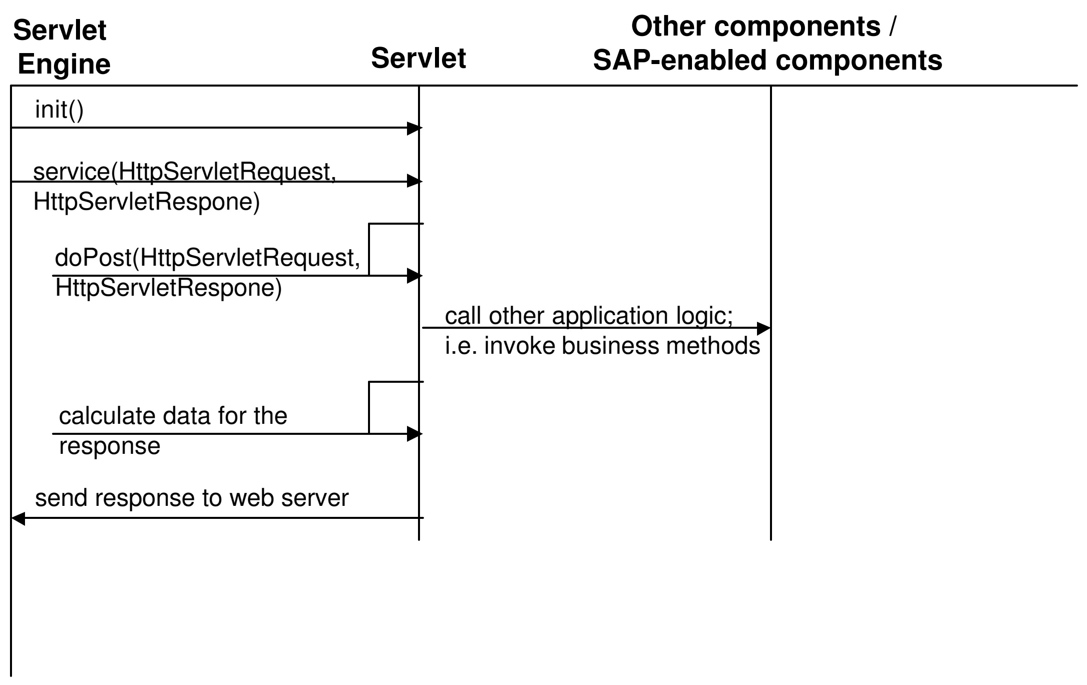
Simple Servlet Scenario (5)



- Servlet provides response to the client using the HttpServletResponse object.

```
{[…]
 res.setContentType("text/html");
 PrintWriter out = res.getWriter();
out.println("<HTML><HEAD><TITLE>");
out.println("Hello</TITLE></HEAD>");
out.println("<BODY>Hello " + name);
out.println("<BR>have a nice day!");
out.println("</BODY></HTML>");
out.close(); […] }
```

Overview: Simple Servlet Scenario


Servlet Engine
Servlet
Other components / SAP-enabled components
init()
service(HttpServletRequest, HttpServletRespone)
doPost(HttpServletRequest, HttpServletRespone)
call other application logic; i.e. invoke business methods
calculate data for the response
send response to web server

HTTP Sessions (1)

- An HTTP servlet session...
  - Maintains state across multiple HTTP-Client/HTTP-Server requests/responses.
  - Uses either cookies or URL rewriting.
  - Has a limited life-span.
- Server-side objects can be bound to an HTTP session
  - Data is stored on server-side.

HTTP Sessions (2)

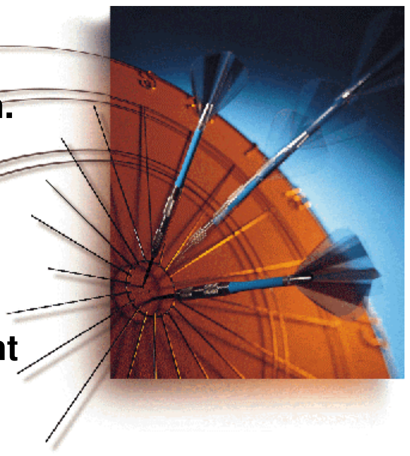
- Creating an HTTP Session
  - If no session object exists, a new session will be created.
- Binding a specific object to the HTTP session

```
HttpSession aSession = req.getHttpSession(true);
MyBean aBean = new myBean();
aBean.setMyText("Hello World");
aSession.putValue("myBean", aBean);
```

- Retrieving an existing HTTP session
  - The HttpServletRequest object returns an existing session object.
- Retrieving reference to a bound object

```
HttpSession aSession = req.getHttpSession(true);
MyBean aBean = (myBean)aSession.getValue("myBean");
```

Java Server Pages (JSP)



- Serves as a view for the web application.
- Support for scripting and tags
  - HTML
  - Embedded Java
- Separation of static and dynamic content
  - Separation of roles
- Supports reuse of components
  - JavaBeans and Enterprise JavaBeans

Simple JSP Scenario (1)

- **Example**

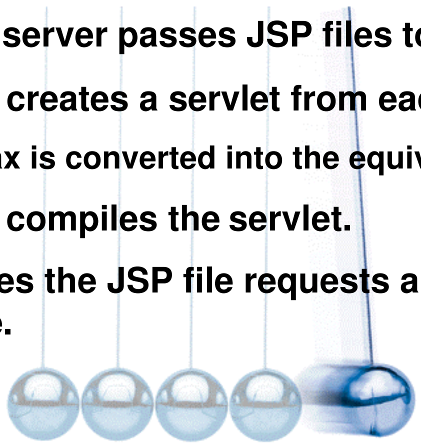
```
<HTML>
        <%@ import = "java.io.*" %>
        <BEAN NAME="aBean" TYPE="myclasses.Time"></BEAN> <BR>
        Year: <INSERT BEAN="aBean" PROPERTY="Year"></INSERT>
        <BR>
        Month: <%= aBean.getMonth()%>
        <% out.println("<BR>Day: " + aBean.getDay());%>
</HTML>
```

- **Result after calling the JSP from a Browser**

```
Year: 2000

Month: 02

Day: 05
```

Simple JSP Scenario (2)

- The Web server passes JSP file requests to the application server.
- The application server passes JSP files to the JSP engine.
- The JSP engine creates a servlet from each JSP file
  - The JSP syntax is converted into the equivalent Java code.
- The JSP engine compiles the servlet.
- Servlet processes the JSP file requests and provides server response.



Simple JSP Scenario (3)

- **JSP directive**
  - **Passes information to the JSP engine.**
    - **i.e. <%@import file= "package.class"> packages to be included**
- **HTML tags**
  - **Fixed template data**
  - **Not recognized by the JSP engine and passed without changes to the client.**

```
<HTML>

   <%@ import = "java.io.*" %>
   <BEAN NAME="aBean" TYPE="myclasses.Time"></BEAN> <BR>
   Year: <INSERT BEAN="aBean" PROPERTY="Year"></INSERT>
   <BR>
   Month: <%= aBean.getMonth()%>
   <% out.println("<BR>Day: " + aBean.getDay());%>

</HTML>
```

Simple JSP Scenario (4)

- JSP Tag
  - Typically implemented as standard tags and processed by the JSP engine.
  - <BEAN> tag instantiates the myclasses.Time bean
    <INSERT> tag inserts value of CurrentTime property into HTML page
- Expressions
  - JSP engine evaluates expression between <%= expression%>
  - Expression must be valid in the scripting language used in the page.

```
<HTML>

   <%@ import = "java.io.*" %>
   <BEAN NAME="aBean" TYPE="myclasses.Time"></BEAN> <BR>
   Year: <INSERT BEAN="aBean" PROPERTY="Year"></INSERT>
   <BR>
   Month: <%= aBean.getMonth();%>
   <% out.println("<BR>Day: " + aBean.getDay());%>

</HTML>
```

Simple JSP Scenario (5)

- **Scriptlet**
  - **A script that performs functions usually not supported by tags.**
  - **Native scripting language for JSP is Java.**
  - **Can use predefined variables e.g.**
    - **out: class defined by java.io.PrintWriter**
    - **request: class defined by javax.servlet.http.HttpServletRequest**
    - **response: class defined by javax.servlet.http.HttpServletResponse**

```
<HTML>

    <%@ import = "java.io.*" %>
    <BEAN NAME="aBean" TYPE="myclasses.Time"></BEAN> <BR>
    Year: <INSERT BEAN="aBean" PROPERTY="Year"></INSERT>
    <BR>
    Month: <%= aBean.getMonth()%>
    <% out.println("<BR>Day: " + aBean.getDay());%>

</HTML>
```

Tomcat Overview

- Reference implementation for Servlets and JSP
- Open source
- Contains its own HTTP server and can be used with popular web servers.

Tomcat Installation

- Download from http://jakarta.apache.org/
- Unzip
- Set environment variable JAVA_HOME to point to your JDK.
- Set environment variable TOMCAT_HOME to point to Tomcat.
- Start via startup.bat
- Verify installation: http://localhost:8080/examples/servlets/index.html
- Stop via shutdown.bat

Application Setup

- Create a directory <app> under <tomcat>\webapps
- Put your main HTML page here.
- Create a directory "Web-inf" under <app>.
- Put the "web.xml" file here.
- Create a directory ”jsp" under <app> and put all your JSP files here.
- Create a directory "classes" under "Web-inf" and put all your class files and resources here.
- Create a directory "lib" under "Web-inf" and put all your jar file (e.g. jCO.jar) here.
- Verify installation: http://localhost:8080/<app>/index.html

Sample web.xml File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
    <servlet>
        <servlet-name>
            SapServlet
        </servlet-name>
        <servlet-class>
            SapServlet
        </servlet-class>
    </servlet>
</web-app>
```

Sample index.html File

```
<HTML>
<HEAD><TITLE>Company Code List</TITLE></HEAD>
<BODY>
  <FORM ACTION="/ca926/servlet/SapServlet" METHOD="POST">
    <INPUT TYPE="submit" VALUE="Get Company Code List">
  </FORM>
</BODY>
</HTML>
```

Browser-based Applications: Unit Summary

- Servlets provide an easy way for using BAPIs in a muli-tier environment.
- JSP separate visual logic from programming logic.
- JSP support componentized development.

Unit: Optional Exercise

Topic: BAPI Return Code Handling

At the conclusion of these exercises you will be able to:

Build a method that checks for errors in BAPIs genericly.

1-1 Create a class SAPServices6 that extends SAPServices5 with a static method using the following signature:

```
public static boolean
isBapiReturnCodeOkay(com.sap.mw.jco.JCO.Record object)
```

1-2 Implement this method to find out whether a BAPI call succeeded completely regardless of whether a structure or table was returned and which SAP dictionary structure the Return parameter was based on.

1-3 Modify one of your existing classes to check for errors using your new method.

C O N T E N T S

November ‘97

Created by D. Han Tran

Basis/ABAP+GUI

1. RFC IN SAP SYSTEMS

From R/2 release 5.0 onwards and in all R/3 releases, **CALL FUNCTION** is an integral part of the ABAP/4 language. It is used to perform a function (function module in ABAP/4) within the same system (R/2 or R/3).

REMOTE FUNCTION CALL (RFC) is an extension of CALL FUNCTION in a distributed environment. Existing function modules can be executed from within a remote system (R/2 or R/3) via an RFC call:

```
CALL FUNCTION       <FUNCTION_NAME>
      DESTINATION   <DEST>
      EXPORTING     -----------
      IMPORTING     -----------
      TABLES        -----------
      EXCEPTIONS    -----------.
```

The ‘destination’ parameter points to an entry in the SAP table **RFCDES** (R/3) or the SAP tables **RFCD** and **XCOM** (R/2). This entry can be defined with **sm59** in R/3 or **tm31** in R/2 and contains all necessary parameter to connect to and login onto the destination SAP system.

RFC can be used between SAP systems (within R/3 and/or R/2).

With R/2 in an IBM environment, RFC is currently possible only with CICS as a DC system from 5.0D onwards. But with IMS, RFC is only available with IMS from 4.1 onwards, which provides a complete support of the LU6.2 protocol (via MVS/APPC).

With R/2 in an SNI environment, RFC is currently available only with supported TCP/IP on BS2000. UTM with UTM-VTV protocol is NOT needed for this communication.

2. RFC WITH EXTERNAL SYSTEMS

SAP provides the **RFC API** in the form of C-routines included in an **RFC library** on several external systems such as Windows, OS/2, McIntosh and all R/3-based UNIX platforms. With this RFC API, it is available to use the RFC functionality between a SAP System (R/3 from release 2.1 and R/2 from release 5.0D onwards) and an external program on the above platforms. The remote function may be provided either in a SAP System or in an external program.

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

The RFC library is **forward and backward compatible**: An RFC library delivered with a higher version of an R/3 system can be used to work with a lower R/3 system and vice cersa.

There are two kinds of RFC programs: **RFC client** and **RFC server**.

RFC client is the instance that calls up the Remote Function Call to execute the function provided by an **RFC server.** In the following, the functions that can be executed remotely will be called **RFC functions** and the functions provided via RFC API will be called **RFC calls.**

3. TECHNICAL DESCRIPTION & IMPORTANT FEATURES

```
--------------------------------------------          --------------------------------------------
|  SAP  R/2                                |          |  SAP  R/2                                |
|  SAP  R/3                                |          |  SAP  R/3                                |
|  External System                         |          |  External System                         |
|                                          |          |                                          |
|                                          |          |                                          |
|         ABAP-Program                     |          |         ABAP-Program                     |
|         External Program                 |          |         External Program                 |
--------------------------------------------          --------------------------------------------
                      |                                                     |
                      |                                                     |
ABAP-Call             |                               ABAP-Call             |
     or               |                                    or               |
RFC-API               |                               RFC-API               |
                      |                                                     |
                      |                                                     |
--------------------------------------------          --------------------------------------------
|                                          |          |                                          |
|     RFC component in SAP                 |          |     RFC component in SAP                 |
|                 or                       |          |                 or                       |
|     RFC library  on ext. system          |          |     RFC library  on ext. system          |
|                                          |          |                                          |
--------------------------------------------          --------------------------------------------
                                |                                     |
                                |               CPI-C                 |
                                |                                     |
                      ----------------------------------------------------------------------
                      |                                                                    |
                      |                                                                    |
                      |                        SAP   gateway                               |
                      |                                                                    |
                      |                                                                    |
                      ----------------------------------------------------------------------
```

GETTING CONNECTED

- Two-stage connection:
  1. Connection from an RFC client to a SAP gateway
  2. Connection from the SAP gateway to an RFC server

- Different ways to start or connect to an RFC server program
  1. An RFC server program can be started by the SAP gateway or by the currently using SAPGUI or by the currently running application server.
  2. From Release 3.0C onwards, an RFC server program can start running before, it must register at a SAP gateway (at least 3.0C) and then wait for RFC requests from SAP systems.

IMPORTANT FEATURES

■ Data Format

Up to Release. 3.0F, only **homogeneous** structures or tables are supported. They can only consist of character-like fields (ABAP-type **C**, **D**, **T**, **N**) or fields which will NOT be converted (ABAP-type **X** or **P**). Integer (ABAP-type **I**) or float (ABAP-type **F**) fields can only be transferred as single fields (IMPORT/EXPORT parameters).

From 3.1G onwards, **heterogeneous** structures or tables will be supported but not with the 16-bit RFC library on Windows and not for Visual Basic using the RFC library directly.

■ Data Compression

From release 3.0 onwards, RFC data will be compressed before sending if both client and server system are capable of this. Otherwise, only the blanks at the end of a table line will be truncated.

■ Data Conversion

The SAP system or the RFC library will convert received data to match its own code page if the two code pages are not the same. Some standard (1100, 0100, ...) code pages are already implemented in the RFC library. It can also use a convert file defined by the environment variable PATH_TO_CODEPAGE.

With the transaction **sm59**, these files can be created and downloaded in a directory accessible from the currently running R/3 application server.

■ Limitations of Data in one RFC Function

RFC library delivered with Release	< **2.1K** / **2.2E**:	**4 MB**
	>= **2.1K** / **2.2E**:	**64 MB**
	>= **3.0**:	**None**

■ Multi-threaded RFC Library on 32-bit Windows (NT/95)

From Release 3.1G onwards, the 32-bit RFC library on Windows (NT/95) will be multi-threaded. Your RFC program can also have more than one thread working with different RFC connections. But the rfc_handle of an RFC connection can only be used in the same thread where this connection is initiated (via **RfcOpen**) or accepted (via **RfcAccept**).

■ No Shared Library on R/3-based UNIX-platforms

SAP does not support shared library on R/3-based UNIX-platforms

■ Visual Basic 3.0 and 4.0/5.0 on 32-bit Windows (NT/95)

The 16-bit RFC Library on 16-bit Windows supports Visual Basic 3.0.

From 3.1G onwards, the 32-bit RFC Library on 32-bit Windows (NT/95) supports VB 4.0/5.0. Hereby, a VB 4.0/5.0-program must use **RfcDefineImportParam** and **RfcGetImportParam** instead of **RfcAddImportParam** in a VB 3.0 program (see saprfc.h for more details).

4. TECHNICAL REQUIREMENTS

4.1. External System

External systems must support TCP/IP.

Currently, there is an RFC SDK for following systems available:

- **R/3-based UNIX-platforms**
- **Windows 3.1 /3.11**
- **Windows NT / 95**
- **OS/2**
- **Apple (McIntosh)**
- **AS/400**
- **VMS**

The RFC SDK for the supported platforms contains the following libraries and include files:

- **saprfc.h:**

This include file contains all data types, structures and the prototypes of the RFC calls.

- **sapitab.h:**

This include file contains all RFC calls for working with internal tables.

- **RFC Library:**

According to the supported platforms a static library or one or more dynamic link libraries (DLL) are provided.

4.2. R/3 system

Available from Release **2.1** onwards.

For RFC between external systems and R/3, there are no additional requirements to the R/3 System.

Contrary to this, **RFC between SAP R/2 in an IBM environment and SAP R/3 or external system** requires an SAP gateway running on a machine that supports the SNA LU6.2 protocol for the IBM host. The SNA product must be installed on this machine and the SAP gateway must operate with this product.

The following SNA products are currently supported:

- **SNA services**or **SNA server** on IBM-AIX machines
- **SNAplusLink** on HP-UX machines
- **Communications Manager** on OS/2
- **SNA server** on Windows NT
- **SNALink SNA peer-to-peer 8.0** on SUN machines
- **TRANSIT-SERVER** and **TRANSIT-CPIC** on SIEMENS-SINIX machines

RFC between **SAP R/2 in an SNI environment** and SAP R/3 or external system requires a SAP gateway running on BS2000 (TCP/IP must be installed on BS2000) and for outgoing RFC calls (from ABAP) another SAP gateway is needed for starting RFC server programs.

4.3. R/2 system

IBM host (CICS):	R/2 Release 5.0D onwards with the following components:	
	082	Communication via Remote Function Call (RFC)
	153	SAP Intersystem Communication
	080	Host communication with DOS, OS/2
or	081	Host communication with other LU6.2 systems
IBM host (IMS):	IMS Release 4.1 onwards with MVS/APPC	
	Same SAP components as for CICS required	
SNI host:	R/2 Release 5.0D onwards with the following components:	
	082	Communication via Remote Function Call (RFC)
	153	SAP Intersystem Communication

083 Host communication via TCP/IP (BS2000)

5. RFC CLIENT PROGRAM

5.1. Introduction

```
-------------------------------------------          -----------------------------------------------------
|  External System                        |          |  SAP  system                                      |
|                                         |          |                                                   |
|                                         |          |                                                   |
|  RFC client program                     |          |  Function Module                                  |
|                                         |          |                                                   |
|  RfcOpen                    ----------------------→|                                                   |
|                                         |          |          -------------------------------          |
|  RfcCallReceive  ('ABC')   -------------------------------→ | FUNCTION ABC.       |          |
|                                         |          |          |                    |          |
|                                         |          |          |                    |          |
|                                         |          |          |                    |          |
|                            ←-------------------------------- |  ENDFUNCTION.      |          |
|  RfcClose                               |          |          -------------------------------          |
|                                         |          |                                                   |
-------------------------------------------          -----------------------------------------------------
```

DIALOG/CPIC user

- Login onto a R/3 system is possible fo both DIALOG user and CPIC user.
- Login onto a R/2 system is only available with CPIC user.

Getting connected with the RFC library prior to 3.0C

There are different ways and conditions to establish an RFC connection to a SAP system (R/2 or R/3).

1. Using a **local 'sideinfo'** file
2. Not using of a **local 'sideinfo'** file but using a **'sideinfo'** file for the SAP gateway
3. Not using of any **'sideinfo'** file

- **'sideinfo'** file is needed for communication via CPI-C.

- For communication via SNA LU6.2 protocol, a **‘sideinfo’** file for the SAP gateway is always required (R/2 in IBM-environment).

- For communication with an R/2 in an SNI environment, the SAP gateway must run on a BS 2000-host. There are otherwise no differences between getting connected to an R/2 system (SNI) or to an R/3 system.

Getting connected with the RFC library from 3.0C onwards

An RFC connection can be established via an entry in the **‘saprfc.ini’** file. Using this feature a ‘sideinfo’ file is not needed, except for connections to an R/2 (IBM).

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

Getting connected with the RFC library from 4.0A onwards

To establish an RFC connection with the RFC library from 4.0A onwards, an RFC client can also use the **RfcOpenEx** where and all necessary connection data are defined as parameters in this call. It is recommended to use this call instead of RfcOpen, RfcConnect, … for a better error handling.

5.2. “Load Balancing” Feature of R/3 Release 3.0 onwards

From release 3.0 onwards, the RFC library supports the feature to connect to an R/3 system via “Load Balancing” principle. With this functionality, the RFC library tries to connect to an application server with the least load within a group of predefined application servers.

This feature has the following advantages:

- The load in an R/3 system is distributed to different application servers.
- Using Load Balancing, the application server as RFC server system will be determined at run time. Therefore, RFC connections are thus independent of a specific application server. (Till now, an RFC connection could only be established to a specific application server.)
- Only the host name of the R/3 message server and its port number are required in the ‘hosts’ file and the ‘services’ file.

5.3. Some Important RFC Calls

- **RfcEnvironment**

Together with the option for allocating storage in the RFC library, an ERROR HANDLING function must be implemented in any RFC client program in order that the RFC library can call up this function for error handling when unexpected RFC errors occur.

- **RfcOpen/RfcOpenEx**

This call is used to establish an RFC connection to the server system. Information about how to reach an SAP system must be provided either by this RFC call, in the 'sideinfo' file or in the 'saprfc.ini' file. The RfcOpenEx is only available with the RFC library from 4.0A onwards.

- **RfcCallReceive**

This call activates the required function module in SAP R/2 or R/3 and the control will be returned to the RFC client program after the function module in SAP system has been executed.

- **RfcCall, RfcReceive and RfcListen**

If the external program only wants to call up a function module and does not want to wait for the result, **RfcCall** can be submitted at first and then **RfcReceive** or **RfcListen** later.

With **RfcReceive**, the client program always waits for the result. Contrary to this, **RfcListen** returns back immediately. If an event has occurred, **RfcReceive** must be used to receive the RFC data.

- **RfcDispatch and RfcInstallFunction/RfcInstallFunctionExt**

In an RFC client program, the **RfcDispatch** and **RfcInstallFunction**/**RfcInstallFunctionExt** can be used to realize the 'call-back' feature. It is also possible to use **RfcGetName** instead of **RfcDispatch**.

See chapter 6 and 8 for more details.

- **RfcLastError**

Use to get more details after an RFC error is returned by any RFC call.

- **RfcAbort**

This call can be used to terminate an RFC connection. An error text can be passed on to the partner system/program. This error text is then available as an Error_Message in the partner system or program (only useful in a 'call-back'-case).

- **RfcClose**

With a specified rfc_handle (returned by successful **RfcOpen**) the current RFC connection will be closed. With RFC_HANDLE_NULL , all RFC connections in this client program will be closed. **RfcClose** must also be submitted after a failed RFC call in order that all internally used control areas except areas for internal tables are released.

- **ItCreate, ItFree and ItDelete**

Storage allocated by the RFC client program using **ItCreate** must be released using the call **ItDelete**. **RfcClose** does not free storage automatically and **ItFree** just initiates the internal tables.

- **RfcGetAttributes**

Available from 3.0E onwards for getting some information about a specified RFC connection.

- **RfcInstallStructure**

Available from 3.1G onwards to inform the RFC library about the structure of received data (Importing parameters and internal tables with heterogeneous structures).

5.4. Programming Examples

Following sections only describe some examples in short form.

See the delivered RFC client program ‘srfctest.c’ for more details.

5.4.1. RFC Client Program with Sending of Internal Tables

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

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

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

if (rfc_handle == RFC_HANDLE_NULL)           /* Check return code                      */
{
  rfc_error_handling("RfcOpen");             /* Get specific error via RfcLastError    */
  return 1;                                  /* and handle error                       */
}

tables[0].name  = “ITAB1000”;                /* Must fit with definition in SAP-FM     */
tables[0].nlen   = 8;                        /* Length of name                         */
tables[0].type   = TYPEC;                    /* Only character (homogen structure)     */
tables[0].leng   = 1000;                     /* Length of a table line                 */
tables[0].itmode = RFC_ITMODE_BYREFERENCE;   /* Recommended                            */

tables[0].ithandle = ItCreate(…);            /* Allocate storage for internal table    */

if (rfc_rc != RFC_OK)                        /* Check return code                      */
{
  rfc_error_handling("ItCreated");                   /* Get specific error via RfcLastError      */
  return 1;                                  /* and handle error                       */
}

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

rfc_rc = RfcCallReceive(...);                /* Call up function module with the filled */
```

```
                                             /* internal table above               */

if (rfc_rc != RFC_OK)                        /* Check return code                  */
{
  rfc_error_handling("RfcCallReceive");      /* Get specific error via RfcLastError */
  return 1;                                  /* and handle error                   */
}

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

ItDelete (…);                                /* Free storage of internal table     */
```

```
/* Fill internal table with text as requested */
void fill_table(ITAB_H itab_h, int table_leng)
{
  int     linenr;                                  /* Actual line number in table        */
  int     lineleng;                                /* Length of a table line             */
  char    *ptr;                                    /* Pointer to a table line            */
  char    table_data[] = "This is a test";         /* Text for test                      */

  if (table_leng == 0) return 0;                   /* Table with no entry                */

  lineleng = ItLeng(itab_h);                       /* Determine length of a table line   */

  for (linenr = 1; linenr <= table_leng; linenr++) /* Fill table as requested            */
  {
    ptr = (char *) ItAppLine(itab_h);              /* Get address of next line           */

    if (ptr == NULL)                               /* Check return code                  */
    {
      printf("\nMemory insufficient\n");           /* Output error message and           */
      exit(1);                                     /* terminate program                  */
    }
    memcpy(ptr, table_data, lineleng);             /* Transfer data to internal table    */
  }
  return;                                          /* Back to main program               */
}
```

5.4.2. RFC Client Program with Receiving of Internal Tables

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

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

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

if (rfc_handle == RFC_HANDLE_NULL)         /* Check return code                      */
{
  rfc_error_handling("RfcOpen");           /* Get specific error via RfcLastError    */
  return 1;                                /* and handle error                       */
}

tables[0].name  = “ITAB1000”;              /* Must fit with definition in SAP-FM     */
tables[0].nlen    = 8;                     /* Length of name                         */
tables[0].type    = TYPEC;                 /* Only character                         */
tables[0].leng    = 1000;                  /* Length of a table line                 */
tables[0].itmode = RFC_ITMODE_BYREFERENCE;   /* Recommended                          */

tables[0].ithandle = ItCreate(…);          /* Allocate storage for internal table    */

if (rfc_rc != RFC_OK)                      /* Check return code                      */
{
  rfc_error_handling("ItCreated");                 /* Get specific error via RfcLastError    */
  return 1;                                /* and handle error                       */
}

rfc_rc = RfcCallReceive(...);              /* Call up function module and receive    */
                                           /* data in created internal table above   */

if (rfc_rc != RFC_OK)                      /* Check return code                      */
{
  rfc_error_handling("RfcCallReceive");    /* Get specific error via RfcLastError    */
  return 1;                                /* and handle error                       */
}
```

```
display_table(table[0].ithandle);        /* Output received int. table on screen    */

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

ItDelete(…);                             /* Free storage of internal table           */
```

```
/* Output received internal table on screen */
void display_table(ITAB_H itab_h)
{
  int     linenr;                        /* Actual line number in table          */
  int     lineleng;                      /* Length of a table line               */
  char    *ptr;                          /* Pointer to a table line              */
  char    table_data[8193];              /* Max. length of a int. table: 8192 B  */

  lineleng = ItLeng(itab_h);             /* Get length of a table line           */

  for (linenr = 1; ; linenr++)           /* Loop at internal table               */
  {
    ptr = (char *) ItGetLine(itab_h);    /* Get address of next line             */

    if (ptr == NULL) break;              /* NULL:  End of table reached          */

    memcpy(table_data, ptr, lineleng);   /* Read a table line into buffer        */

    table_data[lineleng] = '\0';         /* Set string end in buffer for output  */

   printf("'%s'\n", table_data);         /*  Ouput on screen                     */
  }
  return;                                /* Back to main program                 */
}
```

5.5. Getting Connected to R/2 System

As described above, establishing an RFC connection to an R/2 system in an SNI environment is similar to an R/3 system, except the SAP gateway must run on BS 2000-host.Therefore, the following sections just describe how to connect to an R/2 system in IBM environments.

For establishing an RFC connection to an R/2 system in IBM environments (CICS or IMS as DC system) there is always a sideinfo for the SAP gateway necessary!

Using the RFC library from 4.0A onwards, it is recommended to use the **RfcOpenEx** instead of RfcOpen or RfcOpenExt for a better error handling. With this call, the RFC client can work with the saprfc.ini or it can define all necessary connection data as parameters in this call.

5.5.1. Programming example for using saprfc.ini

```
Char * cn_pm[]="dest=K50 client=000 user=RFCTEST passwd=SECRET lang=E";

RFC_ERROR_INFO_EXE   error_info;           /* Error specification              */

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

rfc_handle = RfcOpenEx(cn_pm, &error_info);  /* Open RFC connection            */
.....
```

Entry in saprfc.ini

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

Example on Windows: **set RFC_INI=d:\rfctest\saprfc.ini**

The entry in saprfc.ini for the above program can be defined as follows:

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

Entries in sideinfo for the SAP gateway

1. **Example for a SAP gateway running on IBM-AIX machines**

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

DEST=K50_2
LU=K50T2
TP=X1SA

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

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

2. **Example for a SAP gateway running on HP-UX machines**

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

- Specification about LOCAL_LU is not required if at least one local LU is defined in the **LU pool** in the SNA-configuration on HP-UX.

- The mode 'LU62SAP1' must be defined in the SNA-configuration on HP-UX and in IBM-Host.

5.5.2. Programming example for not using saprfc.ini

```
Char * cn_pm[]="type=2 dest=K50 gwhost=is0001 gwserv=sapgw00 client=000 user=RFCTEST
passwd=SECRET lang=E";

RFC_ERROR_INFO_EXE   error_info;             /* Error specification               */

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

rfc_handle = RfcOpen(cn_pm, &error_info);    /* Open RFC connection               */
.....
```

Entries in sideinfo for the SAP gateway

s. 5.5.1.

5.6. Getting Connected to R/3 System

Contrary to SAP R/2 (IBM), for an RFC connection to an R/3 system the SAP gateway does not need a specific entry in its ‘sideinfo’ file if all necessary parameters are specificied in the local ‘sideinfo’ file or they are defined in the RFC client programs.

Using the RFC library from 4.0A onwards, it is recommended to use the **RfcOpenEx** instead of RfcOpen or RfcOpenExt for a better error handling. With this call, the RFC client can work with the saprfc.ini or it can define all necessary connection data as parameters in this call.

5.6.1. Programming example for using saprfc.ini

Example for connect to an R/3 using Load Balancing

```
Char * cn_pm[]=“dest=BIN client=000 user=RFCTEST passwd=SECRET lang=E“;

RFC_ERROR_INFO_EXE   error_info;             /* Error specification                */

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

rfc_handle = RfcOpenEx(cn_pm, &error_info);  /* Open RFC connection                */
.....
```

Entry in saprfc.ini

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

Example on Windows: **set RFC_INI=d:\rfctest\saprfc.ini**

The entry in saprfc.ini for the above program can be defined as follows:

```
/* Connect to an R/3 using LOAD BALANCING */
DEST=BIN
TYPE=B
MSHOST=hs0311
```

```
R3NAME=BIN
GROUP=PUBLIC
RFC_TRACE=0
ABAP_DEBUG=1
USE_SAPGUI=1
```

The features ‘RFC with SAPGUI’ or ‘RFC with ABAP-debug’ are only available with an R/3 from 3.0C onwards.

s. chapter 7 for more details about the saprfc.ini.

5.6.2. Programming example for not using saprfc.ini

Example for connect to a specific R/3 application server

```
Char * cn_pm[]=“type=3 ashost=hs0311 sysnr=53 client=000 user=RFCTEST passwd=SECRET lang=E“;

RFC_ERROR_INFO_EXE   error_info;          /* Error specification              */

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

rfc_handle = RfcOpen(cn_pm, &error_info); /* Open RFC connection              */
.....
```

5.7. RFC using SAPGUI

From release 3.0C onwards, it is possible for an RFC client program to call ABAP/4 function modules, which are using **'Dynpros'** or **SAP Graphics**. In particular, it is available to call complete SAP transactions from external programs.

For using this functionality, it is necessary to have a SAPGUI of version 3.0C or later installed on the external system where the RFC client program is running.

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

- **Using RfcOpen**

Call **RfcOpen** with the special mode RFC_MODE_VERSION_3 and set the field **use_sapgui** to a non-zero value in the structure RFC_CONNOPT_VERSION_3.

- **Using SYSTEM_ATTACH_GUI in R/3**

An alternative way which can be used without modifying the coding of the RFC client program is to call the function module **SYSTEM_ATTACH_GUI** (without any parameters) within the involved ABAP/4 function module before using any SAPGUI functionality.

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

Another way to do this is to define an entry in the **'saprfc.ini'** file which includes all necessary connection parameters and the RFC specific parameter USE_SAPGUI (=1). An RFC client program can then call **RfcOpen** with the mode RFC_MODE_PARAMETER and a destination pointed to this defined entry. See chapter 7 for more details.

Restrictions

- This new feature is available on all supported UNIX platforms (Motif).

- On Windows NT or Windows 95, it is only available with the 32-bit RFC library together with a 32-bit SAPGUI. In particular, this is not supported with the 16-bit RFC library or with 16-bit SAPGUI (Windows 3.x and the 16-bit subsystems of Windows NT or Windows 95).

- On Windows NT or Windows 95, the SAPGUI program, its DLL's and its auxiliary programs can be installed anywhere. However, you must have started SAPGUI once before it can be started automatically by RFC, since the SAPGUI program must register itself in the Windows registry.

- On UNIX-systems, the SAPGUI program must be installed on the default 'SAP path' **/usr/sap/<system name>/SYS/exe/run**.

5.8. RFC using ABAP Debugger

From release 3.0C onwards, the full functionality of the ABAP/4 debugger can be used while developing or testing an application using RFC.

For using this functionality, it is necessary to have a SAPGUI of version 3.0C or later installed on the external system where the RFC client program is running.

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

- **Environment variable RFC_DEBUG**

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

- **Set -debug on the command line**

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

- **Set the trace field to 'D' or 'E'**

If you can modify the coding of the RFC client program you are using you can set the trace field in the structure RFC_OPTIONS to 'D' or 'E', before calling RfcOpen.

  - 'D': debug without activate RFC trace
  - 'E': debug with activate RFC trace

- **Using the ‘saprfc.ini’ file**

Another way to do this is to define an entry in the **'saprfc.ini'** file which includes all necessary connection parameters and the RFC specific parameter ABAP_DEBUG (=1). An RFC client program can then call **RfcOpen** with the mode RFC_MODE_PARAMETER and a destination pointed to this defined entry. See chapter 7 for more details.

Restrictions

- This new feature is available on all supported UNIX platforms (Motif).

- On Windows NT or Windows 95, it is only available with the 32-bit RFC library together with a 32-bit SAPGUI. In particular, this is not supported with the 16-bit RFC library or with 16-bit SAPGUI on Windows 3.x or on the 16-bit subsystems of Windows NT or Windows 95).

- On Windows NT or Windows 95, the SAPGUI program, its DLL's and its auxiliary programs can be installed anywhere. However, you must have started SAPGUI once before it can be started automatically by RFC, since the SAPGUI program must register itself in the Windows registry.

- On UNIX-systems, the SAPGUI program must be installed on the normal 'SAP path' /usr/sap/<system name>/SYS/exe/run.

6. RFC SERVER PROGRAMS

6.1. Introduction

```
--------------------------------------------          ------------------------------------------------------
|  SAP system                             |           |  External  system                                  |
|                                         |           |                                                    |
|  ABAP program                           |           |  RFC server program                                |
|                                         |           |                                                    |
|  Call Function 'ABC'                    |           |                                                    |
|         Destination 'DEST'              |           |                                                    |
|         .......                ----------------------------> main()                                       |
|                                         |           |  {                                                 |
|                                         |           |         RfcAccept                                  |
|                                         |           |                                                    |
|                                         |           |         RfcInstallFunction('ABC',                  |
|                                         |           |                            abc_function, ...);     |
|                                         |           |         loop in                                    |
|                                         |           |                                                    |
|                                         |           |             RfcDispatch                            |
|                                         |           |                                                    |
|                                         |           |         until rfc_error                            |
|                                         |           |                                                    |
|                                         |           |         RfcClose                                   |
|                                         |           |                                                    |
|                                         |           |         .....                                      |
|                                         |           |  }                                                 |
|                                         |           |                                                    |
|                                         |           |  RFC_RC abc_function(rfc_handle)                   |
|                                         |           |  {                                                 |
|                                         |           |         RfcGetData                                 |
|                                         |           |         ......                                     |
|                                         |           |         RfcSendData                                |
|                                         |           |                                                    |
|                                         |           |  }                                                 |
--------------------------------------------          ------------------------------------------------------
```

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

Alternatively, the RFC server program can use **RfcGetName** to identify the name of the required RFC function and then dispatch itself to the function supported.

6.2. “Registering” Feature of R/3 Release 3.0C onwards

In **Releases prior to 3.0C,** RFC server programs can only be started by a SAP gateway, by currently using SAPGUI or by currently running application server.

From **release 3.0C onwards** an RFC server program can be started earlier. It must register at a SAP gateway and then wait for incoming RFC call requests.

This feature has the following properties:

- A RFC server program registers itself under a **Program-ID** at a SAP gateway (from 3.0C onwards) and **not** for a specific SAP system (R/2 or R/3).

- If an RFC call request from any R/2 or R/3 is transferred to this SAP gateway with the option **‘connect to an already registered program’** with the same **Program-ID**, it will be connected to this RFC server program.

- From R/2, this option will be used by setting **PROTOCOL=R** in the ‘sideinfo’ file for the ‘gwhost’ program.

- In R/3, the destination must be defined with **sm59**, connection type **‘T’** and using ‘**Register Mode’**. Moreover**,** the SAP gateway in this entry should be specified and fit with the SAP gateway at which the RFC server program is registered.

- After executed an RFC function, the RFC connection will be closed and if this RFC server program works with **RfcDispatch in a loop** (recommended!) it will be automatically registered again at the same SAP gateway under the same **Program-ID** and can then wait for next RFC call requests from the same or other SAP systems.

- With this new functionality, an RFC server program can be run on all Windows-PC’s and there are no restrictions about starting server programs on machines without supporting of **Remote Shell**. Moreover, it is now available and easy to run them in debugging mode.

For using this, there are no changes in ABAP programs needed. Only the destination in a ABAP-program must have the new feature defined with **sm59** as above.

On external systems there are two different ways to use this functionality:

- **Without changing the source coding**

All correct running RFC server programs can work with this new feature without changing the source coding. They must only be started with some more parameters needed to connect to the SAP gateway.

■ Using the ‘saprfc.ini’ file

The RFC call **RfcAccept** must include a destination pointed to an entry in the ‘saprfc.ini’ file with **type ‘R’** and contains some parameters needed to connect to the SAP gateway.

6.3. Some Important RFC Calls

■ RfcAccept

This should always be the first activity in an RFC server program after starting. Otherwise, an abnormal termination of the server program before **RfcAccept** will always cause timeout errors in an ABAP program by **'Call Function ... Destination ...'**. An error analysis will be more difficult because it is not easy to determine whether the RFC server program is aborted or could not be started.

■ RfcGetName

This call is used by the server program to identify the RFC function to be performed (the name of the function is returned back). A server program that uses this RFC call can only be tested completely using **sm59** from an R/3 System with the RFC library from 3.0D onwards.

■ RfcInstallFunction/RfcInstallFunctionExt

This call can be repeatedly used for different RFC functions after a successful **RfcAccept** to inform the RFC library which functions the RFC library can call up when requested. (**RfcInstallFunctionExt** is for RFC programs on Windows 3.x.).

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

■ RfcDispatch

This call enables the RFC library to activate a RFC function previously informed via **RfcInstallFunction** or **RfcInstallFunctionExt** on request (call from SAP R/2 or R/3).

As a rule, this call should be in a loop in order that the same or other RFC functions can be called by the ABAP/4 program (client program) after an RFC function has been executed. Since the RFC connection is not closed automatically when an RFC function is completed, this helps to minimize the overhead incurred by establishing the connection and starting the RFC server program. Moreover, the globally defined variables are retained in this RFC server program for other RFC function calls and only RFC server programs working with **RfcDispatch** in a loop can be tested from an R/3 system using transaction **sm59**.

■ RfcListen

Because **RfcDispatch** and **RfcGetName** are blocking calls (it will wait until an RFC request is incoming), an RFC server can use **RfcListen** to check whether an RFC request is incoming. It will return back immediately.

■ RfcGetData

With this call, an RFC server program can get all the IMPORT data and -- if applicable -- internal tables associated with the called RFC function. This call must be called in an RFC function.

■ RfcSendData

After executing an RFC function, the RFC server program uses this call to send the EXPORT data and --if applicable-- internal tables associated with the RFC function. This call also terminates the execution of an RFC function and must be called in an RFC function.

- **RfcRaise**

This call is used to raise an exception during the execution of an RFC function.

- **RfcLastError**

After an RFC error, use this call to get more error information for analysis.

- **RfcAbort**

An RFC server can use this call to terminate an RFC connection at any time. An error text can be passed on to the partner system/program with this call. The error text is then available in the partner system as an Error_Message.

- **RfcClose**

With a specified rfc_handle (returned by successful **RfcAccept**) the current RFC connection will be closed. With RFC_HANDLE_NULL , all RFC connections in this server program will be closed. **RfcClose** must also be submitted after a failed RFC call in order that all internally used control areas are released.

- **ItCreate, ItDelete and ItFree**

Contrary to RFC client programs, storage for internal tables is allocated by the RFC library before an RFC function is called. When an RFC function is completed, this storage will be released automatically.

ItCreate, ItDelete and **ItFree** are usually not used in an RFC server program, unless an RFC function in this server program calls up another function module with internal tables in SAP R/2 or R/3 so that this RFC function must allocate and release the storage for internal tables itself.

- **RfcInstallStructure**

Available from 3.1G onwards to inform the RFC library about the structure of received data (Importing parameters and internal tables with heterogeneous structures).

See the delivered sample program srfctest.c for more details.

- **RfcGetName, RfcInstallFunction/RfcInstallFunctionExt and RfcDispatch**

We recommend you to use **RfcInstallFunction/RfcInstallFunctionExt** together with **RfcDispatch** instead of **RfcGetName** because some standard RFC functions supported by the RFC library can only be called while working with **RfcDispatch**.

From 3.0E onwards, if your RFC server must work with **RfcGetName**, you can

1. install a specific RFC function with the function name **"%%USER_GLOBAL_SERVER"** and get then into a loop with **RfcDispatch**.
2. This specific function will be called by the RFC library if an RFC function is called and the function name is unknown by the RFC library. Within this specific function, you can use **RfcGetName** to get the real function name and work as usual (dispatch yourself).

Moreover, you can see the document belong to your RFC function if you install your real RFC function with the function name and NULL as the function pointer for the according document function.

This is necessary if your RFC server is a transactional RFC server (s. chapter 9) and it has to work with **RfcGetName** or your RFC server is a Visual Basic program. The RfcDispatch does NOT work with VB-programs because the RFC library cann't jump directly to a installed function defined in a VB-program.

See the delivered sample program ‘trfcserv.c’ for more details.

6.4. Programming Examples

The following sections only describe some examples in short form.

See the delivered RFC server program ‘srfcserv.c’ for more details.

6.4.1. RFC Server Program working with RfcDispatch

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

if (rfc_handle == RFC_HANDLE_NULL)          /* Check return code                  */
{
  rfc_error_handling("RfcAccept");          /* Get specific error via RfcLastError */
  return 1;                                 /* and handle error                   */
}

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

if (rfc_rc != RFC_OK)                       /* Check return code                  */
{
  rfc_error_handling("Install ‘ABC’");      /* Get specific error via RfcLastError */
  return 1;                                 /* and handle error                   */
}
do                                          /* Wait for call and execution of     */
{
  rfc_rc = RfcDispatch(...);                /*  installed RFC functions until     */

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

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

/* RFC function: ‘ABC’ */
static RFC_RC abc_function(RFC_HANDLE rfc_handle)
```

```
{
  rfc_rc = RfcGetData(...);                /* Get associated RFC data            */

  if (rfc_rc != RFC_OK)                    /* Check return code                  */
  {
    rfc_error_handling("RfcGetData");      /* Get specific error via RfcLastError */
    return 1;                              /* and handle error                   */
  }
  .....                                    /* Process RFC data                   */

  rfc_rc = RfcSendData(...);               /* Report result to ABAP/4 program    */

  if (rfc_rc != RFC_OK)                    /* Check return code                  */
  {
    rfc_error_handling("RfcSendData");     /* Get specific error via RfcLastError */
    return 1;                              /* and handle error                   */
  }
  return 0;                                /* Back to RFC Library                */
}
```

6.4.2. RFC Server Program working with RfcListen and RfcDispatch

Instead of waiting for next RFC call requests, RFC server program can also ask for next RFC request with **RfcListen** and then **RfcDispatch** if an RFC request is incoming. Only the main program of the example above needs to be changed as follows:

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

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

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

if (rfc_rc != RFC_OK)                       /* Check return code                */
{
  rfc_error_handling("Install 'ABC'");      /* Get specific error via RfcLastError  */
  return 1;                                 /* and handle error                 */
}

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

  rfc_rc = RfcDispatch(rfc_handle);

} while( rfc_rc == RFC_OK );

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

6.4.3. RFC Server Programs with Sending/Receiving of Internal Tables

In RFC functions offered by an RFC server program, all importing, exporting parameters and also all internal tables must be defined before RfcGetData is issued. Nevertheless, it is **not necessary** to use **ItCreate** for internal tables. This will be done automatically by the RFC library. After an RFC function is ended the RFC library will free the storage for all used internal tables in this function.

6.5. Getting Connected from R/2 System

- There are no direct communications from an R/2 system (IBM or SNI environment) to a SAP gateway available. An RFC connection from R/2 system can only be established via a **'gwhost'** program (delivered with the SAP gateway software) and this program will connect via the specified SAP gateway to the RFC server program.

- According to the environment (IBM-Host or SNI-host), there are different ways how the **'gwhost'** programs to be configured.

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

6.5.1. Call from ABAP/4 Program

```
CALL FUNCTION "ABC" DESTINATION "RFCEXTERN"
        IMPORT        ......
        EXPORT        ......
        TABLES        ......
        EXCEPTIONS    .......
```

The destination "RFCEXTERN" identifies an entry in the table RFCD.

6.5.2. Table RFCD

Destination	Client	User	Password	Lang	Argument in Table XCOM
RFCEXTERN	000	TEST	xxxxxxxx	E	RFCEXT

The argument "RFCEXT" identifies an entry in the table XCOM.

6.5.3. Table XCOM

- **R/2 in an IBM Environment (with CICS)**

Symbol. destination	LU (Logical Unit)	TP (Transaction prog.)

RFCEXT	LU02	GWRFCSRV

LU02 is a 4-byte identifier defined in a CICS environment, which represents an SNA LU6.2 in CICS. This LU6.2 must be defined in IBM host and on the external computer with an LU6.2 supported SNA product.

■ R/2 in an SNI Environment

Symbol. destination	**LU (Logical Unit)**	**TP (Transaction prog.)**	**Type**
RFCEXT		GWRFCSRV	H

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

6.5.4. ‘GWHOST’-Configuration

6.5.4.1. R/2 in an IBM Environment (with CICS)

On the external computer a "local program” (also known as "**remotely attachable**" or "**remotely invokable**" program on some SNA products) named GWRFCSRV must be defined in the SNA configuration.

GWRFCSRV is a softlink (on UNIX platforms) or copy (PC platforms) of the **‘gwhost’** program, which is included in the delivery of a R/3 system.

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

‘sideinfo’ for GWRFCSRV

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

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

This **‘sideinfo’** file must be in the **working directory** of the GWRFCSRV program:

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

- For **SNAplusLink** on HP-UX, this is the home directory of the owner of the GWRFCSRV program.

6.5.4.2. R/2 in an SNI Environment

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

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

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

‘sideinfo’ for ‘gwhost’ programs

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

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

See installation guide for SAP gateway on BS2000 for more details.

6.5.5. Starting an RFC Server Program

From withhin an R/2 system (IBM or SNI host), an RFC server program can only be started by a SAP gateway. See chapter 6.6.4 for details how to start an RFC server program by a SAP gateway.

Using the registering feature of the SAP gateway from release 3.0C onwards, an RFC server program can be started before, registers at this SAP gateway and then waits for RFC call requests from R/2 system (IBM or SNI host) via the **'gwhost'** program.

In this case, the entry in the **'sideinfo'** file for this **'gwhost'** program must have **'R'** instead of **'E'** as protocol type (**PROTOCOL=R**). See chapter 7 for more details.

6.6. Getting Connected from R/3 System

6.6.1. Call from ABAP/4 Program

```
CALL FUNCTION "ABC" DESTINATION "RFCEXTERN"
          IMPORT        ......
          EXPORT        ......
          TABLES        ......
          EXCEPTIONS    .......
```

The destination "RFCEXTERN" identifies an entry in the table RFCDES.

6.6.2. Table RFCDES in R/3 Release prior to 3.0C

From within an R/3 system, release < 3.0C, an entry in the table RFCDES can be defined with the transaction **sm59** as follows:

■ **RFC destination**:	RFCEXTERN
■ **Connection type**:	T
■ **Program location**:	explizit or server or user
■ **Target host**:	hs0311
■ **Program**:	/rfctest/srfcserv
■ **Trace**:	Not selected
■ **Gateway**:	No info (Default gateway will be used)

Connection type "T"

RFC server program is an external program based on RFC API and communicate via TCP/IP.

Program location "explicit host"

The computer on which the RFC server program runs can be specified via ‘target host’. In this case the RFC server program will be started by the SAP gateway locally or via Remote Shell (locally if the SAP gateway and the RFC server run on the same computer and via Remote Shell otherwise).

On UNIX platforms, this RFC server program will run under the gateway user ID.

The program location "explicit host" is not possible for RFC server programs on Windows 3.1, 3.11 and Windows 95. On Windows NT, this program location is only possible if an SAP gateway is installed on the NT where the RFC server should run. The Sap gateway will start the RFC server locally.

Program location "application server"

The RFC server program is started by the current R/3 application server. This improves performance because the RFC server program always runs on the same computer as the ABAP/4 program (work process of the current ABAP/4 program).

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

Program location "Front-End-Workstation"

The RFC server program is started by the current SAPGUI.

The program location "Front-End-Workstation" is the only option available to start an RFC server with 16-bit RFC library on Windows or RFC server on Windows 95.

Trace

When this field is selected, trace data can be displayed in the SAP System using the ABAP program RSRFCTRC. On external systems the trace data are stored in the ‘dev_rfc’ file. With the 16-bit RFC library or the 32-bit RFC library from 3.1H onwards, a trace file ‘RFCxxxxx.TRC’ will be created for each connection.

Gateway

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

■ Release prior to 2.1I / 2.2C

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

■ Release from 2.1I / 2.2C / 3.0 onwards

Since these releases, the local gateway on each R/3 application server is the default gateway.

Test Connection

After defined and saved a new destination using the transaction **sm59**, the RFC connection using this destination can be tested by double-clicking on "Test connection”. However, this is only possible if the RFC server program was implemented using **RfcDispatch** in a loop. If the server program uses **RfcGetName**, it is only available with an RFC library from 3.0D onwards.

6.6.3. Table RFCDES in R/3 Release 3.0C onwards

From release 3.0C onwards, you can define an RFC server program which is already started, registered at a SAP gateway and wait for RFC call requests (also via **sm59**).

In this case, you have to define an RFC server program with registering as activating kind and a program ID for that.

■ **Program ID**

An identifier of the RFC server program for the SAP gateway to distinguish one from other RFC server programs. We recommend you to use the name of the RFC server program and also the host name of the machine where the RFC server program is running.

■ **SAP gateway**

If not specified, the SAP gateway of the current application server will be used. We recommend you to define the SAP gateway **explicitly** because an RFC server program usually registers at a specific SAP gateway.

6.6.4. Configurations

It is recommended to specify the complete name of the RFC server program (including the full path name) when defining an entry in the RFCDES using **sm59**.

Since the RFC server program can be started by the currently using SAPGUI, by the currently running R/3 application server or by a SAP gateway, but it has to communicate via a specified SAP gateway, following prerequisites must be defined:

- The user under which the SAPGUI or the application server or the SAP gateway must have execution rights for the RFC server program.

- SAP gateway and RFC server program are running on the same computer:
  - the IP address of this computer must be specified in the 'hosts' files.
  - the service name of the SAP gateway must be specified in the 'services' file.

- SAP gateway and RFC server program are running on diferent computers:
  - the IP addresses of both computers must be specified in both 'hosts' files.
  - the service name of the SAP gateway must be specified in both 'services' files.
  - the SAP gateway must have the authority to start the RFC server program on the target computer via Remote Shell:
    - The user of the SAP gateway must be defined on the target computer.
    - The **'.rhosts'** file, which contains the host name of the gateway computer, must exist in this user's home directory on the target computer.
    - Since the remote shell command is different on different UNIX platforms (remsh, rsh, ...), the command can be defined if necessary in the gateway profile parameter gw/remsh (example: gw/remsh =/usr/ucb/remsh).

      'remsh' is default.

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

6.6.5. Starting RFC Server Program

As described above, an RFC server program can be started by a SAP gateway, by the currently using SAPGUI, by the currently running R/3 application server.

Using R/3 from 4.0A onwards and SAPGUI on 32-bit Windows from 4.0A onwards, if your RFC server is already registered in the path

HKEY_LOCAL_MACHINE, SOFTWARE, Microsoft, Windows, CurrentVersion, App Paths, *****.exe

and should be started by SAPGUI, you only need to define $SHELL/<your RFC server> as program in sm59. Your RFC server can then install locally on your PC or on any network server.

Using the registering feature of the SAP gateway from release 3.0C onwards, an RFC server program can be started before, registers at this SAP gateway and then waits for RFC call requests from R/3 system release 3.0C onwards.

In this case, the RFC server program must be specified (with **sm59**) as **registering** for activating action. See chapter 7 for more details.

7. THE ‘SAPRFC.INI’ FILE

Working with the **‘saprfc.ini’** file is only available with the RFC SDK **delivered with the R/3 release 3.0C** onwards but it is only a new interface in the RFC library. Therefore, it can work with any R/3 or R/2 system.

The RFC library on external systems (such as Windows, OS/2 or any R/3-based system) will read this file to find out the connection type and all RFC specific parameters needed to connect to a SAP system (R/2 or R/3). It can also be used for registering an RFC server program at a SAP gateway and then wait for RFC call from any SAP system.

The advantage of this new feature is all RFC specific parameters known at this time (load balancing, RFC with SAPGUI, RFC with ABAP-debug) and in the future can be used without changes in RFC programs.

The **‘saprfc.ini’** file must be in the same directory as the RFC client/server program or it can completely be defined with a full path and file name by the environment variable **RFC_INI**.

Example on Windows: **set RFC_INI=d:\rfctest\saprfc.ini**

RFC Client Program

For using this new feature, an RFC client program must use the **RfcOpen** with RFC_MODE_PARAMETER as 'mode'-parameter and the destination must point to an entry of type **‘B’**, **‘A’, ‘2’, ‘E’** or **‘R’** in the ‘saprfc.ini’ file.

A ‘sideinfo’ file is not necessary!

RFC Server Program

For using this new feature, an RFC server program must use the **RfcAccept** with

-D<destination> as parameter and the destination must point to an entry of type **'R'** in the ‘saprfc.ini’ file.

Restrictions

- The parameters **ABAP_DEBUG** and **USE_SAPGUI** are only possible if the RFC server system is a R/3 release >= 3.0C and the RFC client program is not working with the 16-bit RFC library on Windows (SAPGUI must have at least release 3.0C as well). See chapter 5.7 and 5.8 for more details.
- Load Balancing feature is only available with R/3 release >= 3.0.

Possible Connection (Entry) Types

There are 5 connection types available:

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

See Chapter 11 for more details about RFC between external programs.

Possible Parameters

■ Type ‘R’

Register a RFC server program at a SAP gateway and wait for RFC calls by a SAP system.

DEST=<destination in RfcAccept or in RfcOpen>

TYPE=<**R**: Server will be started by (or is already registered at) SAP gateway>

PROGID=<Program ID, optional, default: destination>

GWHOST=<Host name of the SAP gateway>

GWSERV=<Service name of the SAP gateway>

RFC_TRACE=<0/1: OFF/ON, optional, default: 0 (OFF)>

Following parameters are only available if working with Secure Network Communications (SNC).

SNC will be supported from 4.0A onwards. See SNC documents for more details.

SNC_MODE=<0/1: OFF/ON, optional, default: 0 (OFF)>

SNC_QOP=<1/2/3/8/9, optional, default: 8>

SNC_MYNAME=<Own SNC name, optional>

SNC_PARTNERNAME=<Partner SNC name>

SNC_LIB=<Path and file name of the SNC library>

In R/3 system the **Program-ID** and this **SAP gateway** must be specified in the ‘destination’ entry defined with **sm59:** Connection type **‘T’** and using **Register Mode**.

For working with R/2 system the entry in the ‘sideinfo’ file for the ‘gwhost’ program must have **‘R’** as ‘PROTOCOL’ parameter instead of **‘E’**.

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

■ Type ‘B’

Connect to a R/3 system and the application server will be determined at run time (using the Load Balancing feature of a R/3 release 3.0 onwards.).

DEST=<destination in **RfcOpen**>

TYPE=<**B**: use load balancing feature>

R3NAME=<Name of R/3 system, opt., default: destination>

MSHOST=<Host name of the message server>

GROUP=<Application servers group name, opt., default: PUBLIC>

RFC_TRACE=<0/1: OFF/ON, optional, default: 0 (OFF)>

ABAP_DEBUG=<0/1: OFF/ON, optional, default: 0 (OFF)>

USE_SAPGUI=<0/1: OFF/ON, optional, default: 0 (OFF)>

Following parameters are only available if working with Secure Network Communications (SNC).

SNC will be supported from 4.0A onwards. See SNC documents for more details.

SNC_MODE=<0/1: OFF/ON, optional, default: 0 (OFF)>

SNC_QOP=<1/2/3/8/9, optional, default: 8>

SNC_MYNAME=<Own SNC name, optional>

SNC_PARTNERNAME=<Partner SNC name: SNC name of the message server>

SNC_LIB=<Path and file name of the SNC library>

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

■ Type 'A'

Connect to a specific R/3 application server.

DEST=<destination in **RfcOpen**>

TYPE=<**A**: RFC server is a specific R/3 application server>

ASHOST=<Host name of a specific R/3 application server>

SYSNR=<R/3 system number>

GWHOST=<optional, default: gateway on application server>

GWSERV=<optional, default: gateway on application server>

RFC_TRACE=<0/1: OFF/ON, optional, default: 0 (OFF)>

ABAP_DEBUG=<0/1: OFF/ON, optional, default: 0 (OFF)>

USE_SAPGUI=<0/1: OFF/ON, optional, default: 0 (OFF)>

Following parameters are only available if working with Secure Network Communications (SNC).

SNC will be supported from 4.0A onwards. See SNC documents for more details.

SNC_MODE=<0/1: OFF/ON, optional, default: 0 (OFF)>

SNC_QOP=<1/2/3/8/9, optional, default: 8>

SNC_MYNAME=<Own SNC name, optional>

SNC_PARTNERNAME=<Partner SNC name>

SNC_LIB=<Path and file name of the SNC library>

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

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

■ Type 2

Connect to a R/2 system.

DEST=<destination in **RfcOpen** and in the ‘sideinfo’ file for the SAP gateway>

TYPE=<**2**: RFC server is a R/2 system>

GWHOST=<Host name of the SAP gateway>

GWSERV=<Service name of the SAP gateway>

RFC_TRACE=<0/1: OFF/ON, optional, default: 0 (OFF)>

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

■ Type E

Connect to another external program as RFC server program.

DEST=<destination in **RfcOpen**>

TYPE=<**E**: RFC server is an external program>

GWHOST=<Host name of the SAP gateway>

GWSERV=<Service name of the SAP gateway>

TPHOST=<host name of the RFC server program>

TPNAME=<name of the RFC server program>

RFC_TRACE=<0/1: OFF/ON, optional, default: 0 (OFF)>

Following parameters are only available if working with Secure Network Communications (SNC).

SNC will be supported from 4.0A onwards. See SNC documents for more details.

SNC_MODE=<0/1: OFF/ON, optional, default: 0 (OFF)>

SNC_QOP=<1/2/3/8/9, optional, default: 8>

SNC_MYNAME=<Own SNC name, optional>

SNC_PARTNERNAME=<Partner SNC name>

SNC_LIB=<Path and file name of the SNC library>

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

See chapter 6.6.4 for details about how to start an RFC server program by a SAP gateway.

Some Examples

```
/*===================================================================*/
/*  Type R:  Register a RFC server program at a SAP gateway          */
/*           or connect to an already registered RFC server program  */
/*===================================================================*/
DEST=RFCEXT_R
TYPE=R
PROGID=hw1145.srfcserv
GWHOST=hs0311
GWSERV=sapgw53
RFC_TRACE=1

/*===================================================================*/
/*  Type B:  R/3 system - load balancing feature                     */
/*===================================================================*/
DEST=BIN
TYPE=B
R3NAME=BIN
MSHOST=hs0311
GROUP=PUBLIC
RFC_TRACE=0
ABAP_DEBUG=0
USE_SAPGUI=0

/*===================================================================*/
/*  Type A:  R/3 system - specific application server                */
/*===================================================================*/
DEST=BIN_HS0011
TYPE=A
ASHOST=hs0011
SYSNR=53
RFC_TRACE=0
ABAP_DEBUG=0
USE_SAPGUI=0

/*===================================================================*/
/*  Type 2:  R/2 system (IBM)                                        */
```

```
/*===================================================================*/
DEST=K50
TYPE=2
GWHOST=iw1008
GWSERV=sapgw00
RFC_TRACE=0

/*===================================================================*/
/*  Type E:  External program (will be started by SAP gateway)       */
/*===================================================================*/
DEST=RFCEXT
TYPE=E
GWHOST=hs0311
GWSERV=sapgw53
TPHOST=hs0311
TPNAME=/rfctest/srfcserv
RFC_TRACE=0
```

8. ‘CALL-BACK’ FEATURE

8.1. Introduction

During the execution of an RFC function, it is sometimes necessary and useful to call another RFC function in the (source/origin) RFC client system to get some more data before continuing with the current RFC function.

This functionality is called **‘call-back’** and will use the same RFC connection established by the first RFC call request.

The next chapters describe how to use and to realize this feature both in ABAP function modules and in external programs.

The ‘call-back’-functionality is available with any R/3. For R/2 systems, it is possible with 5.0H onwards.

8.2. ‘Call-back’ from ABAP Function Module

The following diagram shows a programming example:

```
RFC Client Program                                Function Module in R/3

rfc_rc = RfcOpen(…);                              FUNCTION ABC.

rfc_rc = RfcInstallFunction(‘XYZ’,
                            xyz_function, …);

rfc_rc = RfcCallReceive(‘ABC’, …);   ------------→  .....

                                                  CALL FUNCTION ‘XYZ’
                                                        DESTINATION ‘BACK’
If (rfc_rc == RFC_CALL)              ←------------  ......
{
 rfc_rc = RfcDispatch(…);
```

```
  if (rfc_rc != RFC_OK)                  ------------→      .....
    exit(1);

  rfc_rc = RfcReceive(…);                ←------------      ENDFUNCTION.
}
.....


/* RFC function: 'XYZ' */
static RFC_RC xyz_function(RFC_HANDLE rfc_handle)
{
  rfc_rc = RfcGetData(...);              /* Get RFC data          */

  .....                                  /* Process RFC data            */

  rfc_rc = RfcSendData(...);             /* Report result to ABAP*/

  return 0;                              /* Back to RFC Library         */
}
```

8.3. ‘Call-back’ from RFC Server Program

The following diagram shows a programming example:

```
ABAP Program                          RFC Server Program

.....

CALL FUNCTION ‘ABC’                   rfc_rc = RfcAccept(…);
      DESTINATION ‘RFCEXTERN’
      .....              ----------→  rfc_rc = RfcInstallFunction(‘ABC’,
                                                       abc_function, …);

                                      do
                                      {

                                       rfc_rc = RfcDispatch(…);

                                      } while (rfc_rc == RFC_OK);


                                      /* RFC function: ‘ABC’ */
                                      static RFC_RC
                                      abc_function(RFC_HANDLE rfc_handle)
                                      {
                                        rfc_rc = RfcGetData(...);

                                        .....

                                        /* Call-back in Source R/3 system */

                                        rfc_rc = RfcCallReceive(‘XYZ’, …);

                                        rfc_rc = RfcSendData(...);

                                        return 0;
```

```
                                   ←----------------   }
IF SY-SUBRC NE 0.
  "Error Handling"
ENDIF.
```

Function Module 'XYZ' is any RFC-supported Function Module in this source R/3 system.

```
FUNCTION XYZ.

.....

ENDFUNCTION.
```

9. TRANSACTIONAL REMOTE FUNCTION CALL

9.1. Transactional RFC between R/3 Systems

A ‘little’ weakness of RFC

An RFC client program (ABAP or external program) will normally repeat the call of an RFC function if a network error (CPI-C error) returns by this call. If it is a network error occurred **by calling** an RFC function so it is necessary to repeat this action. But if this network error occurred **during or at the end of the execution of an RFC function** the RFC function may already be successfully executed.

In both cases the RFC-component in SAP systems or the RFC library will have the same CPI-C error code from the CPI-C layer.

A repeat of this RFC call is not always correct because it will execute the required RFC function once more!

The transactional RFC

From release 3.0 onwards, data can be transferred between two R/3 systems **reliably and safely** via **transactional Remote Function Call** (It was renamed from asynchronous RFC to transactional RFC because asynchronous RFC has another meaning in R/3 systems).

This ensures that the called function module(s) will be executed **exactly once** in the RFC server system.

Moreover, the RFC server system or the RFC server program needn’t be available at the time when the RFC client program is doing tRFC.

tRFC: Technical Description

A **transaction** also known as **Logical Unit of Work (LUW)** begins with the first **‘Call Function … In Background Task’** and ends with ‘Commit Work’ in an ABAP program.

A transaction can include one or more RFC calls as follows:

CALL FUNCTION ‘F1’ ….. IN BACKGROUND TASK.

…..

CALL FUNCTION ‘Fn’ ….. IN BACKGROUND TASK.

```
.....

COMMIT WORK.
```

- With each **‘Call Function … In Background Task’** the tRFC-component will store the being called RFC function together with the according data in the SAP database.

- **Only after ‘Commit Work’** the tRFC-component tries to pass on these data together with a **Transaction Identifier** (**TID**) which is **unique world-wide** (also on different R/3 systems) to the R/3 server system.

tRFC: Data Transfer

The tRFC-component in both systems communicate with each other in two phases:

- Function shipping
- Confirmation

Phase 1: Function Shipping

- All RFC functions together with the according data and the TID will be transferred to the RFC server system.
- If CPI-C error occurred during this phase the transfer will be repeated by the tRFC-component in the client system (The number of tries and the time between two tries can be defined with **sm59**).
- The R/3 server system uses the TID to check whether the transaction with this TID has been transferred and executed.
- The tRFC-component in server system informs the client system about the successful execution of this transaction.

Phase 2: Confirmation

- The tRFC-component in client system send a confirmation to the client system and both systems can delete the entry for this TID in their own TID-management.
- The confirmation will not be repeated even if an CPI-C error occurs in this phase. Therefore, the TID-management can grow up if working within a bad network.
- The transaction is completely ended after this second phase.

9.2. Transactional RFC between R/3 and External System

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

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

Therefore, the transaction RFC interfaces from external systems to an R/3 is currently implemented as follows:

- **RFC library**

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

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

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

Both programs have to manage the TID's themselves for checking and executing the requested RFC functions **exactly once** as the tRFC-component in an R/3 system does.

- **R/3 systems**

In R/3 systems, there are no additional changes necessary in ABAP programs working with external RFC programs using tRFC-Interface. For ABAP program as RFC client program, the destination defined in CALL FUNCTION …. must have **'T'** as connection type.

- **Restriction**

The tRFC-Interface is not available with 16-bit RFC library on Windows.

9.3. Transactional RFC Client Program

After connected to an R/3 system (via **RfcOpen**), an RFC client program must use the two following RFC calls for working with the tRFC-Interface:

- **RfcCreateTransID**

With this call, the RFC-Library tries to get a TID created by the R/3 system.

In error cases, the RFC client program has to reconnect later and must try to repeat this call. Otherwise, the RFC client program can assign this TID with the RFC data and if the next RFC call is unsuccessful it can be repeated later.

- **RfcIndirectCall**

With this call, the RFC library will pack all RFC data belong to an RFC function together with the TID and send them to the R/3 system using tRFC protocol.

In error cases, the RFC client program has to reconnect later and must try to repeat this call.

In this case, it has to use the old TID and **must not** get a new TID with **RfcCreateTransID**. Otherwise, it is no guarantee that this RFC function will be executed **exactly once** in R/3 system.

After a successful execution of this call, the transaction is completely ended. The RFC client program can then update its own TID-management (e.g. delete the TID-entry).

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

- **RfcIndirectCallEx and RfcConfirmTransID**

With RfcIndirectCall, the RFC client can have a problem with the **'exactly once'**-functionality because if the RFC client is broken down before it can update its own TID management, it will try to call the according RFC function once more. To avoid this error case the RFC library from 4.0C onwards supports two new calls: RfcIndirectCallEx and RfcConfirmTransID.

The RFC client must update the TID as **'executed'** before it issues RfcConfirmTransID.

Because the tRFC server will delete the TID in its TID management after the RfcConfirmTransID is executed, the RFC client can recall the RFC function with RfcIndirectCallEx at any time before RfcConfirmTransID. The tRFC server will or will not call the RFC function depending on the state of this TID.

See the delivered program trfctest.c for more details.

Technical Description

```
--------------------------------------------          -------------------------------------------------------
|  External System                         |          |  SAP  System                                        |
|                                          |          |                                                     |
|  tRFC client program                     |          |                                                     |
|                                          |          |                                                     |
|  RfcOpen              ---------------------------→  |                                                     |
|                                          |          |          -------------------------------            |
|                                          |          |          | tRFC-component             |            |
|  RfcCreateTransID     ---------------------------------------→ |                            |            |
     |                                    |          |          |    Create a TID            |            |
|                       ←---------------------------------------|                            |            |
|                                          |          |          |                            |            |
|  RfcIndirectCallEx('ABC',                |          |          |                            |            |
|                        TID, ...)  -------------------------------→ |                        |            |
|                                          |          |          |    Check the TID           |            |
|                                          |          |          |    and execute the         |----        |
|                                          |          |          |    required functions      |   |        |
|                                          |          |          |    if necessary            |   |        |
|                       ←--------------------------------------------                        |   |        |
|                                          |          |          |    |                       |   |        |
|                                          |          |     ----------                       |   |        |
|                                          |          |    |     |                            |   |        |
|                                          |          |    |     -------------------------------    |        |
|                                          |          |    |                                     |        |
|                                          |          |    |     -------------------------------    |        |
|                                          |          |    |     | Function Module            |   |        |
|                                          |          |    |     |                            |   |        |
|                                          |          |    |     |  FUNCTION ABC. ←------            |
|                                          |          |    |     |  .....                     |            |
|                                          |          |    |     |                            |            |
|                                          |          |     ----- |  ENDFUNCTION.            |            |
|                                          |          |          -------------------------------            |
|                                          |          |                                                     |
|                                          |          |          -------------------------------            |
|                                |              |       | tRFC-component       |       |
|  RfcConfrimTransID(TID,        |              |       |                      |       |
|                          …)  -----------------------→ |    Delete TID in the |       |
|                                |              |       |    TID-Management    |       |
|                          ←--------------------------------                   |       |
|                                |              |       |                      |       |
                                 |              |       -----------------------        |
|                                |              |                                      |
|  RfcClose                      |              |                                      |
|                                |              |                                      |
----------------------------------              ----------------------------------------
```

The Sample Test Program ‘trfctest.c’

- The C-program **trfctest.c**, delivered with the RFC SDK (executable and source code) is a tRFC client program as an example.

- For getting connected to an R/3 system a ‘saprfc.ini’ file is needed.

- Data for transfer to R/3 via tRFC must be in a file. The file name must be defined by starting this program. Each line in this file is one line in an internal table. Only one internal table with 72 B line length is used.

- Received data in R/3 will be written in the TCPIC-table in an R/3 system (only the first 40 bytes) and the function module **STFC_WRITE_TO_TCPIC** will be activated.

- It uses the file I/O on the running platform for management the TID’s.

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

- It is available to break the program to simulate error cases.

- Anytime, when this program is started it will look at the TID-management for aborted transactions. If any exists it will first try to repeat these transactions.

- Since this program can run on different platforms an according flag (SAPonUNIX, SAPonNT, …) must be set if you want to compile and link this program on your environment.

- See source code for more details.

9.4. Transactional RFC Server Program

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

This function installs following 4 functions (e.g. C-routines) to control transactional behavior:

- **onCheckTid**

This function will be activated if a transactional RFC is called from an R/3 system. The current TID is passed. The function has to store this TID in permanent storage and return 0.

If the same TID will be called later again it must return a value <> 0 . If the same TID is already running by another process, but is not completed, the function has to wait until the transaction completes or to stop the RFC connection with **RfcAbort**.

- **onCommit**

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

- **onRollback**

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

- **onConfirmTid**

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

Attentions

- These 4 functions must be realized in **any** tRFC server program and are independent to the real RFC function offered in a RFC server program. A server program can offer more then 1 RFC function but only 4 functions above and not 4 functions for each RFC function.

- Without installing and working with these 4 functions, all RFC functions offered by such a server program, can be called **more than once** by using tRFC-Interface.

Technical Description

- **Within R/3 System**

```
------------------------------------------------------------------------------------------
| SAP system                                                   |
|                                                              |
|                                                              |
| ABAP program                    tRFC-component               |
|                                                              |
| Call Function 'ABC'                                          |
|        Destination 'DEST'                                    |
|        In Background Task                                    |
|        …….          ---------------------→                   |
|                                 Put RFC data in database      |
|                     ←---------------------                   |
|                                                              |
| Commit Work.        ---------------------→                   |
|                     ←---------------------                   |
|                                 Try to send RFC data          |
|                                 to RFC server system ---------------------------→
|                                 (data in tRFC-format)         |
------------------------------------------------------------------------------------------
```

Following parameters can be configured with **sm59**:

- try or don't try to connect to an RFC server program in error cases
- how many times for trying
- the time between two tries
- Program location 'User' defined in sm59 (start RFC server program via currently using SAPGUI) is not available because the tRFC-component is not assigned to any SAPGUI while running.

The transaction **sm58** shows the running state of a transaction (if the transaction is not already successfully executed).

■ Between tRFC-Component, RFC Library and tRFC Server Program

```
----------------------         ---------------------          ----------------------------------------------------
| tRFC-        |         | RFC library |         |          tRFC server program          |
| component    |         |             |         |                                       |
|              | F1      |             | start   |                                       |
|              |------→  |             |------→  | main()                                |
|              |         |             |         | {                                     |
|              |         |             |         |  RfcAccept                            |
|              |         |             |         |  RfcInstallFunction ('ABC',           |
|              |         |             |         |                   abc_function,...)   |
|              |         |             |         |  RfcInstallTransactionControl(...)    |
|              |         |             |         |                                       |
|              |         |             |         |  do                                   |
|              |         |             |  Tn     |  {                                    |
|              |         |             |------→  |    RfcDispatch                        |
|              |         |             |         |  } while (rfc_rc == RFC_OK)           |
|              |         |             |         |                                       |
|              |         |             |         |  RfcClose                             |
|              |         |             |         | }                                     |
|              |         |             |         |                                       |
|              |         |             |  T1     |                                       |
|              |         |             |------→  | function onCheckTid                   |
|              |         |             |         | {                                     |
|              |         |             |         |   Check and update TID                |
|              |         |             | ←------ | }                                     |
|              |         |             |         |                                       |
|              |         |             |  T2     |                                       |
|              |         |             |------→  | function abc_function                 |
|              |         |             |         | {                                     |
|              |         |             |         |   RfcGetData                          |
|              |         |             |         |   .....                               |
|              |         |             |         |   RfcSendData                         |
|              |         |             | ←------ | }                                     |
|              |         |             |         |                                       |
|              |         |             |  T3     |                                       |
|              |         |             |------→  | function onCommit                     |
|                    |         |                    |         | {                                                  |
|                    |         |                    |         |    Update TID and commit database                  |
|                    |         |                    |         |    if necessary                                    |
|                    | ←------|  ----------------   | ←------| }                                                  |
|                    |         |                    |         |                                                    |
|                    |         |                    |  T3'    |                                                    |
|                    |         |                    |------→ | function onRollback                                |
|                    |         |                    |         | {                                                  |
|                    |         |                    |         |    Update TID and rollback database                |
|                    |         |                    |         |    if necessary                                    |
|                    |         |                    |←------  | }                                                  |
|                    |         |                    |         |                                                    |
|                    |  F2     |                    |  T4     |                                                    |
|                    |------→ |   ---------------- |------→ | function onConfirmTid                              |
|                    |         |                    |         | {                                                  |
|                    |         |                    |         |    Update (delete) TID                             |
|                    | ←------|                    |←------  | }                                                  |
--------------------          --------------------          ----------------------------------------------------
```

■ F1: FUNCTION SHIPPING

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

T1: **onCheckTid**

T2: **the required RFC function 'abc_function'**

T3: **onCommit**

As described above, the function shipping phase will be repeated by the tRFC-component in R/3 system if during this phase any CPI-C error (e.g. network errors) occurs. The max. number of tries and the time between two tries can be defined using **sm59** and **tRFC-options**.

■ F2: CONFIRMATION

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

T4: **onConfirmTID**

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

The Sample Test Program ‘trfcserv.c’

- The C-programs **trfcserv.c**, delivered in form of executable and source code with the RFC SDK is a tRFC server program as an example.
- The ABAP program SRFCTEST can be used for test with this server program.
- Received data from R/3 will be written in the ‘trnn…n.dat’ on the running platform. Each line in this file is one line in an internal table. Only one internal table with 72 B line length is used.
- It uses the file I/O on the running platform for management the TID’s.
- For each TID there is an entry in the TID-management and contains the date and time, the TID itself, the state of this transaction (CREATED, CONFIRMED, …) and the name of the data file.
- Since this program can run on different platforms an according flag (SAPonUNIX, SAPonNT, …) must be set if you want to compile and link this program on your environment.
- See source code for more details.

10. RFC AND SAPROUTER

10.1. Introduction to SAProuter

SAProuter is a SAP software product, available on all R/3-based UNIX-platforms and Windows NT/95. It acts like a firewall system by regulating accesses from/to your network.

SAProuter can be used

- to establish an indirect connection between two programs running on different machines and the network configuration does not allow a direct communication between these machines due to missing IP addresses (or same IP addresses as well) or firewall restrictions.
- to improve network security by allowing accesses from/to your network with or without password-protection only from a specified machine where the SAProuter is running and.
- to control and log all connections between your network and rest of the world.

Important SAProuter Commands

saprouter	Online help (display list of all supported options)
saprouter -r	Start SAProuter with default values
saprouter -s	Stop SAProuter

Route String

A route string can have one or more substrings. Each substring contains parameters how to reach the next SAProuter or the target host or program on the target host.

Such parameters are:

- name or IP-address of the target host
- service (port number) of the program running on the target host
- password for this connection, if needed.

Example for one substring: **/H/host/S/service/P/password**

H: Identifier for host name

S: Identifer for service (port number)

P: Identifer for password

Route Permission Table

The SAProuter regulates accesses to your network via the route permission table in form of a file. You can start your SAProuter with this file name.

An entry in a route permission table has following structure:

<P/D> <source host> <target host> <target service> <password>

P(ermit):	allows connection
D(eny):	prevents connection
<source host>:	host name or IP -addrese, could be a SAProuter
<target host>:	host name or IP -addrese, could be a SAProuter
<target service>:	service (port number) of the program on the target host The default service of SAProuter is ‘3299’.

CAUTION:

- If no route permission table was explicitly assigned to the SAProuter while starting (option -R <name of a route permission table>), the file ‘saprouttab’ in the current directory will be used. If this file is not available, all connections are allowed.
- You can use wildcards (‘*’) to define hosts, services and passwords in your route permission table

See **SAP note 30289 in CSP** for more details about SAProuter.

A Typical Use of SAProuter (Remote Support)

Network_1 (SAP-Walldorf)		**Network_2 (Customer)**	
host_11	**host_r1**	**host_r2**	**host_21**
SAPGUI ------------→	**SAProuter ---------------------→**	**SAProuter ---------→**	**SAP R/3**
	(“3299”)	(“3299”)	

Route Permission Tables

Entry in the route permission table of **SAProuter on host_r1 in Network_1:**

P	host_11	host_r2	3299

Entry in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_21	sapdp<R/3 system number>

The SAPGUI on host_11 will connect to the R/3 application server on host_21 with the following route string, host name and dispatcher service:

/H/host_r1/H/host_r2/H/host_21/S/sapdp<R/3 system number>

The information about services of both SAProuters are not necessary because they are running with the default service (“3299”).

10.2. RFC Client Program and SAProuter

Any RFC client program can connect to a SAP system via SAProuter. The feature RFC using SAPGUI (available from 3.0C onwards) is also possible via SAProuter. Don’t forget to allow the coressponding SAP dispatcher working with the SAProuters as well.

One or more SAProuters can be used. Following example shows how the client program can work with 2 SAProuters.

```
Network_1                                  Network_2

host_11          host_r1                   host_r2      1)     host_21
RFC client ---------> SAProuter -------------------> SAProuter -----> R/3-MS
                 ("3299")                  ("3299")
                                                        2)     host_22
                                                        -----> SAP-GW
                                                               |   |
                                                           ------  V
                                                        3)  |    R/3-AS
                                                           -------
                                                                |
                                                               V
                                                               host_23
                                                               R/3-AS

                                                        4)     host_sna
                                                        -----> SAP-GW
                                                                  |
                                                                  V
                                                               IBM-Host
                                                                 R/2

                                                        5)     host_bs2
                                                        -----> SAP-GW
                                                                  |
                                                                  V
                                                               SNI-Host
```

Route Permission Tables

There is only one entry in the route permission table of **SAProuter on host_r1 in Network_1** for all posiblities above necessary:

P	host_11	host_r2	3299

The entries in the route permission table of **SAProuter on host_r2 in Network_2** are dependent to the type how the RFC connection is established and to which SAP system in Network_2 (in example above: 1, 2, 3, 4 or 5). The RFC client program must inform the RFC library about all used SAProuters via the parameter about host name in **RfcOpen**.

These entries and the host names in example above must be set as follows:

1. Using Load Balancing (R/3, Release 3.0 onwards)

The host name of the message server must contain the route string. For the above example, the RFC client must set the host name of the message server as follows:

MS-Host:	/H/host_r1/H/host_r2/H/host_21.

Entries in the route permission table of **SAProuter on host_r2 in Network_2:**

P	host_r1	host_21	sapms<R/3 name>
P	host_r1	<R/3-AS host 1>	sapgw<R/3 system number>
…..			
P	host_r1	<R/3-AS host n>	sapgw<R/3 system number>

2. Specified R/3 Application Server and Default SAP Gateway

The host name of the specified application server must contain the route string. For the above example, the RFC client must set the host name of the application server as follows:

AS-Host:	**/H/host_r1/H/host_r2/H/host_22**

Entries in the route permission table of **SAProuter on host_r2 in Network_2**:

P	host_r1	host_22	sapgw<R/3 system number>

3. Specified R/3 Application Server and Specified SAP Gateway (R/3)

If working with a specified SAP gateway (not recommended for R/3, release >= 2.1I, 2.2C or 3.0 because of bad performance) the host name of the SAP gateway must contain the route string. The host name of the application server may not contain the route string. For the above example, the RFC client must set the host name of the SAP gateway as follows:

GW-Host:	**/H/host_r1/H/host_r2/H/host_22**
AS-Host:	**host_23**

Entries in the route permission table of **SAProuter on host_r2 in Network_2**:

P	host_r1	host_22	sapgw<GW service number>

4. R/2 in IBM Environment

The host name of the SAP gateway must contain the route string. For the above example, the RFC client must set the host name of the SAP gateway as follows:

GW-Host:	**/H/host_r1/H/host_r2/H/host_sna**

host_sna is a machine where a SNA LU6.2-product is running.

Entries in the route permission table of **SAProuter on host_r2 in Network_2**:

P	host_r1	host_sna	sapgw<GW service number>

5. R/2 in SNI Environment (BS2000)

The host name of the SAP gateway must contain the route string. For the above example, the RFC client must set the host name of the SAP gateway as follows:

GW-Host: /H/host_r1/H/host_r2/H/host_bs2

host_bs2 is a BS2000-host where the SAP R/2 is running.

Entries in the route permission table of **SAProuter on host_r2 in Network_2**:

P	host_r1	host_bs2	sapgw<GW service number>

10.3. RFC Server Program via SAProuter

An RFC server program can be started by a SAP gateway, by the curently running R/3 application server or by the currently running SAPGUI. Using the registering feature of an R/3 release 3.0C onwards, it can be started before and registers at a SAP gateway and then waits for RFC request.

Due to the art of starting an RFC server program and the Release of SAP system, there are different ways how an RFC server program can work with SAProuters.

10.3.1. Using Registering Feature (Rel. 3.0C onwards)

As described in chapter 6.2, the registering feature is only available with an R/3 and SAP gateway release 3.0C onwards.

One or more SAProuters can be used. Following example shows how a server program can work with 2 SAProuters using registering feature.

```
Network_1                                   Network_2

host_11          host_r1                    host_r2          host_21
RFC server  --------> SAProuter  ------------------> SAProuter    -----> SAP-GW
                 ("3299")                   ("3299")            A
                                                                |
                                                                V
                                                             R/3-AS
```

Route Permission Tables

Entry in the route permission table of **SAProuter on host_r1 in Network_1**:

P	host_11	host_r2	3299

Entry in the route permission table of **SAProuter on host_r2 in Network_2**:

P	host_r1	host_21	sapgw<GW service number>

The host name of the SAP gateway must contain the route string. For the above example, the RFC server must set the host name of the SAP gateway as follows:

GW-Name: **/H/host_r1/H/host_r2/H/host_21**

The delivered RFC server program srfcserv.c can be started at the command line as follows:

```
srfcserv -ahost_11.srfcserv -g/H/host_r1/H/host_r2/H/host_21 -xsapgw<GW serv.- nr.>
```

An destination in **sm59** can be defined as follows:

Connection type:	**T**
Activate type:	**Registering**
Program ID:	**host_11.srfcserv**
Gateway host:	**host_21**
Gateway service:	**sapgw<GW service number>**

10.3.2. Start by Application Server

Not available, because in this case the RFC server program will run on the same machine as the application server.

10.3.3. Start by SAP gateway

Because an SAP gateway cannot start an RFC server program with remote shell on another machine via SAProuter, it is necessary to install an SAP gateway on a machine in the network where the RFC server program will run. For better performance, both should run on the same machine.

One or more SAProuters can be used. Following example shows two different networks with two SAProuters and how an RFC server program can be started by an SAP gateway and communicates with an R/3 system running on other network.

```
Network_1                                        Network_2

host_11           host_r1                        host_r2            host_21
SAP-GW   <---------  SAProuter  <------------------  SAProuter  <------  SAP-GW
   A              ("3299")                       ("3299")              A
   |                                                                   |
   V                                                                   V
RFC server                                                          R/3-AS
```

Route Permission Tables

Entry in the route permission table of **SAProuter on host_r2 in Network_2**:

P	host_21	host_r1	3299

Entry in the route permission table of **SAProuter on host_r1 in Network_1**:

P	host_r2	host_11	sapgw<GW service number>

An destination in **sm59** can be defined as follows:

Connection type: **T**

Activate type:	**Start**
Program location:	**explizit host**
Program:	**/rfctest/srfcserv**
Target host:	**host_11**
Gateway host:	**/H/host_r2/H/host_r1/H/host_11**
Gateway service:	**sapgw<GW service number>**

CAUTION:

The max. length of the gateway host is 32 bytes for R/3 release <= 3.0C. With 3.0D onwards, it will be 100 bytes.

10.3.3. Start by SAPGUI

One or more SAProuters can be used. Following example shows two different networks with two SAProuters and how an RFC server program can be started by the currently running SAPGUI and communicates with an R/3 system running on other network.

```
Network_1                                    Network_2

host_11           host_r1                    host_r2          host_21
SAPGUI    <------>  SAProuter  <--------------->  SAProuter  <--->  SAP-GW
   |              ("3299")                   ("3299")            A
   |                                                             |
   V                                                             V
RFC server                                                     R/3-AS
```

Route Permission Tables

Entry in the route permission table of **SAProuter on host_r1 in Network_1**:

```
P       host_11        host_r2        3299
```

Entry in the route permission table of **SAProuter on host_r2 in Network_2**:

```
#Entry for SAPGUI
P       host_r1        host_21        sapdp<R/3 system number>
#Entry for RFC server program
P       host_r1        host_21        sapgw<R/3 system number>
```

An destination in **sm59** can be defined as follows:

Connection type:	**T**
Activate type:	**Start**
Program location:	**Front-end workstation**
Program:	**/rfctest/srfcserv**

CAUTION:

Only available with R/3 3.0D onwards and SAPGUI 3.0E onwards.

11. RFC BETWEEN EXTERNAL PROGRAMS

With the RFC API, SAP provides an interface for communications with external systems using the same internal function calls in SAP systems. For a local test (without SAP systems), you can write both RFC client and server program as external programs and let these programs communicate with each other **via a SAP gateway**. In this case, an RFC server program can only be started by a SAP gateway or it can use the registering feature described in chapter 6.2.

The delivered sample programs **srfctest.c** and **srfcserv.c** can be used for a local test, if you have a SAP gateway running in your test environment.

An RFC client program can work either with a local 'sideinfo' file or with the 'saprfc.ini' file.

If the RFC server is running in register mode, the RFC client must close and reconnect to the RFC server with after each RFC function call

11.1. Using local 'sideinfo'

Sample entry in the local 'sideinfo':

```
DEST=RFCEXT
GWHOST=<host name of the SAP gateway,       e.g. hs0311>
GWSERV=<service name of the SAP gateway,   e.g. sapgw53>
PROTOCOL=<E/R: started by /  already registered at SAP-GW>
LU=<host name of the RFC server program,       e.g. hs0311>
TP=<name of the RFC server program,              e.g. /rfctest/srfcserv>
```

An entry in the 'sideinfo' for the SAP gateway is not necessary.

The parameter **'mode'** in **RfcOpen** must be set to **RFC_MODE_CPIC**.

11.2. Using the 'saprfc.ini'

The RFC communication between two external programs using the 'saprfc.ini' file is only available with the RFC library from 3.0D onwards. The SAP gateway can be from 2.1I, 2.2C or 3.0 onwards. Working with type E is only available with 3.0D onwards.

Sample entry for an RFC client to start an RFC server program by an SAP gateway:

```
DEST=RFCEXT
TYPE=<E: server will be started by a SAP gateway>
GWHOST=<host name of the SAP gateway,            e.g. hs0311>
GWSERV=<service name of the SAP gateway,         e.g. sapgw53>
TPHOST=<host name of the RFC server program,     e.g. hs0311>
TPNAME=<name of the RFC server program,          e.g. /rfctest/srfcserv>
```

The parameter **‘mode’** in **RfcOpen** must be set to **RFC_MODE_PARAMETER**.

Sample entry for an RFC server to register at an SAP gateway
or for an RFC client to connect to an already registered RFC server

```
DEST=RFCEXT_R
TYPE=<R: server is already registered or the client will connect to a registered server>
GWHOST=<host name of the SAP gateway,            e.g. hs0311>
GWSERV=<service name of the SAP gateway,         e.g. sapgw53>
PROGID=<Program ID of the RFC server program,    e.g. hs0311.srfcserv>
```

12. NEW FEATURES OF RELEASE 3.0 D ONWARDS

12.1. New features in 3.0 D and 3.0 E

RFC server program, SAPGUI and SAProuter

From R/3 3.0D onwards and RFC library, SAPGUI 3.0E onwards, it's available to start an RFC server program by an SAPGUI on a workstation which is connected to an R/3 via one or more SAProuters.

RFC server program, SAPGUI and R/3 in different networks

From R/3 3.0D onwards and RFC library, SAPGUI 3.0E onwards, it's available to start an RFC server program by an SAPGUI on a workstation which is connected to an R/3 application server. And this application server has 2 network adapters: one for the network with all R/3 application servers and one for the network with this application server and the Front-End Workstation with SAPGUI.

Host name of gateway and application server in sm59

Host name of a SAP gateway in sm59:

Till 3.0C: 32 bytes,	In 3.0D: 64 bytes,	From 3.0E onwards: 100 bytes

Host name of an application server in sm59:

Till 3.0D: 64 bytes,		From 3.0E onwards: 100 bytes

New RFC call: RfcGetAttributes

The new call RfcGetAttributes is available with the RFC libary 3.0E onwards.

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

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

See saprfc.h and the delivered sample programs srfctest.c and srfcserv.c for more details.

New RFC call: RfcWaitForRequest

The new call RfcWaitForRequest is available with the RFC libary 3.0E onwards.

Using the "Registering"-feature of an R/3 3.0C onwards, an RFC server program can use RfcAccept to register itself at an SAP gateway. After that, it can wait for any RFC request by calling RfcGetName, RfcDispatch or RfcListen.

RfcGetName and RfcDispatch are blocking calls. The server program will then wait until any RFC request is received.

In 3.0C and 3.0D, RfcListen is also a blocking call when using register mode.

From 3.0E onwards, RfcListen is a NON-blocking call. But if you don't issue this call quick enough, the SAP gateway will return error to the RFC client because there was no server program available within the defined time in the gateway.

Instead of RfcListen, you can use this new call with a time in seconds as a parameter to define how long you want to wait for the next RFC request.

See saprfc.h and the delivered sample program srfcserv.c for more details.

More information in error cases

For better error analysis, the RFC library and R/3 from 3.0E onwards will write some more error information into the TRACE file:

dev_rfc0, dev_rfc1, ... withhin an R/3

or dev_rfc or RFCnnnnn.TRC (16-bit RFC library) on external system.

Getting version of the RFC library

Till now, it's possible to get the version of the RFC library and all SAP internal libraries such as cpic, ni,... (via RfcGetAllLibVersions) but this call doesn't return any information about the R/3 release this RFC library belongs to. From 3.0E onwards, the RFC library will also return to this call the release of the R/3 it belongs to.

Start sapinfo -v to get this information.

Self-Dispatching of RFC functions in server program

From 3.0E onwards, an RFC server program can install a specific RFC function named "%%USER_GLOBAL_SERVER". The RFC library will call this function if an RFC request is received and there is no function installed for this. In this specific function, the RFC server can use RfcGetName to get the requested function name and then dispatch for itself.

This functionality must be used if you are working with transactional RFC and you don't want to work with RfcDispatch but RfcGetName.

See the delivered sample program trfcserv.c for more details.

12.2. New features in 3.0 F and 3.1 G

Memory leak

The memory leaks caused by RfcOpen/RfcClose and while working with transactional RFC are now fixed in 3.0F. Other leaks caused by RfcAccept for working with registered mode in error cases is fixed in 3.1G.

Directory for RFC trace files

Till now, RFC trace files on external system will be created in the working directory of RFC programs. From 3.0F onwards, you can set the path of these trace files via the environment variable "RFC_TRACE_DIR".

Inhomogeneous structures and tables

From 3.1G onwards, the RFC library also supports inhomogeneous structures and tables. In that case, RFC client or server programs must inform the RFC library about the structure of inhomogeneous parameters and tables by calling RfcInstallStructure.

See and the delivered sample programs srfctest.c and srfcserv.c for more details.

New RFC call: RfcDefineImportParam and RfcGetImportParam for Visual Basic 4.0/5.0

Because of some changes in accessing strings in Visual Basic 3.0 and Visual Basic 4.0/5.0 while working with DLL's, RFC programs writing in Visual Basic 4.0/5.0 cannot get IMPORTING parameters via RfcAddImportParam before calling RfcCallReceiveExt, RfcReceiveExt or RfcGetDataEx.

From 3.1G onwards, the RFC library also supports Visual Basic 4.0/5.0 programs with directly using the 32-bit RFC library on Windows NT/95. In this case, they must use the RFC library from 3.1G onwards and these two new calls.

See saprfc.h for more details.

New RFC call: RfcSetCodepage

Till now, the RFC library determines the own code page for every RFC connection. With this new call from 3.1G onwards, an RFC client or server can set an own codepage for each connection. In this case, it must be called after RfcOpen or RfcAccept. Don't forget to inform the RFC library via the environment variable PATH_TO_CODEPAGE in which directory the conversion tables are defined.

The 16-bit RFC library on Windows does not support this new call.

12.3. Multi threaded RFC Library on 32-bit Windows (95/NT)

From 3.1H onwards, the RFC library on 32-bit Windows (NT or 95) supports multi-threading. An RFC client or server program can have more than one thread working parallel with RFC.

See the delivered sample programs rfcthcli.c and rfcthsrv.c for more details.

CAUTIONS:

1. The RFC library can also be used in a single or multi threaded environment. No additional RFC calls are needed for using as multi threaded library.

2. For better error analysis, an RFC trace file RFCnn...n.TRC will be created for each RFC connection instead of dev_rfc for all connections.

3. An RFC handle of an RFC connection, built up from one thread can be used in another thread but these threads must synchronize with each other for using RFC calls with this RFC handle. The RFC library does not check whether an RFC call with this handle is still working and the behavior in this case is undefined.

4. A handle of internal table can also be used in different threads but these threads must synchronize by accessing this table. The RFC library does not check or lock this table during the access on it.

12.4. New features in 4.0

Following new features are available from 4.0A onwards.

New RFC call: RfcOpenEx

This new call should be used instead of all other calls such as RfcOpen, RfcConnect, RfcOpenExt, RfcOpenExtV3 to open an RFC connection. Using this call, an RFC client can define all necessary connection data as parameters in this call. It can also use the saprfc.ini file. Moreover, a connect to any R/2, R/3 or External System is possible using this call.

See saprfc.h and the delivered sample program srfctest.c for more details.

Using sapmsg.ini/saproute.ini of SAPLOGON

When using the new call RfcOpenEx for connect to an R/3 via Load Balancing, and if there is no info about the host name of the message server (in connect_param of RfcOpenEx or in saprfc.ini), the RFC library will try to get this host name from the sapmsg.ini customized for the SAPLOGON on Windows.

Usually, these files are installed in the Windows directory. You can also copy them in a directory which is specified by the environment 'RFC_LOGON_INI_PATH'.

On NON-Windows platforms, you can work with this environment or copy these files in your working directory.

See saprfc.h and sapinfo.c or srfctest.c for more details.

New RFC call: RfcLastErrorEx

This new call should be used instead of RfcLastError to get the last error occurs on an RFC connection. An RFC error will be described by three parameters: an error group, an error key and an error message. The error group is just an integer identify the error key and can be used in RFC program to better analyze the error.

See saprfc.h and the delivered sample program srfctest.c for more details.

New RFC call: RfcCheckRegisterServer

Use this new call to check whether and how many RFC servers are registered with a defined 'Program ID' at a SAP gateway.

This call is only available with a SAP gateway from 3.1G onwards.

See saprfc.h and the sample program srfctest.c for more details.

New RFC call: RfcCancelRegisterServer

Use this new call to cancel all RFC servers which are registered with a defined program ID at a SAP gateway.

This call is only available with a SAP gateway from 3.1H onwards.

See saprfc.h and the sample program srfctest.c for more details.

Support 2-byte ISO-language

Using the new call RfcOpenEx, you can login onto an R/3 from 4.0A onwards with a 2-byte ISO-language instead of the 1-byte SAP-language.

The RFC-library will convert the 2-byte ISO-language in an 1-byte SAP-language before sending this to the according SAP system. Therefore, you can also use a 2-byte ISO-language to login onto an R/2 or an R/3 release 2.x or 3.x.

RFC with Remote SAPGUI

Till now, the functionality "RFC using SAPGUI or ABAP-debugger" requires that SAPGUI will be started on the same computer where the RFC client is running. From R/3 4.0A onwards, you can determine on which computer SAPGUI should be started.

Following steps are necessary:

1. Set the environment variable RFC_REMOTE_GUI on the computer where your RFC client program will run as follows:

   ```
   set RFC_REMOTE_GUI=PROGID:my_gui_test GWHOST:hs0311 GWSERV:sapgw53
   ```

2. Make sure that the RFC library on the remote computer can start SAPGUI on this computer and start now the delivered sample program srfcserv (or rfcexec) on this remote computer as follows:

   ```
   srfcserv -amy_gui_test -ghs0311 -xsapgw53
   ```

   The parameters Program-ID, GW-Host and GW-Service have to match the parameters in the environment variable defined on the computer of the RFC client program.

3. Start your RFC client program and the RFC library used by the RFC program srfcserv will start SAPGUI on the required comput er.

For this purpose, you can use any SAP gateway supported the "register"-feature (from 3.0C onwards). There are neither change in your RFC client program nor additional configurations in R/3 necessary.

Queued Remote Function Call

(qRFC)

Contents

Version: 4.6D.025
July 2000
created by
Duong-Han Tran
Basis/RFC

1. Transactional RFC (tRFC) & Queue RFC (qRFC)

What Does tRFC Perform Right Now?

- A LUW is a set of tRFC-calls (Call Function … In Background Task) terminated with Commit Work. Each LUW is automatically assigned to a unique Transaction-ID (TID).

- tRFC guarantees that all tRFC-calls in a Logical Unit of Work (LUW) will be exactly once executed in the target system.

- All function modules belonging to a LUW will be executed in the target system in the sequence in which they are called from within an application in the sending system.

- A tRFC-call with the option AS SEPARATE UNIT is treated as a separate sub LUW within the main LUW while processing (transfer to and excute in the target system). Such a separate sub LUW contains only one tRFC-call and is assigned to a separate TID. The processing of the separate sub LUW is independent of the main LUW.

- If several destinations are used in one LUW then all tRFC-calls that belong to one destination form a sub LUW. However, there is no separate TID assigned to each sub LUW in this case.

Some Problems With tRFC

- All tRFC-LUWs are processed independent of each other. The amount of activated tRFC- processes (one for each LUW) can reduce the performance of the sending and target system.

- There is no guarantee that the tRFC-LUWs will be executed in the same sequence according to the time when they are called by one or more applications. The only guarantee is that these LUWs will be processed sooner or later.

- If a tRFC-call and a "Call Function … In Update Task" are used together with one Commit Work from within a Dialog Task and if the function running in the Update Task is also using tRFC-calls then there will be different tRFC-LUWs with different TIDs: All tRFC-calls from within the Dialog Task are assigned to one LUW/TID and all other tRFC-calls from within the Update Task are assigned to other LUW/TID. There is also no guarantee about the processing sequence of these LUWs.

- If the work process or the R/3 application server shuts down or is restarted during the processing of an LUW (asynchronous to the tRFC application), this LUW can be automatically restarted via the program RSARFCEX (as a periodly running batchjob) or manually via the tRFC monitoring (SM58).

qRFC With Send/Receive Queue

The serialization of tRFC-LUWs uses queues to guarantee a LUW sequence dictated by one or more applications. Send queues are temporary created at run time. This facility is called qRFC with Send Queue.

Conversely, there are applications that want to determine themselves when to process the incoming LUWs within the target system. Some other applications just want to transfer LUWs as soon as possible into the target system. These LUWs will be processed later. Therefore, qRFC also supports receive queues and this facility is called qRFC with Receive Queue.

For NON-SAP systems, the RFC library delivered with the RFC SDK provides new API'ss to use the qRFC with Receive Queue supported within an R/3 system.

Actually, qRFC with Send Queue is available from 4.5B onwards but not with all functionalities described in this document.

qRFC with Receive Queue and the new RFC library supported qRFC with Receive Queue is available from 4.6A onwards.

For some applications such as APO (Advance Planner and Optimizer) or CRM (Customer Relationship Management) qRFC with Send/Receive Queue is also available in some Add-On systems running with the R/3 kernel release 3.1H, 3.1I, 4.0B or 4.5B.

All functionalities described in this document are available from 4.6C onwards. Add-On systems which want to have the newest qRFC-funcitonalities must install the newest qRFC-version.

2. qRFC With Send Queue

2.1. Characteristics

Serialization of tRFC-LUWs

- qRFC with send queue is an enhancement of tRFC. It is a layer between applications and tRFC and allows tRFC only to transfer an LUW to the target system once it has no predecessors in the participating queues. Moreover, after a qRFC-LUW is executed, the qRFC manager will automatically process the next waiting qRFC-LUW according to the queue sequence.

- qRFC with send queue provides a serialization of tRFC-LUWs on the sending system. It is available to use qRFC with Send Queu to work with any tRFC-supported target system (SAP or NON-SAP using the RFC library).

Queue Name, Queue Identifier

- To use qRFC, an application must pass its queue names by calling one of the provided function modules immediately before issuing Call Function … In Background Task.

- A queue name is a 24 byte string. It must not include *, &, % or small (lower-case) letters. Moreover it must not begin with a blank.

- To prevent different applications from unintentional use of same send queue names, each application should choose a specific prefix for queue names (e.g. SFA_... or APO_...).

- As queue identifier for serialization, the qRFC manager uses the current SAP-client, the queue name and the current used destination (target system).

- There is no queue configuration necessary.

Better system performance with qRFC than with tRFC

- An advantage of qRFC with Send Queue vs. tRFC is the minimum load on sending and also target system due to the serialization. However, choosing queue names carelessly (too many queue names) can neutralize this advantage. This is a classical dilemma: Better system performance due to more serialization versus faster processing due to more parallelizing. Therefore, each application must carefully determine the number of queues and the queue names.

Automatic Restart of "hanging" qRFC-LUWs

- Unlike tRFC, it is not necessary to have the program RSARFCEX as a periodly running batchjob to restart hanging LUWs while using qRFC. The qRFC manager will automatically restart any hanging LUW when a new LUW is written into the queue.

Getting qRFC-LUW from a Send Queue

- Instead of sending the qRFC-LUW immediately to the target system, you can also store this LUW in the Send Queue. From the local or a remote system you can use standard qRFC function modules to get a few or all LUWs stored in this queue and process them on your (local/remote) system

"Hanging" and "Traffic Jam" Problems

- If the first LUW in a queue cannot be processed due to communications problem, not only this queue blocks but also all other LUWs that are interdependent with this queue. The qRFC tables will grow and the resulting "traffic jam" could cause a database problem. However, as soon as the communications problem is resolved, all LUWs and thus all interdependent queues will be processed automatically one after the other (automatically via destination configuration in SM59 or manually via SM58 or via some ABAP program using some standard qRFC function modules for Queue Management)

- Such a traffic jam can also occur once a system or application exception is raised during the execution of the first LUW in a queue from within the target system,. There is no automatic restart in this case. The system or application administrator must fix this problem before restart this queue.

Exception Handling of tRFC vs. qRFC

- If an system or application exeption is raised during the processing of a tRFC-LUW the target system will return this error back to the sending system and the status of this LUW will be updated with the exception error message (red color on status message showed by SM58). At the same time, a batchjob will be scheduled for an automatic deleting of this LUW after one year (Changing of this value is only possible in the include <sys000>, parameter %_arfc_erase_in_days, default 365).

- Using qRFC with Send Queue, the administrator must manually fix the problem and either restart the queue or delete the qRFC-LUW. The qRFC manager will not schedule a batchjob for an automatic deleting of this LUW.

tRFC/qRFC in Dialog Task and Update Task

- If you are using tRFC-calls within a Dialog Task and an Update Task in the same transaction then the sequence of these tRFC-LUWs is not defined (different LUWs, different TIDs).
- Using qRFC, you can call a standard qRFC function module in your transaction to set the processing sequence. The default sequence is: all qRFC-calls from within the Dialog Task will be processed before other qRFC-calls from within the Update Task (All LUWs will only be processed after the update is successfully finished).

2.2. LUW, TID & Queue Counter

Different TIDs for different destinations in one qRFC-LUW

- Using tRFC, all tRFC-calls within an LUW with different destinations are assigned to a common TID but they are treated internally as different sub LUWs (according to the destination).

- The qRFC manager, however, bundles them per destination and processes them independent of each other. All qRFC-calls with different destination are therefore assigned to different TIDs (sub LUWs). In this case, there is no guarantee that the sequence of the sub LUWs to different destinations (target systems) is kept; the only guarantee is that these sub LUWs will be processed sooner or later.

Serialization via Queue Counter

- At the "Commit Work"-time, the qRFC manager sets a counter that is used to serialize the LUW-processing in one or more queues.

- The sequence of the qRFC-calls within one LUW using the same destination and queue names is guaranteed. Even if the qRFC-calls are using the option AS SEPARATE UNIT in the same LUW they will be serialized accoding to the called sequence when using the same queue.

"Mixed Mode"

- "Mixed Mode" with tRFC and qRFC is available: One LUW can contain tRFC-calls as well as qRFC-calls. The sequence of these tRFC/qRFC-calls using the same destination is guaranteed.

- The tRFC-calls within an LUW in "mixed mode" and the qRFC-calls using the same destination form a sub LUW in reference to their TID-assignment. If within one LUW no qRFC-call is used for a certain destination, the tRFC-calls using this destination form a sub LUW by themselves and this sub LUW will be processed as a normal tRFC-LUW.

- The separate tRFC-calls (with option AS SEPARATE UNIT) within an LUW in "mixed mode" are treated as before (individual TID): dependent on the using queue but independent of the main LUW.

2.3. Destinations NONE & SPACE

- Using tRFC with destination NONE or SPACE, the called function module will be executed on the local R/3 system. Using SPACE, this function module can be executed in any application server. With destination NONE, it will be executed on the same application server on which the tRFC-call is done. This is e.g. necessary if the function module called by tRFC needs to access some operating system files located on the application server of the tRFC-caller. To get this working, tRFC internally changes the used destination NONE to the SAP-internal destination (e.g. pswdf073_SFA_73) and SPACE to NONE. You can get this SAP-internal destination by a local call of the function module RFC_SYSTEM_INFO.

- For compatibility, qRFC does the same with destinations NONE and SPACE. So you have to pay more attention if using destination NONE and SPACE while managing the send queues (destination using by qRFC-call is not the same for STOP/RESTART/… the queue). Therefore, we recommend you to use destination SPACE instead of NONE if possible.

2.4. User and user-context of a qRFC-LUW

User of a qRFC-LUW

Like tRFC a qRFC-LUW will run in the destination system under the user defined in SM59 via the destination specified in the statement "call function … destination … in background task". If no user is defined in SM59 for this destination the current user will be used.

Using this feature in tRFC you can have the tRFC-LUW running under the current user in the destination system but you will have some problem if the tRFC-LUW must be restarted via a batchjob because it will run under the user which schedules the batchjob. To avoid this problem you have to define a specific destination for each user or for a group of users.

Using qRFC with send queue and if different users can write in this queue and if you don't specify a user in SM59, there is not clear under which user the LUW in the destination will be executed. Therefore, we recommend you always to specify an user in SM59 while working with qRFC with send queue.

User-context of a qRFC-LUW

Like tRFC, each qRFC-LUW will run in its own context. Be careful while using tRFC with the option AS SEPARATE UNIT. In this case, each tRFC-call with this option is a LUW and will have an own context even if all of them belong to one LUW from the send system point of view.

2.5. Programming & Sample Programs

As described above, for using qRFC with Send Queue you have to pass queue names to the qRFC manager before "call function … in background task" and to issue Commit Work at the end of a LUW as usual.

Passing Queue Names

- Call the function module "TRFC_SET_QUEUE_NAME" to pass a queue name to the qRFC manager if the call function in background task (qRFC-call) is only assigned to one send queue.

- Call the function module "TRFC_SET_QUEUE_NAME_LIST" to pass queue names to the qRFC manager if the qRFC-call is assigned to more than one send queue.

- Call the function module "TRFC_SET_QUEUE_RECEIVER_LIST" to pass queue names to the qRFC manager if the qRFC-call is assigned to one send queue and you want to send this LUW to different destinations (target systems). In this case, the qRFC manager will store the LUW-data only once in the database and create a reference counter for that LUW. All data of this LUW will only be deleted after this LUW is successfully processed in all target systems.

- Call the function module "TRFC_QUEUE_INITIALIZE" to inform the qRFC manager that your application is working with tRFC and qRFC but the first call is a tRFC-call. This is necessary because of a compatibility problem between tRFC and qRFC while assigning TID to the sub LUWs in one LUW (tRFC: one TID for different destinations; qRFC: different TIDs for different destinations).

Calling A Function Module In Background Task

After calling one of the 4 function modules described above, issue the statement "Call Function … In Background Task" as usual.

Don't forget the Commit Work at the end of your LUW.

Sample Programs

RSTRFCT0:	**Send Queue:** One send queue and one destination but variable qRFC-calls in one LUW and variable number of LUWs.
RSTRFCT1:	**Send Queue:** Working with tRFC and qRFC in "mixed mode" with differerent send queues and different destinations for one LUW.
RSTRFCT2:	**Send Queue:** Different send queues and different destinations for one LUW.
RSTRFCT3:	**Send Queue:** Different send queues and different LUWs. Some of these LUWs are dependent on other LUWs to show how the qRFC manager works.
RSTRFCT9:	**Send Queue:** Receiver List in qRFC: Process one LUW in different destinations and the qRFC manager keeps the LUW-data only once in the data base.
RSTRFCTA:	**Send Queue:** STOP/RESTART of Queues (different RESTART-conditions)
RSTRFCTB:	**Send Queue:** Getting qRFC-LUWs from a send queue located on the local or a remote system and process them on the currently running R/3.
RSTRFCTD:	**Send Queue:** qRFC in a Dialog Task and an Update Task in one LUW (from the application point of view).
RSTRFCTE:	**Send Queue:** Queue Name List in qRFC: A qRFC-call is assigned to more than one queue.
RSTRFCTF:	**Send Queue:** Performance test with and without Receiver List while using qRFC: Different Calls/LUWs/Queues/Destinations.
RSTRFCTG:	**Send Queue:** "Multi-stage" STOP/RESTARTs (at different times)

2.6. API for Queue Management

TRFC_SET_QUEUE_NAME:

- Use this function module to pass a queue name to the qRFC manager. This queue name is only valid for the next "Call Function ... In Background Task".
- If the importing parameter NOSEND is set to 'X' or 'S' then the qRFC manager won't send the LUW at "Commit Work"-time but keeps it in the qRFC table.
  - 'S': This LUW can be restarted from within SM58 or some provided function modules.
  - 'X': This LUW can only be read from a remote system.

    Once this parameter is set it applies for the whole LUW.

TRFC_SET_QUEUE_NAME_LIST:

- Use this function module if the "Call Function ... In Background Task" is assigned to more than one send queue. It is possible to use this funtion module together with TRFC_SET_QUEUE_NAME in one LUW.
- If the importing parameter NOSEND is set to 'X' or 'S' then the qRFC manager won't send the LUW at "Commit Work"-time but keeps it in the qRFC table.
  - 'S': This LUW can be restarted from within SM58 or some provided function modules.
  - 'X': This LUW can only be read from a remote system.

    Once this parameter is set it applies for the whole LUW.

TRFC_SET_QUEUE_RECEIVER_LIST:

- Use this function module if the function called by "Call Function ... In Background Task" should be processed in different target systems. The qRFC manager will store the data belong to this qRFC-call only once in the database.
- Moreover, you must set the option AS SEPARATE UNIT and DESTINATION as importing parameter for this funtion.
- If the importing parameter NOSEND is set to 'X' or 'S' then the qRFC manager won't send the LUW at "Commit Work"-time but keeps it in the qRFC table.
  - 'S': This LUW can be restarted from within SM58 or some provided function modules.
  - 'X': This LUW can only be read from a remote system.

    Once this parameter is set it applies for the whole LUW.

  Caution:

  This function module must not be used together with anyone of the two function modules described above in one LUW.

TRFC_QUEUE_INITIALIZE:

- Use this function module to inform the qRFC manger that your're working with tRFC and qRFC in one LUW ("mixed mode") and only if the first call is a tRFC-call and not a qRFC- call. This is necessary because of a compatibility problem between tRFC and qRFC while assigning TID for LUW with different destinations.

TRFC_GET_QUEUE_INFO and TRFC_GET_QUEUE_INFO_DETAILS

- Use these function modules to read the current contents of one or all send queues.

See also program RSTRFCQR.

TRFC_QOUT_STOP

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) and a destination to stop processing one or more queues.
- Setting the importing parameter FORCE to SPACE requires the qRFC manager to process only all LUWs existing in the queue at STOP-time. Otherwise, the qRFC manager will immediately stop the queue processing.
- A STOP-counter is realized to count the number of STOP-calls on a queue.
- You can also stop an empty (not existing) queue. In this case, all LUWs assigned to this queue will only stored in the queue.

See also program RSTRFCQ1.

TRFC_QOUT_ACTIVATE

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) and a destination to process one or more queues. This function is only to use if the queue is blocked because of a network or communication error or a system or application exception was raised and the problem is resolved.
- A stopped queue (by calling the function TRFC_QOUT_STOP) cannot be processed by this function.
- Setting the importing parameter FORCE to 'X' requires the qRFC manger to resend this LUW even if this LUW is in state EXECUTING. Normally, you don't need to use this because the qRFC manager will automatically try to solve such hanging problems (occurs during the qRFC manager is processing and the work process or the application server is restarted). This parameter setting is therefore only for system programs.

See also program RSTRFCQ0.

TRFC_QOUT_RESTART

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) and a destination to continue processing one or more queues, regardless of whether the queues are stopped.
- Setting the importing parameter FORCE to SPACE requires the qRFC manger to reduce the STOP-counter and if this counter is 0 it will activate the queue. Otherwise, the qRFC manager will always activate the queue, regardless of the STOP-counter.
- If a queue is stopped use this call and not TRFC_QOUT_ACTIVATE.
- Setting the importing parameter NO_ACTIVATE to 'X' requires the qRFC manger only to unlock and not to activate the queue.

See also program RSTRFCQ3.

TRFC_QOUT_RESTART_COND

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) to be restarted, a "high priority" queue name (single or generic, such as BASIS_TEST_*) and a destination to restart processing one or more queues.
- Setting the importing parameter TILL_STOP to SPACE requires the qRFC manager to restart a queue after the "high priority" queue is empty. Otherwise, this queue is to be restarted after the "high priority" queue is stopped or empty.
- A call of this function with TILL_STOP = SPACE will automatically stop the queue at the calling time, if no other STOP call was issued before.

See also program RSTRFCQ4 (till_stop = space) and RSTRFCQA (till_stop= 'X')

TRFC_QOUT_STATE

- Use this function module with specifying a queue name, a destination and optinally a SAP-client to get the status of this queue.
- A queue can have one of the following states: READY, RUNNING, CPICERR, SYSFAIL, STOP, WAITSTOP, WAITING and NOSEND, NOSENDS and WAITUPDA.
  - The state SYSFAIL results from a serious error and is triggered by a system exception or an application exception during the executing of the current LUW in the target system. In this status the queue is blocked. There is no automatic restart available. You can use SM58 to resend this LUW or to delete it. If you delete it, you have to reactivate or restart the queue.
  - In the state CPICERR an automatic retry is dependent on the configuration of the destination in SM59 (default: "Yes").
  - If the queue state is WAITSTOP or WAITING the exporting parameter WQNAME shows for which queue the current queue is waiting for processing.
  - The difference between state NOSEND and NOSENDS is the queue with state NOSENDS can be activated or restarted via SMQ1 or the according function modules. Queues with state NOSEND can only read and process by other systems or programs.

See also program RSTRFCQ2.

TRFC_QOUT_OVERVIEW

- Use this function module to get an overview about currently existing send queues belong to different optional importing parameters such as queue name, destination, SAP-client.

TRFC_QOUT_DELETE_LUW

- Use this function module to delete the LUW in a send queue defined by the importing parameter TID.

TRFC_QOUT_DELETE_QUEUE

- Use this function module to delete the whole send queue defined by the importing parameters QNAME and DEST.

TRFC_SET_BACKGROUND_SEQUENCE

- Use this function module to set the processing sequence of qRFC-LUWs issued from within a Dialog Task and an Update Task in one transaction.

TRFC_QOUT_GET_AND_PROCESS

- Use this function module to read qRFC-LUWs from Send Queue located in the local or remote R/3 system.
- Even if all or a few of qRFC-LUWs can be read once from a send queue, according to the importing parameter "single_processing" these LUWs will be processed as a LUW for each LUW or as a LUW for all read qRFC-LUWs.
- Using single processing, the qRFC manager returns a error code and error message to the caller if during the processing of an LUW a system or application exception is raised. All successfully processed LUWs will be deleted in the send queue. The qRFC-LUW with error stays in the send queue and the send queue won't be informed about this error situation (only the caller of this function module is informed).
- If not using single processing, a system or application exception will cause a short dump and all read LUWs still stay in the send queue. They can be read and processed again later.

TRFC_QOUT_READ

- Use this function to read the first LUW in a queue (Internal Use: only for BW and CRM)

TRFC_QOUT_CONFIRM

- Use this function to confirm a executed LUW in queue (Internal Use: only for BW and CRM)

TRFC_QOUT_READ_NEXT

- Use this function to read next LUWs in a queue even if the first ones are still not confirmed (Internal Use Only: for BW and CRM)

TRFC_QOUT_CONFIRM_NEXT

- Use this function to confirm a list of executed LUWs in a queue (Internal Use: only for BW and CRM)

2.7. Monitoring

Use the transaction SMQ1 as monitor for qRFC with Send Queue.

Even if the tRFC and qRFC use the same DB-tables (ARFCSSTATE and ARFCSDATA) the tRFC-monitor (transaction SM58) is only for processing tRFC-LUWs. It just displays tRFC-LUWs and a delete or reorganization of these DB-tables from within SM58 will only erase tRFC-LUWs.

The programs RSTRFCQR, RSTRFCQD and RSTRFCDP are only for internal use.

Because qRFC must also run in rel. 3.1, 4.0 and 4.5 and the installation must not only be done via Hot Packages, there is a version defined for qRFC for a better management of the qRFC software. All R/3 systems using qRFC must have at least the qRFC version 4.6A.002. This version has the same qRFC-functionalities as provided in the official release 4.6A.

You can get the current qRFC version on your R/3 system from within this qRFC-monitor via Menu → Info → Version.

2.8. When To Use qRFC With Send Queue Instead Of tRFC

Use qRFC whenever you have to serialize different tRFC-LUWs!

In some other cases, even if you don't have to serialize but your partner system/program can't deal with more than one or a certain number of LUWs at a time (because of programming, technical or performance problems), you should use qRFC with Send Queue instead of tRFC.

Particularly, if your tRFC partner is an external tRFC server running in "register" mode you should you should use qRFC with Send Queue instead of tRFC because

- the RFC library only allows your server program to deal with one tRFC-LUW after another.
- the serialization will be done by the SAP gateway:

  If the server program is busy while processing an LUW all following tRFC-LUWs must wait at the SAP gateway until the server program is ready again or until timeout (def.: 2 min). In timeout cases, a batch job will be scheduled with a delay for each LUW to be restarted (configurable using SM59).

Using qRFC in this case, you should choose the destination used in a tRFC-call as queue name.

Even if you have more than one server program registered at an SAP gateway under the same Program-ID you should use qRFC and have probably the same number of queues as the number of these running server programs . In this case, the queue names to choose are dependent on the business logic of your work.

2.9. Automatic Change From qRFC To tRFC

Normally, if the executing of a qRFC-LUW is broken with error SYSTEM_FAILURE then the queue will hang. If you want to use qRFC instead of tRFC for better system performance and not for serializing, you can set the optional parameter TRFC_IF_SYSFAIL while calling the function module TRFC_SET_QUEUE_NAME or TRFC_SET_QUEUE_NAME_LIST to 'X'.

In this case, all LUW's with error SYSTEM_FAILURE will be automatically changed to tRFC-LUWs and the qRFC-manager will execute the next LUW in this queue. You can then restart the broken tRFC-LUWs via SM58 as usual.

3. qRFC With Receive Queue

3.1. Characteristics

How to work with qRFC with Receive Queue

- Use the qRFC with Receive Queue if an application wants to determine itself when to process the incoming qRFC-LUWs within the target system or it just wants to have all LUWs as soon as possible stored in the target R/3 system. These LUWs will be processed later.

- Using qRFC with Receive Queue requires some extensions in the tRFC protocol to inform the target system that the currently incoming LUW should only be stored in the receive queue of qRFC.

- To do this, an application must use the standard qRFC funciton modules to inform the qRFC manager of the sending system about using qRFC with Receive Queue in the target system. Depending on the sending system, an application must then call a function module from within R/3 or a new provided API in the RFC library to pass a queue name and optionally a queue counter to the target system.

Queue Name, Queue Identifier

- To use qRFC with Receive Queue in the target system, an application must pass the queue names by calling a standard qRFC function module immediately before issuing "Call Function … In Background Task".

- A queue name is a 24 byte string. It must not include *, &, % or small (lower-case) letters. Moreover it must not begin with a blank.

- To prevent different applications from unadventantly using the same receive queue, each application should choose a specific prefix for queue names (e.g. SFA_... or APO_...).

- As queue identifier for serialization, the qRFC manager uses the current SAP-client and the queue name.

- There is no queue configuration necessary.

Serialization

- The serialization of qRFC-LUWs in the receive queue of the target system can be defined by the sending application via the queue counter above. If this counter is 0 the target system will create a queue counter depending on the time when this LUW is incoming.

Activate a Receive Queue

- To process one or more Receive Queues, an application from within the target system must use some provided function modules to manage (activate, …) the receive queues.

qRFC with Send/Receive Queue

- The only way to work with qRFC with Receive Queue in a target system is to use qRFC with Send Queue in the sending system (better system performance and automatic restart of hanging queues).

Handling of CPIC-errors

- Because the activating of an LUW in a receive queue needs the local SAP gateway for processing, CPIC-errors can happen. Contrary to qRFC with Send Queue, there is no automatic restart for the Receive Queue via batchjob. The application which activates or restarts this queue will be informed about the error situation and it must then activate or restart this queue later again.

Reuse feature of qRFC with Receive Queue

- A special feature of qRFC with Receive Queue is you can save a curently executing LUW. Later, you can restore some LUWs saved before in the receive queues (to reuse) as they are just incoming with the old queue information (queue names, queue counters) or you can delete them. This feature is used e.g. by APO with LiveCache.

3.2. LUW, TID & Queue Counter

There is an one-to-one assignment between the TIDs of the sending and target system. Because of a technical problem while using tRFC, the qRFC manager in the target system will create a new TID for an incoming qRFC-LUW and map the sending TID to the new TID before storing the qRFC-LUW in the receive queue for a later processing.

As described above, the queue counter for each LUW can be passed by the sending system or will be created by the target system when an LUW is incoming.

Passing queue counter by the sending application

Pay attention for the following conditions:

- Different TIDs (LUWs) that belong to one receive queue must have different counters.
- Different TIDs in different receive queues can have the same counter.
- If different sending applications/systems write to the same queue they must synchronize for setting the queue counter (different counters for different TIDs in the same receive queue) or if they can't they must let the target system create a queue counter for the incoming LUWs. The counter will be set at the incoming time.

3.3. User and user-context of a qRFC-LUW

Because the LUWs written in a receive queue won't be automatically executed, these LUWs will run under the user which calls the function module for activating or restarting this queue.

If you're using the QIN-Scheduler you can register a queue with specifying an "local" RFC-destination (Generally, this is a logical destination in SM59 and the reference destination is NONE or SPACE and a user is defined in this entry, s.Chapter 3.6 for more details).

3.3. Programming & Sample Programs

As described above, using qRFC with Receive Queue requires using qRFC with Send Queue from within the sending system. So you have to pass a send and a receive queue name and optionally a receive queue counter to the qRFC manager before calling the according function module in background task as usual.

To do this, call "TRFC_SET_QIN_PROPERTIES" with the required parameters and don' forget the Commit Work at the end of your LUW.

Sample Programs

RSTRFCT5:	**Receive Queue: O**ne send queue, one destination and one receive queue but variable qRFC-calls in one LUW and variable number of LUWs.
RSTRFCT6:	**Receive Queue:** tRFC and qRFC in "mixed mode" with differerent send queus, different destinations and different receive queues in one LUW.
RSTRFCT7:	**Receive Queue: D**ifferent send queues, different destinations and different receive queues in one LUW.
RSTRFCT8:	**Receive Queue:** Different receive queues in different LUWs. Some of these LUWs are dependent of other LUWs to show how the qRFC manager works.
RSTRFCTC:	**Receive Queue:** Working with the "reuse"-feature of the Receive Queue.
RSTRFCTF:	**Receive Queue:** "Multi-stage" STOP/RESTARTs (at different times)

3.4. API for Queue Management

TRFC_SET_QIN_PROPERTIES:

- Use this function module to pass a send queue name, a receive queue name and optionally a queue counter to the qRFC manager. These parameters are only valid for the next Call Function In Background Task. The send and receive queue can have the same queue name.

See also program RSTRFCT5.

TRFC_GET_QIN_INFO and TRFC_GET_QIN_INFO_DETAILS

- Use these function modules to read the current contents of one or all receive queues.

See also program RSTRFCIR.

TRFC_QIN_GET_CURRENT_PARAM

- Use this function module to get information about the receive queues such as queue names, queue counter belonging to the current LUW. You will need this information when you're working with the reuseable feature of qRFC with Receive Queue.

TRFC_QIN_STOP

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) to stop processing one or more queues.
- Setting the importing parameter FORCE to SPACE requires the qRFC manager to process only all LUWs existing in the queue at STOP-time. Otherwise, the qRFC manager will immediately stop the queue processing.
- A STOP-counter is realized to count the number of STOP-calls.
- You can also stop an empty (not existing) queue. In this case, all LUWs assigned to this queue will only be stored in the qRFC tables.

See also program RSTRFCI1.

TRFC_QIN_ACTIVATE

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) to process one or more queues. This function is only to be used if the queue is blocked because of a communications error or a system or application exception was raised and the problem is resolved It is also to be used if the LUWs are stored in the Receive Queue and need to be activated.
- A stopped queue (by calling the function TRFC_QIN_STOP) cannot be processed by this function.
- Setting the importing parameter MAXLUW to a non 0 value requires the qRFC manger to activate all LUWs in the receive queue.

- Setting the importing parameter MAXTIME to a value not 0 (0: unlimited; the call is returned after the queue is empty) requires the qRFC manger only to let the queue activate within this time. If the time is exceeded while the last LUW is executing this call will be returned after the last LUW is ended.
- Setting the importing parameter MODE to 'B' requires the qRFC manager to activate the queue (execute the qRFC-LUWs) in a BATCH-Task. The importing parameters for starting this batchjob are optional (Def.: immediately).
- Setting the importing parameter MODE to 'L' requires the qRFC manager to execute alls qRFC-LUWs in a queue in the same context of the caller of this call. If a function module in this queue raises an exception it will cause a short dump because the qRFC manager doesn't have the control to catch this exception running in this mode.
- Setting the importing parameter USERDEST to a "local" RFC-destination (e.g. a logical destination with NONE or SPACE as reference destination) in SM59 requires the qRFC manager to execute alls qRFC-LUWs in a queue under the user defined in this entry. If no destination is specified or the user is not defined, the current user will be used.

See also program RSTRFCI0.

TRFC_QIN_RESTART

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) to continue processing one or more queues, regardless of whether the queues were stopped before or not.
- Setting the importing parameter FORCE to SPACE requires the qRFC manger to reduce the STOP-counter and if this counter is 0 it will activate the queue. Otherwise, the qRFC manager will always activate the queue, regardless of the STOP-counter.
- Setting the importing parameter MAXLUW to a non 0 value requires the qRFC manger to activate all LUWs in the receive queue
- If a queue is stopped use this call and not TRFC_QIN_ACTIVATE.
- Setting the importing parameter NO_ACTIVATE to 'X' requires the qRFC manger only to unlock and not to activate the queue.
- Setting the importing parameter MAXTIME to a value not 0 (0: unlimited; the call is returned after the queue is empty) requires the qRFC manger only to let the queue activate within this time. If the time is exceeded while the last LUW is executing this call will be returned after the last LUW is ended.
- Setting the importing parameter MODE to 'B' requires the qRFC manager to activate the queue (execute the qRFC-LUWs) in a BATCH-Task. The importing parameters for starting this batchjob are optional (Def.: immediately).
- Setting the importing parameter MODE to 'L' requires the qRFC manager to execute alls qRFC-LUWs in a queue in the same context of the caller of this call. If a function module in this queue raises an exception it will cause a short dump because the qRFC manager doesn't have the control to catch this exception running in this mode.
- Setting the importing parameter USERDEST to a "local" RFC-destination (e.g. a logical destination with NONE or SPACE as reference destination) in SM59 requires the qRFC manager to execute alls qRFC-LUWs in a queue under the user defined in this entry. If no destination is specified or the user is not defined, the current user will be used.

See also program RSTRFCI3.

TRFC_QIN_RESTART_COND

- Use this function module with specifying a queue name (single or generic, such as BASIS_TEST_*) to be restarted and a "high priority" queue name (single or generic, such as BASIS_TEST_*) to restart processing one or more queues.
- Setting the importing parameter TILL_STOP to SPACE requires the qRFC manager to restart a queue after the "high priority" queue is empty. Otherwise, this queue is to be restarted after the "high priority" queue is stopped or empty.
- A call of this function with TILL_STOP = SPACE will automatically stop the queue at the calling time, if no other STOP call was issued before.

See also program RSTRFCI4.

TRFC_QIN_STATE

- Use this function module with specifying the queue name and optionally the SAP-client to get the state of this queue.
- A queue can have one of the following states: READY, CPICERR, SYSFAIL, STOP, WAITSTOP and WAITING.
  - The state SYSFAIL results from a serious error and is triggered by a system exception or an application exception during the executing of the current LUW. In this state the queue is blocked. There is no automatic restart available. Currently, you can use the program RSTRFCI0 to reactivate this LUW or use the program RSTRFCID to delete the qRFC-LUW.
  - In state CPICERR there is also no automatic retry available. This error can only occur if the SAP gateway on the local application server doesn't work any longer. Use the RSTRFCI0 to reactivate the queue once the gateway problem is resolved.
  - If the queue state is WAITSTOP or WAITING the exporting parameter WQNAME shows for which queue the current queue is waiting for processing.

See also program RSTRFCI2.

TRFC_QIN_OVERVIEW

- Use this function module to get an overview about currently existing receive queues belong to different optional importing parameters such as queue name, SAP-client.

TRFC_QIN_DELETE_LUW

- Use this function module to delete the LUW in a receive queue defined by the importing parameter TID.

TRFC_QIN_DELETE_QUEUE

- Use this function module to delete the whole receive queue defined by the importing parameter QNAME.

TRFC_QIN_SAVE_CURRENT_LUW

- Use this function module during the execution of any function module from within an LUW to inform the qRFC manger to save the current LUW (for reuse) after this LUW is successfully executed.

TRFC_QINS_RESTORE

- Use this function module to let the qRFC manager restore some or all saved LUWs defined by the importing parameters QNAME and FROM_QCOUNT.

TRFC_QINS_ERASE

- Use this function module to let the qRFC manager delete some or all saved LUWs defined by the importing parameters QNAME and TO_QCOUNT.

TRFC_QINS_OVERVIEW

- Use this function module to get an overview about currently existing "saved" receive queues belong to different optional importing parameters such as queue name, SAP-client.

TRFC_QINS_DELETE_LUW

- Use this function module to delete the LUW in a "saved" receive queue defined by the importing parameter TID.

TRFC_QINS_DELETE_QUEUE

- Use this function module to delete the whole "saved" receive queue defined by the importing parameter QNAME.

TRFC_RECEIVER_INFO

- Use this function module to get some information from the sender of a tRFC- or qRFC-LUW such as TID, client, user, hostname of the application server, program name, transaction code. You can also use this funtion module to check whether a function module is called by tRFC or qRFC with Receive Queue.

3.5. Monitoring

Use the transaction SMQ2 as monitor for qRFC with Receive Queue and SMQ3 as monitor for qRFC with "saved" (reuse-feature) Receive Queue.

The programs RSTRFCIR, RSTRFCID, RSTRFCIS, RSTRFCIT and RSTRFCDQ are only for internal use.

Because qRFC must also run in rel. 3.1, 4.0 and 4.5 and the installation must not only be done via Hot Packages, there is a version defined for qRFC for a better management of the qRFC software. All R/3 systems using qRFC must have at least the qRFC version 4.6A.002. This version has the same qRFC-functionalities as provided in the official release 4.6A.

You can get the current qRFC version on your R/3 system from within the qRFC-monitor (SMQ2 or SMQ3) via Menu → Info → Version.

3.6. The QIN-Scheduler

Normally, all LUWs written in receive queues won't be automatically executed. Each application can activate these queues by using some API described above. To avoid the same coding for a scheduler in each application working in the same way, qRFC supports a general scheduler called the "QIN-Scheduler". Any qRFC-application can also implement its own scheduler to activate the queues it has to deal with.

The qRFC manager will schedule a batchjob for starting the QIN-Scheduler after a qRFC-LUW is written in a queue which is already registered to automatically execute this LUW.

There will be a QIN-Scheduler running for each client in an R/3 system.

A QIN-Scheduler will run under the user which has been written in a registered queue but you can also decide the user under which the LUWs in a registered queue should be executed. This can be done by registering a queue with specifying a RFC-detination and the according entry in SM59 includes the SAP-Logon data (Generally, this is a logical destination in SM59 and the reference destination is NONE or SPACE and a user is defined in this entry).

Moreover, you can also use the transaction SMQR or call the funciton module QIWK_CHANGE_GROUP to change the group of application servers where all registered queues should be executed to a new one. You can use the transaction RZ12 to define the group of these application servers.

The Transaction SMQR

Using this transaction for displaying the state of the QIN-Scheduler and register or unregister a queue.

The QIN-Scheduler can have one of the following states:

- BATCHJOB SCHEDULED	The QIN-Scheduler has been scheduled.
- ACTIVE	The QIN-Scheduler is running.
- INACTIVE	The QIN-Scheduler is not active.
- WAITING	The QIN-Scheduler is waiting for free workprocesses.
- SYSFAIL	System_Failure occurred (The QIN-Scheduler can't work properly)
- CPICERR	Communication_Failure occurred (The QIN-Scheduler can't work properly)

In case of SYSFAIL or CPICERR, a double-click on the state show the error long text.

Moreover, if the QIN-Scheduler is active it will update the time every 2 minutes or if the state of the QIN-Scheduler has been changed to show that it is really working.

Normally, the QIN-Scheduler will execute all registered queues on any application server of the local R/3 system (Name of AS-group: DEFAULT) but you can also decide the group of these application servers. A group can be specified via SM59 and RFC → RFC-Group or directly via RZ12 and you can change this group from within this transaction SMQR (Edit → Change AS-group).

Register A Queue

- You can register a queue by using the transaction SMQR or call the function module QIWK_REGISTER (single or generic such as BASIS_TEST_*).
- Be careful for register a queue with the queue name *. It means all receive queues will be automatically executed by the QIN-Scheduler.
- Specify the executing mode via parameter EXEMODE to 'D' or 'B' to execute the qRFC-LUWs in this queue in a DIALOG-WP or in a BATCH-WP (WP: Workprocess).
- Specify a max. time via parameter MAXTIME to let the queue run nearly within this time. This is necessary for the scheduler to have a fair handling of a large number of different queues but not enough work processes on different application servers.
- If a queue is just registered via transaction SMQR or by calling the function module QIWK_REGISTER and this queue is not empty and the QIN-Scheduler is not running, it will be immediately activated (except the optional parameter NO_ACTIVATE is set to 'X'). If the QIN-Scheduler is already running it will take some seconds to get the new information from the Register-Queue-Table (QIWKTAB).
- Specify the user destination via parameter USERDEST with a defined user in the according entry in SM59, you can let the QIN-Scheduler execute all LUWs in a queue under this defined user.

Unregister A Queue

You can unregister a registered queue by using the transaction SMQR or call the function module QIWK_UNREGISTER (single or generic such as BASIS_TEST_*).

You can also unregister a queue even if a queue with wild card (includes your queue) is already

registered (via SMQR or call the function QIWK_UNREGISTER with parameter LIMITED set to 'X').

Get All Registered / Unregistered Queues

You can get the status of the QIN-Scheduler and all registered and unregistered queues by using the transaction SMQR or call the function module QIWK_GET_ALL.

How does the QIN-Scheduler work?

```
                                                ------------------              -------------------
          ------------------                    |                  |-------------->  |  DIALOG-        |
LUWs      |                  |                  |                  |  exemode=D      |    WP  1        |
------>   |    tRFC-         |                  |     QIN-         |  Activate       -------------------
 come     |  Manager         |                  |   Scheduler      |  via            …
   in     |                  |                  |                  |  parallel RFC   ------------------
          -------------------          |                  |-------------->  |  DIALOG        |
                    |                           -------------------         |    WP  n       |
          It's a    |                             A         |                -------------------
        qRFC-       |               Start          |         |
          LUW       |          --------------------- |------------------------->   -------------------
                   | |                            |-------------------  |  BATCH         |
                   V|                                                |         |    WP 1        |
          -----------------------------------------------------     |         -------------------
          |                                                          |         |          …
          |                    qRFC-Manager                 |        |         -------------------
          |                                                          |         ------>  |  BATCH         |
          -----------------------------------------------------               |    WP n        |
                                     |  Get                                             -------------------
                                     |  registered
                                     |  queues
                                     V
          ------------------------------------------------------------------------------------------------
          | Clt       Q-Name Type       Mode     M-Time        Local Dest with LOGON-data |
          | 000       TEST_* R          D        5                 BASISUSER000           |
          | 000       TEST_2 U          D        0                 BASISUSER000           |
          | 000       TEST_3 U          D        10                BASISUSER000           |
          | …                                                                                              |
          ------------------------------------------------------------------------------------------------
```

Monitor for QIN-Scheduler (SMQR)

The QIN-Scheduler will be activated by the qRFC-manager via a batchjob after a qRFC-LUW is written in a registered queue and the QIN-Scheduler is not already running. There will be a QIN-Scheduler for each client (The receive queues are client-dependent).

Once the QIN-Scheduler is running, it works as follows:

1. It looks at the DB-table for registered queues and checks whether at least one registered queue is not empty.

2. If a queue should be activated in a BATCH-WP, a batchjob will be scheduled for activating this queue in a BATCH-WP.

3. The QIN-Scheduler can't activate all registered queues at the same time because there might be more queues to run parallel than the current resources (work processes of all application servers). Therfore, it will activate these queues as many as possible belonging to the incoming time of the first LUW in each queue by using parallel RFC (call function … starting new task … destination in group GROUP_NAME performing … on end of task).

4. Normally, the QIN-Scheduler runs with DEFAULT as GROUP_NAME. It means every application server of the local R/3 system will be used for activating all registered queues. You can also change this GROUP_NAME as described above.

5. Each queue will only run (nearly) within the configured MAXTIME. Setting this parameter to a value not 0 (0: unlimited) requires the qRFC manager only to let the queue activate within this time (in sec.). There is no guarantee for an exact time because if this time is exceeded while a LUW is still executing, this LUW won't be canceled but it will normally finish.

6. If all DIALOG workprocesses for parallel RFC are busy and the QIN-Scheduler, therefore receives a resource error, it then updates its state (to see via SMQR) and waits for free workprocesses.

7. If a system or communication error occurs while asynchronous start of a workprocess for activating a queue via parallel RFC, the QIN-Scheduler will save the returned error message and finish its work because it is a fatal error. The transaction SMQR shows then SYSFAIL or CPICFAIL as the scheduler state. A double-click on this state displays the error message. The QIN-Scheduler will try again by restarting via the qRFC manager.

8. After each queue is activated once, the QIN-Scheduler continues with step 1. At this time, it will get information about the new registered or unregistered queues or the change of a new group of application servers during the last run.

4. qRFC-Support of the RFC Library

4.1. tRFC Server

Instead of using tRFC, an R/3-application can use qRFC with Send Queue from within an R/3 system to call RFC functions provided in an tRFC server program using the RFC library.

Any R/3-application working with an external tRFC server program can use the qRFC with Send Queue instead of tRFC for a LUW-serialization and better performance on the sending R/3 system.

4.2. qRFC Client

Instead of RfcIndirectCall or RfcIndirectCallEx, an tRFC client program must use the new call RfcQueueInsert to inform the target R/3 system to store the incoming LUW in the defined Receive Queue for a later processing. If the call RfcQueueInsert is successfully returned the qRFC client must then issue RfcConfirmTransID. Otherwise, it must repeat the call RfcQueueInsert using the same TID as before.

A qRFC client program must issue following RFC calls:

RfcOpen

Connect to an R/3 system.

RfcCreateTransID

With this call, the RFC-Library tries to get a TID created by the R/3 system.

In error cases, the qRFC client program has to reconnect later and must try to repeat this call. Otherwise, the qRFC client program can assign this TID with the qRFC data.

RfcQueueInsert

The RFC library will pack all data belong to an RFC function together with the TID, the queue name and optionally the queue counter and send them to the R/3 system using tRFC protocol (with extension for qRFC with Receive Queue).

In error cases, the qRFC client program has to reconnect later and must try to repeat this call. In this case, it has to use the old TID and must not get a new TID with RfcCreateTransID. Otherwise, it is no guarantee that this RFC function will be exactly once executed in R/3 system.

After a successful execution of this call, the LUW is completely stored in the required receive queue. The qRFC client program can then update its own TID-management (e.g. delete the TID-entry).

Contrary to tRFC/qRFC between R/3 systems, an LUW from within an qRFC client program contains only one RFC function.

RfcConfirmTransID

The qRFC client must update the state of this TID as ‘executed’ in its own TID-management before it issues RfcConfirmTransID to confirm the writing of this LUW in the according receive queue.

The delivered program trfctest.c is a tRFC client program. You can replace the RfcIndirectCallEx with RfcQueueInsert, put the required parameter about the queue name and optionally a queue counter to this new API to use it as a qRFC client program using qRFC with Receive Queue in the target R/3 system.

CAUTION:

If the target R/3 system does not support qRFC with Receive Queue, then it will process the incoming LUW immediately and does not return any error to the qRFC client program. In this case, the qRFC client program works as a tRFC client program.

5. History of qRFC-Versions

Version 4.6A.001

- Inofficial first release for APO

Version 4.6A.002

- First release

Version 4.6C.001

- Important corrections
  - ♦ Lost tRFC/qRFC-LUWs during system overloading
- Better qRFC-Performance
- Some new functionalities in qRFC

Version 4.6C.005

- Support the QIN-Scheduler
- Better qRFC-Performance

Version 4.6C.006

- Important corrections
  - ♦ Restart/Resend problem in qRFC and with the QIN-Scheduler
  - ♦ Lost LUWs or Nosend of LUWs in SYSLOAD-situations
- Better qRFC-Performance
- New qRFC monitors for send and receive queues

Version 4.6C.007

- Important corrections
  - ♦ NOSEND didn't work correctly when running in Update-Task.
  - ♦ Each qRFC-LUW runs in an own user-context (Till now, it's undefined).
- Better qRFC-Performance in qRFC-monitors
- New interfaces of qRFC monitors for send and receive queues

- Exception will be raised if the message server is in trouble.

Version 4.6D.009

- Important corrections
  - User-specific SYSLOAD-Handling (SYSLOAD: no work process free for tRFC/qRFC)
- STOP and RESTART a queue at different times possible
- Unregister a queue is now available even if a generic queue (with wildcard) is registered at the QIN-Scheduler.
- Better overview about state of the QIN-Scheduler with SMQR.
- Better qRFC-performance
- Full Version

Version 4.6D.010

- Important corrections
  - In SYSLOAD cases all batch processes were busy by RSTRFCSL..
  - Different SQL-errors possible because SQL-statements weren't written in capital letters.
- Different changes in QIN-Scheduler for better handling of registered Receive Queues

Appendix qRFC: Contents

- Why qRFC
- How to use qRFC
- QIN Scheduler
- Additional information

Why qRFC?

- RFC is not safe in case of network errors.
- tRFC is safe ("exactly once" execution) but no support of LUW serialization
- Executing a tRFC LUW with delay or in a batch work process is not available.
- Bad performance on both sending and receiving system when using lots of tRFCs at the same time

What is qRFC?

- Calling Function Modules via tRFC
- Serialization using queues
- Two types of queues
  - Send Queue (or Outbound Queue or OUT Queue)
  - Receive Queue (or Inbound Queue or IN Queue)
- Automatically activating of registered Receive Queues available via the QIN Scheduler

qRFC Documentation

- R/3 Online Help from 4.6A onwards
- MS Word document
  - sapservX, directory /general/R3server/abap/note.0166096

How To Use qRFC?

1. **qRFC with Send Queue**

2. **qRFC with Receive Queue**

3. **The QIN Scheduler**

How To Use qRFC?

qRFC with Send Queue

qRFC with Send Queue: Sample Program

```
report rstrfct0.

...

call function 'TRFC_SET_QUEUE_NAME'
 exporting
  qname = 'BASIS_TEST_Q1'.

call function 'RFC_FUNCTION'
 destination RFCDEST
 in background task
 ( as separate unit )
 exporting ...
 tables ...

...

commit work.
```

How To Use qRFC with Send Queue?

- Set Queue Name via calling one of the following function modules
  - TRFC_SET_QUEUE_NAME
  - TRFC_SET_RECEIVER_LIST
  - TRFC_SET_QUEUE_NAME_LIST
- Call tRFC as usual (call function … in background task)
- At ‘Commit Work’ time, a counter will be created for the current LUW by the qRFC manager.
- Serialization via Queue Identifier (Client, Q Name, Destination & Q Counter)

qRFC with Send Queue: Characteristics 1

- LUW Serialization in a Send Queue via Queue Identifier (Client, Q Name, Destination & Q Counter)
- Minimum load on sending and receiving system due to serialization
- Better handling of LUWs in overload cases (No work process free to run tRFC asynchronously)
- Automatic restart of “hanging” LUW
- NOSEND(S) Attribute for qRFC LUWs
- Each qRFC LUW runs in its own context.
- qRFC with Send Queue is compatible to tRFC: No changes of tRFC coding on receiving system necessary when using qRFC calls instead of tRFC calls for better system performance

qRFC with Send Queue: Characteristics 2

- Relationship between LUW & TID
  - Each LUW begins with the first “call function … in background task’ (tRFC call) and ends with the first “commit work”.
  - Each LUW can include different SubLUWs
  - A SubLUW includes
    - one tRFC call when using option AS SEPARATE UNIT.
    - all tRFC calls in a LUW using the same destination.
  - A TransactionID (TID) is assigned to a SubLUW.
- A LUW can include tRFC and qRFC calls (mixed mode)

qRFC with Send Queue: Characteristics - 3

- The Queue API exists in form of function modules to manage the queues such as lock, unlock, activate or ask for queue state.
- CAUTION: Executing qRFC LUWs on receiving system under different SAP user possible if no user is defined in SM59 for the currently used destination.
- For applications which use qRFC with Send Queue for better system performance and not for serialization: An automatic change from qRFC LUW with Send Queue into tRFC LUW is possible to avoid a “hanging” queue in case of SYSTEM_FAILURE.

qRFC with Send Queue: Queue Management

- Comfortable queue administration via qRFC Monitor for Send Queues (SMQ1)
- Some important APIs for Queue Management
  - TRFC_QOUT_OVERVIEW
  - TRFC_QOUT_STATE
  - TRFC_QOUT_ACTIVATE
  - TRFC_QOUT_STOP
  - TRFC_QOUT_RESTART
  - TRFC_QOUT_RESTART_COND
  - TRFC_QOUT_READ_NEXT
  - TRFC_QOUT_CONFIRM_NEXT
  - TRFC_QOUT_GET_AND_PROCESS

qRFC with Send Queue and the RFC Library

- qRFC with Send Queue from within the RFC Library is not supported.
- Any application in R/3 can use qRFC with Send Queue instead of tRFC to communicate with an external tRFC server:
  - Minimal changes in ABAP coding
  - No changes in coding of tRFC server necessary
  - Better system performance
- If the tRFC server runs in “register” mode, the R/3 application should use qRFC with Send Queue instead of tRFC and the destination should be set as Queue Name for better system performance and also better data transfer.

How To Use qRFC?

qRFC with Receive Queue

qRFC with Receive Queue: Sample Program

```
report rstrfct5.

...


call function 'TRFC_SET_QIN_PROPERTIES'
 exporting
  qout_name = 'BASIS_TEST_Q1'
  qin_name  = 'BASIS_TEST_Q1'.


call function 'RFC_FUNCTION'
 destination RFCDEST
 in background task
 ( as separate unit )
 exporting ...
 tables ...

...

commit work.
```

How To Use qRFC with Receive Queue?

- Set Name of Send/Receive Queue, optionally a counter via TRFC_SET_QIN_PROPERTIES
- Call tRFC as usual (call function … in background task)
- At ‘Commit Work’ time, a counter will be created for the current LUW (for the Send Queue).
- At the receiving system a counter will be created for serialization if counter isn’t sent by sender.
- Using the QIN Scheduler for automatically activating a receive queue is available.
- Serialization via Queue Identifier (Client, Q Name & Q Counter)

qRFC with Receive Queue: Characteristics 1

- LUW Serialization in a Receive Queue via Queue Identifier (Client, Q Name & Q Counter)
- Sending application can determine its own counter for the LUW to be written in the Receive Queue of receiving system.
- Different Queues can have the same counter.
- Executing of qRFC LUWs asynchronously to writing in receive queue
- Each qRFC LUW runs in its own context.
- LUWs in a Receive Queue can be “reused”.

qRFC with Receive Queue: Characteristics 2

- There are Queue APIs in form of function modules to manage queues such as lock, unlock, activate or ask for queue state.
- To execute qRFC LUWs in a batch process you need to run a program in batch and call the Queue API locally for queue activate.
- Each qRFC application can determine the time when a queue should be activated.
- The QIN Scheduler will automatically activate a Receive Queue if this queue is already registered.
- CAUTION: Executing of qRFC LUWs under different SAP user possible if USERDEST is not defined while registering a queue at the QIN Scheduler.

qRFC with Receive Queue: Queue Management

- Comfortable queues administration via qRFC Monitor for Receive Queues (SMQ2)
- Some important API for Queue Management
  - TRFC_QIN_OVERVIEW
  - TRFC_QIN_STATE
  - TRFC_QIN_ACTIVATE
  - TRFC_QIN_STOP
  - TRFC_QIN_RESTART
  - TRFC_QIN_RESTART_COND
  - TRFC_QIN_SAVE_CURRENT_LUW
  - TRFC_QINS_OVERVIEW
  - TRFC_QINS_RESTORE
  - TRFC_QINS_ERASE

qRFC with Receive Queue and the RFC Library

- qRFC with Receive Queue from within the RFC Library is not supported.
- Use the new API Call RfcQueueInsert instead of RfcIndirectCall or RfcIndirectCallEx in a tRFC client program to write a LUW directly into a Receive Queue of a R/3 system (available with the RFC library from 4.6A onwards).
- No more wait for end of LUW executing necessary when using RfcQueueInsert.
- After the call RfcQueueInsert is returned you have to issue RfcConfirmTransId as usual.
- Register your queue at the QIN Scheduler for automatically activate.

The QIN Scheduler

The QIN Scheduler

The QIN Scheduler: Register A Queue

- Call ‘QIWK_REGISTER’ or transaction SMQR with following parameters:

■ Queue Name:	Wild card (e.g. TEST_*) available
■ EXEMODE:	D/B: Dialog or Batch
■ MAXTIME:	max. runtime for a queue
■ USERDEST:	Logical Destination in sm59 with reference to “NONE” to execute LUWs under a certain SAP user
■ NRETRY:	Number of retries in error cases
■ TDELAY:	Time between 2 retries

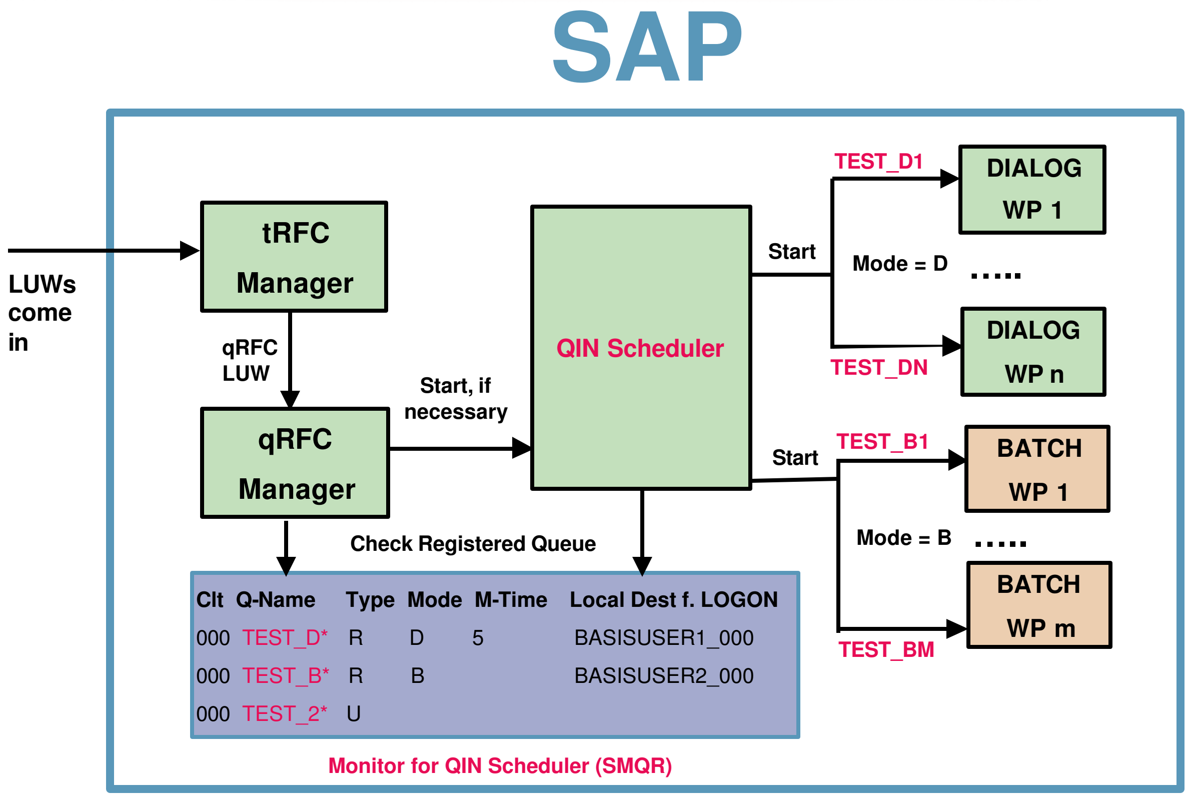
  - With or without immediately activating the QIN Scheduler
  - With or without “Commit Work”
- ‘QIWK_REGISTER’ is RFC-enabled.

The QIN Scheduler: Unregister A Queue

- Call ‘QIWK_UNREGISTER’ or via transaction SMQR with following parameters:
  - Queue Name: Wild card (e.g. TEST_*) available
  - LIMITED: Unregister queue even if a generic queue included this queue is registered. This is necessary to debug the executing of a LUW in a queue but a generic queue is registered.
  - With or without “Commit Work”

- ‘QIWK_UNREGISTER’ is RFC-enabled.

How does the QIN Scheduler work?



Clt	Q-Name	Type	Mode	M-Time	Local Dest f. LOGON
000	TEST_D*	R	D	5	BASISUSER1_000
000	TEST_B*	R	B		BASISUSER2_000
000	TEST_2*	U			

Monitor for QIN Scheduler (SMQR)

What else?

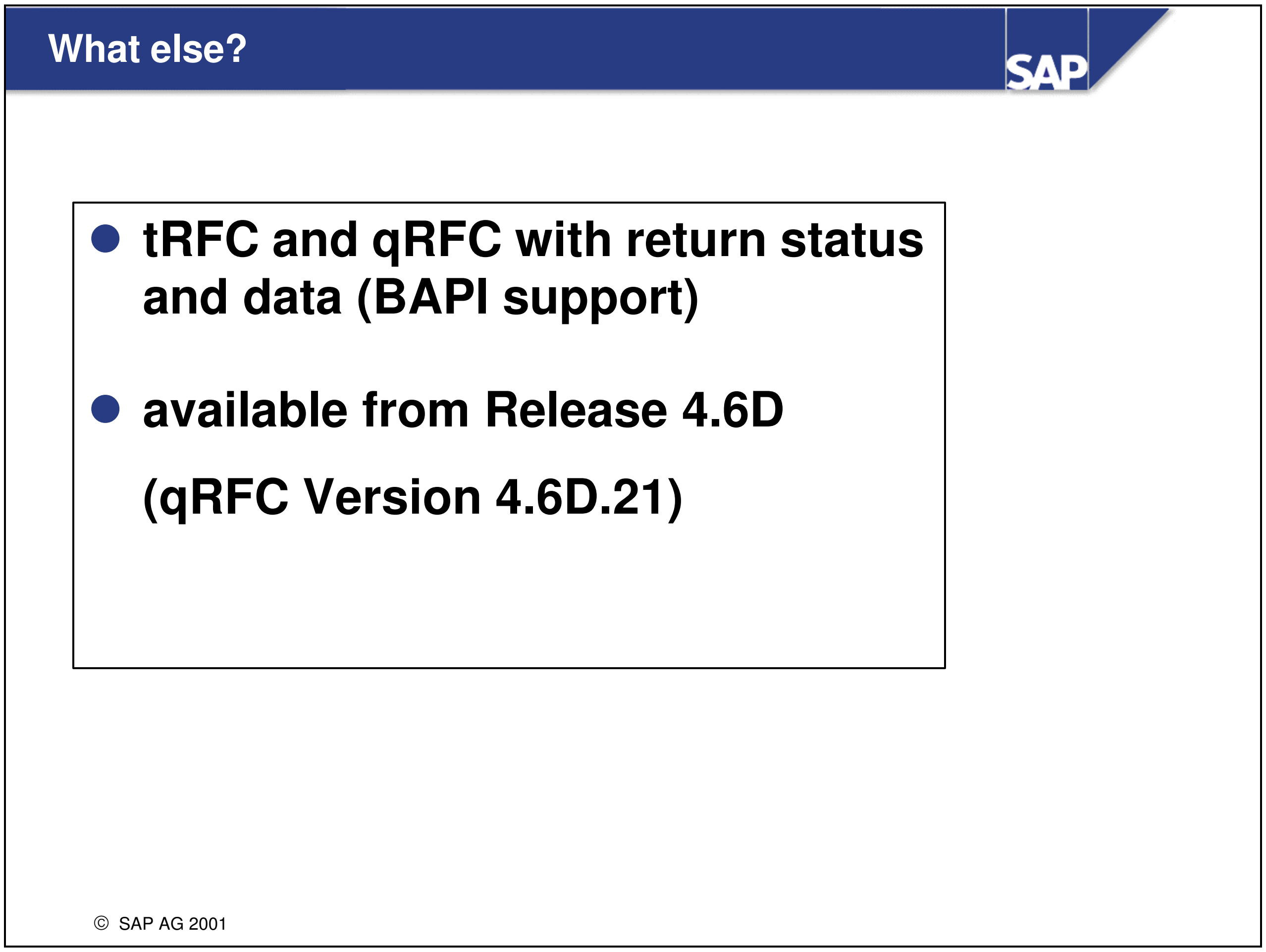
- qRFC version 4.6D.010 will be delivered with 4.6C.
- Newest qRFC version: 4.6D.021
- qRFC is also available for 3.1H, 3.1I, 4.0B, 4.5B, 4.6A & 4.6B via
  - Hot Packages
  - T Transports

Learn German for using the qRFC monitors

if you install qRFC via T Transports!!!!!

What else?

- CSS note 193515 includes a short description of qRFC.
- CSS note 166096 describes how to install qRFC on 3.xx with 3.1I kernel, 4.0x, 4.5x & 4.6x.



What else?

- tRFC and qRFC with return status and data (BAPI support)
- available from Release 4.6D

  (qRFC Version 4.6D.21)

Version 4.6D.011

- Important corrections
  - ♦ Better performance in reading and deleting messages from Send Queue
  - ♦ Different SQL-errors possible because SQL-statements weren't written in capital letters.
  - ♦ Short dump 'CONVT_NO_NUMBER' in RSTRFCM1 and RSTRFCM3
- Different changes in QIN-Scheduler for better handling of registered Receive Queues

Version 4.6D.012

- Important correction
  - ♦ Lost tRFC- and qRFC-LUWs when using qRFC with Receiver List

Version 4.6D.013

- Important correction
  - ♦ Loop in SAPLORFC or SAPLIRFC if more than 2 queues in status STOP.

Version 4.6D.014

- Important correction
  - ♦ Not all queues are returned by TRFC_QIN_OVERVIEW.

Version 4.6D.021

- New Functionality
  - ♦ tRFC/qRFC with return status & data

Version 4.6D.023

- Important corrections
  - ♦ ASSIGN_LENGTH_0 in SAPLQIWK, SAPLORFC oder SAPLIRFC
  - ♦ Some entries with status EXECUTED remain in ARFCRSTATE.
- Multiple Conditional RESTART of a queue is now available.
- Show hanging queues from within SMQ1 and SMQ2
- Reset queue status for automatic restart via Scheduler with SMQ2 available
- Restart of SYSLOAD-LUWs from RSARFCEX available

Version 4.6D.025

- Important corrections
  - ♦ Blocking send queue (status NOSENDS) when calling in update task
- Better performance in SMQ1 and SMQ2
- Multiple tRFC/qRFC-Restart of LUWs after SYSLOAD in certain group of appservers via profile parameter **abap/qrfc_asgroup_for_sysload**.
- Multiple activate of Send Queues in certain group of appservers via profile parameter **abap/qrfc_asgroup_for_activate**.
- API for check of last function call in a LUW
- Syslog entries when deleting tRFC/qRFC-LUWs via SM58, SMQ1 or SMQ2