BAPI Programming Guide (CA-BFA)

HELP.CABFABAPIPG

Release 4.6C

SAP™

Copyright

Icons

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

Contents

BAPI Programming Guide CA-BFA)

The BAPI programming guide describes the development process for developing new BAPIs. The guidelines also apply to enhancing and modifying existing BAPIs.

Introduction

Business Object Types and BAPIs

Business **A**pplication **P**rogramming **I**nterfaces (BAPIs) enable access to SAP functions across formal, stable and dialog-free interfaces. These interfaces can be used by external applications developed by customers and complementary software partners as well as by other SAP applications.

BAPIs are defined as API methods of SAP Business Object Types [Ext.]. These object types are used within the Business Framework to enable object-based communication between components. Business objects and their BAPIs enable object orientation to be used in central information processing in companies. For example, existing functions and data can be reused, trouble-free technical interoperability can be achieved, and non-SAP components can be implemented.
Applications can use BAPIs to directly access the application layer of the R/3 System and, as clients, applications can use the business logic of the R/3 System. BAPIs provide the client with an object-oriented view of the application objects, without needing to know the implementation details.

BAPIs are always developed by defining scenarios. These scenarios are used to map and implement system-wide business processes.

Target Group for Documentation

This documentation describes the guidelines SAP uses for developing and implementing BAPIs to ensure that the development of BAPIs is as standard as possible. These standards make BAPIs easy to use.
This programming guide is targeted at SAP developers, partners and customers who want to implement BAPIs.

This documentation has been written for Release 4.6C. Unless stated otherwise, these guidelines also apply to Releases back to and including 4.0.

See also:

Terminology [Ext.]Further Documentation on BAPIs [Ext.]Changes in Release 4.6C [Ext.]

Overview of the Development Process

This section describes the process of developing new BAPIs. The guidelines also apply to enhancing and modifying existing BAPIs.

For more information on enhancements and modifications, see:

- Customer enhancement of BAPIs [Ext.]
- BAPI Modifications [Ext.]
- SAP Enhancements to Released BAPIs [Ext.]

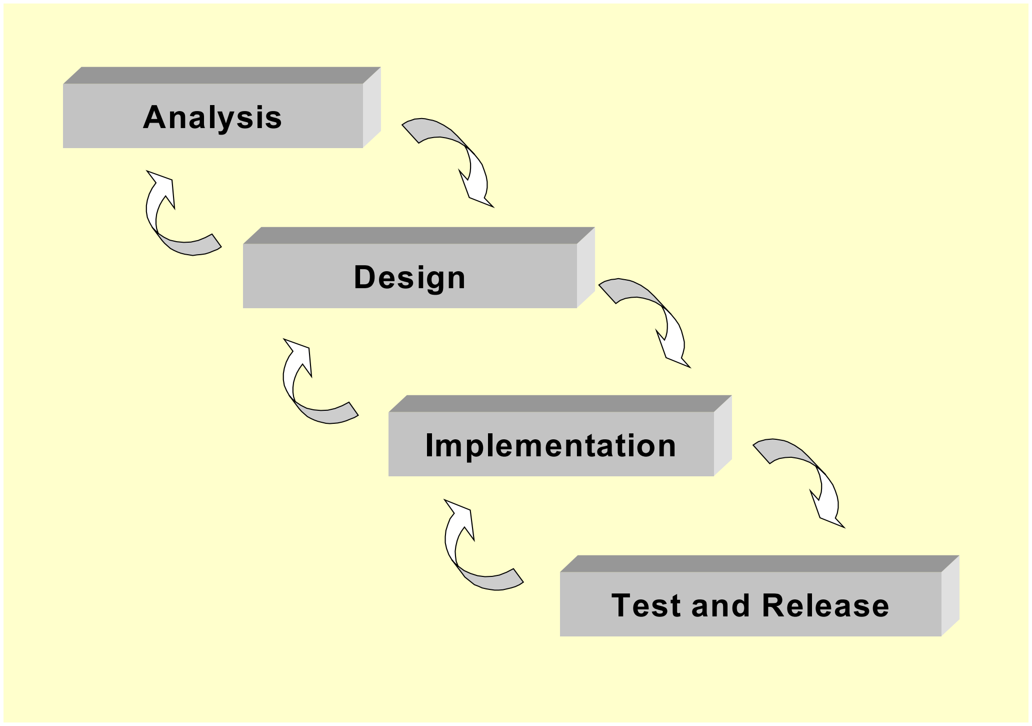
Prerequisites

To implement a BAPI using these programming guidelines, you need:

- Basic understanding of BAPIs
- Knowledge of the ABAP programming language
- General knowledge of the R/3 System

Process Flow

The entire development process can be divided into four phases:



1. **Analysis:**
   The analysis phase involves determining which business process is to be implemented using the BAPI. Scenarios are defined and then the steps are identified which are to be visible externally, (to other components), and implemented by business object types and their BAPIs.

2. **Design:**
   For each BAPI involved in a scenario, the structure of the BAPI's interface is defined at a conceptual level. The purpose of the design phase is to structure the interfaces concisely and intuitively so that the BAPIs will be easy to use. At the same time the interfaces must comply with all technical and formal guidelines.
3. **Implementation:**
   The implementation phase can be divided into three successive stages:
   - Defining the data structures (including domains and data elements) that the BAPI parameters are based on.
   - Implementing the program logic in function modules.
   - Defining the BAPI in the Business Object Repository (BOR) to ensure object-oriented access.
4. **Testing and releasing:**
   The documentation, as well as the functionality and usability of a BAPI are checked in the test phase. The BAPI can be released after the test phase has been successfully completed.

Organizational Support

Throughout the development process the **BFA A**pplication **G**roups provide support for:

- Designing the scenario, the business object types and the BAPIs.
- Technical implementation
- Any problems that may arise

Tool Support

The BAPI Explorer (Transaction ***BAPI***) is the central tool as of Release 4.6A to support your BAPI developments. The BAPI Explorer enables easy access to all the relevant information in the BAPI environment (including general documentation as well details about special business object types and BAPIs) . The Explorer integrates all the tools required for BAPI development and manages all the phases of the development process in project forms.

See also:

Details on BAPI Explorer [Ext.]

Analysis

For analysis purposes the business process and the scenario implementing this business process must be described in detail. The Business Object Types [Ext.] and their BAPIs that are used in the scenario must also be identified. The BAPIs are atomic modules that can be used to create different scenarios.
The main dilemma with developing BAPIs is:
On the one hand BAPIs should be developed so that they can be used in different scenarios and on the other hand the interface must be clearly structured.

It is important that you do not stipulate in this phase exactly what the signature of the individual BAPIs is to look like. The BAPIs required for the scenario are simply identified and their functions and the data they require are defined.

The analysis can be divided into three phases:

Describing the Scenario

The Business Process

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

Creating a sales order in an affiliate with a credit standing check in the central organization.

To define a business process you have to:

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

The Scenario

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

Creating sales orders and credit standing checks in different R/3 Systems (central accounting, decentralized sales and distribution).

The process of defining the scenario involves:

- ☐ I d e n ti f y i n g t h e r e l e v a n t c o m p o n e n t s a n d t h e t a s k s t h e y p e rf o r m .

- Determining whether application systems are to be integrated in the scena

- ☐ Determining the information and process flows. Here you have to:

Describing the Scenario

- ☐ Determine which steps are to be processed system-wide and which steps are to be processed within a single component.
- ☐ Define which data is exchanged between which components and who initiates this exchange.
- ☐ Determine the sequence in which the individual steps are processed.
- ☐ Identify the steps belonging to a single transaction (LUW).

  Example: Developers must ask themselves whether it makes sense to create a customer in one LUW and then create a sales order in the same LUW . Transaction Model for Developing BAPIs [Ext.]

- ☐ Error handling must be much more precise and comprehensive than with local applications.

- You have to decide whether system coupling should be narrow or loose for t

- ☐ All steps critical to performance must be identified.

- You should also consider which R/3 Releases are to be supported within the s

Describing the Scenario

- For each scenario you must identify a person responsible for ensuring that t

Business Object Types and BAPIs

Each component in the scenario must provide services so that the cross-component steps can be carried out. You have to work out how to distribute responsibility for the services between the business object types and their BAPIs. For example, the BAPI *CreateFromData* of business object type *SalesOrder* is used to create a sales order in the R/3 System from an external application.

For more information see also: Business Object Types [Ext.].

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

- Encapsulation of the required functions in business object types. This involves breaking down the whole system into separate responsibilities. The breakdown and encapsulation of functions must be explicit and disjunctive.
- Do business object types already exist for these responsibilities?
- Do design patterns already exist?

  Find out whether any problems have already been dealt with.

  Example: Header/item pattern, for example, for *SalesOrder*, *PurchaseOrder*, etc.
- Delimit the responsibility for other business object types.
- Determine the services provided based on the defined responsibility.

For more information see also: Design Criteria for Business Objects [Ext.].

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

- Each service is implemented by one or more BAPIs (= method of business object type).
- BAPIs make available the functions of a business object type. You should be able to use them independently of individual scenarios and also in different scenarios.

  To make it easier to use a BAPI in different scenarios, the BAPI signature should be created so that the BAPI's parameters and fields are assigned separately according to the application.

- ☐ However, this type of scenario can affect the granularity of the BAPI. Application systems are integrated differently from the integration of alternative frontends:

  1. Integrating application systems

  - The integration of application systems typically involves program-to-program communication, loose asynchronous coupling and the exchange of larger volumes of data.
  - The main requirement of the business object types and BAPIs used in these scenarios is that performance is high, for example, by minimizing the number of calls.
  - The result is a rough granularity of the application that is implemented by more extensive BAPIs.
  - **Example**: A program to create a sales order automatically.
    The program uses the business object type *SalesOrder* with the BAPI *CreateFromData*. The complete sales order is created in the sending system and then sent to the receiving system.

  2. Integrating alternative frontends

  - Scenarios with alternative frontends represent human-to-machine communication and can be implemented synchronously as well as asynchronously .
  - Business object types and BAPIs must be structured to ensure flexibility, configurability and minimal error situations.
  - The result is a finer granularity of the application that should correspond to the dialog processing in R/3.
  - **Example**: Customer sales order is created interactively in the Internet.
    The sales order is can be created using the two methods *CreateFromData* and *AddItem* of the business object type *SalesOrder*. In this case the method *CreateFromData* simply creates the sales order header, whilst the method *AddItem* can add new sales order items.

  SAP Internal:

  Wherever possible, a BAPI should be created so that it can be used for **both** purposes. The most important objective is to develop a range of standardized BAPIs for each business object type which can be used universally.

- ☐ The scenario must be structured so that it is possible to get all the information required for a BAPI call from the R/3 System from other BAPIs beforehand.

- ☐ If there are any BAPIs that are Customizing-dependent, BAPIs must be provided that can export these Customizing settings.

☐ **SAP Internal**

The scenario, the link to the application's object model and the design of new business object types is agreed with one of the BFA Application Groups.

If business object types that do not yet exist are identified during the analysis phase, these object types can only be created by the BFA AGs. You have to create a request for a new business object type in the BAPI Explorer [Ext.].

For examples of BAPI scenarios refer to Overview of ALE Integration Scenarios [Ext.] in the Library of ALE Business Processes.

Defining the Scenario in the BAPI Explorer

Prerequisites

The scenario is fully described.

Process Flow

Define a project form in the BAPI Explorer:

1. Start the BAPI Explorer in the relevant development system using Transaction **BAPI** (as of Release 4.6A).
2. Select the tab page *Project* and create a project to implement new BAPIs.
3. A project form guides you through the entire procedure for creating a BAPI. In the first section the basic data can be created for the defined scenario .

See also:

BAPI Explorer [Ext.]

Reviewing the Scenario

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

Features

The following questions should be answered:

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

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

See also:

Designing the BAPI [Page 24]

Designing the BAPI

After the scenario and the business object types and BAPIs used in it have been identified in the analysis, the BAPI signature is defined conceptionally within the design phase. The content should be described and the parameter names and the parameter structure must be defined.

Because BAPIs are business interfaces and not technical interfaces, when you develop a BAPI, you must fulfil the **central requirement** that**:**
A user must be able to implement a BAPI call correctly in an external application, using only their knowledge of the application and the BAPI documentation. Users are not expected to have any knowledge of R/3, so that they can view the R/3 System as a "black box".

In this phase you also have to decide from the BAPI's functions whether the BAPI is an instance method or a class method and whether it can be implemented as a Standardized BAPI [Page 27].

Instance methods refer to a specific instance of a business object type, for example, the BAPI *SalesOrder.GetDetail* retrieves the details of precisely one sales order. These methods are defined as **instance-dependent** in the BOR.
Class methods do not refer to a specific instance of a business object type, for example, the BAPI *SalesOrder.GetList* supplies a list of all the existing sales orders that match specific criteria. Standardized create methods also belong to class methods. A create method creates a new instance, for example, the BAPI *SalesOrder.Create* creates a new sales order in the R/3 System. These methods are defined as **instance-independent** in the BOR.

The specific instances of a business object type are identified by their key fields, which is why they play a special role in this context. In the design phase, the keys of instance-dependent BAPIs should be identified as special parameters. For more details see Defining the Interface [Page 42].

When designing the BAPI signature you should keep in mind:

1. The design of the method (BAPI) and the parameters must comply with certain **conventions** See also Conventions [Page 25].
2. There are a range of **standard methods and parameters** that provide basic functions. If the BAPIs and their parameters fall into this category, they must be structured according to specified rules. For an overview see Standardized BAPIs [Page 27] and Standardized Parameters [Page 29].
3. So that the signature is clearly structured and easy to use, you should follow the Design Recommendations [Page 31].

Conventions

The following conventions apply to **BAPI methods**.

- ☐ If the BAPI to be implemented is a standardized BAPI, use the generic names, for example, *GetList*, *GetDetail*.
- ☐ The method name must be in English (maximum 30 characters).
- ☐ The individual components of a BAPI name are separated by the use of upper and lower case.

  Example: *GetList*
- ☐ Underscores ("_") are not allowed in BAPI names.
- ☐ Each BAPI has a return parameter that is either an export parameter or an export table.
- ☐ So that customers can enhance BAPIs, each BAPI must have an *ExtensionIn* and an *ExtensionOut* parameter. See also Customer Enhancements to BAPIs [Ext.].

See also: Standardized BAPIs [Page 27]

The following conventions apply to **parameters**:

- ☐ If standardized parameters are used, you have to use the names specified for standardized parameters.
- ☐ BAPI parameter names should be as meaningful as possible.
  Poorly chosen names include abbreviations and technical names (e.g. "flag", table names, etc.).
- ☐ The parameter and field names must be in English with a maximum of 30 characters.
- ☐ The components of a parameter name in the BOR are separated by upper and lower case letters to make them easier to read.

  Example: *CompanyCodeDetail*
- ☐ Values that belong to each other semantically should be grouped together in one structured parameter, instead of using several scalar parameters.
- ☐ For ISO-relevant fields (country, language, unit of measure, currency), additional fields for ISO codes are provided.
- ☐ Unit of measure fields must accompany all quantity fields and currency identifiers must accompany currency amount fields.

See also: Standardized Parameters [Page 29]

Standardized BAPIs

Use

Some BAPIs provide basic functions and can be used for most SAP business object types. These BAPIs should be implemented the same for all business object types. Standardized BAPIs are easier to use and prevent users having to deal with a number of different BAPIs. Whenever possible, a standardized BAPI must be used in preference to an individual BAPI.

Features

The following standardized BAPIs are provided:

Reading instances of SAP business objects

- **GetList ( )**
  With the BAPI *GetList* you can select a range of object key values, for example, company codes and material numbers. The BAPI *GetList()* is a class method.
  For more information see Programming GetList() BAPIs [Ext.].
- **GetDetail ( )**
  With the BAPI *GetDetail()* the details of an instance of a business object type are retrieved and returned to the calling program. The instance is identified via its key. The BAPI *GetDetail()* is an instance method.
  For more information see Programming GetDetail() BAPIs [Ext.].

BAPIs that can create, change or delete instances of a business object type

The following BAPIs of the same object type have to be programmed so that they can be called several times within one transaction. For example, if, after sales order 1 has been created, a second sales order 2 is created in the same transaction, the second BAPI call must not affect the consistency of the sales order 2. After completing the transaction with a COMMIT WORK, both the orders are saved consistently in the database.

- **Create( ) and CreateFromData( )**
  The BAPIs *Create()* and *CreateFromData()* create 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 an SAP business object type, for example, a purchase order. The BAPI *Change ()* 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 or sets a deletion flag.
  The BAPI *Undelete()* removes a deletion flag. These BAPIs are instance methods.
  For more information see Programming Delete() BAPIs [Ext.].

Standardized BAPIs

- **Cancel ( )**
  Unlike the BAPI *Delete(),* the BAPI *Cancel()* cancels an instance of a business object type. The instance to be cancelled remains in the database and an additional instance is created and this is the one that is actually canceled. The *Cancel()* BAPI is an instance method.
  For more information see Programming Cancel() BAPIs [Ext.].
- **Add<subobject> ( ) and Remove<subobject> ( )**
  The BAPI *Add<subobject>* adds a subobject to an existing object instance and the BAPI and *Remove<subobject>* removes a subobject from an object instance. These BAPIs are instance methods.
  For further information see Programming Methods for Sub-Objects [Ext.].

BAPIs for Mass Data Processing

The BAPIs listed above for creating and changing data can also be used for mass processing. For more information see BAPIs for Mass Data Transfer [Ext.]

BAPIs 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.].

Other Less Used Standardized BAPIs

- Programming GetStatus() BAPIs [Ext.]
- Programming ExistenceCheck() BAPIs [Ext.]

Standardized Parameters

Use

There are some parameters that can be created for various BAPIs because they contain the same or the equivalent data in all BAPIs. 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 Extension Parameters [Ext.] and Customer Enhancements to 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 [Ext.].

Selection Parameters

Standardized selection parameters are used in BAPIs that can be used to search for specific instances of a business object type (e.g. in *GetList()*). These parameters enable the BAPI caller to specify the relevant selection criteria.
For more information see Selection Parameters [Ext.].

Test Run Parameters

The parameter *TestRun* is used in write BAPIs (*Create()* and *Change()*), 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 (e.g. the documentation of a business object type), you have to create standardized text transfer parameters.
For more information see Text Transfer Parameters [Ext.].

Design Recommendations for Interfaces

So that BAPIs can be used intuitively, you should follow the design recommendations below:

☐ Parameter data should be grouped together according to business criteria.

☐ The inbound interface should be clearly structured and intuitive. Parameters should in the first instance be structured in accordance with the application.

Example: Structured parameters should be structured so that they contain all the fields required for the most simple application in one logical block. Additional fields required for special cases should be grouped by application and attached to the "main block".

☐ The interface must not contain any fields that external users would not be able to interpret. You must also try to remove any internal details specific to R/3.

☐ Whenever possible, **input help** (F4 help) should be provided for parameters and parameter fields. This enables the client to easily determine all the possible input values for a field by calling the BAPI *Helpvalues.GetList.*
See also: Providing Input Help (F4 Help) [Ext.]

☐ Dependencies between data fields and between parameters must be included in the documentation.

☐ All the information supplied with a BAPI should be able to be used easily without having to change any of the data. So export structures should be able to be reused as the input of a different BAPI.

☐ All key fields in *GetDetail()* and *GetList()* BAPIs must be displayed as text fields.

If there are several texts for one key field, these texts must be stored in a separate table.

☐ When you are structuring the interface, you should consider what the BAPI is used for.

- With BAPIs used for *batch processing* performance is the most important consideration in the design.
- *BAPIs used for alternative front-ends* require a more flexible interface as they tend to be used in more complex processes.

See also: Example [Ext.] of the conceptual design of a BAPI.

Implementing a BAPI

Once you have completed the conceptual design of the BAPI, information specific to SAP must now be considered in the implementation phase. For example, the SAP data structures are determined and the program logic is implemented for the BAPI parameters.

The following documentation provides an overview of the implementation process and the relevant components and tools for the BAPIs are introduced. Each work step is then described in detail.

See also:

The Implementation Process [Page 33]

Actions in the ABAP Dictionary [Page 38]

Actions in the Function Builder [Page 41]

Actions in the BOR [Page 55]

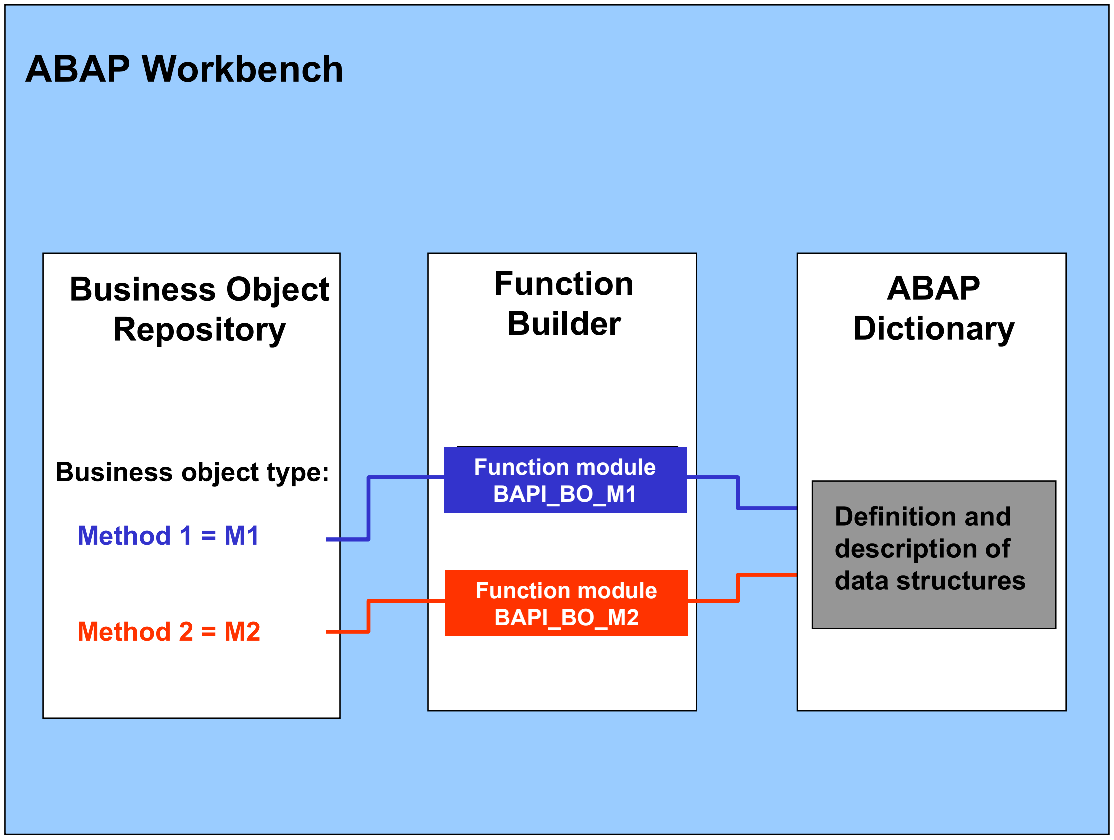
BAPI Documentation [Page 63]

BAPI/ALE Integration [Page 71]

The Implementation Process

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

The graphic below illustrates the relationships between the components:


ABAP Workbench
Business Object Repository
Business object type:
Method 1 = M1
Method 2 = M2
Function Builder
Function module BAPI_BO_M1
Function module BAPI_BO_M2
ABAP Dictionary
Definition and description of data structures

Process Flow

The structure of the tools in the ABAP Workbench determine how the BAPI is implemented.

1. Defining the Data Structures [Page 38] (including domains and data elements) in the ABAP Dictionary.
2. Implementing the Function Module [Page 41] in the Function Builder
3. Defining the Business Object Type and Its Methods [Page 55] in the BOR

The Implementation Process

All three phases have accompanying documentation [Ext.] that is created in the corresponding tools.

All the required work steps can be started from the BAPI Explorer.

See also :

Tools [Page 35].

Further Issues [Page 37]

Tools

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

The BAPI Explorer

As of Release **4.6A** the BAPI Explorer [Ext.] is the central tool for developing BAPIs. To start it, you can enter Transaction code ***BAPI*** or, from the *SAP Easy Access* menu select → *Tools* → *Business Framework* → *BAPI Explorer.* As the central access point into the BAPI world, the BAPI Explorer provides access to all the relevant information by way of context-sensitive links to all the required tools and transactions. Project forms guide you step by step through the development process. The project forms contain checkboxes with important information and details about the tools provided. You can also log the progress of your project.

The ABAP Dictionary

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

For more information see Documentation on the ABAP Dictionary. [Ext.].

The Function Builder

With the Function Builder function modules can be created, implemented, tested and documented within a function group. The Function Builder contains a Function Library that serves as the central storage for all function modules.

For more information about the Function Builder see the documentation on the Function Builder [Ext.].

The Business Object Repository (BOR)

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

- Enables an object-oriented view of the R/3 System data and processes.
  R/3 application functions are accessed using methods (BAPIs) of SAP business object types. Implementation information is encapsulated; only the interface functionality of the method is visible to the user.

- Arranges the various interfaces in accordance with the component hierarchy. Functions can be searched and retrieved quickly and simply.

- Manages BAPIs in release updates.
  BAPI interface enhancements made by adding parameters are recorded in the BOR. Previous interface versions can thus be reconstructed at any time. When a BAPI is created, the release version of the new BAPI is recorded in the BOR. The same applies when any

interface parameter is created.
The version control of the function module that a BAPI is based on is managed in the Function Builder.

- Ensures interface stability.
  Any interface changes that are carried out in the BOR are automatically checked for syntax compatibility against the associated development objects in the ABAP Dictionary.

For further information about the BOR see the documentation on the Business Object Builder [Ext.].

Outlook

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

Further Issues

Before you start the implementation phase, you should keep in mind:

☐ **Releasing BAPIs**

When you release a BAPI, it becomes available as a fully implemented method of a business object type. Releasing also prevents anyone making incompatible changes to the BAPI, because all changes made to a released BAPI are automatically checked for compatibility in the BOR and the ABAP Dictionary, and incompatible changes are rejected.

SAP Internal

If the BAPI is only to be used internally, and therefore not for release to customers, the status of the function module must be set to *Released internally*.
The BAPI must be released in cooperation and agreement with the BFA AGs and with persons responsible for quality control in the development department concerned.

☐ **Enhancements and Modifications of BAPIs**

If you are enhancing or modifying a BAPI, rather than creating a new BAPI, keep in mind the requirements below. For more information see Further Enhancements, Modifications [Ext.].

☐ **BAPIs Used for Outbound Processing**

Up to and including Release 4.0 BAPIs could only be implemented as function modules in the same system that they were defined in.
As of Release **4.5A** BAPIs can also describe interfaces that are implemented in a non-SAP system but that can be called from R/3 Systems. These BAPIs are known as outbound BAPIs and are defined in the BOR as API methods of SAP interface types.
There are always separate systems for defining (client) and for implementing (server) BAPIs Used for Outbound Processing [Ext.].

Actions in the ABAP Dictionary

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

The conventions below are important:

Conventions for BAPI Data Structures

- Each parameter must refer to a data structure in the ABAP Dictionary. In the case of structured parameters this is always to the whole BAPI data structure. If the parameter consists of only one field, it must refer to a field in a BAPI structure.
- You have to create separate data structures for BAPIs, which are independent of the data structures generally, used in the R/3 application.
  Reason: When the BAPI is released the underlying structures of the BAPI are frozen and restrictions apply if you want to later change the structures. See also Enhancements to released BAPIs [Ext.].

  Existing internal structures can be mapped to existing BAPI structures with a field mapping function module that is automatically generated by a mapping tool (Transaction BDBS).
  For more information see Converting Between Internal and External Data Formats [Ext.].
- All data structure names must begin with **<namespace>BAPI**.
- BAPI structure names should be as meaningful as possible.
- You must not use APPENDs and INCLUDEs in BAPI data structures.

  Reason: APPENDs and INCLUDEs can cause Incompatibilities [Ext.], if BAPI data structures are changed.
- Always try to use existing central data elements and domains for fields.

The Example [Ext.] of the BAPI *CompanyCode.GetDetail* shows how data structures should be defined.

Conventions for Field Names

- ☐ The fields in structures must have meaningful English names with a maximum of 30 characters.

  SAP Internal

  Use report BAPIFELD to create a list of English field name proposals for an ABAP Dictionary structure. Refer to the report documentation to find out how to use the report.

- ☐ A meaningful English default field name with a maximum of 30 characters should be defined for each data element.

  SAP Internal

  In the future the default name will always be proposed by the SAP internal report BAPIFELD.

  Procedure: You can take data objects that still need to be edited from the report BAPIFELD. In the ABAP Dictionary you can define the default field name of the data element by choosing *Definition → Default field name*.

Conventions for Input Help

- ☐ You may have to define single values or a value table for the domain so that F4 help is available for use.

- ☐ All the useful Possible entries [Ext.] (which can be used in the service BAPI *Helpvalues.GetList*) must be defined for the data structures/data elements. To do this a foreign key must be specified in the fields of a BAPI structure. If a value table has been defined in a field domain, a foreign key must also be defined.

Technical Conventions

- ☐ The internal data format is used for all parameter fields.
  See also Mapping Between Internal and External Data Formats [Ext.].

- ☐ If an external key as well as an internal key has been defined in the database, the external key must always be used in the BAPI interface.

- ☐ For fields relevant to ISO (country, language, unit of measure, currency), additional fields are provided for ISO-Codes.

- ☐ All currency fields use the domain BAPICURR. In special cases the domain BAPICUREXT can also be used.

- ☐ You must always use a period for the decimal point.

- ☐ All currency amounts and units of measure must have the decimal point in the correct place.

  Reason: The BAPI always uses a standard amount format with commas. So when converting currency amounts use the function modules: BAPI_CURRENCY_CONV_TO_EXTERNAL and BAPI_CURRENCY_CONV_TO_INTERNAL.

- ☐ With quantity fields the reference field/reference table must be correctly set to the unit of measure field.

- ☐ All the data elements for date fields must have the format YYYYMMDD.

Conventions for BAPI/ALE Integration

So that an IDoc for asynchronous communication can be generated from a BAPI, the following conditions must be satisfied:

- ☐ BAPI structure names must not be longer than 27 characters, otherwise the automatically generated name for the associated segment will be too long and will have to be changed manually later.

  Reason: The segment is generated following the convention <BAPI structure name> + <three digit number>.

- ☐ Single fields in a data structure can only be a maximum of 250 bytes.

Conventions for Customer Developments

- ☐ If IBUs, partners and customers are creating data structures (domains or data elements), the Namespaces [Ext.] provided by SAP must be complied with.

Actions in the Function Builder

Once the parameters have been defined, the function module underlying the BAPI is created in the Function Builder (Transaction SE37). Transaction SE37 can be accessed from within the project form in the BAPI Explorer.

When Creating the Function Module [Ext.] you must follow the rules below:

- All function modules must have the naming convention:

  <Name space>BAPI_<business object>_<method>.

  A maximum of 30 characters is allowed.
  If required, you can abbreviate the name but still following the above convention. You must still be able to recognize the business object type assignment.

- All the BAPIs belonging to one SAP business object type should be created in one function group. You should only deviate from this rule in exceptional cases.

- BAPIs belonging to different SAP business object types must not be put in the same function group.

The next steps:

Defining the Interface [Page 42]

Implementing the Function Module [Page 44]

Defining the Interface

A BAPI parameter must be defined in the method of the business object type in the BOR as well as in the function module. In both cases the definition must be identical except for the key fields of the business object type.
The Key Fields [Ext.] of a business object type are used to identify a business object type at a semantic and technical level. The whole key, rather than parts of it is always used (fully defined key). This has semantic reasons (an instance is identified by the key) as well as technical reasons (keys are generated in the BOR and various checks are made against existing key fields).

If a BAPI refers to a specific instance of a business object type, together with other parameters, the key fields of the business object type are transferred to the method in the BOR to identify the instance. In this case the key fields are not listed again under the method parameters in the BOR. This distinction cannot be made in the function module, which is why the key fields must be listed here as parameters.

The following conventions apply to key fields for function module interfaces:

- With **instance methods** all the BOR key fields are mandatory import parameters in the function module. BOR key fields must not be defined as export parameters in the function module.
- With **class methods** BOR key fields must not be defined as export parameters in the function module (exception: Create methods) Neither are BOR key fields allowed to be defined as import parameters in the function module.
- With **create methods** all the BOR key fields of the method are defined as export parameters in the function module. BOR key fields must not be defined as import parameters in the function module (this applies to all class methods). Create methods are classified as instance-independent in the BOR.

If, in accordance with the above guidelines, key fields are defined as parameters in the function module, the conventions below apply:

- The fully defined key is used and not a partial key.
- There is a parameter in the function module for each key field of the business object type.
- The parameter in the function module and the BOR key field have the same name.

In the above explanation a key field means a parameter that is used to identify a runtime instance (in the OO sense) and that has the same name as the key field of the business object type. It is of course possible to use other parameters to display the required business information represented by the key.

Example: In accordance with the above guidelines, a *Create* BAPI of the business object type *CompanyCode* (key *CompanyCodeID*) must not have an import parameter *CompanyCodeID.* The name of the company code to be created could instead be transferred, for example, as field *ID* in the parameter *CompanyCodeData.*

Function module interfaces must also satisfy the conventions below:

- The structure BAPIPAREX must be used for the parameters *ExtensionIn* and *ExtensionOut*.
- The Return Parameter [Ext.] is based on the structure BAPIRET2.
- The function module must be RFC-enabled.
- Underscores ("_") are not allowed in parameter names.

Only upper case letters can be used in the Function Builder, so here the components of parameter names cannot be separated by the use of upper and lower case letters.

Implementing the Function Module

When programming the function module a range of BAPI-specific requirements must be met.

Transaction Model Requirements

- ☐ As of Release 4.0 BAPIs must not execute 'COMMIT WORK' commands.

  Reason: The caller should have control of the transaction. Several BAPIs should be able to be combined within one LUW. For more information see Transaction Model for Developing BAPIs [Ext.].

  The following commands must not be used:

  - CALL TRANSACTION
  - SUBMIT REPORT
  - SUBMIT REPORT AND RETURN

- ☐ Database changes can only be made through updates.

  Reason: The RFC executes an implicit database commit.

- ☐ The global memory must not be used to transfer values.

  Reason: This should avoid any side effects and support the encapsulation of the functionality.

- ☐ If a COMMIT WORK is not executed after a BAPI has been called, you have to delete the lock that has been set.

- ☐ All tables that are not protected by SAP internal locks must always be updated in the same order to prevent a deadlock.

Requirements of the Remote Capability of BAPIs

- ☐ BAPIs must not produce any screen output. This means that lists, queries and dialog boxes must not be created. This is true for the BAPI itself and for all function modules that may be indirectly called by the BAPI.

  Reason: An external client cannot respond to output on the screen. In the worst possible case, the connection in the client program could be broken.

- ☐ If required every BAPI must be able to carry out its own authorization checks. In particular Input Help [Ext.] should contain authorization checks.

Error Handling Requirements

- ☐ No exceptions are used in BAPIs. Instead all error messages are reported back

- BAPIs must not cause the program to terminate. An abort message (message

- Error handling plays a more important role here than it does with normal functio

- ☐ The meaning of error numbers that are used must not be changed.
- ☐ Error messages must be able to be understood by users in external applications. For example, they should not contain table names.
- ☐ The error message must specify whether the cause of the error is business (content) related or due to missing import data.

  Example of an inappropriate error message: "Mandatory field not filled". The error message must specify **which** mandatory fields have not been filled.

Performance Requirements

- The BAPI must perform well. The following guidelines shou

- The function module on which a BAPI is based can be accessed from external p

- ☐ **Handling Large Data Volumes**

  Mass data is treated differently in ABAP programs, which use the SAPGUI as a front end, and in programs developed on external development platforms such as Visual Basic.
  If large volumes of data are read in the R/3 System, for example a list containing many entries, the majority of the data remains on the application server. Only the data that is actually displayed is sent to the frontend. In contrast, if you are programming with Visual Basic, all the data will be sent from the application server to the client system. This increases the load on the network and the amount of memory required in the client system.
  You need to cover the situation when your BAPI has to read mass data. For example, you could specify a limit so that only the first *n* data records are read. Alternatively your BAPI could return a message to the calling program indicating that the amount of data has exceeded a certain limit and that a new selection should be made.

- ☐ To **reduce the duration of database locks**, you should not assign numbers to them individually. Instead, make use of the buffers. Read the numbers to a buffer and assign the numbers directly from the buffer.

- ☐ Lock periods can be minimized by updating the database as close as possible to the COMMIT WORK command. This reduces the duration of database locks and the risk of blocking other processes.

- ☐ The less specific the (partial) key of a modified data record is, the more likely it is that the data record will be accessed by multiple BAPIs, causing the record to be locked. For example, running a statistic on plant level will have a negative impact on the performance of the BAPI, whereas a statistic based on plant and material will cause fewer locks because it will apply to fewer BAPIs.

- ☐ Minimize the use of read transactions which depend on a previous database COMMIT (committed read). These read transactions have to wait for the COMMIT WORK command of the update transaction to be processed.

Further Requirements

- ☐ You must carry out a refresh for tables for which no input is expected.

- ☐ Do not use get and set parameters.

  Reason: This should avoid any side effects and support the encapsulation of the functionality.

- ☐ BAPIs should be independent from Customizing settings. In particular, values relevant for Customizing are not allowed to be changed.

- ☐ Customer exits must be provided so that customers can enhance BAPIs:

  Each BAPI function module must contain two extra exits in addition to the customer exits provided by the application.
  The first customer exit should enable the customer to check all the data passed to the BAPI. In particular, check the content of the ExtensionIn parameter before it is written to the table extensions.
  The second customer exit is provided so that the customer can make enhancements (e.g. writing additional tables). If the customer has enhanced the export parameter of the BAPI, the second customer exit should be used to fill the ExtensionOut Parameter [Ext.].

  See also Customer Enhancements to BAPIs [Ext.]

Actions in the BOR

After the function module has been created, you have to define the BAPI as a method of a business object type. By storing the BAPI in the central Business Object Repository (BOR), you enable object oriented access to the BAPI and the ALE distribution model may be able to be connected to (with asynchronous communication).

The BOR can be accessed from within the project form in the BAPI Explorer.

Process Flow

To define the BAPI:

1. Identify the SAP Business Object Type [Ext.].

2. Append the BAPI as an API method to the business object type. You can use t

- ☐ Upper case/lower case (each new word must be in upper case). See also: Example [Ext.].

- ☐ The import and export behavior of the table parameters must be correctly defined in the BOR.

  Reason: Unlike the function module, in the BOR you can differentiate between import and export tables. You should therefore only select the standard option *Import/export*, if the table is actually going to be imported **and** exported.

- ☐ The return parameter is always defined as an *export* parameter.

As well as making the task much easier, the Wizard also ensures that the BAPI

▪

▪

▪

The following Example [Ext.] shows the relationship between the definition of the BAPI method in the BOR and the function module.

If you cannot use the BAPI Wizard (e.g. to change the BAPI later or to redefine a method), you can change the method in the BOR itself. Make sure that the above points are complied with.

For developments undertaken by IBUs, customers and partners the **additional** points below are important:

- ☐ Read the documentation on Customer Enhancements and Modifications of BAPIs [Ext.]
- ☐ As a customer you can create your own BAPIs for your own business object types or you can define a Subtype [Ext.] of an existing SAP business object type. We strongly advise you NOT to define new BAPIs of SAP business object types because this involves modifications.
- ☐ All development objects (business object types, BAPIs, parameters) have to be created in their own Namespace [Ext.].

Documenting the BAPI

Principles

The quality of your BAPI documentation depends on the following principle:

The documentation must be in sufficient detail so that an external developer familiar with the business background but not with the R/3 System, can use the BAPI.

Features

The documentation for the BAPI covers four areas:

1. The SAP Business Object Type Documentation [Ext.].

2. The [Method Documentation [Ext.]] should answer the following questions:

- ☐ What is the business function of the BAPI and what is it used for?
- ☐ What do the BAPI functions actually do?
- ☐ Are there any important limitations, that is, are there functions that this BAPI cannot perform?
- ☐ What must you pay particular attention to with this BAPI? (e.g. authorization checks)
  What other prerequisites apply to the BAPI?
- ☐ Are there any Customizing dependencies?
- ☐ What dependencies are there between this BAPI and other BAPIs, and between individual method parameters?
- ☐ Is it a BAPI with buffering?

  If it is, the BAPI must be explicitly identified as such.
- ☐ Does the BAPI contain a COMMIT WORK command?

  If it does, it must be documented.

3. The BAPI Parameter Documentation [Ext.] answers the following questions:

- ☐ What is the parameter used for?
- ☐ Which fields of a parameter must be filled, that is, what are the **mandatory** fields?
- ☐ What are the dependencies between fields?
  Are there parameter dependencies and field dependencies within a structure?
- ☐ Are there any fixed values and what do they do?
- ☐ What are the default values of the parameter?
  All the fields that are assigned default values by Customizing and are therefore write-protected, must be documented.
- ☐ Does the documentation of the return parameter comply with the guidelines and does it contain all the relevant error messages?
- ☐ If there is a termination, is a database rollback executed as an exception within the BAPI?

  If it is, you must describe this process in the documentation for the return parameter.
- ☐ Are all the available BAPI table extensions listed in the documentation on the extension parameters (*ExtensionIn*, *ExtensionOut*)?

4. Data Element Documentation [Ext.] underlying the fields in the BAPI paramet

Finally, you must also make sure that the function module and parameter documentation has been saved by the documentation developer in *active version* so that it appears in the translator's work list.

BAPI/ALE Integration

BAPIs are integrated into the ALE communication model. ALE communication is asynchronous and message-based. ALE communication is the preferred way to integrate distributed R/3 Systems, for example, for distributing master data.

Since Release 4.0 **BAPIs are the standardized interfaces for ALE-supported communication**. ALE services, such as asynchronous BAPI calls, distribution model, monitoring and error handling can be used for BAPIs. The IDoc types required for the ALE services can be generated from BAPIs.

See also: Using ALE Services [Ext.].

When you use BAPIs for asynchronous messaging, the application in the sending system calls the generated ALE IDoc interface instead of the BAPI. The ALE IDoc interface performs the following tasks:

- Creates an IDoc from the BAPI data
- Sends the IDoc to the target system
- Receives the IDoc in the target system, creates the BAPI data from the IDoc and calls the BAPI

You can use Transaction BDBG to create the other objects required for the asynchronous communication (message type, IDoc type, inbound and outbound function module).

See also: Maintaining the BAPI-ALE Interface [Ext.]

A BAPI should only be implemented as an asynchronous interface, if at least of the following conditions is satisfied:

- *Database changes are consistent in all systems*
  Data must be updated consistently on a remote system as well as on the local system.
- *Looser coupling*
  With a synchronous interface the coupling between the client and the server system is too narrow. If the connection is interrupted the client system would not be able to function properly.
- *Performance*
  The interface has a large volume of data or database operations to be carried on the server system will take a long time. In this situation a synchronous interface cannot be used because performance would be too low.

BAPI/ALE integration requires that:

- ☐ Message types are created and IDoc types are generated for all write BAPIs.
  See also: Generating the BAPI-ALE Interface [Ext.]
- ☐ All IDoc types and the associated segments are released.
- ☐ The BAPI interface and the IDoc interface are identical. Make sure that the message type, IDoc type and segments are regenerated, particularly if the BAPI is changed.

All BAPI export parameters with the exception of the return parameter are ignored and are not included in the IDoc type that is generated.

SAP Internal:

If the BAPI is used to integrate different SAP products, you have make the interface information available in the relevant systems. To do this you have to move the data structures (including data elements and domains) used by the BAPI as well as the ALE interface, if necessary, into the application basis.

Provided that the applications involved have restricted usability (componentization, e.g. AC/LO and HR, some R/3 New Dimensions), the BAPI function module must be explicitly released for cross-application use.

Please consult your BFA contact person.

Testing and Releasing

To check that the BAPIs, business object types and documentation are all correct, special tests and ToDo checks must be carried out. Once the test phase has been successfully completed, the business object type, BOR methods and function modules can be released.

For more information see:

Testing [Page 74]

Releasing [Page 77]

Testing

After you have implemented the underlying function module of your BAPI, and you have defined the BAPI as a method of an SAP business object type in the Business Object Repository (BOR), you should test the BAPI.

Carry out the test together with the persons responsible for quality control in your development group.

Test Phase

The following steps are carried out in the test phase:

- **Testing the Documentation**

  Check that the documentation for each business object type, each BAPI and for each interface parameter is available and that you understand it. As the BAPI documentation is critical for being able to use the BAPI, this test should be carried out thoroughly.

- **Testing the BAPI Syntax**

  In the test phase you must check that the BAPI meets all the conventions described above, and that you have followed the guidelines for developing BAPIs. You can check that the syntax is correct using the BAPI Explorer and the BAPI ToDo. An implicit check is also carried out when the BAPI is appended in the BAPI/BOR Wizard and when it is released in the BOR.

  To test the BAPI syntax in the BAPI Explorer, position the cursor on the relevant BAPI in the left-hand frame. Select the tab page *Tools* in the right-hand frame and then the option *BAPI Consistency Checks.*

☐ Testing the BAPI Semantics

After you have checked that the syntax is correct, you have to check that the semantics of the BAPI are correct (testing the functions and integrity). You have the following options:

Testing the underlying function module in the Function Builder

You can test the parameters in your function module in one test. Enter the appropriate test values in the parameters to verify that the source code in the function module can run without errors.

However, the test in the Function Builder has the following drawbacks:

- The tests do not update the database because a COMMIT WORK cannot be executed.
- The tests cannot be automated.
- The test data is not transported into other systems and is lost if the function module is changed. So, for example, no regression tests can be carried out.

Testing the function module with the Computer Aided Testing Tool (CATT)

Function modules can be tested within the SAP standard test tool using the CATT test module type **F**. The CATT has the following advantages over testing in the Function Builder:

- The tests do change the database because a COMMIT WORK is automatically executed.
- The tests can be automated.
- The test data is transported with the test module and is therefore available in other systems and releases for regression tests.

Tests with the CATT should be carried out in preference tests in the Function Builder because the test runs are retained and can be optimally integrated into the SAP quality assurance process.

For more information on using CATT for testing function modules, see CATT: Using Function Module Tests [Ext.].

☐ Testing the BAPI call in an external application

To use a BAPI in an external application, the following conditions must be fulfilled:

- The syntax of the BAPI is correct.
  This should be tested by the BAPI developer using the BAPI Explorer.
- The communication functions without errors.
  This should be checked using appropriate tests from the Middleware departments.

☐ Testing the BAPI on different platforms

The platform tests are carried out during the final assembly. The tests can only be carried out when the BAPI semantics have been tested with CATT, and when the CATT procedures created in the tests can be used again in the final assembly.

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

Releasing

Once all the tests have been completed, the BAPIs and all their associated development objects can be released for customers to use. But they can only be released if the following conditions are fulfilled:

- The documentation has been fully completed.
- There are no consistency errors.
- The BFA AG contact person has authorized the release.
- The quality manager has authorized the release.

For more information about releasing BAPIs see also Other Issues [Page 37].

Process Flow

You have to carry out the following steps:

- Release the BAPI Function Module [Ext.] in the Function Builder.
- Release the Business Object Type [Ext.] in the BOR.
- Release the BAPI [Ext.] as a method in the BOR.
- With write BAPIs release the IDoc and its segments.