

BC417

BAPI Development for Accessing SAP Components

SAP NetWeaver

Date _____
Training Center _____
Instructors _____

Education Website _____

Participant Handbook

Course Version: 62 Revision A
Course Duration: 3 Day(s)
Material Number: 50094490



An SAP course - use it to learn, reference it for work

Copyright

Copyright © 2009 SAP AG. All rights reserved.

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

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Trademarks

- Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.
- IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.
- ORACLE® is a registered trademark of ORACLE Corporation.
- INFORMIX®-OnLine for SAP and INFORMIX® Dynamic ServerTM are registered trademarks of Informix Software Incorporated.
- UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.
- Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.
- HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
- JAVA® is a registered trademark of Sun Microsystems, Inc.
- JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
- SAP, SAP Logo, R/2, RIVA, R/3, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Disclaimer

THESE MATERIALS ARE PROVIDED BY SAP ON AN "AS IS" BASIS, AND SAP EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR APPLIED, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH RESPECT TO THESE MATERIALS AND THE SERVICE, INFORMATION, TEXT, GRAPHICS, LINKS, OR ANY OTHER MATERIALS AND PRODUCTS CONTAINED HEREIN. IN NO EVENT SHALL SAP BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR PUNITIVE DAMAGES OF ANY KIND WHATSOEVER, INCLUDING WITHOUT LIMITATION LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS OR INCLUDED SOFTWARE COMPONENTS.

About This Handbook

This handbook is intended to complement the instructor-led presentation of this course, and serve as a source of reference. It is not suitable for self-study.

Typographic Conventions

American English is the standard used in this handbook. The following typographic conventions are also used.

Type Style	Description
<i>Example text</i>	Words or characters that appear on the screen. These include field names, screen titles, pushbuttons as well as menu names, paths, and options. Also used for cross-references to other documentation both internal (in this documentation) and external (in other locations, such as SAPNet).
Example text	Emphasized words or phrases in body text, titles of graphics, and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example SELECT and INCLUDE.
Example text	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, and passages of the source text of a program.
Example text	Exact user entry. These are words and characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.

Icons in Body Text

The following icons are used in this handbook.

Icon	Meaning
	For more information, tips, or background
	Note or further explanation of previous point
	Exception or caution
	Procedures
	Indicates that the item is displayed in the instructor's presentation.

Contents

Course Overview	vii
Course Goals	vii
Course Objectives	vii
Unit 1: BAPI Fundamentals.....	1
BAPI Concepts and Usage	2
BAPI Transaction Model	15
Unit 2: Finding and using SAP supplied BAPIs.....	25
Finding an SAP developed BAPI.....	26
Calling SAP BAPIs from ABAP.	35
Unit 3: Creating customer specific BAPIs	49
The Business Scenario.....	51
Tools and Naming Conventions	58
Creating the Business Scenario	75
BAPI development guidelines	87
Testing an RFC enabled Function Module.....	120
Business Object Repository	127
Business Objects	135
Business Object Components.....	147
Logical Unit of Work	163
Authorizations	175
Locking	182
Update Techniques	199
Unit 4: BAPIs and RFCS	241
External ABAP Call	242
Web Services.....	259
Unit 5: Enhancement of SAP supplied BAPIs.....	283
Planning a BADI in a BAPI	284
Implementing a BADI of a BAPI.....	296
Redefining a method in the BOR	307
Unit 6: BAPIs Using ALE & IDOCs	317
ALE and IDOCs ALE Basic Concepts.....	318

Unit 7: Mass Data Transfer	337
Mass Data Transfer for Special BAPIs	338
Appendix 1: Implementation Project for New BAPIs	355
Index	367

Course Overview

After you complete this course, you should have a good understanding of the use of SAP BAPIs. You'll also understand how to develop your own BAPIs.

Target Audience

This course is intended for the following audiences:

- Project team members responsible for development

Course Prerequisites

Required Knowledge

- BC400
- ABAP Programming experience

Recommended Knowledge

- BC416, BC425, BC430, BC414



Course Goals

This course will prepare you to:

- Use SAP supplied BAPIs, as well as design your own customer specific BAPIs.



Course Objectives

After completing this course, you will be able to:

- Find and use BAPIs developed by SAP
- Design and maintain your own BAPIs
- Locate or create Business Objects
- Enhance SAP Delivered BAPIs
- Perform external calls to BAPIs

Unit 1

BAPI Fundamentals

Unit Overview

This unit describes the basics of the business framework, business objects, the BOR, transaction concepts and logical units of work, and how BAPIs fit into these concepts.



Unit Objectives

After completing this unit, you will be able to:

- Describe the SAP Business Framework
- Define Business Objects and list their components
- Describe the purpose and basic functionality of a BAPI
- Describe the BAPI Transaction Model
- List transaction steps and their timing

Unit Contents

Lesson: BAPI Concepts and Usage	2
Lesson: BAPI Transaction Model	15

Lesson: BAPI Concepts and Usage

Lesson Overview

SAP created the Business Framework to allow the technical integration and exchange of business data among SAP components and between SAP and non-SAP components. A major component of the Business Framework are the Business Application Programming Interfaces (BAPIs), which represent visible interfaces at the component boundaries and whose properties serve to integrate these components.

The integration can include both components within a local network and components that are connected with one another through the Internet. BAPIs allow integration at the business level, not the technical level. This provides for greater stability in the link and independence from the underlying communication technology.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the SAP Business Framework
- Define Business Objects and list their components
- Describe the purpose and basic functionality of a BAPI

Business Example

In today's business environment, the need for up to date and accurate business information is more pressing than ever. It's not enough that the information is in the organization somewhere — it must be concise, precise and in an easily usable form that is available on a timely basis. Further, the users of this information resource may be scattered across the world and operating on different systems from the one which holds the data. Accessibility, then, becomes a major business issue.

Business Framework

The SAP Business Framework provides a structure for functionality based on application components (business components) and object models. The Business Framework enables customers and partners to link their own components to the SAP System. The use of an object-oriented view and object models reduces the complexity of the overall system.

Business Framework architecture provides the basis for developing SAP Business Components. The integration and communication architecture can be shown as follows:

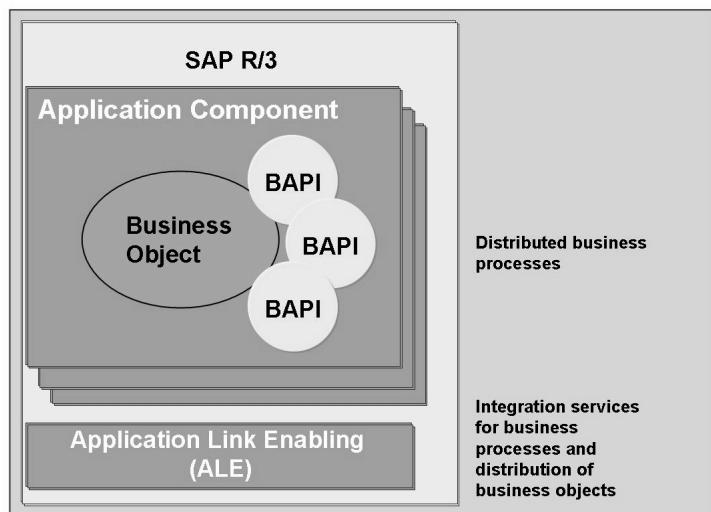


Figure 1: Integration Components

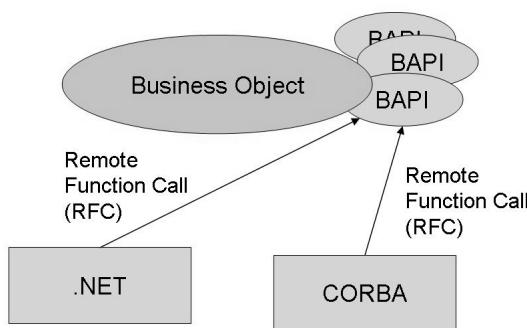


Figure 2: Communication Components

The basic components of SAP Business Framework are:

Business Components

SAP Business Components provide autonomous business functions and consist of business objects. For example, the business objects Employee and Applicant are assigned to the Business Component Human Resources. Business processes are either implemented within a Business Component or across several Components (distributed business processes).

Business Objects

The object-oriented structure of the SAP System is based on Business Objects. They encapsulate business data and functionality and define the functional scope and boundaries of a Business Component.

Business Application Programming Interfaces (BAPI)

BAPIs are interfaces for Business Objects. Together with the Business Objects, BAPIs define and document the interface standard at the business level.

Integration Service, Application Link Enabling (ALE)

The ALE Integration Service enables the integration of business processes that are carried out in different SAP system and non-SAP systems. This service is based on the system-wide distribution of Business Objects using the ALE distribution model.

Communication Services

These are the communication technologies, for example, Distributed Component Object Model (DCOM) and Remote Function Call (RFC) that use the Business Framework to access BAPIs.

SAP Business Objects

Business object technology and business object programming are based on the concept of business objects. Real world objects, for example, an employee or a sales order, are modeled as business objects in business application systems, such as the SAP System. The starting point for this modeling process is the definition of a business object type.

A business object type, which represents of a business entity in the SAP System, encompasses both the functionality (in the form of methods) and the data (in the form of attributes) of this entity. The implementation details of the business object type are hidden from the user. The business object type is accessed through defined functions (methods). This is referred to as encapsulation.

Business object types are used to break the SAP System down into smaller, specific units. As a result, the system's structure is improved while its complexity decreases.

Business object types form the point of entry to the data and the functionality of the SAP System. At the business object type level, both non-SAP systems and the various SAP business components can communicate with each other.

The business object type **SalesOrder** represents a customer's request to the company to supply a particular quantity of material at a certain point in time or to perform services at a certain point in time. A specific sales order is identified by a sales document number. The business object type contains all the necessary information for a sales order: sold-to party, sales organization, document date, net value of the order, and currency of the sales and distribution document.

A business object type should be understood as a business concept and its implementation of the concept in the SAP System. The term business object type corresponds to the term class in object-oriented programming languages.

A specific occurrence of a business object type, for example, a sales order actually stored in the system, must be distinguished from the business object type itself. This occurrence is called an **instance** or a **business object** (the usual term in object-oriented programming languages).

The sales order #102 is an instance of the business object type SalesOrder. This sales order is identified by its sales and distribution document number 102. The object is described by its attributes, including Sold-to party, Sales organization, Document date, and so on.

Structure of a Business Object Type

The graphic illustrates the concept of a business object type:

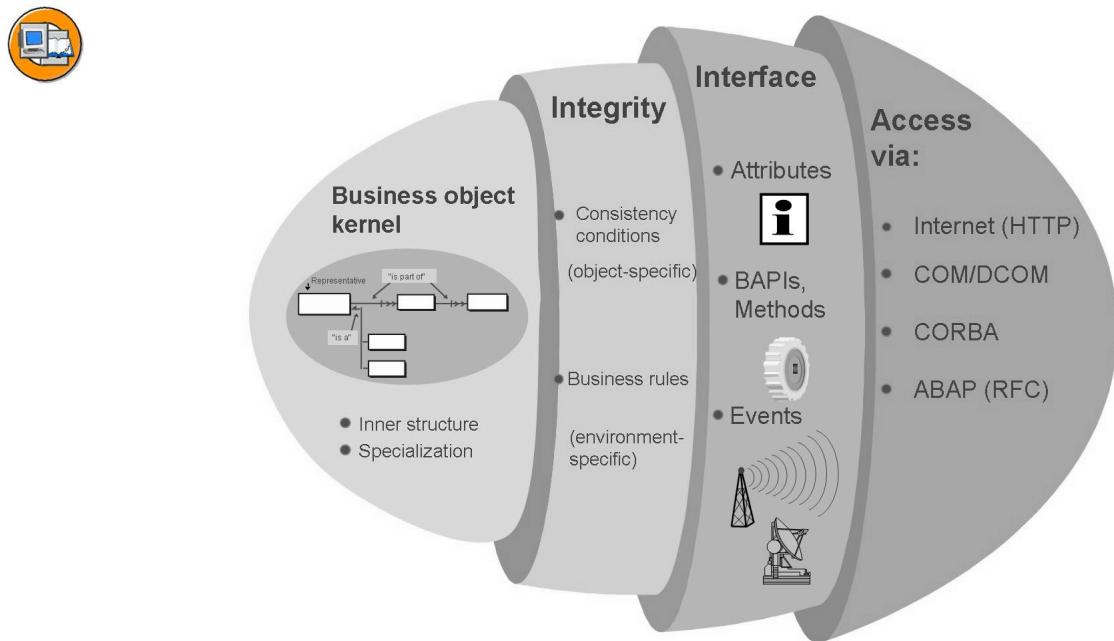


Figure 3: Business Object Type

Important considerations in the previous graphic include:

- The kernel of a business object type describes the inner object structure, in other words its data and the structure of this data.
- A constraint is a rule that ensures the consistency of an object (for example, different pricing conditions are assigned as of a stipulated minimum sales quantity).
- Business rules require that an object must comply with the rules to ensure the object remains consistently embedded in its environment (for example, a sales organization can only sell to a customer for whom business relationship information exists).
- The interface provides the pre-defined means of accessing business object types. These can be public attributes, methods/BAPIs, and events.
- An object can be accessed via COM/DCOM or CORBA (Common Object Request Broker Architecture). Within ABAP applications, a business object can be accessed by Remote Function Call (RFC).

Defining and Implementing an Object Type

Business objects (object types) are defined and described in the Business Object Repository (BOR) by the following characteristics:

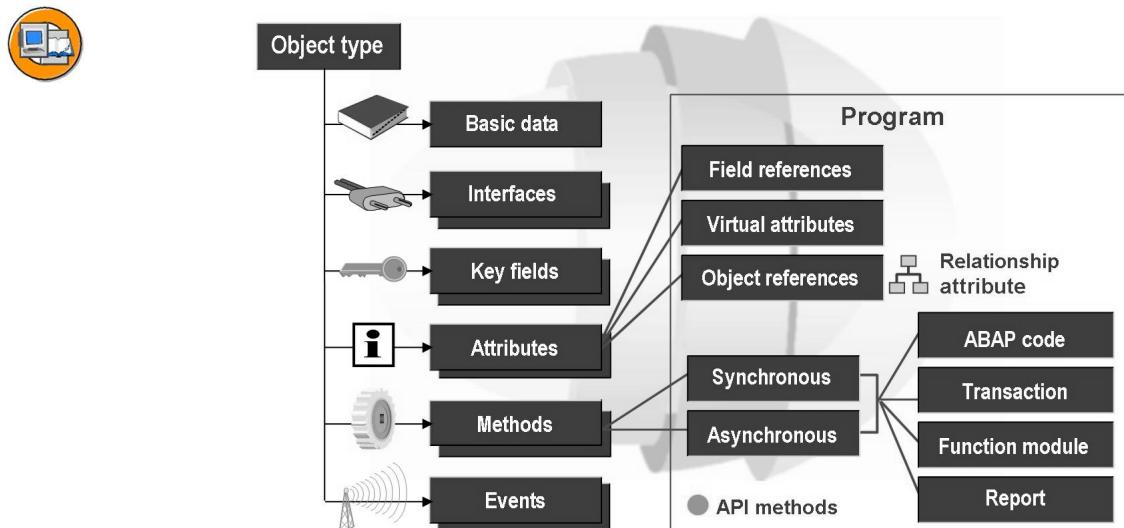


Figure 4: Components of an Object Type

Basic data

comprises the mnemonic object name, the technical name of the object type, its name (meaningful description), its short description, release status, default method, default attribute, and the name of the program containing the implementation.

Interfaces

provide a pre-defined interface (attributes, methods, events) for an object. Interface types generally do not contain an implementation.

Key fields

uniquely identify an object.

Attributes

represent object properties and characteristics.

Methods

encapsulate functionality either by synchronous or asynchronous methods.

Events

indicate the occurrence of a status change of a business object.

Business Object Key Fields

Each business object type has one or more key fields. Each instance has a unique combination of values in the key fields and is identified by the specific values in the key fields. This unique combination of values makes it possible to access a specific instance of a business object type via the values in the key fields. The key fields are an important design consideration for a business object.

In most cases, the key fields of an object type are also the key fields in the table containing the header data of the business object type. Because BOR runtime objects are client-independent, the client should not be used as a key field.

Business Application Programming Interfaces

Overview

Business Application Programming Interfaces (BAPIs) enable access to SAP functions across formal, stable, and dialog-free interfaces. These interfaces can be used by external applications developed by customers and complementary software partners as well as by other SAP applications.



- **Business Application Programming Interface**



- **A BAPI is a precisely-defined interface for processes and data of a business application system and is implemented as a method of an object in the Business Object Repository (BOR).**

Figure 5: What is a BAPI?

BAPIs are defined as API methods of SAP Business Object Types. These object types are used within the Business Framework to enable object-based communication between components. Business objects and their BAPIs enable object orientation to be used in central information processing in companies. For example, existing functions and data can be reused, trouble-free technical inter-operability can be achieved, and non-SAP components can be implemented.

Since BAPIs are used as standard interfaces into the SAP System, They require certain standard, known, and dependable characteristics. These characteristics must be taken into consideration whenever a BAPI is developed. Those characteristics are:

Object-oriented

BAPIs are implemented as object methods of Business Objects defined in the Business Object Repository.

Stable interface

The interface of a BAPI is frozen.

Internal and external use

BAPIs can be used from within SAP system and can also be called from external systems.

No presentation layer

BAPIs do not provide user interface functions. It is the responsibility of the caller to display the results of the request.

Parameter usage for messages

Success and error messages are handled with a RETURN parameter. This may be in a structure or a table, depending on the BAPI.

Applications can use BAPIs to directly access the application layer of the SAP System and, as clients, applications can use the business logic of the SAP System. BAPIs provide the client with an object-oriented view of the application objects, without needing to know the implementation details.

Integration

BAPIs can be used for these types of integration:

- Connecting SAP Systems to the Internet using the SAP Exchange Infrastructure.
- BAPIs also allow the creation of true component software, as they enable standardized communication between SAP components. The ultimate objective is to encapsulate the functionality of the SAP system in independent business components that are integrated through a common interface (the BAPIs).
- The integration of new SAP components (such as Advanced Planner and Optimizer (APO) and Business Information Warehouse (BW)), non-SAP software, and legacy systems.
- Implementation of distributed scenarios with asynchronous connections using Application Link Enabling (ALE).
- Using PC programs as frontends to the SAP System – These can be developed with Visual Basic (Microsoft), for example, or with Visual Age for Java (IBM)
- Workflow applications that extend beyond system boundaries.
- Customers' and partners' own developments.

The following graphic shows how BAPI interfaces enable different types of applications to be linked with the SAP system.

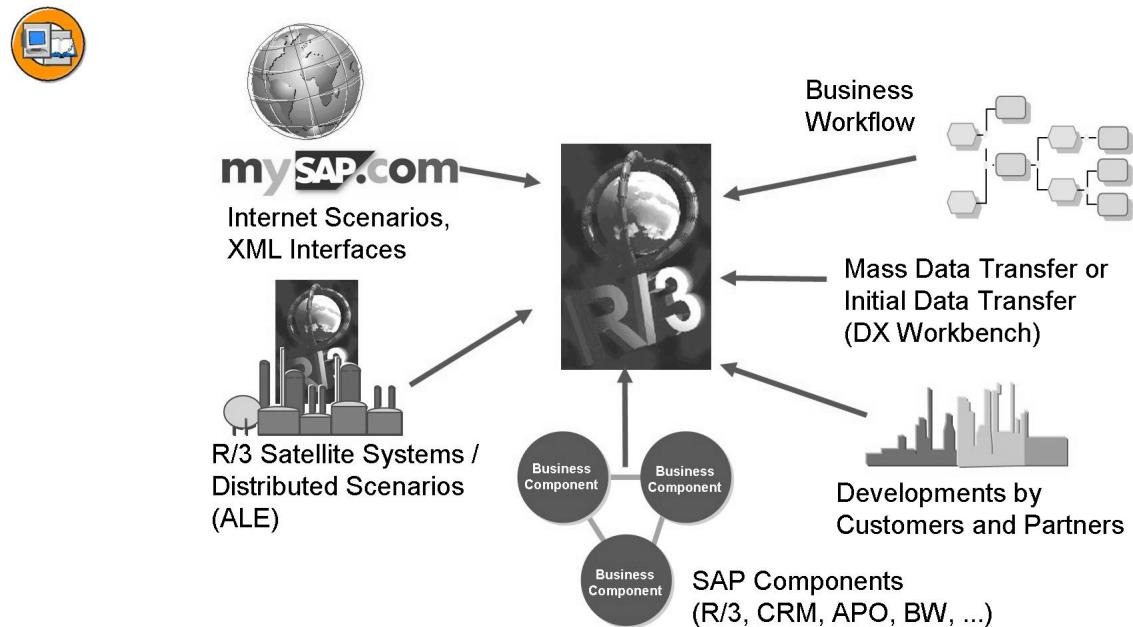


Figure 6: What is the Purpose of BAPIs?

Objectives for Integration and Usability

When implementing BAPIs, you should pursue the goal of avoiding the disadvantages of the postcard effect associated with conventional interfaces: No separation between contents and transport. In the case of a postcard, the text is written on a card that also serves as the information medium. If the information medium changes, then the receiver has to read the text in a different way. As a result, changes to the interface affect the type of access.

BAPIs, in contrast, clearly separate the business contents from the underlying communication technology. This procedure can be compared to a letter in an envelope: It is easy to put a letter in a new envelope, just like it is easy to apply new and/or different communication technologies like COM/DCOM, the CORBA standard, or new Internet standards. The envelope itself is irrelevant to understanding the contents of the letter. In this example, the BAPIs correspond to the letter, meaning they are independent of the programming language and communication mechanisms.

To allow their use in any scenario, BAPIs must support all forms of communication and be available to all types of participants:

- Support for synchronous and asynchronous communication

BAPIs can be used for both synchronous and asynchronous calls of an SAP component. If the call is synchronous, the BAPI is called as a remote function call (RFC). ALE inbound processing enables asynchronous communication with BAPIs. When the BAPI is called in the sender system, an IDoc is generated and dispatched to the receiver system. When the IDoc reaches the receiver, the parameters of the corresponding BAPI are automatically filled with the IDoc data, and the BAPI is called synchronously.

- Support of machine-to-machine and human-to-machine communication

BAPIs can be used both to integrate application systems and to link up alternative frontends.

1. The integration of application systems is characterized by machine-to-machine communication. It is implemented by exchanging large volumes of data and requires high system performance. As a result, the BAPIs used for this type of scenario have a lower level of detail.
 2. The integration of alternative frontends allows human-to-machine communication. BAPIs used for this purpose have a higher level of detail. They must be designed to allow more flexible communication that is free of errors.
- Communication support for components that are narrowly linked and coupled through the Internet

In addition to the use of BAPIs under communication technologies like CORBA and COM/DCOM, BAPIs can also be accessed via the Internet with the SAP Business Connector. The Business Connector generates an XML document from a BAPI call, or transforms an inbound XML document into a BAPI call. This makes it possible to dispatch BAPI calls as XML documents, enabling communication of components that are linked via the Internet.

Benefits

Using BAPIs results in the following benefits:

- BAPIs represent well-defined, internally consistent units that always represent business facts and leave a consistent database state behind.
- The business contents can be standardized because BAPIs not only allow the integration of the SAP system and other software components at a technical level, but also at the business level.
- BAPIs have become a communication standard between business systems. Access is possible through object-oriented interface technologies (such as COM/DCOM from Microsoft). The SAP business objects conform to the guidelines of the OAG (Object Application Group), and meet the CORBA standard from the OMG (Object Management Group).
- Stability and compatibility are assured once SAP has released a BAPI. Its interface definitions and parameters will remain stable in the long term, which means application programs will not be affected by changes to the underlying SAP software or data. If upward-compatible enhancements are made to the BAPIs, the stability of the existing applications is not affected.
- Open systems are supported in that BAPIs can be accessed from any widespread development platform.

Business Object Repository

Definition

The Business Object Repository (BOR) is the central access point for the SAP business object types and their BAPIs. The BOR was originally developed for SAP Business Workflow. Today, in addition to storing SAP business object types and their BAPIs, the BOR is used for ArchiveLink, output control, and other generic object services.

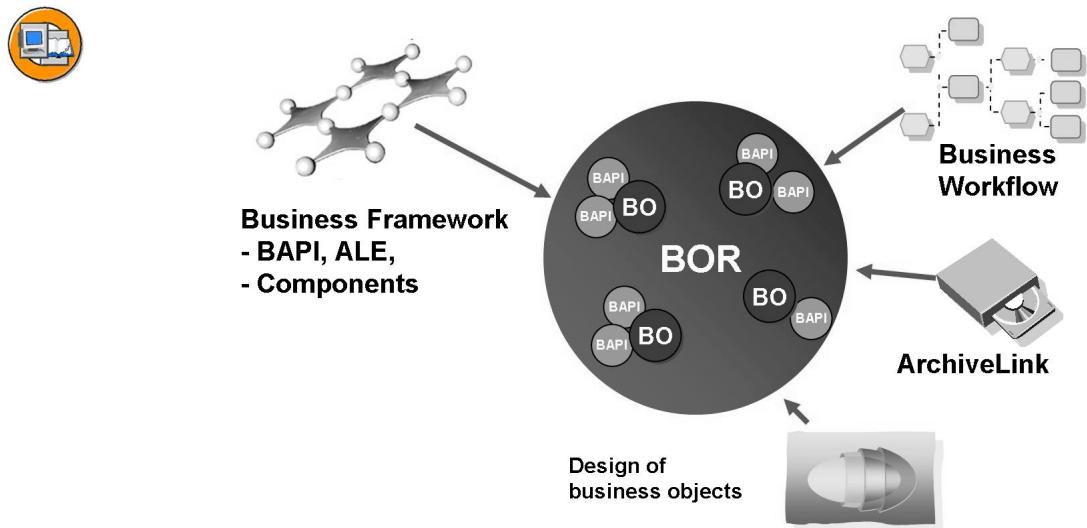


Figure 7: Business Object Repository

Use

The BOR provides the following services in the BAPI context:

- Allows an object-oriented view of all data and processes in the SAP system.
- Arranges the various business object types in accordance with the component hierarchy, enabling functions to be searched and retrieved quickly and easily.
- Stores all relevant information. The BOR contains all the relevant information on the SAP business object types, their key fields, and their BAPI methods that are needed to integrate the correct object type definitions and BAPI calls in an application program. This makes the integration of middleware (such as the DCOM Connector, ActiveX Controls, CORBA Gateway, and so on) possible.
- Ensures interface stability. Any interface changes that are carried out in the BOR are automatically checked for syntax compatibility against the associated development objects in the ABAP Dictionary.
- Manages BAPIs in release updates. BAPI interface enhancements made by adding parameters are recorded in the BOR. Previous interface versions can thus be reconstructed at any time. When a BAPI is created, the release version of the new BAPI is recorded in the BOR. The same applies when any interface parameter is created.
- Creates instances of SAP business objects. The runtime environment of the BOR receives requests to create runtime objects from client applications and creates the appropriate object instances.



Lesson Summary

You should now be able to:

- Describe the SAP Business Framework
- Define Business Objects and list their components
- Describe the purpose and basic functionality of a BAPI

Related Information

- For additional information on developing BAPIs, refer to the BAPI Programming Guide available on SAPNET.

Lesson: BAPI Transaction Model

Lesson Overview

Left to their own devices to create their own bit of functionality, a dozen programmers would probably create a dozen similar, but not exactly alike, methods of satisfying that need. This is fine if that functionality is stand alone and does not fit into an overall process. However, if that piece of functionality is to be part of a process and that function is to be used in multiple ways and at various times, then functionality must be structured in such a way that it performs in a clear, consistent, and dependable way.

The transaction model is a template used to guide how the components of a transaction fit together. The list of components, the order of those components, and what each of them do determines how well the business process is carried out. The transaction model in which BAPIs are used determines how you have to program BAPIs.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the BAPI Transaction Model
- List transaction steps and their timing

Business Example

If you do not properly understand the sequence of events for your transaction model, the resulting application may work incorrectly or not at all.

Transaction and Logical Unit of Work (LUW)

Within the context of the transaction model used to develop BAPIs, a transaction represents one processing step or one logical unit of work (LUW). An LUW is all the steps involved in a transaction including updating the database.

The **ACID** principle applies to transaction models, meaning that transactions are:

Atomic

When a transaction is called, database operations are either fully executed or not at all. Either all relevant data has to be changed in the database or none at all.

Consistent

If a transaction is called more than once, each call must have the same result. No data is imported that may indirectly affect the result.

Isolated

There must be no functional dependencies between two transactions; one transaction must not affect another transaction.

Durable

Changes cannot be reversed and transactions cannot be canceled.

Characteristics

Transactional Orientation

A BAPI must be implemented so that it is transactional. In other words, it complies with the ACID principle. The BAPI transaction model must also enable users to combine several BAPIs into one LUW. The BAPI transaction model, therefore, implies both that individual BAPIs must be transactional and that several BAPIs combined into one LUW must comply with the ACID principle.

Transaction Control in Client

The BAPI transaction model must afford the user explicit transaction control. Therefore, if several BAPIs are called together, the caller can decide when to execute a COMMIT WORK (or, a ROLLBACK WORK). This means that BAPIs themselves cannot (generally) execute a COMMIT WORK command.

The following restrictions apply to combining several BAPIs into one LUW:

- If an instance was created, modified or deleted by a write BAPI, a read BAPI can only access the most recent data if a COMMIT WORK has taken place.
- It is not possible to make two write accesses on the same instance within one LUW. For example, you cannot first create and then change the object within the same LUW.

You can, however, create several instances of the same object type within an LUW.

Transaction Handling via Service BAPIs

A transaction is completed either using a COMMIT WORK command or a ROLLBACK command. A BAPI transaction must be ended by calling the BAPIs `BapiService.TransactionCommit()` or `BapiService.TransactionRollback()`.

 **Note:** The transaction-controlling BAPIs `BAPIService.Transaction-Commit` and `BAPIService.TransactionRollback` are only available as of Release 4.5. In Release 4.0, use the function modules `BAPI_TRANSACTION_COMMIT` and `BAPI_TRANSACTION_ROLLBACK` in their place. The result of these function modules is identical to calling the BAPIs.

Operations that change the database must be carried out through the updating process. Otherwise, there's a risk that both unchecked and unwanted database COMMITs are executed during the RFC call.

Additionally, the call of a BAPI must not trigger further LUWs that are independent of the BAPI. For this reason BAPIs must not contain the following commands:

- CALL TRANSACTION
- SUBMIT REPORT
- SUBMIT REPORT AND RETURN

In Release 3.1, the BAPIs executed the COMMIT WORK command, and BAPIs had the same purpose as an LUW or transaction. Besides the BAPIs from Release 3.1, there are few further exceptions which, for technical reasons, contain a COMMIT WORK command.

 **Note:** If a BAPI executes a COMMIT WORK command, this must be mentioned in the BAPI documentation. This is the only way users are able to know that the BAPI contains a COMMIT WORK command. SAP supplied BAPIs must also be documented in the SAPNet - SAP System Frontend in Note 0131838, “Collective Note for BAPIs with the Commit Work Command”.

BAPI Transaction Model Without Commit

The example below of an external program calling a BAPI to change data in an SAP System, illustrates how the transaction model affects BAPI development. For example, this could involve a transaction implemented with Visual Basic. Only data from the SAP System is to be changed.

The RFC connection is live the whole time the external program is logged on to the SAP System to avoid having to repeatedly connect and disconnect X. When the RFC connection is already established, an RFC call does not essentially take up any more CPU time than a direct call to the function module from within the SAP System.

The process flow of the program consists of the following steps (see graphic below):



```

Log on
(Source code)
Call BAPI
(Source code)
Call BAPI
(Source code)
Call BAPI
BapiService.TransactionCommit()
(Source code)
Call BAPI
(Source code)
Call BAPI
(Source code)
Call BAPI
BapiService.TransactionCommit()
(Source code)
Log off

```

Figure 8: Program Flow - Current Model

The model can be graphically represented as follows:

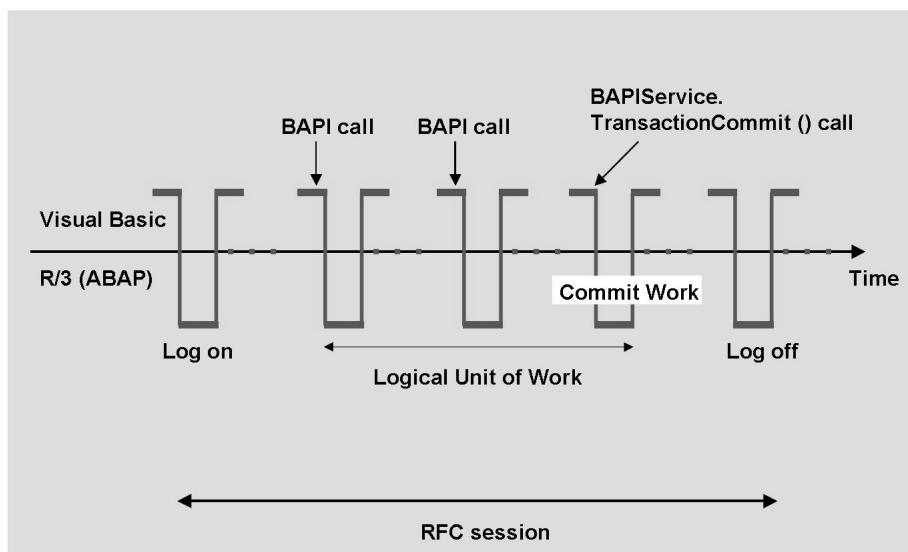


Figure 9: Transaction Model Without Commit (State of the art)

BAPI Transaction Model With Commit (Old)

There is one BAPI call for each transaction in the old transaction model (valid for Release 3.1). BAPIs can only be called synchronously. A BAPI call is essentially the call of the underlying RFC capable function module. The process flow of the program consists of the following steps:

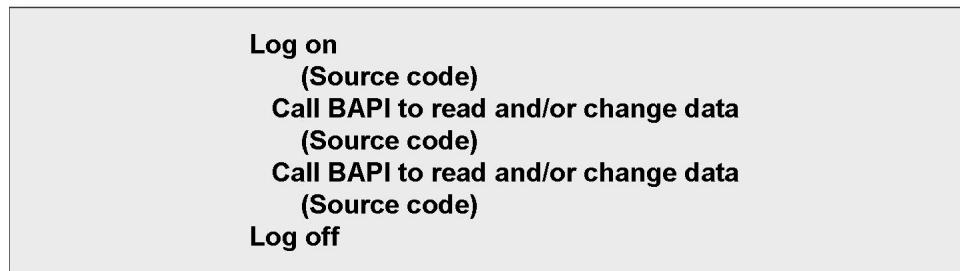


Figure 10: Program Flow - Old Model

The old model can be graphically represented as follows:

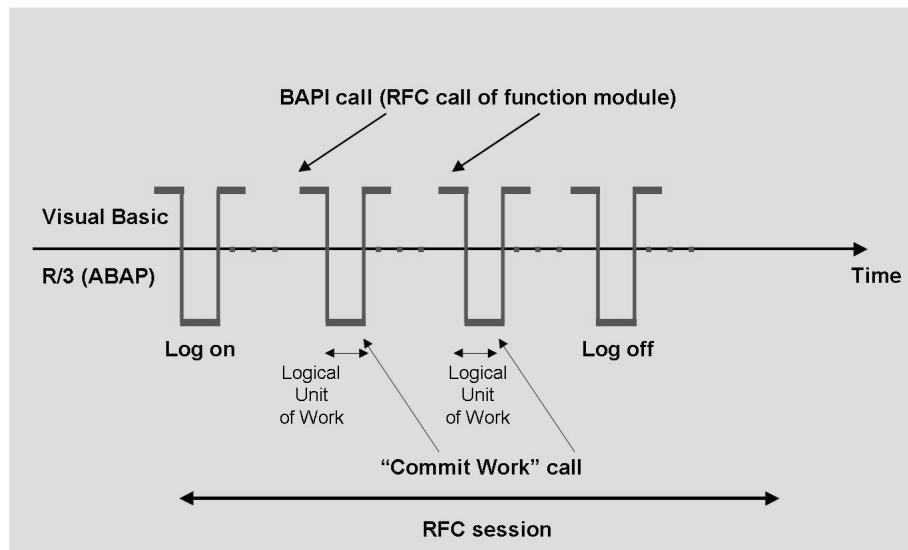


Figure 11: Transaction Model with Commit (obsolete)

The (write) BAPIs developed in Release 3.1 along with a few further exceptions execute a COMMIT WORK command themselves.



Caution: If a BAPI executes a COMMIT WORK command, this must be mentioned in the BAPI documentation. This is the only way users are able to know that the BAPI contains a COMMIT WORK command. These BAPIs must also be documented in the SAPNet - SAP System Frontend in Note 0131838, “Collective Note for BAPIs with the Commit Work Command”.



Lesson Summary

You should now be able to:

- Describe the BAPI Transaction Model
- List transaction steps and their timing

Related Information

- For additional information on developing BAPIs as well as the Transaction Model, refer to the BAPI Programming Guide Reference available on SAPNET.



Unit Summary

You should now be able to:

- Describe the SAP Business Framework
- Define Business Objects and list their components
- Describe the purpose and basic functionality of a BAPI
- Describe the BAPI Transaction Model
- List transaction steps and their timing



Test Your Knowledge

1. Business Objects encapsulate business data and functionality.

Determine whether this statement is true or false.

- True
- False

2. Six of the components for an object type are: _____

_____ and _____.

Fill in the blanks to complete the sentence.

3. What kind of component of a business object are BAPIs ?

Choose the correct answer(s).

- A Interface
- B Definition
- C Event
- D Method
- E Program

4. BAPIs can be used entirely within an SAP System.

Determine whether this statement is true or false.

- True
- False

5. In describing the ACID Principle, the letters stand for what characteristics?

_____, _____, _____, and _____.

Fill in the blanks to complete the sentence.

6. Several BAPIs can participate in a single unit of work.

Determine whether this statement is true or false.

- True
- False



Answers

- Business Objects encapsulate business data and functionality.

Answer: True

Business object definitions include all required data for the object as well as all functions that can be performed against that data.

- Six of the components for an object type are: Basic Data, Interfaces, Key fields, Attributes, Methods, and Events.

Answer: Basic Data,, Interfaces,, Key fields,, Attributes,, Methods,, Events
Each of these object components have their own component makeup as well.

- What kind of component of a business object are BAPIs ?

Answer: D

BAPIs are methods of the business object. This is where the actual functionality is defined.

- BAPIs can be used entirely within an SAP System.

Answer: True

Since BAPIs are implemented as function modules, they can be called from within an SAP System when any other function module is called. That call, however, must reflect the special nature of the BAPI structure and restrictions.

- In describing the ACID Principle, the letters stand for what characteristics? Atomic, Consistent, Isolated, and Durable.

Answer: Atomic, Consistent, Isolated, Durable

The result of a BAPI execution must be dependable, predictable and independent of any other activity.

- Several BAPIs can participate in a single unit of work.

Answer: True

The BAPI Transaction Model does allow combining of several BAPIs into one LUW, but the combination must comply with the ACID Principle.

Internal Use SAP Partner Only

International Use SAP Partner Only

Unit 2

Finding and using SAP supplied BAPIs

Unit Overview

SAP introduced BAPIs in version 4.x of R/3. The purpose of BAPIs was to allow customers a simple access path to SAP business processes and data. SAP decided at that point to implement BAPIs as RFCs (function modules). Unlike ordinary function modules, BAPIs have well-structured component rules that were followed by SAP developers. They also have special considerations regarding such items as error handling. There are also restrictions that are unique only to BAPIs. This Unit reviews the basics required by customers in order to find and use SAP BAPIs from the ABAP programming language.



Unit Objectives

After completing this unit, you will be able to:

- Use the BAPI transaction code.
- Find an SAP Business Object.
- Find the documentation related to SAP BAPIs.
- Understand the basics behind calling BAPIs from ABAP applications.
- Call read/write style BAPIs from ABAP applications.
- Understand the special error handling mechanism used by BAPIs

Unit Contents

Lesson: Finding an SAP developed BAPI	26
Exercise 1: Finding SAP Business Objects and BAPIS	31
Lesson: Calling SAP BAPIs from ABAP.....	35
Exercise 2: Calling SAP BAPIs from ABAP	43

Lesson: Finding an SAP developed BAPI.

Lesson Overview

This lesson will give you the steps required in order to find a BAPI, as developed by SAP.



Lesson Objectives

After completing this lesson, you will be able to:

- Use the BAPI transaction code.
- Find an SAP Business Object.
- Find the documentation related to SAP BAPIs.

Business Example

You are to write some custom applications that need access to data stored in ECC. Instead of trying to figure out what tables SAP maintains its data in, you'd like to use some SAP supplied APIs.

Overview

All BAPIs are stored in the Business Object Repository, commonly called the BOR. To access the BOR, use transaction code BAPI.

The BOR is organized in application areas. By expanding an application area (or subarea), you'll see the Business Objects that refer to that area. By expanding the business object, you'll see the key field attribut(es) of the Business Object as well as all the methods that allow you to maintain the Business Object. The example bellow points out the Business Object **CompanyCode**.

An alternative access to the BOR Business Objects, is via the “Alphabetical” tab strip of the BAPI transaction.

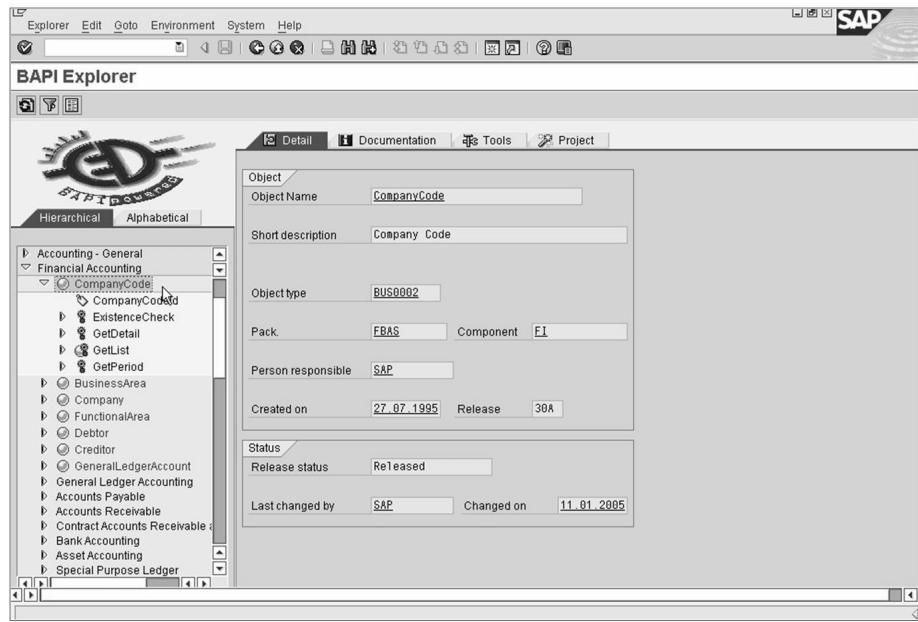


Figure 12: Finding an SAP BAPI (1)

BAPIs are implemented as methods of a business object. By selecting a method in the left side navigation area, you'll see the details of the method. Two important details required before using a BAPI are :

- The name of the ABAP function module that implements the BAPI
- The Release Status of the BAPI (generally marked as **Released or Obsolete**). Customers should only use Released BAPIs. Obsolete BAPIs are generally replaced by a more up to date BAPI implementation, hence another BAPI.

The “Documentation” tab strip will give you details of what business activity the BAPI was designed to accomplish as well as any technical details relating to the specific BAPI.

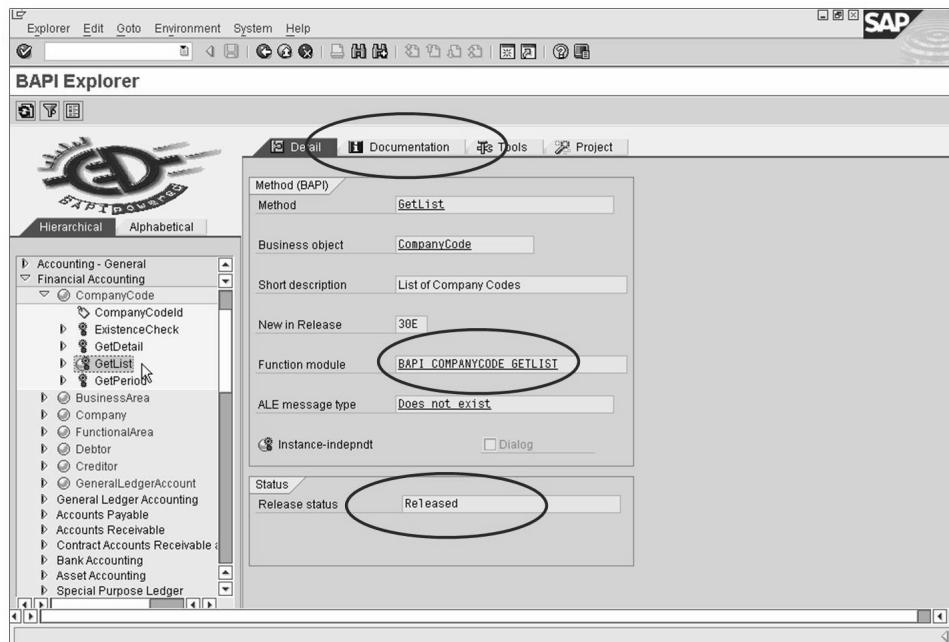


Figure 13: Finding an SAP BAPI (2)

Extra BAPI Explorer features

The BAPI Explorer is the SAP tool used to search for existing BAPIs.

Use

The BAPI Explorer is used internally in SAP to develop BAPIs, but can also be used by customers and partners to add their own BAPIs. The BAPI Explorer also provides access to all the tools used to create BAPIs and the required development objects in an integrated programming environment.

Structure

The BAPI Explorer is divided into **two** areas:

1. Hierarchy display

All the business object types or interface types for which BAPIs have been defined display here.

2. Work area

Here you can view the details and documentation of the development object selected in the hierarchy display. The tools used to develop BAPIs are also available in the work area.

Working with Display Functions

All the business object types or interface types for which BAPIs have been defined display in the hierarchy display of the BAPI Explorer.

Using the tabs Alphabetical and Hierarchical in the hierarchy display, you can select whether the business object types or interface types display alphabetically or as they are assigned in the component hierarchy in the BOR.

By expanding the relevant nodes you can navigate up to the parameter level of individual BAPIs.

Features

The following display functions are also provided, which enables you to directly access BAPIs and their details in the BOR.

Displaying details

Under the Detail view of the work area, all the technical details of the development object selected in the hierarchy display.

In most cases you can double-click on a field in the Detail display to get to the development environment or to display further information. For example, in the detail display for a particular method, by double-clicking on the name of the function module, it displays in the Function Builder.

Displaying and Printing Documentation

The documentation for the development object selected in the hierarchy displays in the Documentation view of the work area.

The data element documentation for each parameter field is contained in the documentation for structured BAPI parameters.

If you have installed Microsoft Internet Explorer Version 4.0 on the front-end computer, you can print the documentation using the standard SAPgui print function.

Changing the level of the hierarchy display

The component hierarchy may have many hierarchy levels. To make it easier to find a business object type and its BAPIs, you can use the function *Goto → Change* hierarchy level to limit the display to two hierarchy levels.

Specifying the BAPIs to be displayed

The default is to display only released BAPIs of business object types. By choosing *Goto → BAPIs* to display, you can display all the BAPIs contained in the BOR, that is BAPIs of SAP interface types and BAPIs that have not yet been released.

Business object types and interface types are identified by different symbols. To display these, choose *Goto → Display legend*.

Searching for BAPIs

Using the standard functions Find and Find next, you can search the displayed business object types or interface types using specified criteria (placeholders such as * can be used):

Object name, for example, BusinessProcess

Object type (technical object name), for example, SAP0001

Object description, for example, Plan*

Method name, for example, GetList

Method description, for example, Object list*

Working with Tools and Projects

The views Tools and Projects in the work area are mainly used for developing BAPIs.

Depending on the development object selected in the hierarchy display in the Tools view, the following tools and navigation options are provided:

- Direct access to the Business Object Builder, Function Builder, and ABAP Dictionary.
- List generator to create lists of BAPIs using specified selection criteria.

In the Project view, you can create projects to assist you with following and documenting the development procedures:

- Implementing new BAPIs
- Changing released BAPIs
- Requesting a new business object type

For each of these projects, there is a project form that takes you step-by-step through the entire development process and provides direct navigation options to the required development tools and information.

Within the project management you can save and delete your projects and you can edit the projects managed by other users by selecting Other users. If you have installed Microsoft Internet Explorer Version 4.0 on the front-end computer, you can print the project form using the standard SAPgui print function.

Exercise 1: Finding SAP Business Objects and BAPIS

Exercise Objectives

After completing this exercise, you will be able to:

- Using the BAPI explorer, browse within the Business Object Repository in order to find SAP BAPIs.

Business Example

You are to design custom applications that need to access SAP data and business processes. You'd like to use SAP standard APIs in order to simply interface to SAP.

Task:

Using the BAPI explorer, you will browse within the Business Object Repository in order to identify different SAP business components.

1. Within the application area 'Materials Management' 'Purchasing', what business objects presently exist on the current training ECC system.
2. For the Business Object *Bank*, what methods exist.
3. What is the name of the function module associated to the *GetDetail* method of the *USER* business object.
4. Find the documentation associated to the BAPI for creating Sales Orders.



Hint: The name of the Business Object for sales orders is *SalesOrder*. The name of the method is *CreateFromDat2*.

Solution 1: Finding SAP Business Objects and BAPIS

Task:

Using the BAPI explorer, you will browse within the Business Object Repository in order to identify different SAP business components.

1. Within the application area 'Materials Management' 'Purchasing', what business objects presently exist on the current training ECC system.
 - a) Start transaction code BAPI. Expand the application area 'Materials Management', then expand the area 'Purchasing' and you will find the following Business Objects :
 - PurchaseReqItem
 - PurchaseOrder
 - PurchSchedAgreement
 - PurchasingContract
 - PurchaseRequisition
 - PurchasingInfo
 - ProcurementOperation
 - SourceOfSupplyDeterm
2. For the Business Object *Bank*, what methods exist.
 - a) Start transaction code BAPI.
 - b) Go to tab strip *Alphabetical*
 - c) Page down list of Business Objects until you find the *Bank* Business Object.
 - d) Expand the *Bank* Business Object.
 - e) You should then find the following methods :
 - Change
 - Create
 - GetDetail
 - GetList
 - SaveReplica

Continued on next page

3. What is the name of the function module associated to the *GetDetail* method of the *USER* business object.
 - a) Start transaction code BAPI
 - b) Go to tab strip *Alphabetical*
 - c) Page down list of Business Objects until you find the *USER* Business Object.
 - d) Expand the *USER* Business Object.
 - e) Select the *GetDetail* method.
 - f) In the right hand section, next to the label 'Function Module', you will see the name of the function module **BAPI_USER_GETDETAIL**.
4. Find the documentation associated to the BAPI for creating Sales Orders.



Hint: The name of the Business Object for sales orders is *SalesOrder*. The name of the method is *CreateFromDat2*.

- a) Start transaction BAPI
- b) Go to tab strip *Alphabetical*
- c) Page down list of Business Objects until you find *SalesOrder*
- d) Expand the Business Object *SalesOrder*
- e) Select method *CreateFromDat2*
- f) Go to the *Documentation* tab strip.



Lesson Summary

You should now be able to:

- Use the BAPI transaction code.
- Find an SAP Business Object.
- Find the documentation related to SAP BAPIs.

Lesson: Calling SAP BAPIs from ABAP.

Lesson Overview

BAPIs are implemented as function modules. SAP designed these function modules with specific technical specifications that need to be understood before using a BAPI from one of your applications.



Lesson Objectives

After completing this lesson, you will be able to:

- Understand the basics behind calling BAPIs from ABAP applications.
- Call read/write style BAPIs from ABAP applications.
- Understand the special error handling mechanism used by BAPIs

Business Example

You identified BAPIs that you would like to use in various custom ABAP applications. You need to understand how to call these BAPIs from an ABAP program as well as any technical details in accomplishing this task.

Calling a Read BAPI.

Many Business Objects offer read services to their data. These methods are generally implemented as either a **GetList** method or a **GetDetail** method.

Once you have identified the BO you require, look for these 2 typical READ style BAPIs :

- GetList : instance independant method that typically returns an internal table of object keys and basic descriptive attributes.
- GetDetail : instance dependant method that typically returns details of 1 specific instance of a business object.

 **Note:** Some Business Objects may have more methods that implement read style functionality. Most, however, have at minimum the 2 described above.

Once you have found the method required, select the method in order to identify the name of the ABAP function module that implements it (see lesson on “Finding an SAP BAPI” for more details).

You should then learn about the functionality of the selected BAPI by looking at its corresponding documentation in the BOR and by testing the function module functionality within transaction SE37.

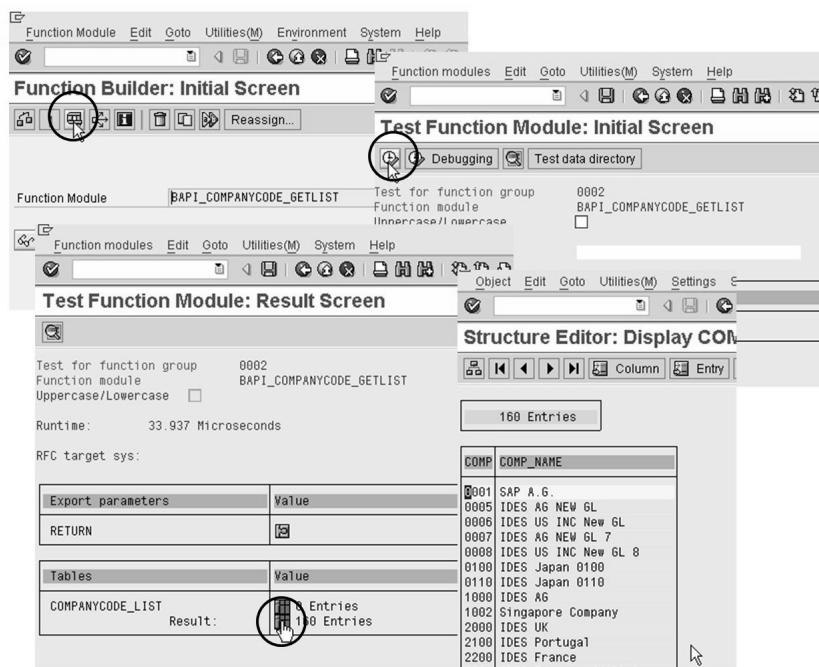


Figure 14: Testing a BAPI from SE37

Since BAPIs are implemented as function modules, calling a BAPI from ABAP is as simple as calling any function module. **The challenge will be the to understand the signature (parameter passing mechanism) specific to each BAPI.** The following code excerpt is an example call to the BAPI_COMPANYCODE_GETLIST BAPI.



```

Report      Z00DISPLAY_COMPANY_CODES Active
1 REPORT  Z00DISPLAY_COMPANY_CODES.
2
3 data it_companycodes type standard table of BAPI0002_1.
4 data wa_return      type BAPIRETURN.
5 field-symbols: <fs_companycodes> like line of it_companycodes.
6
7 start-of-selection.
8
9     CALL FUNCTION 'BAPI_COMPANYCODE_GETLIST'
10    IMPORTING
11        RETURN          = wa_return
12    TABLES
13        COMPANYCODE_LIST = it_companycodes.
14
15    if wa_return-type = 'S' or wa_return-type = ''.
16    loop at it_companycodes assigning <fs_companycodes>.
17        write: / <fs_companycodes>-comp_code, <fs_companycodes>-comp_name.
18    endloop.
19 else.
20     write: / 'Error detected calling BAPI_COMPANYCODE_GETLIST'.
21     write: / wa_return-message.
22 endif.

```

Figure 15: Sample call to a BAPI.

If you have called function modules from ABAP before, you have noticed our first particularity about BAPI function modules. **They do not use FM Exceptions.** BAPIs have their own specific mechanism for notifying the user that an error has

occurred. Each BAPI will have a parameter called **RETURN**. This parameter could be either a flat work area or an internal table. The user program calling a BAPI must test this parameter after the BAPI call in order to validate if the call to the BAPI was successful or not.

The RETURN parameter is based on a dictionary structure called **BAPIRET***. There are many of these structures available for use by the SAP developer.

The developer will initialize the first byte of this structure (typically a field called **TYPE**) with the severity level of the error. For successful calls, the **TYPE** field will either take on the value 'S' or SPACE (' '). Any other values received in the **TYPE** field will probably indicate that an error in the BAPI has occurred. The **MESSAGE** field of the RETURN structure, in this case, will have the descriptive text of the problem.

Calling a Create BAPI

Most Business Objects will offer a create service allowing you to create an instance of the business object via a BAPI.

The sample **PieceOfEquipment** BO we will be looking at seems to have 3 services that will create an instance of an equipment :

- **Create**
- **CreateByReference**
- **CreateFromData**

By looking at each method property and documentation you will notice the following :

- The **CreateFromData** method is **obsolete**
- The **CreateByReference** method requires an input equipment number which will act as a reference equipment during the creation of the new equipment.
- The **Create** method will create a new instance of the **PieceOfEquipment** BO.

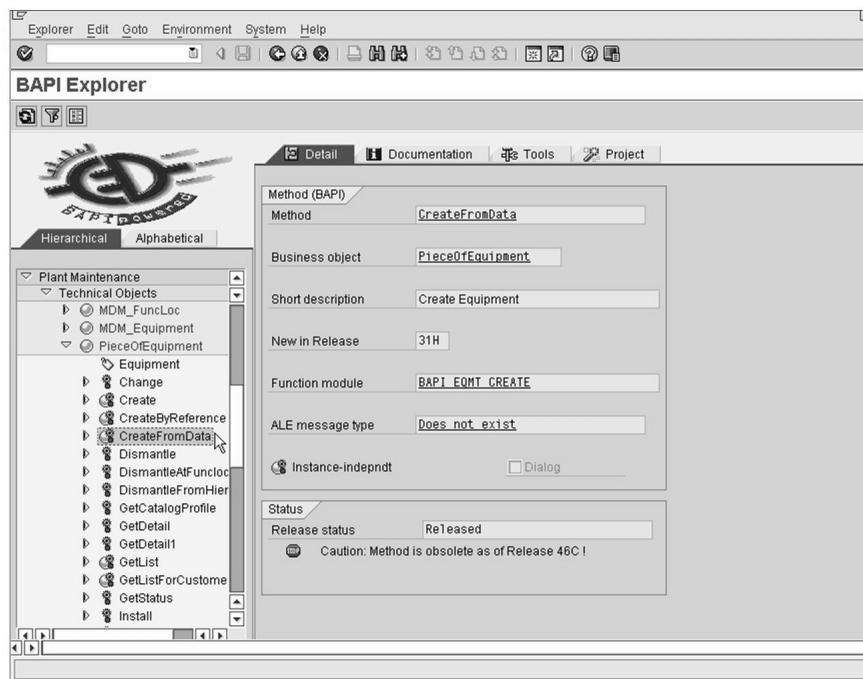


Figure 16: BOR representation of PieceOfEquipment.CreateFromData method.

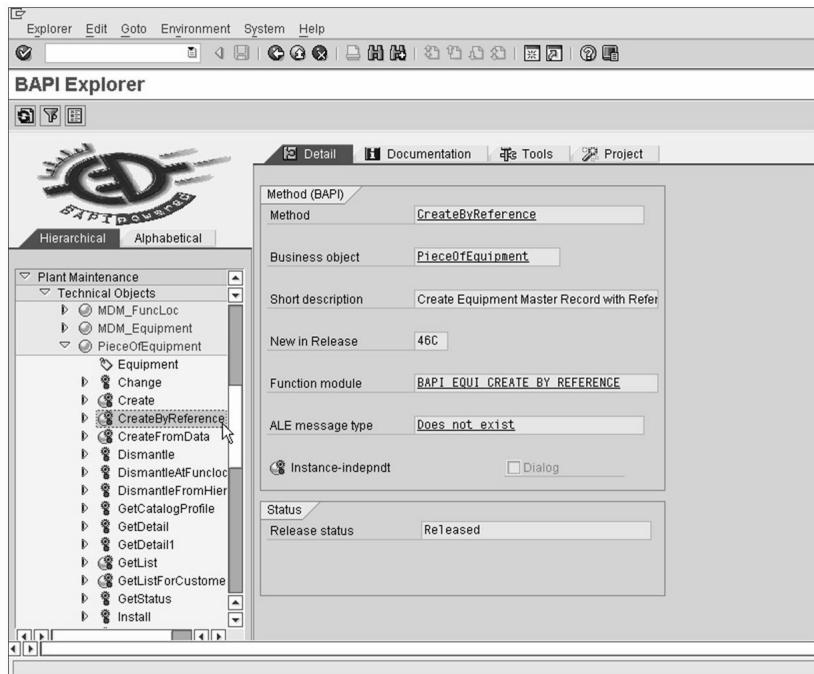


Figure 17: BOR representation of PieceOfEquipment.CreateByReference method.

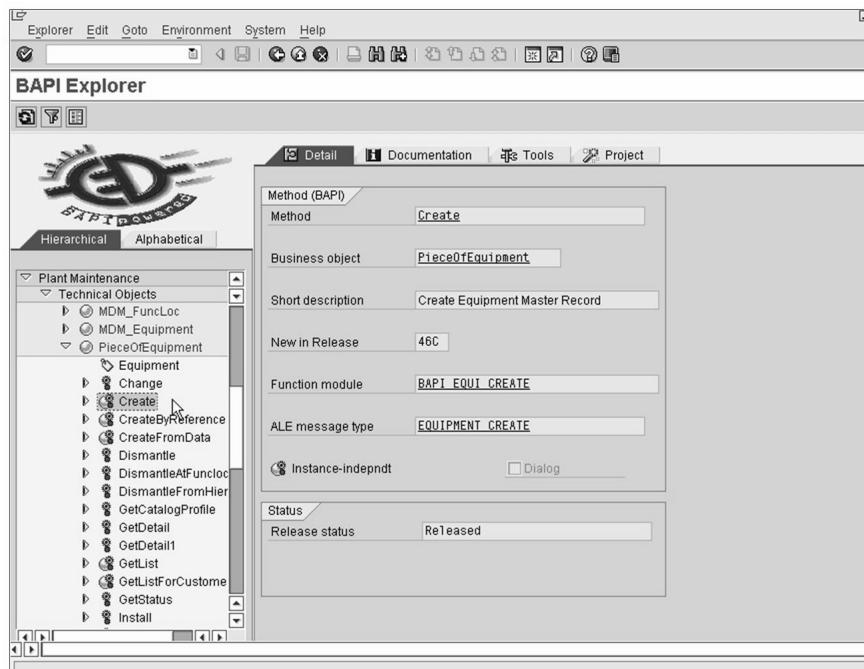


Figure 18: BOR representation of PieceOfEquipment.Create method.

In our business example, we will decide on using the Create method. By doing so, we identify the name of the ABAP function module that implements this process, **BAPI_EQUI_CREATE**.

The challenge in using SAP BAPIs is to understand the signature of the BAPI. This might require many hours of analysis and knowledge of the business process we are interfacing. For this specific example BAPI, the creation of a new equipment, we wish to call the SAP BAPI, giving it the following equipment properties :

- Equipment Description
- Equipment Category
- Equipment Weight
- Equipment Weight Unit Of Measure

By analyzing the BAPI signature, we notice that the BAPI requires as input 2 structures :

- DATA_GENERAL : based on dictionary structure BAPI_ITOB
- DATA_SPECIFIC : based on dictionary structure BAPI_ITOB_EQ_ONLY.

The following code excerpt is an example of filling the BAPI input structures, calling the BAPI passing it the data required, and testing the RETURN code.

Note also the COMMIT that is sent after the BAPI call since this BAPI does not commit the data changes.



```
X E F H H E E  
REPORT Z00DEMO_BAPI_EQUI_CREATE.  
  
data wa_data_general      type BAPI_ITOB.  
data wa_data_specific     type BAPI_ITOB_EQ_ONLY.  
data wa_equipment_number type BAPI_ITOB_PARMS-EQUIPMENT.  
data wa_return            type BAPIRET2.  
  
wa_data_general-obj_weight = '100'.  
wa_data_general-unit_of_wt = 'KG'.  
wa_data_general-descrip  = 'GR## my test equipment'.  
wa_data_specific-equipcatgry = 'M'.  
  
CALL FUNCTION 'BAPI_EQUI_CREATE'  
  EXPORTING  
    DATA_GENERAL          = wa_data_general  
    DATA_SPECIFIC         = wa_data_specific  
  IMPORTING  
    EQUIPMENT             = wa_equipment_number  
    RETURN                = wa_return.  
  
if wa_return-type = 'S' or wa_return-type = ''.  
  CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.  
  write: / 'Equipment number', wa_equipment_number, 'is created.'.  
else.  
  write: / 'Error creating equipment.'.  
  write: / wa_return-message.  
endif.
```

Figure 19: Sample call to BAPI_EQUI_CREATE

Calling a Change BAPI

The business object PieceOfEquipment offers a Change service allowing us to change the properties of an existing equipment. By accessing the BOR via the BAPI Explorer, we can find this service.

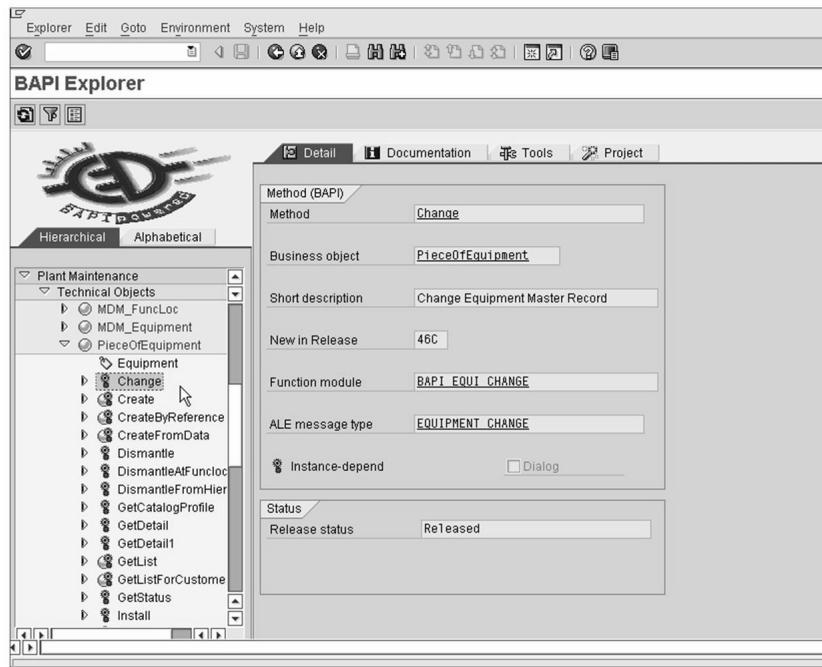


Figure 20: BOR representation of PieceOfEquipment.Change method.

Many Change style BAPIs use an 'X' flag structure in order to allow the user of the BAPI to specify what fields of the data structures has changed data. This is the case of BAPI_EQUI_CHANGE.

By analyzing the signature of this BAPI, we can identify that for each input data structure there is a corresponding 'X' structure. The 'X' structure is a series of 1 byte fields that you need to initialize to the constant 'X', for any data field that you want to have the BAPI change.

The following code excerpt is an example call to the BAPI_EQUI_CHANGE, where the user wants to change the weight of an existing equipment.



```
REPORT Z00DEMO_BAPI_EQUI_CHANGE.

data wa_data_general      type BAPI_ITOB.
data wa_data_generalx     type BAPI_ITOBX.
data wa_data_specific      type BAPI_ITOB_EQ_ONLY.
data wa_data_specificx    type BAPI_ITOB_EQ_ONLYX.
data wa_equipment_number   type BAPI_ITOB_PARMS-EQUIPMENT.
data wa_return             type BAPIRET2.

wa_equipment_number = '00000000010000958'.
wa_data_general-obj_weight = '999'.
wa_data_generalx-obj_weight = 'X'.

CALL FUNCTION 'BAPI_EQUI_CHANGE'
  EXPORTING
    EQUIPMENT          = wa_equipment_number
    DATA_GENERAL       = wa_data_general
    DATA_GENERALX      = wa_data_generalx
    DATA_SPECIFIC      = wa_data_specific
    DATA_SPECIFICX     = wa_data_specificx
  IMPORTING
    RETURN             = wa_return.

if wa_return-type = 'S' or wa_return-type = ' '.
  CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.
  write: / 'Equipment number', wa_equipment_number, 'got changed.'.
else.
  write: / 'Error changing equipment.'.
  write: / wa_return-message.
endif.
```

Figure 21: Sample call to BAPI_EQUI_CHANGE.

Exercise 2: Calling SAP BAPIs from ABAP

Exercise Objectives

After completing this exercise, you will be able to:

- Call SAP developed BAPIs from custom applications

Business Example

You have identified certain SAP BAPIs that you would like to use from your custom ABAP applications. You now need to know how you could do this.

Task:

From an ABAP custom application, you will call SAP BAPIs in order to interface with ECC 'Plant Maintenance' common components. You need this application to retrieve the details of a specific piece of equipment.

1. Find the appropriate BAPI that will return you the data required by your custom application.
2. Write an ABAP program called ZBC427_##_READ_EQUIPMENT.
3. Via a selection-screen (PARAMETER statement), this program should allow the user to enter an equipment number, and should then display (within a simple list), the details associated to that particular equipment. The details required are :

Equipment description, Equipment Category and Equipment Fleet Expiry Date.



Hint: You should use, as a reference to your equipment number, the dictionary field `BAPI_ITOB_PARMS-EQUIPMENT`

4. (Optional) Write sample test ABAP programs that will test out the example BAPI calls shown earlier in this lesson (CompanyCode.GetList, PieceOfEquipment.CreateFromData, PieceOfEquipment.Change). See lesson documentation for examples.

Solution 2: Calling SAP BAPIs from ABAP

Task:

From an ABAP custom application, you will call SAP BAPIs in order to interface with ECC 'Plant Maintenance' common components. You need this application to retrieve the details of a specific piece of equipment.

1. Find the appropriate BAPI that will return you the data required by your custom application.
 - a) Using the BAPI explorer, find Business Object *PieceOfEquipment*. Identify which method you should use. which in this case is method *PieceOfEquipment.GetDetail1*.
 **Note:** Method *PieceOfEquipment.GetDetail* is marked as obsolete and you should not be used.
2. Write an ABAP program called ZBC427_##_READ_EQUIPMENT.
 - a) Using either SE38 or SE80, create your ABAP REPORT.
3. Via a selection-screen (PARAMETER statement), this program should allow the user to enter an equipment number, and should then display (within a simple list), the details associated to that particular equipment. The details required are :

Continued on next page

Equipment description, Equipment Category and Equipment Fleet Expiry Date.



Hint: You should use, as a reference to your equipment number, the dictionary field *BAPI_ITOB_PARMS-EQUIPMENT*

- See below for sample source code.

```

REPORT  ZBC417_00_READ_EQUIPMENT.

parameters: p_eqno type BAPI_ITOB_PARMS-EQUIPMENT.
data wa_general      type BAPI_ITOB.
data wa_specific     type BAPI_ITOB_EQ_ONLY.
data wa_fleet        type BAPI_FLEET.
data wa_return       type BAPIRET2.

start-of-selection.

CALL FUNCTION 'BAPI_EQUI_GETDETAIL'
  EXPORTING
    EQUIPMENT          = p_eqno
  IMPORTING
    DATA_GENERAL_EXP   = wa_general
    DATA_SPECIFIC_EXP  = wa_specific
    DATA_FLEET_EXP     = wa_fleet
    RETURN              = wa_return.

if wa_return-type = 'S' or wa_return-type = ''.
  write: / 'Details for equipment : ', p_eqno.
  write: / 'Equipment description : ',
         wa_general-descript.
  write: / 'Equipment category : ',
         wa_specific-equipcatgry.
  write: / 'Equipment fleet expiry date :',
         wa_fleet-expiry_date.
else.
  write: / 'Error calling BAPI_EQUI_GETDETAIL.'.
  write: / wa_return-message.
endif.

```

- (Optional) Write sample test ABAP programs that will test out the example BAPI calls shown earlier in this lesson (CompanyCode.GetList, PieceOfEquipment.CreateFromData, PieceOfEquipment.Change). See lesson documentation for examples.

- no solution



Lesson Summary

You should now be able to:

- Understand the basics behind calling BAPIs from ABAP applications.
- Call read/write style BAPIs from ABAP applications.
- Understand the special error handling mechanism used by BAPIs



Unit Summary

You should now be able to:

- Use the BAPI transaction code.
- Find an SAP Business Object.
- Find the documentation related to SAP BAPIs.
- Understand the basics behind calling BAPIs from ABAP applications.
- Call read/write style BAPIs from ABAP applications.
- Understand the special error handling mechanism used by BAPIs

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 3

Creating customer specific BAPIs

Unit Overview

This Unit describes, defines, and reviews the Business Scenario to determine what BAPIs are needed for our exercise scenario.

This Unit also covers the BAPI development guidelines used to create BAPIs. In all, you will create 3 BAPI style function modules (GetList, GetDetail, Update).

Once the function modules completed and tested, you will create the BOR Business Object for your scenario and incorporate the function modules as API methods to your Business Object.



Unit Objectives

After completing this unit, you will be able to:

- Describe the Business Scenario
- List the phases of a scenario and their purpose
- Find and access tools needed for BAPI development
- List the naming conventions related to BAPIs and related components
- Describe the business scenario
- Define the business scenario
- Review the business scenario
- Define the components related to the BAPI interface
- Outline the structured components used in writing the source code of the BAPI
- Explain the important role of documentation of the BAPI
- Test the documentation for accuracy and completeness
- Test an RFC-enabled function module
- Describe Business Objects and their relation to the BOR
- Create a Business Object
- Add a BAPI as an API method to a business object.
- Test the BAPI Method
- Define the terms database LUW and SAP LUW

- Explain the need to bundle changes to the database tables
- Apply the knowledge of updating techniques to the distributed application environment and the use of BAPIs
- Describe the authorization concept for BAPIs
- Describe Authorization Objects and Authorizations
- List requirements to implement authority checking
- Use lock modules
- Find, maintain, and generate lock modules
- Explain the role of lock objects
- Explain the different update techniques available
- Perform bundled database updates
- Create and use update function modules

Unit Contents

Lesson: The Business Scenario	51
Lesson: Tools and Naming Conventions	58
Exercise 3: Analyze a BAPI.....	71
Lesson: Creating the Business Scenario	75
Exercise 4: BAPI Business Scenario	81
Lesson: BAPI development guidelines.....	87
Exercise 5: Create the Get List function module	107
Exercise 6: Create a Get Detail function module.....	113
Exercise 7: Document the GetList and GetDetail function modules .	117
Lesson: Testing an RFC enabled Function Module	120
Exercise 8: Test the RFC-enabled function module	123
Lesson: Business Object Repository.....	127
Lesson: Business Objects	135
Exercise 9: Business Object	143
Lesson: Business Object Components	147
Exercise 10: Add the GetList and GetDetail RFC function modules to the BOR as a BAPIs.....	159
Lesson: Logical Unit of Work	163
Lesson: Authorizations.....	175
Lesson: Locking.....	182
Lesson: Update Techniques.....	199
Exercise 11: Create an Update BAPI	219

Lesson: The Business Scenario

Lesson Overview

Before you program a BAPI, you should clearly define the processes and situations the BAPI will be used for. BAPIs should only be implemented within the context of a meaningful business scenario.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the Business Scenario
- List the phases of a scenario and their purpose

Business Example

If you've ever been involved with the internal development of new business applications in a company, you know the importance of analysing the existing systems. The objective is to add new functionality that is currently missing as well as correcting and completing known functions. This can only be done properly if you plan that new functionality and know exactly how it's to be used.

Describing the Scenario

The Business Process

A business process consists of a series of individual business functions. The content of the business process should be described separately from any technical details.

Creating a sales order in an affiliate that includes a credit check being performed at the company headquarters.

To define a business process, you have to:

- Define the purpose and the scope of the business process
- Identify the individual steps in the business process. To do this, you can use a process model or the tool Use Cases.

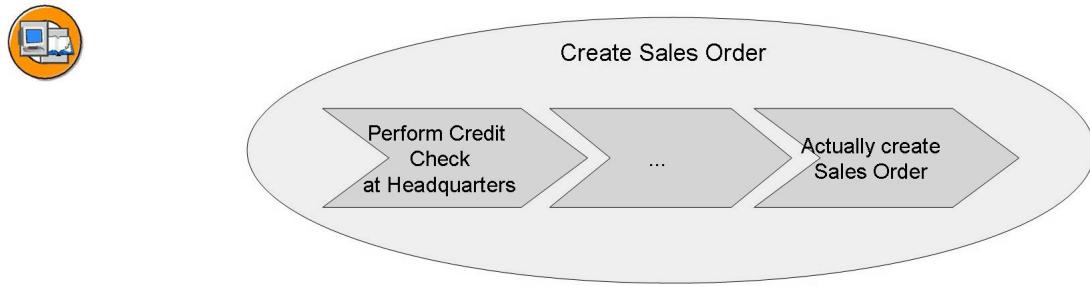


Figure 22: Business Process

The Scenario

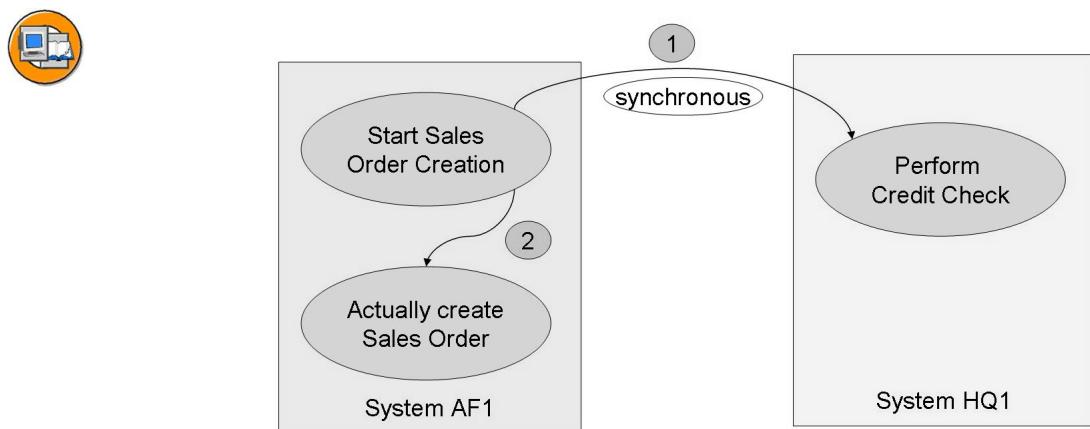


Figure 23: Business Scenario

A scenario, the computerized implementation of a business process, describes the distribution and interaction of the tasks between the participating components. There may be several scenarios that implement the same business process.

You have to create sales orders that require credit checks in different SAP Systems (central accounting, decentralized sales, and distribution).

The process of defining the scenario involves:

- Identifying the relevant components and the tasks they perform.
- Determining whether application systems are to be integrated in the scenario or other frontends are to be connected. This can affect the granularity of the steps, for example.
- Determining the information and process flows. Here you have to:
 - Determine which steps are to be processed system-wide and which steps are to be processed within a single component.
 - Define which data is exchanged between which components and who initiates this exchange.
 - Determine the sequence in which the individual steps are processed.
 - Identify the steps belonging to a single transaction (Logical unit of work or LUW).

For example, developers must ask themselves whether it makes sense to create a customer in one LUW and then create a sales order in the same LUW .

- Handle errors in a more precise and comprehensive manner than with local applications.
- You have to decide whether system coupling should be narrow or loose for the scenario. You should consider factors such as system availability and performance and how often the scenario is used.
- You should identify all steps critical to performance must be identified.
- You should also consider which Releases are to be supported within the scenario.
- For each scenario, you must identify a person responsible for ensuring that the scenario is correct and up-to-date.

Business Object Types and BAPIs

Each component in the scenario must provide services so that the cross-component steps can be carried out. You have to work out how to distribute responsibility for the services between the business object types and their BAPIs. For example, the BAPI CreateFromData of business object type SalesOrder is used to create a sales order in the SAP System from an external application.

For each component, you have to first determine the required business object types. You should consider the following issues:

- Encapsulate required functions in business object types. This involves breaking down the whole system into separate responsibilities. The breakdown and encapsulation of functions must be explicit and exclusively distinct.
- Do business object types already exist for these responsibilities?
- Do design patterns already exist? Find out whether any problems have already been dealt with. For example, Header/item pattern for SalesOrder, PurchaseOrder, etc.
- Delimit the responsibility for other business object types.
- Determine the services provided based on the defined responsibility.

For each business object type you have identified, you have to determine how the services assigned to it can be implemented using BAPIs. You should consider the following issues:

- Each service is implemented by one or more BAPIs (method of business object type).
- BAPIs make available the functions of a business object type. You should be able to use them independently of individual scenarios and also in different scenarios. To make it easier to use a BAPI in different scenarios, the BAPI approach should be such that the BAPI's parameters and fields are assigned separately according to the application.
- Note that this type of scenario can affect the granularity of the BAPI. Application systems are integrated differently from the integration of alternative frontends: **Integrating application systems:** The integration of application systems typically involves program-to-program communication, loose asynchronous coupling, and the exchange of larger volumes of data. The main requirement of the business object types and BAPIs used in these scenarios is that performance is high, for example, by minimizing the number of calls. The result is a rough granularity of the application that is implemented by more extensive BAPIs. An example might be a program to automatically create a sales order. The program uses the business object type SalesOrder with the BAPI CreateFromData. The complete sales order is created in the sending system and then sent to the receiving system. **Integrating alternative frontends:** Scenarios with alternative frontends represent human-to-machine communication and can be implemented synchronously as well as asynchronously. Business object types and BAPIs must be structured to ensure flexibility, configurability, and a minimal number of error situations. The result is a finer granularity of the application that should correspond to the dialog processing in SAP system. An example of this would be a customer sales order that is created interactively on the Internet. The sales order can be created using the two methods CreateFromData and AddItem of the business object type SalesOrder. In this case the method CreateFromData simply creates the sales order header, while the method AddItem adds new sales order items.
- The scenario must be structured to get all the information required for a BAPI call beforehand.
- If there are any BAPIs that are Customizing-dependent, BAPIs must be provided that can export these Customizing settings.

Defining the Scenario

Prerequisites

You must completely describe the scenario before you do the definition. This description will be the input information for the actual definition.

Definition Process

The entire development process of a BAPI takes place in the framework of form-controlled projects to achieve the maximum quality, stability and usability of the BAPI. The BAPI Explorer provides all the tools used to create BAPIs and the required development objects in an integrated programming environment.

For each of these projects, a project form takes you step-by-step through the entire development process and provides direct navigation options to the required development tools and information. Within the project management, you can save and delete your projects. You can also edit the projects managed by other users by selecting Other users. More information on access and usage of the tool is provided in the lesson Tools and Naming Conventions.

Reviewing the Scenario

Before the scenario can be converted and started with a concrete definition and implementation of the BAPI, you should review the scenario. All persons involved in the BAPI development and those responsible for quality control should be involved in this review.

In order for the review to be meaningful, the following questions (at least) should be answered:

- Does the scenario make sense as it is planned?
- Have all the tasks required for the scenario description been properly completed?
- Do all the BAPIs in the scenario work smoothly together?

You should only start developing the BAPI when you have successfully completed the review.



Lesson Summary

You should now be able to:

- Describe the Business Scenario
- List the phases of a scenario and their purpose

Related Information

- For additional information on developing BAPIs, refer to the BAPI Programming Guide available on SAPNET.

Lesson: Tools and Naming Conventions

Lesson Overview

Once you have completed the conceptual design of the BAPIs, information specific to SAP must now be considered in the implementation phase. For example, the SAP data structures required are determined, the program logic needs to be implemented and the business object needs to be implemented in the BOR. This lesson provides an overview of the relevant components and tools.



Lesson Objectives

After completing this lesson, you will be able to:

- Find and access tools needed for BAPI development
- List the naming conventions related to BAPIs and related components

Business Example

You are going to develop a business scenario using BAPIs. Before starting this process, you want to use some of the available tools to locate any existing BAPIs to minimize development time.

The Implementation Process

A BAPI is defined in the Business Object Repository (BOR) as an API method of an SAP business object type. Business object types and their BAPIs are described and created in the BOR. A BAPI is usually implemented as an RFC-enabled function module. These function modules are created and described in the Function Builder. The definitions and descriptions of the data structures used by the BAPI are defined in the ABAP Dictionary.

The graphic illustrates the relationships between the components:

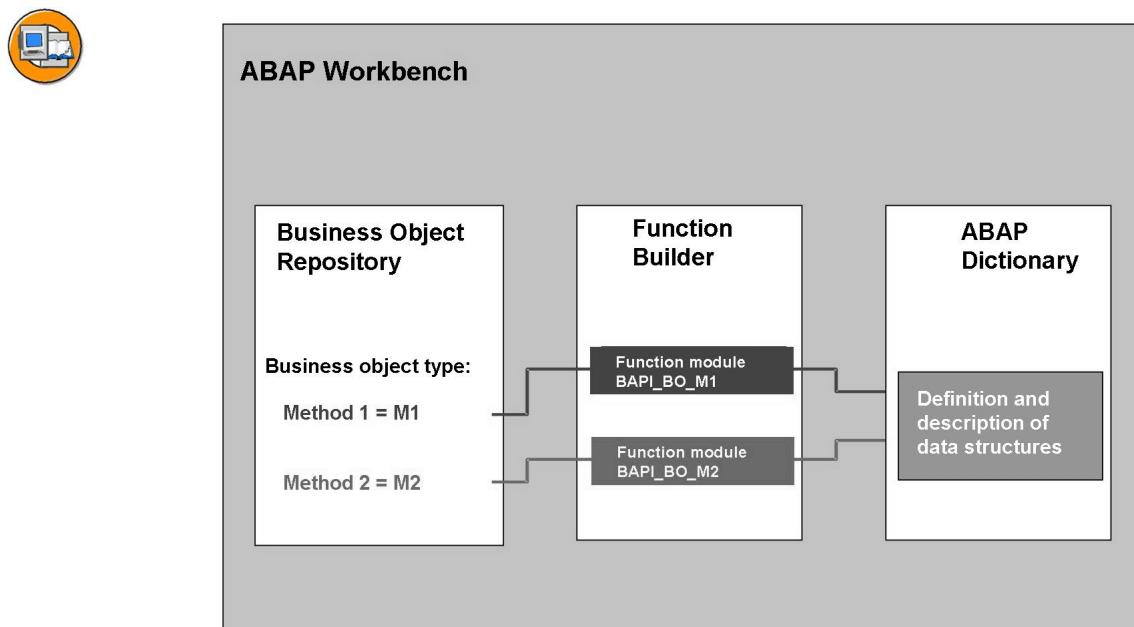


Figure 24: Component Relationships

Process Flow

The structure of the tools in the ABAP Workbench determine how the BAPI is implemented. All three phases have accompanying documentation that is created in the corresponding tools. All the required work steps can be started from the BAPI Explorer.

The most important tools used for developing BAPIs are the BAPI Explorer, the ABAP Dictionary, the Function Builder, and the Business Object Repository.

The BAPI Explorer

The BAPI Explorer is the SAP tool used to search for existing BAPIs. It also offers describing the steps and tools used to develop BAPIs.

Use

From the BAPI Explorer, application developers can get an overview of the status of BAPIs in the BOR. The BAPIs can be determined either by the position of the associated object or interface type in the component hierarchy in the BOR, or from an alphabetical list. The BAPI Explorer provides all information required to use a particular BAPI.

The BAPI Explorer is used internally in SAP to develop BAPIs, but can also be used by customers and partners. The BAPI Explorer provides access to all the tools used to create BAPIs and the required development objects in an integrated programming environment.

Access

To call the BAPI Explorer, choose *Tools → Business Framework → BAPI Explorer*. To call it directly, use Transaction BAPI.

For more details on the BAPI Explorer functionality, refer to the lesson on **Finding an SAP developed BAPI** in the previous chapter.

Working with Tools and Projects

The views Tools and Projects in the work area are mainly used for developing BAPIs.

Depending on the development object selected in the hierarchy display in the Tools view, the following tools and navigation options are provided:

- Direct access to the Business Object Builder, Function Builder, and ABAP Dictionary.
- List generator to create lists of BAPIs using specified selection criteria.

In the Project view, you can create projects to assist you with following and documenting the development procedures:

- Implementing new BAPIs
- Changing released BAPIs
- Requesting a new business object type

For each of these projects, there is a project form that takes you step-by-step through the entire development process and provides direct navigation options to the required development tools and information.

Within the project management you can save and delete your projects and you can edit the projects managed by other users by selecting Other users. If you have installed Microsoft Internet Explorer Version 4.0 on the front-end computer, you can print the project form using the standard SAPgui print function.

The ABAP Dictionary

All the data definitions used in the system can be described and managed in the central ABAP Dictionary. New or modified information is automatically made available to all system components ensuring data integrity, data consistency, and data security.

The first step of the implementation phase is to define all the data structures (including domains and data elements) in the ABAP Dictionary. These data structures are required for the parameters of the BAPI to be implemented. You can access the ABAP Dictionary from the project form in the BAPI Explorer as well as direct access with the menu path *SAP standard menu → Tools → ABAP Workbench → Development → ABAP Dictionary*, or enter the transaction code SE11.

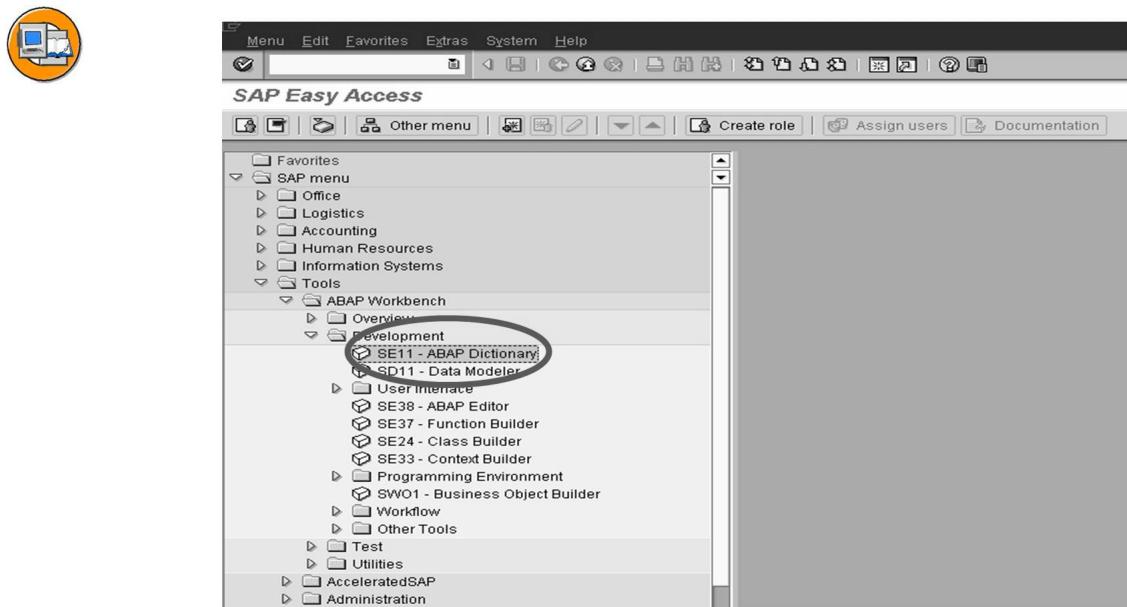


Figure 25: ABAP Dictionary Access

The screenshot shows the 'Dictionary: Display Structure' window. The title bar includes 'Structure', 'Edit', 'Goto', 'Utilities', 'Extras', 'Environment', 'System', and 'Help'. Below the title bar is a toolbar with icons. The main area has tabs: 'Attributes' (selected), 'Components', 'Entry help/check', and 'Currency/quantity fields'. The 'Structure' dropdown is set to 'SYST' and 'Short text' is set to 'ABAP System Fields'. A table displays the following data:

Component	Component type	DTyp	Length	Dec.p...	Short text
INDEX	SYINDEX	INT4	10	0	Loops, number of current pass
PAGNO	SYPAGNO	INT4	10	0	List creation, current page
TABIX	SYTABIX	INT4	10	0	Internal table, current line index
TFILL	SYTFILL	INT4	10	0	Internal tables, current number of lines
TLOPC	SYTLOPC	INT4	10	0	Internal
TMAXL	SYTMAXL	INT4	10	0	Obsolete
TOCCU	SYTOCCU	INT4	10	0	Internal tables, initial main memory requirements
TTABC	SYTTABC	INT4	10	0	Obsolete
TSTIS	SYTSTIS	INT4	10	0	Internal
TTABI	SYTTABI	INT4	10	0	Obsolete
DBCNT	SYDBCNT	INT4	10	0	DB operations, number of table lines processed
FPDPOS	SYFPDPOS	INT4	10	0	Character strings, offset in character string
COLNO	SYCOLNO	INT4	10	0	List creation, current column of list
TMCT	SYTMCT	INT4	10	0	All list processing, page length of list

Figure 26: Dictionary Structure Display

Usage Conventions

To insure usability standards required for BAPIs, the following conventions are important:

- Each parameter must refer to a data structure in the ABAP Dictionary. In the case of structured parameters, this is always to the whole BAPI data structure. But if the parameter consists of only one field, it must refer to a field in a BAPI structure.
- You have to create separate data structures for BAPIs that are independent of the data structures generally used in the SAP application. Reason: When the BAPI is released, the underlying structures of the BAPI are frozen and restrictions apply if you want to later change the structures.
- You must not use APPENDs and INCLUDEs in BAPI data structures. This is because APPENDs and INCLUDEs can cause incompatibilities later if BAPI data structures are changed.
- Always try to use existing central data elements and domains for fields.
- You may have to define single values or a value table for the domain so that F4 help is available.
- All the useful Input Help (which can be used in the service BAPI Helpvalues.GetList) must be defined for the data structures/data elements. To do this, a foreign key must be specified in the fields of a BAPI structure. If a value table has been defined in a field domain, a foreign key must also be defined.
- The internal data format is used for all parameter fields.
- If an external key as well as an internal key has been defined in the database, the external key must always be used in the BAPI interface.
- For fields relevant for ISO (country, language, unit of measure, currency), additional fields for ISO codes are provided.
- All currency fields use the domain BAPICURR. In special cases, you could also use the domain BAPICUREXT .
- You must always use a period for the decimal point.
- All currency amounts and units of measurement must have the decimal point in the correct place because the BAPI always uses a standard amount format with commas. So when converting currency amounts, use the function modules: BAPI_CURRENCY_CONV_TO_EXTERNAL and BAPI_CURRENCY_CONV_TO_INTERNAL.
- With quantity fields, the reference field or reference table must be correctly set to the unit of measurement field.
- All the data elements for date fields must have the format YYYYMMDD.
- Single fields in a data structure can only be a maximum of 250 bytes so that an IDoc for asynchronous communication can be generated from a BAPI.

The Function Builder

With the Function Builder function, modules can be created, implemented, tested, and documented within a function group. The Function Builder contains a Function Library that serves as the central storage for all function modules. You can access the Function Builder from the project form in the BAPI Explorer as well as direct access with the menu path *SAP standard menu → Tools → ABAP Workbench → Development → Function Builder*, or enter the transaction code SE37.

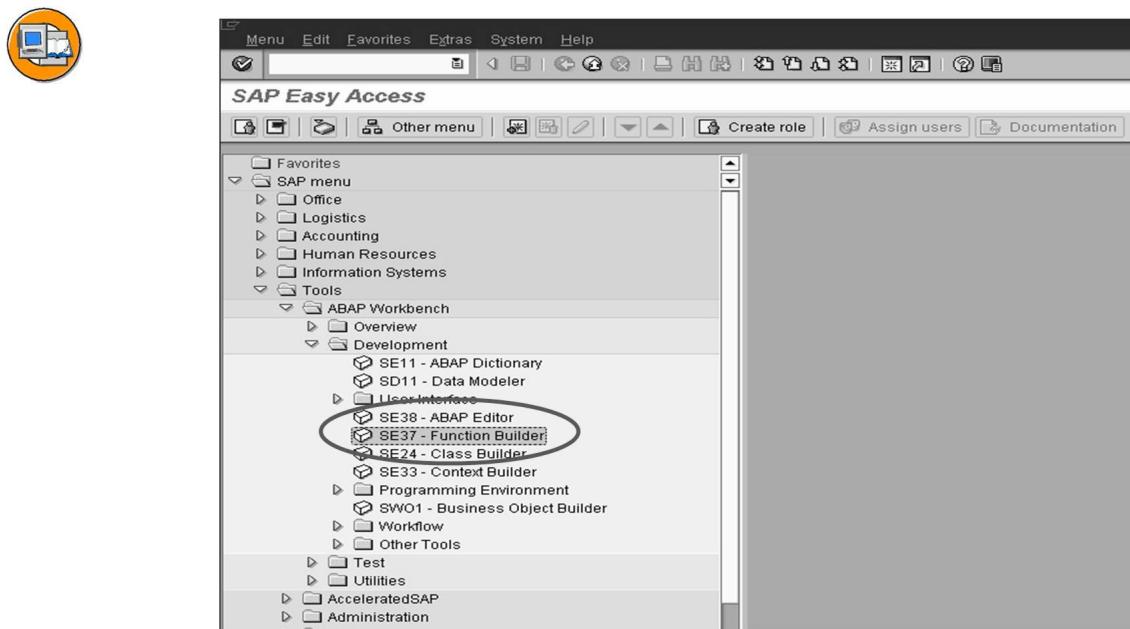


Figure 27: Function Builder Access



```

Function module  Edit  Goto  Utilities  Environment  System  Help
Function Builder: Display ENQUEUE_EMMATAE
Function module  ENQUEUE_EMMATAE  Active
Attributes  Import  Export  Changing  Tables  Exceptions  Source code
FUNCTION ENQUEUE_EMMATAE.
  **Lokale Schnittstelle:
  IMPORTS ...
    VALUE(MODE_MARA) TYPE ENGMODE DEFAULT 'E'.
    VALUE(MODE_MVKE) TYPE ENGMODE DEFAULT 'E'.
    VALUE(MODE_MBEW) TYPE ENGMODE DEFAULT 'E'.
    VALUE(MODE_MLGN) TYPE ENGMODE DEFAULT 'E'.
    VALUE(MODE_MARC) TYPE ENGMODE DEFAULT 'E'.
    VALUE(MANDT) TYPE MARA-MANDT DEFAULT SY-MANDT.
    VALUE(MATNR) TYPE MARA-MATNR OPTIONAL.
    VALUE(VKORG) TYPE MVKE-VKORG OPTIONAL.
    VALUE(VTWE6) TYPE MVKE-VTWE6 OPTIONAL.
    VALUE(BWKEY) TYPE MBEW-BWKEY OPTIONAL.
    VALUE(BWTAR) TYPE MBEW-BWTAR OPTIONAL.
    VALUE(LGNUM) TYPE MLGN-LGNUM OPTIONAL.
    VALUE(WERKS) TYPE MARC-WERKS OPTIONAL.
    VALUE(,MATNR) DEFAULT SPACE.
    VALUE(,VKORG) DEFAULT SPACE.
    VALUE(,VTWE6) DEFAULT SPACE.
    VALUE(,BWKEY) DEFAULT SPACE.
    VALUE(,BWTAR) DEFAULT SPACE.
    VALUE(,LGNUM) DEFAULT SPACE.
    VALUE(,WERKS) DEFAULT SPACE.
    VALUE(,SCOPE) DEFAULT '2'.
    VALUE(,WAIT) DEFAULT SPACE.
    VALUE(,COLLECT) TYPE DDENQCOLL DEFAULT ''.
  EXCEPTIONS ...
    FOREIGN_LOCK.
    SYSTEM_FAILURE.

```

Figure 28: Function Module Display

Once the parameters have been defined, the function module supporting the BAPI is created in the Function Builder.

When creating the function module, you must follow these rules:

- All the BAPIs belonging to one SAP business object type should be created in one function group. You should only deviate from this rule in exceptional cases.
- BAPIs belonging to different SAP business object types must not be put into the same function group.
- Function module must be marked as a Remote Enabled Function (RFC).
- All parameters to the function module must be passed by value.
- Source code specific BAPI standards will be covered later.

The Business Object Repository (BOR)

The Business Object Repository (BOR) contains the SAP business object types and SAP interface types as well as their components, such as methods, attributes, and events. A BAPI is defined in the BOR as a method of an SAP business object type. The BOR has the following functions for SAP business object types and their BAPIs:

- Enables an object-oriented view of the SAP System data and processes. SAP application functions are accessed using methods (BAPIs) of SAP business object types. Implementation information is encapsulated; the user can only see the interface functionality of the method.
- Arranges the various interfaces in accordance with the component hierarchy. Functions can be searched and retrieved quickly and simply.
- Manages BAPIs in release updates. BAPI interface enhancements made by adding parameters are recorded in the BOR. Previous interface versions can thus be reconstructed at any time. When a BAPI is created, the release version of the new BAPI is recorded in the BOR. The same applies when any interface parameter is created. The Function Builder manages the version control of the function module that a BAPI is based on.
- Ensures interface stability. Any interface changes that are carried out in the BOR are automatically checked for syntax compatibility against the associated development objects in the ABAP Dictionary.

You first have to define the names, parameters, and characteristics of the BAPI and determine the structures in the ABAP Dictionary that the BAPI will be based on. Only when you have done this can the BAPI be implemented in the Function Builder and the required programming objects be created in the BOR.

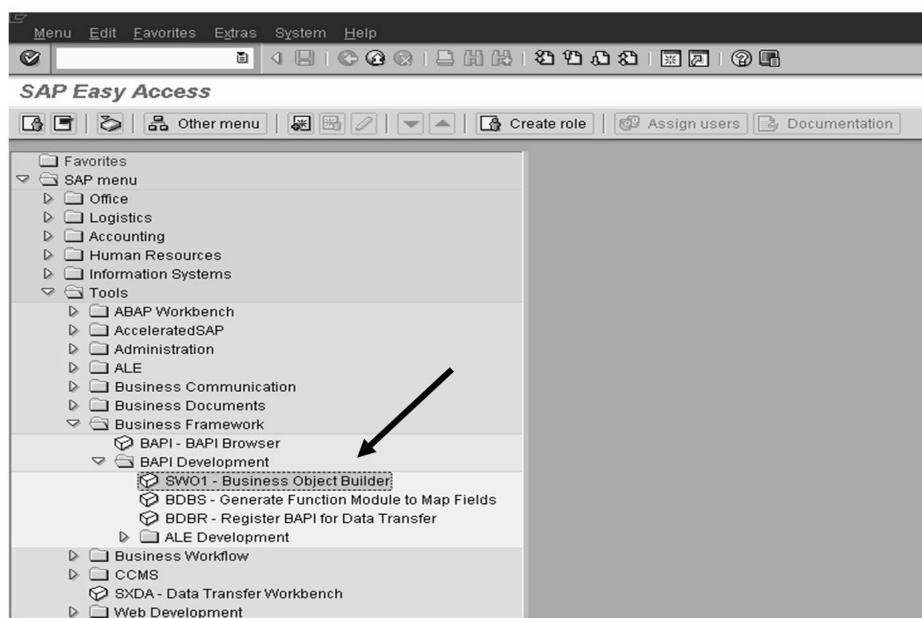


Figure 29: BOR Access (1)

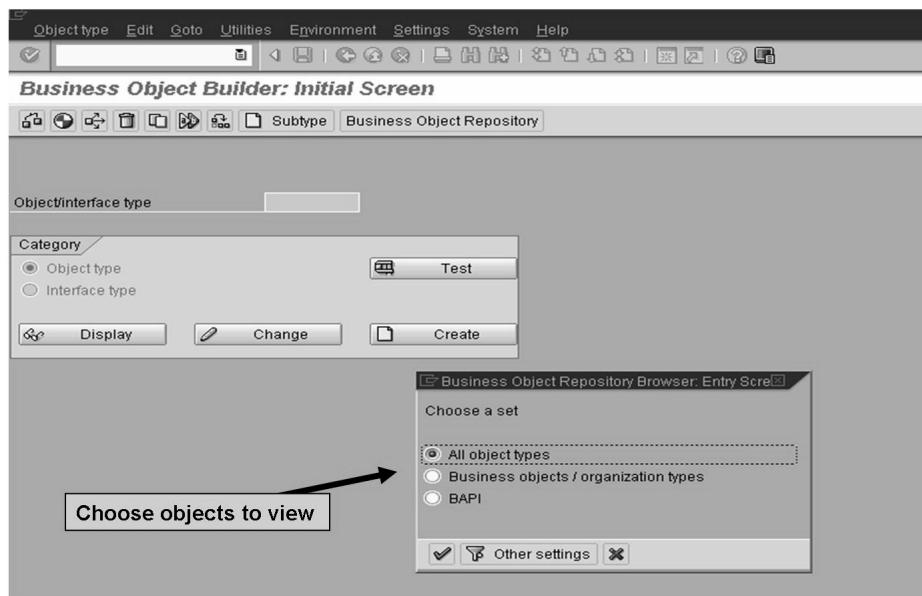


Figure 30: BOR Access (2)

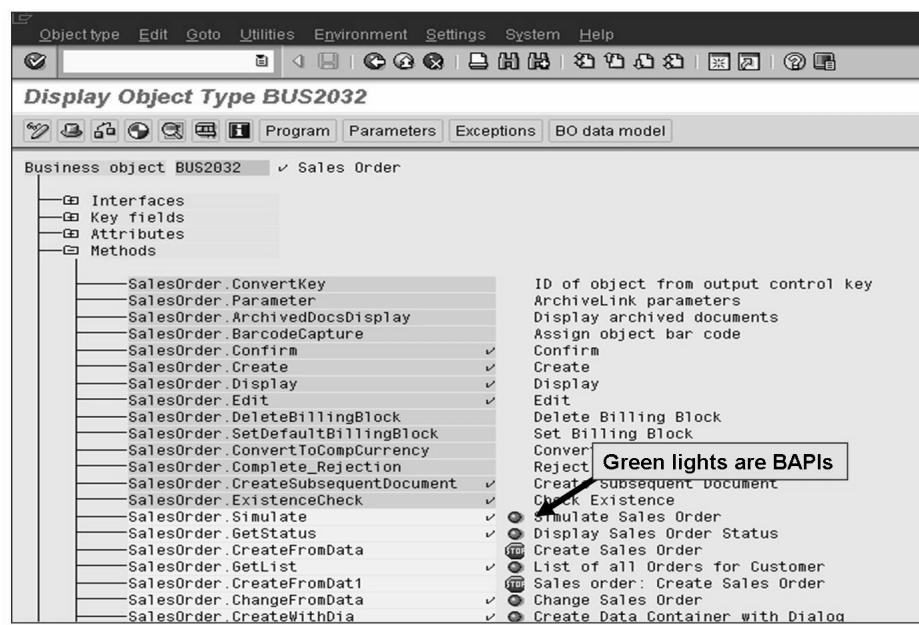


Figure 31: Business Object Display

To find relevant SAP business object types in the BOR:

- Choose *Tools* → *Business Framework* → *BAPI Development* → *Business Object Builder*. On the initial *Business Object Builder* screen, you can directly access the SAP business object type or interface type if you know the technical name of the object (object type). You have already identified the technical name of the object. Otherwise, choose *Business Object Repository*.
- To display object types, in the next dialog box indicate whether you want to display all object types or only business object types. Then choose Continue.
- To display SAP interface types, in the next dialog box choose *Other settings* and then select *Interface*. The application hierarchy displays. Search for the required business object type or interface type in the application hierarchy and select to open it.

Other Tools

Although the tools listed above are the most important of the development tools used in BAPI development, there are others that will be used less frequently. The most significant of these are:

Repository Information System

You use the Repository Information System to perform a generalized search for most objects in the SAP System. To access the SAP Repository Information System from the ABAP Workbench, choose *Overview* → *Repository Infosys* or use transaction code SE84. This tool lets you search for objects, such as function modules or dictionary structures, without having to first access the related development tool. This tool is very powerful in its ability to use attributes or common characteristics as selection criteria.

Data Browser

You use the Data Browser to access table entries without using an ABAP program. With the Data Browser, you can display table records, display all table field values and related text field values, and branch from table entries to their related check-table entries.

To start the Data Browser, choose *Overview* → *Data Browser* from the Workbench tools initial screen. You can also reach the Data Browser from the *Environment* menu in the Repository Browser or the *Utilities* menu in the ABAP Editor. The Data Browser prompts you for a table name.

Message Maintenance

Messages allow you to communicate with users from your programs. They are mainly used when the user has made an invalid entry on a screen or to describe the results of a requested action.

Starting the Message Maintenance function can be accomplished by: Using forward navigation from the ABAP Editor. Choosing *Goto* → *Messages* from your program in the ABAP Editor. The Maintain Messages screen appears. By default, the system displays the message class linked to the current program.

Following the menu path *SAP standard menu* → *Tools* → *ABAP Workbench* → *Development* → *Programming Environment* → *Messages*
Enter **Transaction SE91**.

Naming Conventions

The following naming conventions are important. If IBUs, partners, and customers are creating data structures (or domains or data elements), the Namespaces provided by SAP must be observed.

BAPI Data Structures

- All BAPI specific dictionary data structure names must begin with <namespace>BAPI.
- BAPI structure names should be as meaningful as possible.
- If the data structure is used for ALE Integration, the BAPI structure name must not be longer than 27 characters, otherwise the automatically generated name for the associated segment will be too long and will have to be manually changed later.

Fields

- The fields in structures must have meaningful English names with a maximum of 30 characters.
- A meaningful English default field name with a maximum of 30 characters should be defined for each data element.

Function Modules

- All function modules must have the naming convention: <namespace>BAPI_<business object>_<method>. A maximum of 30 characters is allowed. If required, you can abbreviate the name but still must follow the above convention. You must still be able to recognize the business object type assignment.

Exercise 3: Analyze a BAPI

Exercise Objectives

After completing this exercise, you will be able to:

- Find a BAPI module and check to see if it adheres to the BAPI standards.

Business Example

In order to understand BAPI development standards, you will look at an SAP supplied BAPI and analyze it.

Task:

Find the CompanyCode.GetList method and analyze how the SAP developer implemented it.

1. Go to the BAPI Explorer tool.
2. Find the CompanyCode business object.
3. Find the GetList method.
4. Has the developer of this BAPI, documented it.
5. Has the developer released the BAPI.
6. What is the name of the BAPI function module that implements this method.
7. Does the name of the function adhere to the standards for BAPIs.
8. If the function module an RFC.
9. Does the BAPI use Exceptions.
10. Does the BAPI return a parameter called RETURN to the user.
11. Are all BAPI parameters typed using special BAPI* structures.
12. Does the BAPI do authority-checks.
13. Does this BAPI have code that dialogs with users.

Solution 3: Analyze a BAPI

Task:

Find the CompanyCode.GetList method and analyze how the SAP developer implemented it.

1. Go to the BAPI Explorer tool.
 - a) On the SAP menu, choose *Tools → Business Framework → BAPI Browser* or enter transaction code **BAPI**.
2. Find the CompanyCode business object.
 - a) Go to the Alphabetical tab strip of the BAPI transaction. Page down until you find the business object CompanyCode.
3. Find the GetList method.
 - a) Expand the CompanyCode business object. The GetList method should be there.
4. Has the developer of this BAPI, documented it.
 - a) Select the *GetList* method and go to the Documentation tab strip.
5. Has the developer released the BAPI.
 - a) Select the GetList method and go to the Details tab strip. At the bottom of the tab, there is a Release Status of the method. This BAPI should be marked as released.
6. What is the name of the BAPI function module that implements this method.
 - a) Select the GetList method. In the Details tab strip, you should see the name of the ABAP function module that implements this method (ie **BAPI_COMPANYCODE_GETLIST**).
7. Does the name of the function adhere to the standards for BAPIs.
 - a) Yes. The standard for BAPI function module names is **BAPI_<name_of_business_object>_<name_of_method>**.
8. If the function module an RFC.
 - a) Yes. Double click on the function module name and that should bring you to SE37, editing function module **BAPI_COMPANYCODE_GETLIST**. Go to the Attributes tab strip and note that the function module has the property of being Remote Enabled.

Continued on next page

9. Does the BAPI use Exceptions.
 - a) No. Go to the Exceptions tab strip of the function module and note that the section is empty.
10. Does the BAPI return a parameter called RETURN to the user.
 - a) Yes. Go to the Export tab strip and note that there is a RETURN parameter that the BAPI will initialize for the user of the BAPI.
11. Are all BAPI parameters typed using special BAPI* structures.
 - a) Yes. This BAPI has no import parameters. The export parameter RETURN is typed using structure BAPIRETURN. The tables parameter COMPANYCODE_LIST is typed on structure BAPI0002_1.
12. Does the BAPI do authority-checks.
 - a) Yes. Go to the Source code of this BAPI and note that it calls a subroutine to do an authority-check of object S_TABU_DIS.
13. Does this BAPI have code that dialogs with users.
 - a) No. If you search thru the code, it does not have any WRITE, CALL SCREEN, MESSAGE statements.

 **Note:** This BAPI might have a message statement, but if you analyze the algorithm, you will notice that the message statement will never be executed.



Lesson Summary

You should now be able to:

- Find and access tools needed for BAPI development
- List the naming conventions related to BAPIs and related components

Related Information

For additional information on developing BAPIs, refer to the BAPI Programming Guide and the BAPI Programming Guide Reference on SAPNET.

Lesson: Creating the Business Scenario

Lesson Overview

This unit discusses setting up the business scenario that will be used to create the BAPIs. This includes the three phases: describe the scenario, define the scenario, and review the scenario.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the business scenario
- Define the business scenario
- Review the business scenario

Business Example

You have a business requirement that needs to get information from an SAP System regarding the contacts of certain customers.

For analysis purposes, the business process and the scenario implementing this business process must be described in detail. The Business Object Types and their BAPIs that are used in the scenario must also be identified. The main dilemma with developing BAPIs is the conflicting needs: on the one hand, BAPIs should be developed so that they can be used in different scenarios but the interface also must be clearly structured. It is important that you do not stipulate in this phase exactly what the signature of the individual BAPIs is to look like. The BAPIs required for the scenario are simply identified and their functions and the data they require defined. The analysis can be divided into three phases: Describe the Business scenario, define the Business scenario, and review the business scenario

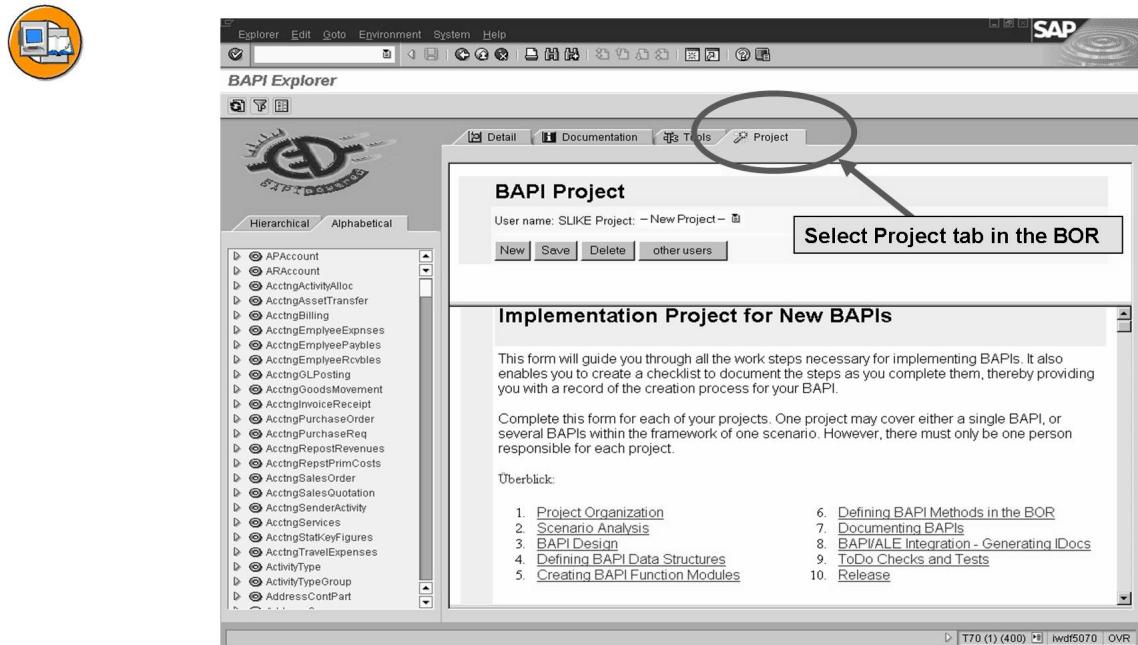


Figure 32: BAPI Project

Describe the Business Scenario

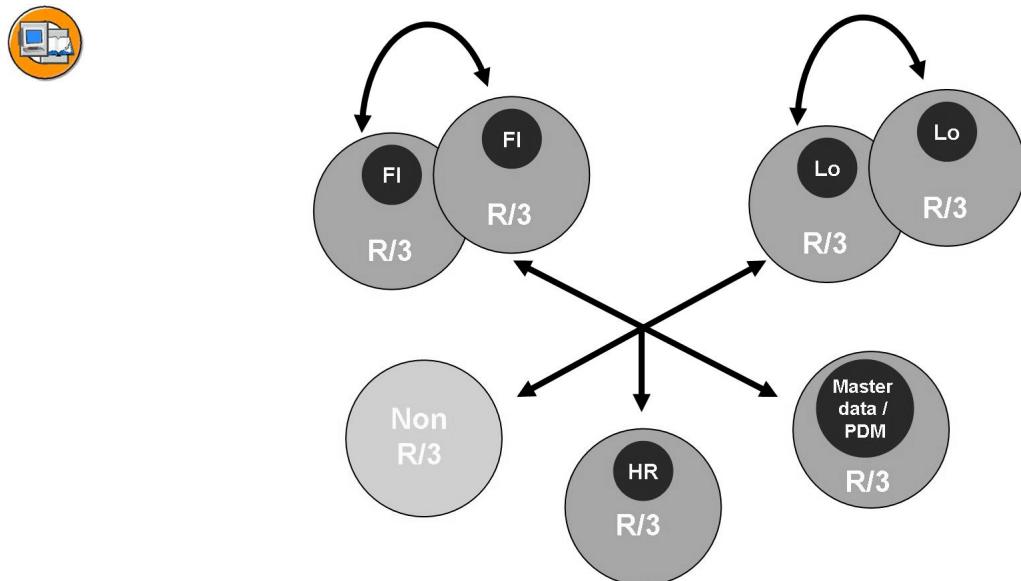


Figure 33: Sample Business Process

The Business Process

A business process consists of a series of individual business functions. The content of the business process should be described separately from any technical details.

Example: Creating a sales order in an affiliate with a credit standing check in the central organization. To define a business process you have to:

- Define the purpose and the scope of the business process
- Identify the individual steps in the business process. To do this, you can use a process model or a Use Case tool. The Use Case document is similar to a UML document. Currently, SAP does not provide tools for the business process step.

The Scenario

A scenario is the computerized implementation of a business process. It describes the distribution and interaction of the tasks between the participating components. There may be several scenarios that implement the same business process.

Example: Creating sales orders and credit standing checks in different SAP Systems (central accounting, decentralized sales and distribution).

The process of defining the scenario involves:

- Determining whether application systems are to be integrated in the scenario or other frontends are to be connected. This can affect the granularity of the steps, for example.
- Identifying the relevant components and the tasks they perform.
- Determining the information and process flows. Here you have to:
 1. Determine which steps are to be processed system-wide and which steps are to be processed within a single component.
 2. Define which data is exchanged between which components and who initiates this exchange.
 3. Determine the sequence in which the individual steps are processed.
 4. Identify the steps belonging to a single transaction (LUW). Example: Developers must ask themselves whether it makes sense to create a customer in one LUW and then create a sales order in the same LUW .
 5. Make error handling much more precise and comprehensive than with local applications.

You have to decide whether system coupling should be narrow or loose for the scenario. You should consider factors such as system availability and performance and how often the scenario is used.

All steps critical to performance must be identified.

You should also consider which Releases are to be supported within the scenario.

For each scenario you must identify a person responsible for ensuring that the scenario is correct and up-to-date.

Business Object types and BAPIs

If you look at a scenario where a BAPI is required to read data about a list of creditors and display them, you see that first a creditor has to be selected by one BAPI and then using another BAPI, specific details about this creditor are to be displayed. Here, the relevant business object for this scenario is Creditor.

In line with the scenario concept, BAPIs must complement each other to create a complete scenario. Their relationships with each other must be clearly defined.

To read a creditor's details as described in the above example scenario, two BAPIs are required; Display a list of creditors, and Display details of a specific creditor.

The interdependency between these two BAPIs is evident because first the creditor list is displayed to obtain the ID of the specific creditor sought. From this ID, details of the creditor can then be displayed.

However, the two BAPIs remain functionally independent of each other, because if the creditor ID is known, the BAPI Display list of specific creditor can be used without calling the BAPI Display list of creditors. Further, the BAPIs required to read creditor details in the above example scenario are only able to access data in the SAP Business Object Creditor. Other object types are not involved.

Once you have considered these issues, you will be able to clearly conceptualize the functionality of the planned BAPI(s). You will also have identified the SAP business objects relevant to the BAPI scenario.

Define the Business Scenario

Now that you have described the business scenario, you need to define the project. You can find the project form in the BAPI Explorer.

1. Start the BAPI Explorer in the relevant development system using Transaction BAPI.
2. Select the *Project* tab page and create a project to implement new BAPIs
3. A project form guides you through the entire procedure for creating a BAPI. In the first section, the basic data can be created for the defined scenario.

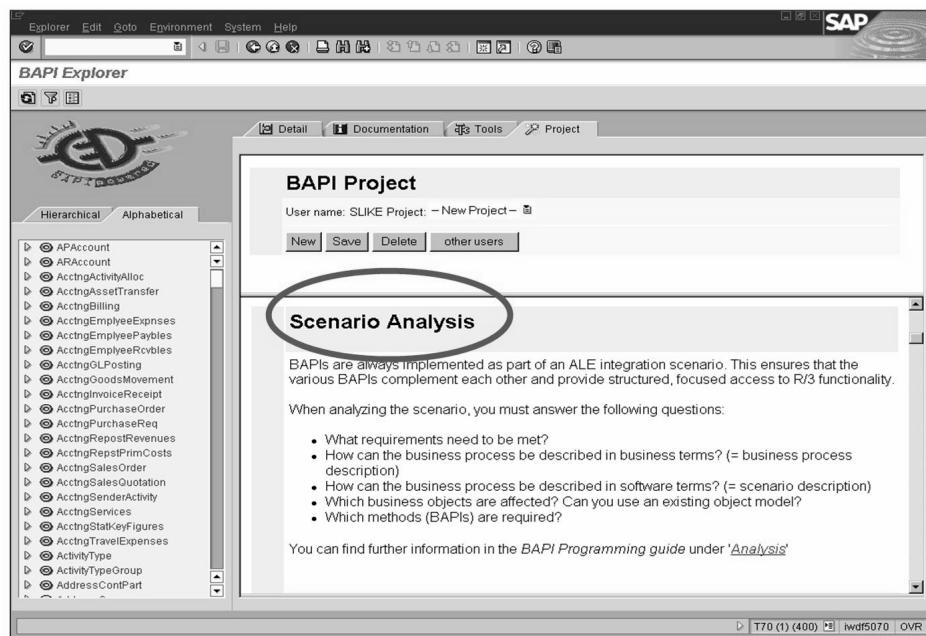


Figure 34: Scenario Analysis

Review the Business Scenario

Before the scenario can be converted and started with a concrete definition and implementation of the BAPI, the scenario should be reviewed. All persons involved in the BAPI development and those responsible for quality control should be involved in this review. The following questions should be answered:

- Does the scenario make sense as it is planned?
- Have all the tasks required for the scenario description been properly completed?
- Do all the BAPIs in the scenario work smoothly together?

You should only start developing the BAPI once you have successfully completed the review.

Exercise 4: BAPI Business Scenario

Exercise Objectives

After completing this exercise, you will be able to:

- Create a business scenario for using 1 or more BAPI in a business process to retrieve a list of customer data from an SAP System using an external call. Once you have this list, you should then be able to take any specific list item and retrieve all of the contacts for this customer. In addition, you want to be able to update the contact's name and/or telephone number if necessary.

Business Example

In order to successfully create the BAPI(s), you will need to retrieve the necessary information from an SAP System, you have to fully understand and document what processes and objects you will need to accomplish this task.

Task 1:

Identifying the Business Scenario Objects

1. Write down the types of BAPIs that you need for your business scenario.

Task 2:

Determine the tables that you will use to retrieve the Customer and Contact information.

1. Using the Repository Information System in sap system, find the Customer Master table and the Customer Contact table for your business scenario.

Task 3:

Verify if a business object already exists that you can use for your process.

1. Using the Business Object Repository, try to find a business object that already exists that uses the Customer Master table *SKNA1*.

Task 4:

Decide if you need to have an authorization check on the table(s).

1. Due to the nature of the data in the tables, you need to decide if you should perform an authorization check on the person attempting to extract data from the tables.

Continued on next page

Task 5:

Determine the possible error messages that you need to use in your business process including a successful message.

1. Step through your business scenario and try to list all of the possible error messages that you feel are needed to let the user of the BAPI know what happened during their call.

Solution 4: BAPI Business Scenario

Task 1:

Identifying the Business Scenario Objects

1. Write down the types of BAPIs that you need for your business scenario.
 - a) You have determined that you need a GetList BAPI to read a list of the key data from the customer table.
 - b) You have determined that you will need a GetDetail BAPI to read a specific customer's contact information.
 - c) You have also determined that you will need an Update BAPI in order to make a change to the contact's name or telephone number.

Task 2:

Determine the tables that you will use to retrieve the Customer and Contact information.

1. Using the Repository Information System in sap system, find the Customer Master table and the Customer Contact table for your business scenario.
 - a) From the main user menu screen, choose *Tools → ABAP Workbench → Overview → Information System (SE84)*.
 - b) Expand the folder ABAP Dictionary.
 - c) Select *Database Tables*.
 - d) In the Short Description entry box, type ***Customer Master***.
 - e) Choose *Execute*.
 - f) In the list of tables, determine and record the name of the *Customer Master* table and the *Customer Contact* table. Many tables will be listed. The actual SAP ECC tables for Customers and Contacts are KNA1 and KNVK. For this training Business Scenario, we will use copies of these tables (ie SKNA1 and SKNVK).

Continued on next page

Task 3:

Verify if a business object already exists that you can use for your process.

1. Using the Business Object Repository, try to find a business object that already exists that uses the Customer Master table *SKNA1*.
 - a) From the main sap system screen, type transaction **BAPI** in the transaction window and select *Enter* or from the SAP menu expand *Tools → Business Framework → BAPI Explorer* and *Enter*. Either of these methods takes you to the BAPI Explorer screen.
 - b) Select the *Alphabetical* tab.
 - c) Scroll through the listing looking for a business object that deals with customers.
 - d) When you have found the business object you believe may work, select the *Object name* and verify the *Object type* in the right window. If this is not listing the table you are going to use, then it will not work for your scenario.
 - e) If no business object type exists that can be used, you will have to create your own.

Task 4:

Decide if you need to have an authorization check on the table(s).

1. Due to the nature of the data in the tables, you need to decide if you should perform an authorization check on the person attempting to extract data from the tables.
 - a) Since this is a manual process, you need to make a note in your scenario that it needs to be included in the coding of your BAPI function modules.

Continued on next page

Task 5:

Determine the possible error messages that you need to use in your business process including a successful message.

1. Step through your business scenario and try to list all of the possible error messages that you feel are needed to let the user of the BAPI know what happened during their call.
 - a) If your BAPI is going to read data from a table and return that data, you would need an error message to notify the caller if No data was found.
 - b) If your BAPI is going to perform any updates to a table, you would need an error message to notify the caller if the 'Update was not successful'. You might want to also return a message stating that the Update was successful.
 - c) It is a good general business practice to return a message to the caller letting them know that their BAPI Call was successful.
 - d) If your BAPI is going to perform an authorization check, you would need an error message to notify the caller if the Authorization check failed.



Lesson Summary

You should now be able to:

- Describe the business scenario
- Define the business scenario
- Review the business scenario

Related Information

- Before you start to implement the BAPI, a review must be done. All persons involved in the BAPI development and those responsible for quality control should be involved in this review.

Lesson: BAPI development guidelines

Lesson Overview

Defining a BAPI and its Interface



Lesson Objectives

After completing this lesson, you will be able to:

- Define the components related to the BAPI interface
- Outline the structured components used in writing the source code of the BAPI
- Explain the important role of documentation of the BAPI

Business Example

Based on the analysis performed, we are to develop a custom Business Object with supporting key field attribute and methods. See the sample class diagram that follows :

Business Scenario to be implemented

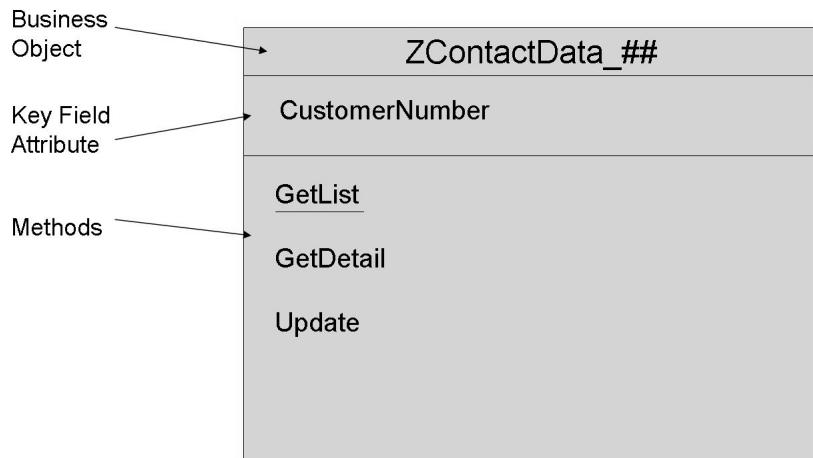


Figure 35: Exercise Scenario

BAPI signature

The parameter passing mechanism can be summarized into 3 areas :



BAPI Signature

- Key field parameter
- Data fields parameters
- Special parameters

Figure 36: BAPI parameter passing mechanism.

- The optional IMPORT **key field parameter(s)** is required if the functionality of the BAPI is specific to a single instance of the Business Object.
- The required **Data fields parameters** are various fields or structures that are specific to the BAPI functionality. They can either be part of the IMPORT, EXPORT or TABLES sections of the BAPI signature.
- The **Special parameters** are the required RETURN error handling parameter, the optional Selections parameters and the optional Extension parameters.

Key field parameter(s)

If the functionality of the BAPI is specific to an existing instance of the Business Object, the BAPI will require an IMPORT parameter for the key field. The key field can sometimes be a combination of individual fields. Examples of key fields for SAP supplied Business Objects are :

- CompanyCodeID for Business Object CompanyCode.
- Username for Business Object USER.
- SalesDocument for Business Object SalesOrder

If a key value is to be passed to the BAPI by the calling program, the key field must be set as an import parameter in the function module of the BAPI. This will make the BAPI an instance specific method in the BOR. For example, this could be the number of a sales document in the BAPI SalesOrder.GetStatus. If you are going to create your Business Object yourself, you must know what its use will be to determine the key fields that are needed and will be required by the BAPI.

If no key value is passed to the BAPI, this will make the BAPI a static method in the BOR ; in other words, instance independent. An example of this type of method is the SalesOrder.GetList.

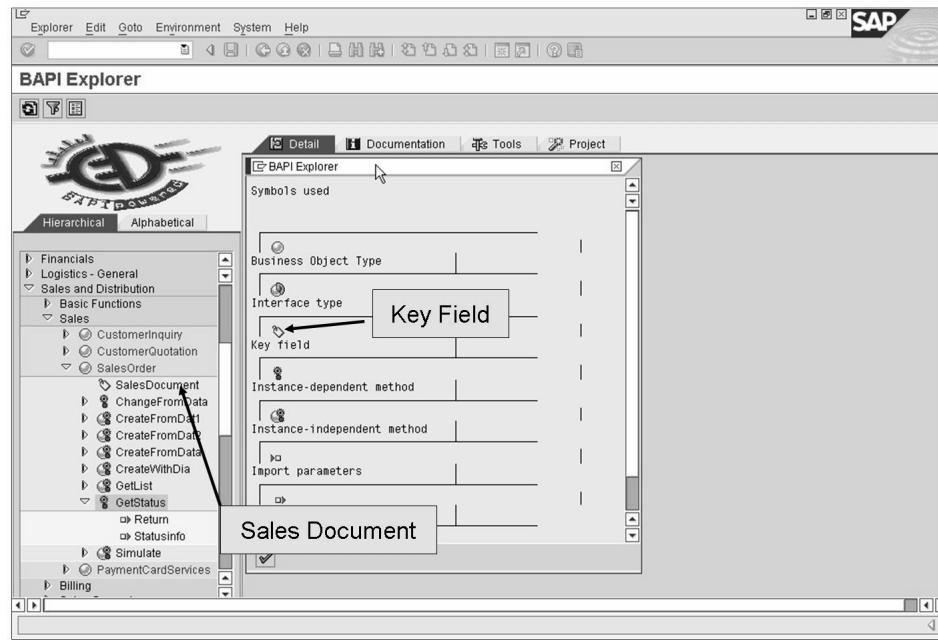


Figure 37: Business Object Key Fields

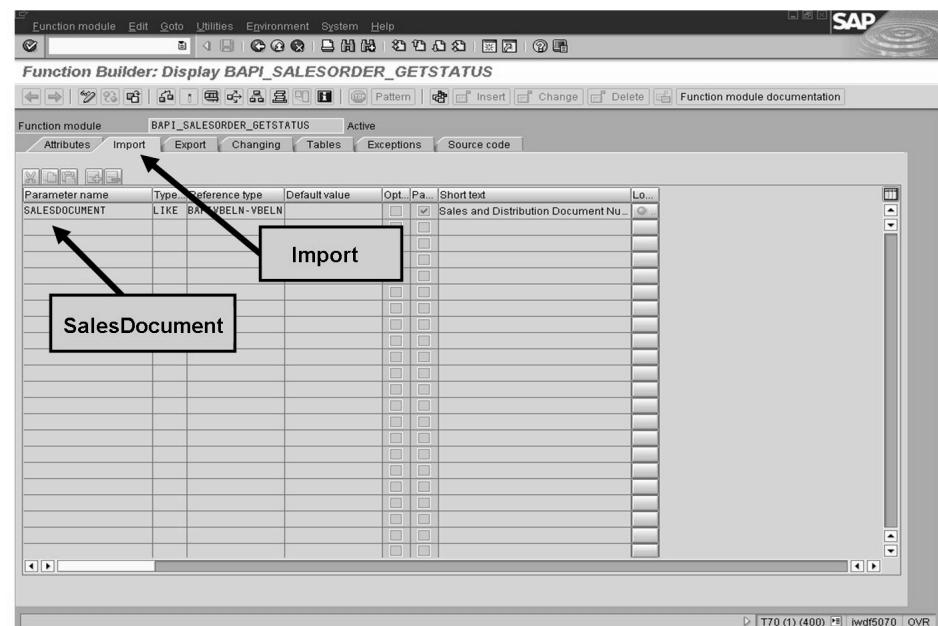


Figure 38: Import Parameter to function module as Key Field

Data parameters

The general data parameters are specific to the functionality of the BAPI. These parameters could be IMPORT, EXPORT and/or in the TABLES section of the BAPI function module signature.



Figure 39: BOR view of a BAPI data parameter

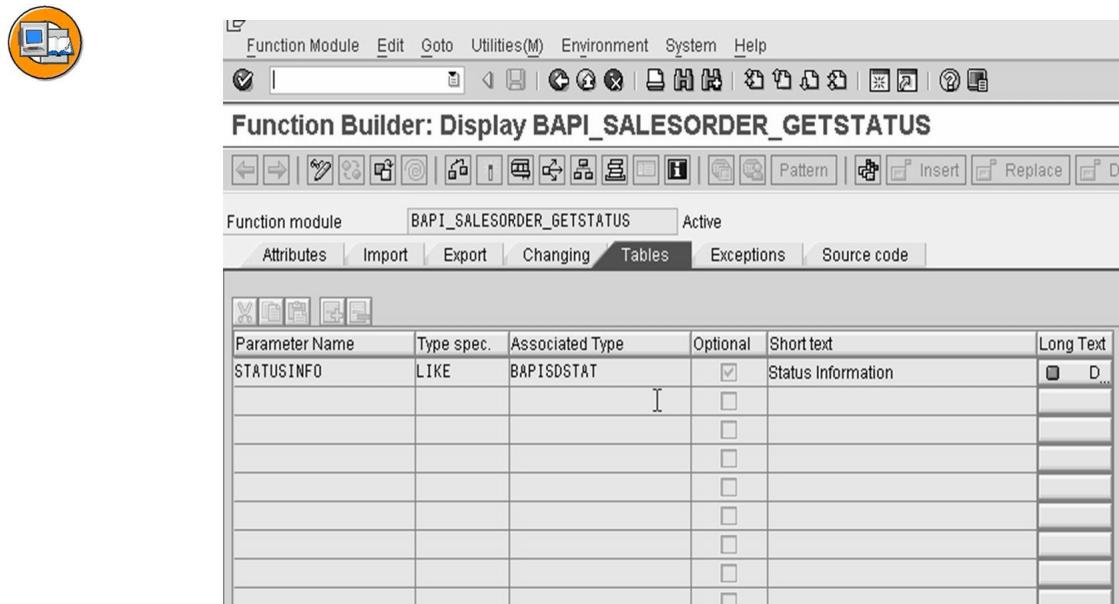
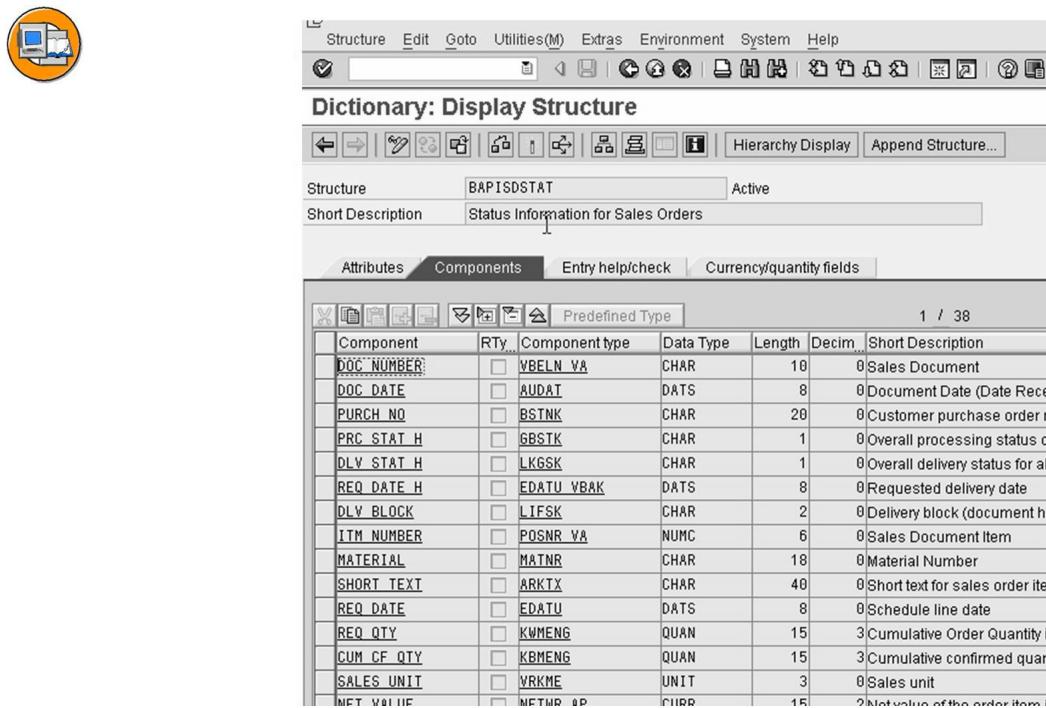


Figure 40: Function module view of the BAPI data parameter



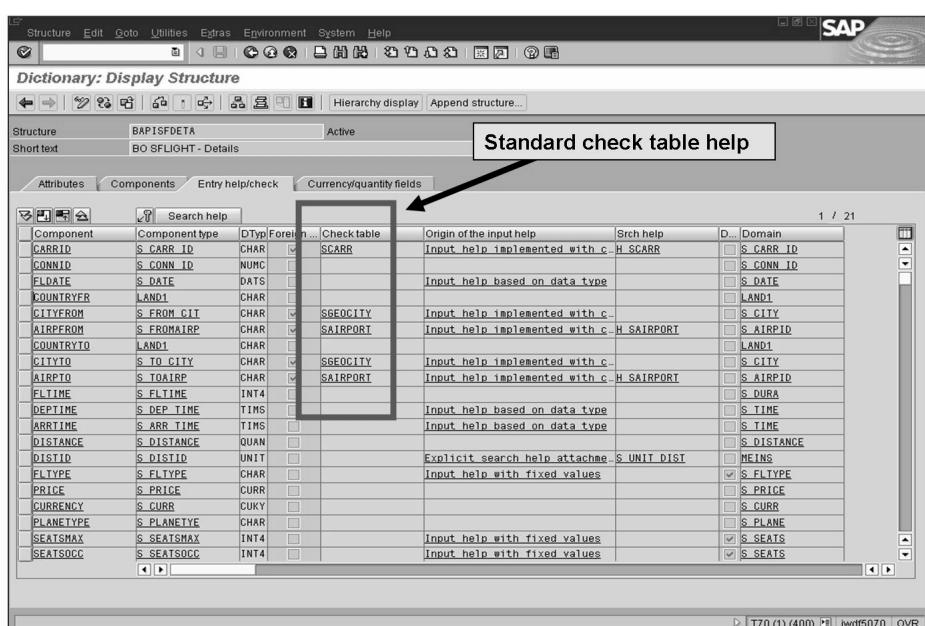
The screenshot shows the SAP Dictionary Display Structure window. The title bar reads "Dictionary: Display Structure". The menu bar includes "Structure", "Edit", "Goto", "Utilities(M)", "Extras", "Environment", "System", and "Help". The toolbar contains various icons for navigating and modifying structures. The main area displays the structure "BAPISDSTAT" which is "Active" and has a "Short Description" of "Status Information for Sales Orders". Below this, tabs for "Attributes", "Components", "Entry help/check", and "Currency/quantity fields" are visible. A large table lists the components of the structure:

Component	RTy...	Component type	Data Type	Length	Decim...	Short Description
DOC NUMBER	<input type="checkbox"/>	VBELN VA	CHAR	10	0	Sales Document
DOC DATE	<input type="checkbox"/>	AUDAT	DATS	8	0	Document Date (Date Rece
PURCH NO	<input type="checkbox"/>	BSTNK	CHAR	20	0	Customer purchase order n
PRC STAT H	<input type="checkbox"/>	GBSTK	CHAR	1	0	Overall processing status c
DLV STAT H	<input type="checkbox"/>	LKGSK	CHAR	1	0	Overall delivery status for al
REQ DATE H	<input type="checkbox"/>	EDATU VBAK	DATS	8	0	Requested delivery date
DLV_BLOCK	<input type="checkbox"/>	LIFSK	CHAR	2	0	Delivery block (document h
ITM NUMBER	<input type="checkbox"/>	POSNR VA	NUMC	6	0	Sales Document Item
MATERIAL	<input type="checkbox"/>	MATNR	CHAR	18	0	Material Number
SHORT_TEXT	<input type="checkbox"/>	ARKTX	CHAR	40	0	Short text for sales order ite
REQ_DATE	<input type="checkbox"/>	EDATU	DATS	8	0	Schedule line date
REQ_QTY	<input type="checkbox"/>	KWMENG	QUAN	15	3	Cumulative Order Quantity i
CUM CF QTY	<input type="checkbox"/>	KBMENG	QUAN	15	3	Cumulative confirmed quar
SALES_UNIT	<input type="checkbox"/>	VRKME	UNIT	3	0	Sales unit
MKT_VALUE	<input type="checkbox"/>	MKTMR SP	CURR	15	2	Net value of the order item

Figure 41: Dictionary view of the BAPI data parameter

The first step of the implementation phase is to define all the data structures (including domains and data elements) in the ABAP Dictionary. These data structures are required for the parameters of the BAPI to be implemented. You can access the ABAP Dictionary from the project form in the BAPI Explorer. The following conventions are important:

- Each parameter must refer to a data structure in the ABAP Dictionary. In the case of structured parameters this is always typed to the whole BAPI data structure. But if the parameter consists of only one field, it must refer to a field within the BAPI structure.
- You have to create separate data structures for BAPIs that are independent of the data structures generally used in the SAP system application. Reason: When the BAPI is released the underlying structures of the BAPI are frozen and restrictions apply if you want to later change the structures.
- All data structure names must begin with <namespace>BAPI.
- BAPI structure names should be as meaningful as possible.
- You must not use APPENDs and INCLUDEs in BAPI data structures. Reason: APPENDs and INCLUDEs can cause incompatibilities, if BAPI data structures are changed. This will be discussed in detail in a later lesson. You may have to define single values or a value table for the domain so that F4 help is available for use. All the useful Input Help must be defined for the data structures/data elements. To do this, a foreign key must be specified in the fields of a BAPI structure. If a value table has been defined in a field domain, a foreign key must also be defined.
- Always try to use existing central data elements and domains for fields.

The screenshot shows the SAP ABAP Dictionary 'Display Structure' window. The title bar says 'Dictionary: Display Structure'. The main area shows a table of fields with the following columns: Component, Component type, DType, Foreign key, Checktable, Origin of the input help, Srch help, and Domain. A specific row for 'SCARR' is highlighted with a red box. A callout arrow points from the text 'Standard check table help' to this row. The 'Checktable' column for 'SCARR' contains 'SCARR'. The 'Origin of the input help' column for 'SCARR' contains 'Input help implemented with c...'. The 'Domain' column for 'SCARR' contains 'S_CARR_ID'. Other rows in the table include 'S_CONN_ID' (checktable 'SCCONN'), 'S_DATE' (checktable 'SCDATE'), 'S_CITY' (checktable 'SCCITY'), 'S_AIRPORT' (checktable 'SCAIRPORT'), 'S_PLANE' (checktable 'SCPLANE'), 'S_SEATS' (checktable 'SCSEATS'), and 'S_SEATSOCC' (checktable 'SCSEATSOCC'). The table has 21 rows.

Figure 42: Example BAPI structure with applied check tables for BAPI fields

Special Parameters

There are a number of standardized parameters that provide the same or equivalent data in all BAPIs and if possible, should be implemented in the same way in all BAPIs. These are :

RETURN parameters

used for returning messages to the calling application. They provide a consistent error handling process for BAPI calls.

Extension parameters

enable application programmers to use pre-programmed user exits to enhance BAPI interfaces without modifications. These parameters are typically called ExtensionIN and ExtensionOut and are based on dictionary structure BAPIPAREX. More details on these are covered in the chapter on **Enhancement of SAP supplied BAPIs**.

Selection parameters

used to search for specific instances of a business object, for example, BAPI GetList, enables the caller of the BAPI to specify appropriate selection criteria. In general, these parameters are based on structures that comply with the Select-Options ranges table (sign, option, low, high).

Error Handling (RETURN)

When a situation occurs within the BAPI execution, the BAPI developer may not use standard function module exceptions in order to notify the user of the BAPI that an error has been detected.

The BAPI developer must return the error condition within a special EXPORT or TABLES parameter called RETURN.

The user of the BAPI must test this field in order to know if the call to the BAPI was successful or not.

Unless otherwise specified in the BAPI documentation, the BAPI user will test the first byte of the RETURN structure (typically a field called TYPE). If the value of this field is SPACE or 'S', he expects the BAPI to have accomplished the process normally. Any other value in this field indicates that the call to the BAPI has failed.

It is then the responsibility of the BAPI developer to initialize the fields of this structure appropriately.

The standard function module BALW_BAPIRETURN_GET2 can be used for this purpose.

While there are several BAPI specific return structures in the dictionary such as BAPIRETURN, BAPIRET1, and BAPIRET2, from release 4.5A, the reference structure BAPIRET1 or BAPIRET2 must be used.

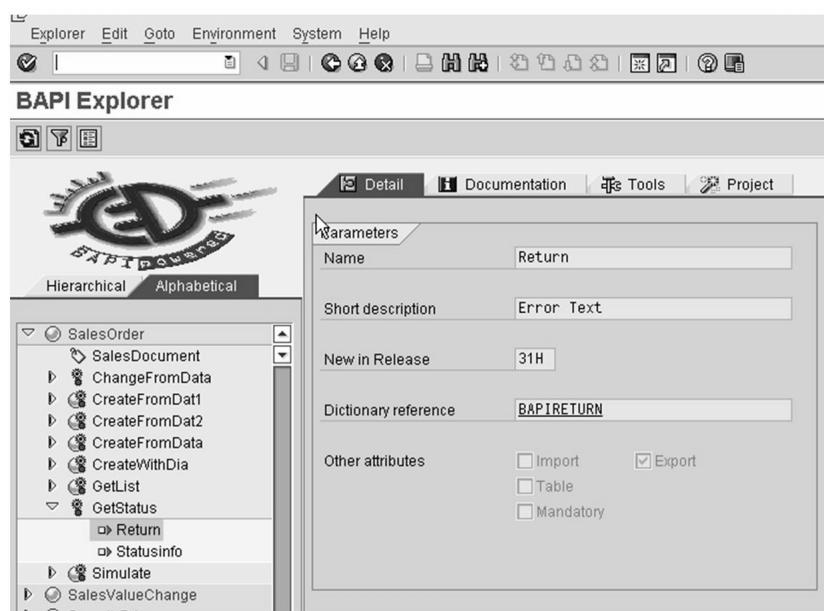


Figure 43: BOR view of the RETURN parameter

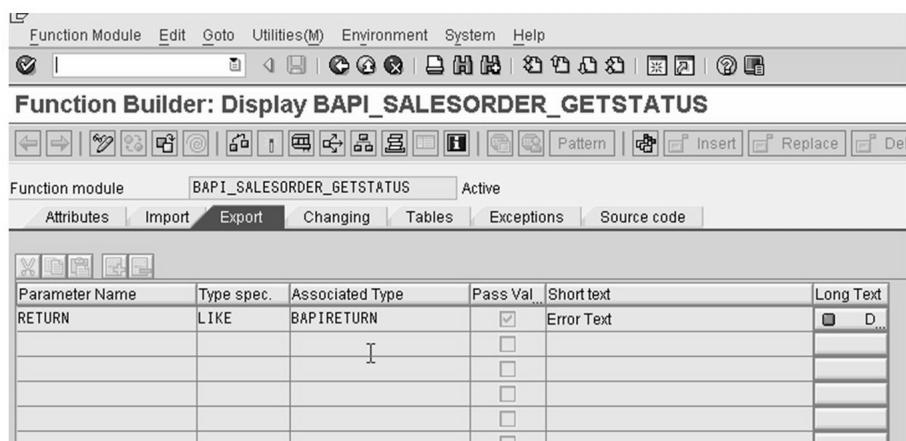


Figure 44: Function module view of the RETURN parameter

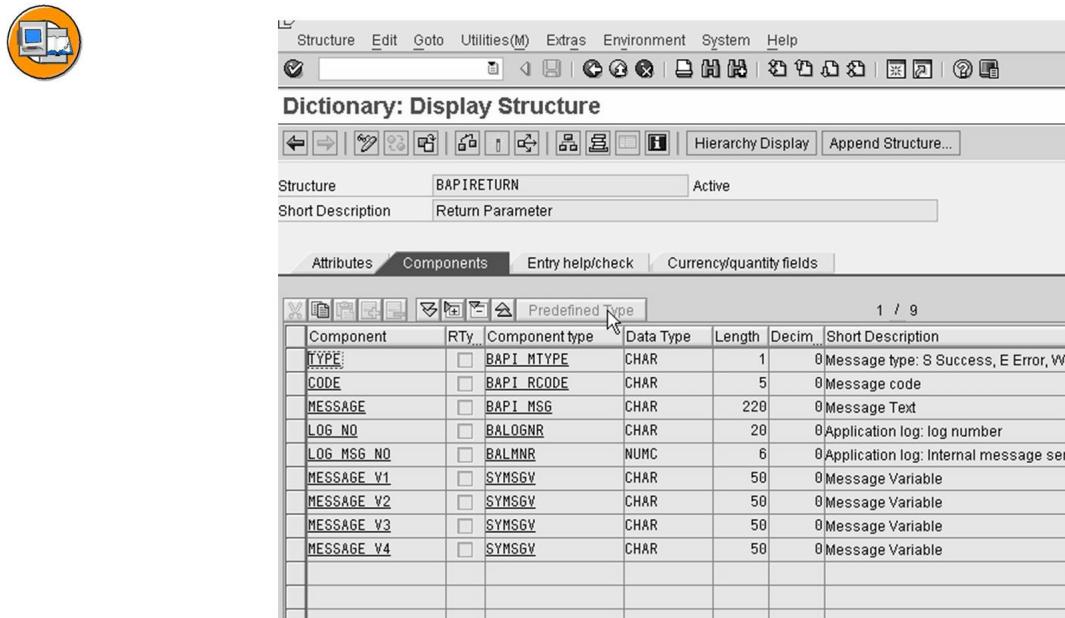


Figure 45: Dictionary view of the RETURN parameter

The common relevant fields of these structures are; *TYPE* (message type: (S)uccess, E(rror), W(arning), and I(nformation); *ID* (message class); *NUMBER* (message number); *MESSAGE* (message text); *MESSAGE_V1 — V4* (message variables).

Source Code

When preparing to write code for your BAPI, you must follow some very important guidelines. Some of the most important include:



Specifications on BAPI source code implementation

- Implemented as a function module (BAPI_<BO>_<METHOD>)
- Remote Enabled
- No EXCEPTIONS
- Must have an EXPORT or TABLES parameter called RETURN
- No COMMIT WORK
- No DIALOG
- No CALL TRANSACTION, SUBMIT REPORT
- Do not use an INCLUDE structure
- Do not use the global memory area to transfer values
- Do not use SET and GET statements
- BAPIs must carry out their own authority checks
- Integrated into BOR

Figure 46: Restrictions on BAPI source code

- The BAPI must not invoke a COMMIT WORK command. This will help to minimize the lock periods when updating the database. This will also reduce the duration of database locks.
- The BAPI must not contain any CALL TRANSACTION, SUBMIT REPORT, or SUBMIT REPORT AND RETURN commands. BAPIs must not produce any screen output.
- Minimize the use of read transactions that depend on a previous database COMMIT to avoid lost wait time from a previous update.
- Only use one function group for all of the BAPIs of the same business object.
- All parameters in the interface (fields, structures, tables) must have a reference to a BAPI data structure in the ABAP Dictionary.
- You can use importing, exporting and tables parameters. All error messages have to use the special RETURN parameter.
- Always perform database changes by using the update task.
- You should incorporate customer enhancement capabilities within your BAPI.
- Use the standard data structures BAPIRET2 for the RETURN structure and BAPIPAREX for any ExtensionIN and ExtensionOut structures.
- A BAPI should not cause a program termination (A message).
- For mapping purposes, transaction BDBS generates function modules that do the mapping between internal and external structures.
- The following two BAPIs can be used for converting currencies to and from internal and BAPI format:

`BAPI_CURRENCY_CONV_TO_EXTERNAL`

`BAPI_CURRENCY_CONV_TO_INTERNAL`



Note: these are not necessary if mapping modules were generated using BDBS.

Some of the things that can be done to improve BAPI performance include:

- Using only complete WHERE conditions to minimize the amount of data to be transferred.
- Avoiding unnecessary database access.
- Making use of arrays.
- **Not** generating programs at runtime.

Conversions for currency and quantity

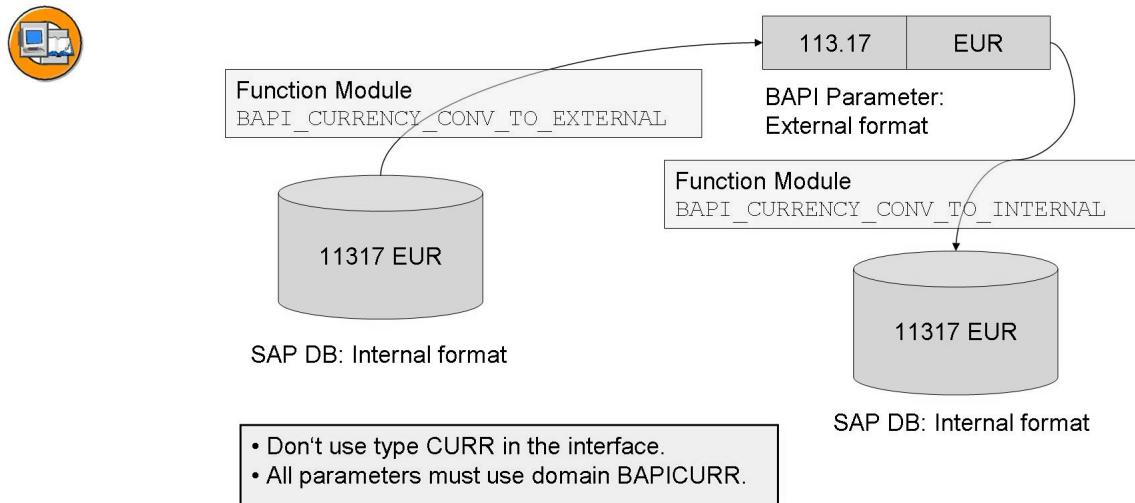


Figure 47: External format for currency amounts

The screenshot shows the SAP Dictionary: Display Table for the SFLIGHT table. The table lists various fields, including currency and quantity fields.

Field	Key	Init	Data element	Data Ty	Length	Decim	Short Description	Group
MANDI			S_MANDI	CLNT	3	0	Client	
CARRID			S_CARR_ID	CHAR	3	0	Airline Code	
CONNID			S_CONN_ID	NUMC	4	0	Flight Connection Number	
FLOATDATE			S_DATE	DATS	8	0	Flight date	
PRICE			S_PRICE	CURR	15	2	Airfare	
CURRENCY			S_CURRCODE	CUKY	5	0	Local currency of airline	
PLANETYPE			S_PLANETYPE	CHAR	10	0	Aircraft Type	
SEATSMAX			S_SEATSMAX	INT4	10	0	Maximum capacity in economy class	
SEATSOC			S_SEATSOCC	INT4	10	0	Occupied seats in economy class	
PAYOUTSUM			S_SUM	CURR	17	0	Total of current bookings	
SEATSMAX_B			S_SEATSMAX_B	INT4	10	0	Maximum capacity in business class	
SEATSOCC_B			S_SEATSOCC_B	INT4	10	0	Occupied seats in business class	
SEATSMAX_F			S_SEATSMAX_F	MAX_F	10	0	Maximum capacity in first class	
SEATSOCC_F			S_SEATSOCC_F	DCC_F	10	0	Occupied seats in first class	

Annotations highlight the 'Price' and 'Currency' fields, both of which are mapped to the 'CURRE' field (Data Type CURR) in the database.

Figure 48: Currency/Quantity Reference Fields - SAP internal format (1)

In SAP Systems, a currency amount field is only useful when a currency code accompanies it so that the decimal point in the amounts can be set correctly. A field for currency code must be assigned to each currency amount field. For example, two yen are stored as 0.02 in the field of data type CURR in the database.



The screenshot shows the SAP Dictionary: Display Table interface. The title bar says 'Dictionary: Display Table'. Below it, there are tabs: 'Transp. Table' (selected), 'SFLIGHT', 'Flight', 'Reference table', 'Reference field', and 'Currency/Quantity Fields' (which has an arrow pointing to it). The main area is a grid table with columns: Field, Data element, Data Ty..., Reference table, Ref. field, and Short Description. The grid contains 14 rows of data, such as 'MANDT' with 'S_MANDT' as the data element and 'CLNT' as the reference table.

Field	Data element	Data Ty...	Reference table	Ref. field	Short Description
MANDT	S_MANDT	CLNT			Client
CARRID	S_CARR_ID	CHAR			Airline Code
CONNID	S_CONN_ID	NUMC			Flight Connection Number
FDATE	S_DATE	DATS			Flight date
PRICE	S_PRICE	CURR	SFLIGHT	CURRENCY	Airfare
CURRENCY	S_CURRCODE	CUKY			Local currency of airline
PLANETYPE	S_PLANETYPE	CHAR			Aircraft Type
SEATSMAX	S_SEATSMAX	INT4			Maximum capacity in economy class
SEATSOCC	S_SEATSOCC	INT4			Occupied seats in economy class
PAYMENTSUM	S_SUM	CURR	SFLIGHT	CURRENCY	Total of current bookings
SEATSMAX_B	S_SMAX_B	INT4			Maximum capacity in business class
SEATSOCC_B	S_SOCC_B	INT4			Occupied seats in business class
SEATSMAX_F	S_SMAX_F	INT4			Maximum capacity in first class
SEATSOCC_F	S_SOCC_F	INT4			Occupied seats in first class

Figure 49: Currency/Quantity Reference Fields - SAP internal format (2)

When you use currency amount fields in a BAPI, adhere to the following guidelines:

- You must not use parameters and fields of data type CURR in the interface.
- All parameters and fields for currency amounts must use the domain BAPICURR.
- The position of the decimal point in currency amount fields must be converted correctly. For example, the value 10.12 must be formatted as 10.12 and not as 1012.

You can use two function modules for this conversion. The function module BAPI_CURRENCY_CONV_TO_EXTERNAL converts currency amounts from internal data formats into external data formats. The function module BAPI_CURRENCY_CONV_TO_INTERNAL converts currency amounts from external data formats into internal data formats.

Quantity fields must also be converted. A unit of measure field must be assigned to every quantity field. Quantity fields have no general domain that can be used for BAPI structures.

Documentation

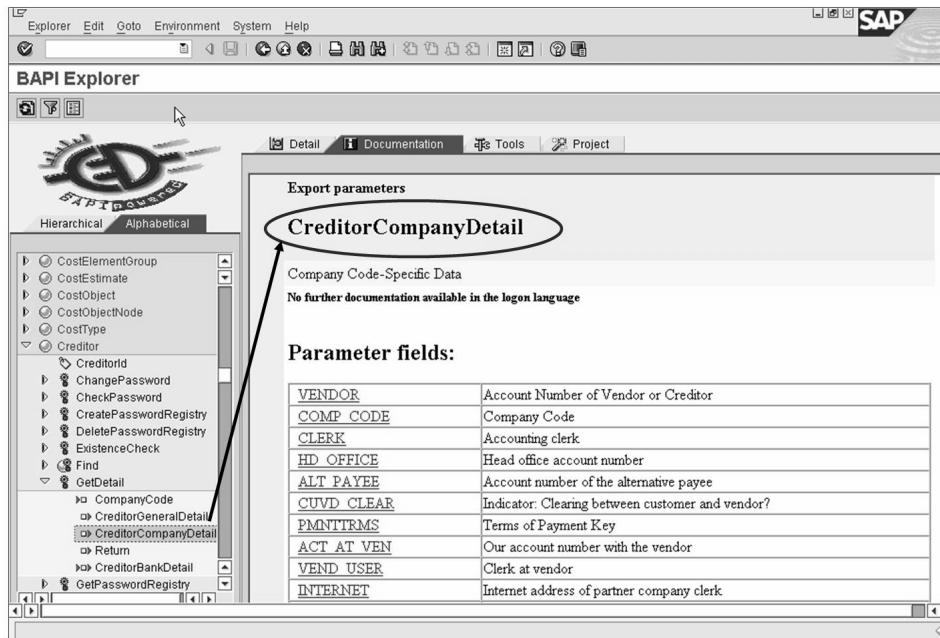


Figure 50: BAPI Explorer Documentation

BAPIs are business interfaces to the SAP System. Users are not required to have a thorough knowledge of the SAP System and the development environment. The documentation must be in sufficient detail so that an external developer familiar with the business background but not with the SAP System can use the BAPI.

The documentation for a BAPI covers **four** areas:

1. Business Object-type Documentation

The BAPI Explorer contains very detailed documentation on every item of the Business Object including fields, method, structures, etc. If you double-click on an item in the left window, and you select the *Documentation* tab in the right window, you will get detailed information about that object.

The documentation associated for the Business Object itself is maintained in the BOR, via SWO1.

2. Method Documentation

Because the Method of a business object is the Function Module, you write this documentation in the Function Builder. The purpose of this documentation is to describe what the method can do exactly. The documentation should help customers decide whether a method can perform the task in question and it should answer the following questions:

- What is the **business** function of the BAPI and what can it be used for?
- What exactly are the functions of the BAPI?
- Are there any important limitations, that is, functions that the BAPI cannot perform?
- What must you pay particular attention to with this BAPI? For example, is special authorization required to use this BAPI?
- What dependencies are there between this BAPI and other BAPIs, and between individual method parameters?
- Does this BAPI contain a COMMIT WORK command?

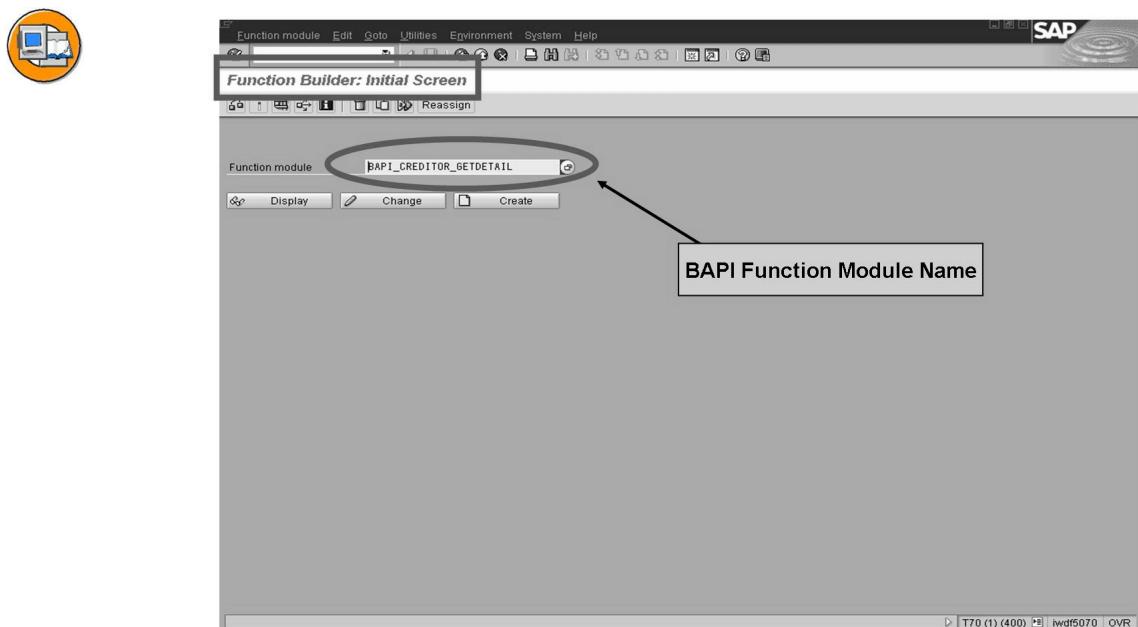


Figure 51: Function Builder (1)

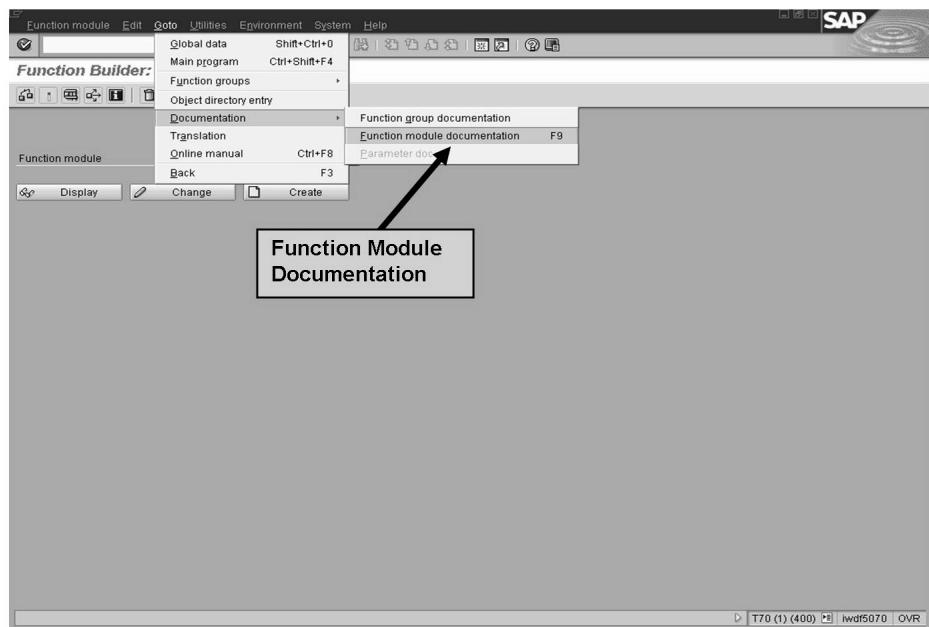


Figure 52: Function Builder (2)

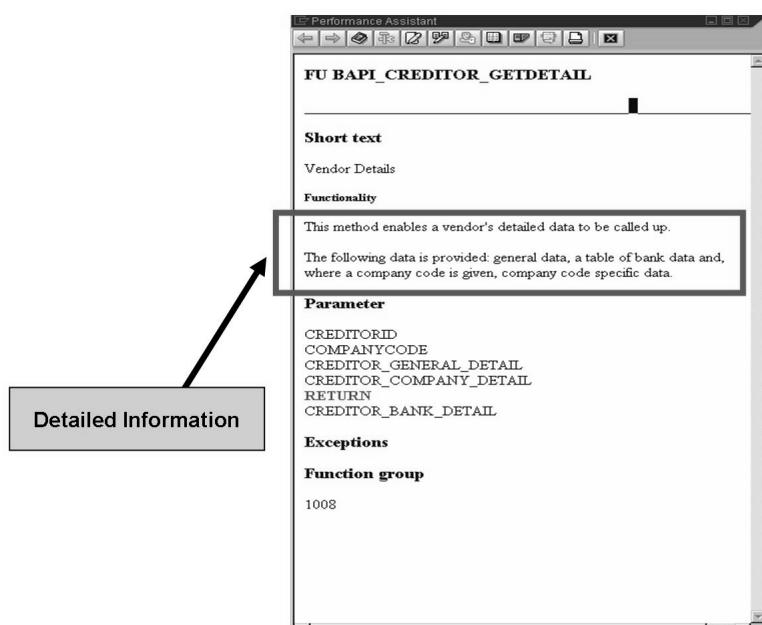


Figure 53: Function Module Documentation

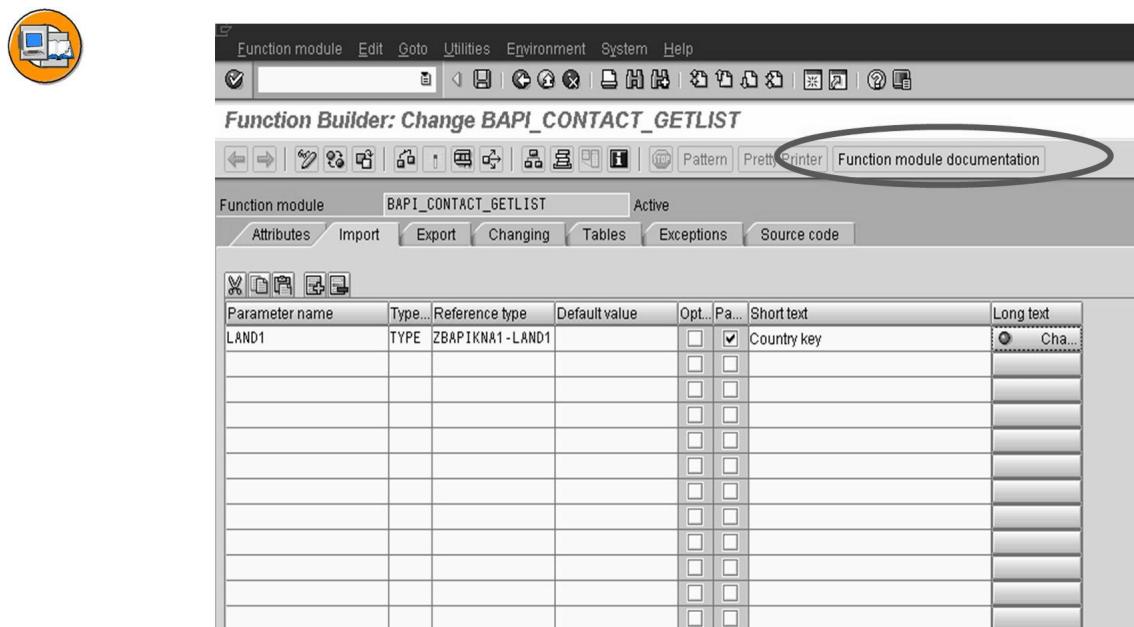


Figure 54: Maintaining the Function Module Documentation (1)

You need to supply the documentation for the function module that implements the BAPI. Some of the documentation gets inserted automatically, for instance the *Short text* from the *Attribute* tab, the names of all of the parameters and the function group. Additional information needs to be supplied such as the purpose of the function module and any other important information that you need to provide to the user.

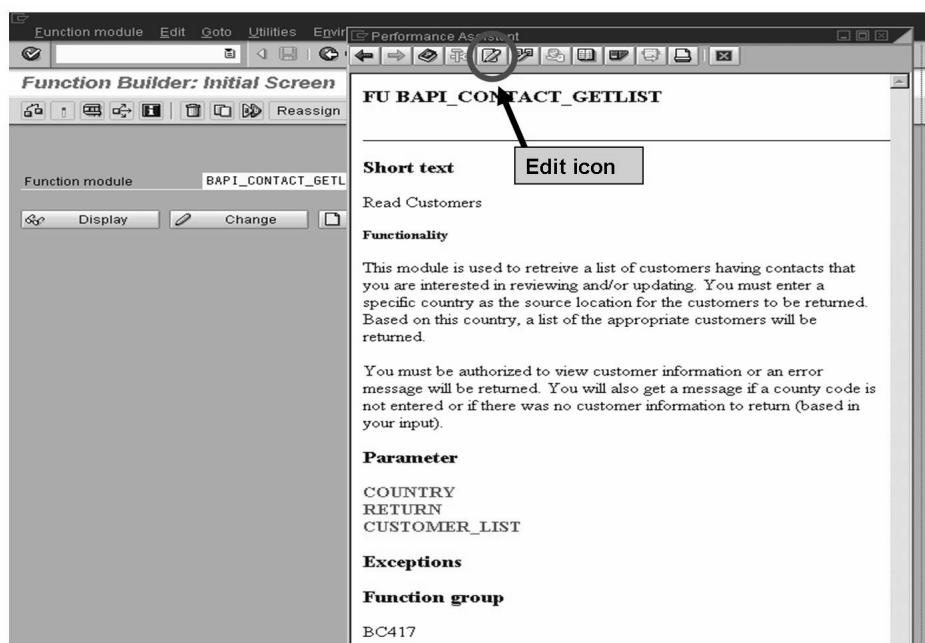


Figure 55: Maintaining the function Module Documentation (2)

To add additional documentation, select the Edit button and add your documentation.

Use tags to create your documentation. You can add your own tags by selecting the *Green light text* pushbutton, which will take you to the documentation screen. Choose the *Insert command* icon and you will get a pop-up window to insert your own tag. In the *Symbols* box, enter the name of your tag inside a pair of ampersands (&). Once you save your tag, put your cursor on it and make sure it displays U4 Header in Module in the *Parag.Formats* field. If not, change the tag to this format so the tag appears as a header.

3. BAPI Parameter Documentation

Documentation on the individual parameters is written in the Function Builder in the associated function module. Parameter documentation should provide answers to the following questions:

- What is the parameter used for?
- What are the **mandatory** fields?
- What are the default values? If there are any fixed values, what do they do?
- Which return codes may be returned to the calling program directly or indirectly in the Return parameter?
- If there is a termination, is a database rollback executed as an exception within the BAPI? If it is, you must describe this process in the documentation for the return parameter.
- Are all of the available BAPI table extensions listed in the documentation on the extension parameters (ExtensionIn, ExtensionOut)?

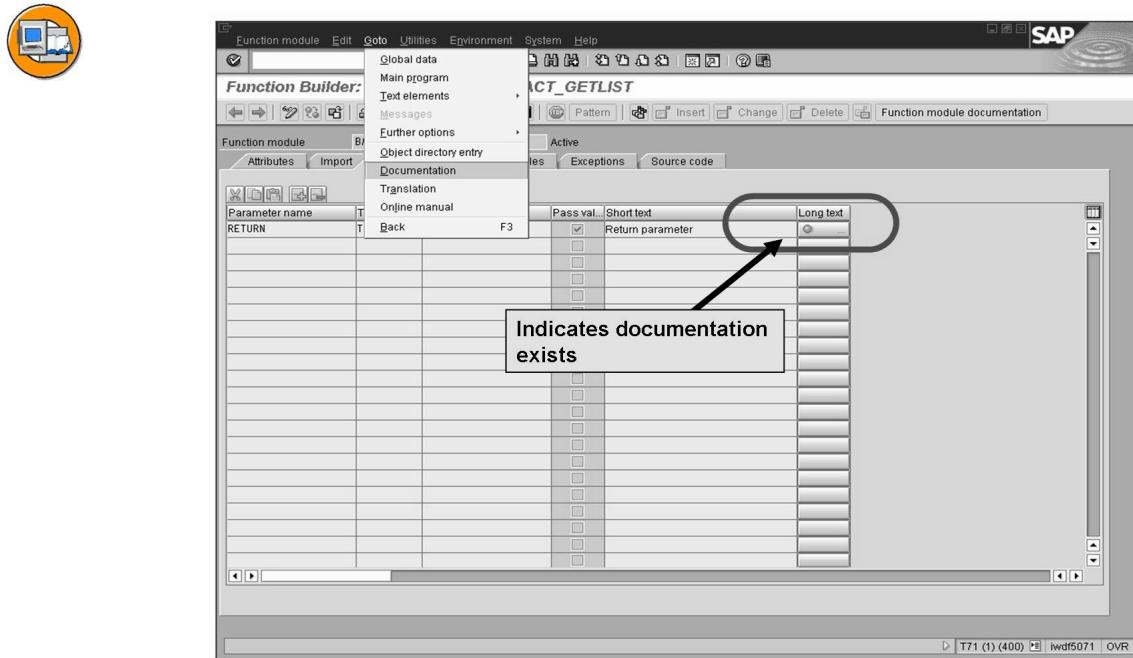


Figure 56: Parameter Documentation (1)

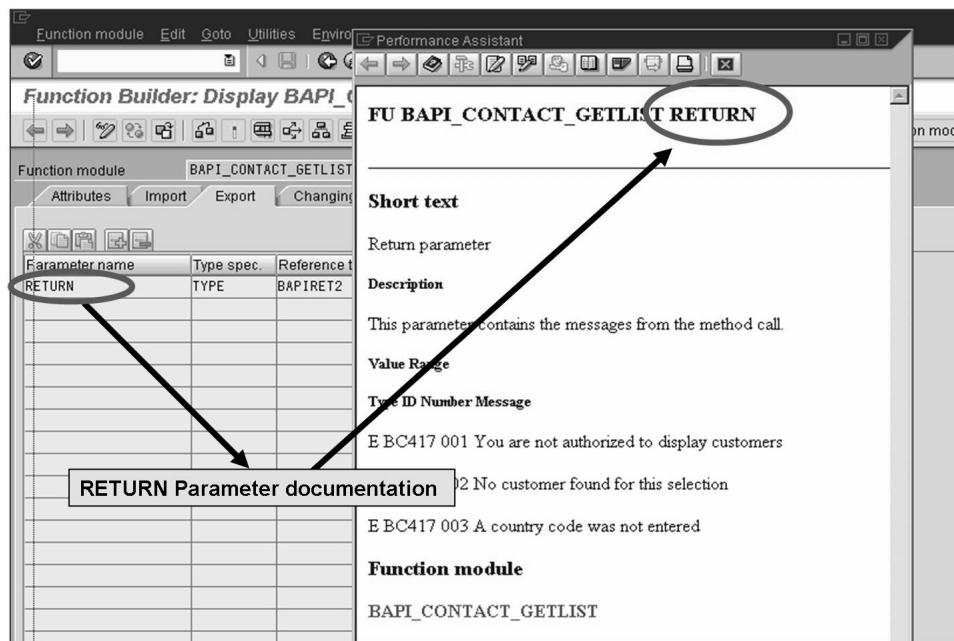


Figure 57: Parameter Documentation (2)

For each parameter, you should provide meaningful documentation about the parameter being used and especially the RETURN parameter. Each parameter has a *Long text* field where you put in the specific documentation about that particular parameter. The documentation should be as meaningful as possible. For the RETURN parameter, you should supply all of the known possible error messages that could be passed to the calling program.

4. Data Element Documentation

A calling program can call data element documentation (F1 help) for a field in a BAPI parameter using the Service BAPI_BapiService.FieldHelpGetDocu. For this reason, you have to write data element documentation for each field in BAPI parameters. Data element documentation is done in the Data Dictionary.

- **Note:** make sure that the function module and parameter documentation has been saved in the active version so that it appears in the translator's work list.

Exercise 5: Create the Get List function module

Exercise Objectives

After completing this exercise, you will be able to:

- Create the Get List function module and its associated ABAP Dictionary structure.

Business Example

To call a Get List BAPI, you need to have the function module that reads the data from the necessary table. In addition, you also need the BAPI structure in the ABAP Dictionary.

Task:

The learner will create a Package, a Dictionary structure, and an RFC-enabled function module including documentation of the parameters and function module.

1. Create a Package **ZBC417_##** for your group.
2. Create the ABAP Dictionary structure **ZBAPISKNA1_##** from the *SKNA1* table using the fields; *KUNNR*, *LAND1*, *NAME1*, *NAME2*, *ORT01*, *PSTLZ*, *REGIO*, and *STRAS*. Include the Component Types as well.

Add the Check Table relationships for the fields *LAND1* and *REGIO*. Hint: you can get this information from the *SKNA1* table. Save and activate your structure.

3. Create a function group **ZBC417_##**, where ## is your group number.
4. Create the **Z_BAPI_CONTACT_GETLIST_##** function module. The function module should be a type Remote Enabled which is set in the Attributes tab. The user needs to enter the desired country as an import parameter. The BAPI requires a table structure for the data list for the calling program. Since the data list is a customer list, it is recommended to use a table parameter called **CUSTOMER_LIST**. This will need to be assigned to your Dictionary structure **ZBAPISKNA1_##**. The RETURN table structure will be assigned to the standard **BAPIRET2** Dictionary structure.

You will use the following structure for passing messages. Data: begin of message, msgty like sy-msgty, msgid like sy-msgid, msgno like sy-msgno, msgv1 like sy-msgv1, msgv2 like sy-msgv2, msgv3 like sy-msgv3, msgv4 like sy-msgv4, end of message.

Continued on next page

The BAPI should read a list of data from the *SKNA1* table and process an error number 002 from message class BC417 if no data is found. Populate the fields *Msgty*, *Msgid*, *Msgno* from the message structure you defined in your global data. Call the **BALW_BAPIRETURN_GET2** function module in a subroutine.

Perform an authorization check using the object **W_AUFT_RMB** to see if the user has Display authority and if not, process error message 001 from message class BC417.

Before you read the table, verify that an entry was made into the Country import parameter and if it was, translate the value to an uppercase value. If no entry was made, process error message 003 from message class BC417.

5. Save, activate, and release your function module.

Solution 5: Create the Get List function module

Task:

The learner will create a Package, a Dictionary structure, and an RFC-enabled function module including documentation of the parameters and function module.

1. Create a Package **ZBC417_##** for your group.
 - a) Execute transaction **SE80**.
 - b) In the entry field in the left screen, set the box to Package.
 - c) Enter the name of your Package as **ZBC417_##** and select the *Display* icon (eyeglasses).
 - d) You will get a pop-up window that says Does not exist, do you want to create it?. Select **YES**.
 - e) On the create Package window, enter a short description for your package (such as 'BC417_## package').
 - f) In the *Application Component* field, select 'Cross-Application' or 'CA'. This step is important since this will be the area under which your Business Objects will appear via the BAPI Explorer.
 - g) Select the *Save* icon.
 - h) At the change request dialog box, select *Own request*. Create a new change request if one is not already available.
 - i) Locate and select your team **ID task number** and select *Choose*.
 - j) Back on the Change request dialog screen, select continue.
2. Create the ABAP Dictionary structure **ZBAPISKNA1##** from the SKNA1 table using the fields; *KUNNR*, *LAND1*, *NAME1*, *NAME2*, *ORT01*, *PSTLZ*, *REGIO*, and *STRAS*. Include the Component Types as well.

Continued on next page

Add the Check Table relationships for the fields *LAND1* and *REGIO*. Hint: you can get this information from the *SKNA1* table. Save and activate your structure.

- a) Go to the ABAP Dictionary (SE11) and select the *Data Type* radio button.
 - b) Type in the structure name **ZBAPISKNA1_##** and select *Create* pushbutton.
 - c) Type in the appropriate description for your structure in the *Short Text* field.
 - d) Enter the Component and the Component Type names. Hint: these can be copied directly from the *SKNA1* table. Select *Enter* when finished. This fills in the rest of the required field information.
 - e) Select the *Entry help/check* tab. Add a foreign key relationship to the *LAND1* and *REGIO* fields. Use the same check table relationship listed in the *SKNA1* table.
 - f) Save and activate your structure.
3. Create a function group **ZBC417_##**, where ## is your group number.
 - a) From the SAP Menu, select *Tools* → *ABAP Workbench* → *Development* → *Function Builder* (SE37). From the menu of the main function builder screen, select *GoTo* → *Function groups* → *Create group*.
 - b) In the *Function Group* field, type in the function group name **ZBC417_##** (example ZBC417_00).
 - c) Enter a short text description for your function group and then *Save*.
 - d) In the pop-up window, select your *Package*.
 4. Create the **Z_BAPI_CONTACT_GETLIST##** function module. The function module should be a type Remote Enabled which is set in the Attributes tab. The user needs to enter the desired country as an import parameter. The BAPI requires a table structure for the data list for the calling program. Since the data list is a customer list, it is recommended to use a table parameter called *CUSTOMER_LIST*. This will need to be assigned to your Dictionary structure **ZBAPISKNA1_##**. The RETURN table structure will be assigned to the standard BAPIRET2 Dictionary structure.

You will use the following structure for passing messages. Data: begin of message, msgty like sy-msgty, msgid like sy-msgid, msgno like sy-msgno, msgv1 like sy-msgv1, msgv2 like sy-msgv2, msgv3 like sy-msgv3, msgv4 like sy-msgv4, end of message.

Continued on next page

The BAPI should read a list of data from the *SKNA1* table and process an error number 002 from message class BC417 if no data is found. Populate the fields *Msgty*, *Msgid*, *Msgno* from the message structure you defined in your global data. Call the **BALW_BAPIRETURN_GET2** function module in a subroutine.

Perform an authorization check using the object **W_AUFT_RMB** to see if the user has Display authority and if not, process error message 001 from message class BC417.

Before you read the table, verify that an entry was made into the Country import parameter and if it was, translate the value to an uppercase value. If no entry was made, process error message 003 from message class BC417.

- a) On the main Function Builder screen, enter the name of your BAPI as **Z_BAPI_CONTACT_GETLIST_##**, where ## is your group number. Select *Create* and assign the function module to your Package **ZBC417_##**. Save your work.
- b) Go to the *Attributes* tab and select *Remote-enabled module*.
- c) Go to the *Import* tab and set up an import parameter for Country and an additional import parameter for the associated ISO Code.
- d) Go to the *Export* tab and set up an export parameter for the return structure.
- e) Go to the *Tables* tab and set up a table parameter for the customer list. Use the statement **LIKE** as your reference type and put in the name of your referenced Dictionary structure.



Note: when you go to the *Source code* tab, you will see the Local Interface documentation for all of your parameters.

- f) You will need some Global Data for your function module. On the menu, choice *Goto → Global* data to get to the Top INCLUDE.
- g) Add a **Message-ID** statement for the message class BC417.
- h) Add a TABLES statement for the *SKNA1* table.
- i) Add a data statement to make your customer list structure a table like your Dictionary structure.
- j) Add the Data statements for your message structure as follows:

```
Data: begin of message, msgty like sy-msgty,
      msgid like sy-msgid, msgno like sy-msgno, msgv1
      like sy-msgv1, msgv2 like sy-msgv2, msgv3 like
      sy-msgv3, msgv4 like sy-msgv4, end of message.
```

Continued on next page

- k) In the *Source code* tab, put statements to CLEAR all of your table declarations.
 - l) Add a statement to perform an authorization check using object **W_AUFT_RMB**.
 - m) Check the return code from the authority check and process error number 001 from message class BC417.
 - n) Add a statement to translate the Country entry to upper case in the event that the user enters it as a lower case value.
 - o) Add a SELECT statement to retrieve the corresponding fields for your structure from the *SKNA1* table.
 - p) Check the return code from the SELECT and process the error number 002 if no records are found and error number 003 if no country code was entered.
 - q) Save and activate the function module.

“For a sample code excerpt, see function module **BAPI_CONTACT_GETLIST**.”
5. Save, activate, and release your function module.
- a) Select the *Save* icon to save your work.
 - b) Select the *Activate* icon (Matchstick). Do **Select All** to activate all the components at one time.
 - c) On the Function Builder Initial Screen, on the menu, choose the *Function Module → Release → Release*. You should receive a message that your function module was successfully released.
-  **Note:** If the function module has not been released, you will later not be able to attach it as a BAPI method to a business object.

Exercise 6: Create a Get Detail function module

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Get Detail RFC enabled function module and its DDIC structure.
- Append the Get Detail function module to your Business Object.

Business Example

You have called a BAPI and have received Customer information. You now need to take a specific customer's information and make a call to another BAPI to get the contacts for that customer.

Task 1:

The learner will create a Dictionary structure and an RFC-enabled function module including documentation of the parameters and function module.

1. Create the ABAP Dictionary structure ZBAPISKNVK_## from the SKNVK table using the fields; KUNNR, NAMEV, NAME1, ADRND, TELF1, ERNAM and PARNR. Include the Component Types as well.

Save and activate your structure.

Task 2:

Create a GETDETAIL function module.

1. Create the Z_BAPI_CONTACT_GETDETAIL_## function module. The BAPI should read a list of contacts data from the SKNVK table for the specific customer entered and process an error if no data is found. Create an import parameter for the customer number. The RETURN structure will be assigned to the standard BAPIRET2 Dictionary structure as an export parameter. The BAPI requires a table structure for the data list for the calling program. Since the data list is a contacts list, it is recommended to use the name CONTACT_DETAIL. This needs to be assigned to your Dictionary structure ZBAPISKNVK_##.

Once your function module has been tested and released, append it as an API method to your business object.

Solution 6: Create a Get Detail function module

Task 1:

The learner will create a Dictionary structure and an RFC-enabled function module including documentation of the parameters and function module.

1. Create the ABAP Dictionary structure ZBAPISKNVK_## from the SKNVK table using the fields; KUNNR, NAMEV, NAME1, ADRND, TELF1, ERNAM and PARRN. Include the Component Types as well.

Save and activate your structure.

- a) Go to the ABAP Dictionary (SE11) and select the Data Type radio button.
- b) Type in the structure name **ZBAPISKNVK_##** and select *Create*. Select the *Structure* radio button and select *Continue*.
- c) Type in the appropriate description for your structure in the Short Text field.
- d) Enter the Component and the Component Type names. Hint: these can be copied directly from the SKNVK table. Select *Enter* when finished. This fills in the rest of the required field information.
- e) *Select the Entry help/check tab.* add a foreign key relationship to the LAND1 and REGIO fields. Use the same check table relationship listed in the sknvk table
- f) *Save* and activate your structure.

Task 2:

Create a GETDETAIL function module.

1. Create the Z_BAPI_CONTACT_GETDETAIL_## function module. The BAPI should read a list of contacts data from the SKNVK table for the specific customer entered and process an error if no data is found. Create an import parameter for the customer number. The RETURN structure will be assigned to the standard BAPIRET2 Dictionary structure as an export parameter. The BAPI requires a table structure for the data list for the calling program. Since the data list is a contacts list, it is recommended to use the name CONTACT_DETAIL. This needs to be assigned to your Dictionary structure ZBAPISKNVK_##.

Continued on next page

Once your function module has been tested and released, append it as an API method to your business object.

- a) On the main Function Builder screen, enter the name of your BAPI as **Z_BAPI_CONTACT_GETDETAIL_##**, where ## is your group number. Choose the *Create* icon and assign the function module to your Package **ZBC417##**.
- b) Go to the *Attributes* tab and select the *Remote-enabled module* radio button.
- c) Go to the *Import* tab and set up an import parameter for Customer and check the pass value.
- d) Go to the *Export* tab and set up an export parameter for the return structure and check the pass value.
- e) Go to the *Tables* tab and set up a table parameter for the contacts list. Use the statement LIKE as your reference type and put in the name of your referenced Dictionary structure. Since this is for contact information, you should use the name **Contact_detail** for your table.
 **Note:** you will need to add a Tables statement to your Global Data area for the SKNVK table.
- f) In the *Source code* tab, put statements to CLEAR all of your table declarations.
- g) Add a SELECT statement to retrieve the corresponding fields for your structure from the SKNVK table based on the value in the Customer entry parameter.
- h) Check the return code from the SELECT. If the return code is not zero, concatenate an error message that includes the customer number that was entered. Example: Customer ### does not exist.
- i) Save and activate the function module.
“For a sample code excerpt, see function module
BAPI_CONTACT_GETDETAIL”
- j) Test your function module in the same manner you did for your GetList function module.
- k) On the Function Builder Initial Screen, on the menu, choose the *Function Module → Release → Release*. You should receive a message that your function module was successfully released.
 **Note:** If the function module has not been released, you will not be able to later attach it as a BAPI method to a business object.

Exercise 7: Document the GetList and GetDetail function modules

Exercise Objectives

After completing this exercise, you will be able to:

- Create documentation for your function module and its parameters

Business Example

Someone who is using your function module needs to know as much about its use as possible. You need to add documentation about each of the parameters and the overall function module.

Task:

Document the parameters and function module.

1. Add documentation to the parameters and the function module.

Solution 7: Document the GetList and GetDetail function modules

Task:

Document the parameters and function module.

1. Add documentation to the parameters and the function module.
 - a) In SE37, put in the name of your function module, (example Z_BAPI_CONTACT_GETLIST_##). Hit the change button. Select the Import tab strip, select the *Long Text* push button to the right of the parameter you are going to document. This takes you to the documentation screen.
 - b) Under the header Meaning, enter in the purpose of the parameter. Add other tags and information that you feel is necessary.
 - c) If you need to add any additional tags, choose *Insert command* icon to get the symbols window. Enter your new symbol in a pair of ampersands.
 **Note:** make sure that the new symbol shows up as a U4 Header in the format window.
 - d) Repeat process for all parameters (import, export, changing, tables).
 - e) Select the *Function module documentation* push button on the function module main screen or from the Function Builder initial screen, choose menu path *Goto → Documentation → Function module documentation*.
 - f) Enter your documentation and any new tags the same way as for a parameter.
 **Note:** if you enter the function module documentation from the Function Builder initial screen, select *Edit* to edit the documentation.
 - g) Repeat parameter and function module documentation steps for the second function module, ie Z_BAPI_CONTACT_GETDETAIL_##.



Lesson Summary

You should now be able to:

- Define the components related to the BAPI interface
- Outline the structured components used in writing the source code of the BAPI
- Explain the important role of documentation of the BAPI

Related Information

For additional information on developing BAPIs, refer to the BAPI Programming Guide on SAPNet.

Lesson: Testing an RFC enabled Function Module

Lesson Overview

Discuss the processes that should take place in the testing phase of BAPI development.



Lesson Objectives

After completing this lesson, you will be able to:

- Test the documentation for accuracy and completeness
- Test an RFC-enabled function module

Business Example

You want to use a BAPI as part of your business scenario. Before it can be used, the BAPI has to be fully tested and fully documented for accuracy.

Testing the Documentation

Because the BAPI documentation is essential to using the BAPI, this test should be thoroughly carried out.

Testing the BAPI in Function Builder



The screenshot shows the SAP Function Builder interface with the title 'Test Function Module: Initial Screen'. The menu bar includes 'Function modules', 'Edit', 'Goto', 'Utilities', 'System', and 'Help'. The toolbar contains various icons for file operations like Open, Save, Print, and Help. The main area displays the following information:

- Test for function group: ZBC417_00
- Function module: BAPI_CONTACT_GETLIST
- Upper/lower case:
- Import parameters: LAND1
- Value: del
- No conversion field: No conversion field
- Tables: SCUSTLIST
- Value: 0 Entries

A callout box points to the 'No conversion field' label, and another points to the 'del' entry in the Value column for the LAND1 parameter.

Figure 58: Error Message Testing

You can test the parameters in your function module in one test by entering the appropriate test values in the parameters to verify that the source code in the function module can run without errors. In addition, you should also test to make sure that your error messages work.

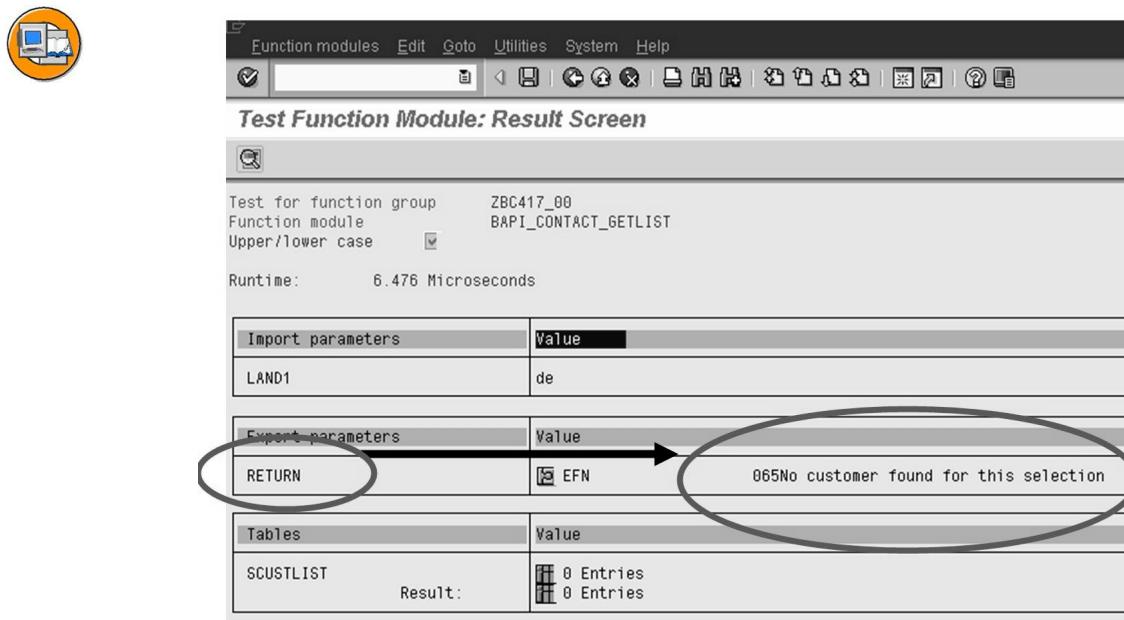


Figure 59: Error Message Results

If you find any errors in the BAPI implementation, correct them and repeat the tests until you and the quality control team in your group are fully satisfied with the BAPI implementation and with the documentation.

Exercise 8: Test the RFC-enabled function module

Exercise Objectives

After completing this exercise, you will be able to:

- Completely test the function module for your BAPI including the documentation.

Business Example

You are preparing to use a function module as part of your business scenario. Before the module can be used, it has to be fully tested.

Task 1:

Test the function module and its parameters.

1. Test the Z_BAPI_CONTACT_GETLIST_## function module.

Task 2:

Test the RETURN parameters error messages.

1. Test the error messages of the function module

Task 3:

Test the documentation of the function module and its parameters.

1. Test the documentation.

Task 4:

Test the Z_BAPI_CONTACT_GETDETAIL_## function module.

1. Repeat steps as you did for your Z_BAPI_CONTACT_GETLIST_## function module.

Solution 8: Test the RFC-enabled function module

Task 1:

Test the function module and its parameters.

1. Test the Z_BAPI_CONTACT_GETLIST_## function module.
 - a) In SE37, enter the name of the function module to test, Z_BAPI_CONTACT_GETLIST_##. Select the *Test / Execute(F8)* icon to get to the test function module initial screen.
 - b) In the *Country Value* field, enter the value **US** or **DE** in either lower case or upper case letters. If entered in lower case, there is a conversion that takes place to upper case via your Translate statement.
 - c) Select the *Execute* icon (F8).
 - d) Select the *CustomerList Results table* icon to view your data results.

Task 2:

Test the RETURN parameters error messages.

1. Test the error messages of the function module
 - a) In the source code of your function module, comment out the Translate statement and activate the change.
 - b) Select the *Execute* icon (F8).
 - c) Place a check mark in the *Upper/lower case* box.
 - d) In the *Value* field, enter in lower case letters **us** or **de**.
 **Note:** the check box in the previous step prevents the value from being converted to upper case by the system.
 - e) Select the *Execute* icon (F8).
 - f) Review the error message in the RETURN parameter for accuracy.

Continued on next page

Task 3:

Test the documentation of the function module and its parameters.

1. Test the documentation.
 - a) On the menu in the main Function Builder screen, choose *Goto → Documentation → Function module documentation*.
 - b) In the popup window, review the documentation under each tag for accuracy and clarity.
 - c) In Display mode, bring up your function module main screen.
 - d) In the Import tab strip, go to your first parameter and select the *Green* icon under the *Long Text* column.
 - e) Review the documentation under each tag for accuracy and clarity.
 - f) Repeat this process for all of the parameters in your function module.

Task 4:

Test the Z_BAPI_CONTACT_GETDETAIL_## function module.

1. Repeat steps as you did for your Z_BAPI_CONTACT_GETLIST_## function module.
 - a) no solution



Lesson Summary

You should now be able to:

- Test the documentation for accuracy and completeness
- Test an RFC-enabled function module

Lesson: Business Object Repository

Lesson Overview

This lesson covers Business Objects, the Business Object Repository (BOR), and the architecture of BAPIs.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe Business Objects and their relation to the BOR

Business Example

In order to use a business object, you must understand what an object is and the various components of an object.

BOR

The Business Object Repository (BOR) is the object-oriented repository in the SAP System. It contains, among other objects, SAP business objects and SAP Interface Types and their methods.

The BOR is a complete development and runtime environment able to handle the following object types:

Business objects:

Include objects such as Customer, Material, and so on. They provide a programming interface to the SAP System.

Technical objects:

Include texts, notes, work items, and archived documents, as well as desktop objects like texts, graphics, and spreadsheets.

Metaobjects:

Document object types, methods, attributes, and events.

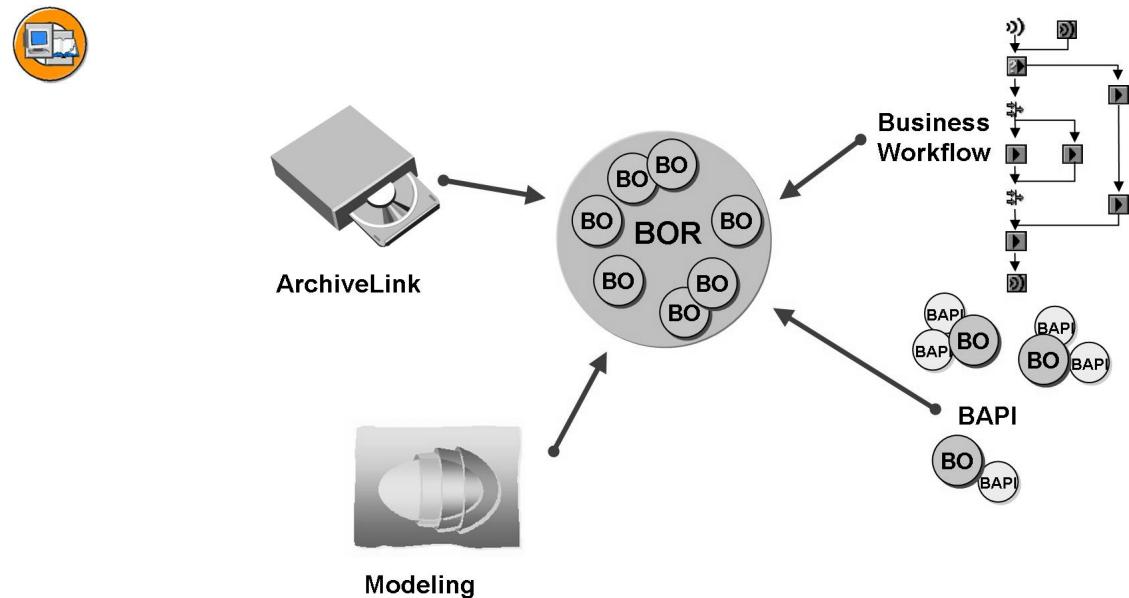


Figure 60: Business Object Repository (BOR)

In the BOR, a Business Application Programming Interface (BAPI) is defined as an API method of an SAP Business Object. Thus defined, the BAPIs become standard with full stability guarantees in regards to their content and interface.

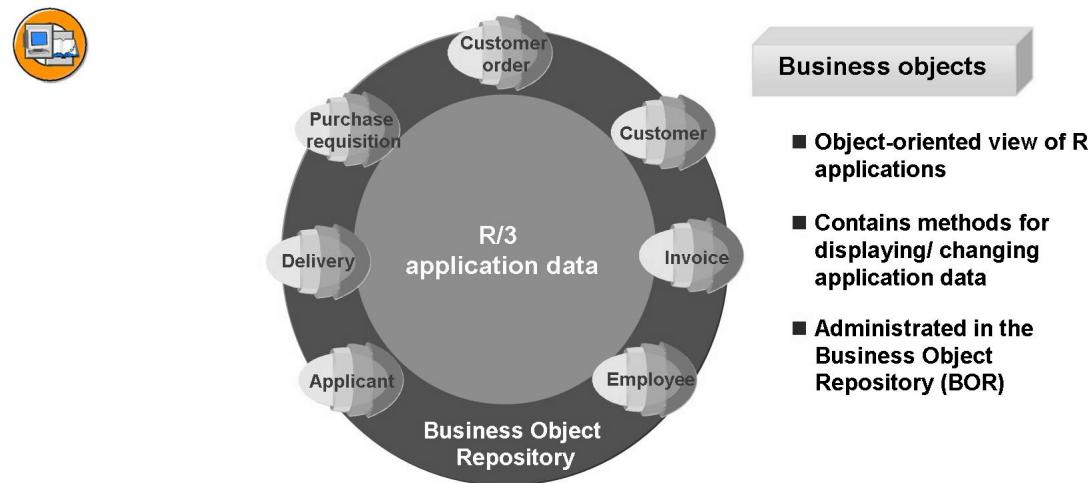


Figure 61: Business Objects in the BOR

The BOR manages business objects, organizational objects, and technical objects in a hierarchical display. With regard to SAP Business Objects and their BAPIs, the BOR has the following functions:

- Provides an object-oriented view of SAP Systems data and processes. SAP system application functions are accessed using methods (BAPIs) of SAP Business Objects.
- Arranges the various interfaces in accordance with the component hierarchy, enabling functions to be searched and retrieved quickly and simply.
- Manages BAPIs in release updates. BAPI interface enhancements made by adding parameters are recorded in the BOR. Previous interface versions can thus be reconstructed at any time.
- Ensures interface stability. Any interface changes that are carried out in the BOR, are automatically checked for syntax compatibility against the associated objects in the ABAP Dictionary.

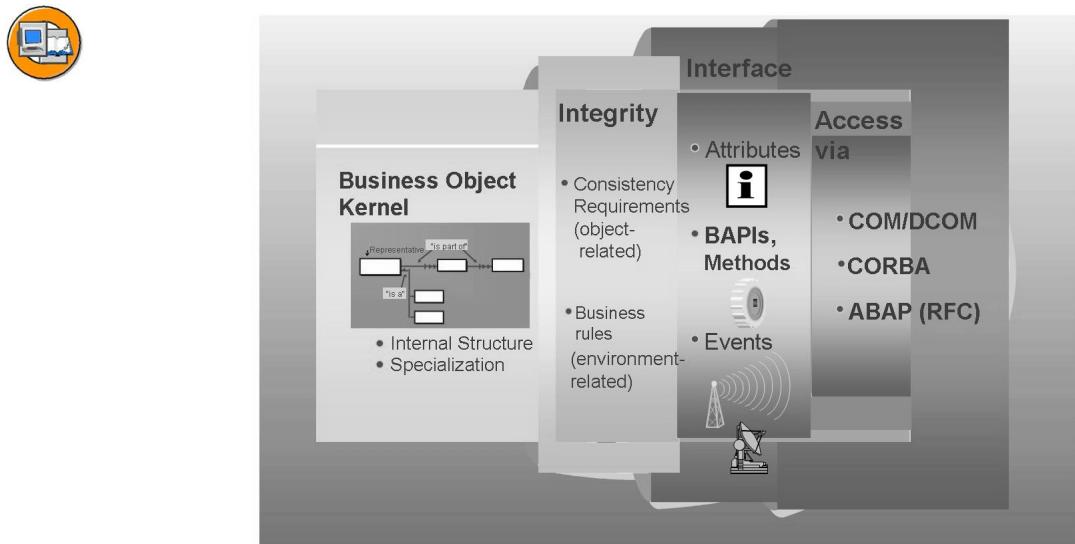


Figure 62: Business Object Type

Business Objects hide their internal structure via several layers. The object is described by the following components:

Basic data

Technical details such as an Internal ID, release level, transport data, and so on.

Interfaces

List of the interface from which the object type adopts attributes, methods, and events.

Key Fields

Attributes that uniquely identify an object. Usually the key fields of the underlying database tables.

Attributes

These are either values from database fields (field references), values that are calculated at runtime (virtual attributes), or pointers to other objects (object references).

Methods

Calls to transactions, function modules, or other ABAP code. BAPIs are known as API methods.

Events

To be used in workflow definitions. The events are only defined in the BOR.



Employee

Attributes:

- Name
- Address
- Salary

Methods:

- Change address
- Increase salary

Customer order

- Customer
- Date
- Item list

- Create
- Display
- Block

Figure 63: Example of a Business Object

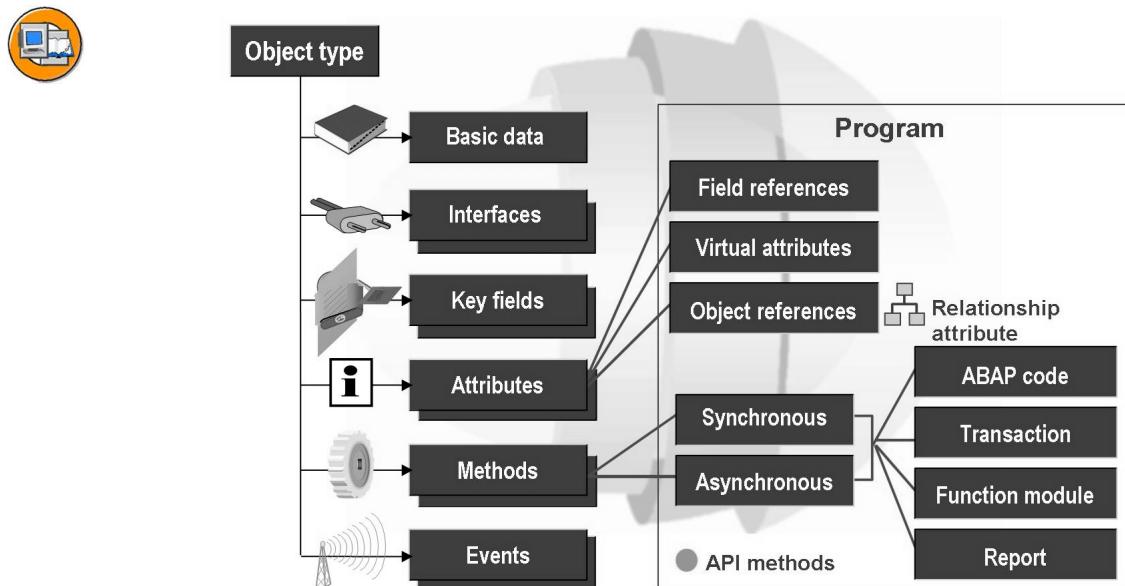


Figure 64: Components of an Object Type

Navigation in the BOR

The BOR is structured according to the application component hierarchy. Business objects are assigned to application component hierarchy according to their Package. For example, the business objects SalesOrder and BillingDocument are assigned to the SD component. The BOR offers the following transactions:

- Business Object Builder (Transaction SWO1)
- Business Object Repository Browser (Transaction SWO3)
- BAPI Browser (Transaction BAPI)

There are several ways of displaying object types in the BOR:

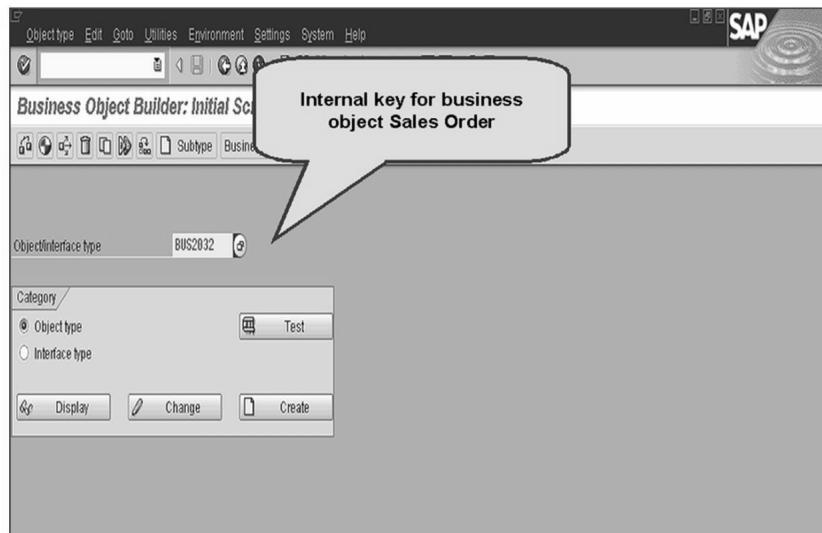


Figure 65: SWO1

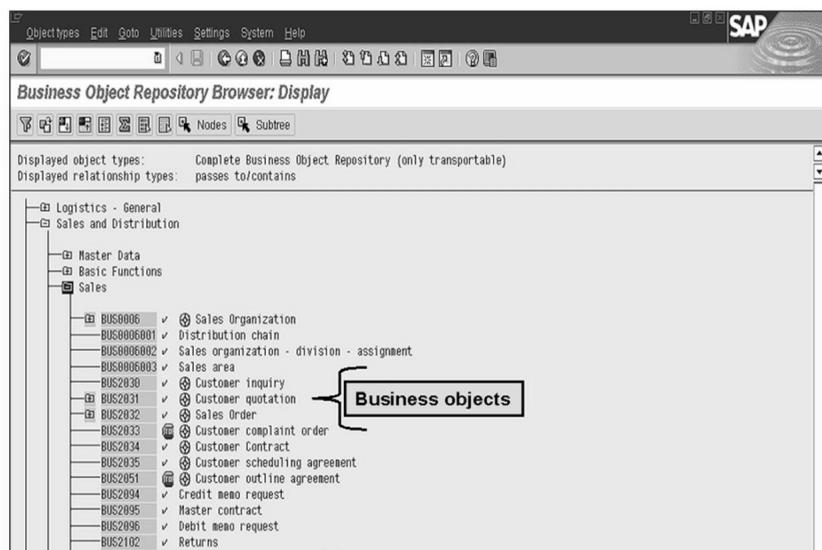


Figure 66: SWO3

If you know the technical name of the object type (for example, BUS2032), go to Transaction SWO1 and enter the name in the *Object type* field. Select *Display*.

If you know in which application the object type exists, go to Transaction SWO3 and search for the object type in the application component hierarchy. Select the required object type. The Business Object Repository Browser displays the object types and their relationships. The Browser can be configured using filters.

If you know the semantic name of the object, go to Transaction BAPI and search for the object type in the alphabetical listing.



The screenshot shows the SAP BAPI Explorer interface. On the left is a tree view of objects, with 'CustomerMaterialInfo' selected. The main panel displays detailed information about this object:

Object	Object Name
	CustomerMaterialInfo
Short description	Entity view 6093, customer-material-info
Object type	BUS3033
Pack.	VSKM
Component	SD-BE
Person responsible	SAP
Created on	30.11.1998
Release	46A
Status	Released
Last changed by	SAP
Changed on	12.05.1999

Figure 67: BAPI



Lesson Summary

You should now be able to:

- Describe Business Objects and their relation to the BOR

Lesson: Business Objects

Lesson Overview

In this lesson, you will learn to create a Business Object.



Lesson Objectives

After completing this lesson, you will be able to:

- Create a Business Object

Business Example

You want to get Customer and Contact information from an SAP System using an external call. To do this, you will need a Business Object with methods (BAPIs) that can perform this task.

Creating a Business Object

A business object is created in the Business Object Builder. You can get to the Business Object Builder by either executing transaction SWO1 or from the SAP menu path *Tools → ABAP Workbench → Development → Business Object Builder*.

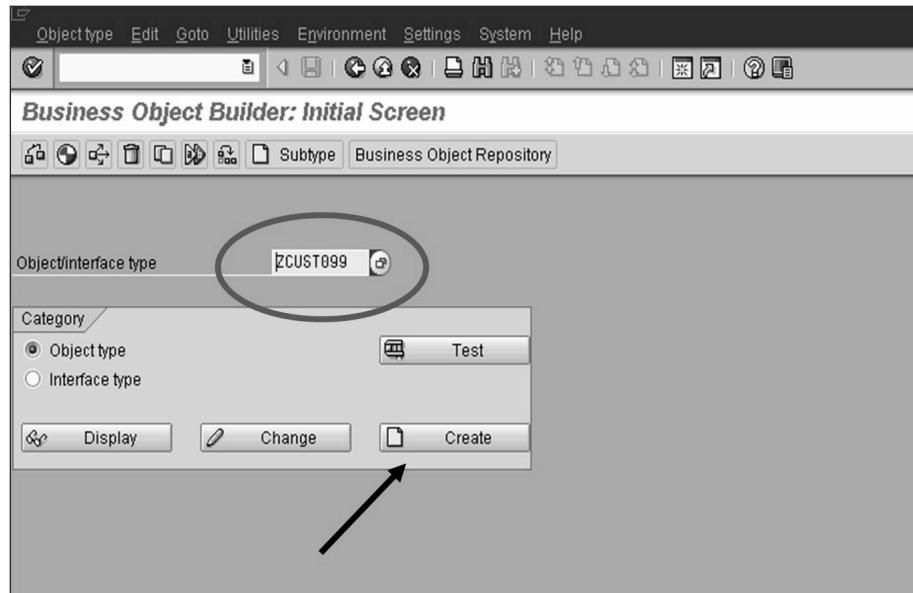


Figure 68: Business Object Builder

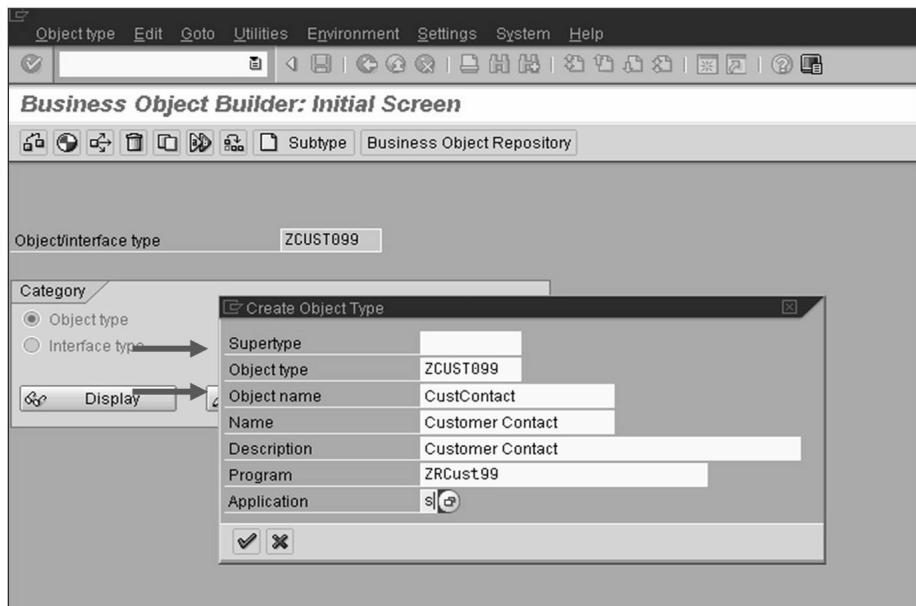


Figure 69: Business Object Builder

The following fields make up your objects characteristics:

Supertype

Only used if the new Business Object is subordinate to another Business Object

Object type

Internal technical key of a business object in the Business Object Repository (BOR). Business objects are identified in the BOR by object type (for example BUS2032) and a descriptive English name. (For example, SalesOrder) Both identifiers must be unique across all object types. The object type can have a maximum of 10 characters.

Object name

Descriptive, English ID of a business object in the Business Object Repository (BOR). The object ID is in English and can be a maximum of 32 characters, using both uppercase and lowercase. (For example: SalesOrder)

Name

A meaningful name used to select an object type.

Description

An identifying text of up to 40 characters.

Program

Each object type has an ABAP program in which the methods of the object type are implemented. The program can be automatically generated when you create or revise an object type. You are advised to carry out this automatic generation to minimize the likelihood of an error.

Application

Each object type belongs to a business and therefore application-specific environment. A categorization is carried out if the respective application ID is specified. (For example, HR)

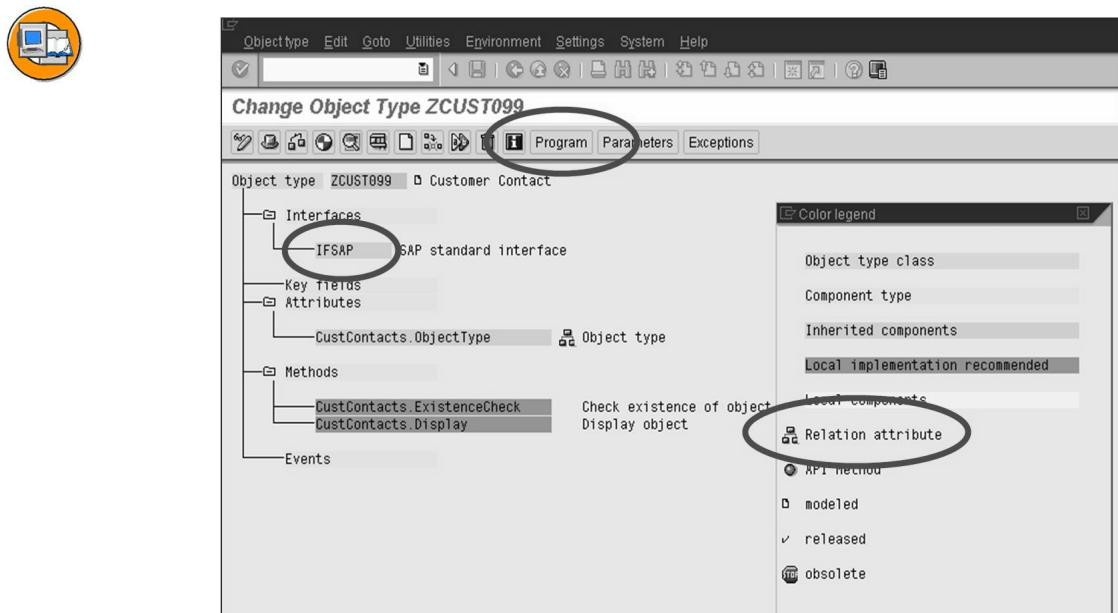


Figure 70: Business Object Builder

Once a Business Object is created, there are some standard components that exist for the object.

Interfaces

The standard IFSAP interface is part of all Business Objects. It is this interface that brings the default methods; ExistenceCheck and Display.

Relation Attribute

This identifies that any attribute in the list has a relationship to the Object type.

Methods

These are the default methods for every Business Object.

Key fields can be defined for the business object with reference to the ABAP Dictionary.



```

1 **** Implementation of object type ZCUST099 ****
2 INCLUDE <OBJECT>.
3 BEGIN_DATA OBJECT. " Do not change.. DATA is generated
4 * only private members may be inserted into structure PRIVATE
5 DATA:
6 " begin of private,
7 " to declare private attributes remove comments and
8 " insert private attributes here ...
9 " end of private,
10 KEY LIKE SWTOBJID-OBJKEY.
11 END_DATA OBJECT. " Do not change.. DATA is generated

```

Figure 71: Generated Code

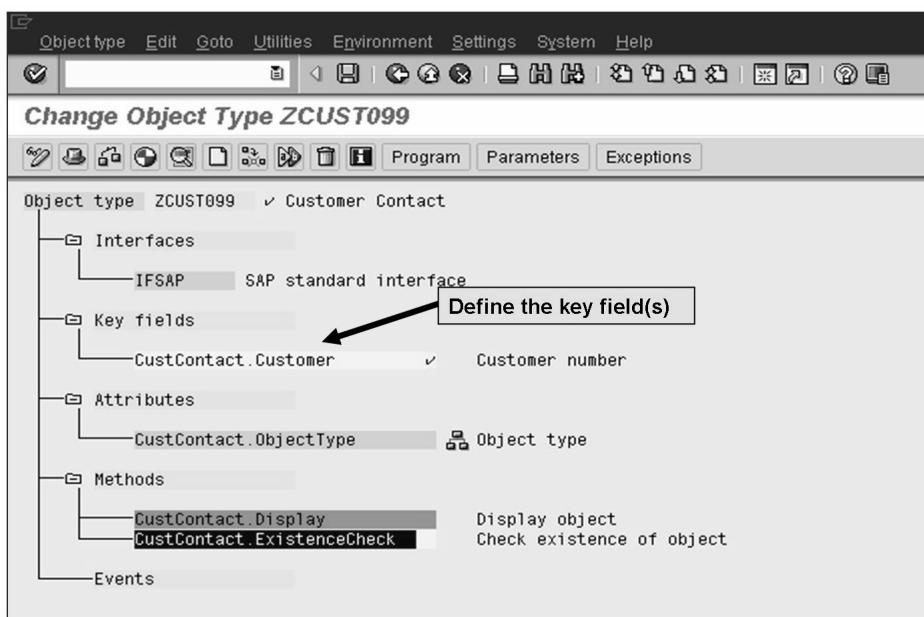


Figure 72: Define Key Fields

There is a default program that is created for the Business Object. As other components (BAPIs) are added to the Business Object, this program expands. When you change the status of your Business Object to Implemented, the program is automatically implemented.

For the Business Object to be finalized, the status has to be changed to Released. If this has been done, a check mark appears next to the object type name. You can now attach your BAPIs to your Business Object.

Optional: ExistenceCheck Method

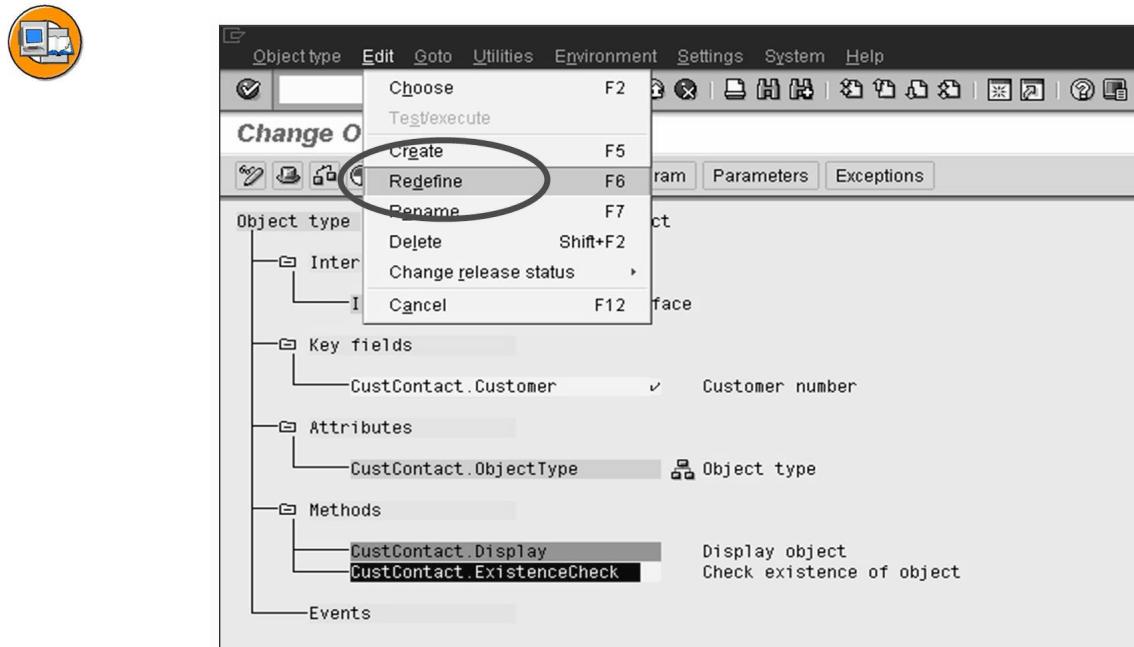


Figure 73: Redefine the Method

The BAPI ExistenceCheck checks whether an entry exists in the database for an SAP Business Object, for example, whether the customer master has been created. The BAPI ExistenceCheck is an instance method.

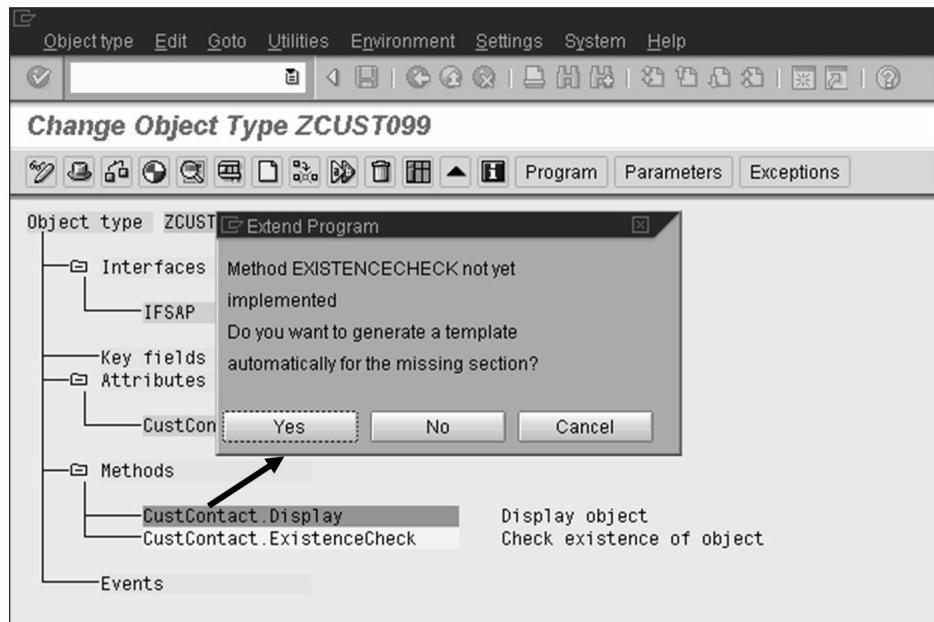


Figure 74: Implement the Method

You can implement this method as a BAPI and/or as a method of SAP Business Workflow. If you implement this method as a BAPI, it only has to be implemented once because an ExistenceCheck BAPI can also be used by SAP Business Workflow.

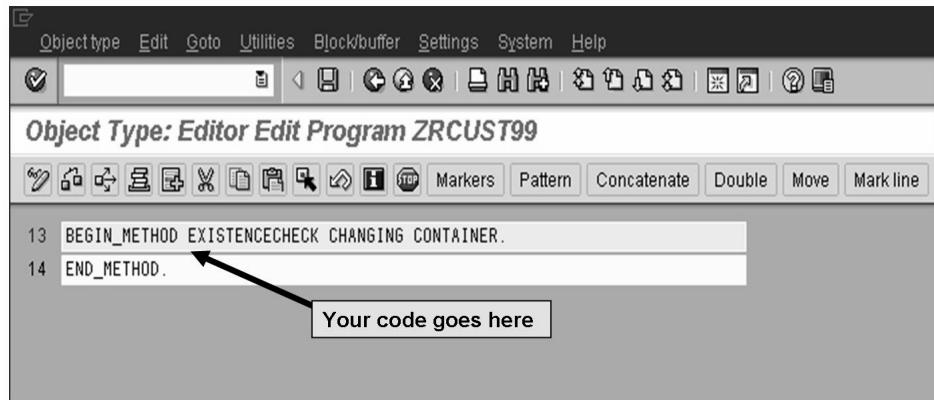
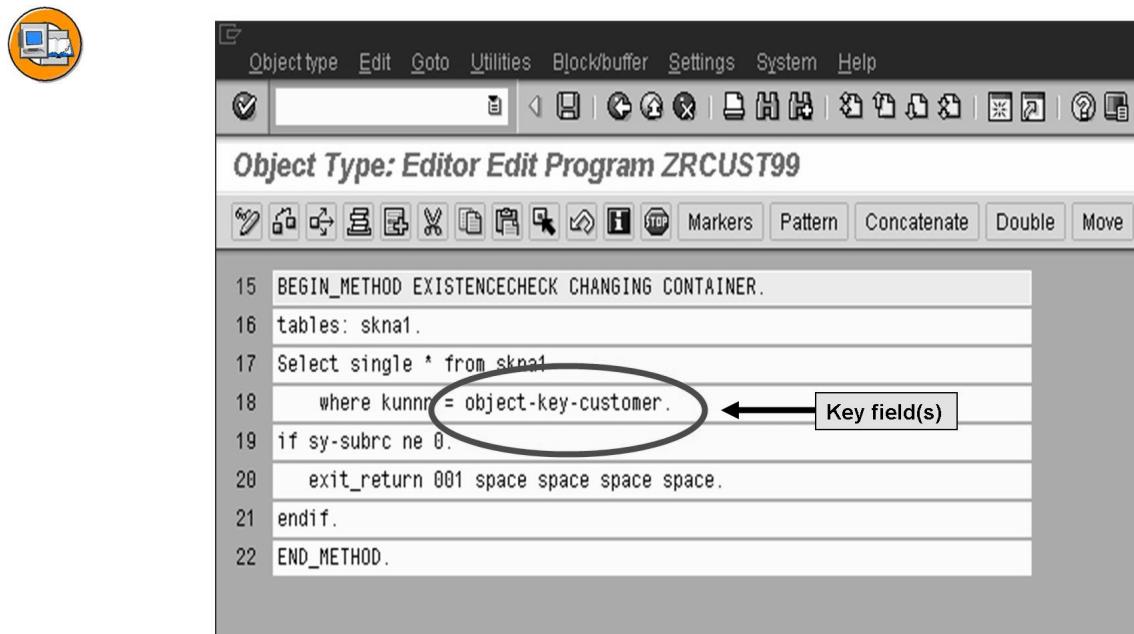


Figure 75: Existence Check Container



The screenshot shows the SAP ABAP editor interface. The title bar reads "Object Type: Editor Edit Program ZRCUST99". The toolbar contains various icons for file operations like Open, Save, Copy, Paste, and Print. Below the toolbar is a menu bar with "Objecttype", "Edit", "Goto", "Utilities", "Block/Buffer", "Settings", "System", and "Help". The main area displays ABAP code:

```
15 BEGIN_METHOD EXISTENCECHECK CHANGING CONTAINER.  
16 tables: skna1.  
17 Select single * from skna1  
18   where kunnr = object-key-customer. ← Key field(s)  
19 if sy-subrc ne 0.  
20   exit_return 001 space space space space.  
21 endif.  
22 END_METHOD.
```

A callout box labeled "Key field(s)" points to the line of code where the variable "customer" is used in a WHERE clause.

Figure 76: Existence Check Code

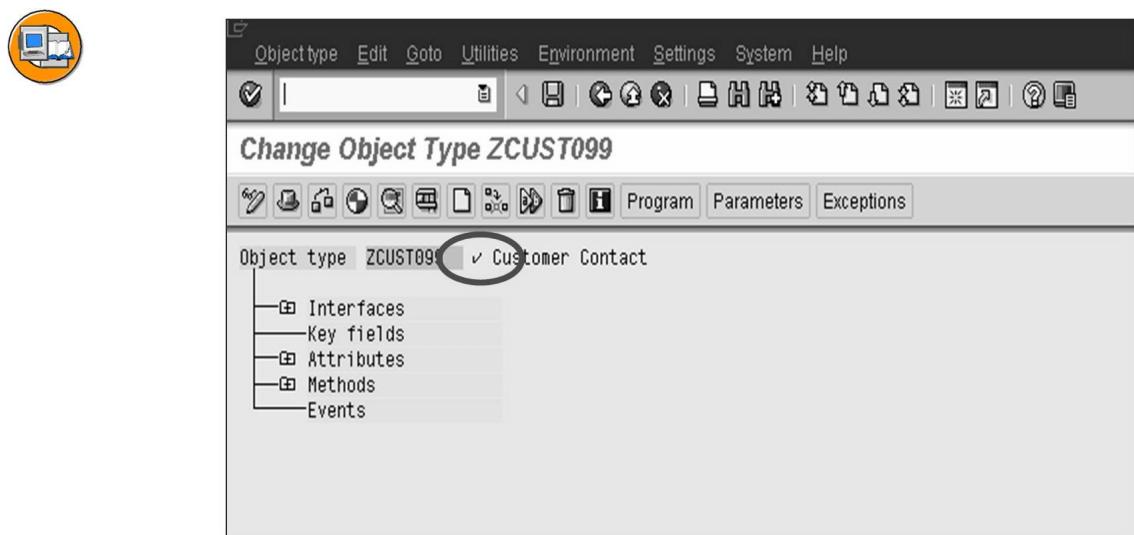


Figure 77: Release Business Objects

When you implement the method, you automatically create a method container area for your code.

To get the highest possible performance from the BAPI ExistenceCheck, you should reduce the number of columns to as few as possible before selecting the table rows.

You need to define the key fields for the table that you are going to read in the ExistenceCheck program. In the program, these fields will be prefixed with object-key.

Exercise 9: Business Object

Exercise Objectives

After completing this exercise, you will be able to:

- Create a Business Object using Business Object Builder

Business Example

A Business Object has to be created to use the proposed BAPIs for getting information from an SAP System externally.

Task:

Create Business Object **ZCON_##**.

1. Go to Business Object Builder.
2. Create a Business Object.
3. *Save* your Business Object.
4. Generate your Business Object.
5. Change the Release status of your Business Object.
6. Add the key field customer number to the business object.

Solution 9: Business Object

Task:

Create Business Object **ZCON_##**.

1. Go to Business Object Builder.
 - a) Either *Execute* transaction SWO1 or from the SAP Menu choose *Tools* → *ABAP Workbench* → *Development* → *Programming Environment* → *Business Object Builder*.
2. Create a Business Object.
 - a) In the *Object type* field, enter **zcon_##**, where **##** is your group number.
 - b) Select *Create*.
 - c) For the *Object type's basic data* fields, enter the following:

Field	Value
<i>Supertype</i>	
<i>Object Name</i>	ZContactData_##, where ## is your group number
<i>Name</i>	Contact Information
<i>Description</i>	Contact Information
<i>Program</i>	ZRCON_##
<i>Application</i>	S (Basis)

- d) Select the *Continue* icon.
3. Save your Business Object.
 - a) Enter your Package **ZBC417_##**, where **##** is your group number.
 - b) *Save*.
 - c) Your Business Object is now created.
4. Generate your Business Object.
 - a) Select the name of your Business Object.
 - b) Select the *Generate* icon (Red or White ball).

Continued on next page

5. Change the Release status of your Business Object.
 - a) On the menu, choose *Edit* → *Change release status* → *Object type* → *To implemented*. You should get a successful message.
 - b) On the menu, choose *Edit* → *Change release status* → *Object type* → *To released*. Enter past the information messages. You should get a successful message. Your Business Object is complete and available for adding BAPIs.
6. Add the key field customer number to the business object.
 - a) On the menu, choose *Goto* → *Obj. type components* → *Key fields*.
 - b) On the Edit key fields screen, select the *Create* icon.
 - c) On the pop-up window asking if you want to create your key field with an ABAP Dictionary field proposal, select **Yes**.
 - d) In the *Table entry* field, type in the name of the table where the key field exists (**SKNA1**) and select *Continue*. The key field names display in the window.
 - e) Select the key field(s) and then select *Continue*.
 - f) In the *Create* pop-up window, select the *Create* icon.
 - g) Save your work by selecting the *Save* icon.

 **Note:** This key field is important for a number of reasons. For example, you cannot generate an ALE interface on a business object that does not have a key field.



Lesson Summary

You should now be able to:

- Create a Business Object

Lesson: Business Object Components

Lesson Overview

This lesson covers how to attach an RFC-enabled function module to a Business Object as a BAPI.



Lesson Objectives

After completing this lesson, you will be able to:

- Add a BAPI as an API method to a business object.
- Test the BAPI Method

Business Example

An RFC-enabled function module that has been created needs to be attached to a Business Object to create a BAPI.

Creating a BAPI

The BOR/BAPI Wizard assists with creating BAPIs in the BOR. It takes you through the creation process step-by-step.

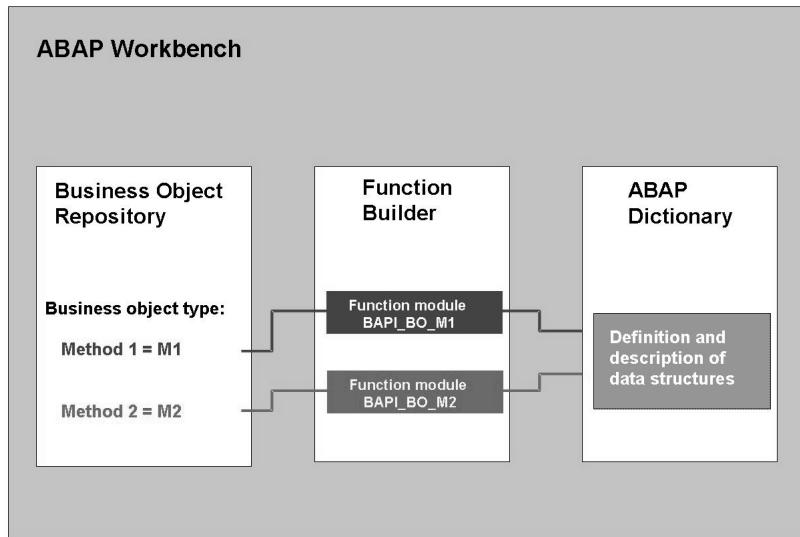


Figure 78: ABAP Workbench Components Needed

Various components of the ABAP Workbench are used when you define and implement a BAPI.

A BAPI is defined in the Business Object Repository (BOR) as an API method of an SAP business object type. Business object types and their BAPIs are described and created in the BOR. A BAPI is usually implemented as an RFC-enabled function module. These function modules are created and described in the Function Builder. The definitions and descriptions of the data structures used by the BAPI are defined in the ABAP Dictionary.

The graphic above shows the components of the ABAP Workbench that are used when BAPIs are defined and implemented.

The Process Flow

To define the BAPI:

- Identify the SAP Business Object type.
- Append the BAPI as an API method to the business object type.

You can use the BAPI Wizard to do this. This generates the method in the BOR based on the BAPI function module you created. You have to perform some additional tasks in this procedure. Keep in mind the following points:

- Upper case/lower case (each new word must be in upper case). Example: CustomerList.
- The import and export behavior of the table parameters must be correctly defined in the BOR. Reason: Unlike the function module, in the BOR you can differentiate between import and export tables. You should therefore only select the standard option Import/export, if the table is actually going to be imported and exported.
- The Return parameter is always defined as an export parameter.

As well as making the task much easier, the Wizard also ensures that the BAPI interface is identical in the BOR and in the function module. In particular this concerns the following points:

- For each function module parameter there is one BOR parameter (or one key field) of the same name (or vice versa).
- The associated parameters are based on the same ABAP Dictionary structure.
- The associated parameters have the same characteristics (mandatory or optional, import, export or table parameters).

The BOR/BAPI Wizard Process

The following process is used to define a Method in the BOR using the BOR/BAPI Wizard:

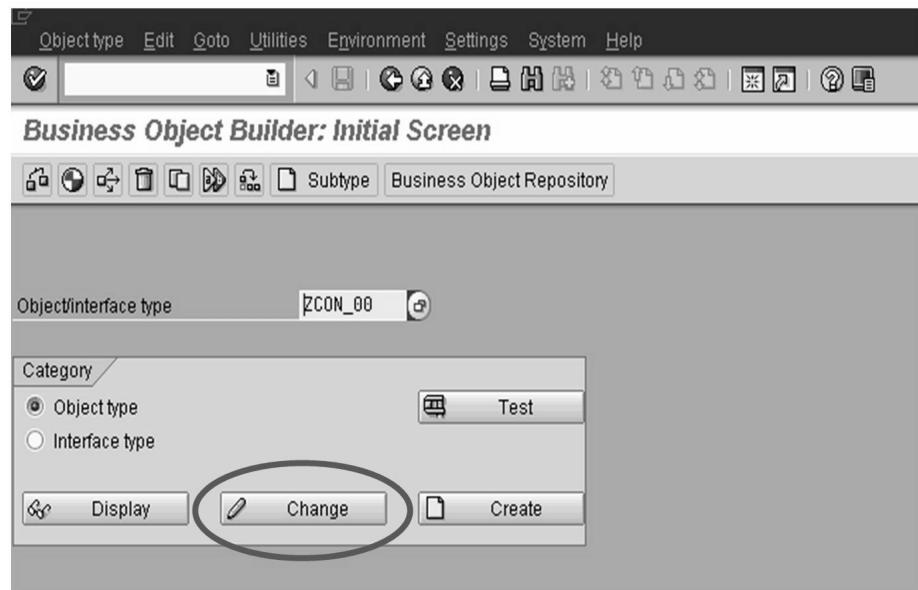


Figure 79: BOR/BAPI Wizard Process (1)

1. Go to the Business Object Builder.
2. Enter the name of the Business Object for the method and select the Change button.

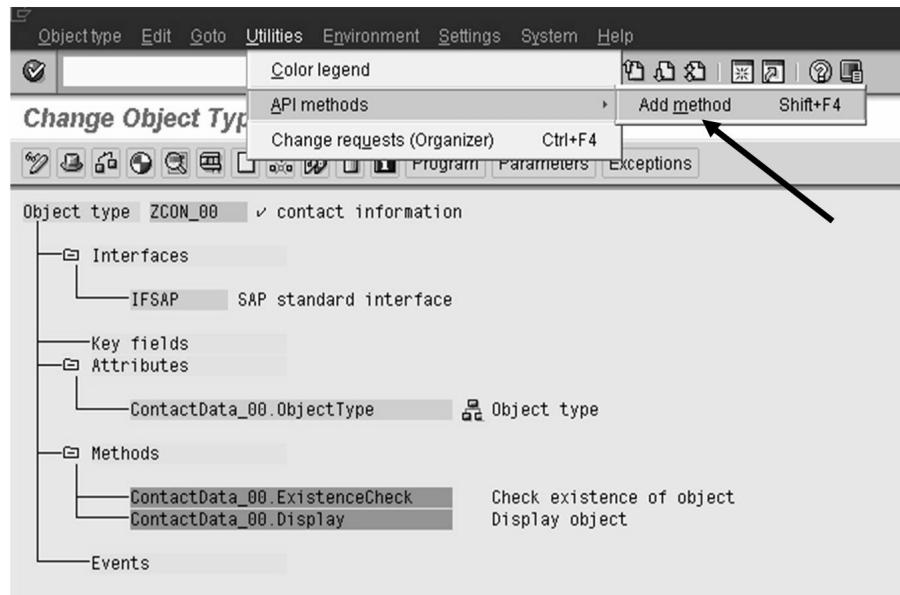


Figure 80: BOR/BAPI Wizard Process (2)

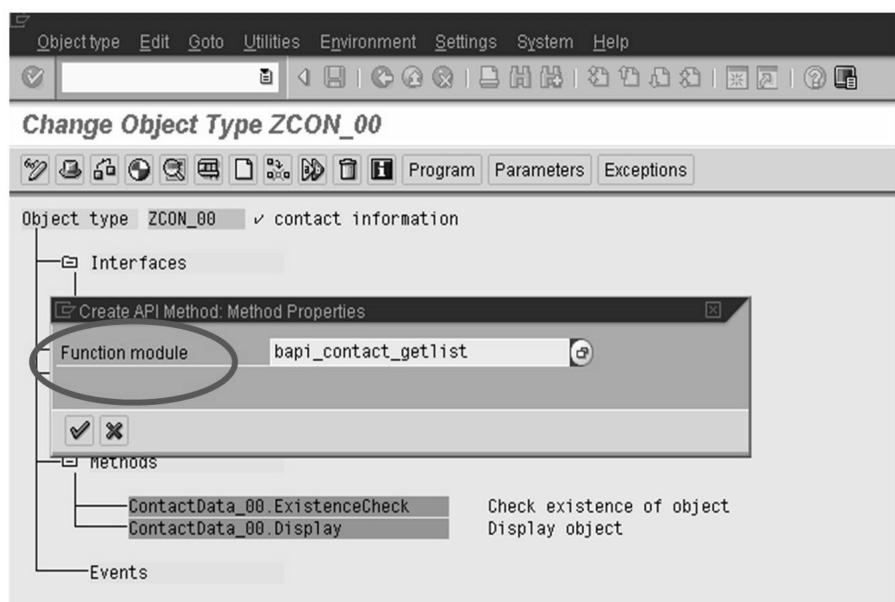


Figure 81: BOR/BAPI Wizard Process (3)

3. On the menu, choose *Utilities* → *API methods* → *Add method*. This displays a dialog box where you enter the name of the function module.

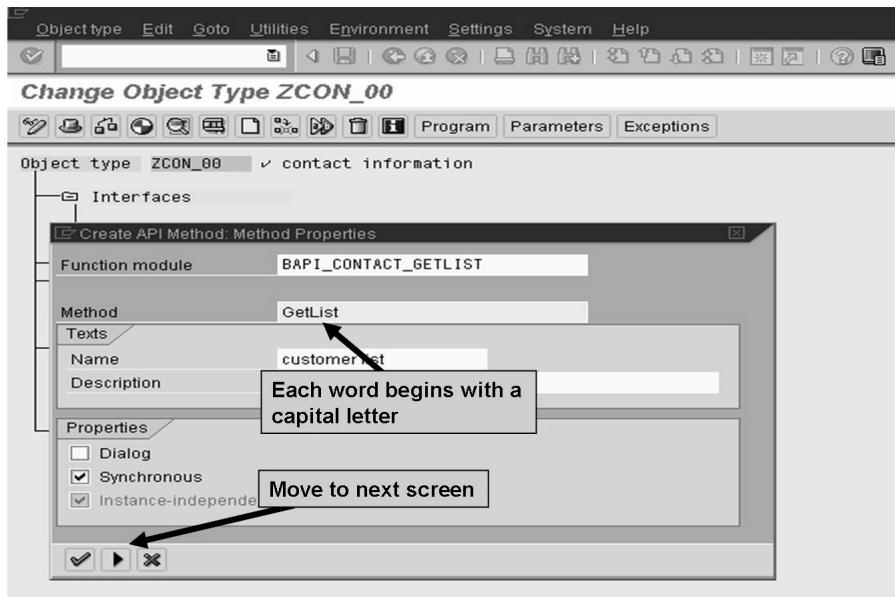


Figure 82: BOR/BAPI Wizard Process (4)

4. Enter the Method name that you want to use for your BAPI.

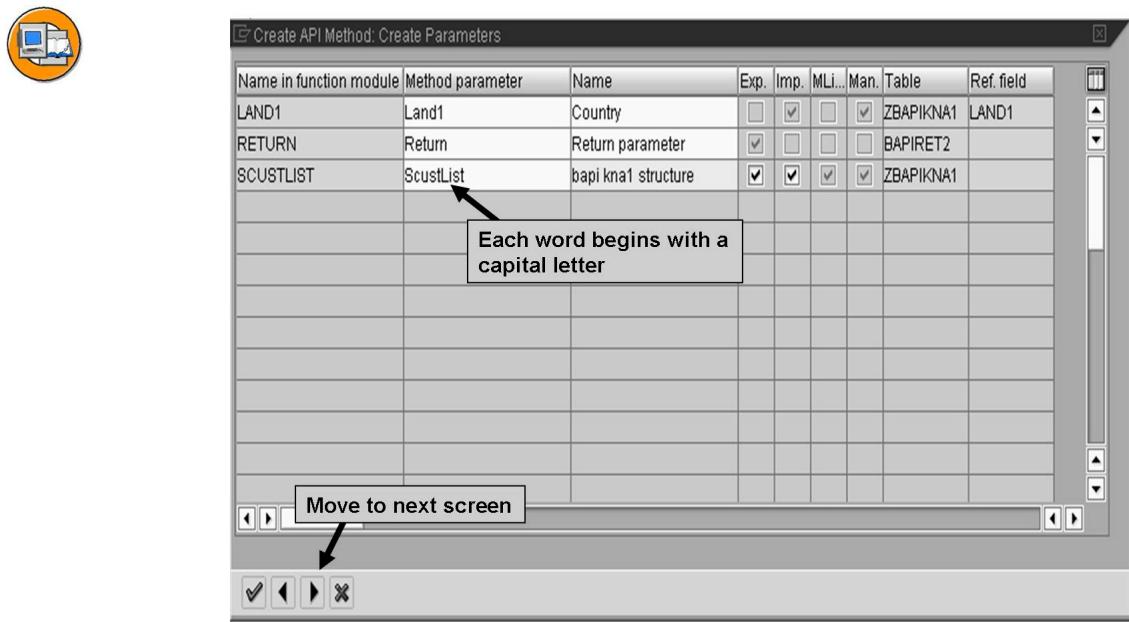


Figure 83: BOR/BAPI Wizard Process (5)

5. Use the right arrow to get to the Create parameter dialog box. Here you will see a list of your parameters and default names that you can modify if needed. Make sure that each new word in a name begins with a Capital letter (i.e. CustomerList). This is a rule of object orientation. Also, make sure that the parameter names in the BOR are identical to the parameter names in the function module except for the Upper/lower case letters.
6. Specify whether the individual table parameters are used for data import or data export. Table parameters are marked with a check in column Mline (multiple lines).

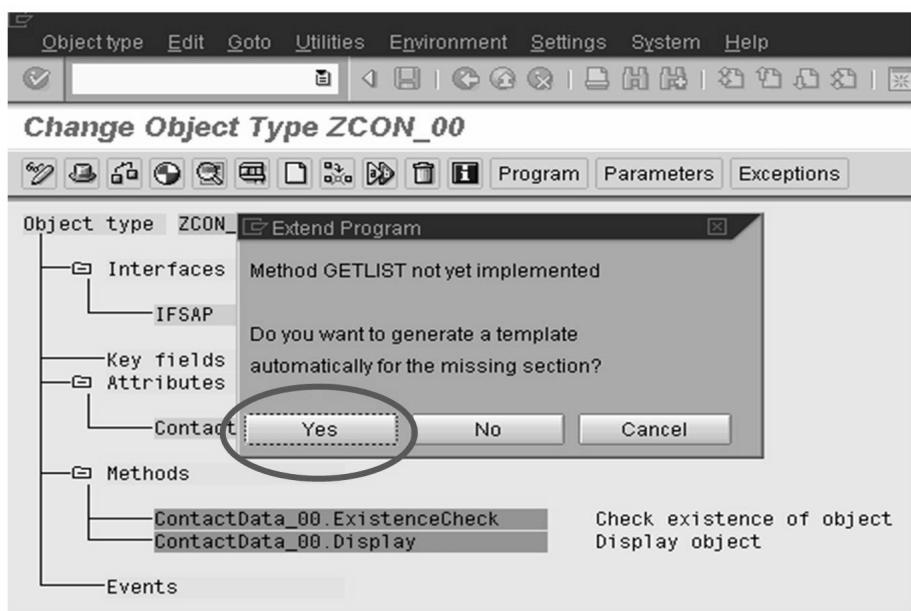


Figure 84: BOR/BAPI Wizard Process (6)

7. In the next step, the Extend Program step, you get a warning that the BAPI is not yet implemented. Select yes to generate the template that will add the wrapper object-oriented code to your function module converting it to a method of an SAP Business Object.
8. Generate the BAPI. You will be back on the Change object type screen. Expand the Methods to display all of the Business Object's methods. Your BAPI should now be listed with a *Green circle* icon (green light). Select and generate your BAPI.

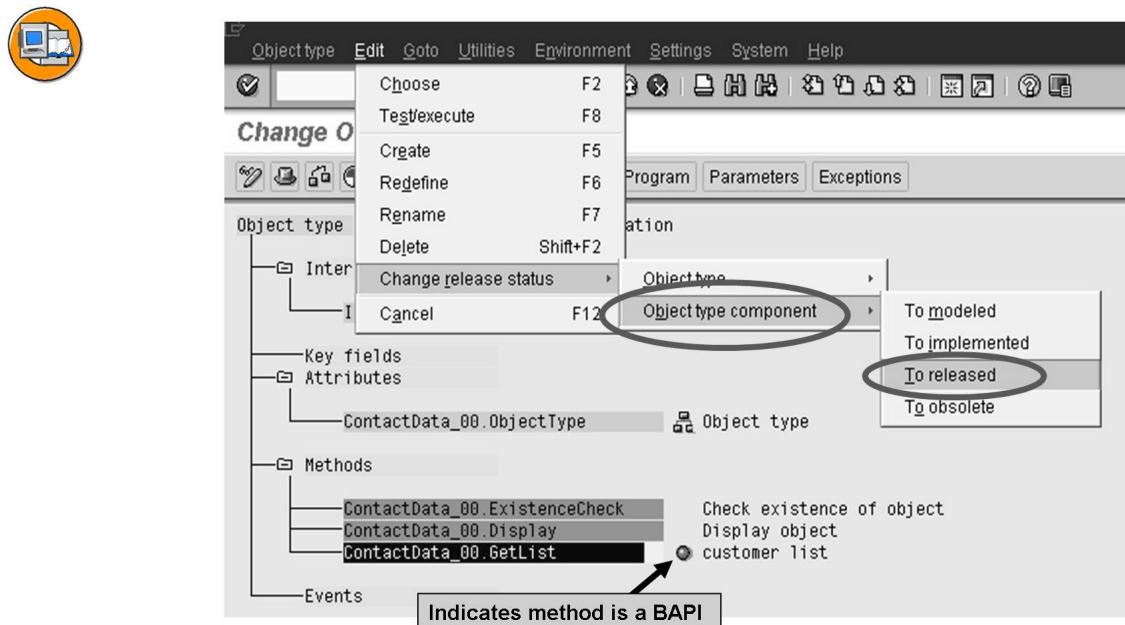


Figure 85: BOR/BAPI Wizard Process (7)

Select your method and choose the menu path *Edit* → *Change release status* → *Object type component* → *To released*. This releases your method.

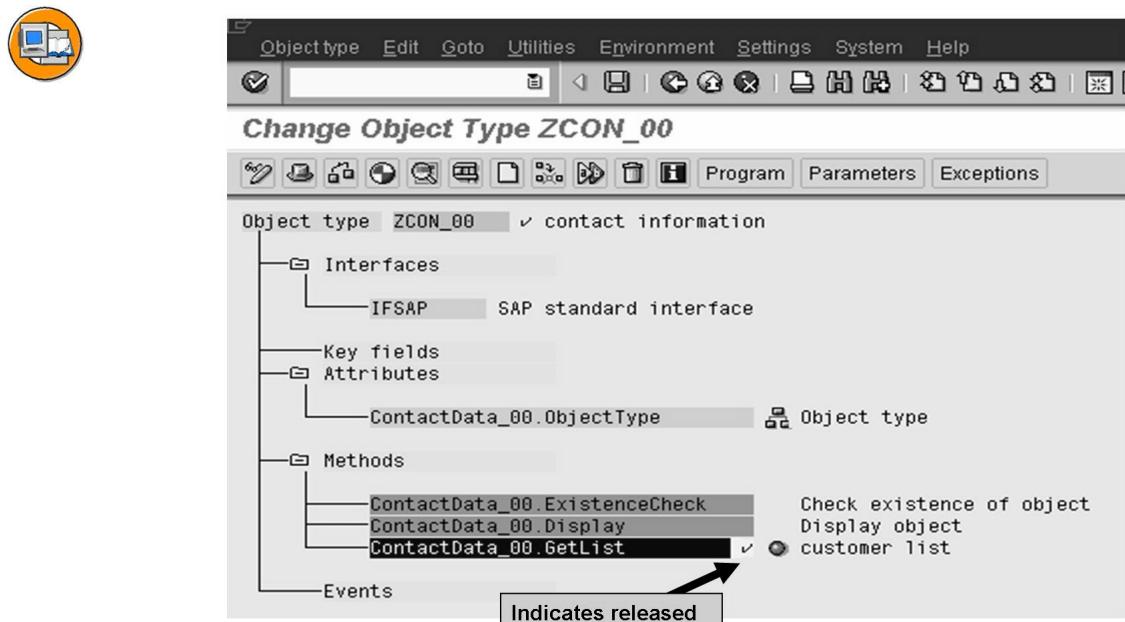


Figure 86: BOR/BAPI Wizard Process (8)

Your BAPI is complete and ready for testing.



Testing the Method

Carry out an individual test and an integration test.

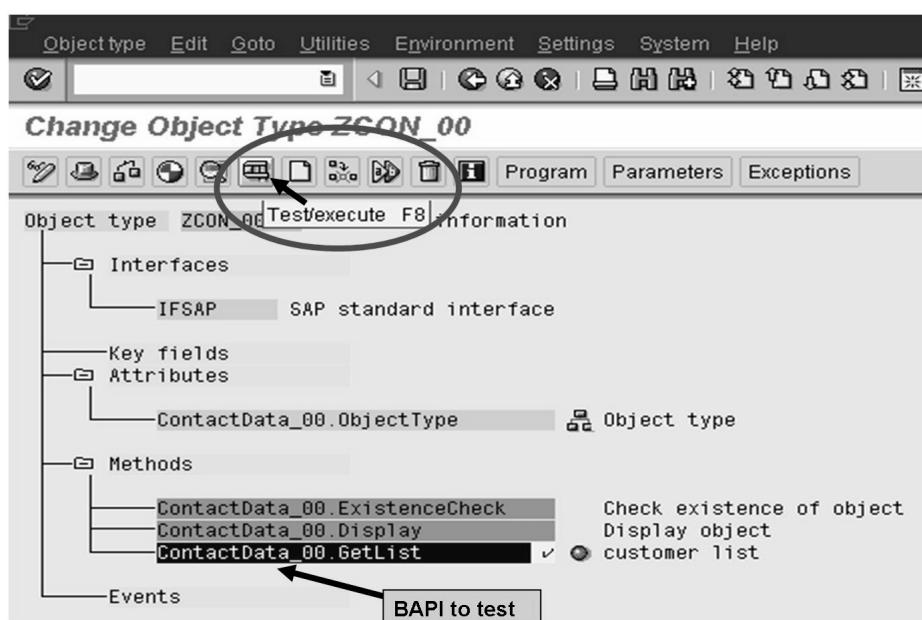


Figure 87: Test the BAPI

Before you begin the testing, make sure that you save your work. Test your BAPI by selecting your BAPI once and then select the test tool in the toolbar or choose *F8*.

The test consists of 2 steps:

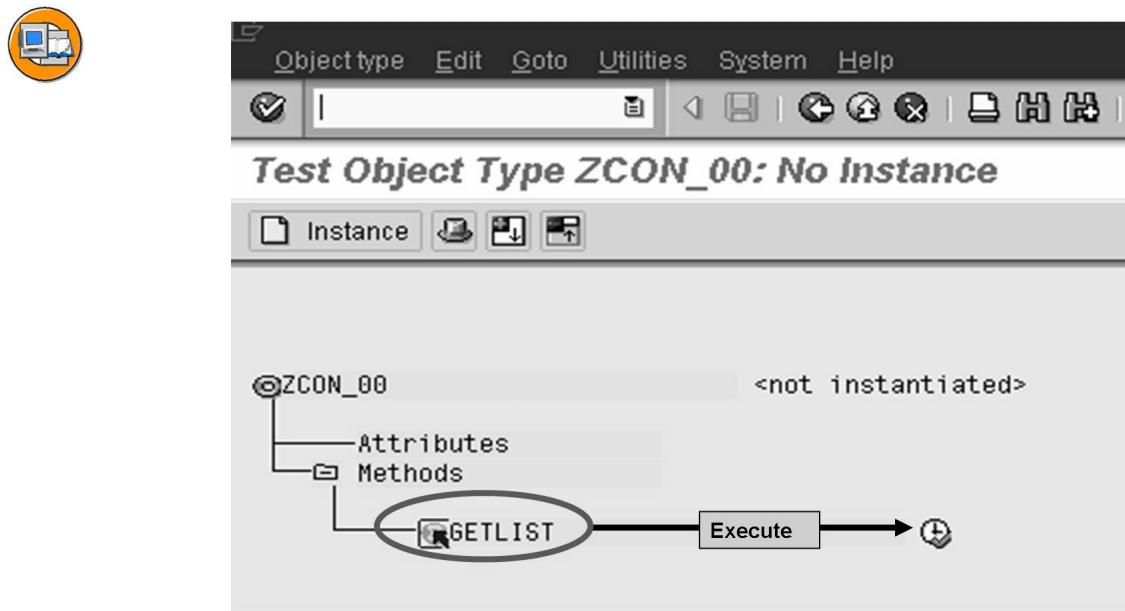


Figure 88: Execute the BAPI

Step 1 is to execute the BAPI. This puts you into the Test mode where you enter any necessary parameter values .

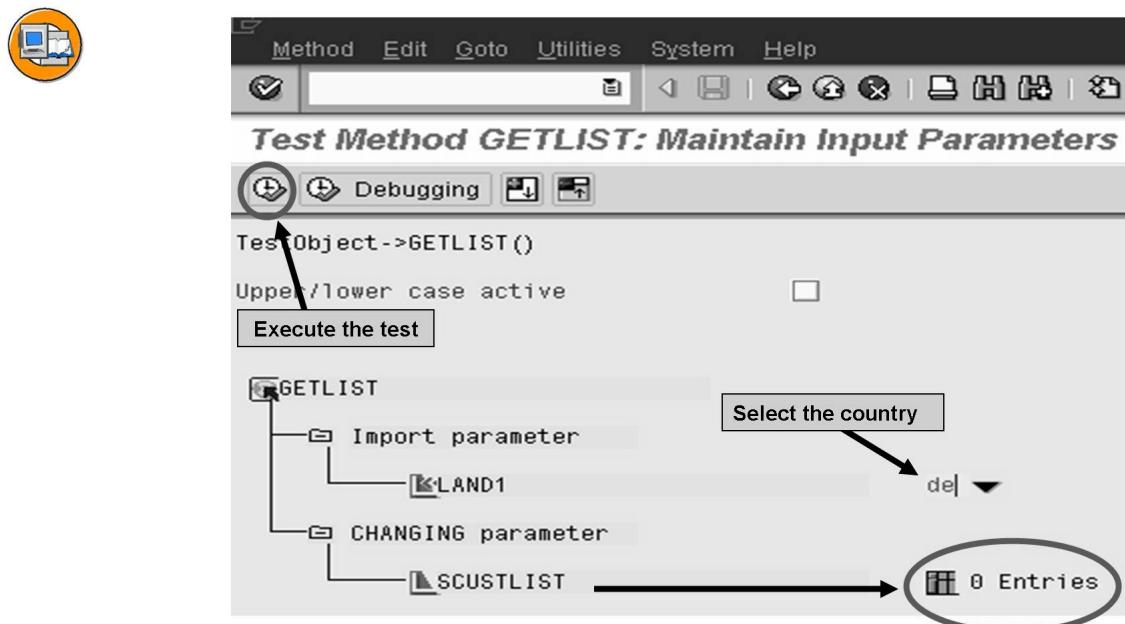


Figure 89: Execute the Test

Step 2 actually executes the BAPI test.

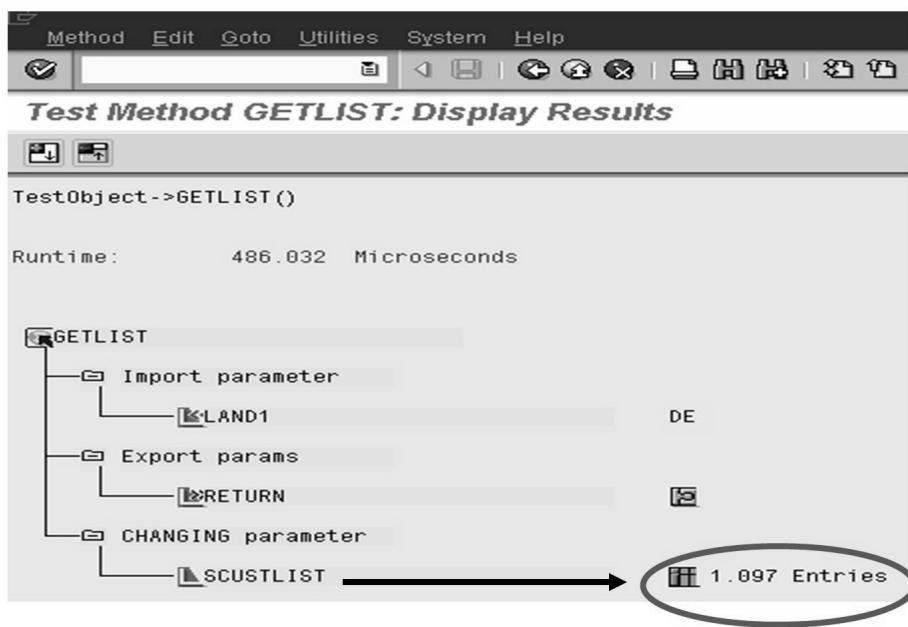


Figure 90: Completed Test

Once the test is completed, you get a display screen with the results of the parameters and any values they contain from the test. If a message was returned, it will appear on the RETURN parameter line. If you have a table of data to be exported, this provides a count of the number of records that were returned.



The screenshot shows the SAP Structure Editor displaying the results of the GETLIST:SCUSTLIST query. The table has the following structure:

BAPI Table Parameter				
1097 Entries				
KUNNR	LAN	NAME1	NAME2	ORT01
2	DE	Wett		Walldorf
99	DE	Eimalkunde		München
110	DE	Auto Clement	Exclusive Automobile	Frankfurt
999	DE	Ship-to location #1		Berlin
1000	DE	Becker Berlin		Frankfurt
1001	DE	Lampen-Markt GmbH		Nuernberg
1002	DE	Omega Soft-Hardware Markt		Berlin
1010	DE	Becker Berlin (Versand)	SAPSOTA Haupthaendler	Muenchen
1012	DE	Autohaus Franzl GmbH		Berlin
1028	DE	Becker Berlin (Lagerung)		Muenchen
1032	DE	Institut fuer Umweltforschung		Muenchen
1033	DE	Karsson High Tech Markt		Freiburg
1034	DE	ERL Freiburg	Ecological Research Laboratories	Berlin
1050	DE	Becker AG		Frankfurt
1100	DE	Phundix KG		Duesseldorf
1110	DE	I.D.O.C. GmbH	Pumpensysteme für die	Nuernberg
1111	DE	P.S.G. GmbH		Frankfurt
1170	DE	Buy & Fly Supermarkt		Hamburg
1171	DE	Hitech AG		

Figure 91: Display Test Results

If your table parameter has records, select the table to see the actual data returned.
Finally, check that the documentation is complete and understandable.

Exercise 10: Add the GetList and GetDetail RFC function modules to the BOR as a BAPIs.

Exercise Objectives

After completing this exercise, you will be able to:

- Create a BAPI method by appending a function module to the Business Object

Business Example

You want to integrate your Z_BAPI_CONTACT_GETLIST_## and your Z_BAPI_CONTACT_GETDETAIL_## function modules into the BOR so that they become methods to your ZCON## business object.

Task:

Add the Z_BAPI_CONTACT_GETLIST_## BAPI to your business object as an API method.

1. Go to the Business Object Builder.
2. Select your Business Object.
3. Add the method to the Business Object.
4. Release the BAPI.
5. Test the BAPI.
6. Repeat the above steps for your Z_BAPI_CONTACT_GETDETAIL_## function module.

Solution 10: Add the GetList and GetDetail RFC function modules to the BOR as a BAPIs.

Task:

Add the Z_BAPI_CONTACT_GETLIST_## BAPI to your business object as an API method.

1. Go to the Business Object Builder.
 - a) On the SAP Menu, choose *Tools* → *ABAP Workbench* → *Programming Environment* → *Business Object Builder* or execute Transaction SWO1.
2. Select your Business Object.
 - a) Type in the name of the Business Object, ZCON_##..
 - b) Select the *Change* icon.
3. Add the method to the Business Object.
 - a) On the menu, choose *Utilities* → *API Methods* → *Add method*.
 - b) In the *Function module* pop-up box, enter the name of your function module, Z_BAPI_CONTACT_GETLIST_##.
 - c) In the *Methods properties* pop-up window, type the name that you want to call your BAPI.

 **Note:** use a capital letter for each new word. Example, GetList.
 - d) Select the *Next Step* icon (right arrow) to move to the parameters screen.
 - e) On the Create Parameters screen, verify that the proposed parameter names are correct. Make any necessary changes.

 **Note:** Use a capital letter for each new word. Example, CustomerList.
 - f) Select the *Next Step* icon (right arrow) to move to the next screen.
 - g) You will get a pop-up window with a message indicating that your method has not yet been implemented. Select *Yes* to implement your method.
 - h) You will now be back on the initial Business Object screen. Your method will now have a green light next to it. This indicates that it is now a BAPI.

Continued on next page

4. Release the BAPI.
 - a) Select the BAPI you want to release.
 - b) On the menu, choose *Edit* → *Change release status* → *Object type component* → *To released*. Your method now has a checkmark next to it. This indicates that it is released.
 - c) Choose your BAPI and select *Save*.
 - d) Choose your BAPI and select the *Generate* icon (red and white ball).
5. Test the BAPI.
 - a) Select the BAPI you want to test and choose *Execute* or *F8*.
 - b) Continue past the information message regarding key fields.
 - c) On the Test Object screen, select the *Execute* icon next to the BAPI you want to test.
 - d) On the Test Method screen, enter any necessary parameter values for the test and select *Execute* or *F8*.
 - e) On the Test Method Display Results screen, your parameters will contain the returned values from the test.
 - f) If you have a table of values returned, select *Edit table* to display the returned records.
6. Repeat the above steps for your Z_BAPI_CONTACT_GETDETAIL_## function module.
 - a) no solution



Lesson Summary

You should now be able to:

- Add a BAPI as an API method to a business object.
- Test the BAPI Method

Lesson: Logical Unit of Work

Lesson Overview

In everyday language, a transaction is a sequence of actions that logically belong together in a business sense and which either procure or process data. It covers a self-contained procedure, for example, generating a list of customers, creating a flight booking, or sending reminders to customers. From the point of view of the user, it forms a logical unit. The development process must insure that this logical consistency is maintained, regardless of the technical implementation.



Lesson Objectives

After completing this lesson, you will be able to:

- Define the terms database LUW and SAP LUW
- Explain the need to bundle changes to the database tables
- Apply the knowledge of updating techniques to the distributed application environment and the use of BAPIs

Business Example

When you transfer an amount in financial accounting, this must first be deducted from one account before being credited to another. In between the two postings, the data is inconsistent, since the amount that you are posting does not exist in either account.

Overview

The completeness and correctness of data must be assured within a transactional unit. In the middle of a transaction, the data will usually be inconsistent. It is essential for application programmers to know that their data is consistent at the end of the transaction. If an error occurs, it must be possible to undo the changes made within a logical process.

In the SAP System, there are three terms frequently used in this connection:

- Database Logical Unit of Work (LUW): A database LUW is the mechanism used by the database to ensure that its data is always consistent.
- SAP Logical Unit of Work (LUW): An SAP LUW is a complete logical unit consisting of dialog steps, whose changes are written to the database in a single database LUW.
- SAP Transaction: An SAP transaction is an application program that you start using a transaction code. It may contain one or more SAP LUWs.

The following sections of this lesson explain these three terms in more detail as well as describing the concepts used to address them.



Database Logical Unit of Work

From the point of view of database programming, a database LUW is an inseparable sequence of database operations that ends with a database commit. The database LUW is either fully executed by the database system or not at all. Once a database LUW has been successfully executed, the database will be in a consistent state. If an error occurs within a database LUW, all of the database changes since the beginning of the database LUW are reversed. This leaves the database in the state it had before the transaction started.

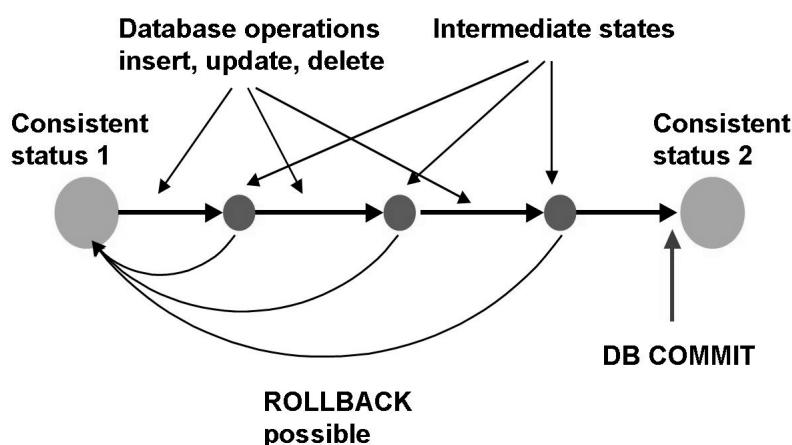


Figure 92: Database LUW

The database changes that occur within a database LUW are not actually written to the database until after the database commit. Until this happens, you can use a database rollback to reverse the changes. In the SAP System, database commits and rollbacks can be triggered either implicitly or using explicit commands.

Implicit Database Commits in the SAP System

The three-tier architecture of the SAP System has certain consequences for process handling. When a work process is released for use by another user (client), an implicit database commit is triggered for the database process assigned to it (via a basis program).

Work processes on the application server and database are released before each user dialog. This ensures that long user dialogs in which the system is "only displaying a screen" are not included in database LUWs. The duration of the user interaction will be longer than the DB LUW duration. Shorter database LUWs lead to less load on the database.

Implicit commits on the database are triggered whenever the work process has to wait.

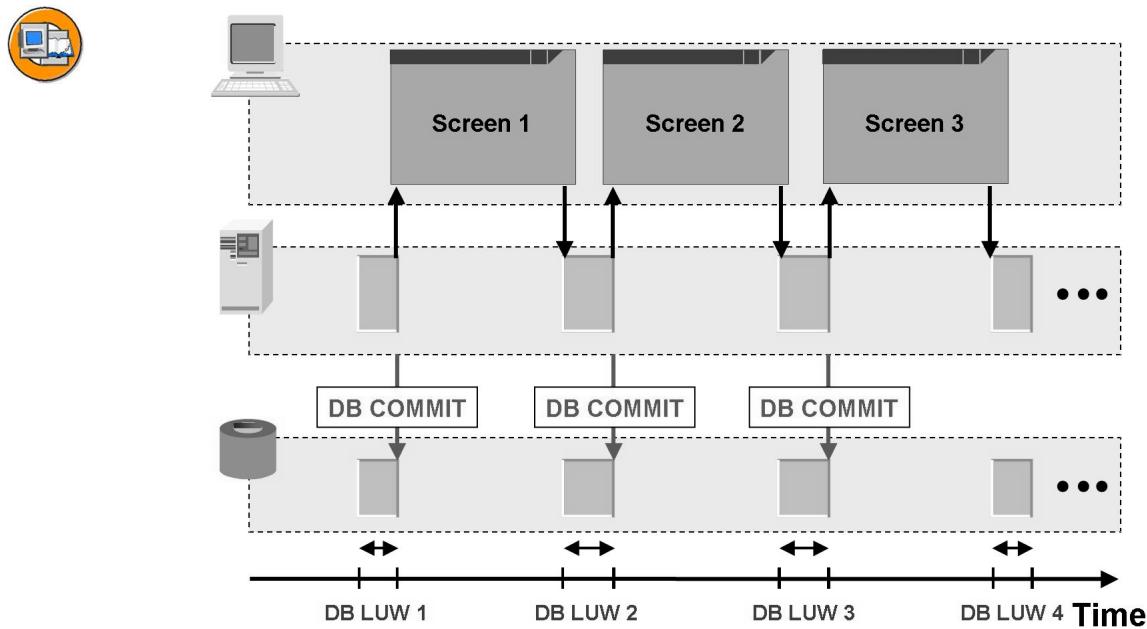


Figure 93: Implicit Database Commits

A work process can only execute a single database LUW. The consequence of this is that a work process must always end a database LUW when it finishes its work for a user or an external call. There are four cases in which work processes trigger an implicit database commit:

- When a dialog step is completed, control changes from the work process back to the SAPgui.
- When a function module is called in another work process (RFC). Control passes to the other work process.
- When the called function module (RFC) in the other work process ends. Control returns to the calling work process.
- Error dialogs (information, warning, or error messages) in dialog steps. Control passes from the work process to the SAPgui.

Explicit Database Commits in the SAP System

There are two ways to trigger an explicit database commit in your application programs:

- Call the function module DB_COMMIT: The sole task of this function module is to start a database commit.
- Use the ABAP statement COMMIT WORK: This statement starts a database commit, but also performs other tasks (refer to the keyword documentation for COMMIT WORK).

Implicit Database Rollbacks in the SAP System

The following cases lead to an implicit database rollback:

- Runtime error in an application program: This occurs whenever an application program has to terminate because of an unforeseen situation (for example, trying to divide by zero).
- Termination message: Termination messages are generated using the ABAP statement MESSAGE with the message type A or X. In certain cases (updates), they are also generated with message types I, W, and E. These messages end the current application program.

Explicit Database Rollbacks in the SAP System

You can trigger a database rollback explicitly using the ABAP statement ROLLBACK WORK. This statement starts a database rollback, but also performs other tasks (refer to the keyword documentation for COMMIT WORK).

From the above, we can draw up the following list of points at which database LUWs begin and end.

- A Database LUW Begins
 1. Each time a dialog step starts (when the dialog step is sent to the work process).
 2. Whenever the previous database LUW ends in a database commit.
 3. Whenever the previous database LUW ends in a database rollback.
- A Database LUW Ends
 1. Each time a database commit occurs. This writes all of the changes to the database.
 2. Each time a database rollback occurs. This reverses all of the changes made during the LUW.

Database LUWs and Database Locks

As well as the database changes made within it, a database LUW also consists of database locks. The database system uses locks to ensure that two or more users cannot change the same data simultaneously, since this could lead to inconsistent data being written to the database. A database lock can only be active for the duration of a database LUW. They are automatically released when the database LUW ends. In order to program SAP LUWs, we need a lock mechanism within the SAP System that allows us to create locks with a longer lifetime (refer to the lesson on Locking.)

SAP Logical Unit of Work

An SAP logical unit of work (LUW) is a functionally complete set of steps within a business process in the SAP System. Those process steps must be logically related.

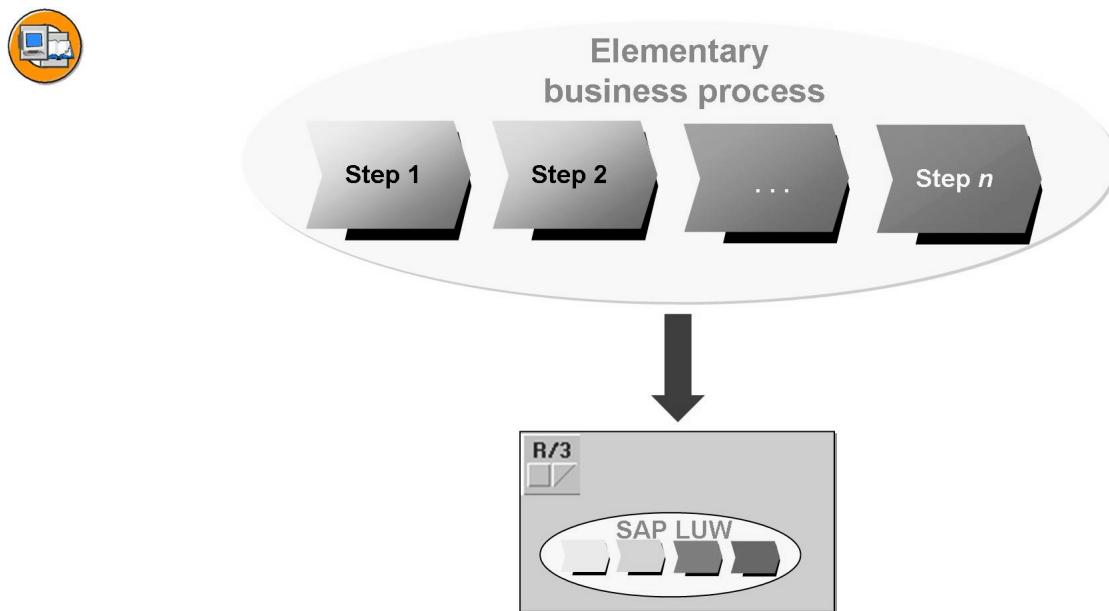


Figure 94: SAP LUW

The business process to be mapped must be basic. For example, you would not have a single SAP LUW consisting of all of the steps between a customer processing an order and an invoice being produced. Instead, you would split this up into separate parts, each of which would then be represented in the SAP System by its own LUW. What constitutes an "elementary" process depends on the overall process and how you have modeled it.

The Open SQL statements INSERT, UPDATE, MODIFY, and DELETE allow you to program database changes that extend over several dialog steps. Even if you have not explicitly programmed a database commit, the implicit database commit that occurs after a screen has been processed concludes the database LUW.

Under this procedure, you cannot roll back the database changes from previous dialog steps. It is therefore only suitable for programs in which there is no logical relationship between the individual dialog steps. Normally, however, the database changes in individual dialog steps depend on those in other dialog steps, and must therefore all be executed or rolled back together. These dependent database changes form logical units, and can be grouped into a single database LUW using the bundling techniques listed below.

A logical unit consisting of dialog steps, whose changes are written to the database in a single database LUW is called an SAP LUW. Unlike a database LUW, an SAP LUW can span several dialog steps, and be executed using a series of different work processes. If an SAP LUW contains database changes, you should either write all of them or none at all to the database.

Bundling Changes

To insure that SAP LUWs work on an all-or-nothing principle, you must include a database commit when your transaction has ended successfully and a database rollback in case the program detects an error. However, since database changes from a database LUW cannot be reversed in a subsequent database LUW, you must make all of the database changes for the SAP LUW in a single database LUW. To maintain data integrity, you must bundle all of your database changes in the final database LUW of the SAP LUW. The following diagram illustrates this principle:

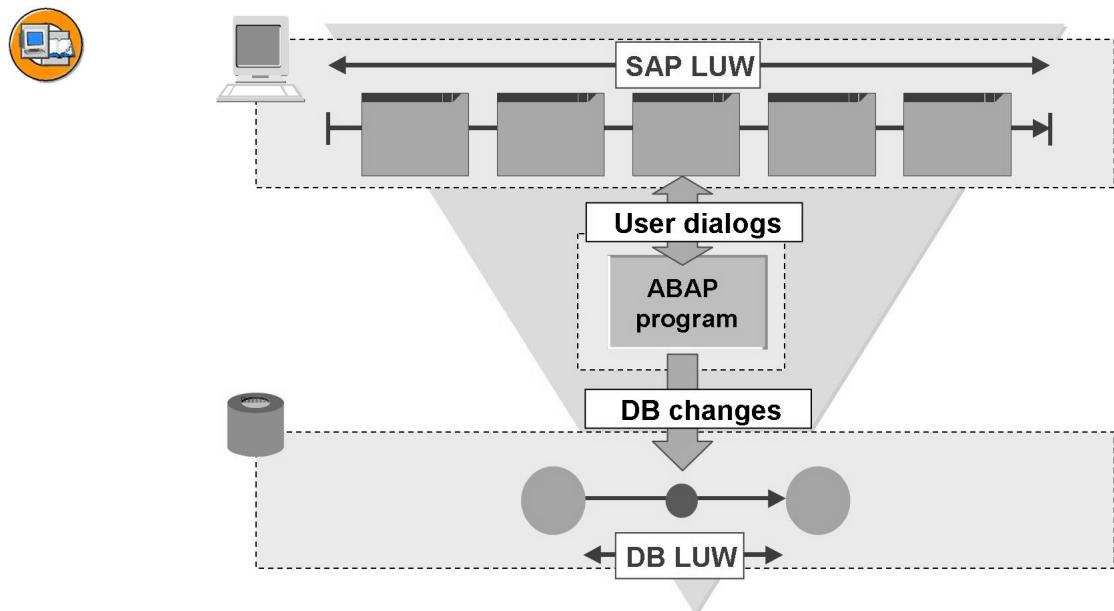


Figure 95: Bundling Changes

The bundling technique for database changes within an SAP LUW ensures that you can still reverse them. It also means that you can distribute a transaction across more than one work process, and even across more than one SAP System.

The simplest form of bundling would be to process a whole application within a single dialog step. Here, the system checks the user's input and updates the database without a database commit occurring within the dialog step itself.

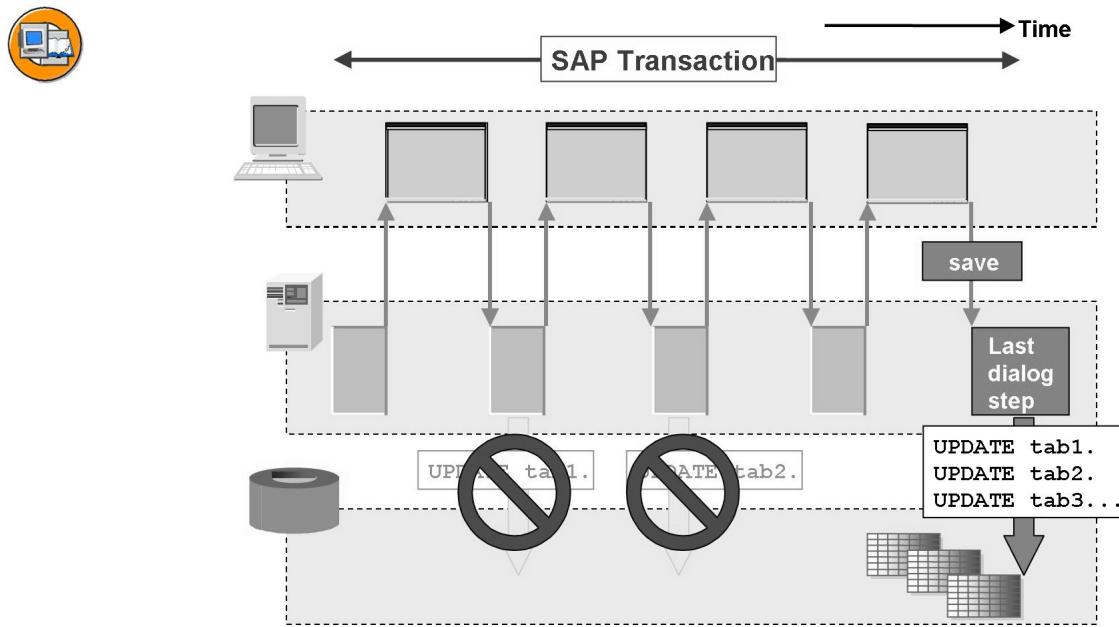


Figure 96: Direct Changes

Of course, this is not suitable for complex business processes. Instead, the SAP Basis system contains the following bundling techniques.

Bundling Using Subroutines

The statement `PERFORM ON COMMIT` calls a subroutine in the dialog work process. However, it is not executed until the system reaches the next `COMMIT WORK` statement. Here, as well, the ABAP statement `COMMIT WORK` defines the end of the SAP LUW, since all statements in a subroutine called with `PERFORM ON COMMIT` that make database changes are executed in the database LUW of the corresponding dialog step.

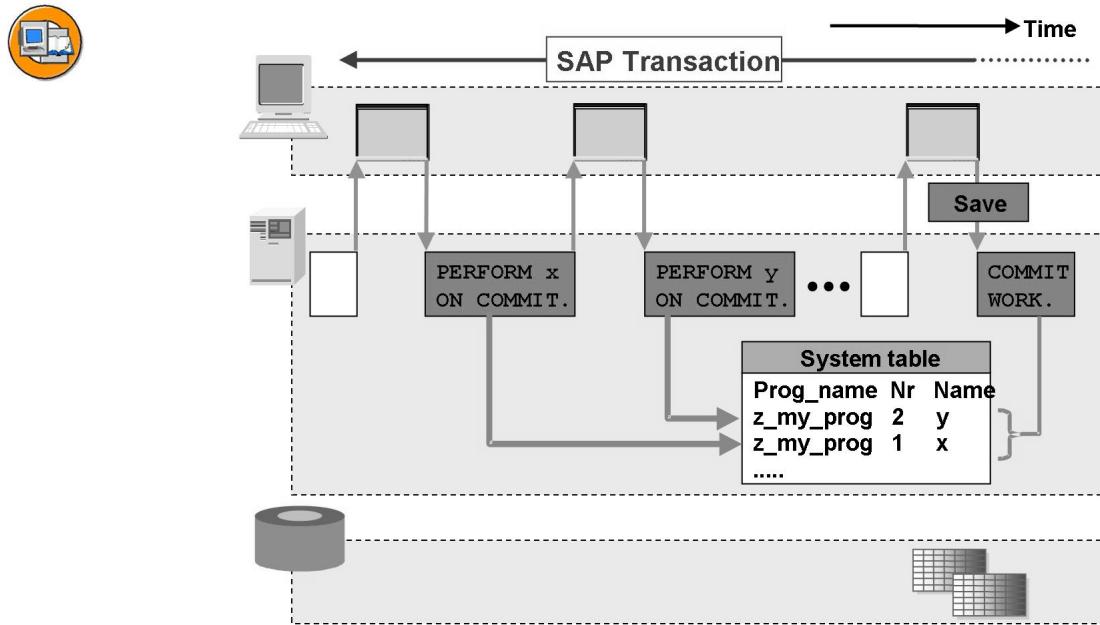


Figure 97: Perform on Commit (1)

Database updates from the dialog can be executed in bundled form by using the special subroutine technique `PERFORM <subroutine> ON COMMIT`. The statement registers the subroutine that has been called up. This subroutine will not be executed until the next `COMMIT WORK` statement is reached.

If the database updates are encapsulated in the subroutines, they can be separated from the program logic and relocated to the end of the LUW processing. Each subroutine registered with `PERFORM ON COMMIT` is only executed once per LUW. Calls can be made more than once (no errors); the subroutine, however, is only executed once.

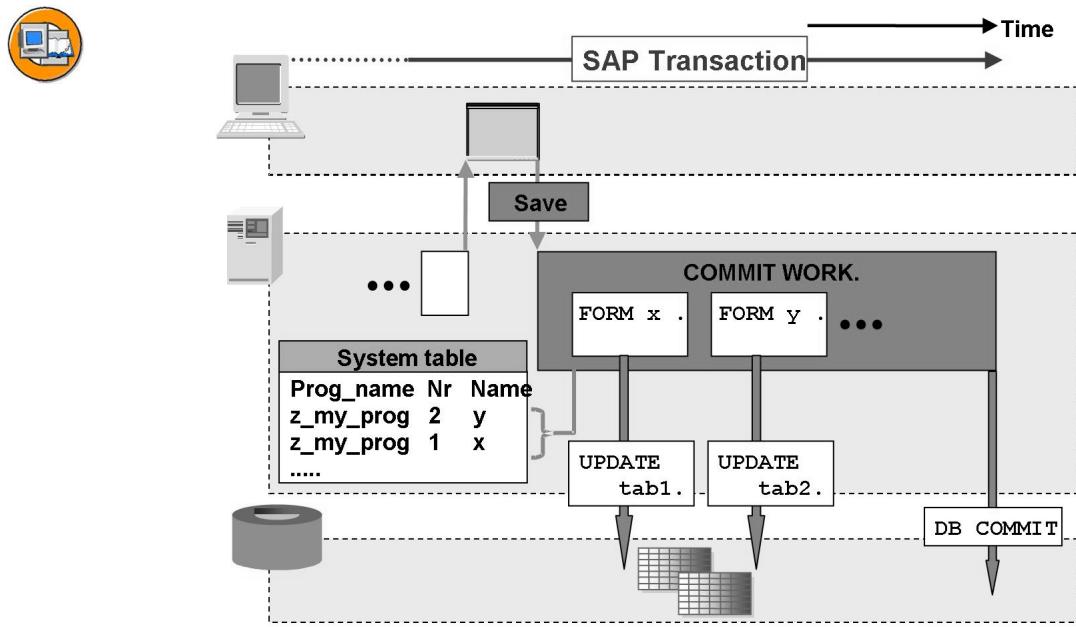


Figure 98: Perform on Commit (2)

The COMMIT WORK statement carries out all subroutines registered to be executed and triggers a database commit (ends the DB LUW).

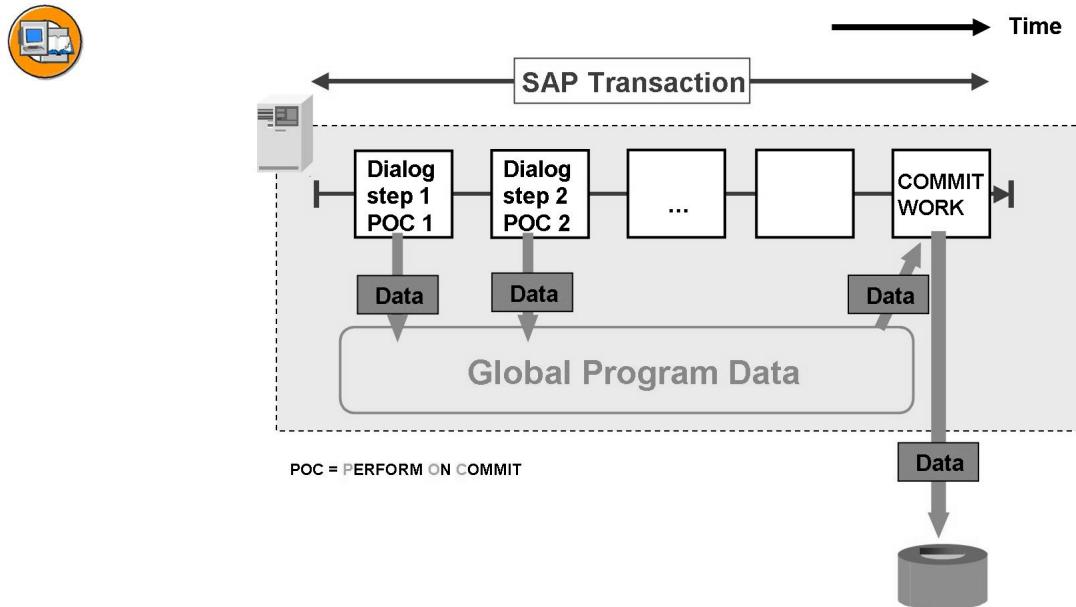


Figure 99: Perform on Commit Data Flow

Unlike normal subroutines, those that you call using the ON COMMIT addition do not have an interface. They work instead with global data, that is, the values of the data objects at the point where the subroutine is actually run. This can also include Imports from memory.

The PERFORM ... ON COMMIT technique can also be used in the update process. This will be discussed later.

Bundling using Function Modules for Updates

Update techniques allow you to separate user dialogs from the database changes. Both are executed by different programs, which generally run in different work processes.

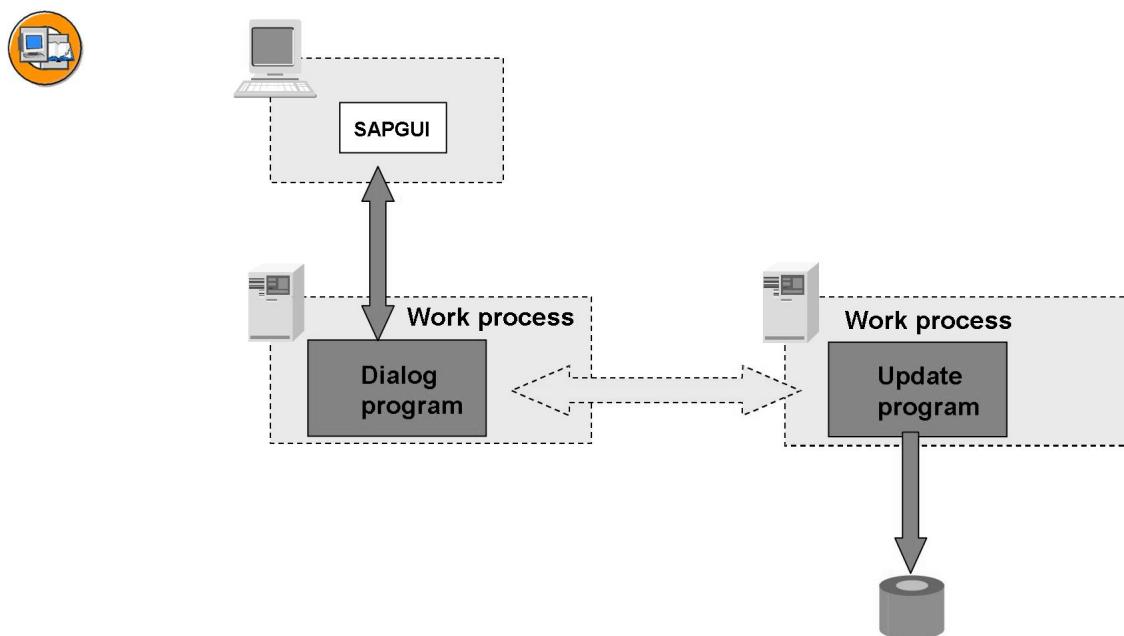


Figure 100: Principle of the Update Process

If you call a function module using the CALL FUNCTION... IN UPDATE TASK statement, the function module is flagged for execution using a special update work process. This means that you can write the Open SQL statements for the database changes in the function module instead of in your program, and call the function module at the point in the program where you would otherwise have included the statements. When you call a function module using the IN UPDATE TASK addition, it and its interface parameters are stored as a log entry in a special database table called VBLOG.

The function module is executed using an update work process when the program reaches the COMMIT WORK statement. After the COMMIT WORK statement, the dialog work process is free to receive further user input. The dialog part of the transaction finishes with the COMMIT WORK statement. The update part

of the SAP LUW then begins, and this is the responsibility of the update work process. The SAP LUW is complete once the update process has committed or rolled back all of the database changes.

Bundling Using Function Modules in Other SAP Systems

Function modules that you call using CALL FUNCTION... IN BACKGROUND TASK DESTINATION... are registered for background execution in another SAP System when the program reaches the next COMMIT WORK statement (using Remote Function Call). After the COMMIT WORK, the dialog process does not wait for these function modules to be executed (asynchronous update). All of the function modules that you register in this way are executed together in a single database LUW. These updates are useful, for example, when you need to maintain identical data in more than one database.

SAP Transaction

An SAP transaction is an application program that you start using a transaction code. It may contain one or more SAP LUWs. Whenever the system reaches a COMMIT WORK or ROLLBACK WORK statement that is not at the end of the last dialog step of the SAP transaction, it opens a new SAP LUW.

For our purposes in this lesson, we should assume that we are not going to be executing SAP transactions, as such. We should, instead, assume that we will need to use other mechanisms to implement the same logical control over the execution.



Lesson Summary

You should now be able to:

- Define the terms database LUW and SAP LUW
- Explain the need to bundle changes to the database tables
- Apply the knowledge of updating techniques to the distributed application environment and the use of BAPIs

Related Information

- For additional information on LUWs and transactions, refer to Transactions and Logical Units of Work in the Online Documentation.

Lesson: Authorizations

Lesson Overview

Not all users should have access to all data that is available simply by using the transactions and reports that they are allowed to use. However, once you have released a program, any user with authorization for it can run it and get its information. This means that the programmer is responsible for checking that the user is authorized to access the data that the program processes.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the authorization concept for BAPIs
- Describe Authorization Objects and Authorizations
- List requirements to implement authority checking

Business Example

You might want to allow users to display data for all airline carriers, but only allow them to change data for certain selected ones. In this case, the system must look for a combination of the fields 'activity' and 'airline carrier' each time it performs an authorization check.

Authorization Concept

In general terms, the concept of SAP Authorizations is no different than the security concept anywhere else. What may be the biggest difference is that we are an open system and, as such, do not control the underlying software. Since we can run our system on different operating systems, database management systems, and hardware, we cannot depend on the platform to provide consistent and flexible support for our needs, regardless of the installation considerations.

We have created our own process to address the assignment of access capabilities to SAP data. Although it is powerful and very flexible, it is still relatively simple in concept. It's important to understand that an object definition in SAP is a symbolic representation. We do not tie access to tables directly. Rather, we create a representative definition of the item and relate access definitions to it.

For example, to control the flow of traffic, many street intersections have a light indicating who has right-of-way and who should wait. There is nothing about a red glowing light bulb that makes my car stop. The reason this mechanism works is that the users (drivers) agree to check the light and to abide by the information shown.

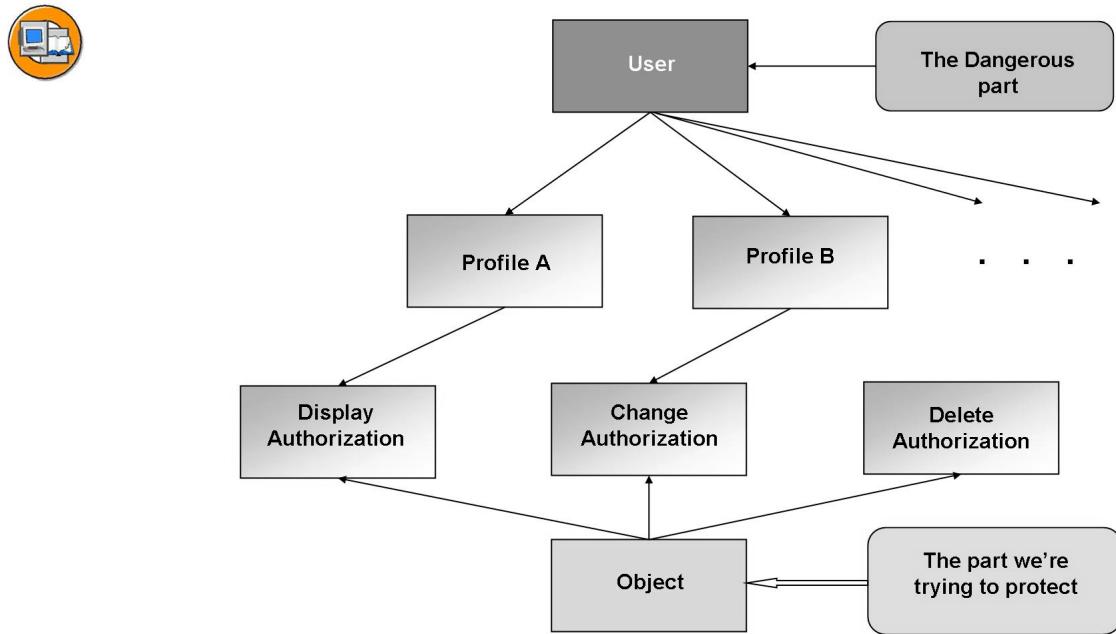


Figure 101: Authorization Concept

The authorization objects are the business items we are trying to protect from unauthorized access. Whether this access is intentional or not is irrelevant. Since permission to access the data must be granted, we have defined the sets of permissions we can assign. These sets of permissions are called authorizations.

The number of these authorizations and exactly what they entitle the user to do is a function of the security strategy of the organization and the person creating the definition. For example, in the slide above, permissions have been defined for display, change, and deletion of the object. However, there does not appear, at face value, to be any authorization to create more of them. This would be a business issue, not a technical one.

The other side of that same issue is that we must be able to assign the permissions needed to people so that they can do their jobs. In our system, a set of authorizations appropriate to a given role or function is called a profile. A given user may have one or more profiles associated with their user master record, depending on the variety of functions they perform.

From a more technical point of view, authorization objects are repository objects and are maintained in the ABAP Dictionary. They consist of a name and up to ten logically-related fields that are used in the authorization checking process. An authorization for an authorization object is a concrete set of values for the fields of an authorization object.

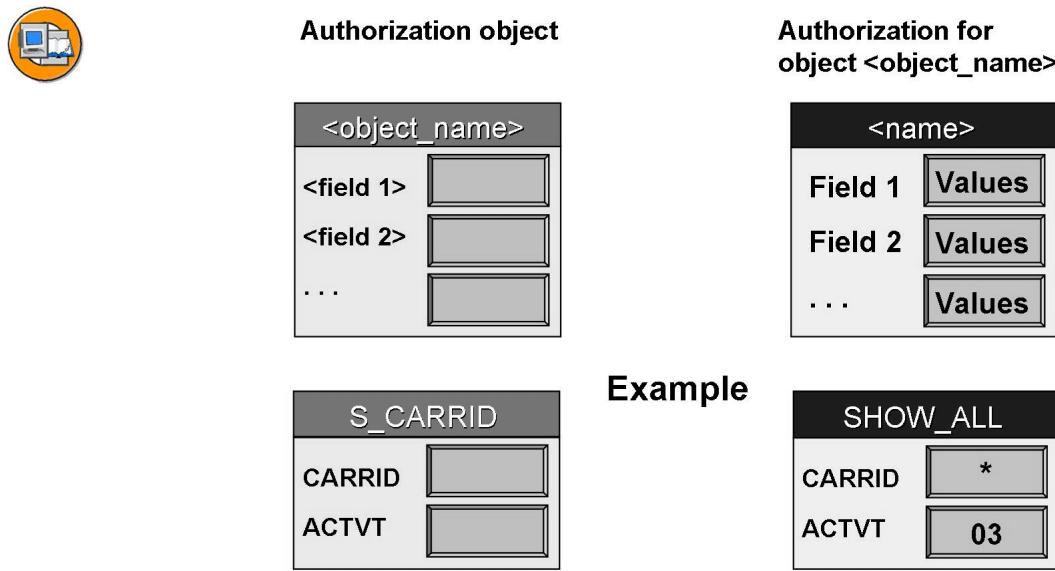


Figure 102: Authorization Objects and Authorizations

The system administrator assigns user authorization when maintaining user master data. During this process, you should determine exactly which data users are allowed to access and what kind of access should be allowed. Authorizations are grouped by profiles (business activities), which are assigned to users in their user master records.

Authorization Checks

An important consideration in the checking of access permissions in SAP is that it is an active participation process only. The system does no default checking for you in your program. This means that if your program does not request the check, it simply isn't done.

This checking process in your program is based on a single ABAP command whose keyword is AUTHORITY-CHECK.

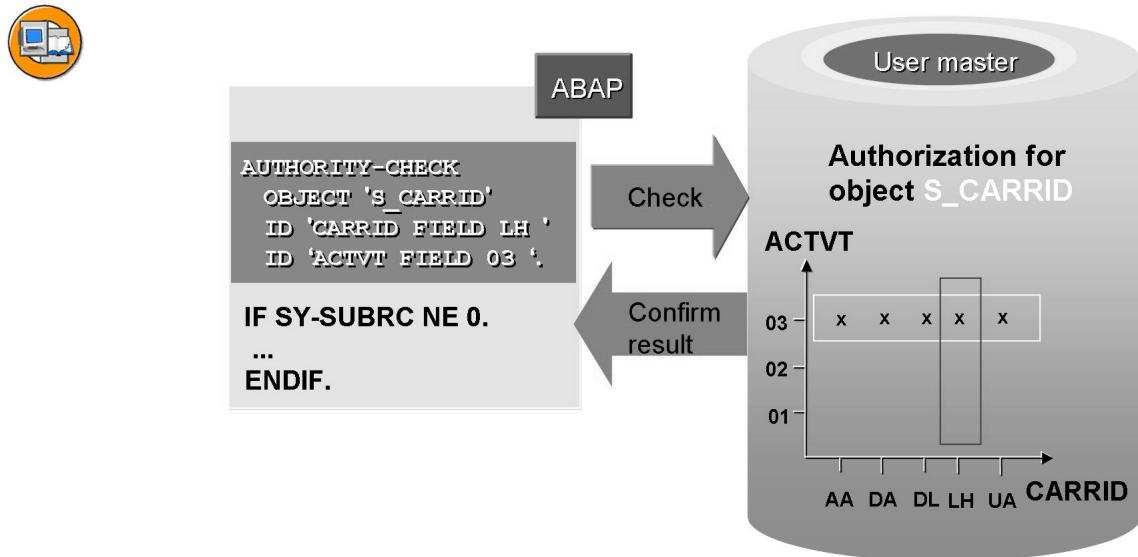


Figure 103: Performing Authorization Checks

When making authorization checks in programs, you specify the object and values the user needs in an authorization to be able to access the object. You do not have to specify the name of the authorization.

In our example, we want to check whether the user has authorization for the object S_CARRID in which the field CARRID (airline) has the value 'LH' and the field ACTVT (activity) has the value '03' for 'display'. The activity codes are listed in tables TACT and TACTZ and are also documented in the relevant authorization objects.

In the AUTHORITY CHECK, you must specify all fields of the object, otherwise, the return code will be unequal to zero. If you do not want to perform a check for one field, enter DUMMY in the field.

Important: The Authority-Check statement performs the authority check and returns an appropriate return code value in SY-SUBRC. When checking this return code, you should specify the consequences of a missing authorization (for example: terminate the program or display a message and skip some lines of code).

Use the Pattern button in the ABAP Editor to insert the AUTHORITY-CHECK command in your program. This model inserts all the names of the authorization object fields.

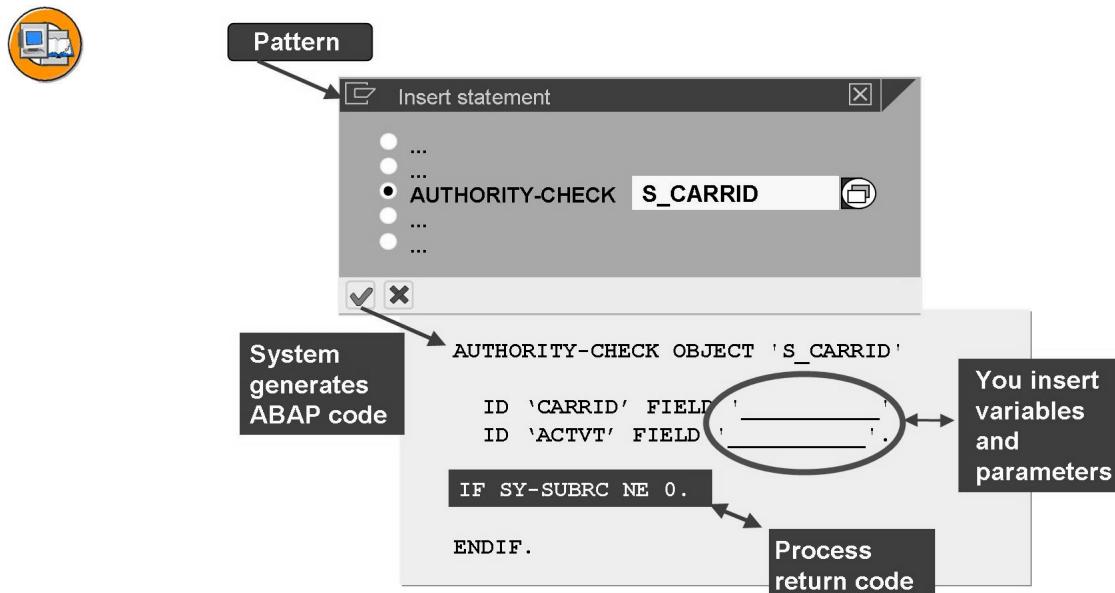


Figure 104: Inserting AUTHORITY-CHECK in Programs

If you do not use the Pattern Button or you make changes to what is inserted, it's important to remember that the names for the authorization object and the fields to be checked must be entered in single quote marks and be upper case.

The most important return codes for AUTHORITY-CHECK are:

- 0: The user has an authorization containing the required values.
- 4: The user does not have the required authorization.
- 8: The check could not successfully be carried out since not all fields of the object were specified.

The keyword documentation for AUTHORITY-CHECK contains a complete list of return codes.

You can only specify a single field after the FIELD addition, not a selection table. There are function modules which carry out the AUTHORITY-CHECK for all values in the selection table.

Example Program

Within Transaction TZ80 you can only display and change flight data if you have the appropriate authorization in your user master record for the authorization object S_CARRID.



```
MODULE USER_COMMAND_0100 INPUT.  
CASE OK_CODE.  
  WHEN 'SHOW'.  
    AUTHORITY-CHECK OBJECT 'S_CARRID'  
    ID 'CARRID' FIELD '*'  
    ID 'ACTVT' FIELD '03'.  
    IF SY-SUBRC NE 0.  
      WRITE 'No authorization'.  
    ELSE.  
      SELECT SINGLE * FROM SPFLI  
      WHERE CARRID = SPFLI-CARRID  
      AND CONNID = SPFLI-CONNID.  
      IF SY-SUBRC NE 0.  
        MESSAGE E005 WITH SPFLI-CARRID SPFLI-  
        CONNID.  
      ENDIF.  
      CLEAR OK_CODE.  
      SET SCREEN 200.
```

Can I display any airline's data

No?
Give them the error message

Only if Yes,
Retrieve the data

Figure 105: Example Code



Lesson Summary

You should now be able to:

- Describe the authorization concept for BAPIs
- Describe Authorization Objects and Authorizations
- List requirements to implement authority checking

Related Information

- For additional information on developing authorization checks and their use, refer to the topic Checking Authorizations in the ABAP Programming section of the Online Help.

Lesson: Locking

Lesson Overview

To complement the SAP LUW concept, in which bundled database changes are made in a single database LUW, the SAP System also contains a lock mechanism, fully independent of database locks, that allows you to set a lock that spans several dialog steps. These locks are known as SAP locks. They are a way of synchronizing the access to protect the consistency of your data.



Lesson Objectives

After completing this lesson, you will be able to:

- Use lock modules
- Find, maintain, and generate lock modules
- Explain the role of lock objects

Business Example

Suppose a travel agent wants to book a flight. The customer wants to fly to a particular city with a certain airline on a certain day. The booking must only be possible if there are still free places on the flight. To avoid the possibility of overbooking, the database entry corresponding to the flight must be locked against access from other transactions. This ensures that one user can find out the number of free places, make the booking, and change the number of free places without the data being changed by another transaction in the meantime.

Overview

If several users are competing to access the same table or set of tables, you need to find a way of protecting the data against concurrent access or any other use that could provide inaccurate or incomplete data. You also need a guarantee that critical data cannot be changed while you are working with the program.

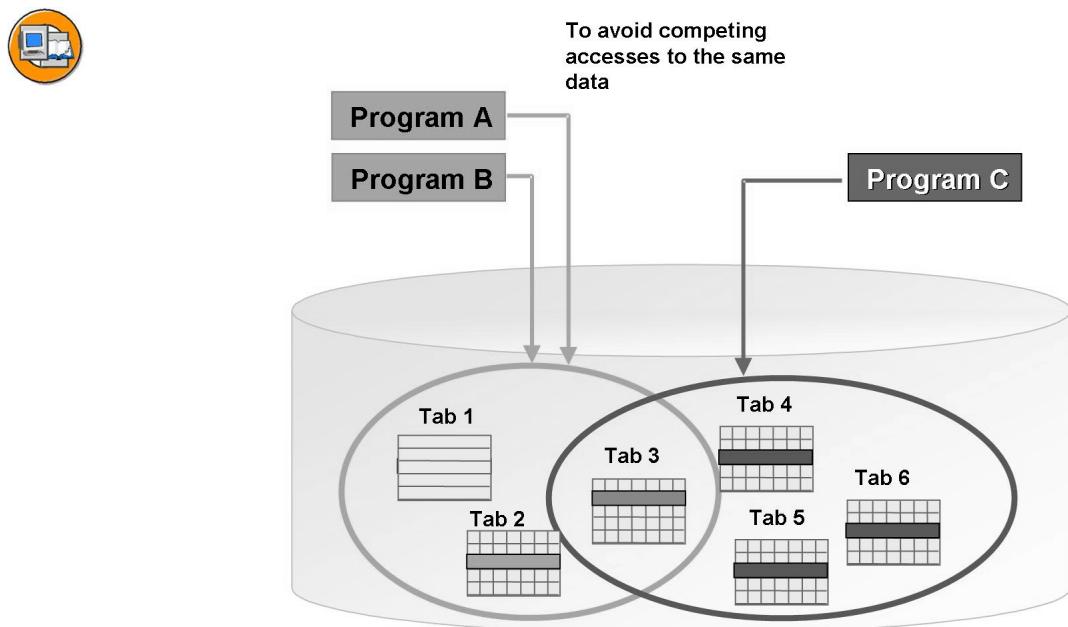


Figure 106: Why Set Locks?

Locks are a way of coordinating competing accesses to a resource. Each user requests a lock before accessing critical data. Of course, it is just as important to release the lock as soon as possible after you are finished, so as not to unnecessarily hinder other users.

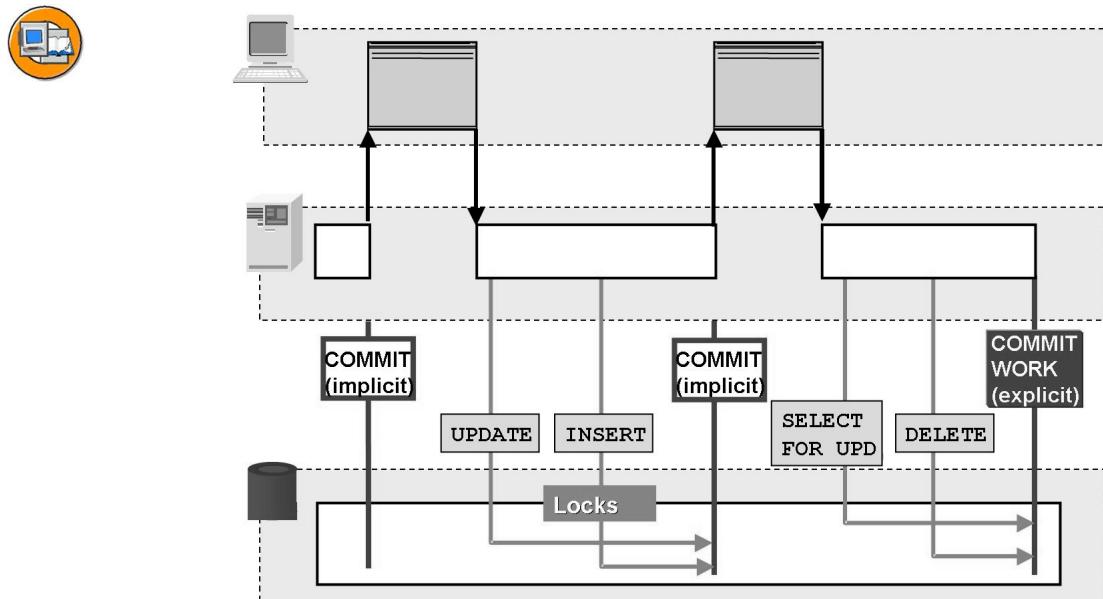


Figure 107: Database Locks Are Not Enough

Whenever you make direct changes to data on the database in a transaction, the database management system physically locks the table entries that you want to change (INSERT; UPDATE, MODIFY), and those that you read from the database and intend to change (SELECT SINGLE <field> from <dbtab> FOR UPDATE). Other users who want to access the locked record or records must wait until the physical lock has been released. In such a case, the ABAP program waits until the lock has been released again.

At the end of the database transaction, the database releases all of the locks that it has set during the transaction. In the SAP System, this means that each database lock is released when a new screen is displayed, since a change of screen triggers an implicit database commit.

To keep a lock set through a series of screens (from the dialog program to the update program), the SAP System has a global lock table at the application server level, which you can use to set logical locks for table entries.

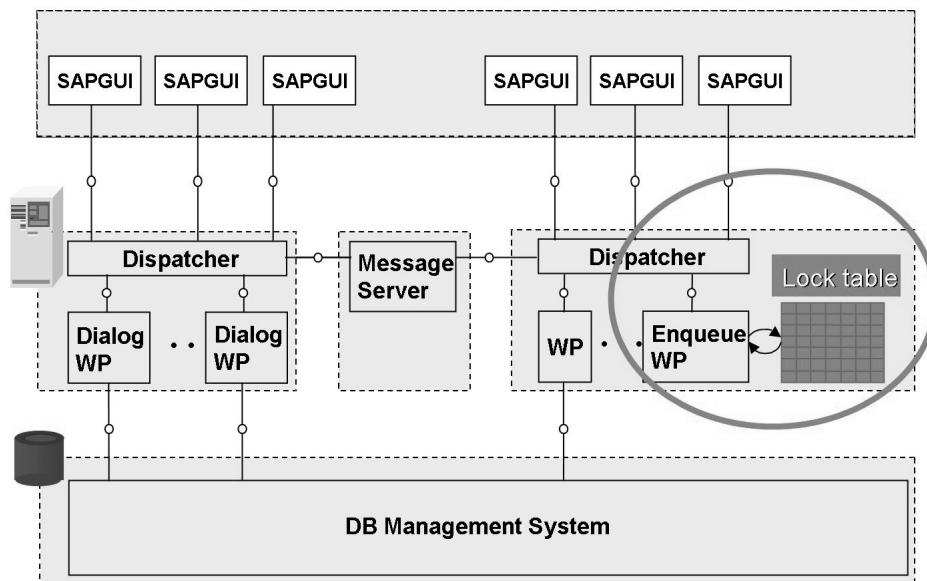


Figure 108: SAP Lock Concept: Logical Locks

One application server contains this lock table and a special enqueue work process, which administers all requests for logical locks in the SAP System. All logical lock requests of the SAP System run using this work process.

You can also use logical locks to lock table entries that do not yet exist on the database (inserting new lines). You cannot do this with physical database locks.

Setting and Releasing Locks

Coding Considerations

Logical locks are generated when an entry is written in the lock table. You use function modules to do this. You can only set a lock if the relevant table entry is not already locked.

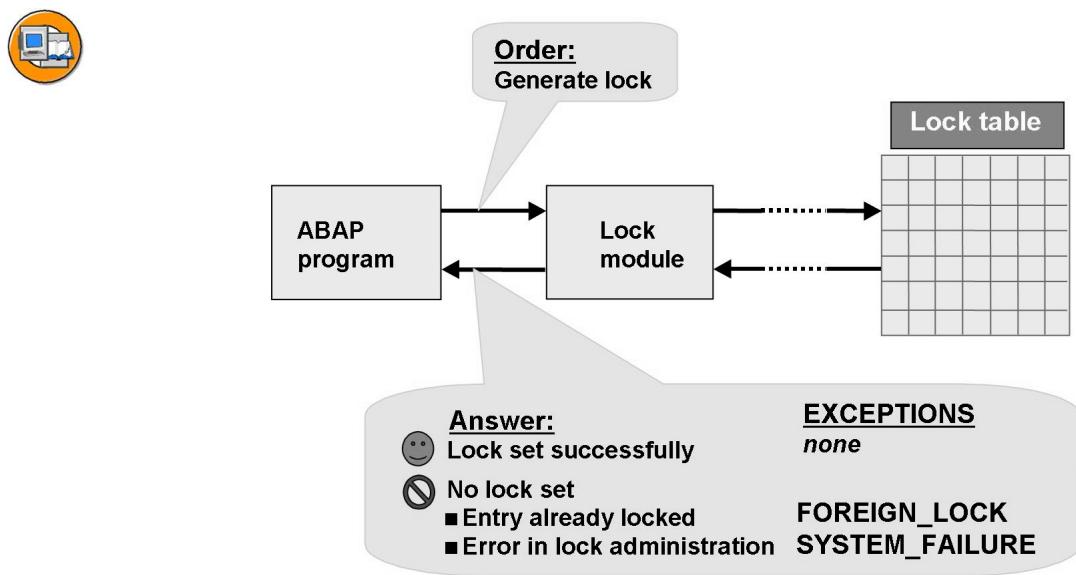


Figure 109: Setting and Releasing Logical Locks

The SAP transaction receives information on the success of a lock request from a return code sent via the EXCEPTION interface of the function module. In other words, the control is returned to the program using the function module. The SAP transaction can react appropriately by analyzing the return code. Another user cannot gain access to work with the same table entries that are already locked.

The program must delete the lock entries it generated using a lock module or have them deleted indirectly.

If the user terminates the program that generated the lock entries (usually a dialog program), the locks are released automatically (implicitly). You can do this by entering /n in the command field, or with the statements LEAVE PROGRAM, LEAVE TO TRANSACTION, and A or X messages.

When you call an ENQUEUE function module, the dialog program tries to generate a lock entry. The export parameters identify the table entry (or entries) that you want to lock.

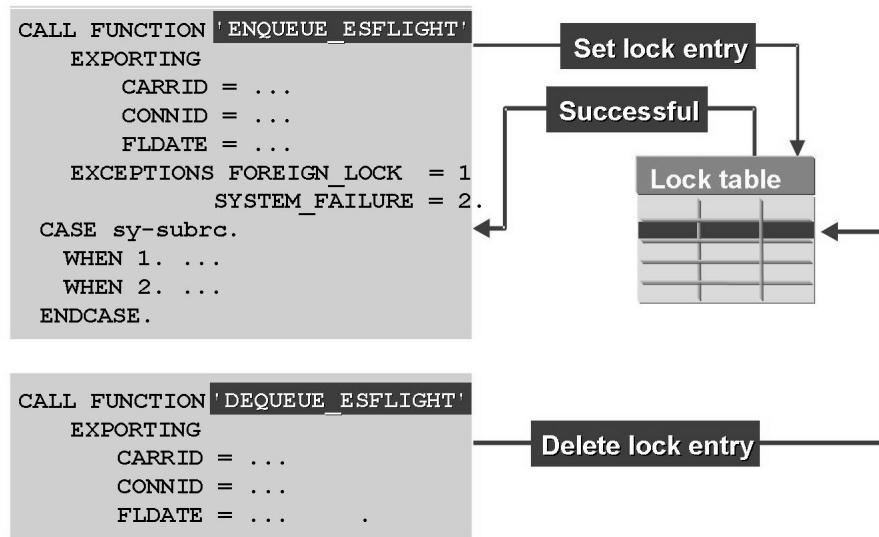


Figure 110: Calling the Lock Modules

The program that generates the locks (usually dialog program) analyzes the return code for lock requests and reacts accordingly. If the lock could not be set, you should normally output an error message.

At the end of the dialog program, you can use the corresponding DEQUEUE function module to delete the entries from the lock table. DEQUEUE function modules have no exceptions. If you try to release an entry that is not locked, this has no effect and does not generate an error. If you want to release all of the locks that you have set at the end of your dialog program, you can use the function module DEQUEUE_ALL.

System Handling of Locks

The lock table contains an entry for each logically locked table entry in this SAP System. This entry includes the user ID, the client they're working on, when it was set, and so on.

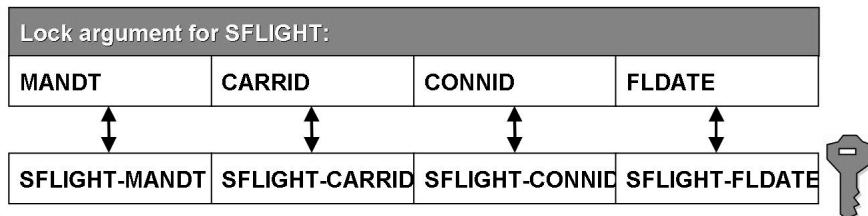


Client	User	Time	Shared Table	Lock argument	
007	Meier	11:00	SFLIGHT	LH 470	
007	Wang	11:01	SFLIGHT	UA 250	
007	Mueller	11:01	LFB1	0074712 0815	
007	Smith	11:01	LFC1	0074712 08151994	
007	Jones	11:02	X YLFA	LIEF1	
007	Marchal	11:03	X YLFA	LIEF1	
007	Bauer	11:04	YLFB1	00764715	

Figure 111: Lock Table

The information used to actually set and check a lock is called the lock argument. To display the lock table, use transaction SM12.

The entries in the lock table are standard. Locks are always set using the values of the key fields in a table. These form the lock argument. You pass the values for the lock argument to the lock modules via their interface (function module IMPORT parameters).



Examples for table SFLIGHT:

Call	Lock argument	Result
800 LH 0400 20000101	800LH 040020000101	Flight LH 400 on Jan 01, 2000 in client 800
800 LH 0400	800LH 0400#####	All flights for LH 400 in client 800
800 LH	800LH #####	All flights for LH in client 800

Figure 112: Lock Arguments



If you fail to set any of these parameters, the system interprets it generically, that is, the lock is set for all table lines that meet the criteria specified in the other parameters. The client parameter is an exception to this rule, where the default client SY-MANDT applies.

Lock entries must be assigned to a lock mode. This is the indicator that tells the system how requests from other user sessions are to be handled.

Lock mode	Description	Meaning
E	Write lock (extension)	Data will be changed
S	Read lock (shared)	Data cannot be changed while other users have access, data itself is not changed
X	Extended write lock	Data will be changed, lock may only be set once

Figure 113: Lock Mode

There are three different lock modes:

1. Mode E for write locks: This is set if you want to write data to the database (change, create, or delete).

Example: You want to book a seat for a flight. Once you have chosen the flight you want to book, you should ensure that no other customer books the same flight, to prevent the last free seat from being occupied more than once. (Technically speaking, you must lock the flight in the SFLIGHT table → SEATSOCC field = number of occupied seats).

2. Mode S for read locks: This is set if you want to ensure that the data, which you are reading from the database in your program, is not changed by other users while the program is running. You do not want to change the data itself in your program.

Example: You are a travel agent and quote a customer the price for a flight that he or she is considering booking. While the customer is considering whether to buy the flight, you want to ensure that the price is not changed by another employee.

3. Mode X for write locks: Like mode E, mode X is used for writing data to the database. The technical difference between mode X and mode E is that locks of mode X are not accumulated while a program is being executed.

The effect of lock mode processing can be shown as follows:

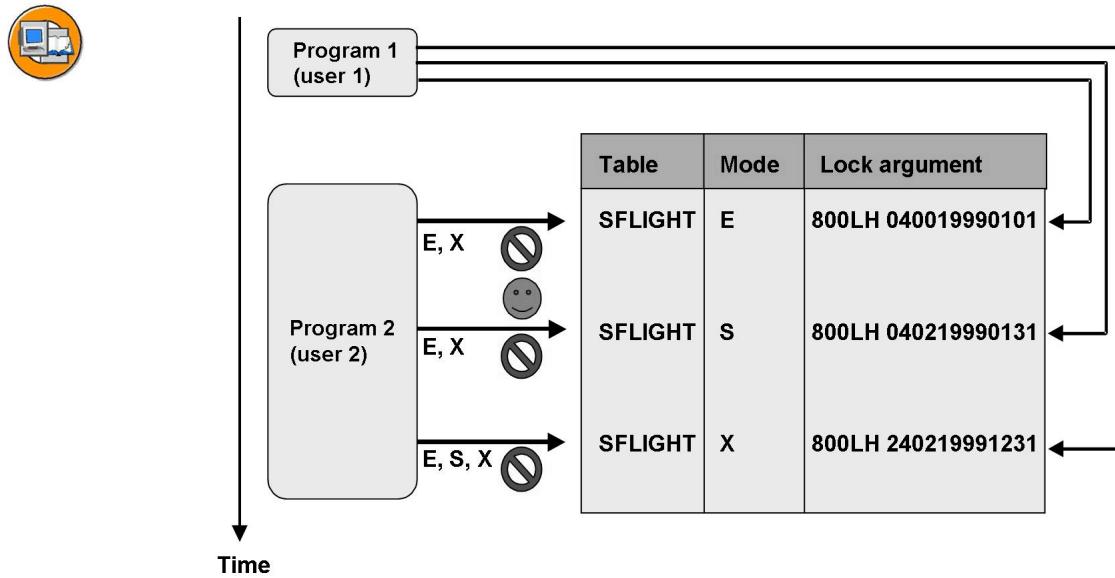


Figure 114: Using the Lock Mode: Other Users

If someone tries to lock the same data record again with a second program (different user), the various lock modes take effect as follows:

- Write locks (E or X) mean that any lock attempts from other users are refused, irrespective of the mode in which the lock is attempted.
- If a data record is locked in mode S (shared), further locks in mode S may be set by other users. Lock attempts in other lock modes (E or X) are refused.

It is possible for a given user to request an additional lock during the execution.

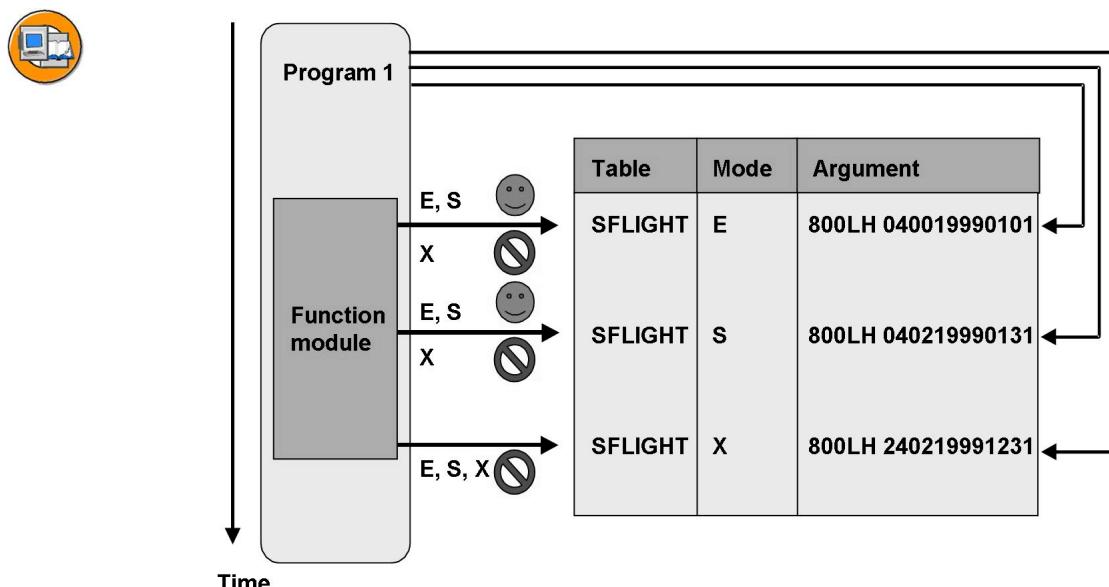


Figure 115: Using the Lock Mode: Same Program

If you want to try to lock a data record more than once while a program is running (for example, using a function module that you call up, which sets locks itself), the lock system reacts in the following way:

- Mode E write locks are not refused. Instead, a cumulative counter is incremented. The same applies to read locks (mode S).
- If a data record is locked in mode E, a lock request generates a second lock, which is marked as a read lock.
- If a data record is locked in mode S and no further read locks are set by other users, a lock attempt in mode E is possible. This generates a second entry in the lock table (for mode E).
- If a data record is locked in mode X, all further lock requests are refused.

The sequence of events is important for the effective use of locks.

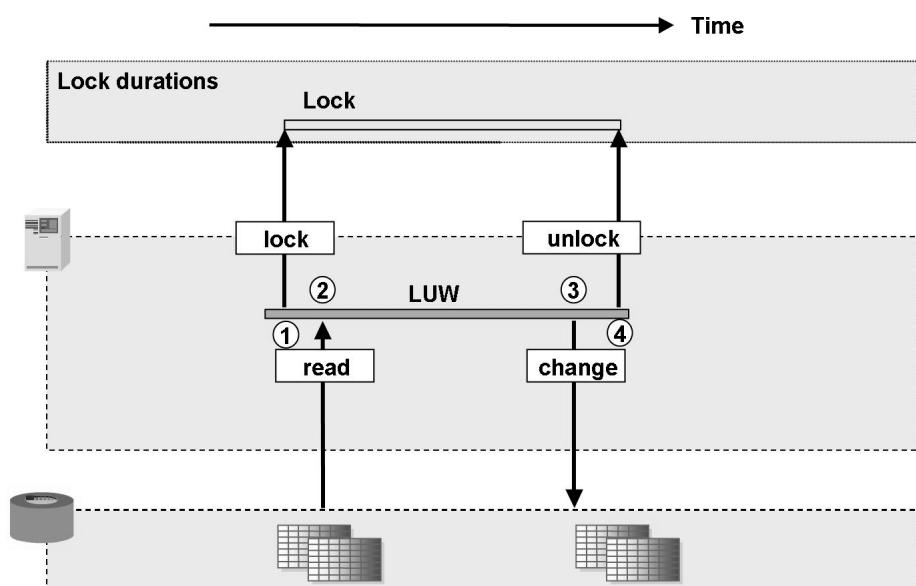


Figure 116: Locks: Timescale

If you want to ensure that you are reading up-to-date data in your program (with the intention of changing and returning this to the database), you should use the following procedure for lock requests and database accesses in your program:

1. Lock the data that you want to edit.
2. Read the current data from the database.
3. Process (change) the data in your program and write this to the database.
4. Release the locks that you set at the beginning.

This procedure ensures that your changes run fully with lock protection and that you only read data that has been changed consistently by other programs (provided that these also use the SAP lock concept and follow the procedure described here).

This procedure has the added benefit of protecting you from wasting resources to retrieve data that you cannot use later. If you retrieve data from the database before you set a lock, another user might lock that entry and change it before you can act. This would render your data obsolete and require another access to refresh it.

Lock Objects

A lock object is a logical object composed of a list of tables that are linked by foreign key relationships. Lock modules are generated for these objects and enable common lock entries to be set for all tables contained in the lock object. This allows combinations of table entries to be locked. Lock modules are created for a lock objects and not for the tables that are in it.

Creating Lock Objects

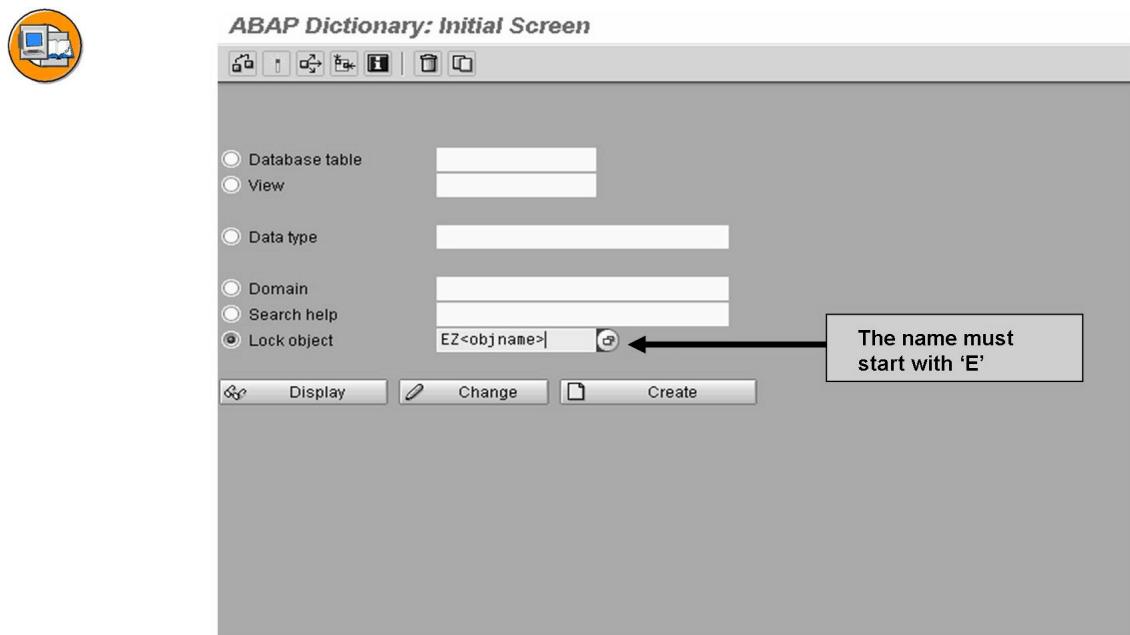


Figure 117: Select Lock Object Maintenance

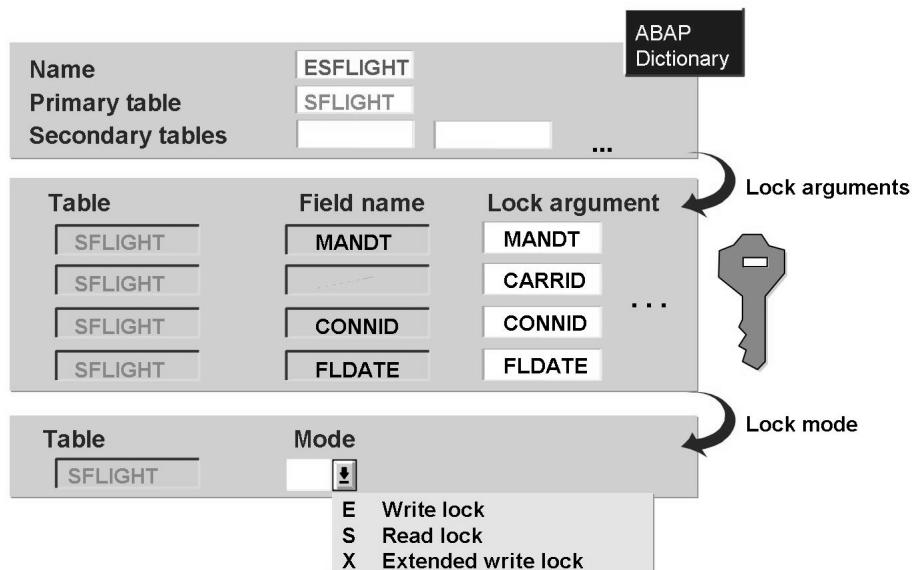


Figure 118: SAP Lock Objects

Lock objects are easy to define using the ABAP Dictionary. The procedure for creating them is as follows:

1. Select object type Lock object in the initial screen of the ABAP Dictionary, enter an object name, and choose Create. The name of a lock object should begin with an E (Enqueue).

The maintenance screen for lock objects displays.

2. Enter an explanatory short text in the field *Short text*.

You can then use the short text to find the lock object at a later time, for example with the SAP Repository Information System.

3. Enter the name of the primary table of the lock object.

All other tables in the lock object must be linked with the primary table using foreign keys. There are also some restrictions on the valid foreign key relationships.

4. Select the lock mode of the primary table in the field below it.

The lock mode is used as the default value for the corresponding parameters of the function modules generated from the lock object.

5. Choose Add if you want to lock records in more than one table with the lock object.

A list of all the tables linked with the primary table using valid foreign keys displays. Select the appropriate table. The lock mode of the primary table is copied as lock mode. You can change this setting as required, for example, you can assign the lock mode separately for each table.

Similarly, you can add a table linked with the secondary table just added with foreign keys. To do this, place the cursor on the name of the secondary table and choose *Add*.

 **Note:** If no lock mode is assigned to a table, no lock is set for the entries in this table when the generated function modules are called. You should not assign a lock mode if a secondary table was only used to define a path between the primary table and another secondary table with foreign keys.

6. Save your entries.

A dialog box appears in which you have to assign the lock object a Package.

7. You can (optionally) exclude lock parameters from the function module generation on the *Lock parameter* tab page. This makes sense if you always want to lock a parameter generically.

To do this, simply deselect the *Weight flag* for the parameter. The parameter is not taken into consideration in the generated function modules. This parameter is then always locked generically.

The name of a lock parameter is usually the name of the corresponding table field. If two fields with the same name are used as lock parameters in the lock object from different tables, you must choose a new name for one of the fields in *Lock parameter* field.

8. You can define whether the function modules generated from the lock object should be RFC-enabled on the *Attributes* tab page.

If you set the Allow RFC flag, the generated function modules can be called from within another system with Remote Function Call.

If you permit Remote Function Calls for an existing lock object, you must ensure that the generated function modules are called from within an ABAP program with parameters appropriate for the type. You should therefore check all programs that use the associated function modules before activating the lock object with the new option.

9. Choose *Activate*.

When you activate the lock object, the two function modules ENQUEUE_<lockobjectname> and DEQUEUE_<lockobjectname> are generated from its definition to set and release locks.

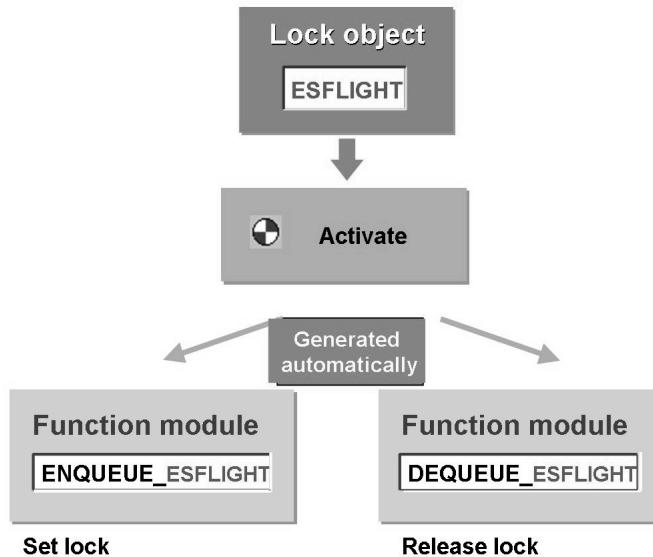


Figure 119: Generating Lock Modules

You can find information about the activation flow in the activation log, which you can display with *Utilities* → *Activation log*. If errors occurred during activation, the activation log displays immediately.

Deleting Lock Objects

When you delete a lock object, the function modules generated when you activated the lock object are automatically deleted as well. These generated function modules might still be in use in programs or classes. Therefore, before deleting a lock object, find all programs or classes that contain these function modules and remove the calls to the function modules.

The procedure for deletion is:

1. In the initial screen of the ABAP Dictionary, select object type Lock object and enter the lock object name.

Choose the *Where-used* button to find all the programs or classes that are still using the lock object. Remove the lock module calls in the objects you found.

2. Choose *Delete*.

A dialog box appears in which you must confirm the deletion request. If the function modules belonging to the lock object are still in use in programs or classes, a corresponding warning appears. In this case, you must adjust the programs or classes affected before deleting the lock object.

3. Confirm the deletion request.

The lock object is deleted together with the function modules generated from this lock object.

Function Modules for Lock Requests



Caution: The generated function modules are automatically assigned to function groups. You should not change these function modules and their assignment to function groups since the function modules are generated again each time the lock object is activated.

Never transport the function groups, which contain the automatically generated function modules. The generated function modules of a lock object could reside in a different function group in the target system.

Always transport the lock objects. When a lock object is activated in the target system, the function modules are generated again and correctly assigned to function groups.

Parameter Usage

Field names

The keys to be locked must be passed here.

A further parameter `X_<field>` that defines the lock behavior when the initial value is passed exists for every lock field `<field>`. If the initial value is assigned to `<field>` and `X_<field>`, then a generic lock is initialized with respect to `<field>`. If `<field>` is assigned the initial value and `X_<field>` is defined as `X`, the lock is set with exactly the initial value of `<field>`.

Parameter for Passing Locks

A lock is generally removed at the end of the transaction or when the corresponding DEQUEUE function module is called. However, this is not the case if the transaction has called update routines. In this case, a parameter must check that the lock has been removed.

Parameter `_SCOPE` controls how the lock or lock release passes to the update program. You have the following options:

- `_SCOPE = 1`: Locks and lock releases are not passed to the update program. The lock is removed when the transaction ends.
- `_SCOPE = 2`: The lock or lock release passes to the update program. The update program is responsible for removing the lock. The interactive program with which the lock was requested no longer has an influence on the lock behavior. This is the standard setting for the ENQUEUE function module.
- `_SCOPE = 3`: The lock or lock release also passes to the update program. The lock must be removed in both the interactive program and in the update program. This is the standard setting for the DEQUEUE function module.

Parameter for Lock Mode

A parameter MODE_<tab> exists for each base table TAB of the lock object. The lock mode for this base table can be set dynamically with this parameter. Valid values for this parameter are S (shared), E (exclusive) and X (exclusive but not cumulative). The lock mode specified when the lock object for the table is created is the default value for this parameter. This default value can, however, be overridden as required when the function module is called.

If a lock set with a lock mode is to be removed by calling the DEQUEUE function module, this call must have the same value for the parameter MODE_<tab>.

Controlling Lock Transmission

Parameter _COLLECT controls whether the lock request or lock release should be performed directly or whether it should first be written to the local lock container. This parameter can have the following values:

- Initial value: The lock request or lock release is sent directly to the lock server.
- X : The lock request or lock release is placed in the local lock container. The lock requests and lock releases collected in this lock container can then be sent to the lock server at a later time as a group by calling the function module FLUSH_ENQUEUE.

Behavior for Lock Conflicts (ENQUEUE only)

The ENQUEUE function module also has the parameter _WAIT. This parameter determines the lock behavior when there is a lock conflict. You have the following options:

- Initial value: If a lock attempt fails because there is a competing lock, the exception FOREIGN_LOCK is triggered.
- X : If a lock attempt fails because there is a competing lock, the lock attempt is repeated after waiting for a certain time. The exception FOREIGN_LOCK is triggered only if a certain time limit has elapsed since the first lock attempt. The waiting time and the time limit are defined by profile parameters.

Controlling Deletion of the Lock Entry (DEQUEUE only)

The DEQUEUE function module also has the parameter _SYNCHRON. If X is passed, the DEQUEUE function waits until the entry has been removed from the lock table. Otherwise, it is deleted asynchronously, that is, if the lock table of the system is read directly after the lock is removed, the entry in the lock table may still exist.

Exceptions of the ENQUEUE Function Module

The exceptions on enqueue processing are:

- FOREIGN_LOCK: A competing lock already exists. You can find out the name of the user holding the lock by looking at system variable SY-MSGV1.
- SYSTEM_FAILURE: This exception triggers when the lock server reports that a problem occurred while setting the lock. In this case, the lock could not be set.

If the exceptions are not processed by the calling program itself, appropriate messages are issued for all exceptions.

Reference Fields for RFC-Enabled Lock Objects

The type of an RFC-enabled function module must be completely defined. The parameters of the generated function module therefore have the following reference fields for RFC-enabled lock objects:

Parameters	Reference fields
X_<field name>	DDENQ_LIKE-XPARFLAG
_WAIT	DDENQ_LIKE-WAITFLAG
_SCOPE	DDENQ_LIKE-SCOPE
_SYNCHRON	DDENQ_LIKE-SYNCHRON



Lesson Summary

You should now be able to:

- Use lock modules
- Find, maintain, and generate lock modules
- Explain the role of lock objects

Related Information

- For additional information on developing Lock Objects and their use, refer to the topic Lock Objects in the ABAP Programming section of the Online Help.

Lesson: Update Techniques

Lesson Overview

Within the context of the transaction model used to develop BAPIs, a transaction represents one processing step or one logical unit of work (LUW). An LUW is all the steps involved in a transaction including updating the database. When a transaction is called, database operations are either fully executed or not at all. Either all relevant data has to be changed in the database or none at all.

It is up to the programmer to make sure the all-or-nothing principle is implemented. To do this, you must know and understand the available options.



Lesson Objectives

After completing this lesson, you will be able to:

- Explain the different update techniques available
- Perform bundled database updates
- Create and use update function modules

Business Example

Users in the field need to update contact information in an SAP System.

Overview

For elementary, straightforward transactions in SAP System, the update process used for data can be very straightforward as well. The concept of the work to be done is simple and the transaction can be implemented quickly.

Since this process treats all the work as a single step, there are functional considerations that must be addressed:

- The dialog work process is not released
- User has to wait until the updates are finished
- Errors are not logged
- Many small DB updates = poor DB performance
- LUW and SAP Locking not supported by system functions

This technique is useful, then, for programs without any special LUW requirements, such as the ABAP Editor or any transaction where each screen change is, in fact, standalone.

This is not the case with the use of BAPIs as part of a (potentially) distributed transaction. In order to adhere to the ACID principle and the All-or-Nothing principle, BAPI processing requires the following:

- Database updates are always done via update processes
- No COMMIT WORK / ROLLBACK WORK in the BAPI
- BAPI-LUW is closed/rejected by calling a service BAPI
 - BapiService.TransactionCommit
 - BapiService.TransactionRollback

In order to address these issues, we need to understand the implications of the techniques available. We will, therefore, discuss those techniques and how to apply them to the definition and use of our function modules.

Process

As a review, update techniques allow you to separate user dialogs from the database changes. Both are executed by different programs, which generally run in different work processes.

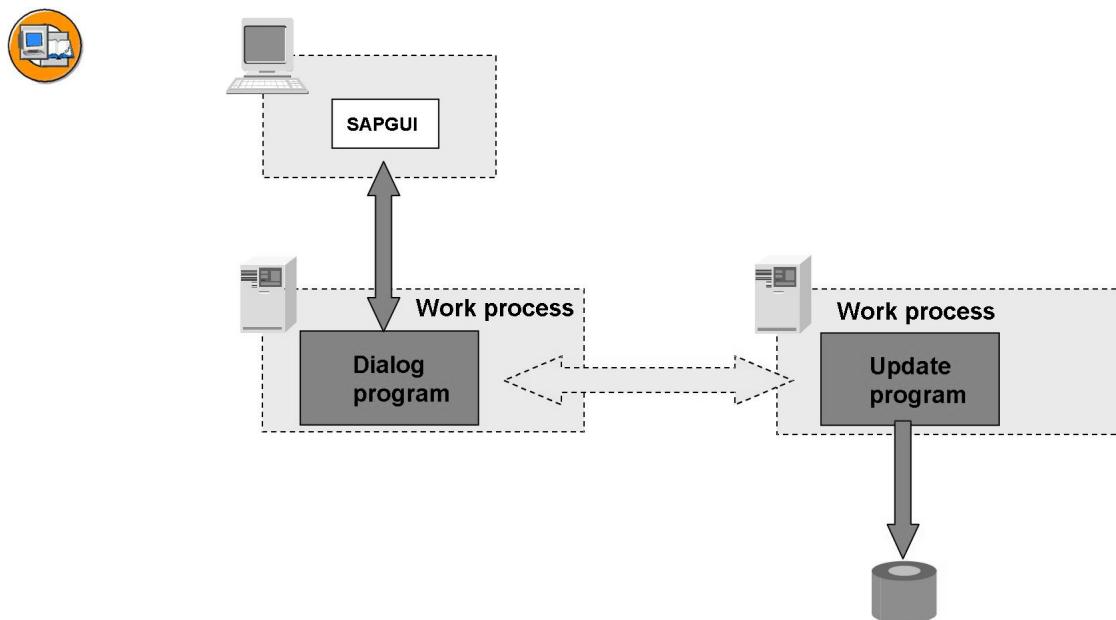


Figure 120: Update: Principle (Reprise)

If you call a function module using the CALL FUNCTION... IN UPDATE TASK statement, the function module is flagged for execution using a special update work process. This means that you can write the Open SQL statements for the database changes in the function module instead of in your program, and call the function module at the point in the program where you would otherwise have

included the statements. When you call a function module using the IN UPDATE TASK addition, it and its interface parameters are stored as a log entry in a special database table called VBLOG.



Use of Update Task

- Use of Update Task (CALL FUNCTION ... IN UPDATE TASK) is mandatory.
- Direct Changes or PERFORM ... ON COMMIT are prohibited.
- A BAPI must not trigger further LUWs. Therefore don't use:
 - CALL TRANSACTION
 - SUBMIT <report> [AND RETURN]

Figure 121: Calling a function module asynchronously

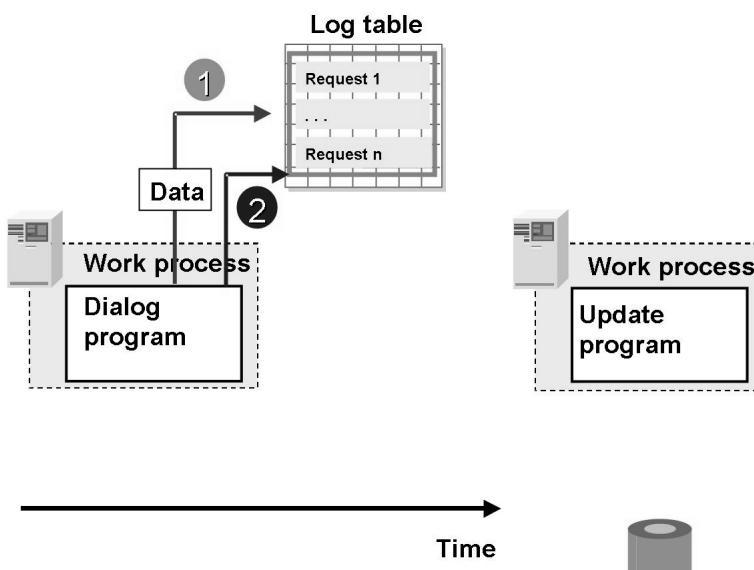


Figure 122: Writing and Completing Requests

Step 1: The dialog program receives the data changed by the user and writes it to a special log table. The entries in this table function as requests. The data contained in the log table will be written to the database later by the update program. A dialog program can write several entries to the log table.

The entries in the log table represent an LUW, in other words they will either be executed on the database together or not at all (all-or-nothing principle).

Step 2: The dialog program completes the logical data packet that was written to the log table. The SAP LUW finishes in the dialog part and informs the Basis system that a packet needs to be updated.

The function module is executed using an update work process when the program reaches the COMMIT WORK statement. After the COMMIT WORK statement, the dialog work process is free to receive further user input. The dialog part of the transaction finishes with the COMMIT WORK statement. The update part of the SAP LUW then begins, and this is the responsibility of the update work process. The SAP LUW is complete once the update process has committed or rolled back all of the database changes.

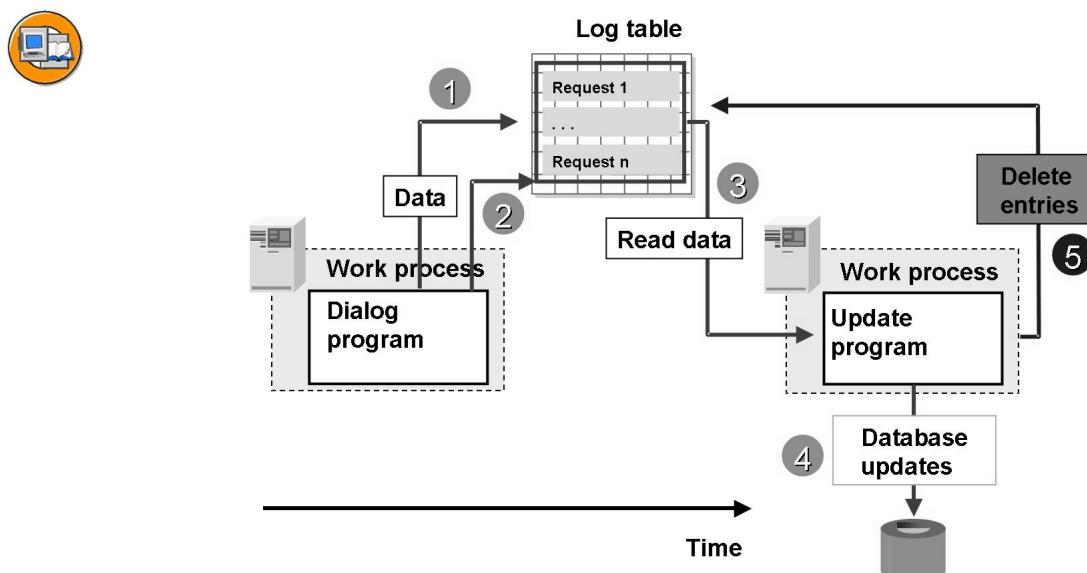


Figure 123: Processing the Requests

Step 3: A basis program reads the data associated with the LUW from the log table and supplies it to the update program.

Step 4: The update program accepts the data transferred to it and updates the database entries.

Step 5: If the update program runs successfully, a Basis program deletes all entries for the LUW from the log table.

During the update, errors only occur in exceptional cases, since the system checks for all logical errors, such as incorrect entries, in the dialog phase of the SAP LUW. If a logical error occurs, the program can terminate the update using the ROLLBACK WORK statement. Then, the function modules are not called, and the log entry is deleted from table VBLOG. Errors during the update itself are usually technical, for example, memory shortage. If a technical error occurs, the update work process triggers a database rollback, and places the log entry back into VBLOG. It then sends a mail to the user whose dialog originally generated the

VBLOG entry with details of the termination. These errors must be corrected by the system administrator. After this, the returned VBLOG entries can be processed again. The monitor transaction for update orders is SM13.

Process Flow

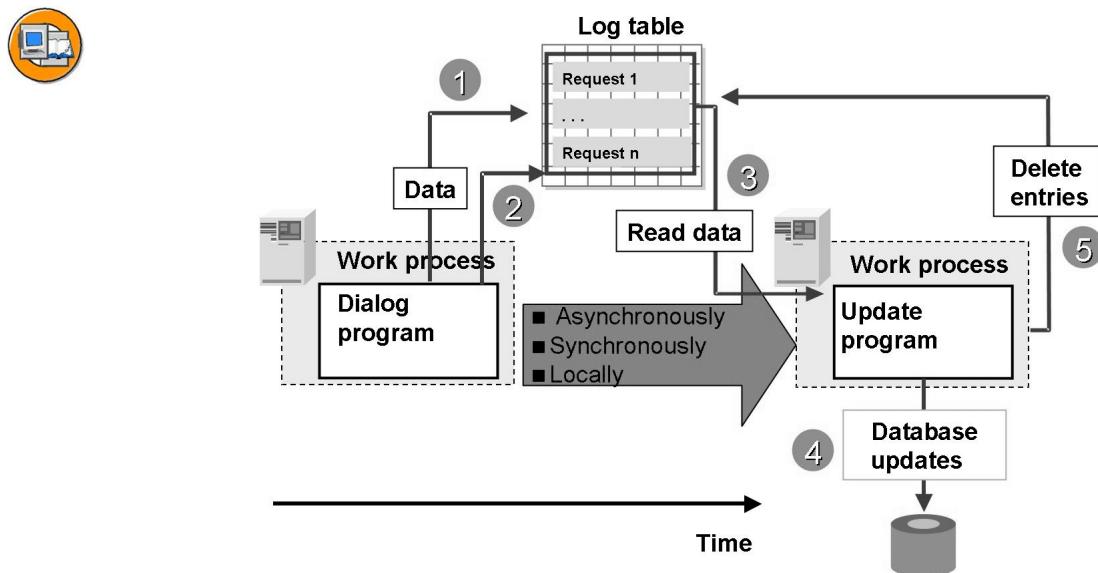


Figure 124: Process: Flow

The dialog program and the update program can be linked in various ways:

- Asynchronously
- Synchronously
- Via a local update

Asynchronous Update

A typical SAP System installation contains dialog work processes and at least one update work process. The update work processes are responsible for updating the database. When an ABAP program reaches a COMMIT WORK statement, any function modules from CALL FUNCTION... IN UPDATE TASK statements are released for processing in an update work process. The dialog process does not wait for the update to finish. This kind of update is called asynchronous update.

Asynchronous update is useful when response time from the transaction is critical, and the database updates themselves are so complex that they justify the extra system load of logging them in VBLOG. If you are running a transaction in a background work process, asynchronous update offers no advantages.

Synchronous Update

In synchronous update, you do not submit an update request using CALL FUNCTION... IN UPDATE TASK. Instead, you use the ABAP statement COMMIT WORK AND WAIT. When the update is finished, control passes back to the program. Synchronous update works in the same way as bundling update requests in a subroutine (PERFORM ON COMMIT). This kind of update is useful when you want to use both asynchronous and synchronous processing without having to program the bundles in two separate ways.

Local Update

In a local update, the update program is run by the same work process that processed the request. The dialog user has to wait for the update to finish before entering further data. This kind of update is useful when you want to reduce the amount of access to the database. The disadvantage of local updates is their parallel nature. The updates can be processed by many different work processes, unlike asynchronous or synchronous update, where the update is serialized due to the fact that there are fewer update work processes (and maybe only one).

Technical Implementation

Technical implementation of the update concept requires an update program as well as the program that manages the user dialog. The update program tasks are carried out by special function modules called update modules.

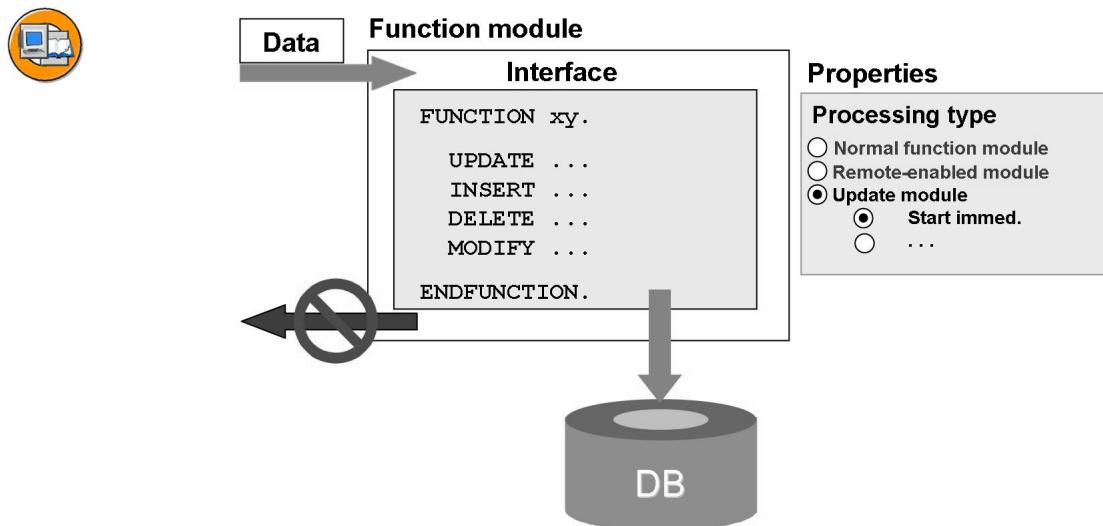


Figure 125: Update Modules

Definition considerations for update modules are:

- Choose the processing radiobutton property 'update module'.
- Update modules, like other function modules, have an interface for transferring data. The interface for update function modules only includes IMPORTING and TABLES parameters. These must be typed using reference fields or structures.
- Export parameters and exceptions are ignored in update modules.
- The function module contains the actual database update statements.

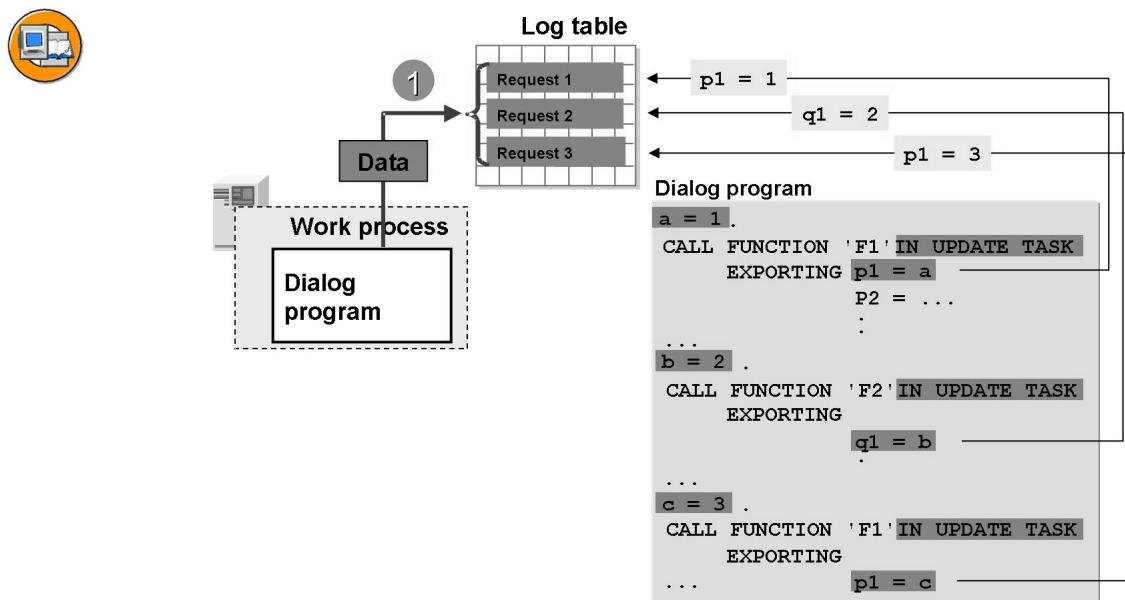


Figure 126: Writing Requests

The entries in the log file are generated from the dialog program. The function module must be called using the addition IN UPDATE TASK. This ensures that the module is not executed immediately. Instead, the current data from the function module interface is written to the log table.

For every CALL FUNCTION ... IN UPDATE TASK statement in the dialog program, the system generates an entry in the log table containing the name of the update function module and the associated parameters.

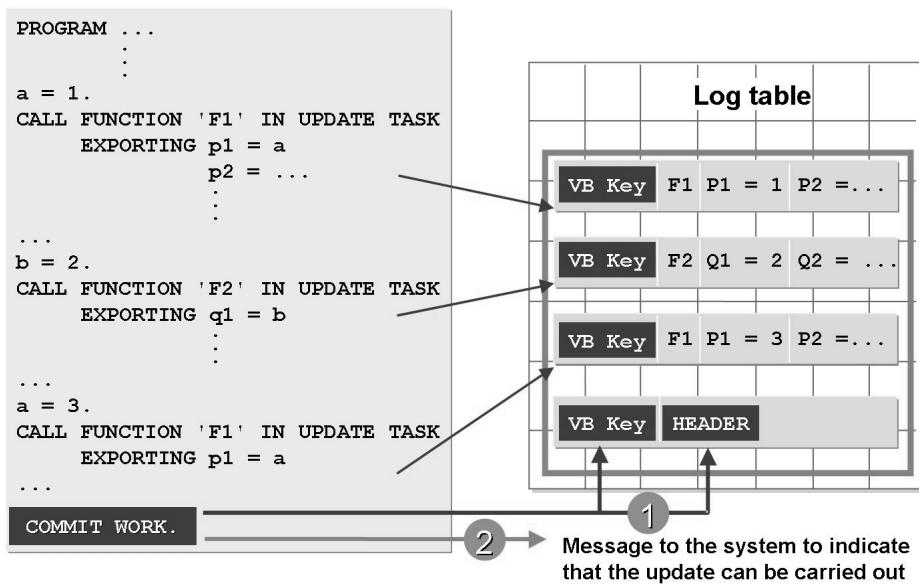


Figure 127: Completing Requests

All of the update requests in an SAP LUW are stored under the same update key (VB key). The update key is a unique key.

When the system reaches the next COMMIT WORK statement, a log header is generated for the corresponding log entries, concluding the set of update entries for that SAP LUW. The log header contains information on the dialog program that wrote the log entries, as well as information on the update modules to be executed.

As well as the header entry, the ABAP command COMMIT WORK ensures that the dispatcher process is informed about the availability of a further update packet.

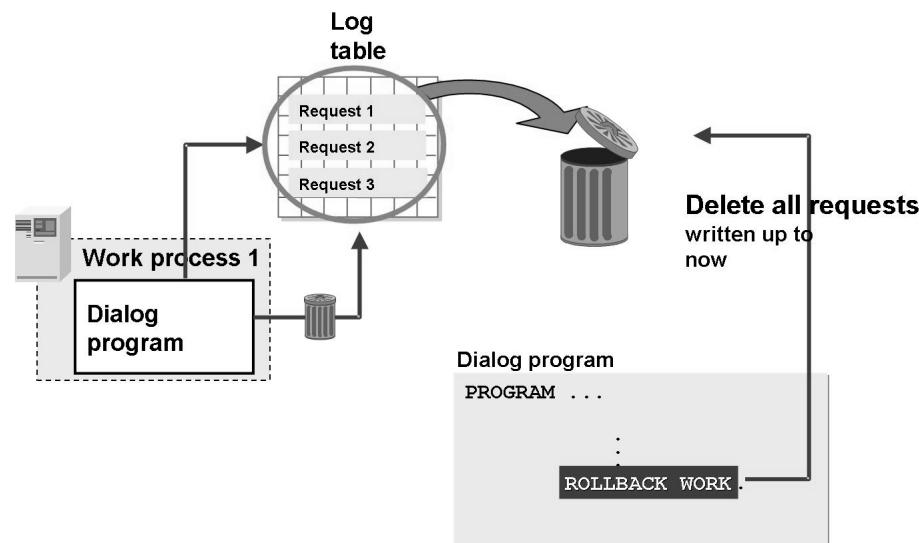


Figure 128: The Result of ROLLBACK WORK

In a dialog consisting of several steps, you can store multiple entries in the update log table that are then processed following the ABAP COMMIT WORK command. However, you may also need to delete the update requests of the current SAP LUW using a ROLLBACK WORK statement.

In a ROLLBACK WORK statement, the system:

- Deletes all form routines registered using PERFORM <subroutine> ON COMMIT
- Deletes all database update requests from the log
- Triggers a rollback on the database, followed by a database commit
- Starts a new SAP LUW

With relation to database changes already completed in the dialog, the ROLLBACK WORK statement means that all changes in the current database LUW are undone. It also deletes all lock entries generated up to now from the dialog program.

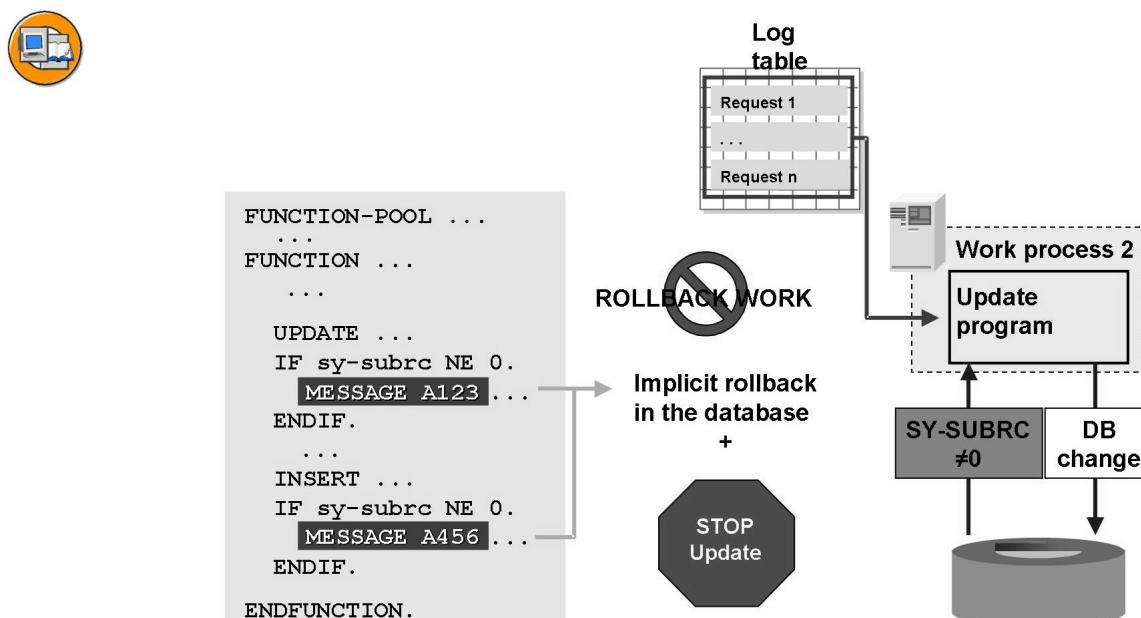


Figure 129: Rollback in the Update Program

The task of an update module is to pass the requests for database updates to the database and to evaluate their return codes. If the database cannot successfully complete an update, the update function module must be able to react.

If you want to trigger a database rollback in the update task, you can use a termination message. This triggers an implicit database rollback. The rollback ends the update task. The log entry belonging to the SAP LUW is flagged as containing an error. The termination message is also entered in the log.

The system sends an express mail to the relevant user, telling him or her that the update was unsuccessful. You can examine the log entry by using Transaction SM13.

 **Note:** You may not use the explicit ABAP statements COMMIT WORK or ROLLBACK WORK in an update module.

If your program is to run using locks, you must record the locks in the lock table. These are inherited by the update modules with the ABAP command COMMIT WORK and can then no longer be accessed by the dialog program.

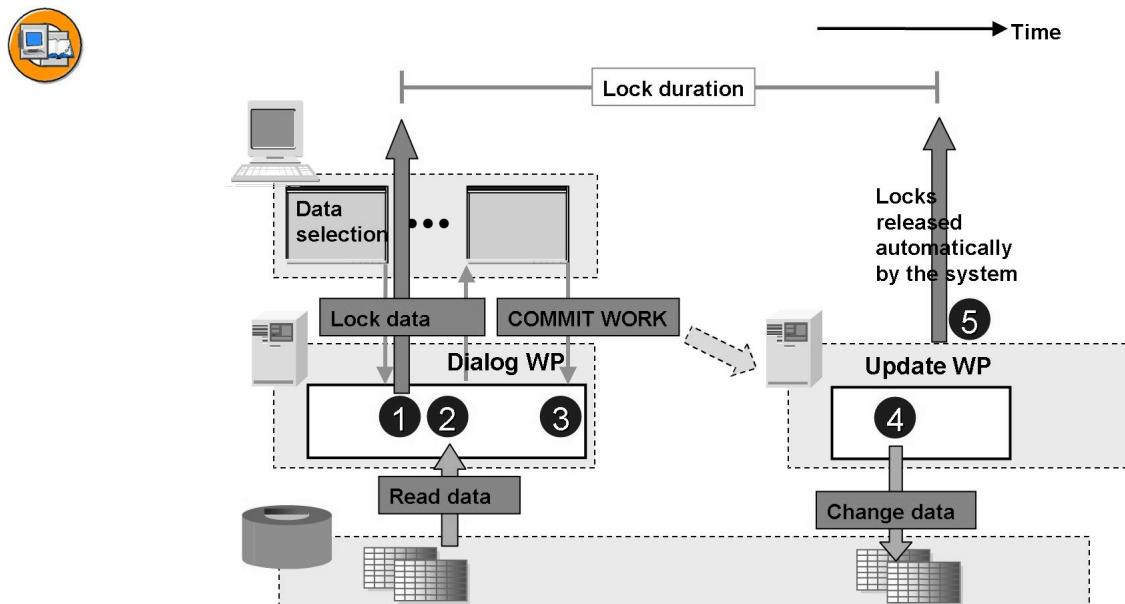


Figure 130: Update: Locks

To ensure that the update modules run with the protection of locks, the lock entries must not be released before the COMMIT WORK. You do not need to release the locks explicitly in the update modules, since they are automatically released at the end of the update process by a basis program.

The locks are also released if one of the update modules triggers a database rollback by sending a termination message.

If the update modules allow failed update requests to be reprocessed (see V1 update), you should note that the data in the database tables at the point of reprocessing may be different from that at the point of the failed update attempt. Reprocessing failed update requests is only useful if the data to be updated is not dependent on the database status (e.g. writing of a document failed because of a tablespace overflow). Failed update requests are reprocessed without locks.

Use

There are several ways the update process can be linked to the dialog program that triggers it.

Asynchronous Update

A typical SAP installation contains dialog work processes and at least one update work process. The update work processes are responsible for updating the database.

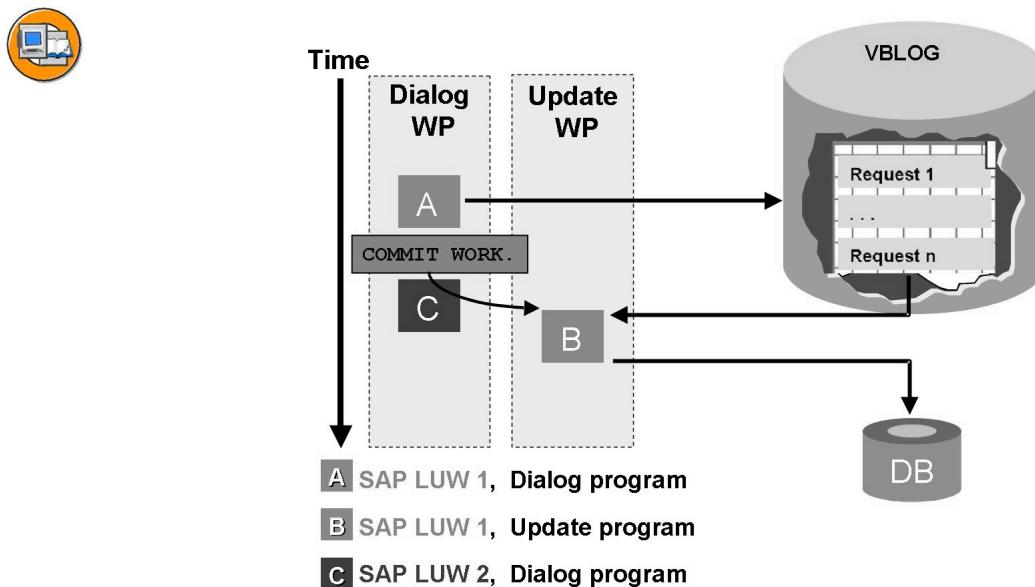


Figure 131: Asynchronous Update

In asynchronous update, the dialog program and update program run separately.

1. The dialog program writes the update requests into the log table VBLOG in the database.
2. You conclude the dialog part of the SAP LUW with the COMMIT WORK statement. A new SAP LUW immediately starts in the dialog program, which can carry on processing user dialogs without interruption. The dialog program does not wait for the update program to finish.
3. The update program is run on a special update work process. This need not be on the same server as the corresponding dialog work process.
4. The SAP LUW that began in the dialog program is continued and then closed by the update program.

The log table VBLOG can be implemented as a cluster file in your system, or be replaced with the transparent tables VBHDR, VBMOD, VBDATA, and VBERROR.

Asynchronous updates are useful in transactions where the database updates take a long time and the "perceived performance" by the shorter user dialog response time is important. Asynchronous update is the standard technique for dialog processing.

The entries that have a HEADER can be analyzed in SM13.

Synchronous Update

With synchronous updates, the dialog program waits for the end of the update modules. The dialog program does not begin to process the new SAP LUW until the update modules have terminated.

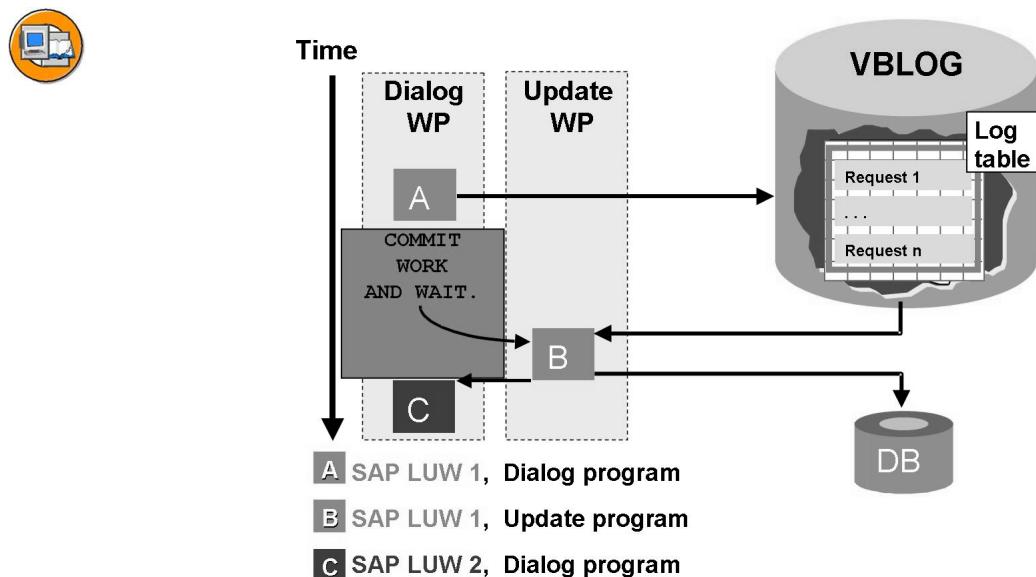


Figure 132: Synchronous Update

To switch from asynchronous to synchronous update, use the AND WAIT addition in the COMMIT WORK statement.

Local Update

In local update, the update functions are run on the same dialog process used by the dialog program containing the COMMIT WORK statement.

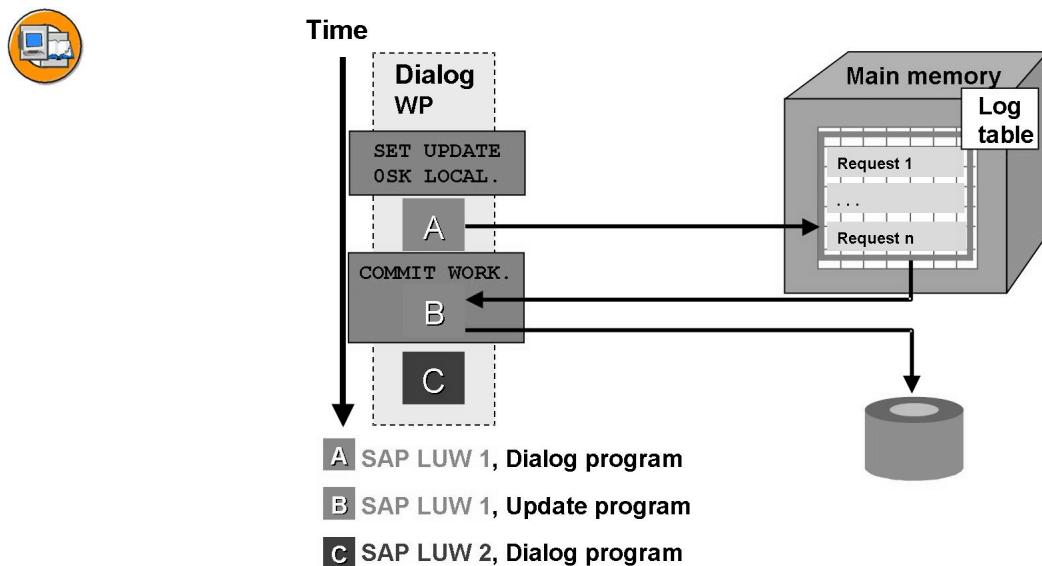


Figure 133: Local Update

The dialog user has to wait for the update to finish before entering further data. This kind of update is useful when you want to reduce the amount of access to the database. The disadvantage of local updates is their parallel nature. The updates can be processed by many different work processes, unlike asynchronous or synchronous update, where the update is serialized due to the fact that there are fewer update work processes (and maybe only one).

To do this, you must include the SET UPDATE TASK LOCAL statement in the dialog program. The effect of this is that update requests are kept in main memory rather than being written into table VBLOG in the database.

When the system reaches the COMMIT WORK statement, the corresponding update modules are processed in the dialog work process currently being used by the dialog program. If all of the update modules run successfully, a database commit is triggered. If not, a database rollback occurs. Once the update function modules have been processed, the dialog program resumes with a new SAP LUW.

The SET UPDATE TASK LOCAL flag can only be set if no other update requests were generated for the same LUW before the program was called up. The flag is effective until the next COMMIT WORK or ROLLBACK WORK command.

As an example, suppose you have a program that uses asynchronous update that you normally run in dialog mode. However, this time you want to run it in the background. Since the system response time is irrelevant when you are running the program in the background, and you only want the program to continue processing when the update has actually finished, you can set the SET UPDATE TASK LOCAL switch in the program. You can then use a system variable to check at runtime whether the program is currently running in the background.

Comparison

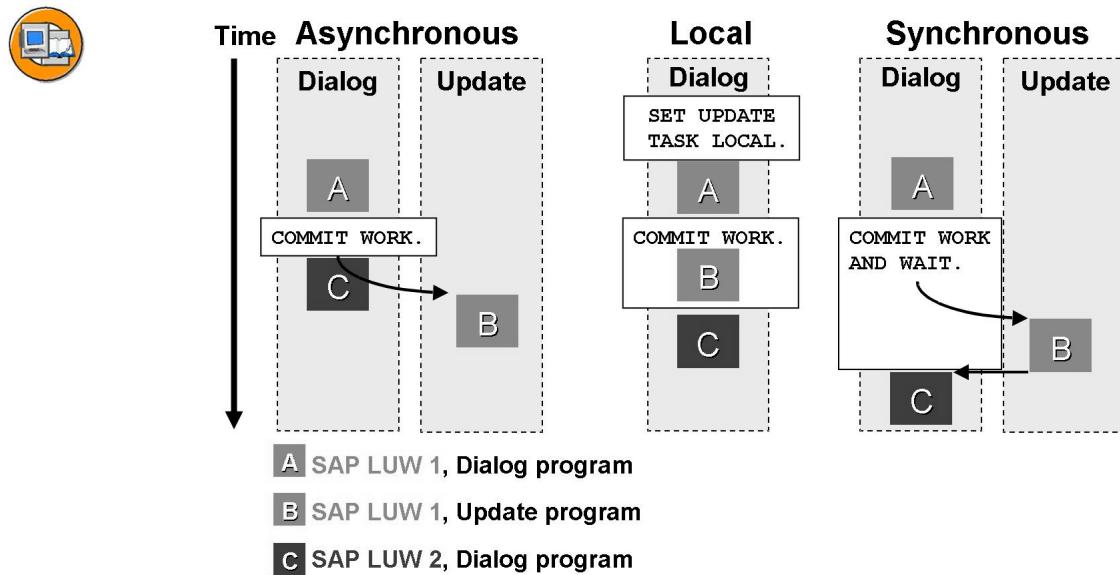


Figure 134: SAP LUW: Comparison of the Timescale

Asynchronous update is useful in transactions where subsequent user dialogs do not depend on the database updates being made immediately. Once the update task has been called, control returns directly to the user.

Local update is particularly useful for processing dialog transactions in the background. There is no contact with the database table VBLOG, and if the program is running alone on the server, local update is faster than either synchronous or asynchronous update. If, as is the usual case, several users are using the server, the speed of the program depends on the total server load.

Synchronous update is useful in transactions where you want to use the advantages of update techniques (logging, opportunity to reprocess failed update requests), but where subsequent user dialogs nevertheless do depend on the results of the update. One particular application for this technique is in "transactions within transactions" - where one transaction uses other transactions as modularization units (CALL TRANSACTION <tcode>). When you use this method, you can determine in the call the update technique that you want the transaction to use. For further information, see the keyword documentation in the ABAP Editor for the term CALL TRANSACTION.

V1 and V2 Updates

When you process a VBLOG entry asynchronously, you can do it in two update steps. This allows you to divide the contents of the update into primary and secondary steps. Update function modules used to do this can be separated into two groups.

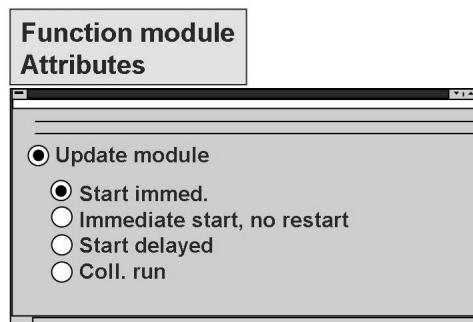


Figure 135: V1 and V2 Update Modules

The group determines when the function module is processed: Function modules that are classified as V1 can be further divided into two subclasses: Start immediately or Start immediately, no restart. V2 function modules are processed asynchronously after all V1 update modules have finished running.

If you have used the Start immediately (V1) option, you can update any records that contained errors manually, using Transaction SM13. If you use the Start immediately, no restart (V1) option, this is not possible. V2 update function modules (Start delayed) can always be manually updated.

V1 update function modules do not normally run using the SAP lock concept. In other words, the V1 update program is executed with the protection of the locks from the dialog program. Any lock entries are released at the end of the V1 update. V2 update function modules always run without logical locks.

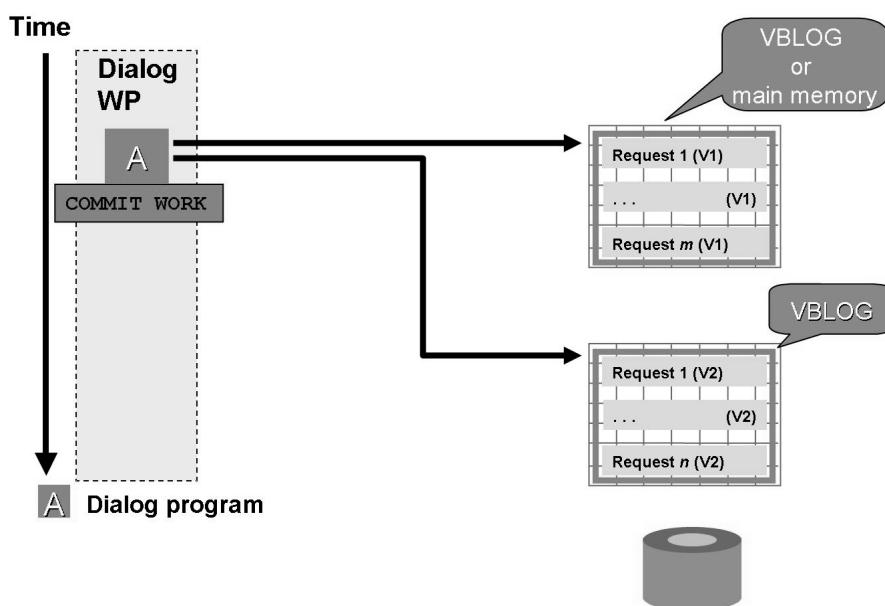


Figure 136: Generating V1 and V2 Updates

The flow diagrams discussed up to now all deal with V1 updates. Update requests for V2 update modules are also generated by the dialog program.

V1 update modules generate update requests in table VBLOG in synchronous and asynchronous update, and in main memory in local update. V2 update modules generate entries in VBLOG and always run asynchronously.

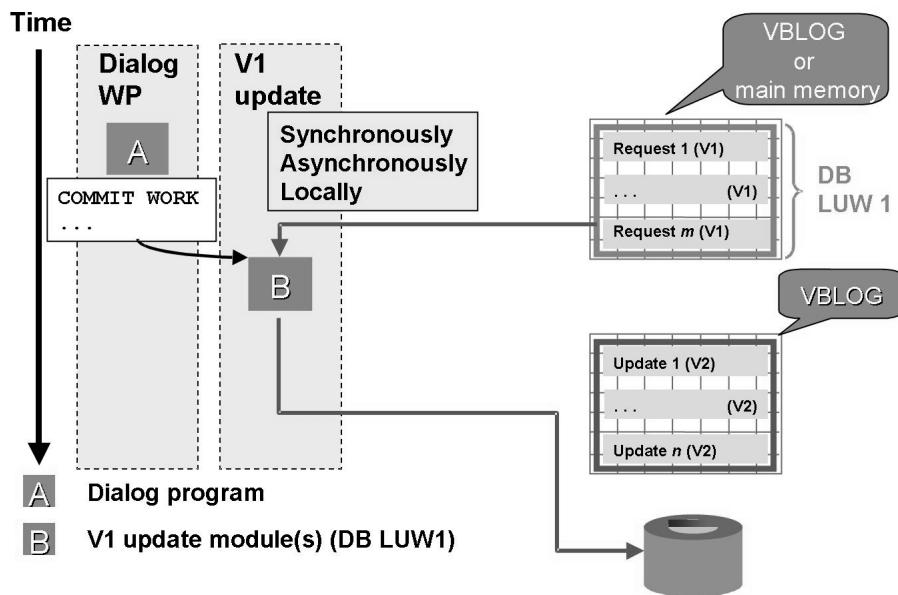


Figure 137: V1 Update

V1 update modules are handled by the system with priority and are executed before the V2 update requests. V1 updates can be performed synchronously, asynchronously, or locally.

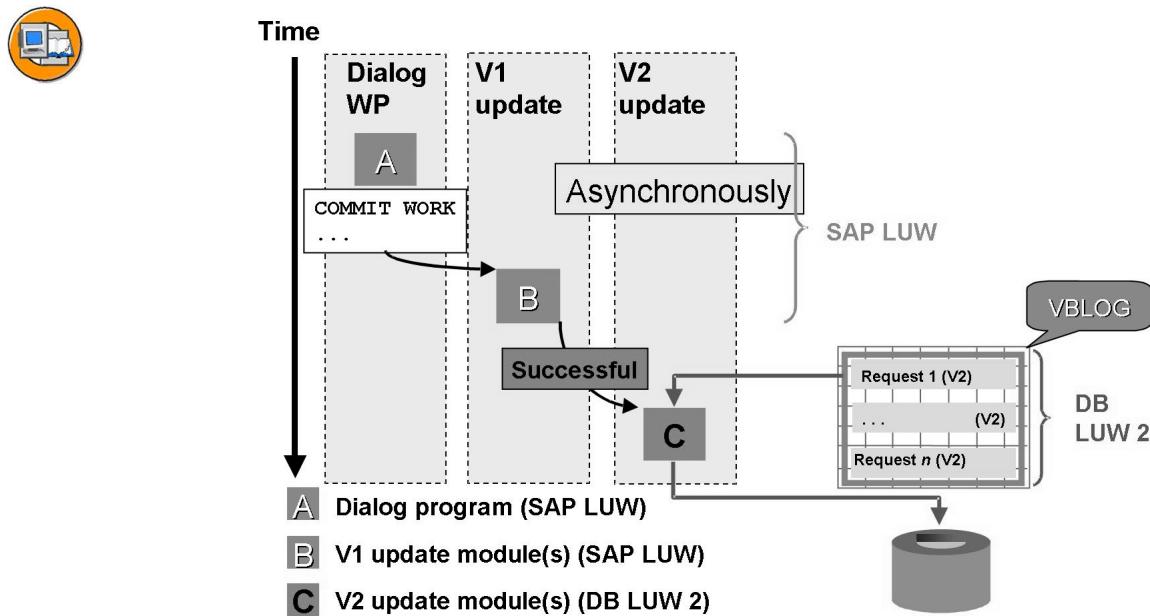


Figure 138: V2 Update

The V2 update function modules run in a separate DB LUW. They are executed in a V2 update work process. If there are no V2 update work processes set up in your system, the V2 update function modules run in a V1 update work process.

→ **Note:** V2 update function modules are not processed until all V1 update function modules have been successfully processed.

Once all of the V2 update function modules have been executed successfully, the V2 update requests are deleted from VBLOG.

If an error occurs in a V2 update function module to which the function module reacts with a termination error message, the system triggers a database rollback. All of the V2 changes in the SAP LUW are undone and an error flag is set in table VBLOG for all of the V2 update requests. V2 update function modules run without SAP locks.

The division between V1 and V2 update function modules allows you to set 'high priority' and 'low priority' updates. V2 update function modules are used for low-priority tasks, such as writing statistics to the database.

The locks generated in the dialog program are usually inherited by the V1 update modules when the update takes place. This is controlled by the SCOPE interface parameter of the lock modules.

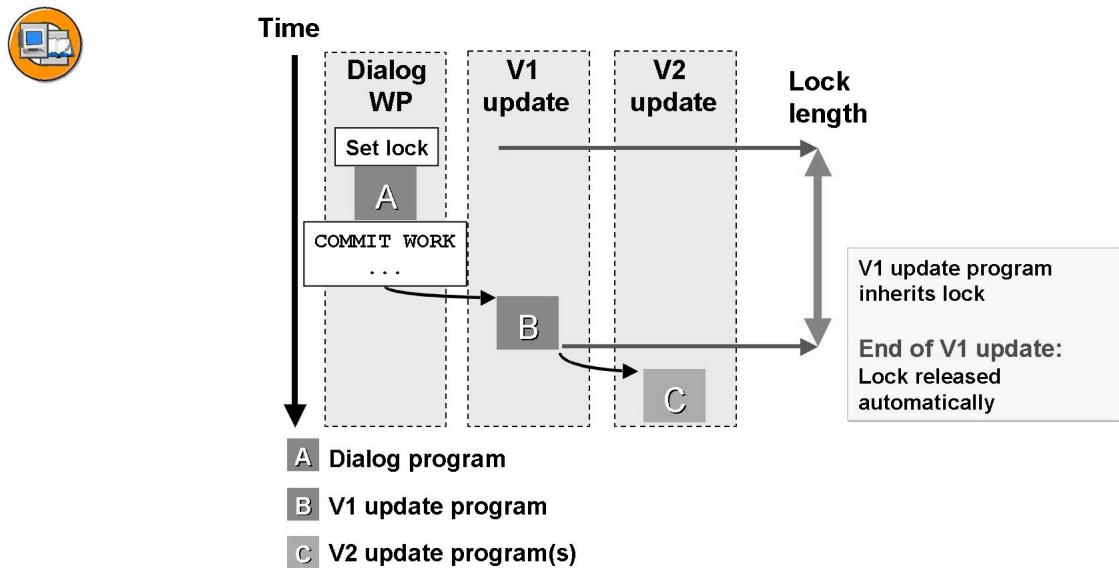


Figure 139: Update and Lock Durations (Scope = 2)

When SCOPE = 2, the V1 update programs inherit the locks that are set in the dialog program (2 is the default setting).

You do not need to release the locks explicitly in your program, since they are automatically released at the end of the V1 update process. The locks are also released if one of the V1 update modules triggers a database rollback by sending a termination message.

Creating Update Function Modules

To create an update function module, you first need to start the Function Builder. Choose *Tools → ABAP Workbench → Function Builder (SE37)*.

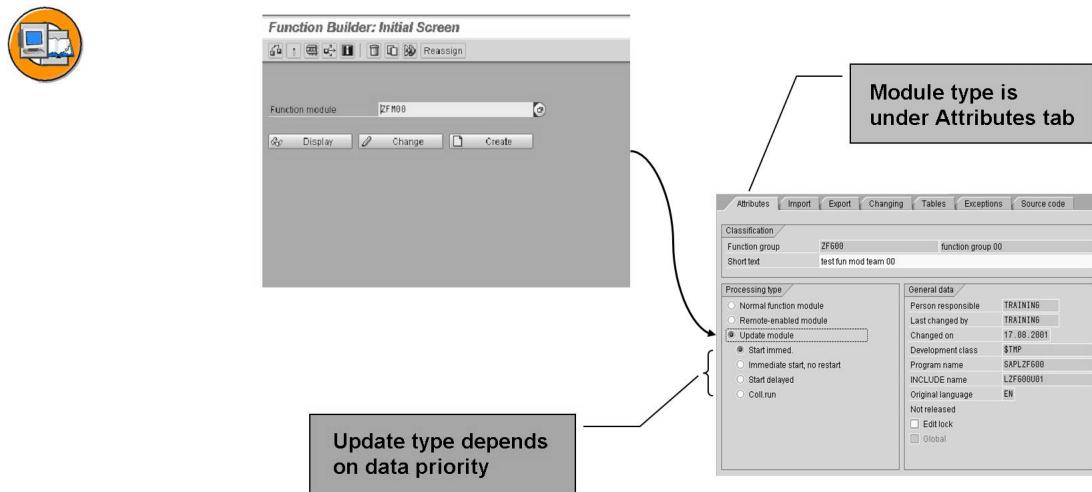


Figure 140: Accessing the Function Builder

Attributes

To be able to call a function module in an update work process, you must flag it as an update module in the Function Builder. When you create the function module, set the Process Type attribute to one of the following values:

- Update with immediate start
Set this option for high priority ("V1") functions that run in a shared (SAP) LUW. These functions can be restarted by the update task in case of errors.
- Update with immediate start, no restart
Set this option for high priority ("V1") functions that run in a shared (SAP) LUW. These functions may not be restarted by the update task.
- Update with delayed start
Set this option for low priority ("V2") functions that run in their own update transactions. These functions can be restarted by the update task in case of errors.

To display the attributes screen in the Function Builder, choose the Attributes tab in the Function Builder detail screen.

Defining the Interface

Function modules that run in the update task have a limited interface:

- Result parameters or exceptions are not allowed since update-task function modules cannot report on their results.
- You must specify input parameters and tables with reference fields or reference structures defined in the ABAP Dictionary.

Source Code

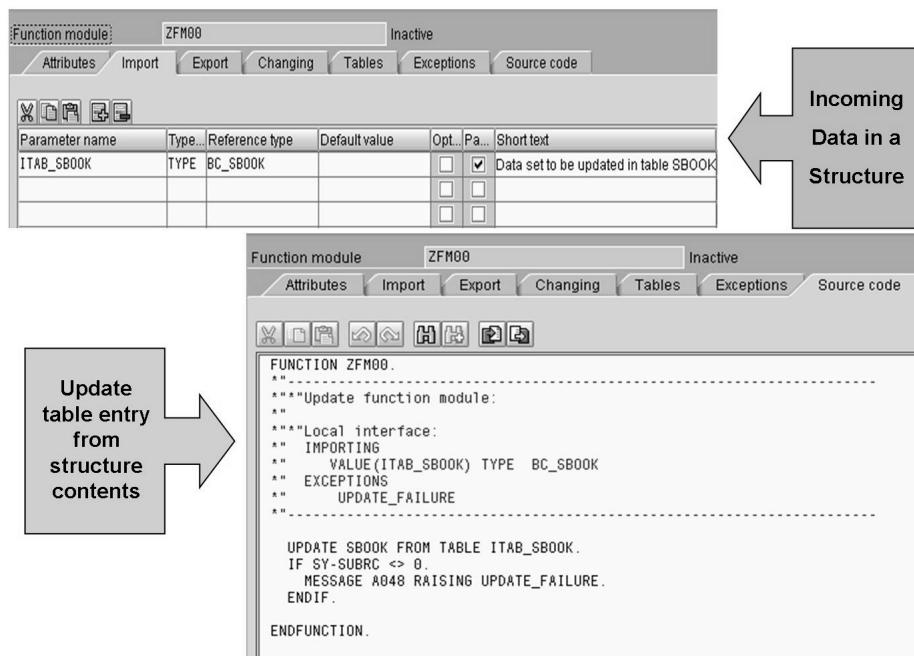


Figure 141: Import Params and Source Code

This is the module that holds the actual update command for the database. Since it is being triggered from the update process, there are no Exceptions for this module. Also, there is no coding of COMMIT WORK or ROLLBACK WORK commands.

The update process iteratively calls the update module and controls the completion of the database unit of work. If any of the individual updates fail, the function module should issue an ABORT message to trigger the rollback at the database level. The exact data recovered depends on the amount of change in the unit of work.

Exercise 11: Create an Update BAPI

Exercise Objectives

After completing this exercise, you will be able to:

- Create the update function module, lock object, BAPI RFC, and integrate into BO as API method.

Business Example

In order to use the update process to change data, you need to have a special function module that simply receives data and issues the update request. This module is very straightforward and does not include other application logic, as such. Once the update function module created, create a BAPI style function that will call it. Finally, integrate the BAPI into the BOR as a method of your BO.

Task:

In order to update data in the database using a BAPI, you need to have a function module that receives input information from the calling program and passes that information to an update module using the update technique. In addition, you must insure the integrity of the SAP LUW by locking the appropriate data against any outside interference. Since these components do not already exist, they must be created.

1. Create lock object **EZSKNVK##** that can place a logical lock on a specific entry in the customer contact table. This lock object should prohibit any access to your data while your work is in process.



Hint: Customer contact information is maintained in table SKNVK.

Save and activate your lock object.

2. Create update function module **Z_CONTACT_UPDATE##**. This module will require an import parameter to receive the data used for the update. Since this module is used in the update technique, there will be limited code and proper error handling must be addressed.
3. Create the **Z_BAPI_CONTACT_UPDATE##** function module. The BAPI will require import parameters to receive the contact number, customer number, telephone number, and the entry person's id. Data typing should be done using dictionary structure **BAPI0417_2**. The RETURN structure will be assigned to the standard **BAPIRET2** Dictionary structure.

Continued on next page

The BAPI should receive the input data for the update and set a logical lock for the appropriate entry in the database. If the lock is successful, pass the information to the update function module for processing asynchronously. If the lock is not successful, check the return code in SY-SUBRC. If it's a value of 1, respond with error message 004 from message class BC417. If it's a value of 2, respond with error message 005 from message class BC417.

Check the return code from the call to the update function module. If it is not zero, issue error message number 006 from message class BC417. If the call was successful, issue success message number 007 from message class BC417.

4. Append your update BAPI to Business Object ZCON_## as a method.
5. Test the BAPI.

Solution 11: Create an Update BAPI

Task:

In order to update data in the database using a BAPI, you need to have a function module that receives input information from the calling program and passes that information to an update module using the update technique. In addition, you must insure the integrity of the SAP LUW by locking the appropriate data against any outside interference. Since these components do not already exist, they must be created.

1. Create lock object **EZSKNVK_##** that can place a logical lock on a specific entry in the customer contact table. This lock object should prohibit any access to your data while your work is in process.



Hint: Customer contact information is maintained in table SKNVK.

Save and activate your lock object.

- a) Go to the ABAP Dictionary using the path *Tools → ABAP Workbench → Development → ABAP Dictionary* and select the Lock Object radiobutton.
- b) Enter the Lock Object name **EZSKNVK_##** and choose Create.
- c) Enter an appropriate description for your lock object in the Short text field and then select the Tables tab.
- d) Enter SKNVK as the primary table name and Exclusive, not cumulative as the lock mode (from the dropdown list). Select the Lock Parameter tab.
- e) Verify that MANDT and PARNR are listed as the lock parameter fields for the lock object.
- f) Save and activate your lock object.

Continued on next page

2. Create update function module Z_CONTACT_UPDATE_##. This module will require an import parameter to receive the data used for the update. Since this module is used in the update technique, there will be limited code and proper error handling must be addressed.
- On the main Function Builder screen, enter **Z_CONTACT_UPDATE_##** as the name of your update function module and select Create.
 - Assign the function module to Function Group ZBC417_## and enter an appropriate Short text description. Choose Save.
 - Select the **Attributes** tab and select the Update Module and Start Immed. radiobuttons.
 - Select the **Import** tab and enter an Import parameter name for the incoming contact data. Use SKNVK as the reference type. Also, select the Pass by Value check box and enter a suitable short text.
 - Select the **Source code** tab and enter the ABAP update command for table SKNVK. The values to use should be in the import parameter already specified. Check the result of the update request and, if not successfully completed, issue an abort message to roll back the pending changes. Use message-id BC417 and message number 6.
 - Save and activate the function module.
 - Test and document your function module in the same manner as the GetList function module.
3. Create the Z_BAPI_CONTACT_UPDATE_## function module. The BAPI will require import parameters to receive the contact number, customer number, telephone number, and the entry person's id. Data typing should be done using dictionary structure BAPI0417_2. The RETURN structure will be assigned to the standard BAPIRET2 Dictionary structure.

The BAPI should receive the input data for the update and set a logical lock for the appropriate entry in the database. If the lock is successful, pass the information to the update function module for processing asynchronously. If the lock is not successful, check the return code in SY-SUBRC. If it's a value of 1, respond with error message 004 from message class BC417. If it's a value of 2, respond with error message 005 from message class BC417.

Check the return code from the call to the update function module. If it is not zero, issue error message number 006 from message class BC417. If the call was successful, issue success message number 007 from message class BC417.

- On the main Function Builder screen, enter the name of your BAPI as Z_BAPI_CONTACT_UPDATE_##, where ## is your group number. Choose Create.

Continued on next page

- b) Assign the function module to Function Group ZBC417_## and enter an appropriate Short text description. Choose Save.
- c) Select the **Attributes** tab and select the radio button Remote-enabled module
- d) Select the **Import** tab and set up import parameters for contact id, customer number, telephone number, and entered by person's id as individual parameter fields. The data typing should be supplied by BAPI0417_2-PARNR, BAPI0417_2-KUNNR, BAPI0417_2-TELF1, and BAPI0417_2-ERNAM, respectively.
- e) Select the **Export** tab and set up an export parameter for the return structure.
- f) Select the **Source code** tab and insert statements to CLEAR all of your table declarations, if any.
- g) Before proceeding to the database, place a lock on the contact entry you are trying to update. Using the Pattern button, insert a call to the ENQUEUE_EZSKNVK_## function module. Associate the lock module's PARNR parameter with the contact id parameter you specified in earlier in the interface.
- h) If the lock was not successfully set, determine the value of sy-subrc. If the value equals 1, respond with an error message 004 from message class BC417. If the value equals 2, respond with an error message 005 from message class BC417.
- i) If the lock was successful, select the entire single entry from the SKNVK table using the imported contact id. If the select was not successful, construct an error message using literals and the contact id. Insert this concatenated message in the message-msg1 field.
- j) If a new telephone number was supplied, move it to the corresponding field of the SKNVK table. If a new entered by person id was supplied, move the id to its corresponding field of the SKNVK table as well.
- k) Use the Update Technique to modify the database by calling your contact update function module using the IN UPDATE TASK addition. Send the new contents of the SKNVK work area to the function module for processing.
- l) If the change was not completed, issue error message number 006 from message class BC417. If the update was successful, issue a success message number 007 from message class BC417.
- m) Save, activate, test, and release your function module.

For a sample code excerpt, see function module
BAPI_CONTACT_UPDATE.

Continued on next page

4. Append your update BAPI to Business Object ZCON_## as a method.
 - a) Go to the Business Object Builder using the path *Tools → ABAP Workbench → Programming Environment → Business Object Builder (SWO1)*.
 - b) Enter the name of the business object to which you want to append a method. Select Change.
 - c) Choose *Utilities → API Methods → Add method*.
 - d) In the function module pop-up box, enter the name of your function module.
 - e) In the methods properties pop-up window, type in the name you want to use for your BAPI. Use a capital letter for each new word, but include no spaces. For example, UpdateDetail.
 - f) Select the Next Step icon to proceed to the parameters screen.
 - g) Verify that the proposed parameter names are correct. Make any necessary changes. Use a capital letter for each new word, but include no spaces. For example, StreetAddress.
 - h) Select the Next Step icon to proceed to the next screen.
 - i) You will get a pop-up window with a message indicating that your method has not yet been completed. Select Yes to implement your method.
 - j) You are again at the main business object screen. Your method now has a green light indicating that it is a BAPI.
 - k) To release your BAPI, you must select it. Then, follow the menu path *Edit → Change release status → Object type component → To release*. The check mark next to it indicates that it is released.
 - l) Save and generate the method.
5. Test the BAPI.
 - a) Select the BAPI you want to test and choose Execute or press F8.
 - b) On the Test Object screen select the Execute icon next to the BAPI you want to test
 - c) On the Test Method screen, enter any necessary parameter values for the test and select the Execute icon or press F8.
 - d) On the Test Method Display Results screen, your parameters will contain the returned values from the test.
 - e) Verify that your update worked by executing the Get Detail BAPI and displaying the record you modified.



Lesson Summary

You should now be able to:

- Explain the different update techniques available
- Perform bundled database updates
- Create and use update function modules

Related Information

- For additional information on developing update function modules and their use, refer to the topic Update Techniques in the ABAP Programming section of the Online Help.



Unit Summary

You should now be able to:

- Describe the Business Scenario
- List the phases of a scenario and their purpose
- Find and access tools needed for BAPI development
- List the naming conventions related to BAPIs and related components
- Describe the business scenario
- Define the business scenario
- Review the business scenario
- Define the components related to the BAPI interface
- Outline the structured components used in writing the source code of the BAPI
- Explain the important role of documentation of the BAPI
- Test the documentation for accuracy and completeness
- Test an RFC-enabled function module
- Describe Business Objects and their relation to the BOR
- Create a Business Object
- Add a BAPI as an API method to a business object.
- Test the BAPI Method
- Define the terms database LUW and SAP LUW
- Explain the need to bundle changes to the database tables
- Apply the knowledge of updating techniques to the distributed application environment and the use of BAPIs
- Describe the authorization concept for BAPIs
- Describe Authorization Objects and Authorizations
- List requirements to implement authority checking
- Use lock modules
- Find, maintain, and generate lock modules
- Explain the role of lock objects
- Explain the different update techniques available
- Perform bundled database updates
- Create and use update function modules

Related Information

For additional information on creating, documenting and testing BAPIs, refer to the BAPI Programming Guide for ECC 6.0 on SAPNet.



Test Your Knowledge

1. You should review the business scenario for completeness only after the process goes live.

Determine whether this statement is true or false.

- True
- False

2. Which tool is not part of the ABAP Workbench?

Choose the correct answer(s).

- A Function Builder
- B BAPI Explorer
- C ABAP Dictionary
- D Repository Information System
- E Message Maintenance Tool

3. What steps you through the entire development process and provides direct navigation options to required tools? _____

Fill in the blanks to complete the sentence.

4. All data elements for date fields must have the format YYYYMMDD.

Determine whether this statement is true or false.

- True
- False

5. All customer developed BAPIs must be put into one special function group.

Determine whether this statement is true or false.

- True
- False

6. You can only search for BAPIs from outside the BAPI Explorer.

Determine whether this statement is true or false.

- True
- False

7. The three phases of the Business Scenario analysis are; describe the business scenario, review the business scenario, and implement the business scenario.
Determine whether this statement is true or false.
- True
 False
8. In which phase would you use a project form from the BAPI Explorer?
Choose the correct answer(s).
- A Describe the business scenario
 B Define the business scenario
 C Review the business scenario
 D Implement the business scenario
9. One of the important questions of the review phase is to ask if all of the BAPIs in the scenario work smoothly together.
Determine whether this statement is true or false.
- True
 False
10. If a key field value is passed to the BAPI by the calling program, the key field must be set as an Import parameter in the function module.
Determine whether this statement is true or false.
- True
 False
11. What is the name of the export parameter used to send messages back to the calling program?
Choose the correct answer(s).
- A BAPIRET
 B BAPIRET2
 C RETURN
 D SENDBACK
12. Within the source code of a BAPI, you can execute a CALL TRANSACTION statement.
Determine whether this statement is true or false.
- True
 False

13. Because the Method of a Business Object is a function module, you write its documentation in the

Choose the correct answer(s).

- A Function module itself
- B Business Object Repository
- C Function Group
- D Function Builder

14. Business objects include objects such as Customer and Material.

Determine whether this statement is true or false.

- True
- False

15. Which of the following is NOT a function of the BOR?

Choose the correct answer(s).

- A Provides an object oriented view of the SAP System data and processes?
- B Manages BAPIs in release updates?
- C Accepts any function module as a BAPI?
- D Ensures interface stability?

16. You can navigate in the BOR using transaction SWO1, SWO3 or BAPI.

Determine whether this statement is true or false.

- True
- False

17. When creating a Business Object, you get default methods from a standard SAP interface.

Determine whether this statement is true or false.

- True
- False

18. A Business Object has 2 names; the Object type ID and the Object name. The object name is only for naming purposes and is not used anywhere in the SAP System.

Determine whether this statement is true or false.

- True
- False

19. Once a BAPI has been implemented as a Method of a Business Object, you can verify this if it has a Green Light next to it.

Determine whether this statement is true or false.

- True
- False

20. A database Logical Unit of Work is a mechanism used by the database to ensure that the data is always consistent.

Determine whether this statement is true or false.

- True
- False

21. The mechanism that triggers the actual database update is _____.

Fill in the blanks to complete the sentence.

22. You can undo all the database changes back to the beginning of the current _____.

Fill in the blanks to complete the sentence.

23. A bundling technique is _____.

Choose the correct answer(s).

- A a procedure to update multiple databases.
- B a compression algorithm for data.
- C grouping transactions to form applications
- D an encryption routine for data packets
- E a way to implement the all-or-nothing principle

24. If a field of the authorization object should not be checked, you should enter _____.

Fill in the blanks to complete the sentence.

25. Even though a user is refused update access to data, they will still be given display only access.

Determine whether this statement is true or false.

- True
- False

26. The easiest way to insert an authority-check command into a program is to use

Fill in the blanks to complete the sentence.

27. The system prohibits you from accessing or changing data once you have failed an authorization check.

Determine whether this statement is true or false.

- True
- False

28. You create the Lock Table as a type standard internal table with key fields including client, user ID, lock type, and so on.

Determine whether this statement is true or false.

- True
- False

29. If you're not sure that you will be updating anything, you should set a shared lock on your data.

Determine whether this statement is true or false.

- True
- False

30. You can create lock objects using

Choose the correct answer(s).

- A Data Dictionary
- B Lock Manager
- C Object Navigator
- D a method call to the Constructor
- E Function Builder

31. The Enqueue/Dequeue function modules should be created

Choose the correct answer(s).

- A at the beginning of the SAP Logical Unit of Work
- B only by experienced ABAP developers
- C by the system that generates them for you
- D by copying an existing one and changing the copy
- E only by SAP developers

32. The parameter that controls how the lock or lock release is passed to the update program is _____

Fill in the blanks to complete the sentence.

33. The purpose of update techniques is to divide the SAP LUW into two separate, but related, logical units of work.

Determine whether this statement is true or false.

- True
- False

34. The command(s) that signal the end of input processing and the start of database processing for this SAP LUW is _____

Fill in the blanks to complete the sentence.

35. The three ways that a dialog program and an update program can be linked are: _____, _____, and _____

Fill in the blanks to complete the sentence.

36. V2 updates can be run without locks.

Determine whether this statement is true or false.

- True
- False

37. Which of the following statements about lock handling and control during updates is not true.

Choose the correct answer(s).

- A One setting of the scope parameter creates two sets of locks
- B The default setting for an update function module gives the locks to another program.
- C Even with updating, you can hold onto locks as long as you want.
- D Locks for BAPI updates are stored in the VBLOG.
- E BAPIs use locks on data just like SAP programs do.



Answers

1. You should review the business scenario for completeness only after the process goes live.

Answer: False

You should make sure the process is functionally complete before you actually define anything. If you do not, you may have to discard portions of your work to repair gaps that were overlooked.

2. Which tool is not part of the ABAP Workbench?

Answer: B

The BAPI Explorer is part of the Business Framework group not the ABAP Workbench.

3. What steps you through the entire development process and provides direct navigation options to required tools? [The Project Form](#)

Answer: The Project Form

The Project Form, which can be found in the BAPI Explorer, walks you through the development process for Business Objects and BAPIs.

4. All data elements for date fields must have the format YYYYMMDD.

Answer: True

Non SAP Systems cannot reformat the date according to the User Master record. Therefore, we deal with the date in it's internal format.

5. All customer developed BAPIs must be put into one special function group.

Answer: False

All BAPIs belonging to one SAP business object type should be created in one function group.

6. You can only search for BAPIs from outside the BAPI Explorer.

Answer: True

In the Function Module screen of the Repository Information System, select the *All Selections* button and mark the checkbox for the *BAPI module type*.

7. The three phases of the Business Scenario analysis are; describe the business scenario, review the business scenario, and implement the business scenario.

Answer: False

The three phases of the business scenario analysis are; describe the business scenario, define the business scenario, and review the business scenario.

8. In which phase would you use a project form from the BAPI Explorer?

Answer: D

The project form would be used in the Design phase and you can find this form in the BAPI Explorer. The form steps you through the various parts necessary to complete your BAPI.

9. One of the important questions of the review phase is to ask if all of the BAPIs in the scenario work smoothly together.

Answer: True

There are several questions that need to be answered in the review phase of which this is one of the most important.

10. If a key field value is passed to the BAPI by the calling program, the key field must be set as an Import parameter in the function module.

Answer: True

Any key field that will receive a value when the BAPI is called must be set as an Import parameter in the function module

11. What is the name of the export parameter used to send messages back to the calling program?

Answer: C

A RETURN parameter is required in a BAPI and is used to send any messages back to the calling program.

12. Within the source code of a BAPI, you can execute a CALL TRANSACTION statement.

Answer: False

Any statement that would generate output whether it is in the form of a list, a query or a dialog screen, is not allowed in a BAPI.

13. Because the Method of a Business Object is a function module, you write its documentation in the

Answer: D

Since the Method of a business object is really the RFC function module, the documentation is created in the Function Builder.

14. Business objects include objects such as Customer and Material.

Answer: True

Business objects provide a programming interface to the SAP System.

15. Which of the following is NOT a function of the BOR?

Answer: C

In order for a function module to be used as a BAPI, it must be an RFC-enabled function module?

16. You can navigate in the BOR using transaction SWO1, SWO3 or BAPI.

Answer: True

Any of these transactions provides navigation in the BOR depending on what information you know about the Business Object.

17. When creating a Business Object, you get default methods from a standard SAP interface.

Answer: True

The standard SAP interface IFSAP is provided as a default with a Business Object and provides 2 default Methods; ExistenceCheck and Display.

18. A Business Object has 2 names; the Object type ID and the Object name. The object name is only for naming purposes and is not used anywhere in the SAP System.

Answer: False

The English name called Object name is used in the BAPI Explorer while the Object name ID is used in the Business Object Builder.

19. Once a BAPI has been implemented as a Method of a Business Object, you can verify this if it has a Green Light next to it.

Answer: True

Only Methods that are RFC enabled function modules can be implemented as BAPIs and they will have a Green Light next to them.

20. A database Logical Unit of Work is a mechanism used by the database to ensure that the data is always consistent.

Answer: True

A database LUW takes the database from one consistent state to the new (changed) consistent state.

21. The mechanism that triggers the actual database update is A database commit.

Answer: A database commit.

The database is not notified that temporary changes are permanent until a commit is issued.

22. You can undo all the database changes back to the beginning of the current database LUW.

Answer: database LUW.

Anything that was previously updated is a permanent change to the database.

23. A bundling technique is

Answer: E

It is a way to group the changes from several steps in a SAP LUW into one database LUW.

24. If a field of the authorization object should not be checked, you should enter Dummy.

Answer: Dummy

All fields for the object must be entered in the check request, including those to be ignored.

25. Even though a user is refused update access to data, they will still be given display only access.

Answer: False

The check is performed for each specific request and the response is for only that check.

26. The easiest way to insert an authority-check command into a program is to use The pattern button in the ABAP Editor.

Answer: The pattern button in the ABAP Editor.

This function automatically inserts an ID statement for each field in the object. It also inserts case—sensitive object and field names.

27. The system prohibits you from accessing or changing data once you have failed an authorization check.

Answer: False

The system just sends an answer to the request. It's up to the program to react appropriately.

28. You create the Lock Table as a type standard internal table with key fields including client, user ID, lock type, and so on.

Answer: False

The lock table is completely defined and maintained by the system. Users simply request that locks be set or released.

29. If you're not sure that you will be updating anything, you should set a shared lock on your data.

Answer: False

The only way to protect your update option is to set the exclusive lock up front.

30. You can create lock objects using

Answer: A

Lock Objects are logical objects composed of a list of tables linked by foreign key relationships. This is done in the Data Dictionary.

31. The Enqueue/Dequeue function modules should be created

Answer: C

The function modules are created during the activation of the lock object. They are automatically assigned to function groups.

32. The parameter that controls how the lock or lock release is passed to the update program is SCOPE.

Answer: SCOPE.

The standard setting for the ENQUEUE module is _SCOPE = 2. This passes complete control of the lock to the update program.

33. The purpose of update techniques is to divide the SAP LUW into two separate, but related, logical units of work.

Answer: False

Update techniques are used to insure the integrity of the SAP LUW. What is separated is the user input phase from the database update phase. This maximizes end-user resources.

34. The command(s) that signal the end of input processing and the start of database processing for this SAP LUW isCommit Work and Rollback Work.

Answer: Commit Work and Rollback Work.

35. The three ways that a dialog program and an update program can be linked are:Asynchronous, Synchronous, and Local Update.

Answer: Asynchronous, Synchronous, Local Update.

Asynchronous is the default and is generally the fastest.

36. V2 updates can be run without locks.

Answer: True

V2 update function modules always run without locks.

37. Which of the following statements about lock handling and control during updates is not true.

Answer: D

Locks are always stored in the Lock Table. The updates themselves are stored in the VBLOG.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 4

BAPIs and RFCs

Unit Overview

This Unit explains the various components needed to create Remote Function Calls (RFCs).

This Unit also covers Web Services, an XML-based Internet communication standard.



Unit Objectives

After completing this unit, you will be able to:

- List the requirements for RFC calls
- Describe the types of RFC calls and their use
- Use an RFC call to access a remote system
- Describe the basic technology surrounding Web services.
- Expose a BAPI as a Web Service using the Web Service Wizard.

Unit Contents

Lesson: External ABAP Call.....	242
Exercise 12: Create an RFC Destination.....	255
Lesson: Web Services	259
Exercise 13: Making your BAPI Web Service enabled	275

Lesson: External ABAP Call

Lesson Overview

This lesson demonstrates how to create and use an RFC call to execute a BAPI in an external SAP System.



Lesson Objectives

After completing this lesson, you will be able to:

- List the requirements for RFC calls
- Describe the types of RFC calls and their use
- Use an RFC call to access a remote system

Business Example

A user needs to extract some data that resides in another SAP System.

Remote Function Calls (RFC)

Types of RFCs

The majority of this lesson contains material that is applicable to all the SAP-supported platforms. These are the topics dealing directly with the Remote Function Call (RFC). RFC is SAP's platform-independent core technology for all the various types. To properly take their place as SAP's standard interface mechanism for accessing SAP components, they must be able to take (this technology).

Overview

A remote function call is a call to a function module running in a system different from the caller's. The remote function can be called from within the same system (as a remote call), but usually the caller and the called will be in different systems. RFC communication with the remote system happens as part of the CALL FUNCTION statement.

RFC functions running in an SAP system must be actual function modules, and must be registered in the SAP System as remote.

When both the caller and called program are ABAP programs, the RFC interface provides both partners to the communication. The caller may be any ABAP program, while the called program must be a function module registered as remote. The implications of this in the use of BAPIs is very important, as a BAPI is physically implemented as a remote-enabled function module. A BAPI can be both a sender and a receiver of a remote call.

Accessing BAPIs from ABAP

You can only access BAPIs currently from an ABAP program using the relevant direct function call (local or remote). SAP is planning for a later release to integrate all BOR business objects into the class library of ABAP so that you will be able to access BAPIs from ABAP using Object-Oriented programming techniques.

RFC Features

The parameter value Dest can be either a literal or a variable: its value is a logical destination (for example, hw1071_53) known to the local SAP System. The destination parameter displays the name of an entry in the RFCDES table. This entry contains all necessary parameters to connect to and log in the destination system. RFC frees the ABAP programmer from having to program his own communications routines. When you make an RFC call, the RFC interface takes care of:

- Converting all parameter data to the representation needed in the remote system. This includes character string conversions, and any hardware-dependent conversions needed (for example, integer, floating point). All ABAP data types are supported.
- Calling the communication routines needed to talk to the remote system.
- For non-BAPI situations, handling communications errors, and notifying the caller, if desired. (The caller requests notification using the EXCEPTIONS parameter of the CALL FUNCTION statement).

 **Note:** there is no support for Dictionary structures.

The RFC interface is effectively invisible to the ABAP programmer. Processing for calling remote programs is built into the CALL FUNCTION statement. Processing for being called is generated automatically (in the form of an RFC stub) for every function module registered as remote. This stub serves as an interface between the calling program and the function module.

A distinction is made between an RFC client and RFC server. RFC client is the instance that calls up the Remote Function Call to execute the function that is provided by an RFC server. The functions that can be executed remotely will be called RFC functions and the functions provided via RFC API will be called RFC calls.

All RFC functions available in a remote RFC server system, which are called by an RFC client, are processed transactionally. This means that after execution of the first RFC function in the RFC server system, the complete context (all globally defined variables in the RFC server program or in the main program of a function module) is available for further RFC functions. The RFC connection is closed only:

1. when the context of the calling ABAP program has ended or
2. when explicitly requested by RfcAbort or RfcClose in the external program.

To make the execution of RFC functions reliable, safe, and independent from the availability of the RFC server or RFC server system, SAP Systems from release 3.0 onwards uses the transactional RFC (tRFC). This ensures that the called function module is executed only once in the RFC server system.

In tRFC calls, the data that belongs to an RFC function must first be stored temporarily on the SAP database in the RFC client system. When processing is completed, this must be reported back to the calling ABAP program. Everything else is handled by the tRFC component in the SAP System.

Since a database is not always available on external systems, the link to the tRFC interfaces is implemented such that the client or server programs based on RFC API must take on some administrative functions to ensure that the respective function module is executed only once.

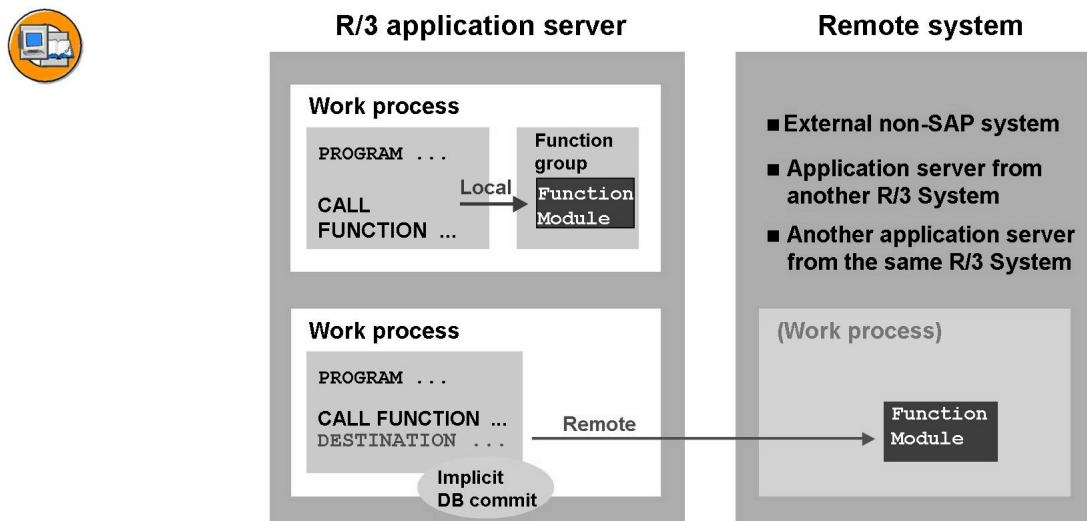


Figure 142: RFC from SAP System

If a customer needs to make a call to a function module that resides on some other system than they are currently using, the Remote System can be used. The Remote System can connect to an external non-SAP system, an R/2 system, or to another SAP System.

If a function module is called remotely, it runs in its own work process (its own SAP LUW) if the remote system is an SAP System.

→ **Note:** the calling program is rolled out for each remote function call, which triggers an implicit database commit.

The logon for remote SAP partner systems is automatic. The user and password are stored in the table RFCDES. You maintain the RFC destinations using transaction SM59.

Synchronous calls can be made using the Remote System. Each of these RFC calls makes up a single logical unit of work (LUW) in the remote system. Since the preferred method is a Synchronous call, the lesson concentrates on this method.

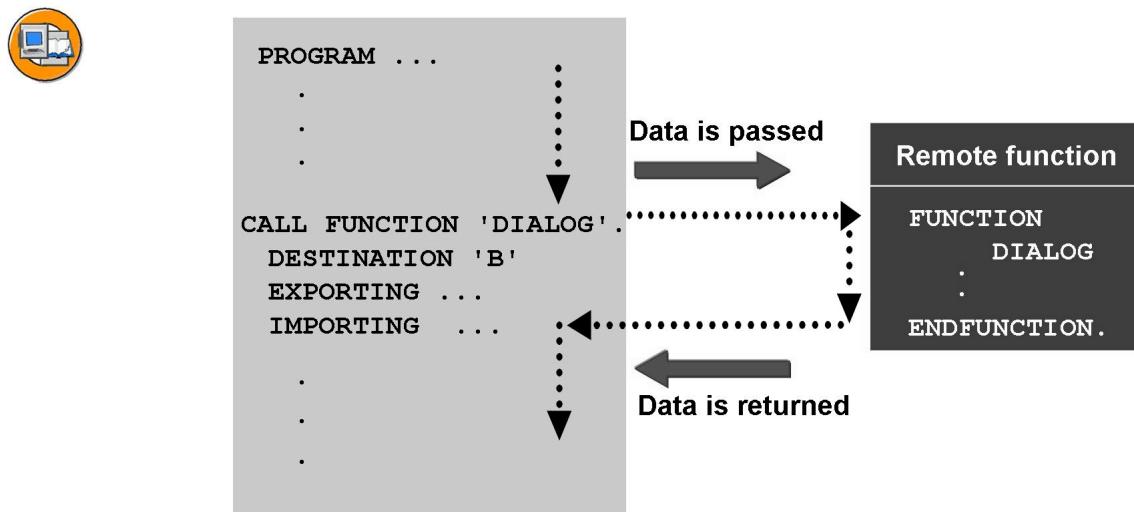


Figure 143: Process Flow: Synchronous RFC (sRFC)

With synchronous RFC, processing stops in the calling program until the called remote function is processed and its output is returned. Then in the calling program, the processing continues after the call. Using this method, you will add a Destination statement in your program's function module call.

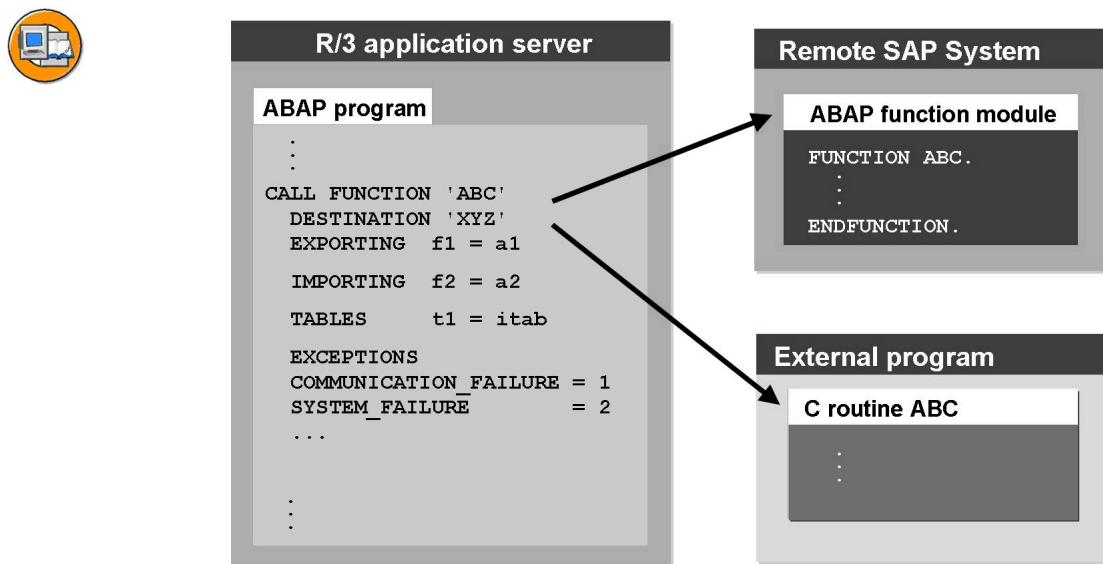


Figure 144: Synchronous RFC (Syntax)

The statement CALL FUNCTION ... DESTINATION enables you to call remote ABAP function modules or C routines in external server programs. When you call a function in this way, always include handling for the standard exceptions COMMUNICATION_FAILURE and SYSTEM_FAILURE.

The exception COMMUNICATION_FAILURE is resolved by the system if the specified destination in the RFCDES sideinfo table is not maintained, or if the connection to the remote system cannot be established.

The exception SYSTEM_FAILURE is resolved if the function module or C routine that you want to start in the remote system is not available.

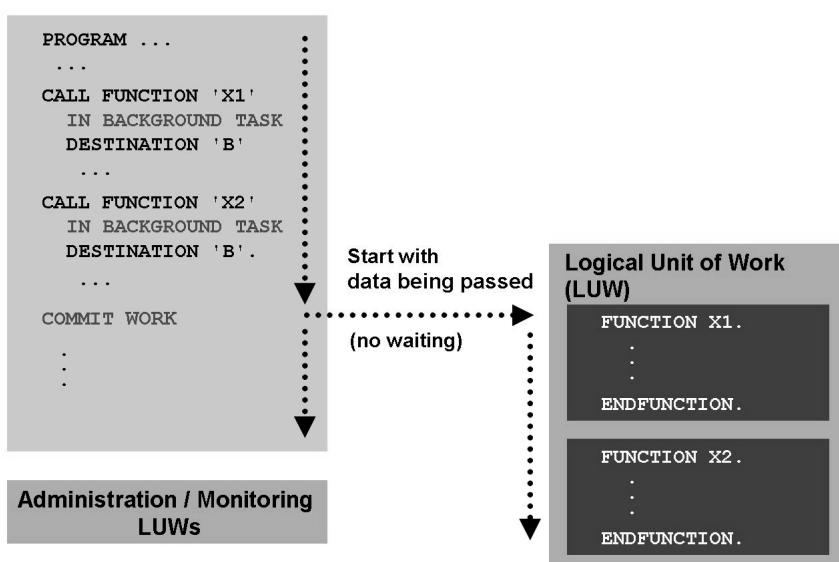


Figure 145: Process Flow: Transactional RFC (tRFC)

You can use transactional RFC (tRFC) to bundle several remote functions into one LUW (with an automatic rollback mechanism in case of error). With tRFC, generated LUWs are processed independently of each other. This means, the order in which they are processed is not always the order in which they are generated.

You can use tRFC with the addition IN BACKGROUND TASK, which you must place before the DESTINATION entry. If you specify a COMMIT WORK statement, you bundle all the previously transmitted tRFCs in one LUW.

tRFCs are called asynchronously. The output from the called function module cannot be received.

- No IMPORTING . . . / PERFORMING . . . ON END OF TASK when making the call.
- No RECEIVE RESULTS FROM FUNCTION . . .

In the source system, you can use the menu path *Tools → Administration → Monitor → SM58* or execute transaction SM58 that lets you display and modify tRFC-LUWs.

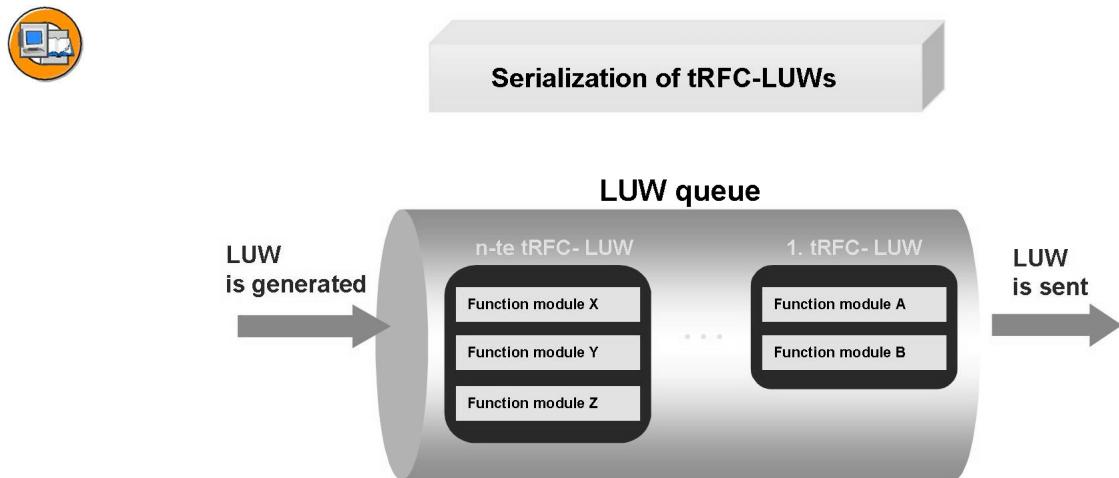


Figure 146: Process Flow: Queued RFC (qRFC)

To ensure that tRFC-LUWs are processed in the same order as they were generated, you can use qRFC as an extension of tRFC. qRFC is available as of Release 4.6A and can be used in SAP connections as well as SAP-external connections.

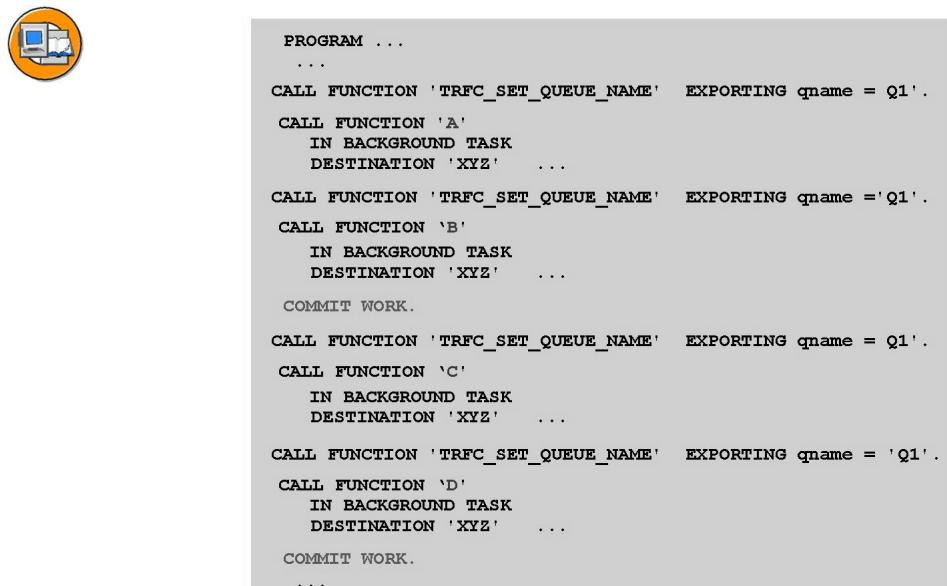


Figure 147: qRFC Syntax

To place tRFC-LUWs in a First-In-First-Out (FIFO) queue, you must specify the queue you want to use via the function module TRFC_SET_QUEUE_NAME before every single tRFC call. You can freely choose the queue name you need to specify, and it can have up to 24 characters. The queue name must not start with or contain:

- an empty character
- the % symbol, or
- contain the * symbol

Table RFCDES

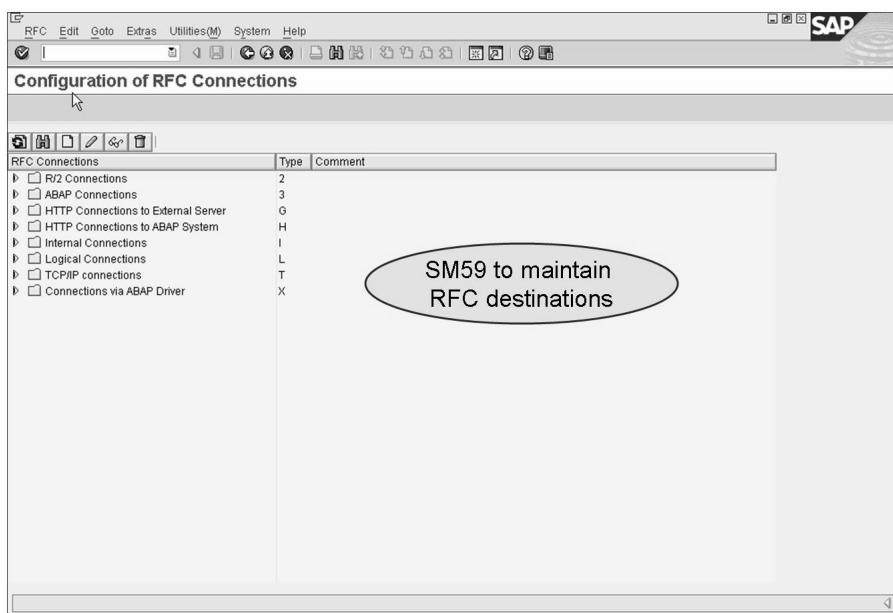


Figure 148: Maintaining the RFC Sideinfo Table RFCDES

You maintain the RFCDES using RFC sideinfo table Transaction SM59. Depending on the remote system, maintain one destination for the corresponding type. The initial screen displays a tree. Different connection types (example, partner systems or programs) are possible.

To display all information for a given designation, expand the type category, select the desired entry, or place the cursor on the entry and choose *F2*. You get a list of all entries matching your selection.

Remote destinations are stored in RFCDES table. The RFCDES table describes logical destinations for remote function calls. It is not possible to maintain the RFCDES table directly.

Creating Destinations

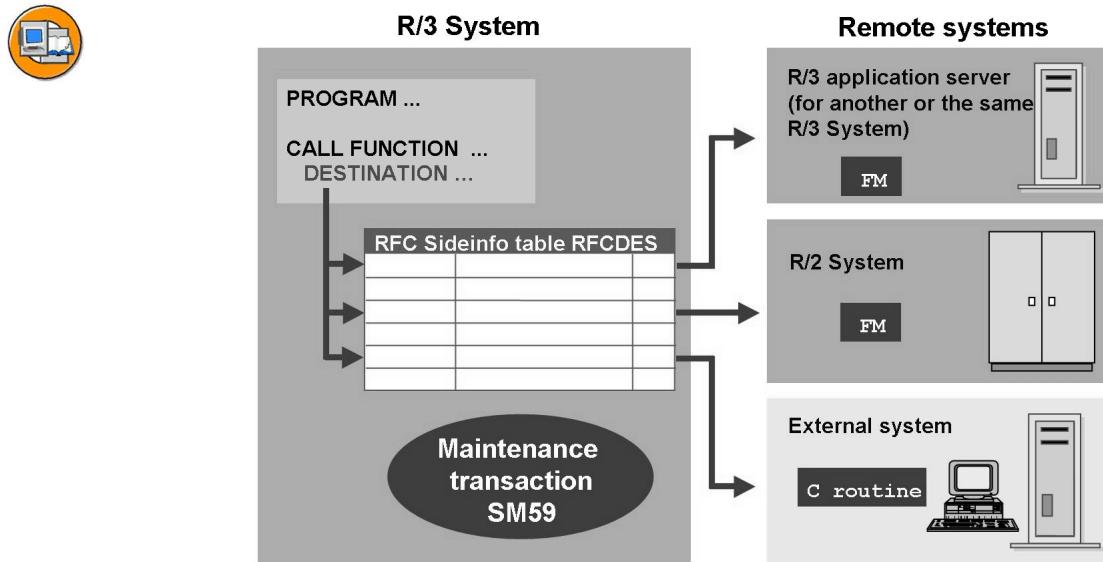


Figure 149: Remote Destinations (Table RFCDES)

From the destinations overview screen, select the *Create* button to create a new RFC destination. As you create the remote destination, you can specify a particular application server or a group of servers for a balanced distribution of system load.

Destination Parameters

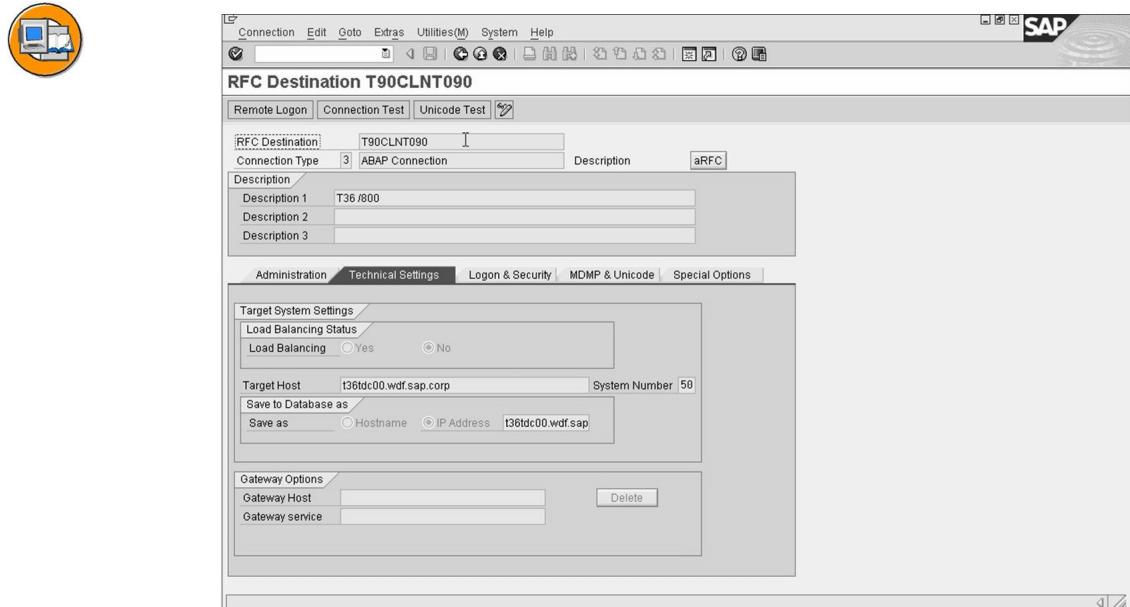


Figure 150: Destination for SAP-SAP System Connection

In addition to the RFC destination name, you must enter the following information:

Technical settings:

Connection type

Enter an existing connection type or choose one via the field entry help.
All available connection types are explained in Types of Destinations in Online Help.

Trace

Mark the Trace option to have the RFC communication logged and stored in a file. You can then display the file, both in the calling and receiving system, using RSRFCTRC report.

Load balance

If you choose load balancing, you must specify the following information:

Target system (for a list of available server, log on to the target system and choose *Tools → Administration → Monitor → System monitor → Servers*)

Message server (log on to the target system and choose *Tools → CCMS → Control/Monitoring → Control Panel*. It is the server that offers the service M. listed in the services column.)

Otherwise, you must specify the following information:

Target Host (the name of a server host of the target system that you want to use as a port to the system).

System number (Communications service used with the target system. To obtain it, choose *Tools → Administration → Monitor → System monitor → Servers*).

Security Options

The following options are available only with some connection types:

- Trusted system (for type 3 only)
If the target system is a trusted system, choose yes.
- SNC (Secure Network Communications, available for types 3 and T only).
If you have an active SNC-supported security system, you can activate additional security options which you must set via *Destinations → SNC options*.

Description

A description is the text description of the entry.

Logon:

Language

System language to be used

Client

Client code

User

User name to be used for remote logon, if different from current user name

Password

User password (be careful since passwords might be case sensitive in target system)

Current user

The current user name is to be used for remote logon

Unencrypted password (2.0)

If the target system is an SAP System or Release 2.0, the password must not be encrypted

If you want to connect to another SAP System, you need a destination with type 3. Note that the name of a type 3 destination is case-sensitive. If you want to ensure that the RFC logon is to a particular application server, set the Load distribution option to No. In this case, you must specify the SAP application server using the target host and system number parameters. If you do want to use load distribution when you log on, set the Load distribution option to Yes. In this case, you must specify the system ID, the message server, and the required server group. You can find out this information in the SAP target system from Transactions SM51 (overview of SAP servers) and SMLG (overview of logon groups). The message server of an SAP System is contained in the profile parameter rdisp/mshost. You can display this using Transaction RZ11 or report RSPFPAR.

In the destination, you must enter the logon data for the user created in the target system. When the destination is used, the logon to the target system occurs automatically. If the logon language is not entered in an RFC destination, the logon language of the current caller is used. If the logon data is not complete or is faulty, a logon screen from the target system appears at runtime.

Destinations SPACE, NONE and BACK

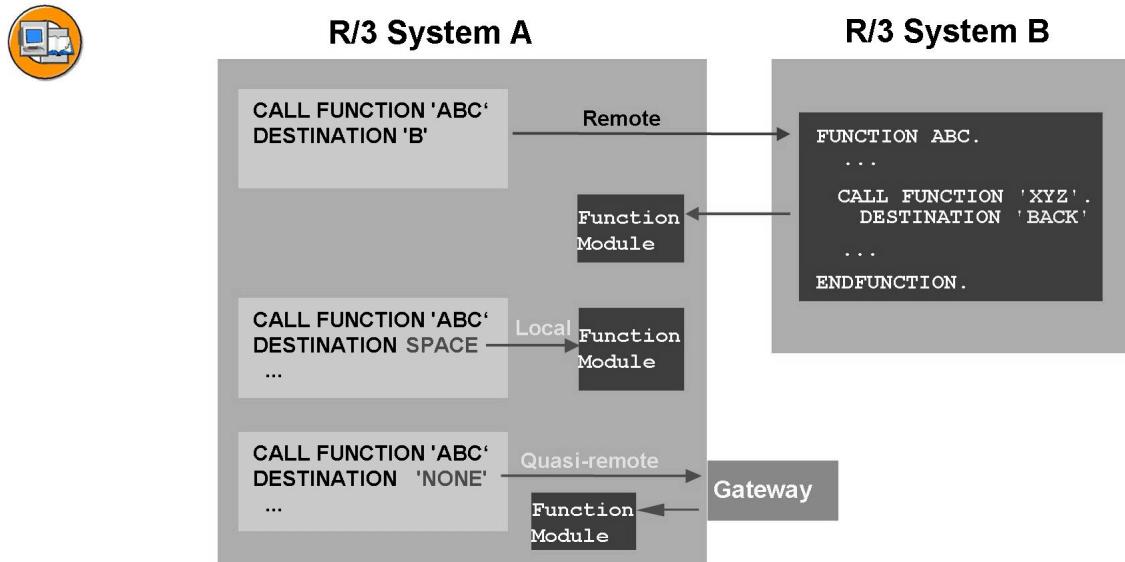


Figure 151: Destinations SPACE, NONE and BACK

These destinations are predefined standard destinations that have the following meanings:

SPACE

This is for a local call of a function module.

NONE

This is also a local call, but the call request is sent to the default gateway of the current application server and is handled there like an external call request. This destination is suited to the RFC test if you only have one system to test.

BACK

You can use this destination if you want to start a function module from the called function module in the current system. This only functions with synchronous RFC.

- **Note:** destination BACK cannot be used for SAP System— R/2 connections.

Security

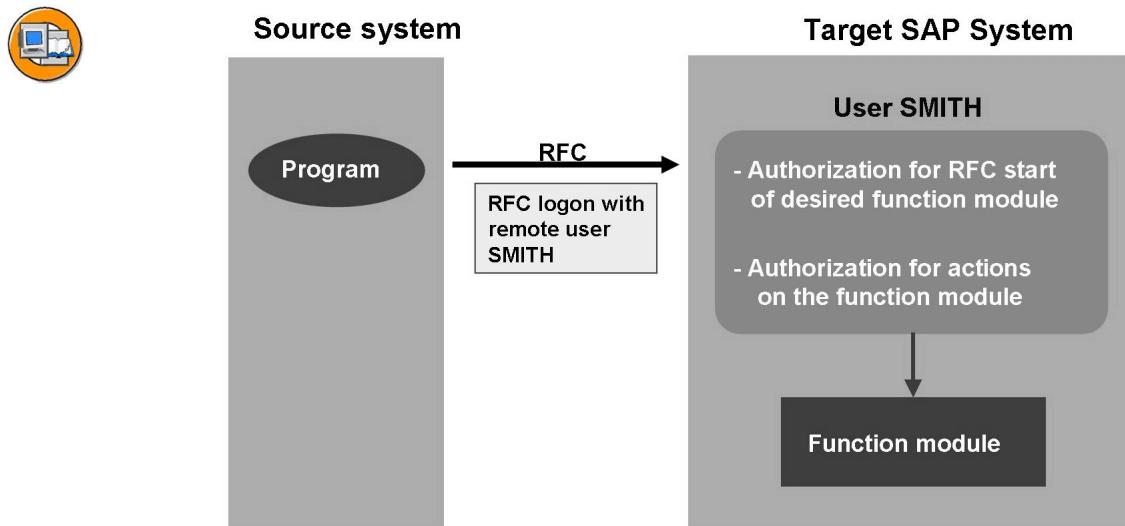


Figure 152: Authorization Check for RFC

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

Exercise 12: Create an RFC Destination

Exercise Objectives

After completing this exercise, you will be able to:

- Create a destination in your SAP System through which it can communicate with the target system using RFC.

Business Example

In a distributed business environment, it is sometimes necessary to keep data synchronized and/or to obtain data that is maintained on another system. Those target systems must be properly identified and have clear access paths.

Task:

Data that you require is stored on another SAP System. Create a definition that allows your source system to communicate with that target system (The names of those systems will be given to you by your course trainer).

1. Access the Remote Destination maintenance screen.
2. Begin the Destination definition process.
3. Specify an SAP System application server for the target system.
4. Enter the RFC logon data.
5. Save your work.
6. Test your RFC destination.

Solution 12: Create an RFC Destination

Task:

Data that you require is stored on another SAP System. Create a definition that allows your source system to communicate with that target system (The names of those systems will be given to you by your course trainer).

1. Access the Remote Destination maintenance screen.
 - a) To create destinations, choose *Tools* → *Administration* → *Administration* → *Network* → *RFC destinations* or enter transaction code SM59.
2. Begin the Destination definition process.
 - a) Choose the *Create* button. A screen appears, on which you should enter the name of the new destination **BC417_##**. Enter **3** as the connection type (target system is an ABAP System) and a meaningful description (example: group number ##'s RFC connection). Hit the <enter> key.
3. Specify an SAP System application server for the target system.
 - a) Log into the target system and execute transaction SM51. Find the server you want the RFC destination to connect to, and note the host name and system number.
 - b) Back in your RFC destination session, enter the target host name (example: iwdf5070) and the SAP System number (example: 00) for the server identified above.
4. Enter the RFC logon data.
 - a) Go to tab strip 'Logon & Security'. Fill in the 'Language', 'Client', 'Username' and 'Password' fields of the target server you want your RFC destination to log into.
5. Save your work.
 - a) Select the *Save* icon.



Note: Remember that passwords could be setup as case sensitive on remote server.

Continued on next page

6. Test your RFC destination.
 - a) Select *Test Connection* button. This returns a screen with time indications. This means that the test was successful.
 - b) Select *Remote Logon* button and you will receive the main SAP menu screen of the remote system. This means that the test was successful.
 - c) On the menu choose, *Utilities → Test → Authorization Test*. This will return a screen with time indications. This means that the test was successful and the user ID has the proper authorizations to be in the remote system.



Lesson Summary

You should now be able to:

- List the requirements for RFC calls
- Describe the types of RFC calls and their use
- Use an RFC call to access a remote system

Lesson: Web Services

Lesson Overview

SAP Web Service

This lesson provides an Introduction and overview of Web Services



Lesson Objectives

After completing this lesson, you will be able to:

- Describe the basic technology surrounding Web services.
- Expose a BAPI as a Web Service using the Web Service Wizard.

Business Example

A user needs to make an external Internet call into SAP System to get Customer Information. He'd like to use standard web technology for the connection between his system and the SAP system.

What is a Web Service ?

Business processes are divided into a certain number of process steps. You can assign one or more functions to each of these steps and an executing software component to each of these functions. If you look at a typical heterogeneous system landscape in an organization, it is quickly apparent that the necessary functions in a business process are not all implemented using the same technology and the same components.

In particular, the integration of an ever increasing number of business partners further complicates this problem. A modern software infrastructure must therefore be capable of integrating functions that are implemented on very different software components into an efficient global process.

Internet technology already provides the basis for communicating with distributed services. Superimposed onto this simple, globally accepted communication standard, XML (eXtensible Markup Language) provides the basis for defining additional necessary standards. It is only when we turn away from proprietary definitions and move towards generally accepted standards that there can be any guarantee of smoothly integrating all of the functions and partners involved in the process. The result is Web services.



Web Services...

- Are application functions/services
- Can be used using Internet Standards
- Are self-descriptive
- Can be published and traced
- Form the basis of ESA

Figure 153: Web Service Basics

A Web service is an independent, modularized, self-describing application function or service. Based on XML standards, these application functions can be described, made available, located, transformed, or called via standard Internet protocols. Each Web service therefore encapsulates a piece of functionality that can be used, for example, to forward a price query to a provider, check the availability of an item in an enterprise resource planning system, locate a telephone number, or even to run credit card checks, convert currencies, or implement payroll functionality.

The Web Service Paradigm

The provider of a service is generally called a Service provider. If the service is a Web service, the service provider will have a corresponding XML-based description (WSDL document). In principle, any programming language can be used to implement this service. Based on the HTTP transport protocol, Simple Object Access Protocol (SOAP) is now established as the quasi-standard access protocol. In a client/server relationship, the service provider can be regarded as the server.

When publishing a service, the service provider transmits information about itself and a description of the service it is offering and transfers this to the service registry. A service registry can be described as a type of Yellow Pages. for Web services. In addition to other data, it also provides information on calling the Web service, for example. The service registry therefore provides a description of the Web service only. This description forms an abstraction layer and is therefore not dependent on the corresponding implementation. The Web service itself is hosted by the service provider.

The user of a Web service is called a service requester. A service requester can be, for example, someone who locates a Web service using a Web browser and then uses this service. In most cases however, the service requester is an application that accesses the Web service. The application can also bind. to the service on request, that is, the application can dynamically create a Web service client proxy at runtime in order to access the Web service. The application obtains the

necessary information for this from the service description, which is in turn stored in the service registry. However, if the application recognizes the provider and the call details, it can obviously use the Web service without having to access the service registry. In a client/server relationship, the service requester is the application client.

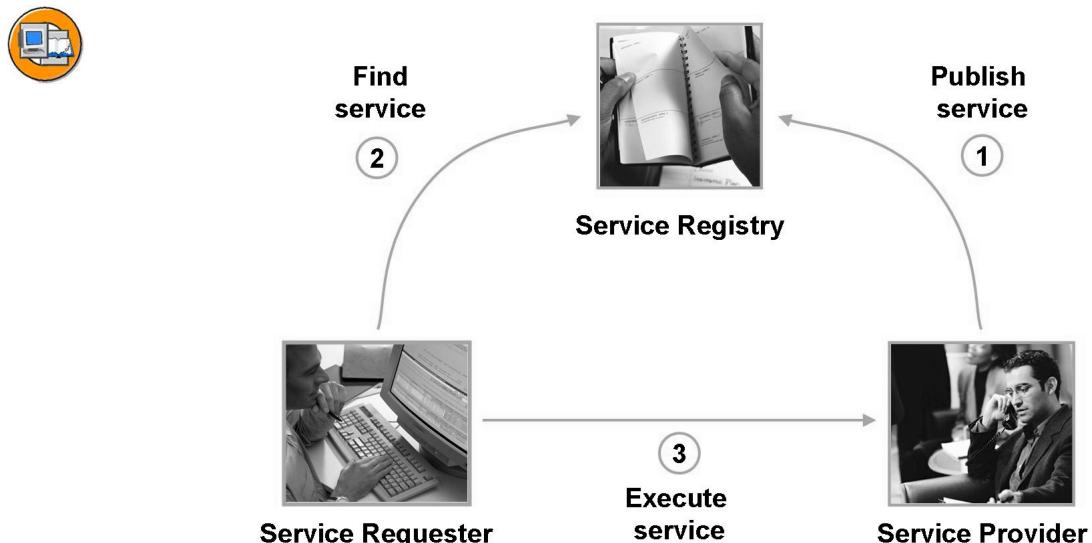


Figure 154: Typical Web Service flow

Central Internet Standard for Web Services

Web services can exist in any implementation. If Web services are to be called from any application, a standardized description is required. Web Services Description Language (WSDL) has been shown to best meet this demand.

A Web services description in WSDL alone, however, is not sufficient. To find the right business partner and corresponding service quotation, you will need a register of companies. to help you to find the service you need. The Web service provider must also be able to make its offer publicly available as easily as possible. Universal Description, Discovery and Integration (UDDI) offers a solution. See .

UDDI provides the necessary tools with its UDDI Business Registry and UDDI specification. The specification provides a detailed description of how to locate and register services. The UDDI Business Registry contains a list of registered companies with their service offerings. The UDDI Business Registry is accessed either manually via Internet pages or via XML-based messages, which are described in the UDDI specification. Examples:

An appropriate protocol definition is required to call Web services based on Internet technologies. SOAP (Simple Object Access Protocol) provides a more straightforward standard that allows you to call Web services in decentralized, distributed landscapes. Similar to the standards discussed earlier, SOAP is also

based on an XML language definition. SOAP defines what is known as an Envelope. This Envelope contains the actual XML-based message and additional information on how the message is to be processed, for example. A further series of conventions was also adopted for describing the technical constraints.

XML (eXtensible Markup Language)

XML is an extensible markup language for exchanging structured documents over the Internet. XML documents are increasingly used to support the exchange of business documents and messages, and thereby strengthen cooperation between companies.

SOAP (Simple Object Access Protocol)

SOAP specifies a package of XML documents for transport via Internet protocols such HTTP(S), SMTP, or FTP. This protocol is used to call Web services in distributed system landscapes. A SOAP message has a header (additional information concerning security and transaction) and a body (content of the message).

WSDL (Web Service Description Language)

WSDL is an XML-based description language for Web services. WSDL documents are broken down into the names of the services, messages that are exchanged to use these services, links to specific transport protocols, and addresses at which a Web service is available. WSDL is an integral part of, and is used by, UDDI.

UDDI (Universal Description, Discovery and Integration)

UDDI is a Web-based registry that can be accessed via the Internet. The registry consists of a list of Web services in WSDL format and is used to locate these services. UDDI is different from other registry services insofar as it does not store documents or specifications, but only references them.

Web Service Creation Wizard

To create an ABAP Web service in the ABAP Workbench we use the Web Service Creation Wizard. Once a process and an interface which should be made available are identified, the easiest way to start is with the *Web Service Creation Wizard* in the *ABAP Workbench*. With this wizard, configuring a Web service is a highly automated process based on predefined configuration profiles developed by SAP that bundle settings used in typical Web services scenarios. With this wizard, even a developer who is less experienced with detailed technical aspects of Web services can still create one simply by clicking through the three steps of the wizard. All of the required objects are generated in the background.

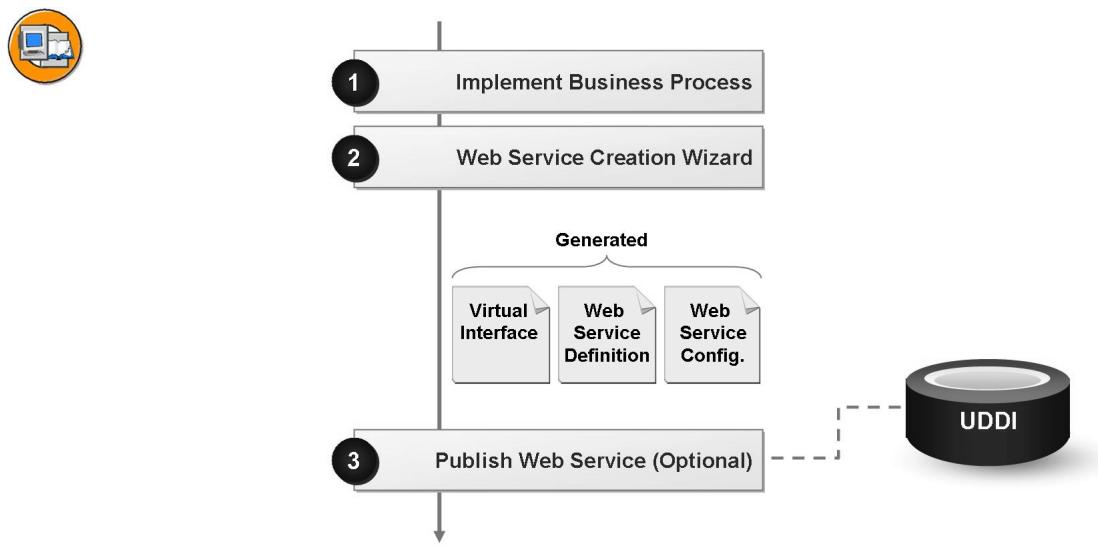


Figure 155: Web Service creation process

The following objects are generated by the *Web service creation wizard*:

Virtual Interface

A virtual interface (VI) is the visual representation of a Web service to the outside world. Using VIs, you can define several views of one implementation (Web service end point, e.g. RFC-enabled function module) and publish them separately as a Web service. When you create a virtual interface, you can hide or rename operations and parameters. For example, you can replace technical names with descriptive names. You can also define default values for parameters and convert parameter types. At present, a virtual interface can be created for the following objects: RFC-enabled function module, function group (with RFC-enabled function module), BAPI, and XI message interface.

Defining and Configuring a Web Service

Features such as the communication type, authentication level, and transport properties are assigned in abstract form in the Web service definition (WSD). The technical details are specified during administration and the release of the Web service for the SOAP runtime. The assignment of abstract features ensures that a Web service definition can be used for different application servers that are configured differently. The proxy generation on the client side relates to the Web service definition. Technical details specified during configuration of the Web service are configured separately in the Web service client runtime. For one virtual interface, you can create several Web service definitions with differing features.

Using the Web Service Creation Wizard for a BAPI

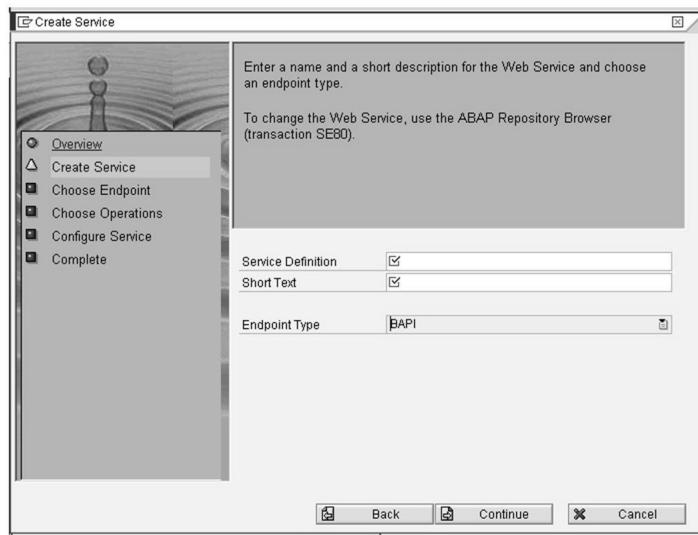


Figure 156: Web Service Creation Assistance Wizard (1)

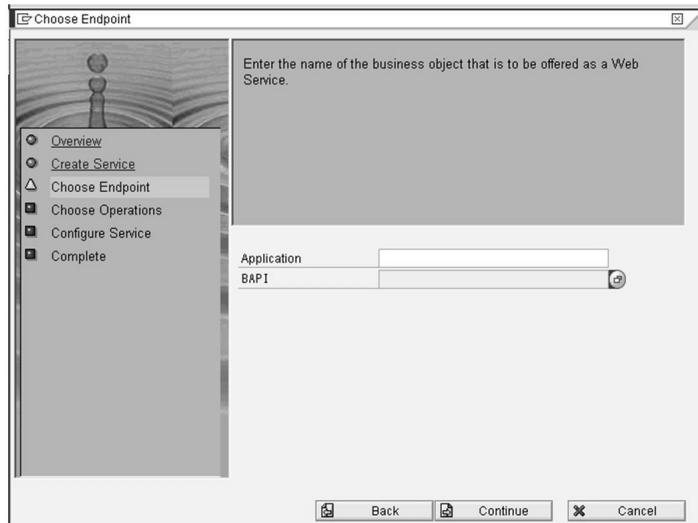


Figure 157: Web Service Creation Assistance Wizard (2)

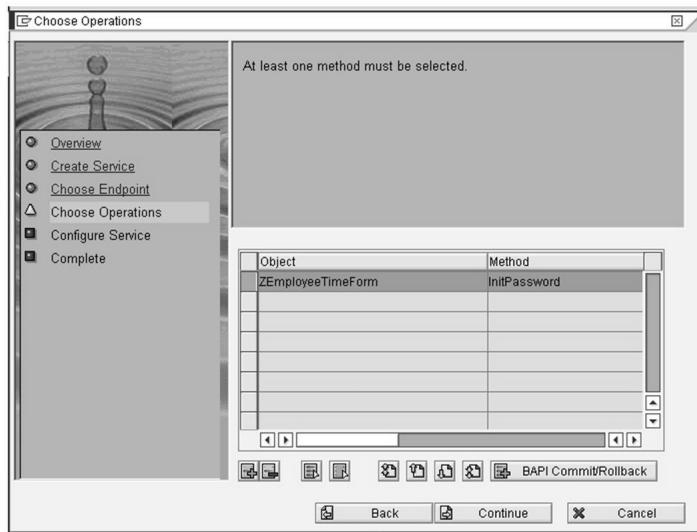


Figure 158: Web Service Creation Assistance Wizard (3)

You can start the Web service creation wizard using the WS_WZD_START transaction, or within the Object Navigator (SE80) using the context menu for the package (*Create → Enterprise Service/Web Service → Web Service*). First, enter a name and short description for the web service. You must also specify the end point type and click continue. Next, enter the name for the Business object that is offered as a Web service and click continue. In the next screen select at least one method and click continue. Finally, select one of the available profiles for the predefined feature quantity for the Web service definition. You can choose from the following options:

Basic Authorization SOAP Profile: The communication type is stateless, and the caller is verified by a user name and password.

Secure SOAP Profile: Again, the communication type is stateless. Authorization occurs via client certificates, and data is transmitted in encrypted form using the Secure Socket Layer protocol (SSL).

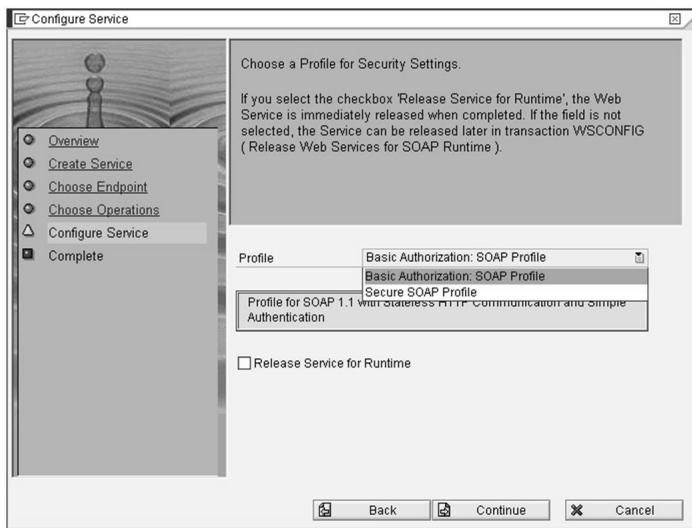


Figure 159: Web Service Creation Assistance Wizard (4)

With a few clicks of the mouse you can create the Web service definition using the Web service creation wizard. The release for the SOAP runtime occurs automatically. In the object list an entry is included for the generated objects. To release the Web Service definition for the SOAP runtime use the WSCONFIG transaction and enter the name of the web service definition in the *web service definition* field. Choose *Create*.

For function groups and function modules, the Web service creation wizard can also be called from the function library (SE37). Choose the BAPI, display it, and from the menu choose *Utilities* → *More Utilities* → *Create Web Service* → *From the Function Module* or From the Function Group. Please note that function groups must contain at least one RFC-enabled function module. To call the wizard for a BAPI, you can position the cursor on the required BAPI in the BAPI Explorer. Select the Tools tab page followed by Create Web Service. Then choose Start Wizard.

Web Service Home page

The Web service home page offers resources for developing and utilizing Web services. You can use the home page, for example, to test Web services. A prerequisite for calling the home page is the availability of a J2EE server on the application server. The Web service home page provides an interface to a Web service client and offers the functionality to send a SOAP message to the application server. The application server receives this SOAP message, extracts the input parameters, and executes the relevant Web service. The result is sent back to the Web service home page via SOAP, and is displayed there. You can also view the SOAP request and SOAP response on the Web service home page.

You access the Web service home page via the transaction WSADMIN. Here, choose the Web service definition service you require, and start the Web service home page using the test icon on the application tool bar.

Uses of WSDL (Web Service Description Language)

The same Web service can be called from different systems, independently of their technical features. The implementation of the Web service client is not dependent on the technical structure of the server. Consumers are generated from an implementation-independent service definition. The technical details are configured in the Web services consumer's runtime environment.

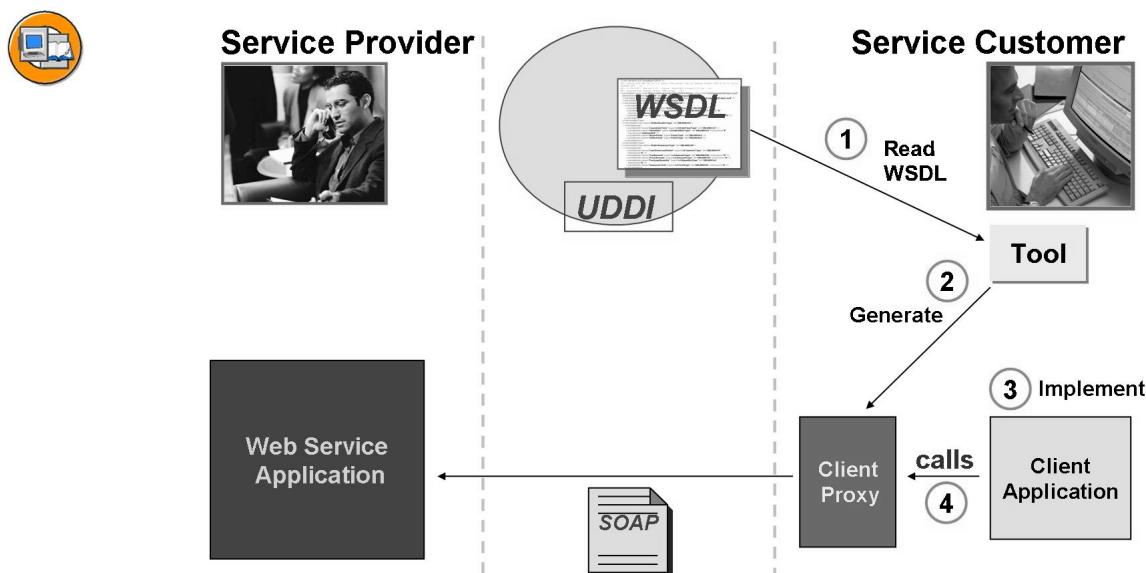


Figure 160: Function of Web Service Definition

The developer, on the part of the consumer, obtains the description of the Web service in the form of a WSDL document. The developer creates a Web service proxy using a programming language generation tool. This encapsulates Web service access, for example, as a class that can be used in the application, at a suitable point, and as often as is required. At runtime, the client proxy undertakes communication with the Web service, for example via the SOAP protocol.

The following elements are outlined in a WSDL:

- **Definition** - the logical root element in which the different namespaces are defined as attributes.
- **Types** - a container for data type definitions when using a typing system, for example XSD (XML Schema Definition).
- **Messages** - an abstract name for communicated data. A message element consists of a number of part elements. Every part element refers to a type defined in the types element. A part can represent a message parameter or a function call parameter.
- **Operation** - Operation is an abstract name for the possible actions of a service. Every operation consists of its name and one to three messages - an input, output, and possibly fault message. Depending on whether an operation involves an input and output message, or only one of these two messages, there are different types of operations in a port type (one-way, notification, etc.):- One-way: The operation can contain a message, but it does not return a message.- Request/Response: The operation identifies an input message and returns a message.- Expected response: The operation can send a message, and waits for a response.- Message: A message can be sent, and no response is expected.
- **Port Type** - an abstract grouping of operations from one or more ports
- **Binding** - a concrete protocol and data format definition (SOAP, http-GET, etc.) for a specific port type
- **Port** - a single port as a combination of a binding and a network address
- **Service** - a grouping of used ports; grouping of related interfaces, consisting of "ports" for a service
- **Part** - abstract definition of a parameter

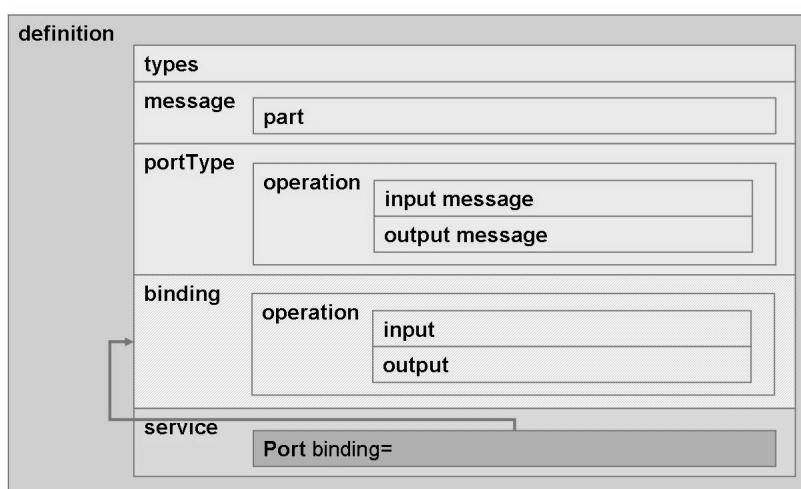


Figure 161: Structure of WSDL Document

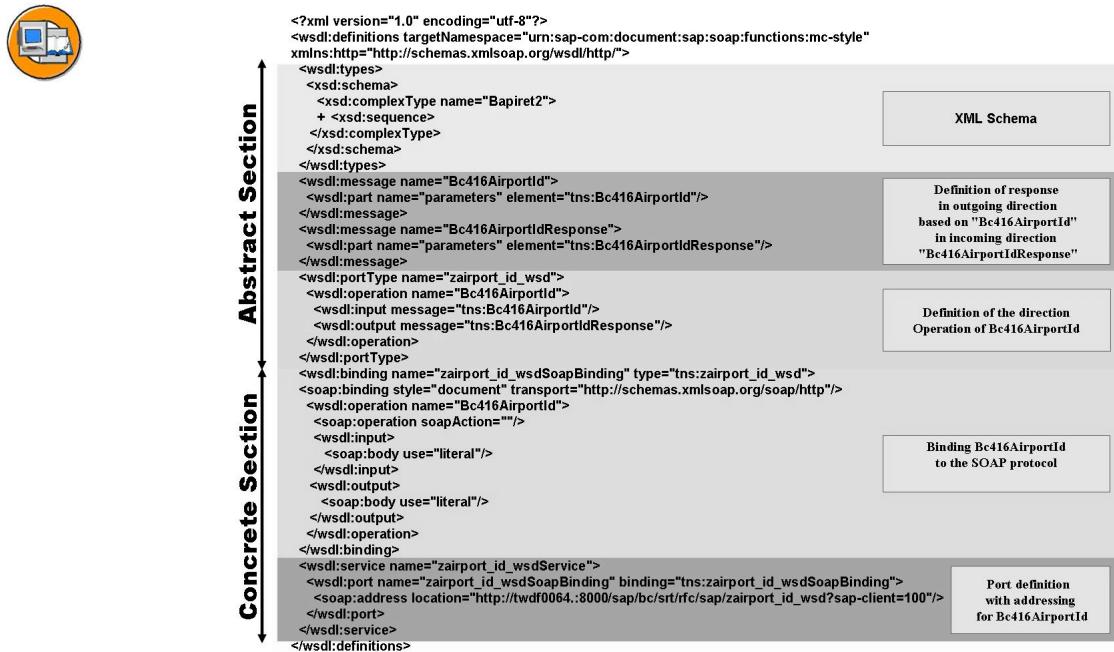


Figure 162: WSDL Document Detailed Example

The WSDL of a Web service generated under ABAP can be viewed with the WSADMIN transaction. Use *Web Service → WSDL* or the circled WSDL icon to print the WSDL in a browser window.

SAP Exchange Infrastructure

Purpose : SAP Exchange Infrastructure (SAP XI) enables you to implement cross-system processes. It enables you to connect systems from different vendors (non-SAP and SAP) in different versions and implemented in different programming languages (Java, ABAP, and so on) to each other. SAP Exchange Infrastructure is based on an open architecture uses open standards (in particular those from the XML and Java environments) and offers those services that are essential in a heterogeneous and complex system landscape:

- Modeling and design of messages, transformations, and cross-component integration processes
- Configuration options for managing collaborative processes and message flow
- Runtime for message and process management
- Adapter Engine for integrating heterogeneous system components
- Central monitoring for monitoring message flow and processes

SAP XI supports internal company scenarios and cross-company scenarios.

Features

SAP XI is based on general standards so as to enable external systems to be integrated. At the center of the infrastructure is an XML-based communication that uses HTTP (Hyper Text Transfer Protocol). The application-specific contents are transferred in messages in user-defined XML schema from the sender to the receiver using the Integration Server.

Senders and receivers that exchange messages using the Integration Server are separated from one another. This separation makes it easier to connect systems that are technologically different. Every system that can exchange messages with the Integration Server can also exchange messages with all other systems that are connected to the Integration Server. SAP XI supports the following methods of communication with the Integration Server:

- Direct communication using proxies, which you generate in the application systems using a description in WSDL (Web Service Description Language).
- Communication using adapters. In this case, you create interfaces for message exchange in the application system, or use existing interfaces.

Simple message processing on the Integration Server is stateless. This means that the Integration Server does not know of any connections between various messages. Cross-component integration processes, on the other hand, describe related processes, which can use the knowledge about messages that have already been processed to further control the process (for example, waiting for the corresponding response for a message in order to start further actions). You can use SAP XI to model, change, and manage these cross-component integration processes centrally. These processes are executed on the Integration Server and are included in message processing by configuration.

As with cross-component integration processes, you save the entire integration knowledge of a collaborative process centrally in SAP XI: Objects at design time in the Integration Repository and objects at configuration time in the Integration Directory. In this way, SAP Exchange Infrastructure follows the principle of shared collaboration knowledge: You no longer need to search for information about a collaborative process in each of the systems involved, but can call this information centrally instead. This procedure considerably reduces the costs for the development and maintenance of the shared applications.

Setting Up Point-to-Point Connections

The Integration Server provides a variety of services that are required in cross-system processes, for example, routing, integration processes, and mappings. In those cases where such services are not required, communicating by using the Integration Server unnecessarily slows down message exchange. This is usually the case for new applications where both the sender and receiver use message interfaces to communicate. In such cases, mappings are often not required because the outbound and inbound messages use the same message type.

The process below explains how client and server proxies, which were generated by means of message interfaces from the Integration Repository, can communicate with each other directly by using the Web service runtime. This has the following advantages:

- Performance is improved by bypassing the Integration Server.
- If the services of the Integration Server are required at a later date, it is possible to switch from the Web service runtime to the XI runtime, without having to change the program code

Process flow

1. Create a Web service.
2. Log on to the source client of the sender system (client side).
3. Create a corresponding HTTP destination by calling transaction SM59.
4. Enter the following data on the Technical Settings tab page:
 - Enter the path of the Web service that you released in transaction WSADMIN as the Path Prefix.
 - Specify a Target Host (including the port, if applicable).
5. Enter the logon data on the Logon/Security tab page and save your entries.
6. In the same client, call transaction LPCONFIG and create a logical port for the client proxy.
 - On the Runtime tab page, choose Web Service Infrastructure.
 - Assign the http destination to the logical port.
 - c. Activate the logical port.

You can now send messages directly to the logical port and the server proxy by using the client proxy call in the source client of the sender system

Sending a message

Use

You can use a client proxy to do the following:

- To call a service using the SAP XI Integration Server
- To call a Web service

How the server proxy is used depends on the runtime configuration. The general programming model of the ABAP proxy runtime supports synchronous communication. The XI runtime also supports asynchronous communication.

Procedure

To send a message using the ABAP proxy runtime, call the corresponding client proxy in your application program.

1. Declare the following variables: DATA:

```
* Reference variables for proxy and exception class
lo_clientProxy      TYPE REF TO [Generated proxy class],
lo_sys_exception    TYPE REF TO cx_ai_system_fault,
* Structures to set and get message content
ls_request          TYPE [Output message type],
ls_response         TYPE [Input message type].
```

2. Complete the structure ls_request for the request message.

3. Instantiate your client proxy.TRY.

```
* create proxy client
CREATE OBJECT lo_clientProxy( 'LOGICAL_PORT_NAME' ).
```

 **Note:** LOGICAL_PORT_NAME is the name of the logical port that you want to use, which is used to define the receiver. You can omit this parameter if you are using a default port or the XI runtime (see runtime configuration).

4. To send a message, call the corresponding client proxy method.

WSDL allows several such methods (specified by the element <operation>). In XI, there is only one method, with the default name EXECUTE_SYNCHRONOUS or EXECUTE_ASYNCNCHRONOUS. Catch at least the exception cx_ai_system_fault:

```
* do synchronous client proxy call
```

```
CALL METHOD lo_clientProxy->execute_synchronous
      EXPORTING output  = ls_request
      IMPORTING input   = ls_response.
      CATCH cx_ai_system_fault INTO lo_sys_exception.
*   Error handling
ENDTRY
```

Runtime Configuration

Use

Before messages can be exchanged at runtime, you must define whether you want to use the Web service runtime or the XI runtime.

Integration

The ABAP proxy runtime supports the following approaches:

- XI customers can use the XI runtime to define the receiver or receivers of a message centrally in the Integration Directory. You can then use the integration logic of the Integration Server (routing, mapping, and Business Process Management). (Also see: Configuration).
- SAP Web AS customers and XI customers can use the Web services runtime to call Web services over the Internet for simple services. You must then configure the receiver locally in the client system. Furthermore, you must encapsulate server proxies as a Web service by using virtual interfaces.

If you require a special protocol, which the other runtime does not provide, it is also possible to support both runtimes in your application program.

Features

Local Runtime Configuration

- You must create a logical port for client proxies that use the Web Service Runtime. You create this port in the system in which you want to call the client proxy. Call transaction LPCONFIG and activate the Web services infrastructure on the Runtime tab page. Using the logical port, you can define the receiver data and the settings for error handling. It is also possible to create multiple logical ports for a client proxy and then select one of them dynamically.
- Client proxies that use the XI runtime do not require a logical port. However, you may still want to create a logical port because, for example, a customer wants to be able to switch between the XI runtime and the Web service runtime as required. To set the XI runtime for a port, choose Exchange Infrastructure on the Runtime tab page.

Evaluating at Runtime

You can specify a logical port by using the constructor of a client proxy. The ABAP proxy runtime determines which runtime must be active as follows:

- If no port is specified and there is no default port, the system assumes that the message is to be sent by using XI.
- If a port is specified, but it is not available, the system raises an exception. In all other cases, the port settings described above determine the runtime that the system will use to send the message.

Exercise 13: Making your BAPI Web Service enabled

Exercise Objectives

After completing this exercise, you will be able to:

- Exposing your BAPI as a Web Service

Business Example

External systems non-SAP systems, need to call your BAPI using a standard Web based protocol.

Task:

Generate a Web Service wrapper to your RFC enabled BAPI
Z_BAPI_CONTACT_GETLIST_##.

1. Create an ABAP Web service in the ABAP Workbench using the Web Service Creation Wizard.
2. Release the Web Service definition
3. Generate the WSDL for the Web Service and Test the Web Service.

Solution 13: Making your BAPI Web Service enabled

Task:

Generate a Web Service wrapper to your RFC enabled BAPI
Z_BAPI_CONTACT_GETLIST_##.

1. Create an ABAP Web service in the ABAP Workbench using the Web Service Creation Wizard.
 - a) Go to the Object Navigator (SE80). Find the package that you saved your RFC BAPI function modules in (ie ZBC417_##).. On the context menu for your package,, select Create →Enterprise Service / Web Service →Web Service.
 - b) The Web Service Creation Wizard appears, specifying that with this wizard, you can create web service definitions for various components such as RFCs and BAPIs. Hit the <continue> button.
 - c) On the 'Create' screen, enter the service definition name 'ZWS##_CONTACT_GETLIST' and the short text 'GR## contact getlist WS definition'. Specify the 'End Point' as 'BAPI'. Hit the <continue> button.
 - d) On the 'Choose Endpoint' screen, leave the 'Application' field blank and for the BAPI field, pick your Contact business object (ie ZCON_##). Hit the <continue> button.
 - e) On the 'Choose Operations' screen, select your GetList method and hit the <continue> button.
 - f) On the 'Configure Service' screen, select the 'Profile' 'Basic Authorization : SOAP profile'. Select the check box 'Release Service for runtime'. Hit the <continue> button.
 - g) On the 'Complete' screen, hit the <complete> button.
 - h) A Web service definition has been successfully created using the Web service creation wizard.

Continued on next page

2. Release the Web Service definition
 - a) Go to Transaction WSCONFIG which is used to release the web service
 - b) Type in the name of Web Service Definition (ie ZWS##_CONTACT_GETLIST) and give the Variant name ZWS##_CONTACT_GETLIST.
 - c) Click on create.
 - d) Click on the <save> button and acknowledge that you want to create an external alias for the ICF.
3. Generate the WSDL for the Web Service and Test the Web Service.
 - a) Goto Transaction WSADMIN which is the home page for Web Services.
 - b) Under the SOAP Application for BAPIs area, you will find your Released Web Services in the list.
 - c) Expand the sub-tree and select the web service definition you have created for which you need the WSDL.
 - d) Click on the circled WSDL icon or use the menu path Web Service → WSDL
 - e) In the popup, pick the radio button 'Document Style'. This will open up an Internet Explorer session, showing you the content of your WSDL web service definition.
 - f) Back in WSADMIN, select the web service definition you have created and click on the <test/execute> button.
 - g) In the following Dialog box, choose Document style
 - h) Enter your user name and password if the Web Service requires authentication.
 - i) On the Overview page, the features of the Web Service will be displayed.
 - j) Call the Test page and select the method to test (ie your GetList method).
 - k) Fill in value for the method input parameter 'Country' and hit the <send> button. The data returned by the Web Service/BAPI will be displayed under the *Response* heading.



Lesson Summary

You should now be able to:

- Describe the basic technology surrounding Web services.
- Expose a BAPI as a Web Service using the Web Service Wizard.



Unit Summary

You should now be able to:

- List the requirements for RFC calls
- Describe the types of RFC calls and their use
- Use an RFC call to access a remote system
- Describe the basic technology surrounding Web services.
- Expose a BAPI as a Web Service using the Web Service Wizard.

Related Information

- For additional information on developing Business Objects, refer to the BAPI programming Guide for Ecc6.0 on SAPNet.
- For additional information on Web Services from ABAP, look for course BC416.
- For additional information on XI, look for course BIT400.



Test Your Knowledge

1. If a function module is called remotely, it runs in its own work process if the remote system is an SAP System.

Determine whether this statement is true or false.

- True
- False

2. You can declare a trust relationship between two SAP Systems so that you can perform certain RFC logons in the trusting system without a password for released users.

Determine whether this statement is true or false.

- True
- False



Answers

1. If a function module is called remotely, it runs in its own work process if the remote system is an SAP System.

Answer: True

When calling a function module remotely in an SAP System, the function module runs in its own work process.

2. You can declare a trust relationship between two SAP Systems so that you can perform certain RFC logons in the trusting system without a password for released users.

Answer: True

If you declare a trust relationship with a remote system, you can perform certain logons in the trust system without a password.

Internal Use SAP Partner Only

International Use SAP Partner Only

Unit 5

Enhancement of SAP supplied BAPIs

Unit Overview

An SAP developer of a BAPI could preplan in his BAPI, an area where a customer can add his own logic in order to extend the BAPI. In many cases, BADI (Business-AddIn) technology was used for this purpose.

This unit will show students how SAP software developers will typically prepare their BAPIs for enhancement capabilities using BADIs (Business-AddIns).

This unit will also show customers how to implement the BADI.

Finally, this unit will also show customers how to, if needed, redefine a method in the BOR, so that it now calls a customer specific method instead of the SAP supplied method.



Unit Objectives

After completing this unit, you will be able to:

- Identify if an SAP BAPI has Classic BADI customer extension capabilities.
- Implement customer specific processing in an SAP BAPI using a predefined BADI.
- Create a subtype of an SAP Business Object and redefine one of the methods within the subtype.

Unit Contents

Lesson: Planning a BADI in a BAPI	284
Exercise 14: Prepare BAPI for enhancement	291
Lesson: Implementing a BADI of a BAPI	296
Exercise 15: Implement BADI for BAPI.....	303
Lesson: Redefining a method in the BOR	307

Lesson: Planning a BADI in a BAPI

Lesson Overview

In this lesson we will see one possible technique that an SAP developer could use within his BAPI so that customers can extend the default SAP supplied business logic with their own.



Lesson Objectives

After completing this lesson, you will be able to:

- Identify if an SAP BAPI has Classic BADI customer extension capabilities.

Business Example

Customers often have to make some changes to the standard system supplied by SAP to meet their own special requirements. This may also be true for a BAPI. For example, a new parameter may have to be added to a BAPI or a change may be needed to the function module coding.

For a customer to be able to add their own logic to an SAP BAPI, the SAP developer pre-plans in his BAPI this functionality. In many cases, SAP developers have used Classic BADI technology for this purpose.

Customer Enhancement

When making a change to a BAPI, you should use the customer enhancement concept instead of the modification concept whenever possible, since changes to the standard BAPI implemented by SAP (in a new release, for example) will be activated automatically in an enhanced BAPI. In contrast, if the BAPI has been modified, these changes will not be activated automatically.



Customer enhancements of BAPIs

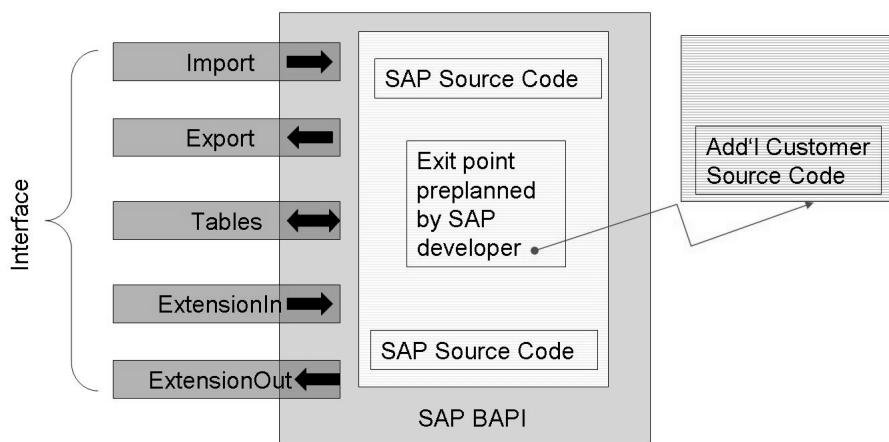


Figure 163: Customer enhancements of BAPIs principles



Customer enhancements types



Figure 164: Customer enhancement types

Customer enhancements are passed on to the BAPI via containers located near the interface, and are thus, not directly visible in the interface. This concept is useful when customers want to enhance the underlying database tables for a BAPI, or when they want to use an existing BAPI to manipulate their own tables.

 **Note:** This concept described was new in Release 4.5B. As a result, not all of the existing BAPIs have the containers required for enhancement in their interfaces. In these cases, either modify the BAPIs or extend them in accordance with the old concept. Customer exits that are implemented using the enhancement concept do not have to be reprogrammed. BADI technology has evolved. Within the scope of this course, we will look at the Classic BADI technology. The new Kernel based BADI technology is not covered here. SAP Enhancements are covered in courses BC425 and BC427.

The following table lists some of the advantages and disadvantages of customer enhancements.

	Customer Enhancement
SAP changes to the BAPI activated automatically	Yes
SAP changes have no effect on own enhancement	Yes
Own changes to BAPI interface remain explicitly visible	No
Changes to SAP coding are possible	No

BADI definition basics

BADI (or Business Add-Ins), is an enhancement type that was introduced in version 4.x of R/3. The basics behind BADI is simple : **the SAP developer creates in interface, customer creates an implementing class(es) for that interface.**

The SAP developer uses transaction SE18 in order to define and document the BADI definition. Via this BADI definition, he will create an ABAP Object Interface, in which he will specify a list of method names and signatures for each method. The customer will later be able to, if desired, implement these methods..

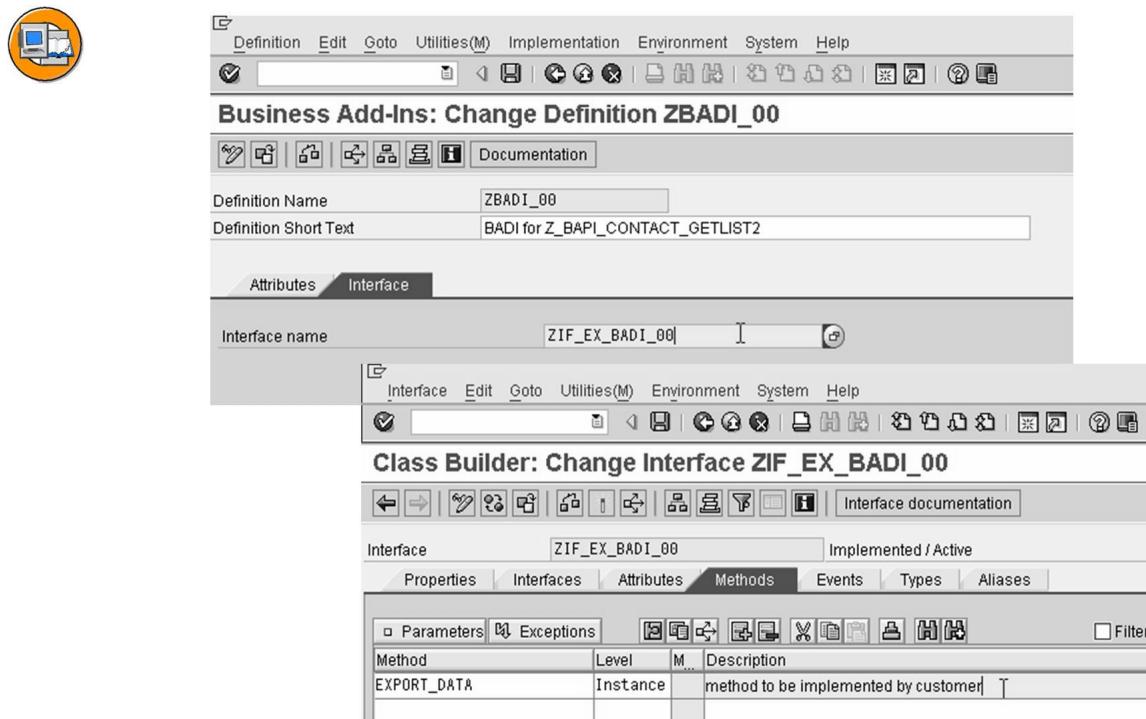


Figure 165: Defining a BADI and the object Interface

In our current example, the SAP developer creates a BADI definition for 1 method that will be implemented by the customer. The SAP developer will pass to this customer method the country field. It will be up to the customer to then fill the internal table *extension_out* with his own data via his own algorithm.

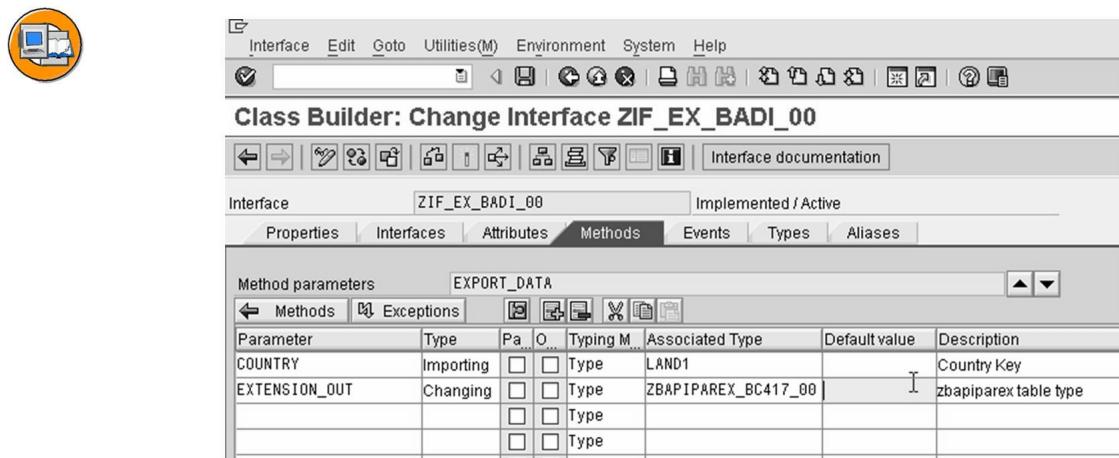


Figure 166: Defining the BADI interface method signature

Coding required inside of BAPI for BADI capabilities

The SAP developer needs to incorporate into his BAPI, the appropriate algorithm that will call the customers implementation of the BADI. The following code excerpt demonstrates this :

```
data exit_ref type ref to ZIF_EX_BADI_00.  
...  
call method cl_exithandler=>get_instance  
    changing instance = exit_ref.  
...  
call method exit_ref->export_data  
    exporting country = country  
    importing extension_out = extension_out.  
...
```

Preparing the BAPI signature

BAPIs that have enhancement capabilities implemented into them will typically also have a special signature. This specially adapted signature will include 2 parameters called *extension_in* and *extension_out*. These parameters could either be flat structures or internal tables.

- **Note:** *extension_in* will be used by the SAP developer to receive extra user data in the BAPI
- **Note:** *extension_out* will be used by the SAP developer to return back to the user, extra data that the customer will typically fill via his BADI implemented method.

The basic data type typically used for these parameters is a dictionary structure called *BAPIPAREX*.



Structure	BAPIPAREX					
Short description	Ref. structure for BAPI parameters ExtensionIn/ExtensionOut					
<hr/>						
Component	Component type	DType	Length	DecPl	Short description	
STRUCTURE	TE_STRUC	CHAR	30	0	Structure name of BAPI table ext	
VALUEPART1	VALUEPART	CHAR	240	0	Data part for BAPI extension	
VALUEPART2	VALUEPART	CHAR	240	0	Data part for BAPI extension	
VALUEPART3	VALUEPART	CHAR	240	0	Data part for BAPI extension	
VALUEPART4	VALUEPART	CHAR	240	0	Data part for BAPI extension	

Figure 167: The BAPIPAREX structure

The individual fields have the following purpose:

- ***Structure***

The *Structure* field contains the name of the BAPI table extension to which the respective data record refers. In turn, because a table extension is allocated to exactly one database table, the program can determine the extended table in which to save the data record (name of A TE structure in a BAPI where the BAPI needs to update extra customer ZZ data fields).

- ***VALUEPART1 through VALUEPART4***

Each data record of the extension container contains—in addition to the name of the table extension—the key values and the values that will be inserted in the additional table fields. The key values have to be passed on in order to determine the line in the database table where the data in the data record will be written. Therefore, the *VALUEPART1* through *VALUEPART4* fields contain both the key value(s) that identify the table line and the data fields to be inserted into the table. The assignment of key values and data to *VALUEPART1* through *VALUEPART4* is performed by consecutively filling the *VALUEPART* fields. *VALUEPART1* first saves all the key fields of a data record. If *VALUEPART1* still has capacity, then the value of the first customer-specific table field is also saved in *VALUEPART1*. This is continued with the further table fields until the capacity of *VALUEPART1* is exhausted. The process is continued with *VALUEPART2* through *VALUEPART4*, until all the values in the data record have been included in the container.

 **Note:** In situations where the remaining capacity of *VALUEPART1* (for example) is less than the maximum length of the next table field to add, then the value of this table field is split between *VALUEPART1* and *VALUEPART2*. As much of the value as fits is saved in *VALUEPART1*, and the rest is allocated to *VALUEPART2*.

Exercise 14: Prepare BAPI for enhancement

Exercise Objectives

After completing this exercise, you will be able to:

- Prepare a BAPI so that it may be enhanced using BADI enhancement technology.

Business Example

As a BAPI developer, you need to adapt your BAPI so that other developers can enhance it. For example purposes, you will copy your GetList BAPI to a GetList2. The GetList BAPI will remain as it was (ie. fixed non-enhancable). The GetList2 BAPI will allow customer extensions to it. A customer will be able to have this BAPI return extra data to the user, data that by default the BAPI does not return.

Task 1:

Copy your Z_BAPI_CONTACT_GETLIST_## BAPI function module to Z_BAPI_CONTACT_GETLIST2_##.

1. Copy the function module

Task 2:

Create a table type for the BAPIPAREX structure. This table type will be used by the BAPI in order to store the extra data the user will want to have the BAPI return. It will be the type used by the *extension_out* parameter.

1. Create the required dictionary table type.

Task 3:

Create a Business Add-in(BAdi) for your enhancement.

1. Create the **BAdi definition ZBADI_##**.

Task 4:

Adapt the BAPI function module so that it may call the customer BADI implementations.

1. Change the BAPI function module parameter passing mechanism.
2. Change the BAPI source code in order to call customer implemented BADIs.

Solution 14: Prepare BAPI for enhancement

Task 1:

Copy your Z_BAPI_CONTACT_GETLIST_## BAPI function module to Z_BAPI_CONTACT_GETLIST2_##.

1. Copy the function module
 - a) Go to transaction SE37
 - b) In the Function Module name field enter your original Function Module name Z_BAPI_CONTACT_GETLIST_##.
 - c) Hit the 'Copy' button.
 - d) In the 'Fr. Function Module' field enter your original function module name (ie. Z_BAPI_CONTACT_GETLIST_##)
 - e) In the 'To Function Module' field enter your new function module name (ie Z_BAPI_CONTACT_GETLIST2_##).
 - f) In the 'Function Group' field enter your ZBC417_## function group name.
 - g) Hit the 'Copy' button.
 - h) Activate your function module using the 'Activate' button.

Task 2:

Create a table type for the BAPIPAREX structure. This table type will be used by the BAPI in order to store the extra data the user will want to have the BAPI return. It will be the type used by the *extension_out* parameter.

1. Create the required dictionary table type.
 - a) In the Data Dictionary (SE11), select the *Data type* radio button.
 - b) Enter the name of your table type **ZBAPIPAREX_BC417_##**
 - c) Select *Create* and then select *Table Type*.
 - d) Enter an appropriate description in the *Short text* field.
 - e) Pick the *Line Type* radio button and put in the name of the structure *BAPIPAREX*.
 - f) Save your table type in your package and activate.

Continued on next page

Task 3:

Create a Business Add-in(BAdi) for your enhancement.

1. Create the **BAdi definition ZBADI_##**.
 - a) Go to transaction SE18
 - b) In the menu, pick *Utilities → Create Classic BAdi*.
 - c) Enter the name of your BAdi as **ZBADI_##** (where ## is your group number) and select the *Continue* button.
 - d) In the *Definition Short text* field, enter 'Exit for Z_BAPI_CONTACT_GETLIST2_##'
 - e) Select the *Interface* tab. You see that the system has already proposed the name for the *Interface*. Select *Save* to save this information (in your Package).
 **Note:** Write down this name because you will need it in the DATA declaration of your function module.
 - f) Double click on the *Interface name* to get to the Class Builder screen.
 - g) Under the *Methods* column, type in the name **Export_Data**. Under the *Level* column, select *Instance*. Under the *Description* column, enter some realistic description of your Method.
 - h) Select *Parameters* button and enter the following parameters:
Country; **Type = Importing**, and type it to the same *Land1* field as in your Getlist function module. Select *Enter* to fill in the description.
Extension_out; **Type = Exporting**, and type it to the ZBAPIPAREX_BC417_## table type you created earlier. Select *Enter* to fill in the description.
 - i) Save and Activate your BAdi.
 **Note:** If you receive a popup saying there are inconsistencies between your BADI and the Interface you are designing, hit the 'Change Add-In' button.

Continued on next page

Task 4:

Adapt the BAPI function module so that it may call the customer BADI implementations.

1. Change the BAPI function module parameter passing mechanism.
 - a) Go to transaction SE37
 - b) Put in the name of your BAPI function module (ie Z_BAPI_CONTACT_GETLIST_##).
 - c) Hit the 'Change' button.
 - d) Go to the 'Export' tab strip.
 - e) Add a parameter named *Extension_Out*. Type the parameter based on dictionary table type ZBAPIPAREX_BC417##. Mark the parameter as Passed by value.
2. Change the BAPI source code in order to call customer implemented BADIs.
 - a) Add the the following code at the beginning of the function module :

```
data exit_ref type ref to ZIF_EX_BADI_##.
call method cl_exithandler=>get_instance
      changing instance = exit_ref.
```
 - b) At the end of the function module, add the following :

```
call method exit_ref->export_data
      exporting country = country
      importing extension_out = extension_out.
```



Lesson Summary

You should now be able to:

- Identify if an SAP BAPI has Classic BADI customer extension capabilities.

Related Information

- For more details refer to the BAPI programmers guide in the SAP help portal.

Lesson: Implementing a BADI of a BAPI

Lesson Overview

In this lesson, we will see how a customer implements a BADI implementation of a BAPI. This BADI needed to be previously defined and preplanned within the BAPI by the SAP developer.



Lesson Objectives

After completing this lesson, you will be able to:

- Implement customer specific processing in an SAP BAPI using a predefined BADI.

Business Example

You've identified an SAP BAPI that you would like to use in your custom applications. However, the BAPI does not do exactly what you'd like it to do. The SAP developer of the BAPI has preplanned in his BAPI, a BADI. You would like to implement to extend the SAP BAPI logic with your own extra business logic.

Identifying if BAPI has BADI implementation capabilities

The SAP developer should have documented, as part of the BAPI documentation, if the BAPI supports or not any enhancement capabilities.

SAP developers that pre-plan in their codes, the use of BADIs, need to call a special service method within the that will initialize the BADI environment for their application.

The method, which is called statically, resides in class CL_EXITHANDLER and is called GET_INSTANCE.



```

Function module Z_BAPI_CONTACT_GETLIST2_00 Active
Attributes | Import | Export | Changing | Tables | Exceptions | Source code

FUNCTION Z_BAPI_CONTACT_GETLIST2_00.
  *"- Local Interface:
  *"  IMPORTING
  *"    VALUE(COUNTRY) TYPE BAPI0417_1-LAND1
  *"  EXPORTING
  *"    VALUE(RETURN) TYPE BAPIRET2
  *"    VALUE(EXTENSION_OUT) TYPE BAPIPAREX_BC417_TAB
  *"  TABLES
  *"    CUSTOMER_LIST STRUCTURE BAPI0417_1
  *"- 

  data exit_ref type ref to zif_ex_badi_00.
  ...

  CALL METHOD CL_EXITHANDLER=>GET_INSTANCE
    CHANGING
      INSTANCE
      = exit_ref.
  ...

ENDFUNCTION.

```

Figure 168: Identifying if BAPI has BADI

Identifying the BADI definition name

The SAP developer created a BADI definition for his BAPI within SE18. Before implementing this BADI, you must identify the name of the BADI he has created.

A common technique in order to derive the name of the BADI is by looking at the interface name that is used. The name of the BADI is generally at the end of the interface name (if_ex_<name_of_badi>). If the interface name starts with a Y or a Z, the name of the BADI definition will be Y or Z<name_of_badi>.

→ **Note:** Generally, SAP developers do not create components that start with a Y or a Z. Those characters are reserved for the customer namespace.

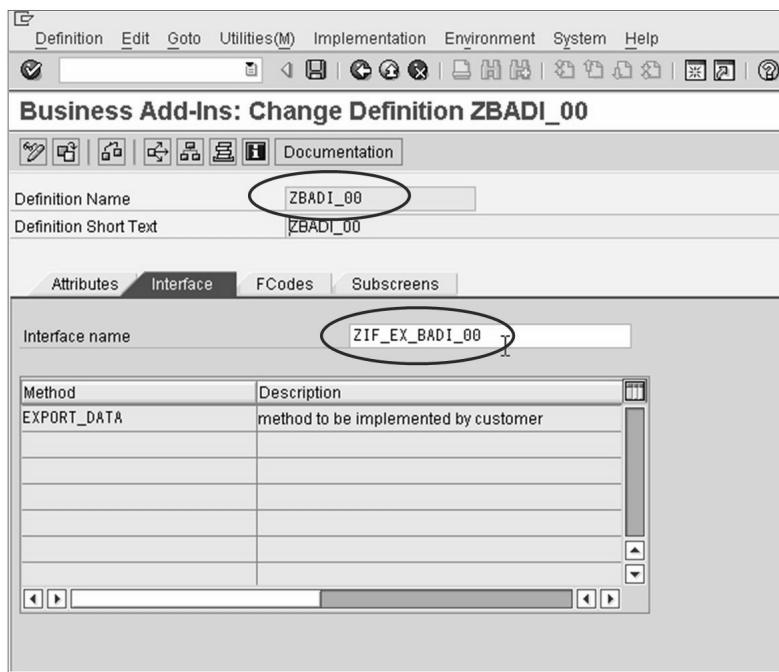


Figure 169: Interface associated to BADI

Implementing the BADI

Once you have identified the BADI that the developer of the BAPI has created, you may go to SE19 and implement the BADI.

Within SE19, you will actually create an SE24 ABAP object class that will implement the interface that the SAP developer created whilst creating his BADI. You then insert your desired code within the interface methods that the SAP developer has declared in his interface.

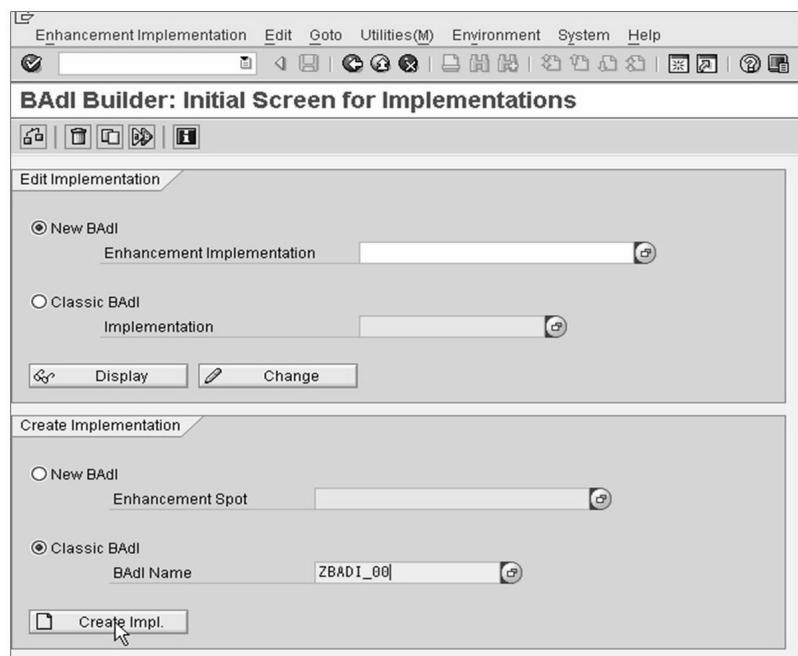


Figure 170: Creating a BADI implementation (1)

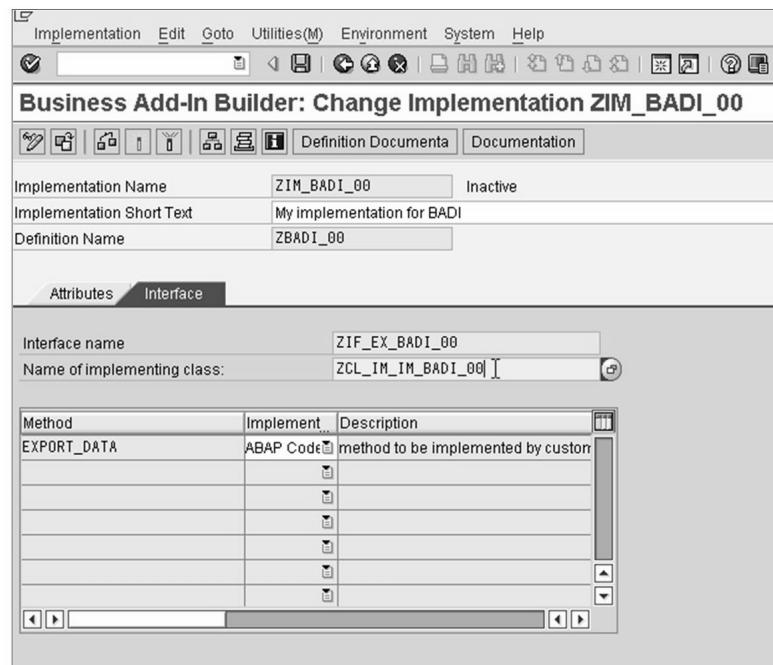


Figure 171: Creating a BADI implementation (2)

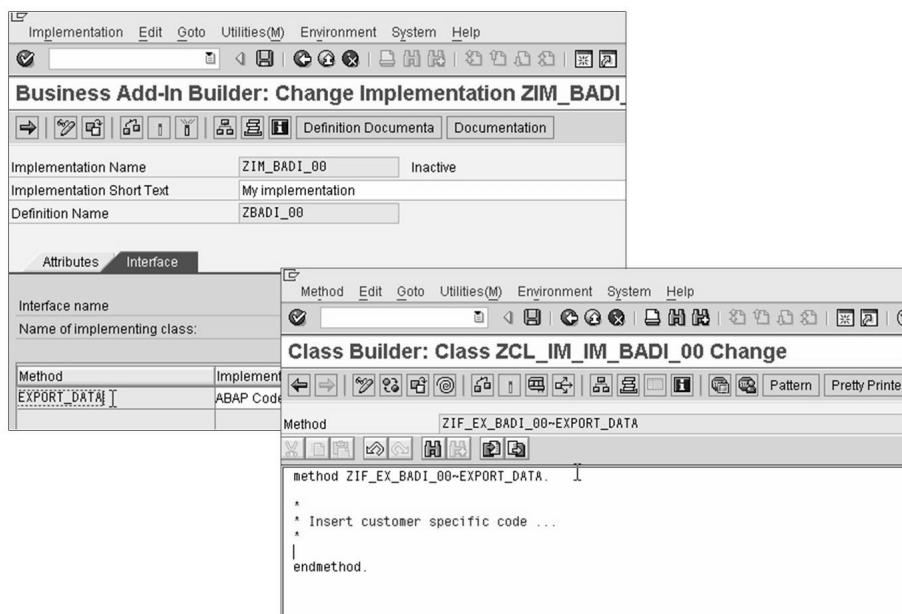


Figure 172: Creating a BADI implementation (3)

Activating the BADI

Once your implementing class has been created and activated, you'll need to activate your BADI implementation from SE19. Activating a BADI will then make your implemented class methods executed any time the BAPI is ran. If errors occur in the BAPI caused by your BADI implementation, you can always deactivate your BADI implementation. Once deactivated, your class methods of your implementation will no longer be called by the BAPI.

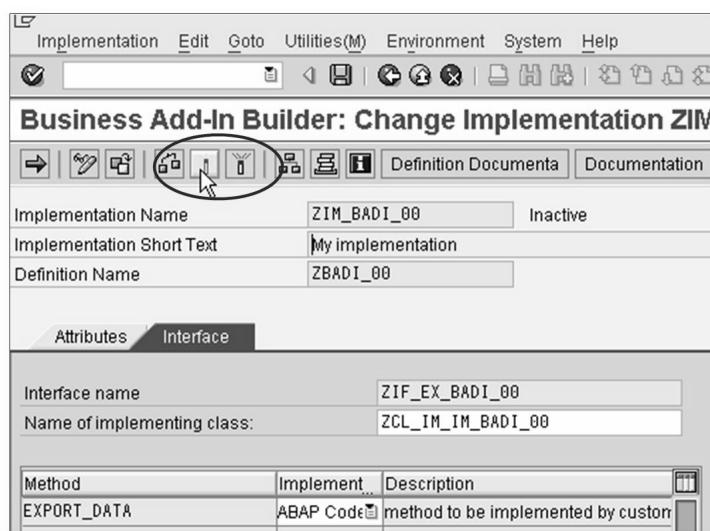


Figure 173: Activating the BADI implementation

Testing the BADI implementation

To test your BADI implementation either write an ABAP program that calls the BAPI or, if possible, test the BAPI function module via SE37.

Exercise 15: Implement BADI for BAPI

Exercise Objectives

After completing this exercise, you will be able to:

- Implement the BADI which will enhance the functionality of the BAPI.

Business Example

You've identified the Contact GetList2 BAPI developed by an SAP developer. You've also identified that this BAPI allows customer enhancements using BADI technology. You need to extend the BAPI so that it now returns more data than the default exported data as developed by SAP.

Task:

Implement the Business Add-in(BAdi) for your enhancement.

1. Create an implementation ZIM_BADI_## for the BAdi defined by the SAP developer.
2. Test the function module via SE37 and see if the BAPI now returns the extra data via the *extension_out* parameter.

Solution 15: Implement BADI for BAPI

Task:

Implement the Business Add-in(BAdi) for your enhancement.

1. Create an implementation ZIM_BADI_## for the BAdi defined by the SAP developer.
 - a) Go to transaction SE19.
 - b) In the 'Create Implementation' section of the screen, pick the radio button 'Classic BAdI'. In the 'BAdI Name' field enter 'ZBADI_##'. Hit the 'Create Implementation' button.



Note: It is recommended that implementation names begin with the letters IM. This would be preceded by a Z or Y for customer definitions.

- c) In the 'Implementation name' field enter 'ZIM_BADI_##' followed by the 'Continue' button.
- d) In the *Implementation Short text* field, type in a meaningful description for your implementation. For example, `Implement customer code for Z_BAPI_Contact_Getlist2_##.`
- e) Select the *Interface* tab and you will see the method that you will need to implement. *Save* this information.
- f) Double click on the method name in order to start implementing it via the Class Builder. This is where you type in the code that you want the SAP BAPI to now call. This is a sample of the code you will use although it may need to be modified to fit your own object names:


```

data: Hold_area(960),
      comma value ','.
data wa_extension_out like line of extension_out.
types: begin of mytype,
      kunnr type skna1-kunnr,
      telf1 type skna1-telf1,
      telfx type skna1-telfx,
      mcod3 type skna1-mcod3,
end of mytype.
data it_extra_data type standard table of mytype.
data wa_extra_data like line of it_extra_data.
clear extension_out.
select kunnr telf1 telfx mcod3
from skna1
      into corresponding fields of table it_extra_data
where land1 = country.
```

Continued on next page

```
loop at it_extra_data into wa_extra_data.  
concatenate wa_extra_data-kunnr  
          wa_extra_data-telf1  
          wa_extra_data-telfx  
          wa_extra_data-mcod3  
           into hold_area separated by comma.  
move hold_area to wa_extension_out+30.  
Append wa_extension_out to extension_out.  
endloop.
```

- g) Save and Activate your method. Make sure all class components are selected during the activation.
h) Back out of the class builder in order to return to the BADI implementation transaction. On this screen, you will see icons that look like the Activation icon. One is to activate the implementation, the other is to deactivate the implementation. Hit the 'Activate' icon in order to have your BADI implementation active.
2. Test the function module via SE37 and see if the BAPI now returns the extra data via the *extension_out* parameter.
a) no solution



Lesson Summary

You should now be able to:

- Implement customer specific processing in an SAP BAPI using a predefined BADI.

Related Information

- For more information on BADI technology in general, refer to the SAP Help portals documentation on Enhancements and BADIs.]

Lesson: Redefining a method in the BOR

Lesson Overview

This lesson discusses the steps involved in redefining a method in the Business Object Repository..



Lesson Objectives

After completing this lesson, you will be able to:

- Create a subtype of an SAP Business Object and redefine one of the methods within the subtype.

Business Example

Within the BOR, you'd like to redefine the business logic of a method. Since the BOR is based on object oriented concepts, you can create a subtype to a business object and redefine one of the methods in the BOR in order to implement your own business logic.

Creating Subtypes

Subtypes in the BOR are comparable to subclasses in pure object oriented environments. One of the advantages of developing in object oriented is the notion of inheritance allowing you to easily create subclasses to a superclass, and adapting the subclass components to your specific needs. In the example we will be looking at, we will create a subtype to a Business Object. In the subtype, we will redefine the business logic of one of the methods inherited from the supertype.

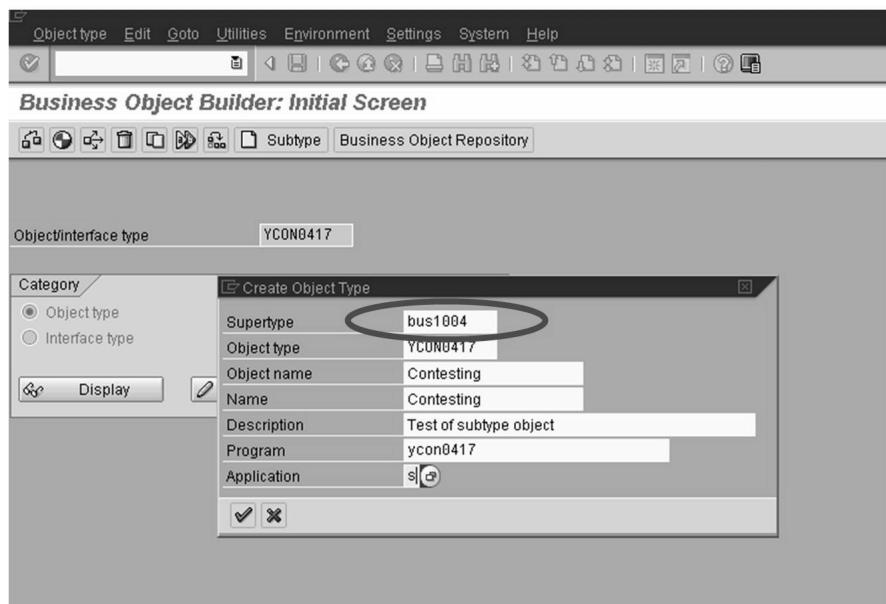


Figure 174: Create a subtype (1)

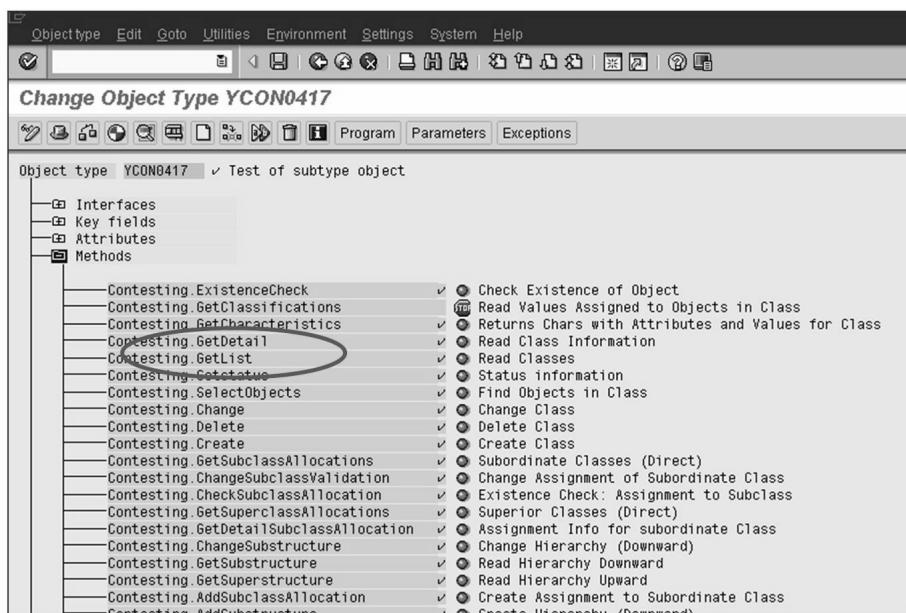


Figure 175: Create a subtype(2)

To create a subtype:

1. From the SAP menu, choose *Tools → Business Framework → BAPI Development → Business Object Builder*, or enter the transaction code SWO1.
2. In the *Object type* field, enter the name of the subtype you want to create and choose *Create*.
3. In the next dialog box, enter the required details:
 - In the *Supertype* field, enter the name of the object type for which you want to create a subtype.
 - In the *Object type* field, enter the name of the subtype you want to create.
 - Enter appropriate values in the remaining fields.

Redefining a BAPI

Redefining a method in the BOR will allow you to implement your own algorithm for the method in question. Although in this example we are redefining a BOR method which implements a BAPI call, you can redefine non-BAPI methods as well.

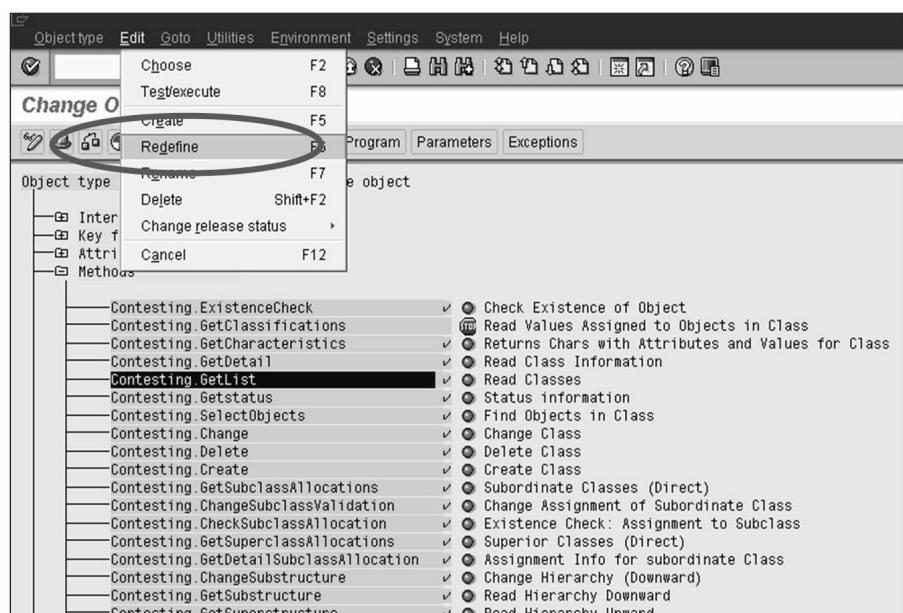


Figure 176: Redefine a BAPI (1)

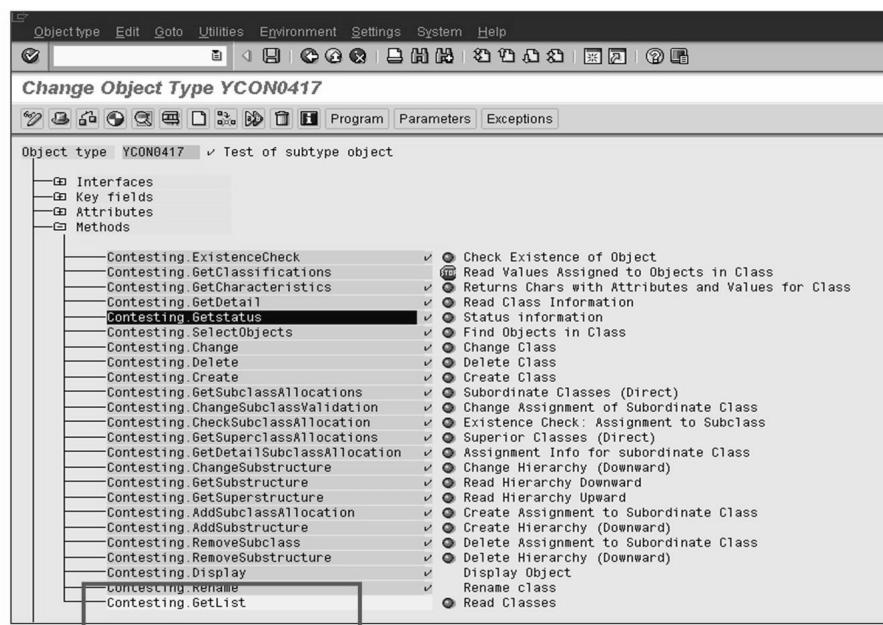


Figure 177: Redefine a BAPI (2)

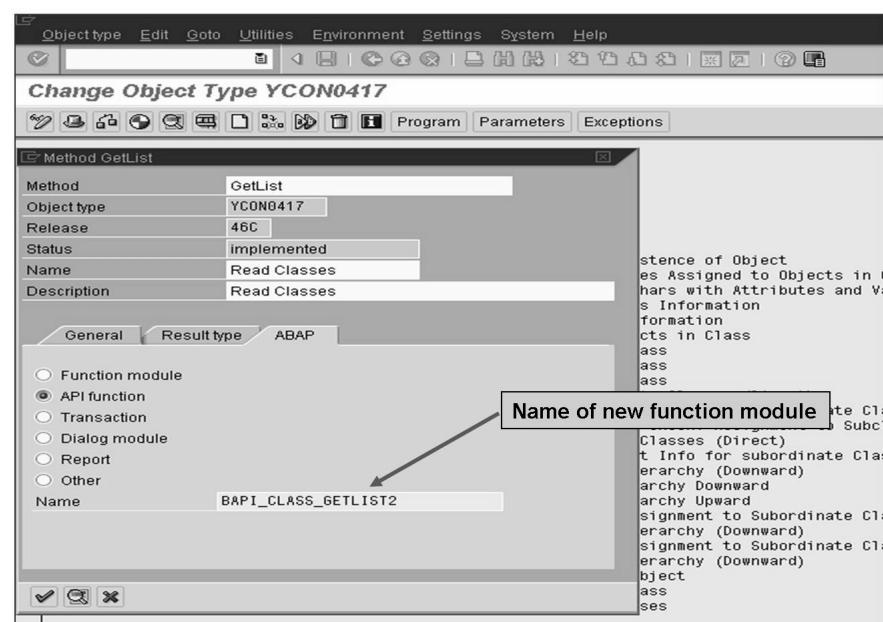


Figure 178: Redefine a BAPI (3)

To redefine a BAPI of a subtype, perform the following steps:

1. Choose *Tools → Business Framework → BAPI Development → Business Object Builder*, or enter transaction code SWO1.
2. Display the subtype just created in the change mode.
3. Position the cursor on the BAPI you want to modify and choose *Edit → Redefine*.
4. Select the BAPI and then the *ABAP register*.
5. In the *Name* field of the modified function module, *Enter* and *Save* your information.

Defining Delegation Relationships

The principle of **Delegation** in the BOR is somewhat specific to the BOR and not commonly found in pure object oriented environments. It is frequently used in WebFlow development.

Once you have created a subtype to a Business Object, you might want to have that subtype used instead of the original supertype. Delegation will allow you to accomplish this. After delegating a supertype to a subtype, all references to the supertype object will be passed over to the subtype. Example : SAP delivers a BO called EQUI. You create a subtype to EQUI called ZEQUI. You then delegate EQUI to ZEQUI. Without knowing it, all WebFlow developers who think they are calling methods from the EQUI business object are actually calling methods of ZEQUI. Note that in the subtype, you might have redefined the logic of some of these methods.

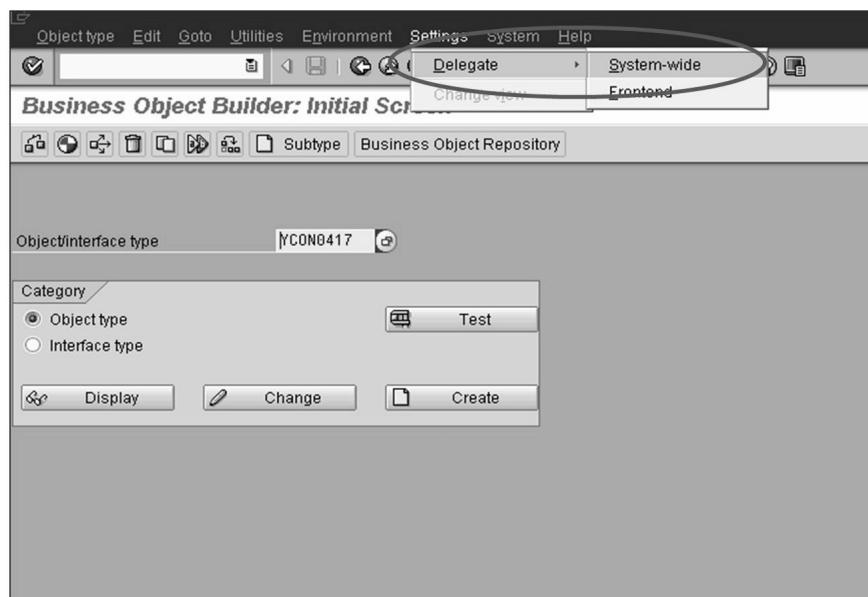


Figure 179: Delegate a relationship (1)

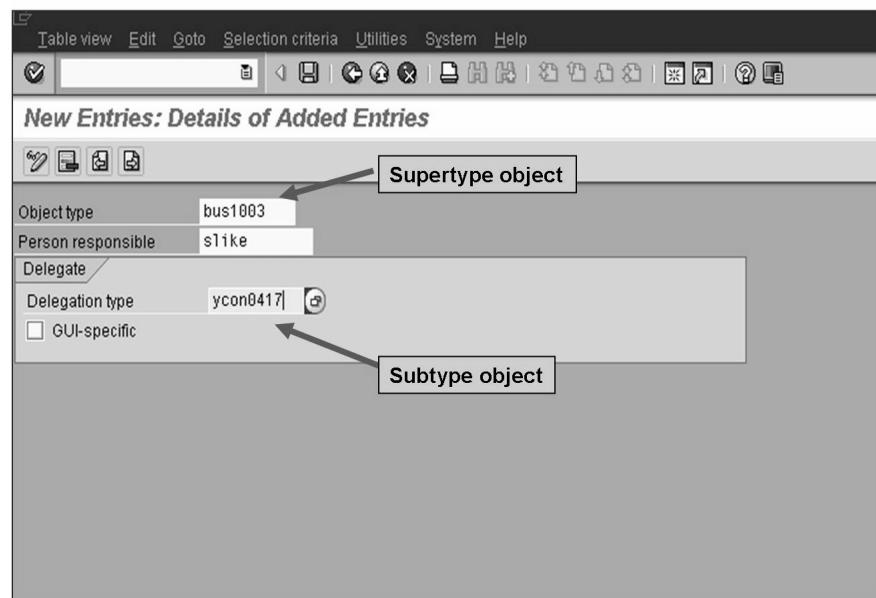


Figure 180: Delegate a relationship (2)



Obj.type	Descript.
BUS1003	Class
BUS2002	Network
BUS2005	Production order
BUS2032	Sales Order
KNA1	Customer
LIKP	Outbound delivery
PD0TYPE_SH	HR Position
PD_1001	PD Infotype Relationship (1001)
Z00MARA	BC610: Material

Figure 181: Delegate a relationship (3)

To define the delegation relationship between the supertype and subtype:

1. From the SAP menu, choose *Tools* → *Business Framework* → *BAPI Development* → *Business Object Builder*, or enter the transaction code SWO1.
2. Choose *Settings* → *Delegate* → *System-wide*.
3. Switch to the change mode and select *New entries*.
4. Enter the name of the original object type (supertype) in the *Object type* field and the name of the subtype in the *Delegation type* field.
5. Deactivate the *GUI-specific* checkbox.
6. Save your entries.



Lesson Summary

You should now be able to:

- Create a subtype of an SAP Business Object and redefine one of the methods within the subtype.



Unit Summary

You should now be able to:

- Identify if an SAP BAPI has Classic BADI customer extension capabilities.
- Implement customer specific processing in an SAP BAPI using a predefined BADI.
- Create a subtype of an SAP Business Object and redefine one of the methods within the subtype.

Internal Use SAP Partner Only

International Use SAP Partner Only

Unit 6

BAPIs Using ALE & IDOCs

Unit Overview

This Unit provides an overview of the ALE technology with particular attention to BAPI usage.

The Unit also discusses and demonstrates the use of the ALE Distribution Model.



Unit Objectives

After completing this unit, you will be able to:

- Describe how to use BAPIs in distribution systems (ALE)

Unit Contents

Lesson: ALE and IDOCs ALE Basic Concepts	318
Exercise 16: Generate an ALE Interface	331

Lesson: ALE and IDOCs ALE Basic Concepts

Lesson Overview

This lesson provides a brief overview of the ALE concept.



Lesson Objectives

After completing this lesson, you will be able to:

- Describe how to use BAPIs in distribution systems (ALE)

Business Example

A company uses distributed business processes to pass data between country-wide installations.

Distributed Systems

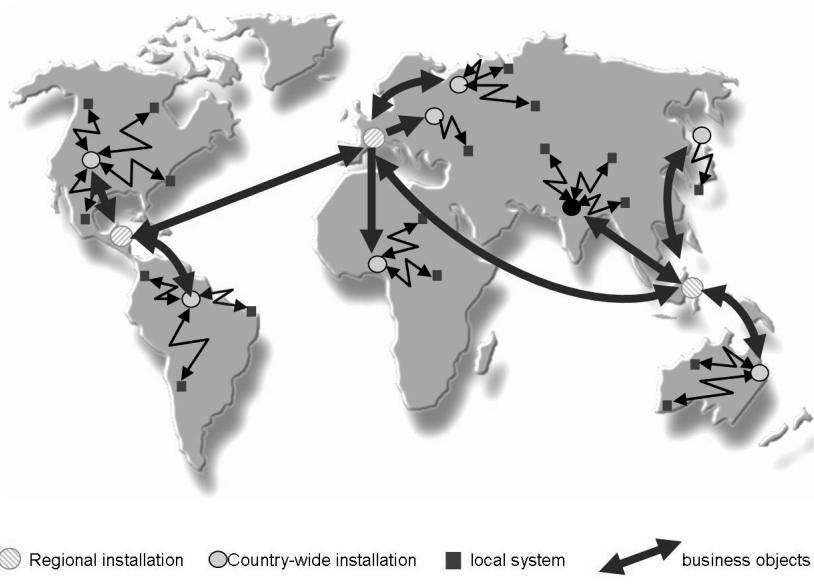


Figure 182: Distributed Business Processes

Basic Concepts of ALE Technology

ALE supports the configuration and operation of distributed applications. It allows the controlled exchange of business messages between distributed applications with consistent data retention. The coupling between the distributed systems can be either narrow or loose. Both types of coupling can be implemented by calling BAPIs.

The application integration is not achieved through a centralized database. Instead, the applications access a local database. Data retention is redundant. ALE ensures the distribution and synchronization of master, control, and transaction data through asynchronous communications. ALE uses synchronous connections to read data.

The use of ALE offers a number of advantages:

- Distribution of application data between SAP Systems with different releases
- Data exchange can continue after an upgrade without any further adjustment
- Customer-specific enhancements
- The integration of non-SAP systems through communication interfaces
- Coupling R/3 and R/2 systems

Implementing a BAPI

A BAPI method should only be implemented as an asynchronous interface if at least one of the following conditions is satisfied:

- Database changes are consistent in all systems. The data is updated consistently in both the local and remote systems.
- When looser coupling is required. With synchronous interface, the coupling between the client and the server system is too narrow. If the connection is interrupted, the client system would not be able to function properly.
- Performance load considerations. The interface is frequently used or the interface handles large volumes of data. In this situation, a synchronous interface cannot be used because performance would be too low.

If you implement a BAPI as an asynchronous interface, keep the following information in mind in addition to following the standard programming BAPI guidelines:

- The BAPI must not issue a COMMIT WORK command.
- The BAPI return parameter must use reference structure BAPIRET2.
- All BAPI export parameters with the exception of the return parameter are ignored and are not included in the IDoc that is generated.
- After the function module, which converts the IDoc into the corresponding BAPI call in the receiving system has been called, status records are written for the IDoc in which messages sent in the return parameter are logged.

If, in at least one of the entries of return parameter, the Type field in the return parameter is filled with A (abort) or E (error), this means:

Type A

Status 51 (error, application document not posted) is written for all status records, after a ROLLBACK WORK has been executed.

Type E

Status 51 (error, application document not posted) is written for all status records and a COMMIT WORK is executed. Otherwise, status 53 (application document posted) is written and a COMMIT WORK executed.

Defining Hierarchies Between BAPI Parameters

When hierarchical dependencies exist between the table parameters of a BAPI, these hierarchies must be explicitly defined to allow parameter filtering.

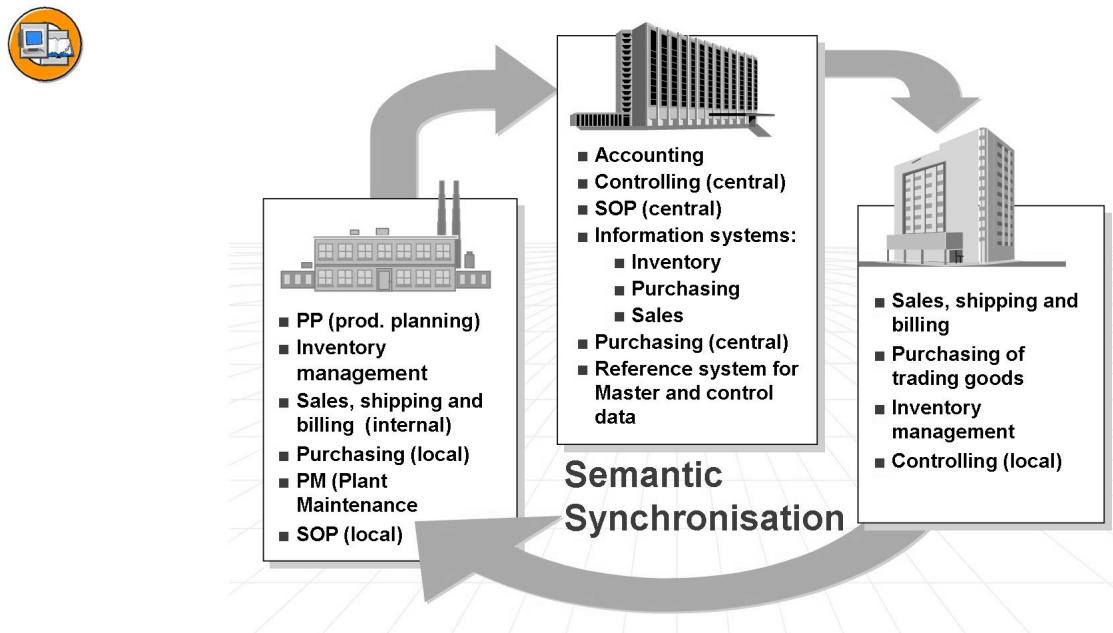


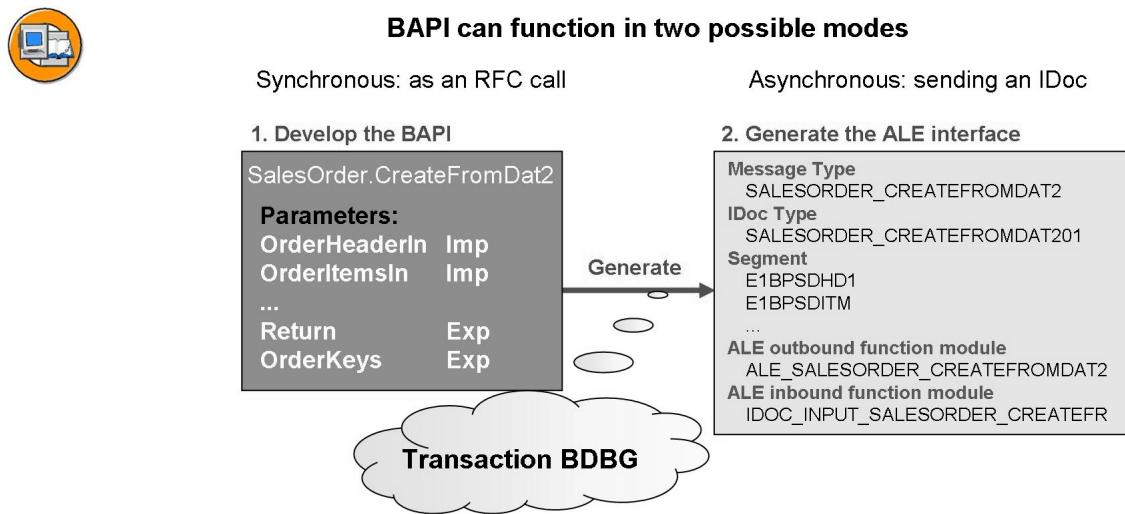
Figure 183: ALE Integration Technology

Example:

A BAPI for material master data contains, for example, the tables for plant data and the corresponding warehouse data. The table for plant data refers to the table for warehouse data through the *WERKS* key field. A hierarchical dependency exists between the plant data and the warehouse data: if plant 001 is not replicated for a material as a result of the parameter filtering, then the warehouse data for plant 001 should not be replicated either.

The hierarchical dependencies are defined using field references between the parameters of the BAPI tables. These references must be defined before you create the BAPI-ALE interface, as this information is generated with the interface. You can define the hierarchical dependencies using transaction **BDBP**.

Maintaining the BAPI-ALE Interface



Guideline from Release 4.0 on:
All (new) asynchronous interfaces (IDocs) are BAPI-based

Figure 184: Generate IDocs from BAPIs

Use

To allow the asynchronous BAPI call in ALE business processes, you have to create a BAPI-ALE interface for the BAPI. In the process, the following objects are generated for a BAPI:

- A message type
- An IDoc type, including its segments
- A function module that is called on the outbound side (it uses the BAPI data to generate and dispatch the IDoc)
- A function module that fills the BAPI with the IDoc data on the inbound side

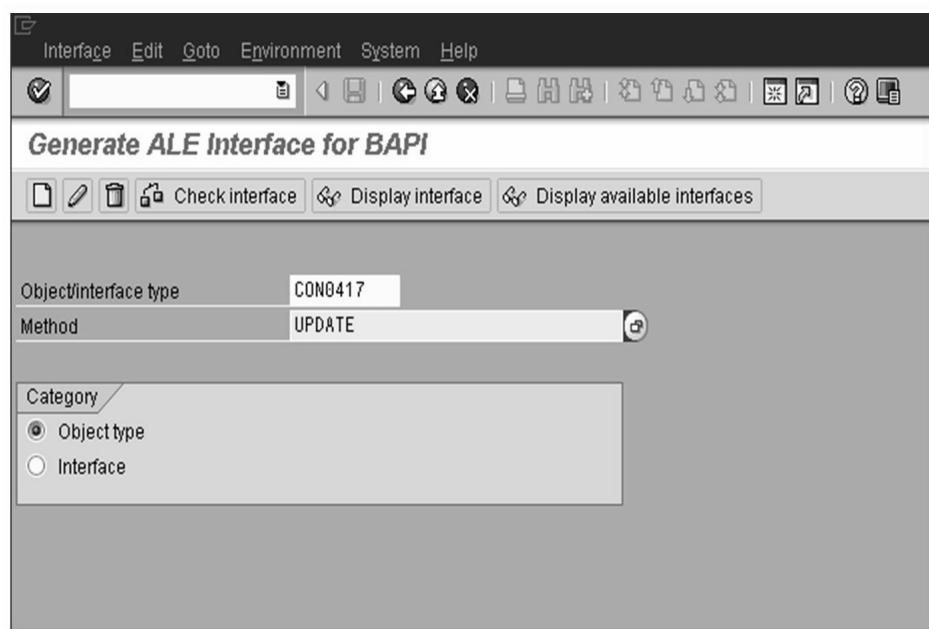


Figure 185: Generate ALE Interface (1)

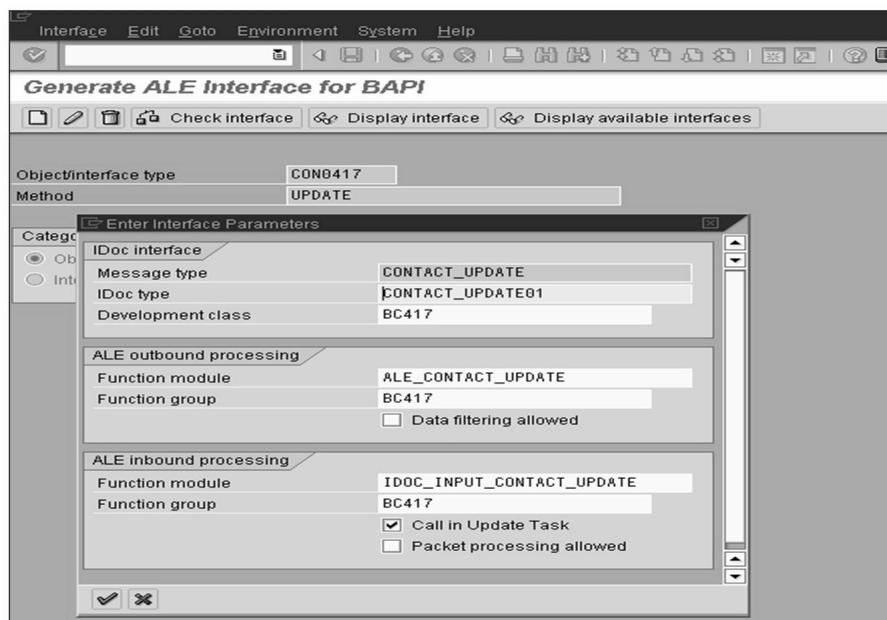


Figure 186: Generate ALE Interface (2)

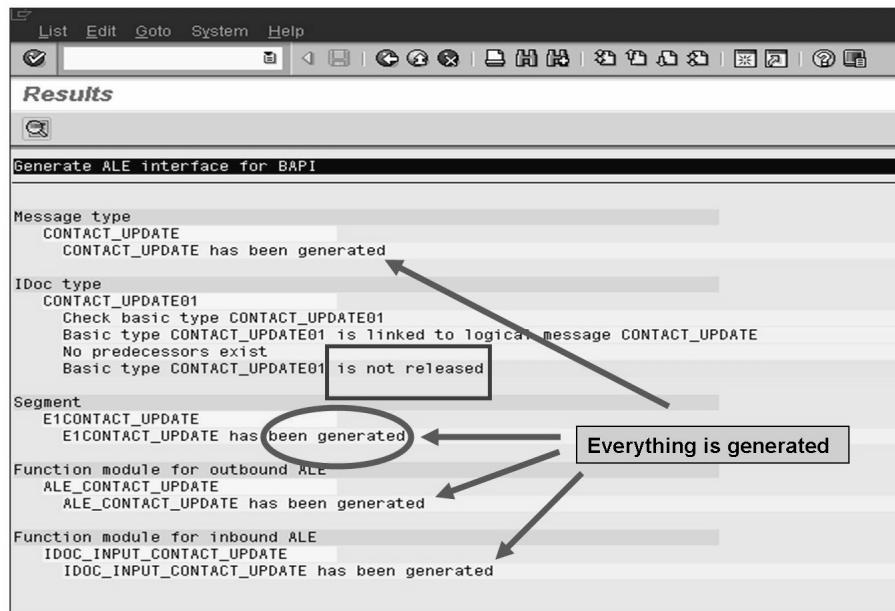


Figure 187: Generate ALE Interface (3)

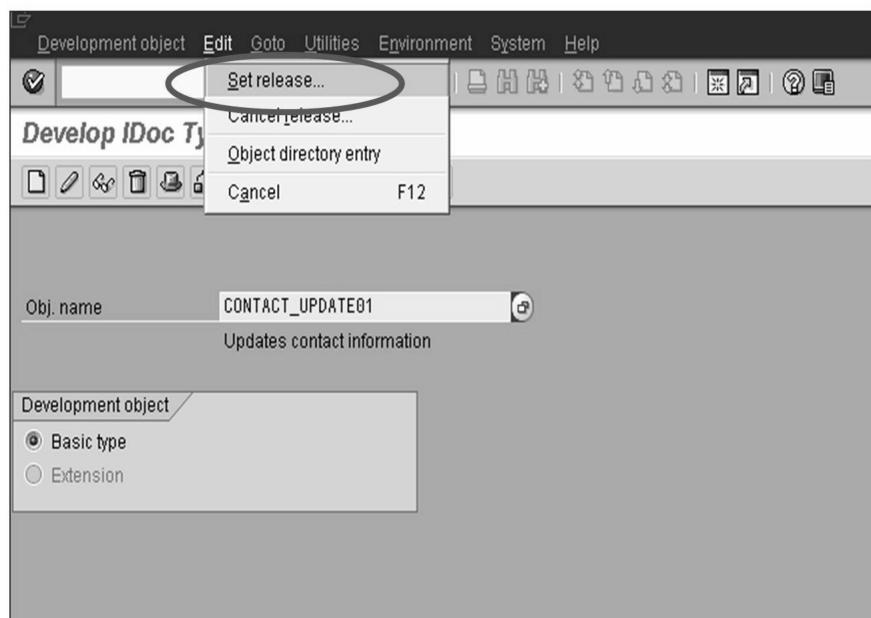


Figure 188: Release IDoc Type (1)

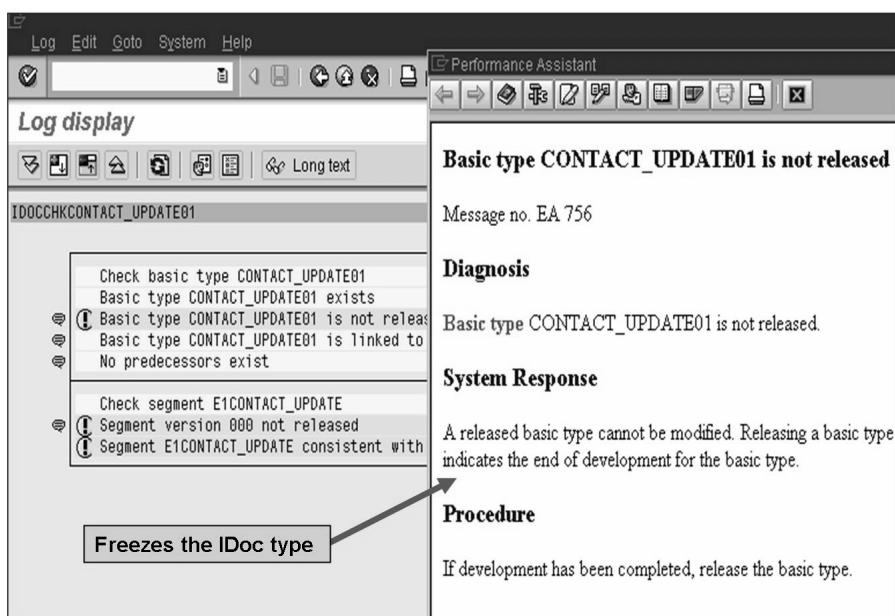


Figure 189: Release IDoc type (2)

In contrast to manually maintained message types, the function module that evaluates the change pointers does not create an IDoc; instead, it fills the appropriate BAPI structures, determines the receivers, and calls the generated ALE function module.

Prerequisites

The following **two** prerequisites must be met:

1. The BAPI must exist.
2. The following alternatives are possible:
 - A new BAPI has been developed.
 - A customer wants to generate a BAPI-ALE interface for a BAPI in the standard shipment, or change a BAPI-ALE interface supplied by SAP after modifying the corresponding BAPI. To do this, proceed as follows: Copy and modify the function module for the original BAPI. In the BOR, create a sub-object type in the customer namespace for the business object type that belongs to the BAPI (SWO1). Set the status of the object type to Implemented. All methods of the subtype can then be changed, deleted, or supplemented with your own methods. You then generate the BAPI-ALE interface for the subtype and an assigned method in the customer namespace.

After an upgrade, the customer-generated interface will continue to work like it did in the old release, even if SAP issues a new version of a BAPI-ALE interface with the new release. If there is not a new interface from SAP, the customer can regenerate the old interface to adjust to any new parameters.

If SAP issues a BAPI-ALE interface for a BAPI for which the customer has already generated an interface, then the customer should use the new interface and delete the old one. The old interface can still be used, but only in the old release.

Procedure

To maintain the ALE interface, proceed as follows:

1. In the initial screen, enter the basic business objects (object type) for your interface and the method.
2. Select one of the options in the interface menu to maintain, display, check or delete an interface:

Creating an interface

There has to be no interface previously generated for this BAPI. Creating an interface involves the following process flow:

- a) Enter a name for the message type in the first dialog box. The proposed names are selected according to the following naming conventions:
The message type should be some business term. Example: MYTEST.
- b) You can enter the following information in the remaining fields:
The IDoc type would be the message type plus an **01**. Example: MYTEST01.

The outbound function module would be ALE plus the object name plus the method name. Example: ALE_EXAMPLE_TEST.

The inbound function module would be IDOC_INPUT plus the message type. Example: IDOC_INPUT_MYTEST.

The system proposes the Package and function group to which the BAPI function module belongs. When you generate your own interfaces, use your own Package and function groups.

The following options are also available:

Data filtering allowed

if you want to perform data filtering for the BAPI, you will have to choose *Data filtering allowed*.

Call in update task

you have to set this flag if you want to perform database changes using methods of the update task.

Package processing allowed

you have to set this option if you want to allow package processing of IDocs. The corresponding BAPI must support package processing. Use ALE customizing to configure the package size.



Note: to generate only the message type (required field) and IDoc type (required field), or only one of the function modules (optional fields), simply leave the corresponding fields blank.

- c) Confirm your entries. The *Enter segment name* dialog box displays.
- d) Enter a segment name. The proposed segment type is derived from the message type or BAPI structures:
 - E1 plus message type. Example: E1MYTEST.
 - E1BP_#. Example: E1BP_HUGO for BAPI_HUGO.



Note: if you define a segment as containing more than 1000 bytes of data, child segments automatically generate. The names of the child segments are then appended to the original segment name with digits 1, 2 and so on, as long as the original segment name is shorter than 27 characters.

Changing an interface

Objects must have already been created for this BAPI. Choose *Change* to regenerate the objects of an existing ALE interface after changes to the BAPI. The IDoc type and IDoc segments are regenerated when the interface structures and object methods have changed. The function modules are only regenerated when the IDoc type or one of the segments has changed. When the dialog box is displayed, it shows the objects that already exist in the system. These fields are non input fields. If a field is blank, you can generate the corresponding object.

Displaying an interface

These objects already exist for this BAPI. All the existing objects for this BAPI are displayed. This gives you an overview of the relationship between the BAPI method and the IDoc message type.

Deleting an interface

These objects have already been created for this BAPI. Any function modules that exist in the system are deleted. The IDoc structure is deleted if it has not been released yet. The IDoc segments are only deleted if they have not been released yet and are not used in any other IDocs. Finally, the message type is deleted if it no longer has an assignment to an IDoc type.

Checking the interface

These objects have already been created for this BAPI. All objects are checked to see whether they already exist in the system. The system also checks whether objects (IDoc type and segments) have already been released.

3. You can release the interface. Developers can change the release status of IDoc types and segments. To do this, you need authorizations S_IDCDFT_AL+ and S_IDCDFT_ALL of authorization object S_IDOCDEFT.
-  **Note:** to release an object, the corresponding BAPI must already be released.

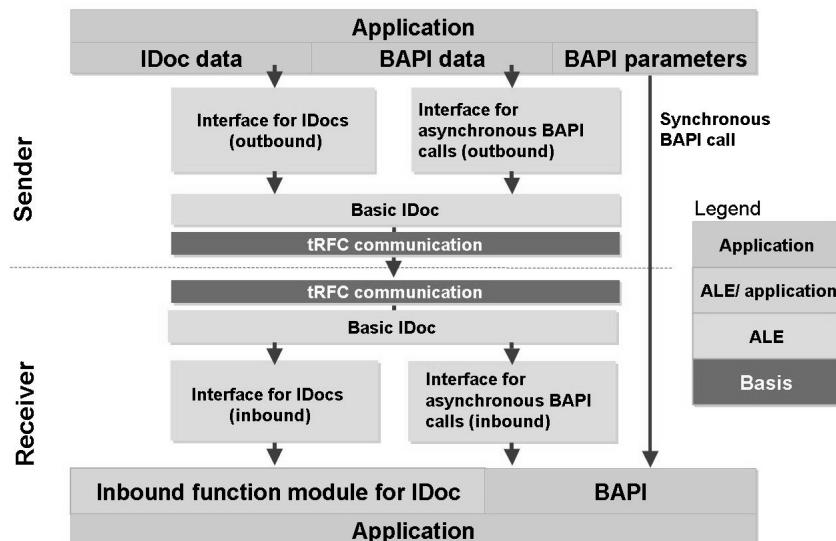


Figure 190: ALE data flow between SAP Systems

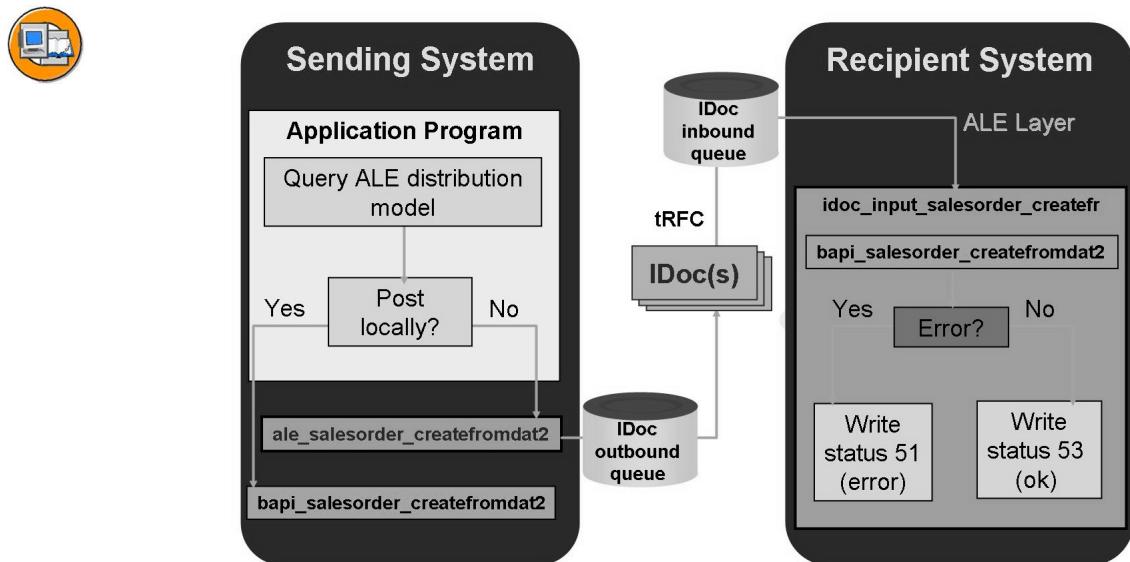


Figure 191: Sending an Asynchronous BAPI

Maintaining the Distribution Model

Use

The distribution model describes the ALE message flow between logical systems. That is, it determines which messages are distributed to which logical systems. This BAPI-based distribution of data to recipients can be restricted according to your specific requirements.

For example, you can make distribution dependent on conditions that you define as filters in the ALE distribution model.

The prerequisite for this is that a filter object type has been assigned to the corresponding BAPI in the SAP application. SAP has already defined some filter object types and assigned them to different BAPIs. You can also define your own filter object types and assign them to BAPIs.

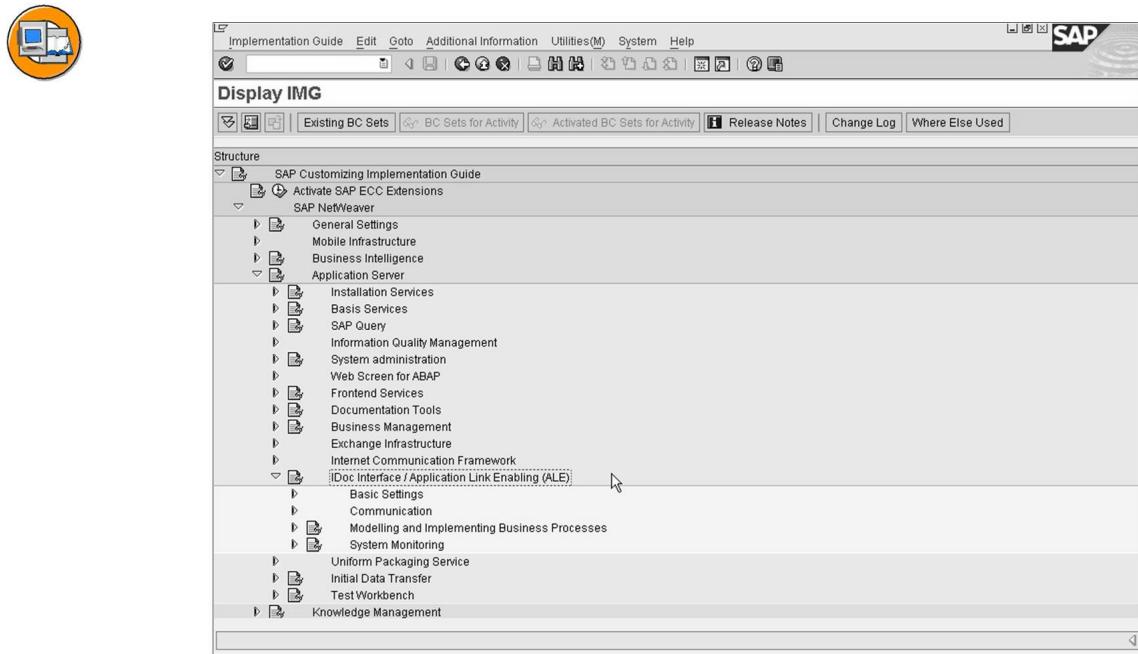


Figure 192: ALE Functions in the Implementation Guide

Receiver filtering

Before a BAPI or generated BAPI-ALE interface is called, its receivers have to be determined. To link the receiver determination with conditions, the business object methods are assigned to filter object types during receiver filtering. The values of these objects are used to determine the permitted receivers. The valid filter object values must be defined in the distribution model.

The following dependencies can be modeled in the ALE distribution model:

- Between a BAPI and a message type
- Between BAPIs

If a dependency of this type is defined as a condition in the ALE distribution model, then the receiver of the referenced BAPI or message type is determined.

Filtering Data

Two data filtering services are provided for asynchronous BAPI calls via the BAPI-ALE interface:

1. **Interface reduction**

The BAPI reduction is without conditions, that is, it involves a projection of the BAPI interface. In this process, the distribution model suppresses values for optional BAPI parameters and/or fields during data transmission. The developer of the BAPI whose interface is to be reduced must create the BAPI as reducible using appropriate parameter types. If you reduce the BAPI interface, you do not have to define any filter object types.

2. **Parameter filtering**

BAPI parameter filtering is linked with content-specific conditions: lines in table parameters of an asynchronous BAPI are determined depending on the values in the lines for the receiver. The table dataset of a BAPI is determined for parameter filtering. Hierarchical relationships can be defined between table parameters of the BAPI. To implement parameter filtering, business object methods are assigned to the business object method of filter object types. The valid filter object values must be defined in the distribution model.



Note: BAPI filtering is the term used for the shared use of both the filter services of the BAPI interface. BAPI filtering is only implemented as a service in ALE outbound processing.

Exercise 16: Generate an ALE Interface

Exercise Objectives

After completing this exercise, you will be able to:

- Create an ALE interface for a BAPI.

Business Example

For companies that use ALE to define and control their distributed systems, an interface would be necessary.

Task:

Student's will create an ALE interface definition.

1. Create an ALE interface definition using your business object name **ZCON_##** created in a previous exercise. When you receive the message type popup window, add a **Z** or **Y** in front of the proposed message type name.
2. Release the IDOC type.

Solution 16: Generate an ALE Interface

Task:

Student's will create an ALE interface definition.

1. Create an ALE interface definition using your business object name **ZCON_##** created in a previous exercise. When you receive the message type popup window, add a **Z** or **Y** in front of the proposed message type name.
 - a) Execute transaction **BDBG**.
 - b) Enter the business object name you created in a previous exercise (**ZCON_##**).
 - c) Select the F4 help button for the *Method* field to get a list of available BAPIs.
 - d) Select the update BAPI.
 - e) Select the *Create* icon.
 - f) In the *Message type* dialog box, you will get a proposed message type name. Select *Continue*. If the message type doesn't already exist, you will get another dialog box that says your message type is not in a valid name range. If you received this dialog box, you must X out of the window. **DO NOT SELECT THE CONTINUE BUTTON**. If you selected *Continue* by mistake, delete your entire interface definition and create it again. Once you have added a Z or Y to the message type name, select *New Entry*. This automatically creates your message type in the system and takes you to the next window.
 - g) Select *Continue* to generate the ALE Interface components. You will receive a list of the resulting components and the confirmation of successful generation or any possible errors.
 - h) If the IDOC type result indicates that the IDOC component is not released, you need to release it. This freezes the IDOC type against future changes, similar to a BAPI.
2. Release the IDOC type.
 - a) On the Results screen, select the IDOC type name. This takes you to the Develop IDOC Type(s) Initial screen.
 - b) On the menu, select *Edit → Set Release*.
 - c) You will receive a dialog box indicating that once the IDOC has been released, not further changes can be made. If there are no further changes to be made to the IDOC type, select *Yes*. If not, select *No* and wait until after all changes are made to release.



Lesson Summary

You should now be able to:

- Describe how to use BAPIs in distribution systems (ALE)



Unit Summary

You should now be able to:

- Describe how to use BAPIs in distribution systems (ALE)

Related Information

- For additional information on ALE technology and the Distribution Model, refer to the Using ALE Services Ecc6.0 document on SAPNet.



Test Your Knowledge

1. Some of the advantages of using ALE are; distribution of application data between SAP Systems with different releases, data exchange can continue after an upgrade without any further adjustment, and the integration of non-SAP systems through communication systems.

Determine whether this statement is true or false.

- True
- False

2. Hierarchical dependencies are defined using field references between the parameters of the BAPI tables.

Determine whether this statement is true or false.

- True
- False

3. The Distribution Model describes the ALE message flow between logical systems.

Determine whether this statement is true or false.

- True
- False



Answers

1. Some of the advantages of using ALE are; distribution of application data between SAP Systems with different releases, data exchange can continue after an upgrade without any further adjustment, and the integration of non-SAP systems through communication systems.

Answer: True

These advantages are all true for using ALE.

2. Hierarchical dependencies are defined using field references between the parameters of the BAPI tables.

Answer: True

When defining dependencies between BAPI table parameters, field references are used.

3. The Distribution Model describes the ALE message flow between logical systems.

Answer: True

The task of the Distribution Model is to describe the ALE message flow between the logical systems.

Internal Use SAP Partner Only

Internal Use SAP Partner Only

Unit 7

Mass Data Transfer

Unit Overview

This Unit explains the use of BAPIs in regard to mass data transfer and how to locate the special BAPIs created for this purpose.



Unit Objectives

After completing this unit, you will be able to:

- Explain the concept of Mass Data Transfer

Unit Contents

Lesson: Mass Data Transfer for Special BAPIs	338
Exercise 17: Mass Data Transfer	347

Lesson: Mass Data Transfer for Special BAPIs

Lesson Overview

This lesson describes how BAPIs are used for Mass Data Transfer.



Lesson Objectives

After completing this lesson, you will be able to:

- Explain the concept of Mass Data Transfer

Business Example

A customer has multiple data records in a table on their legacy system that need to be uploaded into their SAP System.

Introduction

When a customer implements a new SAP System, an important part of the migration process is copying objects from the legacy system. To transfer this mass data, each business object type in the SAP System currently has its own program for performing this initial data load with batch input, direct input, or other SAP developments (such as IS-B).

SAP developed the Data Transfer (DX) Workbench in Release 3.1 to give users a uniform startup screen for loading the different business object types, as well as simplified handling of standard files.

Starting in Release 4.6A, the DX Workbench allows BAPIs to be used for the data load in the SAP System. The use of BAPIs is increasingly important, because the previous techniques are only of limited use (or none at all) for data transfer in Release 4.6 and later:

- The batch input procedure cannot be used for the new transactions because the batch input recorder does not support the controls used in these transactions. Until SAP implements a data transfer BAPI for these new transactions, SAP will continue to support the old transactions that do not use these controls. Of course, the disadvantage here is that users have to deal with an additional transaction.
- The administration transaction associated with the direct input method may no longer be supported in the future. This means that existing direct input programs can be used, but the data transfer should be converted to BAPIs in the medium term.

BAPIs for initial data transfer replaced data transfer programs starting in release 4.6. Currently, there are approximately 30 BAPIs that have been developed especially for Mass Data Transfer.

Basics of Mass Data Transfer

The key feature of the data transfer is that data can only be transferred from a non-SAP system to an SAP System when it is available in SAP format. For this reason, the data from the non-SAP system initially has to be extracted into a file, and then converted into a transfer file in SAP format. In turn, this transfer file is the foundation for the actual data import into the SAP System.

The entire data transfer usually takes place in five technical phases, which are illustrated in the following diagram.

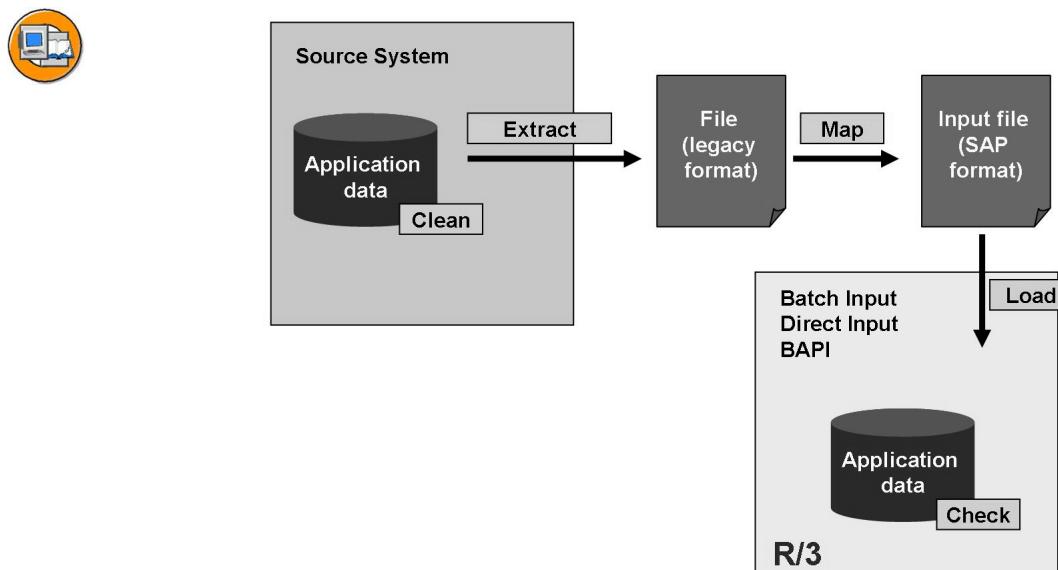


Figure 193: Data Transfer Process

These phases are described in detail:

1. Analysis and cleanup of data in the non-SAP system

This phase involves answering **three** related questions:

- Which data exists in the non-SAP system?
- How is this data structured (length, sequence)?

This step is required to assign the transferred data to the correct SAP structures later.

- Which data can be transferred unchanged, which data has to be supplemented, and which data cannot be transferred at all?

Example:

A power company manages its customers for electricity, gas, and water in three separate systems. These three systems are to be merged in a single SAP System. If customers exist who purchase both gas and

water from this company, then the data on these customers may have a different status in the different systems. For this reason, the customer data from the individual systems first has to be cleaned up and unified before it can be transferred to the SAP System.

2. Extraction of data from the non-SAP system

In this step, the determined data is extracted from the non-SAP system to a file. The data in this file still has the format of the non-SAP system. The following options are available for extracting the data:

- Using an extraction tool supplied by the non-SAP system
- Using a database tool
- Using the extraction function of a mapping tool

3. Mapping the data in SAP format

Data from a non-SAP system can only be transferred to an SAP System when it is available in SAP format. For this reason, the file containing the data in the non-SAP system format has to be transformed into a transfer file in SAP format.

Because the format of the data extracted from the non-SAP system usually differs from the SAP format of the transfer file, mapping between the different structures is required. To perform this mapping, you will have to analyze both the fields and structures of the data to transfer and the involved SAP structures, in order to determine how the conversion should be performed.

The following options are available for performing the actual mapping:

- Writing an ABAP program
- Using a commercial mapping tool
- Using LSMW as SAP's mapping tool

4. Transferring the data to the SAP System

Before you can import the data from the transfer file to the SAP System, you have to answer the following questions:

- In which sequence will the source data be copied to the business object types, and in which sequence will the involved business object types be filled with data? This question must be answered because dependencies may exist between business object types that require the data to be transferred in a specific order.
- Which data transfer technique will be used for the individual business object types? The following three standard techniques are available for the data transfer; BAPIs, Batch Input, and Direct Input.



Note: You should use BAPIs exclusively to transfer data in Release 4.6A and later.

5. Checking the data

Once the data has been transferred to the SAP System, you must verify its completeness. For example, you have to make sure that the total number of data records transferred to the SAP System is identical to the total number of data records in the non-SAP system.

These checks are often implemented through separate check programs that are run in the SAP System. These programs can be supplied either by SAP or by an external provider.

Tool Support

SAP created the DX Workbench for Release 3.1 and later to support the transfer of mass data. Starting in Release 4.6A, this tool provides integrated project management for all the steps involved in transferring data to the SAP System.

In particular, the Data Transfer Workbench offers the following functions:

- Managing and organizing data load projects
- Tools for analyzing the required SAP structures
- Integration of the standard data transfer programs
- Registration and integration of separate data transfer programs
- Support of various techniques for converting, checking, loading data (or other tasks) into the SAP System

Process Flow of the Mass Data Transfer via BAPI

To perform the mass data transfer using BAPIs, SAP supplies the DX Workbench, which integrates all the required steps in a single-user interface. This section describes at a conceptual level how the process flow of the mass data transfer via BAPI takes place in the DX Workbench.

The mass data transfer via BAPI utilizes the same ALE technique that is used for continuous data transfer. This means the data to copy from the source system is imported into the target system in IDoc format. In the target system, the IDocs are processed to generate BAPI calls that make sure the data is written to the database.

Before the mass data transfer can be performed, you have to identify the involved business objects in the target system and determine which BAPIs will be used to import the data. To determine which objects are involved, you have to establish a connection between the online transactions used during production operation and the underlying business objects.

For a BAPI to be suitable for mass data transfer, it has to have certain attributes, which are described under the topic Developing BAPIs for Mass Data Transfer.

Developing BAPIs for Mass Data Transfer

The following aspects are relevant for developing a BAPI that can be used for mass data transfer in the DX Workbench:

1. Implementing a BAPI

BAPIs for mass data transfer must be developed as described in the BAPI Programming Guide. This was discussed in detail in previous units. All write BAPIs can potentially be used for mass data processing however, the BAPIs should be able to process multiple records of data at one time. .

2. Generating the BAPI-ALE Interface

Because the data for the BAPI call is received in the SAP System in IDoc format, the BAPI-ALE interface has to be generated to automate the mapping of the IDoc to the parameters of the BAPI.

3. Writing a report

The report is responsible for writing existing SAP System objects in IDoc format to a file. Because users are normally only familiar with the online transactions, but not the BAPIs or IDocs, the reports can be used to find out how the objects that are created in the online transaction have to be represented in the IDoc. This information is required during the mapping process to correctly structure the IDocs for the objects to transfer.

This report is required, since it is the only way for users to establish the connection between the online transaction, the BAPI, and the IDoc.

4. Registering the BAPI

This step is required if the BAPI will be used in the DX Workbench. To register the BAPI, including its corresponding report, use transaction BDLR or use transaction SXDA and on the menu select *Goto → Register Programs*.

Generating the BAPI-ALE Interface

To allow a BAPI call to be generated automatically for each IDoc contained in the transfer file, the corresponding ALE interface must be generated for all mass data-capable BAPIs. The interfaces are generated using transaction BDBG.

A detailed description of the generation process flow is available in the previous unit on ALE.

The following components are generated for each BAPI:

- A message type
- An IDoc type
- An ALE inbound function module that reads the segments of an inbound IDoc, fills the parameters of the corresponding BAPI, and calls the BAPI
- An ALE outbound function module that generates an IDoc from the parameters of the BAPI and places it in ALE outbound

Writing a Report

A report can be written for each BAPI that can be used in the DX Workbench, in order to support the analysis phase. It is needed to generate IDocs for existing SAP System objects in a file. In particular, it generates IDocs for a BAPI call whose parameters have been filled with data from an existing SAP object.

The following conventions apply to the report interface:

- A parameter for the receiver of the IDoc is a required import parameter, in addition to the fields for selecting the objects.
- Because the DX Workbench uses this report to write the IDocs to a special file, the receivers are automatically passed on as soon as the report is called. As a result, this field must be defined with parameters: receiver like tbdlst-logsyst no-display or select-options: receiver for tbdlst-logsyst no-display.
- Additionally, optional parameters may be present, but they are not used by the DX Workbench.

The flow logic of the report is written as follows:

- The report first collects all the relevant data for the objects the user has selected. This can be performed, for example, using the corresponding GetDetail() BAPIs for the objects.
- It then calls the generated ALE outbound function module to generate the IDoc with the data collected above. This generates outbound IDocs in the database.

If outbound partner agreements and a file port are maintained, then these outbound IDocs are written to an outbound file—where the DX Workbench finds them and converts them to inbound IDocs, which are available to analyze the underlying structures

There are a couple of prerequisites for the report:

- The report must have parameter RECEIVER of type TBLDLST-LOGSYS.
- The report does not determine ALE receivers. No receiver determination may be performed, because although the DX Workbench requires customizing of partner agreement and port, it does not require a logical system to be assigned to the current client.

Registering the BAPI

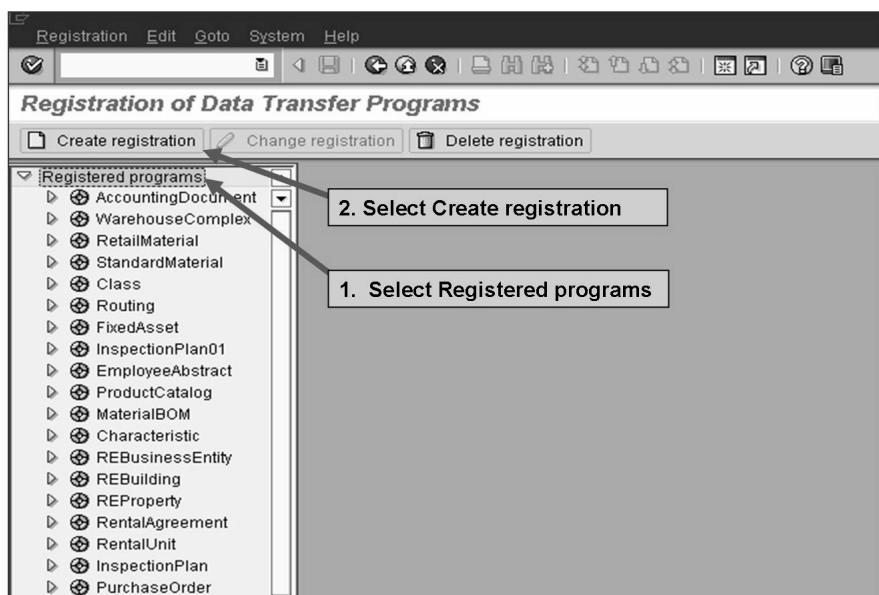


Figure 194: Register BAPI for Data Transfer (1)

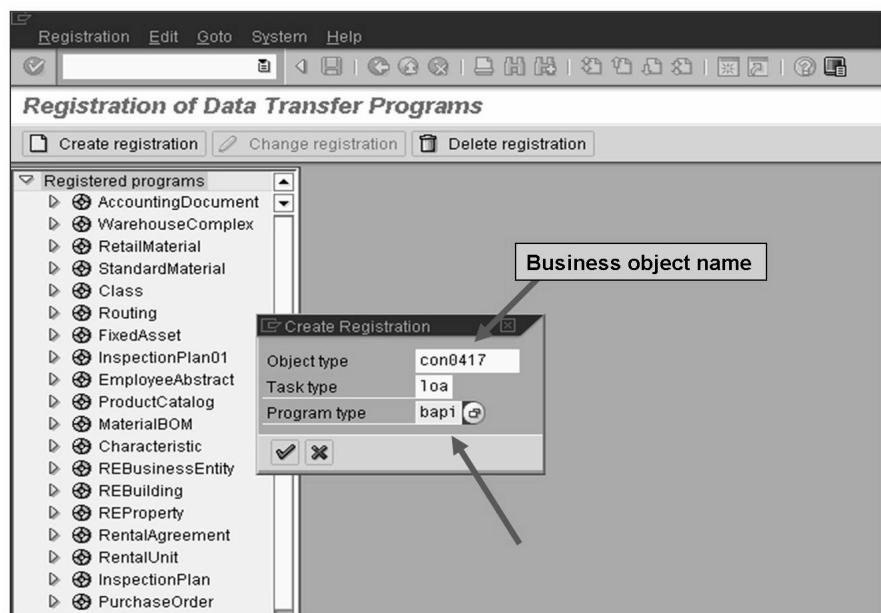


Figure 195: Register BAPI for Data Transfer (2)

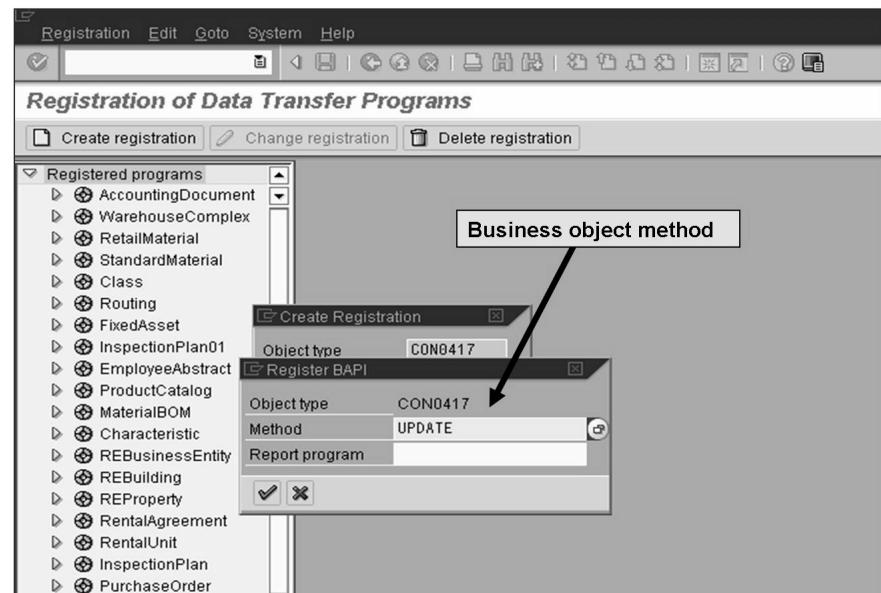


Figure 196: Register BAPI for Data Transfer (3)

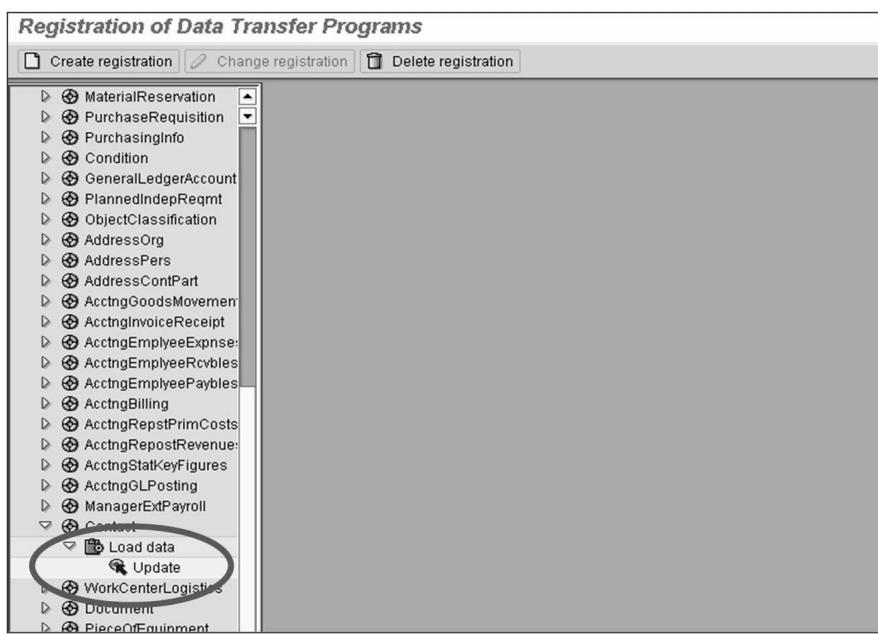


Figure 197: Register BAPI for Data Transfer (4)

To use a BAPI in the DX workbench, you have to register it and its corresponding report. This can be done using transaction BDLR or using transaction SXDA and on the menu select *Goto → Register Programs*.

Since any BAPI can be registered for Mass Data Transfer, you must make sure that the BAPI does not contain a dialog.

The DX workbench only registers a BAPI. It does not ensure that the BAPI is used for mass data. This is the responsibility of the developer. For instance, the DX workbench would not prevent a BAPI that performs just a single update record at a time from being registered.

Exercise 17: Mass Data Transfer

Exercise Objectives

After completing this exercise, you will be able to:

- Register a BAPI for Mass Data Transfer

Business Example

Legacy data from an existing customer system needs to be loaded into an SAP System.

Task:

Register a BAPI

1. Create the BAPI registration in the DX Workbench. Use the object name **ZCON_##**.

Solution 17: Mass Data Transfer

Task:

Register a BAPI

1. Create the BAPI registration in the DX Workbench. Use the object name **ZCON_##**.
 - a) From the SAP menu, select *Tools → Data Transfer Workbench or execute transaction SXDA*.
 - b) On the menu, select *Goto → Register programs*. You can also use transaction BDLR to go directly to the Register programs screen.
 - c) Select the *Registered programs* title and select *Create registration*.
 - d) In the create registration window, enter the following information:
 - Object type (example: **ZCON_##**)
 - Task type (example: **LOA** (Load Data))
 - Program type (example: **BAPI**)
 - e) Select *Continue*.
 - f) In the *Register BAPI* dialog box, select *F4* and the Method you want to register (example: Update)
 - g) If you are using a report program, enter the name of the program in the *Report program* field.
 **Note:** this program would be used to create the IDoc.
 - h) Select *Continue*.
 - i) Save your work to your Change Request and Package.
 - j) If the registration is successful, you will receive a message stating that the Method was registered for your object.



Lesson Summary

You should now be able to:

- Explain the concept of Mass Data Transfer



Unit Summary

You should now be able to:

- Explain the concept of Mass Data Transfer

Related Information

- For additional information on the transfer of mass data, refer to the BAPIs for Mass Data Transfer Ecc6.0 on SAPNet.



Test Your Knowledge

1. The following aspects are relevant for developing a BAPI that can be used for Mass Data Transfer in the DX workbench; Implementing a BAPI, Generating the ALE interface, Writing a report, and Registering the BAPIs.
Determine whether this statement is true or false.
 True
 False

2. When a BAPI is registered in the DX workbench for Mass Data Transfer, the DX workbench checks to ensure that the BAPI can process multiple records of data.
Determine whether this statement is true or false.
 True
 False

3. In the technical phase of Analysis and Cleanup of data in the non-SAP system, legacy data can be transferred unchanged to the SAP system.
Determine whether this statement is true or false.
 True
 False



Answers

1. The following aspects are relevant for developing a BAPI that can be used for Mass Data Transfer in the DX workbench; Implementing a BAPI, Generating the ALE interface, Writing a report, and Registering the BAPIs.

Answer: True

Yes, all of these are relevant aspects for developing a BAPI that can be used for Mass Data Transfer.

2. When a BAPI is registered in the DX workbench for Mass Data Transfer, the DX workbench checks to ensure that the BAPI can process multiple records of data.

Answer: False

Any BAPI can be registered in the DX workbench. It is the responsibility of the developer to ensure that the BAPI can process multiple records of data.

3. In the technical phase of Analysis and Cleanup of data in the non-SAP system, legacy data can be transferred unchanged to the SAP system.

Answer: True

It is possible to transfer legacy data as-is to an SAP system if the data is already in a format that is accepted in the SAP system.

Internal Use SAP Partner Only

Internal Use SAP Partner Only



Course Summary

You should now be able to:

- Find and use BAPIs developed by SAP
- Design and maintain your own BAPIs
- Locate or create Business Objects
- Enhance SAP Delivered BAPIs
- Perform external calls to BAPIs

Appendix 1

Implementation Project for New BAPIs

This form will guide you through all the work steps necessary for implementing BAPIs. It also enables you to create a checklist to document the steps as you complete them, thereby providing you with a record of the creation process for your BAPI.

Complete this form for each of your projects. One project may cover either a single BAPI, or several BAPIs within the framework of one scenario. However, there must only be one person responsible for each project.

1. Project Organization
2. Scenario Analysis
3. BAPI Design
4. Defining BAPI Data Structures
5. Creating BAPI Function Modules
6. Defining BAPI Methods in the BOR
7. Documenting BAPIs
8. BAPI/ALE Integration - Generating IDocs
9. ToDo Checks and Tests
10. Release

Project Organization

Project name

Person responsible for project

A

Quality Manager

Scenario Analysis

BAPIs are always implemented as part of an ALE integration scenario. This ensures that the various BAPIs complement each other and provide structured, focused access to R/3 functionality.

When analyzing the scenario, you must answer the following questions:

- What requirements need to be met?
- How can the business process be described in business terms? (= business process description)
- How can the business process be described in software terms? (= scenario description)
- Which business objects are affected? Can you use an existing object model?
- Which methods (BAPIs) are required?

 **Note:** You can find further information in the *BAPI Programming Guide* under “Analysis.”

Name of the ALE Integration scenario



Object model exists

The following BOR methods (BAPIs) must be implemented for the scenario:

Business Object	Method (BAPI)	Function module
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>

BAPI Design

During the design phase, you must define the BAPI signature at a conceptual level. Then describe the contents of the BAPI and specify the parameter names and parameter structure. You can find detailed information on the design phase in the *BAPI Programming Guide* under “Designing the BAPI.”

During the design phase, you must bear in mind that there is a series of **standard methods and parameters** that provide certain basic functions. If the BAPIs you want to implement and their parameters are included in this category, then

you must adhere to the relevant guidelines. You can find an overview in the *BAPI Programming Guide* under “Standardized BAPIs” and under “Standardized Parameter.”

When you are working on the implementation of a BAPI, you must consider whether, based on its functionality, it is an instance method or a class method.

Instance methods refer to a specific instance of a business object; for example, the SalesOrder.GetDetail BAPI displays the detailed data for precisely one sales order (which you must specify). Such methods are defined as **instance-dependent** in the BOR.

Class methods do not refer to a specific instance of a business object; for instance, the SalesOrder.GetList BAPI gets a list of all available sales orders that fulfill certain criteria. Class methods include standardized create methods. Create methods are used to create new instances; for example, the SalesOrder.Create BAPI creates a new sales order in the SAP System. Such methods are defined as **instance-independent** in the BOR.

You must adhere to the following conventions for **BAPI methods**:

- You must establish whether you are dealing with an instance method or a class method.
- The method name must be in English (maximum 30 characters).
- The individual name components of BAPI names are marked by initial capital letters. Underscores (_) are not permitted in BAPI names.
- Every BAPI must have a return parameter that is either an export parameter or an export table.
- Every BAPI must have an ExtensionIn and an ExtensionOut parameter to enable customer enhancements. You can find further information on customer enhancements of BAPIs in the document titled “Customer Enhancement and Modification of BAPIs”.

The following conventions are valid for **parameters**:

- The parameter and field names must be in English (maximum 30 characters).
- The individual name components of parameter names are marked by initial capital letters to ensure maximum readability.
- Fields that are ISO-relevant (country, language, quantity, currency) have additional fields for ISO codes.
- You must include quantity unit fields for all quantity fields, and currency descriptors for all currency amount fields.

Defining BAPI Data Structures

You must now define, in the ABAP Dictionary, all data structures (including domains and data elements) that are required for the parameters of the BAPI to be implemented. You can find further information on defining BAPI data structures in the *BAPI Programming Guide* under “Actions in the ABAP Dictionary.” ABAP Dictionary (SE11)

The following conventions are valid in this phase:

Conventions for BAPI Data Structures:

- Each parameter must refer to a data structure in the ABAP Dictionary. For structured parameters, this is always the overall BAPI data structure. However, if the parameter consists of only one field, then it must refer to a field in a BAPI data structure.
- You must create specific data structures for BAPIs, which must be independent of the data structures that are used generally in R/3 applications.

Tool: Existing internal structures can be mapped to existing BAPI structures using a module to map fields that is automatically generated by a mapping tool (Transaction BDBS). Mapping Tool
- All data structure names must begin with < namespace > BAPI.
- You must not use APPENDs or INCLUDEs in BAPI data structures.

Conventions for field names.

- Fields in structures must have easily understandable English names that are no longer than 30 characters.

Tool: Internally at SAP, you can use the BAPIFELD report to create a list of proposals for English field names for an ABAP Dictionary structure. You can find a description of how to use this report in the report documentation. Report BAPIFELD
- You must also create an easily understandable English default name that is no longer than 30 characters for each data element. This name then becomes the proposal always given by the BAPIFELD SAP internal report.

Tool: The BAPIFELD report shows which data elements still need to be processed. Enter the default field name for the data element in the ABAP Dictionary using Definition ® Default field name. Report BAPIFELD

Conventions for Input Help

- Individual values or a value table may be maintained for a domain, which enables you to use the F4 help.
- All useful input help (that can be used with the HelpvaluesGetList service BAPI) must be maintained for the data structures/data elements. To do this, you must specify foreign keys for the fields of a BAPI

Creating BAPI Function Modules

Once the parameters have been defined, the function module on which the BAPI is based is implemented in the Function Builder (Transaction SE37). Function Builder (SE37) When creating this module, you must observe the following rules:

- The BAPI is a transactional BAPI. For more information, see the *BAPI Programming Guide* under “Transaction Model for Developing BAPIs.”
- All function modules must correspond to the following naming convention: BAPI_<Business-Object>_<Method>. A maximum of 30 characters is permitted. If you need to, you can use abbreviations that correspond to the above convention, as long as the assignment to the business object remains clear.
- All BAPIs for one SAP business object type should be summarized to a function group. You should only deviate from this rule in exceptional cases.
- BAPIs for different object types must not be summarized to a function group.

Interface Definition

A BAPI parameter must be defined both in the function module and in the method for that business object type in the BOR. The definitions must be identical in both cases, excepting only the key fields of the business object type.

You can find further information on interface definition in the *BAPI Programming Guide* under “Defining the interface.”

The following conventions are valid for key fields with function module interfaces :

- For **instance method**, all BOR key fields are mandatory import parameters in the function module. BOR key fields cannot be export parameters in the function module.
- For **class methods**, BOR key fields cannot be export parameters in the function module (exception: create methods). BOR key fields also cannot exist as import parameters in the function module.
- For **create methods**, all BOR key fields for that method are export parameters in the function module. BOR key fields (as for all class methods) cannot be import parameters in the function module. Create methods are defined as instance-independent in the BOR.

If key fields occur as parameters in function modules, then you must observe the following conventions:

- You must use the fully defined key, not the partial key.
- Each key for the business object type has its own parameter in the function module.
- The function module parameter and the BOR key field have the same name.

You must also observe the following conventions for function module interfaces:

- You must use the BAPIPAREX structure for parameters *ExtensionIn* and *ExtensionOut*.
- You must base the return parameter on structure BAPIRET2.
- The function module must be RFC-capable.

Implementing the Function Module

The most important BAPI-specific requirements for function module coding are as follows:

- A BAPI must not execute a COMMIT-WORK command.
- The database can only be changed by updating.
- A BAPI cannot display anything on the screen, that is, it cannot create lists, queries or dialog boxes. This is valid not just for the BAPI itself, but also for all function

Defining BAPI Methods in the BOR

Once the implementation of the function module is complete, you must define the BAPI as a business object method. This not only enables central storage in the Business Object Repository (BOR), but also object-oriented access to the BAPI and the option to link to the ALE distribution model (for asynchronous BAPIs). BOR (SWO1)

Use the BOR/BAPI Wizard to implement the BAPI as a method in the BOR. This automatically generates the BOR method from the function module. In the BOR, first choose the change mode for the object you are working on and then the menu path:

Utilities → API methods → Add method

Follow the steps in the wizard. You will have to complete a few additional tasks in the wizard. Note the following points in particular:

- Initial capital letter for each word.
- Ensure that the import and export behavior of the table parameters is correctly maintained in the BOR. Reason: Unlike function modules, in the BOR you can differentiate between import and export for tables. You should only choose Import/Export if the table has actually been imported or exported.
- The return parameter must always be defined as export.

Apart from simplifying the tasks, the wizard also ensures that the BAPI interface is identical in both the BOR and the function module. This is particularly important for the following aspects:

- For every function module parameter there is a BOR parameter (or a key field) with the same name (and vice versa).
- These two parameters refer to the same ABAP Dictionary structure.
- These two parameters have the same characteristics (mandatory or optional, import, export or table parameters).



Note: If you cannot use the BOR/BAPI wizard (for example, if you are changing the BAPI at a later date or redefining a method), work on the method in the BOR itself. If you are doing this, you must take particular care to observe the points listed above. BOR (SWO1)

IBUs, customers and partners must also observe the following additional points:

- All BAPIs and all parameters are fully documented.
- The return parameter documentation contains all relevant error messages.
- The documentation developer then saves the active version of the function module and parameter documentation (otherwise the documentation cannot be translated).
- The business objects have been documented in the BOR.

BAPI/ALE Integration - Generating IDocs

BAPIs are integrated into the ALE distribution model. ALE distribution is asynchronous and based on messages. ALE distribution is the preferred means of integration for distributed SAP Systems, for instance, for distributing master data.

As of Release 4.0, **BAPIs are the standardized interfaces for ALE-supported distribution**. ALE services, such as asynchronous calls, use of the distribution model, monitoring and error processing, can all be used for BAPIs. The IDocs required for ALE services can be generated from the BAPIs.

- Message types are created and IDocs are generated for all write BAPIs.
Generate ALE Interface for BAPI (BDBG)
- All IDocs are released.
Releasing IDocs is currently linked to authorizations.
- The BAPI interface and the IDoc interface are identical.
Ensure that, when you have changed a BAPI, the message type, the IDoc type and the segments are regenerated.

You can find further information in the document titled *Using ALE-Services*.

ToDo Checks and Tests

You must run special tests and carry out ToDo checks in order to ensure that the BAPIs, the business object types and the documentation are correct. You can find further information on the test phase in the *BAPI Programming Guide* under "Testing and Releasing."

- A ToDo check has been carried out for the BAPIs and no messages with priority 1 or 2 were displayed.
- The single test and the integration test were successful.
- The documentation has been checked for completeness and clarity.

Release

Once all the tests have been completed successfully, the BAPIs and all related development objects must be released in order for them to be available to the customer. This can happen once the following conditions have been met:

- All required documentation has been created.
- No consistency errors were found.

You must carry out the following steps when releasing a BAPI:

- Release the BAPI function module (in the Function Builder).
- Release the business object type (in the BOR).
- Release the BAPI as a method in the BOR.
- For potential write BAPIs: Release the IDoc and its segments.

Function Module

Naming Convention and Documentation

Function module identifiers:

- The name of a function module comprises an object and an action.
- The name of the object appears first.
- The components of the name are English words.
- The components of the name are separated by the underscore character.
- Abbreviations are avoided.
- The customer naming space is not used: this means that a name cannot begin with **Z_** or **Y_**.

Short text of a function module:

- The short text is understandable and meaningful.
- The object and the action are briefly described.
- The text is formulated in an abbreviated fashion.

Function documentation:

- The documentation is understandable and meaningful.
- It does not include details of the technical implementation unless these are relevant for the user.
- The documentation is structured as follows:
 - Functionality (What does the function module do?)
 - Examples (How do I use the function module?)
 - Hints (What should I watch out for when using the function module?)
 - Further sources of information (Where can I find more information?)

Parameter naming:

- The components of the name are English words.
- The components of the name are separated by the underscore character.
- Abbreviations are avoided.

Parameter documentation:

- The short text is understandable and meaningful.
- The long text is understandable and meaningful.
- The action of the control parameters is described.
- The documentation is structured as follows:
 - Meaning (What is the effect of each parameter?)
 - Valid ranges (Which values are valid/possible?)
 - Pre-assigns (Are import parameters pre-assigned? If so, how?)

Exception naming:

- The name comprises an object and a state.
- The components of the name are English words.
- The components of the name are separated by the underscore character.
- Abbreviations are avoided.

Exception documentation:

- The short text is understandable and meaningful.
- The long text is understandable and meaningful.
- The documentation is structured as follows:
 - Meaning (What caused the exception to be raised?)
 - Method of generation (How was the exception raised? Were any parameters passed with messages? If so, which?)

Interface Structure

The interface as a whole:

- This function module is the only one in the Function Library with this or a similar task.
- The interface is as simple as possible.
- Each function module provides a clearly defined function (The interface should not be used to choose between several different functions contained within one function module).
- The import parameters are pre-assigned in a consistent way.
- The interface should not be globally defined.

Parameters:

- A reference field, a reference structure or a type should be used wherever possible.
- Reference fields and reference structures are chosen as generally as possible.
- Sensible pre-assignments are used for import parameters.
- All import parameters specified in the interface are used.
- All export parameters are assigned values within the function module.

Exceptions:

- An error always causes an exception to be raised.
- All exceptions should be generated using MESSAGE ... RAISING.
 - The error messages have sufficient parameters.
 - The situation causing the error is described in detail in the long text of the error message.
 - The standard for error message short texts is adhered to. (See *Style Guide*, Chapter 3, “General Messages”.)

Implementation Notes

Structure:

- In order to reduce I/O, the function group memory is used.
- Whenever possible, data is defined and used locally.

Function Group

Naming Conventions and Documentation

Function group identifiers:

- The first letter of a function group name is the departmental abbreviation of the department responsible.
- The customer naming space is not used: this means that a name cannot begin with **Z** or **Y**.

Function group short text:

- The short text is understandable and meaningful.
- The key objects and activities are briefly described in context.
- The short text is formulated in an abbreviated fashion.

Function group documentation:

- The documentation is understandable and meaningful.
- The documentation is structured as follows:
 - Functionality (What does the function group do?)
 - Overview (Which function modules are contained in this group?)
 - Examples (How can I use the function modules in this group?)
 - Hints (Which is the most appropriate function module?)
 - Further sources of information (Where can I find more information?)

Function Group Design

Structure:

- The function modules within a single function group do not just belong together logically but are also structurally interdependent.
- The function group should not contain too many function modules.

Index

A

ABAP

- Business Application Programming Interface (BAPI), 243
- ABAP Dictionary, 58, 60, 148
 - Authorization object, 176
- ABAP Workbench, 59
 - Business Application Programming Interface (BAPI), 147
- ACID principle, 15
- ALE distribution model, 328
 - dependencies, 329
- Application Link Enabling (ALE), 4, 318
 - advantages, 319
 - BAPI-ALE interface, 321
 - distribution model, 328
- Authorization, 175
 - Authorization check, 177
 - Authorization object, 176
 - profiles, 177
- Authorization check, 177
 - performing, 178
- Authorization object, 176
 - ABAP Dictionary, 176

B

BAPI Browser

- Business Object Repository (BOR), 131
- BAPI development, 56
- BAPI ExistenceCheck
 - business object, 139
- Business Workflow, 140
- BAPI Explorer, 28, 56, 59

Business Object Repository (BOR), 59

- display functions, 28
 - documentation, 100
 - projects, 30, 60
 - structure, 28
 - tools, 30, 60
- BAPI testing, 120
- BAPI-ALE interface, 321
 - changing, 326
 - checking, 327
 - creating, 325
 - deleting, 326
 - displaying, 326
 - generating, 342
 - prerequisites, 324
 - procedure, 325
- BOR/BAPI Wizard, 147–148
 - process, 148

- Business Application Programming Interface (BAPI), 4, 7, 11, 56, 284
 - ABAP, 243
 - ABAP Workbench, 147
 - BAPI Explorer, 28, 56, 59
 - BAPI-ALE interface, 321
 - benefits, 12
 - BOR/BAPI Wizard, 147
 - Business Connector (BC), 11
 - Business Object Repository (BOR), 12, 58, 127
 - business object type, 8
 - characteristics, 8
 - conventions, 62
 - currency, 62, 99

- customer enhancement, 284
- defining, 148
- eXtensible Markup Language (XML), 11
- external key, 62
- Function Builder, 58
- hierarchies between parameters, 320
- implementation, 59
- implementing, 319
- integrating application systems, 55
- integration, 9
- internal key, 62
- Logical Unit of Work (LUW), 16
- naming conventions, 68
- parameter, 93, 106
- projects, 56
- scenario, 78
- source code, 95
- support, 11
- testing, 120, 154
- transaction handling, 16
- transaction model, 15
- update task, 16
- usability, 10
- Business Application Programming Interfaces (BAPI)**
 - integrating alternative frontends, 55
 - service implementation, 55
- business component, 3
- Business Connector (BC)**
 - Business Application Programming Interface (BAPI), 11
- Business Framework, 2
 - Application Link Enabling (ALE), 4
 - architecture, 2
- Business Application Programming Interface (BAPI), 4
- business component, 3
- business object, 3
- communication services, 4
- components, 3
- business object, 3
 - application, 137
 - BAPI ExistenceCheck, 139
 - Business Object Builder, 135
 - Business Object Repository (BOR), 6, 12
 - business rules, 6
 - characteristics, 6, 136
 - components, 137
 - constraint, 6
 - creating, 135
 - description, 136
 - interface, 6, 137
 - key field, 7
 - methods, 137
 - name, 136
 - object name, 136
 - object type, 136
 - program, 137
 - relation attribute, 137
 - Remote Function Call (RFC), 6
 - status, 139
 - supertype, 136
 - type, 4, 129
- Business Object Builder
 - business object, 135
- Business Object Repository (BOR), 131
- Business Object Repository (BOR), 6, 12, 58, 65, 127
 - BAPI Browser, 131
 - BAPI Explorer, 59
 - BOR/BAPI Wizard, 147

- Business Object Builder, 131
- Business Object Repository Browser, 131
 - functions, 65, 129
 - navigation, 131
 - object type, 127
 - services, 13
- Business Object Repository Browser
 - Business Object Repository (BOR), 131
- business object type, 4, 129
- Business Application Programming Interface (BAPI), 8
- Business Object Repository (BOR), 131
 - components, 129
 - structure, 5
- business process, 51, 77
 - defining, 51, 77
 - scenario, 52, 77
 - Use Case, 77
- business rules, 6
- Business Workflow
 - BAPI ExistenceCheck, 140
- C**
 - communication services, 4
 - currency
 - guidelines, 99
 - customer enhancement, 284
 - advantages/disadvantages, 286
- D**
 - Data Browser, 68
 - data structure
 - conventions, 92
 - defining, 92
 - Data Transfer Workbench (DX Workbench), 338–339
- functions, 341
- mass data transfer, 341
- reports, 343
- database
 - database locks, 166
 - explicit commit, 165
 - explicit database rollback, 166
 - implicit commit, 164
 - implicit database rollback, 166
 - locks, 183
 - update, 199
- database locks, 183
- database Logical Unit of Work (LUW), 166
- function modules, 195
- lock object, 191
- logical locks, 184
- parameters, 195
- update modules, 208
- database Logical Unit of Work (LUW), 163–164
 - database locks, 166
- database rollback, 207
- database update, 199
 - asynchronous update, 203, 209
 - database locks, 208
 - database rollback, 207
 - dialog program, 201, 207
 - local update, 204, 211
 - process, 200
 - synchronous update, 204, 210
 - technical implementation, 204
 - timescale comparison, 212
 - update key, 206
 - update modules, 204
 - update program, 200
 - V1 function module, 213
 - V2 function module, 213
- documentation
 - parameter, 106
 - testing, 120

E

eXtensible Markup Language (XML), 11

F

Function Builder, 58, 63, 148
 documentation, 105
 Function Library, 63
 function module, 64

function group
 function module, 64
 Function Library
 Function Builder, 63
 function module, 64
 creating, 216
 documentation, 103
 function group, 64
 V1, 213
 V2, 213

L

lock object, 195
 creating, 192
 deleting, 194
 logical locks, 184
 coding considerations, 185
 function modules, 195
 lock argument, 187
 lock mode, 188
 lock object, 191
 lock table, 187
 parameters, 195
 releasing, 185
 setting, 185

Logical Unit of Work (LUW)
 Business Application Programming Interface (BAPI), 16
 database, 163
 SAP Logical Unit of Work (LUW), 163, 166
 transaction, 15
 transactional RFC (tRFC), 246

M

mass data transfer, 338–339
 Data Transfer Workbench (DX Workbench), 338–339, 341
 developing BAPIs for, 342
 phases, 339
 process flow, 341
 registering BAPI for, 346
 tool support, 341
 Message Maintenance, 68

P

parameter
 extension parameter, 93
 RETURN parameter, 93
 selection parameter, 93
 project
 defining, 78

Q

queued RFC (qRFC)
 process, 247
 Remote Function Call (RFC), 247
 syntax, 248

R

Remote Function Call (RFC), 6, 242, 245
 asynchronous RFC, 245
 Authorization check, 253
 creating remote destinations, 249
 destination, 249
 features, 243
 queued RFC (qRFC), 247
 Remote System, 244
 RFC client, 243
 RFC server, 243
 synchronous RFC, 245
 table RFCDES, 248
 transactional RFC (tRFC), 244
 Remote System

- Remote Function Call (RFC), 244
- Repository Information System, 68
- RFC client, 243
- RFC server, 243
- S**
- SAP Logical Unit of Work (LUW), 163, 166
bundling changes, 168
SAP transaction, 163
- SAP transaction, 163, 173
- scenario, 52
- Business Application Programming Interface (BAPI), 78
- business object types, 54
- business process, 77
- components, 53
- defining, 53, 77
- definition, 56
- person responsible, 78
- project, 78
- reviewing, 56, 79
- source code
- guidelines, 95
- synchronous RFC
- process, 245
- Remote Function Call (RFC), 245
- syntax, 246
- T**
- testing
- Business Application Programming Interface (BAPI), 120
- documentation, 120
- error message, 121
- transaction
- handling, 16
- transaction handling, 16
- transaction model, 15
- ACID principle, 15
- client control, 16
- Logical Unit of Work (LUW), 15
- orientation, 16
- with commit, 18
- without commit, 17
- transactional RFC (tRFC), 244
- Logical Unit of Work (LUW), 246
- process, 246
- U**
- update modules
- database locks, 208
- Use Case, 77
- V**
- V1 function module
- generating updates, 214
- V2 function module
- generating updates, 214–215
- Virtual Interface, 263
- W**
- Web Service Definition, 263

Feedback

SAP AG has made every effort in the preparation of this course to ensure the accuracy and completeness of the materials. If you have any corrections or suggestions for improvement, please record them in the appropriate place in the course evaluation.