BAPI User Guide (CA-BFA)

HELP.BCMIDAPIINTRO

Release 4.6C

Copyright

Icons

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

Contents

BAPI User Guide CA-BFA)

BAPI User Guide

Use

The Business Framework - the open, component-based architecture, which allows software components from SAP and third parties to interact and integrate with each other, is becoming more and more important.

SAP business objects are at the heart of the Business Framework, enabling the interoperability of software components. They cover a wide spectrum of R/3 business data and processes.

SAP business objects are accessed through BAPIs (Business Application Programming Interfaces), which are stable, standardized methods. SAP business objects and their BAPIs provide an object-oriented view of R/3 business functions.

SAP provided the first BAPIs for customers and external providers in Release 3.1, enabling them to integrate their software components in the R/3 System and the Business Framework. The number of BAPIs is increasing with each R/3 Release and with this the extent of object-oriented access to the R/3 System.

This User Guide is an introduction to the technical concepts behind BAPIs. It also shows you how you can use BAPIs to achieve seamless integration between the R/3 System and external applications, legacy systems and add-ons.

Prerequisites

The Use Guide is targeted at application developers wanting to create new integrated activities using SAP Business Objects and their BAPIs. For example:

- Integration of external applications with the R/3 System
- Integration of existing business applications with the R/3 System
- Integration of new R/3 components, for example, Advanced Planner & Optimizer (APO) and Business Information Warehouse (BW).
- Alternative front-end interfaces to R/3 Systems, so that, for example, occasional users can access R/3.

  These front-ends can be implemented as Windows-based client applications, for example, as macros in desktop applications (Microsoft Excel or Access), which are usually written in Visual Basic for Applications or implemented as applications in Visual Basic, Java or C++.
- Web-based access to the R/3 System through Internet or Intranet applications.
- Componentization of the R/3 System within the Business Framework

Required Knowledge

To use BAPIs to access SAP business objects you will need a good understanding of object-oriented programming. You should also have a basic knowledge of the R/3 System.

BAPIs can be accessed from various programming environments, for example, Delphi/Connect from Inprise (formerly Borland), Microsoft's Visual Studio, C++ and Java platforms. You must be familiar with the development environment used to access BAPIs.

BAPIs are available from development platforms external to R/3 that support the Remote Function Call (RFC) protocol. If you are developing your application in a non-object oriented programming language, you need to have RFC programming skills to call BAPIs via RFC. In particular, you need to know how to perform RFC calls.

Further Documentation

In the *Open BAPI Network* under *SAP Business Technology/Business Framework* in **SAPNet**, application developers can view up-to-the-minute information on BAPI developments undertaken by SAP and by partners.

Further documentation on topics relating to BAPIs can be found under CA-BFA in the SAP Library and under *SAP Business Technology/Business Framework* in the **BAPI Knowledge Base** in **SAPNet**.

See also:

- Customer Enhancements and Modifications for BAPIs [Ext.]
- SAP Enhancements to Released BAPIs [Page 57]
- BAPIs for Mass Data Transfer [Ext.]
- Use of ALE Services [Ext.]

Terminology

Definition

The table below contains definitions of the terms used in this document:

Term/Synonym	**Definition**
Object type Business object type SAP business object type	Generic definition of business objects. Describes the general characteristics and common attributes of instances of SAP business object types.
Business object Business object instance SAP business object instance	One instance of an SAP business object type
BOR Business Object Repository	Central repository within the R/3 System containing all the SAP business object types and SAP interface types together with their definitions and methods.
BAPI Business Application Programming Interface	Standardized programming interface enabling external applications to access business processes and data in the R/3 System. BAPIs are defined in the BOR as methods of SAP business object types that carry out specific business functions. BAPIs are implemented as RFC-enabled function modules and are created in the Function Builder of the ABAP Workbench.
Interface type SAP interface type	Definitions of attributes, methods and events grouped together in one context. An object type can support one or more interface types. The functional group of associated attributes, methods and events defined for the interface type together with their interfaces are used by the object type and are inherited by the respective subtypes of the object.

Introduction

Use

SAP has introduced object-oriented technology in the R/3 System by making R/3 processes and data available in the form of SAP business objects.

External applications can access SAP business objects through standardized, platform-independent interfaces - Business Application Programming Interfaces (BAPIs). SAP business objects and their BAPIs provide an object-oriented view of R/3 business functionality.

This guide provides an overview of the Business Framework, SAP business objects and SAP interface types and their BAPIs as well as the Business Object Repository where business objects and BAPIs are defined and stored.

Business Framework

Use

The SAP R/3 Business Framework provides a structure for R/3 functionality based on application components (business components) and object models. The Business Framework enables customers and partners to connect their own components to the R/3 System.

The use of object-oriented technology and object models reduces the complexity of the global system.

Features

The architecture of the Business Framework provides the basis for developing SAP business components. The basic components of the 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 R/3 System is based on business objects. They encapsulate business data and functions and define the scope and the boundaries of a business component.

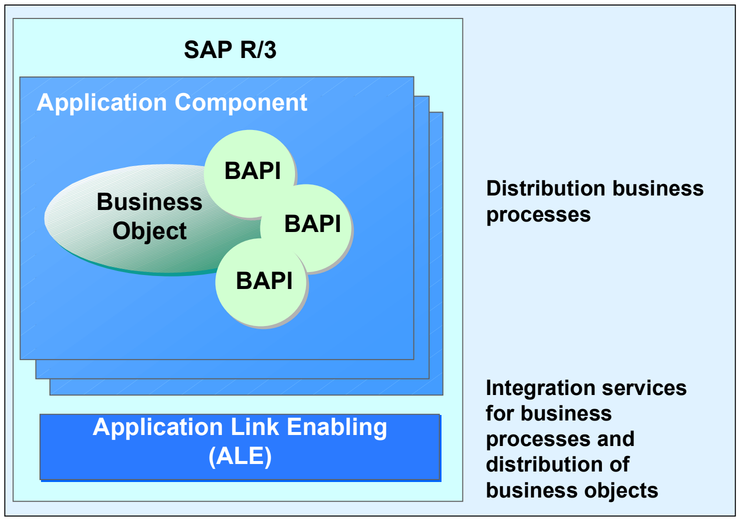
- Business Application Programming Interfaces (BAPIs)

  BAPIs are interfaces for business objects. Together with the business objects, BAPIs define and document the business interface standard.

- The integration service, Application Link Enabling (ALE)

  The ALE integration service enables business processes carried out in different R/3 and non-SAP systems to be integrated. It involves distributing business objects across the systems using the ALE distribution model.

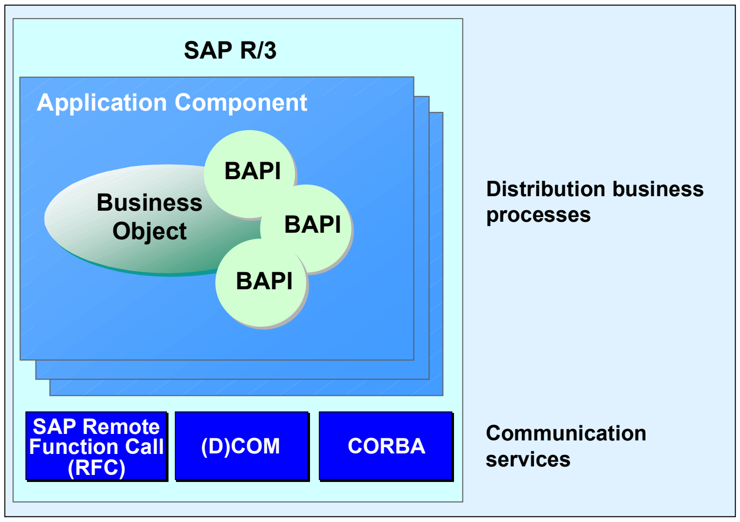
  The graphic below illustrates this architecture.


SAP R/3
Application Component
Business Object
BAPI
BAPI
BAPI
Application Link Enabling (ALE)
Distribution business processes
Integration services for business processes and distribution of business objects

- Communication services

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

  The graphic below illustrates this architecture.



SAP Business Objects

Definition

Business object technology and programming are based on the concept of business objects. Real objects, such as an employee or a sales order can be represented as business objects in business application systems, such as the R/3 System.

SAP business objects encapsulate R/3 data and business processes whilst hiding the structure and implementation details of the underlying data.

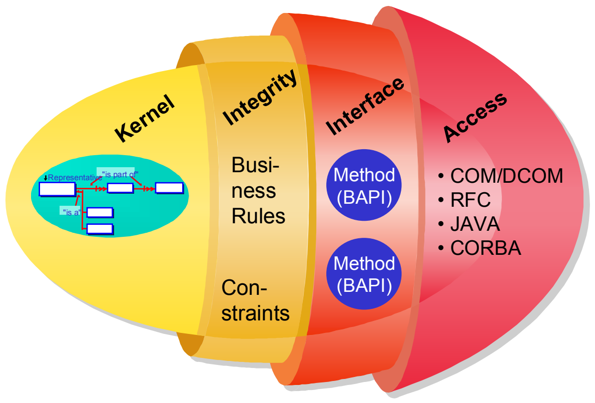
Structure

To achieve this encapsulation, SAP business objects are created as entities with several layers.

- At the core of an SAP business object is the kernel, which represents the object's inherent data.
- The second layer, the integrity layer, represents the business logic of the object. It comprises the business rules for consistent embedding in the environment and the constraints regarding the values and domains that apply to the business object.
- The third layer, the interface layer, describes the implementation and structure of the SAP business object, and defines the object's interface to the outside world.
- The fourth and outermost layer of a business object is the access layer, which defines the technologies that can be used for external access to the object's data, for example, COM/DCOM ((Component Object Model/Distributed Component Object Model). It specifies the technologies that enable object data to be accessed from external programs, for example, COM/DCOM (Component Object Model/Distributed Component Object Model).

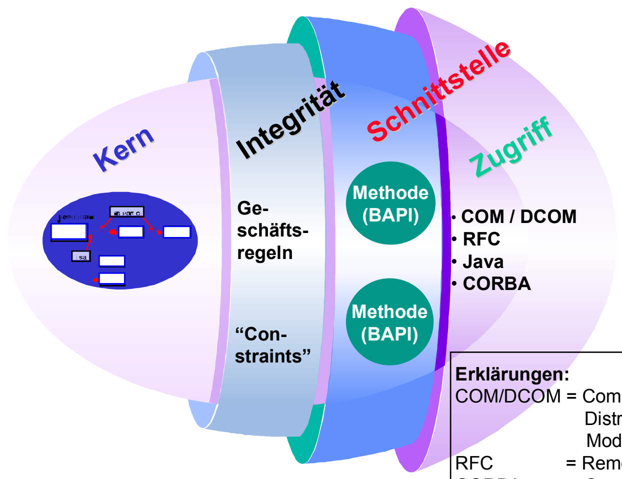
The graphic below illustrates the different layers of a business object.

SAP Business Objects



Legend:

COM/DCOM	=	Component Object Model/ Distributed Component Object Model
RFC	=	Remote Function Call
CORBA	=	Common Object Request Broker Architecture



Erklärungen:

COM/DCOM	=	Component Object Model / Distributed Component Object Model
RFC	=	Remote Function Call
CORBA	=	Common Object Request Broker Architecture

the applications
; reveal only
ın only access

nly needs the
; can work with
ɔut the object's

ıavior. When a
ernal **state**, that

› "check for the

nd described in

Each individual business object belongs to a specific object class, depending on the nature and general characteristics of the object. These object classes are called **object types**. For example, the individual employees working in an organization are all part of the *Employee* object type.

The object types are descriptions of actual SAP business objects that can exist in R/3; that is, each individual SAP business object is a representation, or **instance**, of its object type. For example, the employee with the name *Charlie Jones* and the employee number *1234* is an instance of the *Employee* object type.

When writing object-oriented application programs, developers identify the object types that are to be used in their programs. At runtime, the application program accesses the specific instances of the defined object types.

When an instance of a business object is used by an application program, the object instance responds only to the set of characteristics and methods defined for its own object type. SAP business object types are defined by the following:

- **Object types**

  The object type describes the features common to all instances of that object type. This includes information such as the unique name of the object type, its classification, and the data model.

- **Key fields**

  The key fields determine the structure of an identifying key, which allows an application to access a specific instance of the object type. The object type *Employee* and the key field *Employee.Number* are examples of an object type and a corresponding key field.

- **Methods**

  A method is an activity that can be performed on a business object and that provides access to the object data. A method is defined by a name and a set of parameters and exceptions, which can or must be provided by the calling program in order to use the method. BAPIs are examples of such methods.

- **Attributes**

  An attribute describes a specific property of a business object. For example, *Employee.Name* is an attribute of the *Employee* object type.

- **Events**

  An event signals that the status of a business object has changed.

Inheritance and Polymorphism

One objective and the main benefit of object-oriented technology is software reusability.

The reuse of software code is achieved by deriving new object types from existing ones. When an object type is generated from an existing object type, the new object type is called the subtype and the existing object type is called the supertype. For example, the object type *Employee* is a subtype which has been derived from the supertype *Person*. The synonyms subclass and superclass are sometimes used for subtype and supertype.

A subtype **inherits** all the properties and methods defined for the supertype from which it descends, but can also have additional properties and methods. A subtype may also implement a different behavior for the methods inherited from the supertype. **Polymorphism** is the term used to describe when the same method triggers different types of behavior in different business object types.

For more information about creating business objects refer to the documentation on SAP Business Workflow [Ext.].

Business Application Programming Interface (BAPI)

Use

The SAP business objects held in the Business Object Repository (BOR) encapsulate their data and processes. External access to the data and processes is only possible using specific methods - namely **BAPIs** (Business Application Program Interfaces).

A BAPI is defined as a method of an SAP Business Object Type [Page 13] or of an SAP Interface Type [Page 22].

The functionality implemented in the SAP business object type *CostCenter* includes for example, listing all the available cost centers. The business object type *CostCenter* provides the BAPI *GetList()* for this purpose.

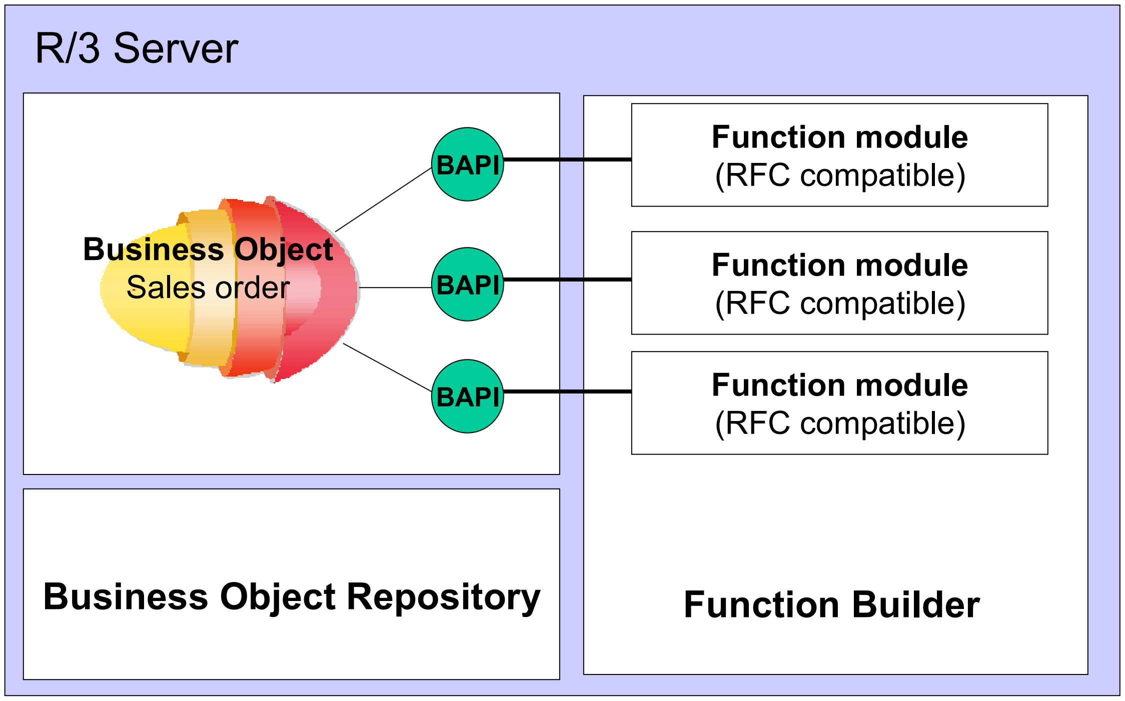
The BAPIs in the R/3 System are currently implemented as function modules which are created and managed in the Function Builder. Each function module underlying a BAPI:

- Supports the Remote Function Call (RFC) protocol
- Has been assigned as a method to an SAP business object type in the Business Object Repository
- Is processed without returning any screen dialogs to the calling application

Integration

The graphic below shows the relationship between an SAP business object type, its associated BAPIs and its function modules.

Business Object Type with BAPIs and Associated Function Modules


R/3 Server
Business Object
Sales order
BAPI
BAPI
BAPI
Function module (RFC compatible)
Function module (RFC compatible)
Function module (RFC compatible)
Business Object Repository
Function Builder

This architecture enables SAP to change the details of a BAPI's implementation without affecting external applications that use the BAPI. For information on SAP policy for enhancing BAPIs see SAP Enhancements to BAPIs [Page 57].

Prerequisites

To use a BAPI method to access data in SAP business objects, an application program only needs to know how to call the method. The information required is:

- The name of the BAPI
- Details of the BAPI interface:
- Import parameters, which contain data to be transferred from the calling program to the BAPI
- Export parameters, which contain data to be transferred from the BAPI back to the calling program
- Import/export (table) parameters for both importing and exporting data

Application programmers can work with SAP business objects and implement their BAPIs without needing to know the underlying implementation and coding details.

Advantages of BAPIs

Use

The section below outlines some of the benefits of using BAPIs to access SAP business objects.

Business Standard

SAP business object types and their BAPIs are the standard for the business contents in the R/3 System. They enable business functions in the R/3 System to be integrated with business functions in non-SAP software.

Conforming to Standards

The development of BAPIs is a part of a shared initiative between SAP, customers, partners and leading standards organizations. BAPIs have become a communication standard between business systems.

SAP business object types and their BAPIs can be accessed through object-oriented interface technologies, such as Microsoft COM/DCOM (Component Object Model/Distributed Component Object Model).

SAP business objects conform to the specifications laid down by the Open Applications Group (OAG) and, in cooperation with ObjectBridge from VisualEdge, meet the CORBA guidelines (Common Object Request Broker Architecture) established by the Object Management Group.

Stability and Downward Compatibility

Once a BAPI has been developed and released by SAP, its interface definition and parameters remain stable in the long term. This ensures that an application program is not affected by changes in the underlying R/3 software and data.

If BAPIs need to be enhanced, for example by adding optional parameters, SAP can do this without affecting the stability of existing applications. At the same time new applications can benefit from the enhanced functions.

For information on SAP release strategy for BAPIs, see SAP Enhancements to BAPIs [Page 57].

Object-Oriented Technology

As methods of SAP business objects BAPIs can offer access to R/3 data and processes. They follow an object-oriented programming model. BAPIs can be called using object-oriented interface technologies such as COM/DCOM. In this way they enable free interaction between SAP and non-SAP software components.

Openness

Using BAPIs, you can access all development platforms which support the SAP Protocol Remote Function Call (RFC).

Characteristics of BAPIs

Use

This section tells you what you should know about BAPIs before you start integrating BAPI calls into your application programs.

Naming Conventions

BAPIs are identified by the name of the business object type followed by the name of the BAPI. (BAPI names are in English and describe what the BAPI does to the business object). A period separates the two parts of the name.

For example, the full name of the BAPI *CheckAvailability()* of the business object *Material* is *Material.CheckAvailability().*

Standardized BAPIs and Parameters

There are some standardized BAPIs that can be implemented for most business object types. These BAPIs have specific functions, for example, the BAPI *GetList(),* retrieves a list of the instances of a business object type.

These BAPIs are implemented according to specified rules and have the same name for all business objects. The BAPI *GetList()* can be used for many business objects, for example, *CompanyCode.GetList()* and *Material.GetList()*. For information about these BAPIs see Standardized BAPIs [Page 43].

As far as possible, the parameters of BAPIs are also standardized. Standardized Parameters [Page 46] that can be implemented and defined for BAPIs are used in the BAPI interface.

Database Consistency

Each BAPI that creates an instance of an object or updates the data of an object is responsible for database consistency. All database changes are carried out completely or not at all.

However, the COMMIT WORK command must not be executed by the BAPI itself; it must be executed by the calling program. For further information see the Transaction Model for Developing BAPIs [Page 51].

No Dialog Orientation

BAPIs do not return any screen dialogs from the R/3 server system to the calling application.

Authorization

To interact with the R/3 System users must have a certain set of authorizations. To execute a BAPI as part of your application program, the users of your application must have the appropriate authorizations set up in their R/3 master records. The documentation provided with the BAPI contains information about the required authorizations.

Any attempts to execute a BAPI that fail as a result of insufficient authorizations are reported back to the calling application.

Data Types and Data Display

BAPIs are programming interfaces for business applications. Neutral (internal) data types and data formats are therefore used almost exclusively in BAPI interfaces. There are some exceptions, for example:

- Currency codes
- Internal keys
- Quantities
- Specific fields in address structures

For information about these exceptions see Internal and External Data Formats [Ext.] in the BAPI Programming Guide.

Conversion BAPIs are provided for the appropriate conversions between internal and external formats. For information see Service BAPIs for Help Functions [Page 48].

BAPI parameters can use most of the data types supported by SAP. The data types are documented in the structure entries of each parameter in the ABAP Dictionary.

Processing Modes

When BAPIs are called directly, for example, via BAPI ActiveX Control, the R/3 DCOM Component Connector or by directly calling the underlying function module, BAPIs are processed in synchronous mode.

For information about calling BAPIs in a distributed environment (ALE) see Programming Distributed Applications [Page 41].

Business Object Attributes

The attributes of business objects are accessed through the BAPI interface itself. An example of a BAPI that retrieves such attributes is *GetDetail()* of the business object *CompanyCode*.

BAPIs of SAP Interface Types

Use

As of Release 4.5A, BAPIs can also describe interfaces implemented outside the R/3 System, which can be called in external systems by R/3 Systems. These BAPIs are known as **BAPIs used for outbound processing**. The target system is determined for the BAPI call in the distribution model of Application Link Enabling (ALE).

BAPIs used for outbound processing are defined in the Business Object Repository (BOR) as API methods of SAP interface types. In contrast to BAPIs of SAP business object types, the BAPIs in the R/3 System are not implemented as function modules.

Functions implemented outside the R/3 System can be standardized and made available as BAPIs.

With the exception of the above mentioned points, the information in the BAPI User Guide and the BAPI Programming Guide [Ext.] refers to the BAPIs of both SAP interface types and SAP business object types. If this is not the case, the documentation will explain what the differences are.

Business Object Repository

Definition

All SAP business object types and SAP interface types and their methods are defined and described in the R/3 Business Object Repository (BOR). The Business Object Repository was introduced in R/3 Release 3.0, at the same as time as SAP business objects and SAP Business Workflow. At first, the BOR was mainly used by SAP Business Workflow.

Use

With the introduction of BAPIs in R/3 Release 3.1, the BOR assumed an important new role - it is now the central access point for external applications to access SAP business object types, SAP interface types and their BAPIs.

In this context, the BOR serves two essential purposes:

- It defines and describes SAP business objects and SAP interface types and their BAPIs.

  If you are developing an application program, you retrieve details of the SAP business object types or SAP interface types, their key fields and their BAPI methods from the BOR. The BOR contains all the information you need to include the appropriate object type definitions and BAPI invocations in your application program.

- It 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.

For further information about creating business object types refer to the documentation on SAP Business Workflow [Ext.].

Programming with BAPIs

Use

Assume that the application you developed is to use the data held in business objects in the R/3 System and that this data is to be accessed by calling BAPIs. Your application can be as simple or as complex as you like and can include more than one BAPI call.

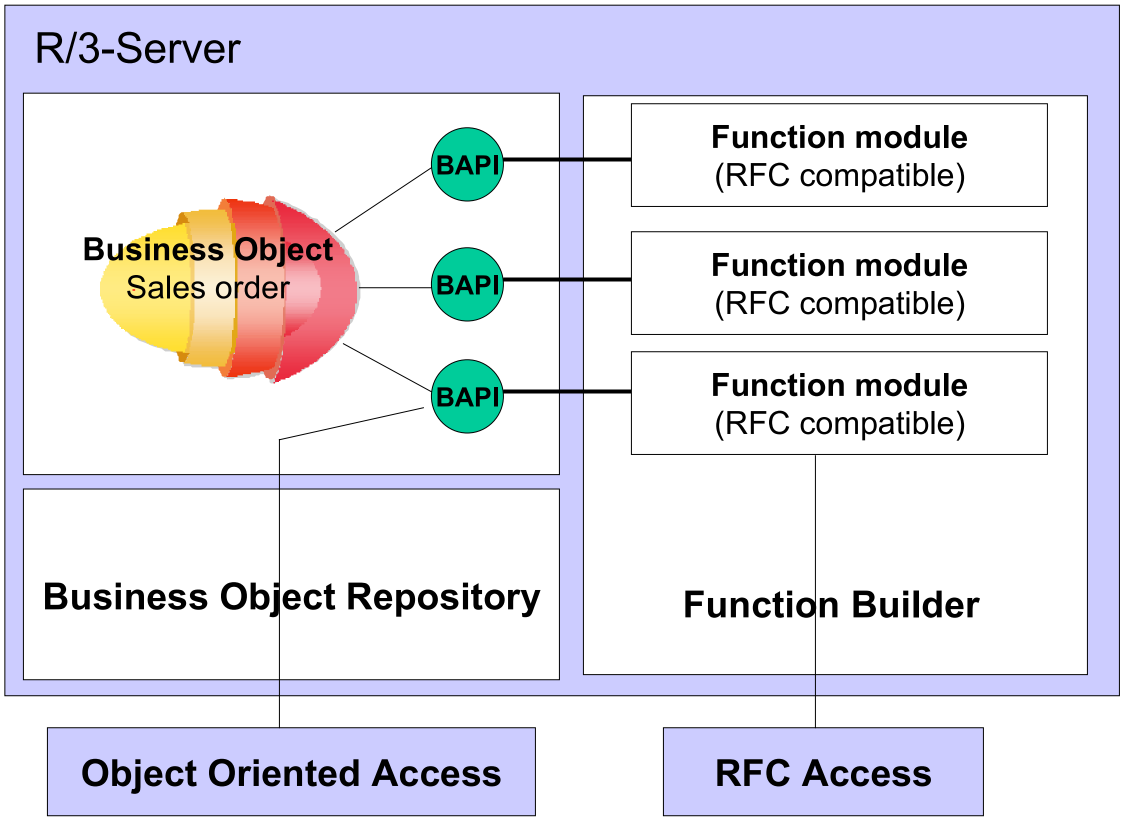
Integration

BAPIs are defined in the Business Object Repository (BOR) as methods of SAP business object types or SAP interface types and are implemented as function modules. As the BAPI definition is separation from its actual implementation, you can access a BAPI in two ways:

- You can call the BAPI in the BOR using object-oriented method calls (see Object-oriented Access to BAPIs [Page 35]).
- You can make RFC calls to the function module on which the BAPI is based (see Function-oriented Access to BAPIs [Page 38]).

The two approaches are illustrated in the graphic below.

Ways of Accessing BAPIs


R/3-Server
Business Object
Sales order
BAPI
BAPI
BAPI
Function module (RFC compatible)
Function module (RFC compatible)
Function module (RFC compatible)
Business Object Repository
Function Builder
Object Oriented Access
RFC Access

Features

Applications that access BAPIs can be broadly divided into two categories:

- Dialog applications

  With dialog applications the caller usually waits until the method call has been processed, for example, waits for the results of a *GetList()* BAPI.

  For more information see Programming Dialog Applications [Page 39].

- Distributed applications

  BAPIs can also be used in Application Link Enabling (ALE) to exchange and replicate data between two distributed systems. For more information see Distributed Applications Programming [Page 41].

Activities

Regardless of the approach you use, you need to complete the following tasks in order to invoke BAPIs from your application program:

Task	For Further Information:
1. Identify the SAP business object type or the SAP interface type and the required BAPI and determine the parameter information for the BAPI interface.	Determining BAPI Details [Page 26]
2. Include the BAPI call or function call and the parameter declarations in your application program.	Programming Dialog Applications [Page 39] or Programming Distributed Applications [Page 41] For examples of calling BAPIs from various development platforms see Examples of BAPI Calls [Page 64].

Determining Details of BAPIs

Purpose

To use a BAPI, you have to determine the information you have to forward to call the BAPI or function in your application program.

Prerequisites

You can find this information in the BAPI Explorer or in the BAPI Browser.

The **BAPI Explorer** is available as of Release 4.6A and replaces the BAPI Browser used in earlier releases. The BAPI Explorer uses up-to-date HTML control technology which requires SAPgui version 4.6A.

If you are using a SAPgui from an earlier release, the BAPI Browser is displayed instead of the BAPI Explorer.

Details of BAPIs of Business Object Types

The following information is required to use BAPIs of SAP business object types:

- The import, export and import/export parameters of the BAPI

  ➡

  Standardized parameters supply the same or equivalent data and can therefore be used in different BAPIs. These parameters have the same name in all BAPIs and are implemented using the same guidelines. For further information see Standardized Parameters [Page 46].

- They key fields of the SAP business object type

  The key fields of SAP business object types specify the identification structure with which client programs can uniquely access a specific instance of the object type. The key fields are required if the BAPI is using specific instances of an object type. BAPIs are divided into these categories:

– Instance-dependent BAPIs

  Instance-dependent BAPIs use specific instances of an object type which the client application must specify in the key fields of the business object. An example of an instance-dependent BAPI is *CompanyCode.GetDetail()* and when used the client application must specify the company code ID, for example, *GetSAPObject("CompanyCode","0001").*

– Instance-independent BAPIs

  Instance-independent BAPIs do not use specific instances of an object type. They usually return a list of object type instances in tables to the calling program. Instance-independent BAPIs are also called class methods. The BAPI *CompanyCode.GetList(),* which returns a list of company codes, is an example of an instance-independent BAPI.

  Some instance-independent BAPIs generate object instances and report back information on the generated object instance to the calling application. An example of this type of BAPI is *SalesOrder.CreateFromData()*, which creates a customer order and reports back information to the calling program.

- The documentation on the business object type, the key fields, the BAPI and its parameters.
- The name of the function module, if the RFC calls are sent to the function module underlying the BAPI.

Once you have retrieved all the required information, you can use the BAPI in your application program.

Details of BAPIs of Interface Types

The following information is required to use BAPIs of SAP business object types:

- The import, export and import/export parameters of the BAPI
- The name of the function module that is to implement the interface
- The documentation on the BAPI and its parameters.

Once you have retrieved all the necessary information, you can implement the interface and use the BAPI in your application program.

Process Flow

From the initial R/3 screen, choose *Tools* → *Business Framework* → *BAPI Explorer* or enter the transaction code `BAPI` in the SAPgui command field.

If you are using a SAPgui of Release 4.6A or later, the BAPI Explorer will be displayed. For more information about using the BAPI Explorer see BAPI Explorer [Page 28].

If you are using a SAPgui in a release earlier than 4.6A, the BAPI Browser appears instead of the BAPI Explorer. For further information see BAPI Browser [Page 33].

BAPI Explorer

Definition

The BAPI Explorer is the R/3 System working environment for developing BAPIs themselves and for developing with BAPIs.

Use

Prerequisites

The BAPI Explorer is available as of Release 4.6A and enhances or replaces the BAPI Browser used in earlier releases. The BAPI Explorer uses the latest HTML control technology. For this reason to work with the BAPI Explorer, you must be using a SAPgui of version 4.6A.

If you are using a SAPgui from an earlier release, the BAPI Browser (Transaction BAPI45) is automatically called instead of the BAPI Explorer. For information about working in the BAPI Browser see BAPI Browser [Page 33] in the BAPI User Guide.

Purpose

In 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 obect or interface type in the component hierarchy in the BOR, or from an alphabetical list. All information required to use a particular BAPI is provided in the BAPI Explorer.

The BAPI Explorer is used internally in SAP to develop BAPIs, but can also be used by customers and partners. The BAPI Explorer provides all the tools used to create BAPIs and the required development objects in an integrated programming environment. The entire development process of a BAPI takes place in the framework of form-controlled "projects" to achieve maximum quality, stability and usability of the BAPI.

Structure

The BAPI Explorer is divided into two areas:

- Hierarchy display

  Here all the business object types or interface types for which BAPIs have been defined are displayed. For further information see Working with Display Functions [Page 29].

- Work area

  Here the details and documentation of the development object selected in the hierarchy display can be viewed.

  The tools used to develop BAPIs are also available in the work area. For further information see Working with Tools and Projects [Page 31].

Integration

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

Working with Display Functions

Use

In the hierarchy display of the BAPI Explorer all the business object types or interface types for which BAPIs have been defined are displayed.

Using the tabs *Alphabetical* and *Hierarchical* in the hierarchy display, you can select whether the business object types or interface types are displayed 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 enable 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 are displayed.

  In most cases you can double-click on a field in the *Detail* display to get to the developent environmnt 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 will be displayed in the Function Builder.

- Displaying and Printing Documentation

  In the *Documentation* view of the work area the documentation for the development object selected in the hierarchy is displayed.

  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, e.g. BusinessProcess
- Object type (technical object name) e.g. SAP0001
- Object description, e.g. Plan*
- Method name, e.g. GetList
- Method description, e.g. object list*

Working with Tools and Projects

Use

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

Features

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 below:

- Implementing new BAPIs (for internal SAP use, BAPI developments must be carried out and documented using a project in the BAPI Explorer)
- Changing released BAPIs (only for SAP internal use)
- Requesting a new business object type (only for SAP internal use)

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.

Finding Information in the BAPI Explorer

To display details of BAPIs in the BAPI Explorer [Page 28]:

1. Choose *Tools* → *Business Framework* → *BAPI Explorer* or enter the transaction code `BAPI` in the SAPgui command field.

   If you are using a SAP GUI in a release earlier than 4.6A, the BAPI Browser [Page 33] appears instead of the BAPI Explorer.

   By default only the released BAPIs of business objects are displayed in the BAPI Explorer hierarchy. To display all the BAPIs contained in the BOR, that is, BAPIs of SAP interface types and BAPIs that have not yet been released, choose *Goto* → *BAPIs to display.*

   To display a list of the symbols used in the hierarchy choose *Goto* → *Display Legend.*

2. To display the required view of the object types or interface types select either *Alphabetical* or *Hierarchical.*

   The component hierarchy may have many layers which can make it difficult to find a particualr business object type and its BAPIs. You can restrict the hierarchy display to two levels, by choosing *Goto* → *Change hierarchy level.*

3. Navigate to the required business object type or interface type or use the standard search functions *Find* and *Find next* under the *Edit* menu path (you can use placeholders such as "*").

4. Expand the node of the business object type/interface type to display its key fields and methods. Expand the node of the method to display its parameters.

5. From the hierarchy display select the development object you require details for.

6. To display the technical information select *Detail* in the right hand frame. To read the documentation, select *Documentation.*

Finding Information in the BAPI Browser

If you are using a SAPgui from a release before 4.6A, the BAPI Browser is displayed instead of the BAPI Explorer [Page 28].

To display a description of individual symbols in the BAPI Browser choose *Utilities → Legend.*

BAPIs of SAP interface types cannot be displayed in the BAPI Browser. You have to use the Business Object Browser, as described in Determining Details of BAPIs of Interface Types [Page 34].

In the application hierarchy of the BAPI Browser you can display all the SAP business object types for which BAPIs have been implemented:

1. Choose *Tools → Business Framework → BAPI Browser* or enter the transaction code `BAPI45` in the SAPgui command field. The *BAPI Browser* of the BOR appears and all business object types with BAPIs are displayed in the R/3 application hierarchy.
2. Expand the nodes and the subordinate nodes of one of the application components until you get to the level where the SAP business object types are assigned. Only those business object types for which BAPIs have been implemented are displayed.
3. To open an object type, double-click on it.

   The nodes *Key fields* and *Methods* are displayed.
4. To display a list of the key fields of the object, expand the node *Key fields*. Select the information icon to display descriptions of individual key fields.
5. To display a list of the BAPIs available for this SAP business object type, expand the node *Methods*.

   To display details of individual BAPIs and the name of the function module that implements the BAPI, select the information icon. The name of the function module is given at the start of the documentation. The names of BAPI function modules always begin with “BAPI”.
6. To display the parameters of a BAPI, expand the node for the BAPI and then the *Parameters* node.

   A list of all the BAPI’s parameters is displayed. Select the relevant icon in the BAPI Browser to display further information about the individual parameters.

Determining Details of BAPIs of Interface Types

SAP interface types and their BAPIs are managed in the Business Object Repository (BOR) in a structure based on the R/3 Application Component hierarchy. In **the Business Object Builder y**ou can display all the SAP interface types in the component hierarchy .

1. Choose *Tools* → *Business Framework* → *BAPI Development* → *Business Object Builder* or enter the transaction code `SWO1` in the SAPgui command field.
2. Select *Business Object Repository*.
3. In the next dialog box select the filter *Other settings*. In the field *Type* select the setting *Interface* and deactivate the setting *Object,* in the field *Type* select the setting *Others* and mark all the settings in the field *Status.*

   The *Business Object Repository Browser* is displayed in which all the SAP Interface Types with BAPIs are displayed in the R/3 application hierarchy.
4. Expand the nodes and the subordinate nodes of one of the application components until you get to the level where the SAP interface types are assigned.
5. To open an SAP interface type, double-click on it. The nodes *Interfaces, Attributes, Methods* and *Events* are displayed.
6. Expand the node *Methods* to display a list of the methods available for the selected SAP interface type. BAPIs are marked by a green circle next to the method name.
7. To display the parameters, select the BAPI and then the button *Parameters.*
8. To display the documentation, select the BAPI and then choose *Goto* → *Documentation.*

Object-Oriented Access to BAPIs

Purpose

Object-oriented access to BAPIs in the BOR is possible from a number of platforms.

Development Tool /Middleware	**Further Information**
BAPI ActiveX Control (Windows 95 and Windows NT)	SAP's BAPI ActiveX Control allows external client applications to access SAP business objects in the BOR by invoking BAPIs through OLE Automation. For more information see BAPI ActiveX Control [Ext.].
BAPI C++ Class Library in SAP Assistant	A BAPI C++ Class Library is available in the SAP Assistant providing a C++ proxy class for each SAP business object and its BAPIs. For further information see The C++ BAPI Proxy Classes [Ext.] in the SAP Assistant documentation.
BAPI Java Class Library in SAP Assistant	A dynamic BAPI Java Class Library is available in SAP Assistant enabling object-oriented access to BAPIs from Java platforms. To call BAPIs, the standard data types and classes in Java can be used instead of the SAP internal data types and structures. For further information see The Java BAPI Proxy Classes [Ext.] in the SAP Assistant documentation.
R/3 DCOM Component Connector (Windows NT)	The R/3 DCOM Component Connector developed jointly by SAP and Microsoft enables seamless integration of R/3 objects and COM objects. SAP business objects and their BAPIs can be accessed transparently from COM-based development tools. The other way round COM components can be accessed from the SAP development environment. The R/3 DCOM Component Connector Software Development Kit (SDK) is shipped as part of RFC SDK for Windows NT. For more information see The DCOM Connector [Ext.].
ObjectBridge from Visual Edge	Visual Edge's middleware product "ObjectBridge" enables automatic access to BAPIs from all CORBA 2.0-enabled Object Request Brokers (ORBs) as well as from other object-oriented protocols such as ActiveX/DCOM. For further details see the ObjectBridge product information.
Programming Environment	**Further Information**
Delphi/Connect for SAP from Inprise	For further details see Delphi/Connect product information.
Access Builder for SAP R/3 from IBM	Access Builder for SAP R/3 is used to integrate BAPIs into IBM's development environment, Visual Age for Java. Access Builder for SAP R/3 enables the development of BAPI-enables Java applications, Java applets and JavaBeans for the R/3 System. For further details see the product information for Access Builder for SAP R/3.

Visual Studio (Enterprise Edition) from Microsoft	From now on BAPIs implemented in R/3 Releases 3.1H and 4.0A will be available locally from Microsoft's development environment Visual Studio (Enterprise Edition).

Process Flow

Applications run differently on different development environments and the steps required to start them are also different.

The example below shows the steps involved when the BAPI ActiveX Control is used to access BAPIs. When the R/3 System is connected to, the client application accesses the SAP business objects at runtime by forwarding the OLE automation requests to the BAPI ActiveX Control.

The steps are:

- Create a BAPI ActiveX Control object

  ```
  Set oBAPICtrl = CreateObject("SAP.BAPI.1")
  ```

- Create a logon control object:

  ```
  Set oLogonCtrl = CreateObject("SAP.Logoncontrol.1")
  ```

- Create a connection object to the R/3 System:

  ```
  Set oBAPICtrl.Connection = oLogonCtrl.NewConnection
  ```

- Log on to R/3 System by calling the logon method of the connection object:

  ```
  If oBAPICtrl.Connection.Logon(frmStart.hwnd,FALSE)  = FALSE then
    MsgBox"R/3 Connection failed"
    End
  Endif
  ```

- Request the creation of a local instance of the SAP Business Object

  Before your application can invoke a BAPI on an SAP Business Object, it must first request the creation of an instance of the object.

  The command below taken from a Visual Basic program uses the BAPI ActiveX Control object and the *GetSAPObject()* method to request the creation of an instance of the business object *SalesOrder.*

  ```
  Set boOrder = oBAPICtrl.GetSAPObject("SalesOrder")
  ```

- Create the parameter objects

  ```
  Set oOrderHeader = oBAPICtrl.DimAs(boOrder, _
                  "CreateFromDat1", "OrderHeaderIn")

  Set otabItems = oBAPICtrl.DimAs(boOrder, _
                  "CreateFromDat1", "OrderItemsIn")

  Set otabPartners = oBAPICtrl.DimAs(boOrder, _
                  "CreateFromDat1", "OrderPartners")
  ```

  Then the relevant data must be entered in the parameter objects.

- Call the BAPIs of the business objects

  Once the object instance is created, the BAPIs can be invoked on it.

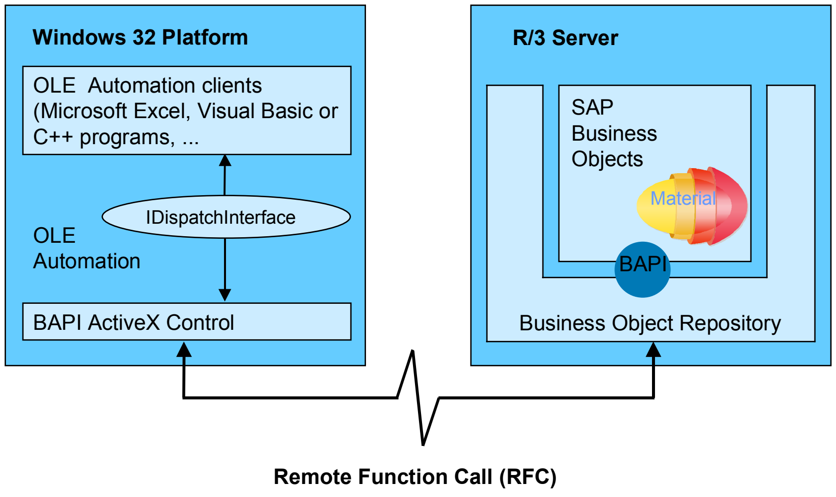
  The Visual Basic command below calls a BAPI:

```
boOrder.CreateFromDat1 OrderHeaderIn:=oOrderHeader, _
                OrderPartners:=otabPartners, _
                OrderItemsIn:=otabItems, _
                Return:=oReturn
```

- Release the SAP business object and the BAPI ActiveX Control object

The graphic below illustrates how SAP business objects and their BAPIs are accessed via BAPI ActiveX Control.

Using BAPI ActiveX Control


Windows 32 Platform
OLE Automation clients (Microsoft Excel, Visual Basic or C++ programs, ...
IDispatchInterface
OLE Automation
BAPI ActiveX Control
R/3 Server
SAP Business Objects
Material
BAPI
Business Object Repository
Remote Function Call (RFC)

Function-Oriented Access to BAPIs

Purpose

You can access BAPIs from platforms that do not provide direct object-oriented access to SAP business objects by making Remote Function Calls (RFCs) to the function module underlying the BAPI.

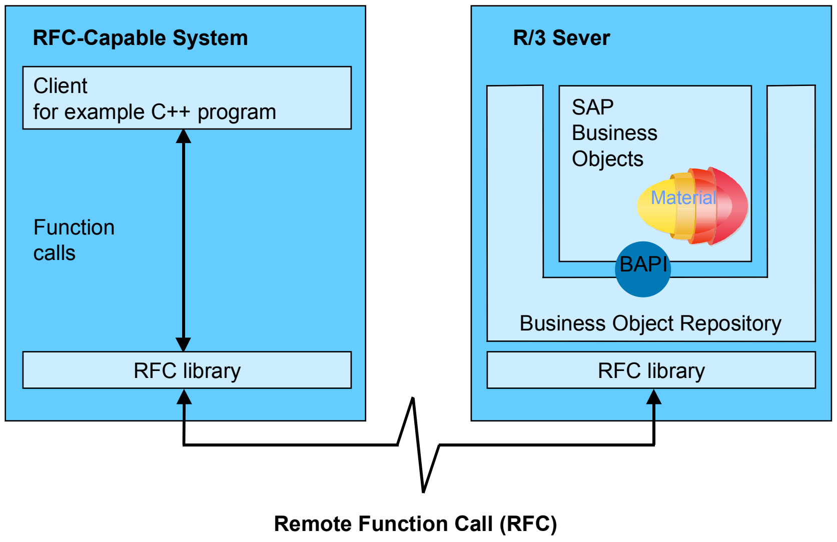
This approach can be used on all development platforms supporting the RFC protocol, for example, ABAP or external platforms using C/C++ Class Library.

Process Flow

During runtime your application program uses the RFC Library or the C/C++ Class Library to make an RFC call to the function module underlying the BAPI. The Library translates client calls into communication steps in accordance with the RFC protocol and the client requests are forwarded to the relevant function module in the R/3 server system.

This approach is illustrated in the graphic below.

Accessing a BAPI Function Module Through RFC Calls



For further information about RFC calls to function modules see Remote Communications [Ext.].

Programming Dialog Applications

Use

SAP provides various help functions in the form of service BAPIs to support the programming of dialog applications.

Features

In dialog applications you can use the following help functions: The service BAPIs can be found in the BOR component hierarchy under *Basis Components → Middleware.*

Transaction Control

Each dialog transaction that uses BAPIs to change data in one or more objects must directly execute the COMMIT WORK command to save the data. The BAPI *BapiService.TransactionCommit()* is used for this and it writes the changes to the database.

If data is not to be written to the database, the BAPI *BapiService.TransactionRollback()* can reset the changes, provided that the changes have not already been passed to the database with the BAPI. This works provided that the BAPI *BapiService.TransactionCommit* has not already passed the changes to the database.

Input Help (F4 Help)

To display the possible entries (F4 help) for an input field transferred when a BAPI is called, you should include the service BAPI *HelpValues.GetList()* in your program. This BAPI supplies the input values allowed for the field in a BAPI parameter.

The method *HelpValues.GetList()* method uses the help view for the check table, matchcodes or domain fixed values linked to the field in the ABAP Dictionary. For information about check tables, matchcodes and domain fixed values see the ABAP Dictionary [Ext.].

F1 Help (Field Help)

To provide F1 help (field help) for input fields in your application you can use the BAPI *BapiService.FieldHelpGetDocu().* This method reads the documentation (F1 help) on the fields in a BAPI parameter.

Interpreting Return Messages

Each BAPI contains a parameter named *Return*. This parameter reports exception messages or success messages back to the calling program.

Two service BAPIs can diagnose and process error messages from BAPI calls:

- *BapiService.MessageGetDetail()* displays the short and long texts of BAPI error messages.
- *BapiService.ApplicationLogGetDetail()*, displays the entries in application logs.

Further Information

For further information refer to the documentation on the appropriate BAPI in the Business Object Repository.

A list of all the service BAPIs available can be found under Service BAPIs for Help Functions [Page 48].

Programming Distributed Applications

Use

From R/3 Release 4.0 onwards, BAPIs can also be used in Application Link Enabling (ALE) to exchange and replicate data between two distributed systems. When data is transferred between distributed systems you decide between using synchronous and asynchronous BAPIs.

Features

Synchronous BAPIs

Synchronous BAPIs are generally used to read data from a remote logical system, for example to display a customer list.

Before calling the BAPI, it is therefore necessary to determine the RFC destination of the remote system or the BAPI server. The application developer has access to an API for determining the RFC destination of the BAPI.

Asynchronous BAPIs

Asynchronous BAPIs are generally used for replicating data on one or more logical systems, for example to distribute article master data.

To implement an asynchronous data transfer with a BAPI, an ALE IDoc interface must be generated. This interface controls all of the data communication between logical systems using IDoc technology. The application then calls the generated ALE IDoc interface locally instead of the BAPIs.

As with synchronous BAPIs, the corresponding logical systems of the BAPI must be determined before you call the ALE-IDoc interface. These are transferred to the ALE-IDoc interface as input parameters.

Processing the Return Parameter

Each BAPI has the standardized *Return* parameter for returning messages to the calling application. For more information about this parameter, see Return Parameters (Error Handling) [Ext.] in the BAPI Programming Guide.

First the function module that converts the IDoc into the corresponding BAPI in the receiving system is called. After this status records are written for the IDoc in which messages sent in the Return parameter are logged.

If the field *type* contains A (abort) in at least one of the transferred return parameter entries, the status 51 (error, application document has not been posted) is assigned to all the status records of the IDoc and a ROLLBACK WORK is executed. If the field *type* contains E (error) in at least one of the transferred Return parameter entries, the status 51 (error, application document has not been posted) is assigned to all the status records of the IDoc and a COMMIT WORK is executed anyway. Otherwise status 53 (application document posted) is written and a COMMIT WORK is executed.

ALE Distribution Model

As with the message types, synchronous and asynchronous BAPIs are maintained in the ALE context in the ALE distribution model. Based on the distribution model, the logical systems (for asynchronous BAPIs) or RFC destinations (for synchronous BAPIs) are determined at runtime.

For more detailed information about using BAPIs in an ALE context, see ALE Programming [Ext.] and the document Using ALE Services [Ext.].

Standardized BAPIs

Use

Some BAPIs and methods provide basic functions and can be used for most SAP business object types. Such BAPIs are known as “standardized” BAPIs.

Features

With object methods and especially with BAPIs, you can differentiate between **instance methods** and **class methods**. Instance methods refer to precisely one instance (one specific occurrence) of an SAP Business Object type, for example, to one explicitly specified customer order. Whereas class methods are instance-independent.

BAPIs for Reading Data

The following BAPIs provide you with read-only access to data in the associated business object type:

GetList()

With the BAPI *GetList()* you can select a range of object key values, for example, company codes and material numbers. The BAPIs *GetList()* is a class method.

For more information see Programming GetList() BAPIs [Ext.].

GetDetail()

The BAPI *GetDetail()* uses a key to retrieve details about an instance (one specific occurrence) of a business object type and returns this data to the calling program. Then this information is reported back to the calling program. The BAPI *GetDetail()* is an instance method.

For more information see Programming GetDetail() BAPIs [Ext.].

GetStatus()

The BAPI *GetStatus()* is used to query the status of an SAP business object instance, for example, the processing status of a sales order. This BAPI is used only for displaying the status of an object and does not retrieve full details like the BAPI *GetDetail()*. The BAPI *GetStatus()* is an instance method.

For more information see Programming GetStatus() BAPIs [Ext.].

ExistenceCheck()

The BAPI *ExistenceCheck()* checks whether an entry exists for a business object instance, for example, whether the customer master has been created. The *ExistenceCheck()* BAPI is an instance method.

For more information see Programming ExistenceCheck() BAPIs [Ext.].

BAPIs for Creating or Changing Data

The following BAPIs can create, change or delete instances of a business object type:

Standardized BAPIs

Create() or *CreateFromData()*

The BAPI *Create()* or *CreateFromData()* creates an instance of an SAP business object type, for example, a purchase order. These BAPIs are class methods.

For more information see Programming Create() BAPIs [Ext.].

Change()

The BAPI *Change()* changes an existing instance of a SAP business object type, for example, a purchase order. The *()* BAPI is an instance method.

For more information see Programming Change() BAPIs [Ext.].

Delete() and *Undelete()*

The BAPI *Delete()* deletes an instance of an SAP business object type from the database, for example, a purchase order.

The BAPI *Undelete()* removes a deletion flag.

These BAPIs are instance methods.

For more information see Programming Delete() BAPIs [Ext.].

Cancel()

Unlike the BAPI *Delete ()* the BAPI *Cancel()* cancels an instance of a business object, that is the instance to be cancelled remains in the database and an additional instance is created that is canceled).

The *Cancel()* BAPI is an instance method.

For more information see Programming Cancel() BAPIs [Ext.].

Add<subobject> and *Remove<sub-object>*

The BAPI *Add<sub-object>* adds a sub-object to an existing object instance and the BAPI and *Remove<sub-object>* removes a sub-object from an object instance. These BAPIs are instance methods.

For further information see Programming Methods for Sub-Objects [Ext.].

If you are implementing BAPIs that create or change data you should consider using buffering. For further information see Buffering for Write BAPIs [Ext.].

BAPIs for Mass Processing

The BAPIs listed in the above section, “BAPIs for Creating or Changing Data”, can also be used for mass processing. Here, when a BAPI is called, several business object instances are processed at the same time.

With BAPIs for mass processing, the suffix “Multiple” is added to the method name, for example, *ChangeMultiple(), CreateMultiple(), DeleteMultiple()*. The BAPIs here are always class methods.

We strongly recommend that you create instance-dependent BAPIs with buffering instead of Multiple() BAPIs. For further information see Buffering with Write BAPIs [Ext.].

BAPIs for Replicating Business Object Instances

The following BAPIs are used for replicating business object instances:

Replicate() and *SaveReplica()*

The BAPIs *Replicate()* and *SaveReplica()* are implemented as methods of replicable business object types(). They enable specific instances of an object type to be copied to one or more different systems. These BAPIs are used mainly to transfer data between distributed systems within the context of Application Link Enabling (ALE). These BAPIs are class methods.

For more information see Programming Replicate()/SaveReplica() BAPIs [Ext.].

Standardized Parameters

Use

There are some parameters that can be created for various BAPIs because they contain the same or equivalent data in all BAPIs. Such parameters are known as “standardized” parameters. They should be implemented the same in all BAPIs.

Features

Address parameters

Specific reference structures are defined for address parameters in BAPIs. You should copy these structures to use in your BAPI, especially if the underlying object type uses the central address management (CAM).

For more information see Address Parameters [Ext.].

Change Parameters

In BAPIs that cause database changes (for example, Change() and Create() BAPIs) you must be able to distinguish between parameter fields that contain modified values and parameter fields that have not been modified. This distinction is made through the use of standardized parameters.

For more information see Change Parameters [Ext.].

Extension parameters

The parameters *ExtensionIn* and *ExtensionOut* provides customers with a mechanism that enables BAPIs to be enhanced without modifications. For further information see Customer Enhancement Concept for BAPIs [Ext.].

Return Parameters

Each BAPI must have an export *Return* parameter for returning messages to the calling application. To provide application programmers with a consistent error handling process for BAPI calls, all Return parameters must be implemented in the same, standardized way.

For further information see Return Parameters (Error Handling) [Ext.].

Selection Parameters

The parameters in BAPIs used to search for specific instances of a business object type, for example, BAPI *GetList(),* have to enable the caller of the BAPIs to specify appropriate selection criteria. Standardized selection parameters are used to do this.

For more information see Selection Parameters [Ext.].

Test Run Parameters

The parameter *TestRun* is used in the BAPIs that generate instances - *Create()* or *CreateFromData()*, to check the entries for the object instance in the database before actually creating the object instance. The creation of the object instance is only simulated and data is not updated.

For further information see Test Run Parameters [Ext.].

Text Transfer Parameters

To transfer BAPI documentation texts, for example, documentation of a business object type, you have to create standardized text transfer parameters.

For more information see Text Transfer Parameters [Ext.].

Service BAPIs for Help Functions

Use

A number of service BAPIs provide basic help functions. Service BAPIs provide information or services for BAPIs from individual application components.

The service BAPIs can be found in the BOR component hierarchy under *Basis* → *Middleware.*

For details of individual service BAPIs refer to the BAPI documentation in the BOR.

Features

BAPIs for Accessing Interface Documentation

Using the BAPIs below, you can display the documentation and descriptions of BAPI interfaces:

- *HelpValues.GetList()*

  This method determines the input values (F4 help) for a BAPI parameter field. The method displays valid input values for a specific BAPI parameter field, enabling end-users to enter a correct value in the input field.

- *BapiService.FieldHelpGetDocu()*

  This method reads the documentation (F1 help) on the BAPI parameter fields. It allows you to provide end-users with descriptions of BAPI parameter fields.

- *BapiService.InterfaceGetDocu()*

  This method reads the entire interface documentation of a BAPI. You can access documentation on the business object type, method, parameters and parameter fields.

- *BapiService.HyperLinkGetText()*

  This method reads R/3 object documentation that has hyperlinks to a document which was itself created with either the *BapiService.InterfaceGetDocu()* or the *BapiService.HyperLinkGetText()* method.

BAPIs for Interpreting Error Messages

The BAPIs below help you to interpret BAPI error messages:

- *BapiService.MessageGetDetail()*

  This method displays the short and long texts of BAPI error messages.

- *BapiService.ApplicationLogGetDetail()*

  This method reads the entries in the application log.

BAPIs for Controlling COMMIT and ROLLBACK

The transaction model (see Transaction Model for Developing BAPIs [Page 51]) stipulates that every BAPI which creates object instances or changes object data is responsible for database consistency.

The commands used to write changes to the database or to reset changes (COMMIT and ROLLBACK), must not be executed by the BAPI itself, they have to be called directly from the external application program.

External programs can use the following service BAPIs for these calls:

- *BapiService.TransactionCommit()*

  This method executes a COMMIT WORK command. When you call BAPIs in your program that change data in the R/3 System, you must then call this method to pass the changes to the database.

- *BapiService.TransactionRollback()*

  This method executes a ROLLBACK WORK command. When you call BAPIs in your program that change data in the R/3 System, you have to then call this method to forward the changes to the database. This works provided that the BAPI *BapiService.TransactionCommit()* has not already forwarded the changes to the database.

BAPIs for Converting Between Internal and External Data Formats (Domain Conversion)

BAPIs are programming interfaces, not end-user interfaces. For this reason a neutral data format (with some exceptions) must be used in BAPIs. Fields in BAPI parameters are displayed in the BAPI interface in the internal format used in the database, not in a formatted form.

When you call a BAPI in your program, you need to use the external data format, to display data on the screen, for example. You can use the following conversion BAPIs to display the data in the required format:

- BapiService.DataConversionInt2Ext1()

  This BAPI converts data from the internal format into the required external format.

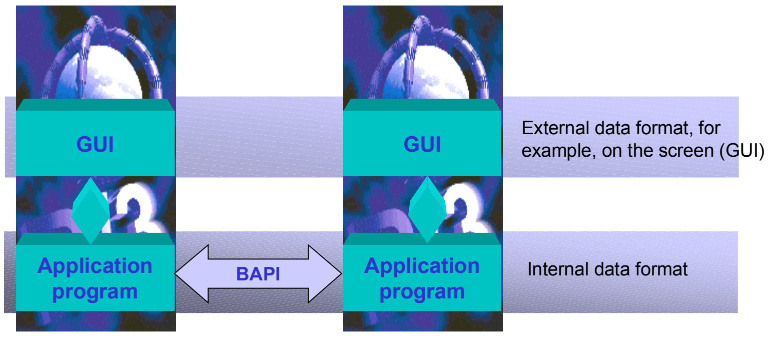
- *BapiService.DataConversionExt2Int1()*

  This BAPI converts data from the external format into the required internal format.

The data can only be converted provided that the conversion routines for the domains underlying the data to be converted are maintained in the ABAP Dictionary. Otherwise, the data is returned unconverted.

The graphic below shows where conversion BAPIs are used:

Conversion BAPIs



Transaction Model for Developing BAPIs

The transaction model in which BAPIs are used determines how you have to program BAPIs.

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 R/3 LUW is all the steps involved in a transaction including updating the database.

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

- **A**tomic

  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.

- **C**onsistent

  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.

- **I**solated

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

- **D**urable

  Changes cannot be reversed and transactions cannot be canceled.

Characteristics

Transactionality

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 in one LUW.
The BAPI transaction model, therefore, implies both that individual BAPIs must be transactional and that several BAPIs combined in 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 him/herself when to execute a COMMIT WORK (or, as the case may be, a ROLLBACK WORK). This means that BAPIs themselves cannot (generally) execute a COMMIT WORK command.

The following restrictions apply to combining several BAPIs in 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.Transaction Commit()* or *BapiService.TransactionRollback()*.

The transaction-controlling BAPIs *BAPIService.TransactionCommit* and *BAPIService.TransactionRollback* are only available as of **Release 4.5**. In Release 4.0, please 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.

Use of the Update Task

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

For an example, see BAPI Transaction Model Without Commit [Page 55]

Comment:

In Release 3.1, the BAPIs themselves executed the COMMIT WORK command, 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.

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 - R/3 Frontend in Note 0131838, "Collective Note for BAPIs with the Commit Work Command".

For an example of the old transaction model, see Old Transaction Model for BAPIs (with Commit) [Page 53].

Example: Old BAPI Transaction Model (with Commit)

Use

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 below (see graphic below):

Log on

(Source code)

Call BAPI to read and/or change data

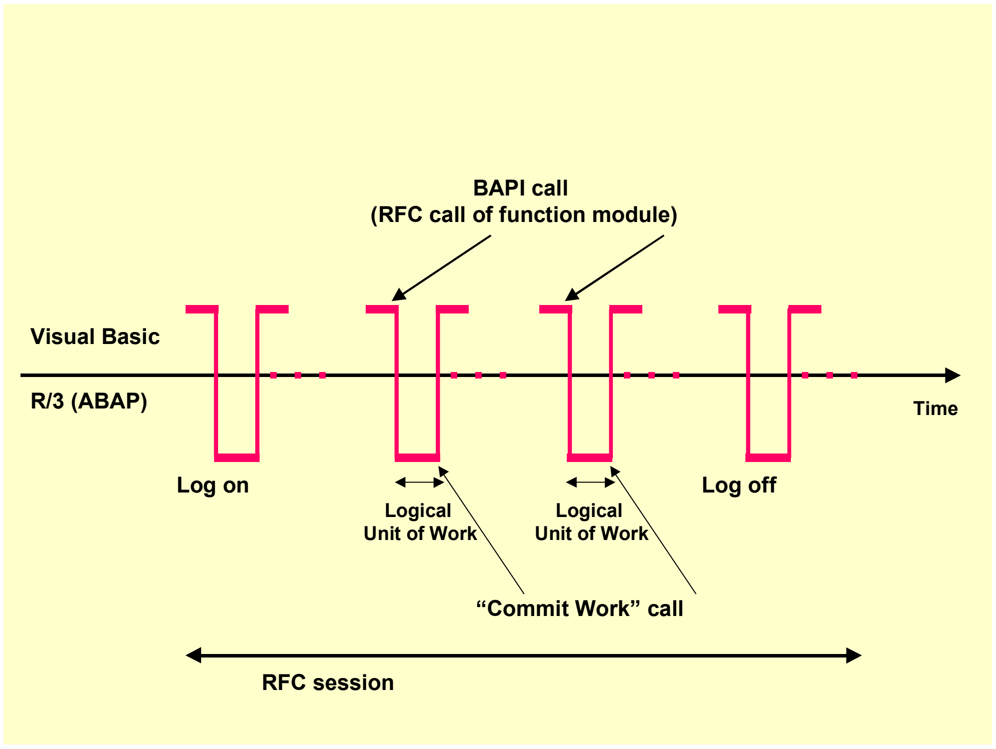
(Source code)

Call BAPI to read and/or change data

(Source code)

Log off

BAPI Transaction Model with Commit



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

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.

Example: BAPI Transaction Model (Without Commit)

Use

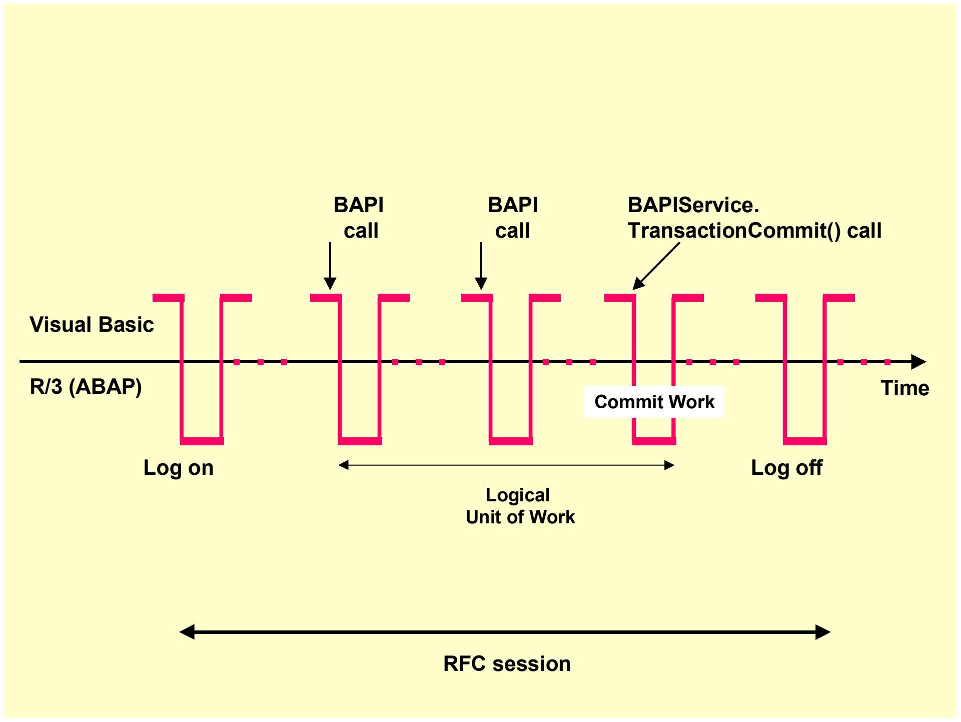
The example below of an external program calling a BAPI to change data in an R/3 System, illustrates how the transaction model affects BAPI development. For example, this could involve a transaction implemented with Visual Basic. Only data from the R/3 System is to be changed.

The RFC connection is live the whole time the external program is logged on to the R/3 System to avoid having to connect and disconnect repeatedly. 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 R/3 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
```

BAPI Transaction Model Without Commit



SAP Enhancements to Released BAPIs

Purpose

Application developers who use BAPIs in their programs must be able to rely on the BAPI interface remaining the same. As a result of this, once a BAPI is released, it must fulfill certain requirements regarding the stability of its interface.

Whenever SAP enhances a BAPI, downward compatibility of syntax and contents must be guaranteed whenever possible. Downward compatibility means that applications that were programmed with BAPIs from a specific R/3 Release will not be affected in later R/3 Releases if the syntax or the content of this BAPI changes.

Examples of *syntax changes* are changes to parameter names, or changes to the type or length of a domain. The ABAP Dictionary can automatically test whether syntax changes are compatible.

Content changes are involved, for example, when existing coding of the BAPI function module is changed or new coding is added. Only the developer can ensure that content changes are downwardly compatible.

Accordingly, when you enhance a BAPI, you can differentiate between a compatible enhancement [Page 59] and an incompatible enhancement [Page 61], depending on whether the downward compatibility of the BAPI can be guaranteed.

To protect the stability of the interface, compatible enhancements are always preferred to incompatible enhancements.

For SAP internal development, each enhancement to a BAPI must be created in a project in the BAPI Explorer.

Integration

Tool Support for Enhancements

Tool support covers the following aspects:

- Changes and version management in the BOR
  Changes made to a BAPI only take effect when the changes are defined in the Business Object Repository (BOR), that is, they have been saved and generated.
  Version management of BAPIs is also carried out in the BOR.
- Checking in the ABAP Dictionary
  Changes to the syntax of BAPIs are automatically checked by the ABAP Dictionary, thereby preventing the BAPI data structure being changed by mistake.
  - The ABAP Dictionary rejects incompatible changes to data elements, domains or structures that are being used by a BAPI that has been released.
  - Compatible changes or changes to data elements, domains or structures of a BAPI that has not been released are accepted by the ABAP Dictionary.

See also:

Compatible Enhancements [Page 59]

Incompatible Enhancements [Page 61]

Compatible Enhancements

Use

Compatible enhancements are interface enhancements that change the BAPI without effecting the downward compatibility of the BAPI. Applications which access the BAPI are not affected by compatible enhancements.

Integration

Compatible enhancements are:

- *New optional parameters*
  A parameter is considered to be optional if it does not have to be included in a BAPI call. A new optional parameter can be added in any place in the interface.

  A new parameter is added to the BAPI Sales*Order.GetList()*, which can be used as an additional selection criteria for selecting purchase orders.

- *New optional fields in structures*
  A field is considered to be optional if it can be left out completely in a BAPI call. The fields must be added to the end of a structure. This is because the function module upon which the BAPI is based is called via RFC. It does not matter how the fields are arranged in a structure or table because the whole structure is always forwarded. It is not first broken up into fields.

  An additional input field for the applicant's educational background is added to the BAPI *Applicant.CreateFromData().*

The table below lists the compatible changes in the function module. We cannot guarantee that this list is exhaustive.

Compatible Changes to Function Modules	
In interface	New optional parameter as a field
	New optional parameter as a structure
	New optional parameter as a table
	Adding new optional field to the structure
	Adding new optional field to the table
	Compatible changes to field types (in the ABAP Dictionary)
	Converting mandatory fields to optional fields
In program code	New additional coding that does not involve changes to the interpretation/processing logic

	Changes to the existing code which do not involve changing the interpretation or processing logic
	Using customer exits

When making changes, be sure to follow the guidelines in the BAPI Programming Guide [Ext.].

Incompatible Enhancements

Purpose

Changes to the contents or functionality of a BAPI often result in the introduction of new parameters without which the interface can no longer function. Often, these changes also cause existing parameters to lose their original meaning. Such modifications are considered to be incompatible enhancements, because they no longer enable the BAPI to be downward compatible.

Syntactically incompatible enhancements are:

- Changes to the field length
- Changes to the field type
- Renaming parameters in the function module or in the method
- Inserting a field within a structure
- Deleting parameters and fields
- Inserting new mandatory parameters and fields
  Parameters can be flagged as mandatory in the BOR. However, this is not the case with fields. Fields can only be categorized as mandatory at a semantic level and not at a technical level. This is why the documentation for each parameter must specify which fields can be filled.

The table below lists the incompatible changes to function modules. We cannot guarantee that this list is exhaustive.

Incompatible Changes to Function Modules	
In interface	New mandatory parameter
	Adding new fields between existing fields in the structure
	Adding new fields between existing fields in the table
	Adding new mandatory fields to the structure
	Adding new mandatory fields to the table
	Incompatible changes to field types (in the ABAP Dictionary)
	Changing optional fields to mandatory fields
	Renaming parameters
In program code	New additional source code that involves changes to the interpretation/processing logic
	Changes to the existing source code that involve changing to the interpretation/processing logic
	Adding or removing COMMIT WORK commands in the program

Process Flow

In cases of incompatible changes to a BAPI, you should work through the following three steps: **Create an additional BAPI, Support and label the expiring BAPI** and **Delete the replaced BAPI.**

Create an Additional BAPI

To ensure that the interface stability of an existing BAPI is not impaired, you must not make any incompatible changes to the existing BAPI. Instead, create one or, if necessary, several additional BAPIs to replace the existing one.

The new BAPI must retain the same name as the BAPI to be replaced. A numeric suffix is simply added to it. This suffix changes if further incompatible changes are made.

A number of incompatible changes must be made to the BAPI *FixedAsset.Create().* To implement these changes, a new BAPI, *FixedAsset.Create1(),* is created, in order to maintain the interface integrity of the BAPI *FixedAsset.Create().*

If further incompatible modifications must be made to the BAPI at a later date, yet another BAPI, *FixedAsset.Create2(),* must be created.

When creating the additional BAPIs, you must follow the guidelines in the BAPI Programming Guide [Ext.].

Support and Label the Expiring BAPI

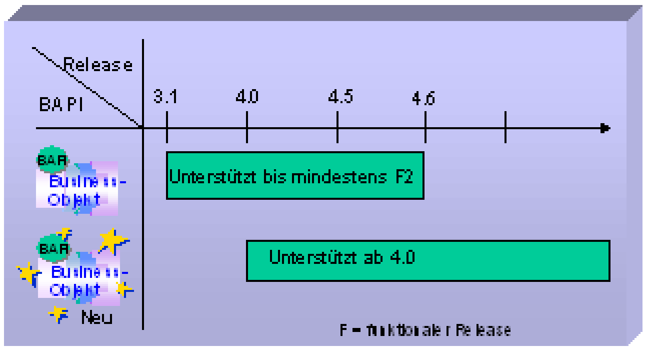
After the BAPI has been superseded by a new BAPI, you should not remove the replaced BAPI from the Business Object Repository (BOR). Instead, you first label this BAPI as expired, and continue its support in the release in which you have introduced the new BAPI as well as in the next functional release. During this time the original BAPI must remain fully functional and executable.

The following tasks are required when a BAPI has expired (become obsolete):

- Select the relevant SAP business object type in the Business Object Builder, and open the node *Methods*. Position the cursor on the BAPI, and choose *Edit* → *Change status to* → *Obsolete*.
- In the BOR, document which method(s) are to replace the expiring BAPI.
- Record the BAPIs that have been set to “obsolete” in a particular release in note number 0107644, “Collective Note for Obsolete BAPIs from Release 4.5A”, in SAPNet – R/3 Frontend.
- Inform your users about the relevant successor BAPIs in Release Notes.

The diagram below illustrates the expiry phase of BAPIs: In this example, the successor BAPI was implemented in Release 4.0. The replaced BAPI will therefore be supported in Release 4.0 (that is, in the correction release in which the successor BAPI was implemented) and in the next functional release, “F1”. In the following functional release “F2”, this BAPI will no longer be available.

BAPI Expiry Phase



Delete the Replaced BAPI

If the expiry phase of a BAPI set to obsolete has come to an end, you can remove the BAPI from the BOR. You should delete an obsolete BAPI as close to the beginning of a new release as possible, so that developers have time to adjust to its successor.

To delete a BAPI, follow the steps below:

- Delete the method from the BOR
  To do this, display the relevant SAP business object type in the Business Object Builder in change mode. Expand the node *Methods* Place the cursor on the appropriate BAPI, and delete it by choosing *Edit* → *Delete.*
- Delete the function module that implements the BAPI
  In the Function Builder, enter the name of the function module in the *Function module* field, and choose *Function module* → *Other functions* → *Delete*.
- Record the release in which the BAPI was deleted from the BOR in the note number 0107644, “Collective Note for Obsolete BAPIs from Release 4.5A”, in SAPNet – R/3 Frontend.

Examples of BAPI Calls

The following examples illustrate how BAPIs are called from different development platforms.

- Calling BAPIs from Java [Page 65]
- Calling BAPIs from Visual Basic [Page 71]
- Calling BAPIs from ABAP [Page 76]

Calling BAPIs from Java

This is an example program for calling a BAPI from the IBM development platform, Access Builder for SAP R/3.

Detailed program examples are shipped with the Access Builder for R/3.

Access Builder for SAP R/3

```
//Importing the required classes:

import com.sap.rfc.*;
import com.sap.rfc.exception.*;
import com.ibm.sap.bapi.*;
import com.ibm.sap.bapi.generated.*;

//Connecting to the R/3 System:

static private IRfcConnection establishConnection(MiddlewareInfo
aMiddlewareInfo)

      throws JRfcRemoteException

{

      IRfcConnection aConnection = null ;

      ConnectInfo aConnectInfo = null ;

      UserInfo aUserInfo = null ;

      String orbServerName = aMiddlewareInfo.getOrbServerName() ;

      // Please adjust the values written in UPPERCASE LETTERS
      // in the lines below so that they fit to your needs!
      // If you don't know the correct values ask your system
      // administrator!
      // After correcting these values you should change the
      // <bAdjusted> variable in the following line
      // from "false" to "true".
      // Then you can re-compile ("javac SampleCompanyCode.java") and
      // re-run ("java SampleCompanyCode -conn JNI") this sample...

      boolean bAdjusted = true;

      if (!bAdjusted) {

            throw (new JRfcRfcConnectionException (

                  "Please adjust the Connection-Parameters to your
                 needs! (See method \"establishConnection\")"));

      }

    //Connection information:

      aConnectInfo = new ConnectInfo (

            3,          // int aRfcMode 3=R/3 or 2=R/2

            null,       // String aDestination
            "9.7.12.7", // String aHostName YOUR  HOSTNAME (e.g. IP-
                     //address)
            0,         // int aSystemNo YOUR SYSTEM-NUMBER
            null,                   // String aGatewayHost
            null,                   // String aGatewayService
            null,                   // String aSystemName
            null,                   // String aGroupName
            null,                   // String aMsgServer
            false,                  // Boolean isLoadBalancing
            true);                  // Boolean isCheckAuthorization
    //User information:
      aUserInfo = new UserInfo (
            "MUSTER",         // String aUserName,    YOUR USERID
            "IDES",           // String aPassword,    YOUR PASSWORD
            "800",            // String aClient, YOUR CLIENT NUMBER
            "e",              // String aLanguage, YOUR PREFERRED
                             //LANGUAGE
            1103);            // int aCodePage YOUR REQUIRED CODEPAGE
    //Technical conversion for the selected middleware;
    // Open connection:
      IRfcConnectionFactory aConnectionFactory =
FactoryManager.getSingleInstance().getRfcConnectionFactory() ;
      aConnection =
aConnectionFactory.createRfcConnection(aConnectInfo, aUserInfo) ;
      aConnection.open() ;
   //Returning the connection:
      return aConnection ;
}
//Calling the main method:
public static void main (java.lang.String[] args)
  //Setting up the connection using the selected middleware:
{
      MiddlewareInfo aMiddlewareInfo = new MiddlewareInfo(args) ;
      FactoryManager aFactoryManager =
FactoryManager.getSingleInstance() ;
      aFactoryManager.setMiddlewareInfo(aMiddlewareInfo) ;
    //Initializing the connection object:
      IRfcConnection aConnection = null ;
      try
      {
            aConnection = establishConnection(aMiddlewareInfo) ;
      }
      catch (Exception ex)
      {
            System.out.println("ERROR : Could not create connection : "
+ ex) ;
            System.exit(-1) ;
      }
      System.out.println("Connection established.");
   // --- TEST CODE (start) --------------------------------------
      try
      {
            printList(aConnection) ;
//Calling the BAPI:
  //Declare an empty Object ID for the Business Object
  //CompanyCode:
            objectId = CompanyCode.getEmptyObjectId() ;
      //Entering a value in the object ID:
            objectId.getKeyField("COMPANYCODEID").setString("1000") ;
      //Instantiate the object CompanyCode with the object ID:
            companyCode = new CompanyCode(objectId) ;// Create 2nd
         CompanyCode
            System.out.println ("Successfully created new CompanyCode :
'" + companyCode + "'") ;
            printDetails(companyCode, aConnection) ;
      }
      // --- TEST CODE (end) ------------------------------------------
      catch (Exception ex)
      {
            System.out.println ("Unexpected exception occurred:");
            System.out.println (ex);
      }
```

```
}

private static void printDetails(CompanyCode companyCode,
IRfcConnection connection)

{

      try

      {

    //Declare the parameters of the BAPI CompanyCode.GetDetail:

            CompanyCodeGetdetailParams aCompanyCodeGetdetailParams =

                  new CompanyCodeGetdetailParams() ;

    //Aufruf des BAPIs CompanyCode.GetDetail auf die Objektinstanz:

            companyCode.getdetail(connection,
      aCompanyCodeGetdetailParams);

      //Splitting the parameter object into its separate components
      //(Struktur):

            Bapi0002_2Structure struct =
aCompanyCodeGetdetailParams.getCompanycodeDetail() ;

            System.out.println ("The details of the companycode are :
") ;

      //Splitting the structure into individual fields:

            System.out.println ("CompCode :            '" +
struct.getCompCode()  + "'");

            System.out.println ("CompName :            '" +
struct.getCompName()  + "'");

            System.out.println ("City1 :               '" +
struct.getCity()  + "'");

            System.out.println ("Country1 :            '" +
struct.getCountry() + "'");

            System.out.println ("Currency :            '" +
struct.getCurrency() + "'");

            System.out.println ("Langu1 :              '" +
struct.getLangu() + "'");

            System.out.println ("ChrtAccts :           '" +
struct.getChrtAccts() + "'");

            System.out.println ("FyVariant :           '" +
struct.getFyVariant() + "'");

            System.out.println ("VatRegNo :            '" +
struct.getVatRegNo() + "'");

            System.out.println ("Company :             '" +
struct.getCompany() + "'");
```

```
            System.out.println ("AddrNo :             '" +
struct.getAddrNo() + "'");
            System.out.println() ;
      }
      catch (Exception ex)
      {
            System.out.println("Exception in printDetails() : " + ex) ;
      }
      return;
}
private static void printList(IRfcConnection connection)
{
      try
      {
      //Declaring the parameter object:
            CompanyCodeGetlistParams aCompanyCodeGetlistParams =
                  new CompanyCodeGetlistParams() ;
      //Actual BAPI call:
            CompanyCode.getlist(connection, aCompanyCodeGetlistParams);
      //Splitting the parameter objects into its separate components
      //(Table):
            Bapi0002_1Table table =
aCompanyCodeGetlistParams.getCompanycodeList();
            int rowCount = table.getRowCount() ;
            System.out.println ("Returned table has " + rowCount + "
lines.");
      //Evaluating the table row by row:
            for (int i = 0; i < rowCount; i++)
            {
                  Bapi0002_1TableRow row = table.getRow(i) ;
                  System.out.println("\t" + row.getCompCode() + "\t" +
row.getCompName()) ;
            }
            System.out.println() ;
      }
      catch (Exception ex)
```

```
        {
                System.out.println("Exception in printList() : " + ex) ;
        }
        return;
}
}
```

Calling BAPIs from Visual Basic

This example illustrates a BAPI call in Visual Basic using the BAPI ActiveX Control. This report uses the service BAPI *BapiService.MessageGetDetail(),* to display the short text and the long text of error messages.

```
'
' Visual BASIC 5.0
' Copyright SAP AG Walldorf Juli 1998
'
' read a message short and longtext using the BAPI
' BAPI_MESSAGE_GETDETAIL of the object BapiService


' constant for user identification
Const cstrMUsrClient        As String = "000"
Const cstrMUsrUser          As String = "MYUSER"
Const cstrMUsrPassword      As String = "MYPASS"
Const cstrMUsrLanguage      As String = "EN"

' constant for system identification
Const cstrMSysSystem        As String = "P45"
Const cstrMSysMessageServer As String = "p45main.wdf.sap-ag.de"
Const cstrMSysGroupName     As String = "PUBLIC"
'
' constant values for reading message texts
Const cstrMMsgId            As String = "SX"
Const cstrMMsgNumber        As String = "101"
Const cstrMMsgVariable1     As String = "var1"
Const cstrMMsgVariable2     As String = "var2"
Const cstrMMsgVariable3     As String = "var3"
Const cstrMMsgVariable4     As String = "var4"
Const cstrMMsgLanguage      As String = "DE"

' other constant
Const cstrMPathfile         As String = "D:\A\saptext.rtf"

' password for login in R/3
Dim strMUsrPassword         As String
```

```
' react on button START
Private Sub cmdMsgStart_Click()

'   define object for BAPI ActiveX control
    Dim oBAPICtrl       As Object
'   define object for R/3 logon control
    Dim oLogonCtrl      As Object
'   business object BapiService
    Dim boBapiSercice   As Object

'   for BAPI: BapiService.MessageGetDetail
    Dim oMsgReturn      As Object
    Dim oMsgText As Object
    Dim intCounter      As Integer
'   to open the file you need a file channel
    Dim intChannel      As Integer



'   create BAPI ActiveX control object
    Set oBAPICtrl            = CreateObject("SAP.BAPI.1")
'   create R/3 logon control object
    Set oLogonCtrl           = CreateObject("SAP.Logoncontrol.1")
'   connection object is part of the BAPI ActiveX Control object
    Set oBAPICtrl.Connection = oLogonCtrl.NewConnection

'   fill logon parameters for system to use
    oBAPICtrl.Connection.System        = txtSysSystem
    oBAPICtrl.Connection.MessageServer = txtSysMessageServer
    oBAPICtrl.Connection.GroupName     = txtSysGroupName
'   fill logon parameter for user
    oBAPICtrl.Connection.Client        = txtUsrClient
    oBAPICtrl.Connection.User          = txtUsrUser
    oBAPICtrl.Connection.Password      = strMUsrPassword
    oBAPICtrl.Connection.Language      = txtUsrLanguage
```

```
'   user logon to R/3
    If oBAPICtrl.Connection.Logon(frmStart.hWnd, False) = False Then
        MsgBox "R/3 connection failed"
        End
    End If

'   create BAPI service object
    Set boBapiService = oBAPICtrl.GetSAPObject("BapiService")

'   call method of BapiService
    boBapiService.MessageGetDetail id:=txtMsgId, _
                                   Number:=txtMsgNumber, _
                                   Language:=txtMsgLanguage, _
                                   Textformat:=cboMsgTextformat.Text, _
                                   message:=strMsgShorttext, _
                                   Return:=oMsgReturn, _
                                   Text:=oMsgText

'   fill field in form
'    If txtMsgShorttext = "" Then
'        MsgBox "No message read"
'    End If

'   user logoff from R/3
    oBAPICtrl.Connection.Logoff

'   error handling check if RETURN parameter is not empty and react
    If oMsgReturn.Value("TYPE") <> "" Then
        lblReturn.Caption = oMsgReturn.Value("TYPE") + _
                            ". " + _
                            oMsgReturn.Value("ID") + _
                            ". " + _
                            oMsgReturn.Value("NUMBER") + _
                            ". " + _
```

```
                                oMsgReturn.Value("MESSAGE") + _
                                ". " + _
                                oMsgReturn.Value("MESSAGE_V1") + _
                                ". " + _
                                oMsgReturn.Value("MESSAGE_V2") + _
                                ". " + _
                                oMsgReturn.Value("MESSAGE_V3") + _
                                ". " + _
                                oMsgReturn.Value("MESSAGE_V4") + _
                                ". " + _
                                oMsgReturn.Value("LOG_NO") + _
                                ". " + _
                                oMsgReturn.Value("LOG_MSG_NO")

    Else

'       fill form fields
        txtMsgShorttext = strMsgShorttext
        arrayText       = oMsgText.Data

'       handling of non RTF texts
        If cboMsgTextformat.Text <> "RTF" Then
            For intCounter = 1 To oMsgText.RowCount
                If intCounter = 1 Then
                    rtfMsgLongtext.Text = arrayText(intCounter, 1)
                Else
                    rtfMsgLongtext.Text = rtfMsgLongtext.Text + _
                                          Chr(13) + Chr(10) + _
                                          arrayText(intCounter, 1)
                End If
            Next intCounter
        End If

'       handling of RTF texts
        If cboMsgTextformat.Text = "RTF" Then
```

```
'           save text as rtf file
            intChannel = FreeFile
            Open cstrMPathfile For Output As #intChannel
                For intCounter = 1 To oMsgText.RowCount
                    Print #intChannel, arrayText(intCounter, 1)
                Next intCounter
            Close #intChannel
            rtfMsgLongtext.LoadFile cstrMPathfile, rtfRTF
        End If
    End If
End Sub
```

Calling BAPIs from ABAP

This report uses the service BAPI *BapiService.MessageGetDetail()*, to display the short text and the long text of error messages.

```
*----------------------------------------------------------------------
--*
*        read a message short and long text using the BAPI
*
*        BAPI_MESSAGE_GETDETAIL of the object BapiService.
*
*----------------------------------------------------------------------
--*
```

*** Data declaration**

```
DATA : MY_ID          LIKE BAPIRET2-ID,
       MY_NUMBER      LIKE BAPIRET2-NUMBER,
       MY_TEXTFORMAT  LIKE BAPITGA-TEXTFORMAT,
       MY_MESSAGE_V1  LIKE BAPIRET2-MESSAGE_V1,
       MY_MESSAGE     LIKE BAPIRET2-MESSAGE,
       MY_RETURN      TYPE BAPIRET2.
DATA BEGIN OF MY_TEXT OCCURS 1.
     INCLUDE STRUCTURE BAPITGB.
DATA END OF MY_TEXT.
```

*** Enter values in object**

```
MOVE 'FI'   TO MY_ID.              "message id of message to read
MOVE '024'  TO MY_NUMBER.          "message number of message to read
MOVE 'ASC'  TO MY_TEXTFORMAT.      "text format, here ASCII
MOVE '0001' TO MY_MESSAGE_V1.      "text to fill into message
```

***BAPI call**

```
CALL FUNCTION 'BAPI_MESSAGE_GETDETAIL'
     EXPORTING
          ID         = MY_ID
          NUMBER     = MY_NUMBER
*         LANGUAGE   = SY-LANGU
          TEXTFORMAT = MY_TEXTFORMAT
          MESSAGE_V1 = MY_MESSAGE_V1
```

```
*         MESSAGE_V2 =
*         MESSAGE_V3 =
*         MESSAGE_V4 =
     IMPORTING
          MESSAGE    = MY_MESSAGE
          RETURN     = MY_RETURN
     TABLES
          TEXT       = MY_TEXT
         .
```

*** Print results**

```
WRITE : / 'Input' COLOR 5.
WRITE : / 'my_id...........:', MY_ID.
WRITE : / 'my_number.......:', MY_NUMBER.
WRITE : / 'my_textformat...:', MY_TEXTFORMAT.
WRITE : / 'my_message_v1...:', MY_MESSAGE_V1.
WRITE : / 'Output' COLOR 5.
WRITE : / 'my_message........:', MY_MESSAGE.
WRITE : / 'my_return.........:', MY_RETURN.
WRITE : / 'Text output' COLOR 5.
LOOP AT MY_TEXT.
  WRITE : / MY_TEXT.
ENDLOOP.
```

Adapting BAPIs

Use

You can change SAP Business Object types and their BAPIs, for example, if you want to add a parameter to a BAPI or add a method to a business object type.

Features

BAPIs can be modified in various ways to suit customer needs:

Type of Adaptation	**Further Information**
Extension parameters	This refers to enhancements to BAPIs that do not involve modifications. For more information see Customer Enhancement Concept [Ext.].
Modifications	For more information see Modifying BAPIs [Ext.].
Customer enhancements	For more information see Modifying BAPIs [Ext.]. For more detailed information about implementing and programming BAPIs see BAPI Programming Guide [Ext.].