BAPI ActiveX Control

HELP.BCFESDE8

Release 4.6C

SAP™

Copyright

Icons

Icon	Meaning
⚠	Caution
	Example
	Note
	Recommendation
	Syntax

Contents

BAPI ActiveX Control

BAPI ActiveX Control

Definition

The BAPI ActiveX control enables external applications to access business functions in the R/3 System by calling BAPIs (Business APIs) through OLE Automation. Client programs access proxy objects, which are local instances of SAP business objects managed by the BAPI ActiveX control. These proxy objects correspond to the real SAP business objects stored in R/3's Business Object Repository (BOR).

The details of this process are invisible to the client program because the data structures exported from the R/3 System through the BAPI ActiveX control are encapsulated in objects. However, accessing objects is easy because they look and feel native to local desktop objects.

Use

Before your client program can access SAP business objects through the BAPI ActiveX control, it must create a local instance of the BAPI ActiveX control and declare an object variable. It can then request local instances of business objects.
When created on the desktop, local instances of SAP business objects dynamically adapt their interfaces to the interfaces of the real SAP business objects, which they represent. Interfaces of business objects are thus only known at runtime.

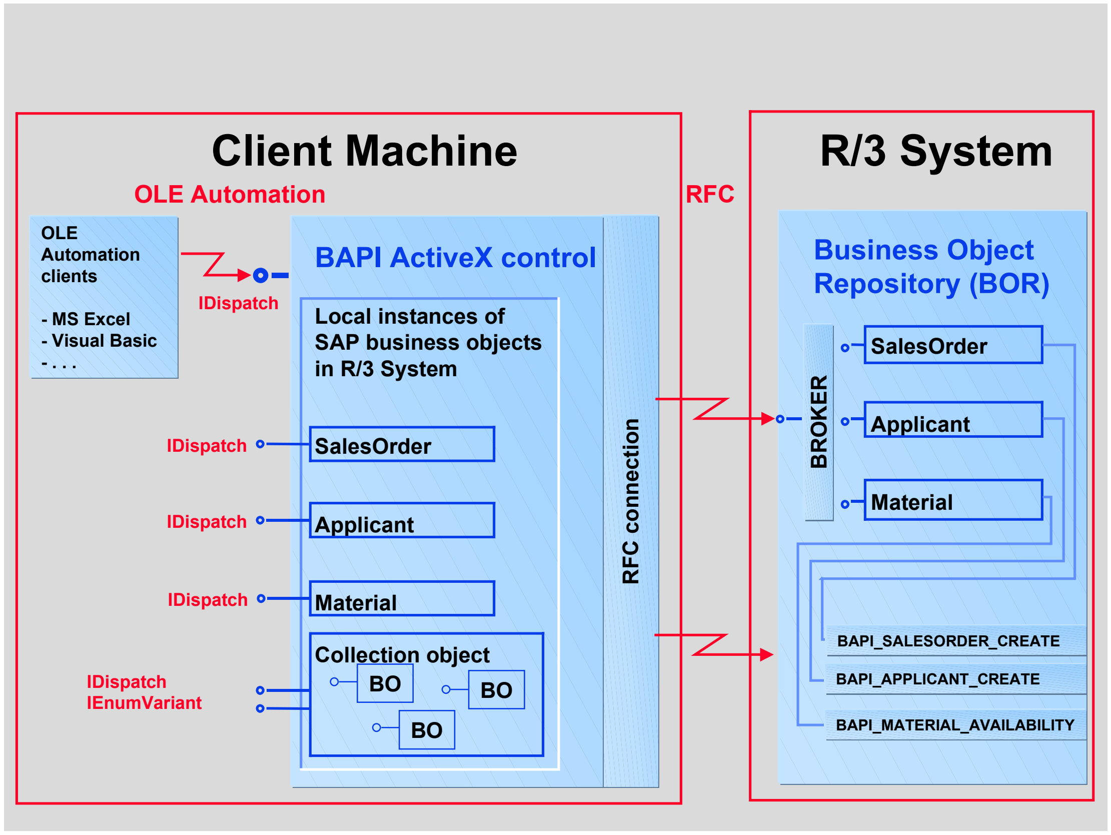
Even if you are unfamiliar with object-oriented programming, the BAPI ActiveX control is still easy to use. If you are familiar with languages like Visual Basic, you will not even notice that you are dealing with a remote R/3 System. Client programs may be written in Visual Basic or any other language that supports OLE Automation.

For an introduction to this technique, see a simple example in Visual Basic in the corresponding unit of Tutorial: Communication Interfaces [Ext.].

Structure

Client programs access SAP business objects by making OLE Automation calls to the BAPI ActiveX control, which forwards the requests to the R/3 System via Remote Function Call (RFC), as shown in the following graphic:

BAPI ActiveX Control



When a client program calls a method to manipulate a business object, the BAPI ActiveX control looks for an appropriate method:

- If the called method is a BAPI, the BAPI ActiveX control calls the relevant function module in R/3 directly, using the Function ActiveX control
- If the called method is not a BAPI, access is through the BOR runtime

Making OLE Automation calls to BAPIs directly is faster, because it avoids the overhead involved in calling a method from the BOR.

Local Object Hierarchy

Since the BAPI ActiveX control manages access to all other objects, the client program must first create a local instance of the BAPI ActiveX control. Only then is it possible to access business objects and process them by reading properties and calling methods.

The BAPI ActiveX control manages a local hierarchy of business object instances. These include:

- Business objects
- Collection objects

Business Objects

You create business objects:

- Explicitly in the client program
- Implicitly through the BAPI ActiveX control as a return value when calling a method

BAPI ActiveX Control

When you create a business object, the client program - or the Business Object Broker (BOB) - specifies the object type needed. Once created, each business object responds only to the properties and methods of its own object type, as specified in the BOR.

Business objects are only “alive” while the BAPI ActiveX control object that created them remains logged on to the R/3 System. If an unhandled exception occurs in the R/3 System, the RFC connection may be broken. You must then reconnect to each object.

Collection Objects

Collection objects are sets of business object instances, which you process together. There are methods for iterating over the objects in a set.

Integration

The BAPI ActiveX control is currently implemented to run under Windows95 and WindowsNT.

Business Objects

Business objects are real world entities modeled as objects in an information system.

Business objects **encapsulate** both data structures and the functions applied to the data, but hide their full complexity from other objects. This encapsulation of data and functions makes it easier to modify program components, because you can program with the relevant entities without having to know all the implementation details. You can also reuse existing functions.

Client programs access business objects by reading their **attributes**, or by calling the **methods** that make up the object's **interface**:

- Attributes

  Attributes describe the data stored in an object through a set of properties. They provide direct read access to the data structures of objects, but client programs cannot change them from outside.

- Methods

  Methods provide a way to encapsulate the data structures of business objects, and to process them. When accessing an object, the client program calls a method with parameters and gets back return parameters.

- Interface

  The interface is the set of methods associated with a business object, and determines how an object interacts with the outside world.

The client program defines the object types to be used and, at runtime, creates object instances of those object types.

SAP business objects are managed in R/3's Business Object Repository (BOR).

BAPIs

BAPIs (Business APIs) are special methods that are applied to SAP business objects stored in R/3's Business Object Repository (BOR), in order to perform specific business tasks such as creating a sales order or checking the availability of a material.

In R/3, BAPIs are implemented and stored as RFC-enabled function modules in the Function Builder of the ABAP Workbench. They have standard business interfaces, which enable external applications to access R/3 processes, functions and data through SAP business objects.

Client programs that use BAPIs to access SAP business objects can be part of the R/3 System, part of an external system like a Visual Basic frontend, an HTTP gateway, or another R/3 System:

- Within R/3, a client program can access a business object by requesting a method call from the Business Object Broker (BOB)

  The method used in this case may or may not be a BAPI.

- An external client program can call a BAPI directly through the BAPI ActiveX control by creating an instance of a business object

  In this case, the BAPI ActiveX control looks for an appropriate BAPI to process the object:

  – If a BAPI exists for the purpose, the BAPI ActiveX control calls it directly.

  – If no BAPI exists for the purpose, access is through the BOR.

Making OLE Automation calls to BAPIs directly through the BAPI ActiveX control is faster than access through the BOR runtime system, because it avoids the overhead involved in requesting a method call from the BOB.

Business Object Repository

The Business Object Repository (BOR) is a development and runtime environment in the R/3 System that manages all SAP business object types and their methods. Business object types include:

- Business objects

  Business objects are objects such as “Customer”, “Material”, or “CompanyCode”. They provide both a high-level business-oriented view of, and a programming interface to, the R/3 System.

- Technical objects

  Technical objects are texts, notes, work items and archived documents, as well as desktop objects like texts, graphics and spreadsheets. Desktop objects can be described in condensed form in the BOR.

- Metaobjects

  Metaobjects comprise object types, methods, attributes and events. Each object has an attribute “ObjectType” which refers to the metaobject to which it is assigned. The methods, attributes and events available for a particular object can be retrieved from its “ObjectType”.

You can use the functions available in the BOR to create new object types or change existing ones. To encapsulate objects from your own system, or from any other external system, the BOR also allows you to define your own object types.

The core of the BOR runtime system is the Business Object Broker (BOB). Client programs send requests to the BOB, which:

1. Creates the addressed runtime object.
2. Chooses the correct method implementation.
3. Calls the method found.
4. Transports the results back to the caller.
5. Destroys the runtime objects.

Called methods can themselves request method calls.

For information on browsing SAP business objects in the BOR, see:

Browsing SAP Business Objects [Page 12]

Browsing SAP Business Objects

SAP business objects and their associated methods - BAPIs or non-BAPIs - are managed in R/3's Business Object Repository (BOR) and stored in a structure based on the business application hierarchy.

Procedure

To browse SAP business objects in the BOR, use the Business Object Browser:

1. Log on to R/3 through the SAPgui

   You only need the SAPgui while you are programming. It is no longer required when the client program is running (provided the business objects you are using do not send screens).

2. Choose *Tools* → *ABAP Workbench* → *Overview* → *Business Object Browser*
3. Expand the nodes for any application until you reach the level at which business objects are attached
4. Select a business object by double-clicking on it
5. Expand the *Methods* node

   You see a list of methods for the object. The methods with a green light to the right of the method name are BAPIs.

6. If you are browsing for BAPIs, click the green light next to the desired BAPI name

   You see the source code of the BAPI function module.

7. To get parameter information, choose *Goto* → *Import/export parms*
8. To display the individual fields of a table, together with field names, types, and lengths, double-click on one of the reference fields

   Before you use an object type, you must ensure that it has been released for operational use. Some of the object types in the BOR are modeling objects and are therefore not yet implemented.

If you want to browse just those SAP business objects for which BAPIs are implemented, use the BAPI Browser. See:

Using the BAPI Browser [Page 13]

Using the BAPI Browser

To browse just those SAP business objects for which BAPIs have been implemented, use the BAPI Browser, which is a BAPI-oriented view of the Business Object Repository (BOR):

1. Log on to R/3 through the SAPgui
2. Choose *Tools* → *Business Framework* → *BAPI Browser*
3. Expand the nodes for any application until you reach the level at which business objects are attached
4. Select a business object by double-clicking on it
5. Expand the node *Methods*

   You see a list of all the BAPIs implemented for the selected business object.
6. To get parameter information, expand the *Parameters* node
7. To display documentation for a BAPI, click on the appropriate information icon

BAPI ActiveX Control: User's Guide

Purpose

This process lists the tasks that any external client program should perform when using the BAPI ActiveX control to access SAP business objects in R/3 and/or manipulate the data structures of those business objects with BAPIs.

Process Flow

1. Create the BAPI ActiveX control object - an instance of the BAPI ActiveX control
2. Establish a connection to the R/3 System, and log on
3. Create one or more business object instances
4. Access business object properties and/or call BAPIs to manipulate business objects
5. Release business objects
6. Log off from the R/3 System
7. Release the BAPI ActiveX control object

For a simple program example, see the corresponding unit of Tutorial: Communication Interfaces [Ext.].

Creating the BAPI ActiveX Control Object

Prerequisites

Before creating the BAPI ActiveX control object in the client program, you must activate the BAPI ActiveX control in your Visual Basic project:

1. Choose *Project → Components*
2. Select *Controls*
3. Select *SAP BAPI Control*

   This adds the BAPI ActiveX control object to the Visual Basic toolbox.

Procedure

There are 2 ways to create the BAPI ActiveX control object:

- Drag the BAPI ActiveX control from the Visual Basic toolbox to the project form
- Specify the following lines of code:

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

Logging on to R/3

Prerequisites

Before you can access SAP business objects, and/or call BAPIs to manipulate those objects, you must:

1. Establish a connection to the R/3 System
2. Log on to the R/3 System

If the connection is not in the "logged on" state, any attempt to access business object properties or call methods will raise an exception.

Procedure

Establishing a Connection

You can either:

- Set the connection directly through the Logon ActiveXControl

  The Logon ActiveX control encapsulates the connection process and provides a connection object in the form of the logon object's *NewConnection* method.

- Set the connection implicitly through the BAPI ActiveX control

  The connection object of the Logon ActiveX control is also a property of the BAPI ActiveX control.

Neither of the above options necessarily guarantees that a connection to the R/3 System has been established.

To check the status of the connection object and the R/3 connection, look at the *IsConnected* connection property.

Logging on to R/3

Perform the logon to the R/3 System, using the *Logon* method of the connection object:

- If the value of the second parameter of the *Logon* method is '*False*' and not all parameters containing logon information have been set, a dialog box is displayed for the user to enter the missing values
- If the value of the second parameter of the *Logon* method is '*True*' and all parameters containing logon information have been set, there is no need to display a dialog box to the user. In this case, a "silent" logon is performed

  You can predefine logon information by maintaining the user and system properties of the BAPI ActiveX control in Visual Basic's design environment.

Setting the Connection Directly Through the Logon ActiveX Control

Visual Basic code	Comments
`Set oLogonCtrl = CreateObject("SAP.LogonControl.1")`	Creates the Logon ActiveX control
`Set oBAPICtrl.Connection = oLogonControl.NewConnection`	Creates a new connection and sets it in the BAPI ActiveX control. This does not necessarily mean that a connection to the R/3 System has been established.

`oBAPICtrl.Connection.Client = "<client>"`
`oBAPICtrl.Connection.User = "<user name>"`
`oBAPICtrl.Connection.Language = "<language>"`
`...`
	Provides logon information for the R/3 System.

Setting the Connection Implicitly Through the BAPI ActiveX Control

Visual Basic code	Comments

`oBAPICtrl.Connection.Client = "<client>"`
`oBAPICtrl.Connection.User = "<user name>"`
`oBAPICtrl.Connection.Language = "<language>"`
`...`
	Creates a new connection object.
	Provides logon information for the R/3 System.

`oBAPICtrl.Connection.Logon (0, FALSE)`	Opens the connection to the R/3 System. A dialog is displayed for the user to enter the missing logon parameters.

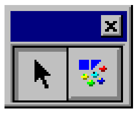
Predefining User and System Logon Information

Procedure

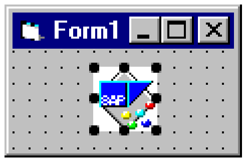
To predefine user and system logon information, maintain the properties of the BAPI ActiveX control in Visual Basic's design environment.

The following pictures show the appearance of the BAPI ActiveX control object in the Visual Basic toolbox and how it looks when placed on a form.

BAPI ActiveX Control in Visual Basic Toolbox

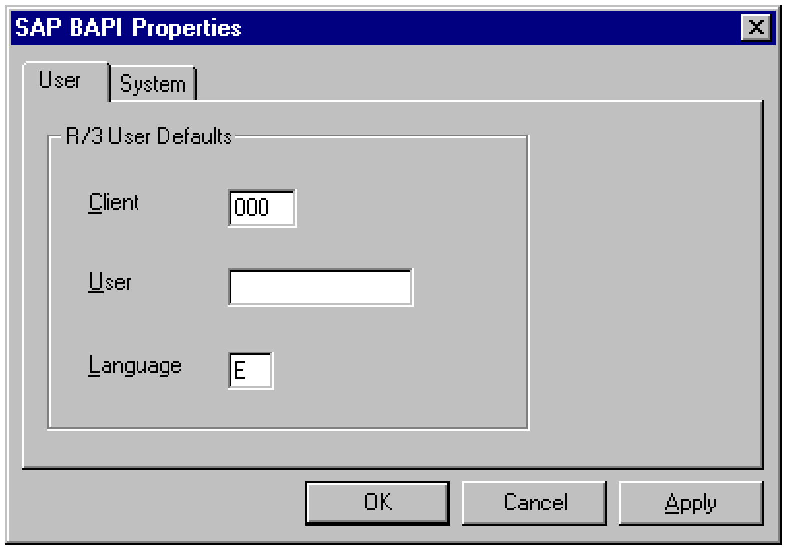


BAPI ActiveX Control on Visual Basic Form



When you place the BAPI ActiveX control object on a form, you can edit the properties with the right mouse button. The following pictures show the layout of the property pages:

User Page



SAP BAPI Properties

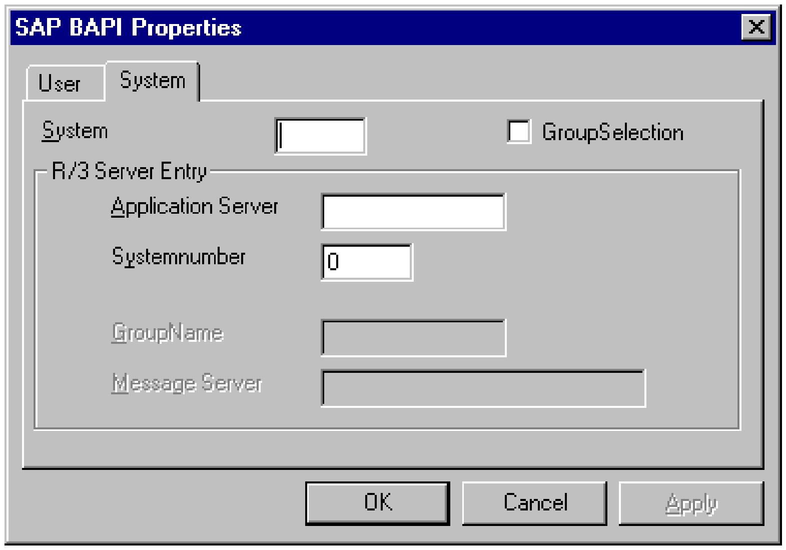
User | System

R/3 User Defaults

Client	000
User	
Language	E

OK Cancel Apply

System Page



System	
GroupSelection	☐
R/3 Server Entry	
Application Server	
Systemnumber	0
GroupName	
Message Server	

Creating Business Objects

Prerequisites

Before you can create local instances of SAP business objects, you must create the BAPI ActiveX control object, and log on to the R/3 System.

Procedure

To create local instances of SAP business objects, use the *GetSAPObject* method.

GetSAPObject has the following syntax:

```
Object.GetSAPObject(String ObjectType, Variant ObjectKey1,..., Variant
ObjectKey10)
```

GetSAPObject has the following parameters:

- *ObjectType*

  This parameter identifies the SAP business object. It is a case-sensitive string which matches an object type defined in the Business Object Repository (BOR).

  You can specify either the name of the business object type or its technical name. For example, the *SalesOrder* business object type has the technical name *BUS2032*. Both names are unique identifiers of the same business object.

- *ObjectKey1... ObjectKey10*

  The *ObjectKey* parameters uniquely identify a persistent instance of the SAP business object to which you want to connect.

  If an object type has several key attributes, each *ObjectKey* parameter corresponds to one key attribute. The order of the *ObjectKey* parameters must be the same as the order of the key attributes defined in the BOR.

  The *ObjectKey* parameters are **optional**.

  You have to pass *ObjectKey* parameters if:

  – You want to connect to a persistent instance of an SAP business object

  – You are creating a **new** instance of a persistent object

  – You are creating an object with only one instance

    When you do this, a check is first performed to ascertain whether a business object with the specified key exists. If so, an instance of the business object is created and initialized with data read from the database.

  You do not need to pass *ObjectKey* parameters if:

  – You are creating a **new** instance of a persistent object

  – You are creating an object with only one instance

Connecting to a Persistent Instance of an Object Type

Create a local instance of SAP business object *SalesOrder* with sales document number 784:

```
Set boOrder = oBAPICtrl.GetSAPObject("SalesOrder",
"0000000784")
```

The object is initialized with data read from the R/3 database.

Creating a New Instance of a Persistent Object

In this case, **new** means that no such object already exists in the R/3 database.

1. Create an anonymous business object without specifying any key values
2. Construct the object by setting properties and calling methods
3. Commit the object with the method provided for the purpose

   This method - usually called *Post, Submit, Create* or *Commit* - writes the objects to the R/3 database. At this point, it is given a key you can use for further reference.

Create an anonymous business object of the type *SalesOrder*

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

Get table/structure objects of the exported parameters and fill in data

```
...
```

Commit business object *SalesOrder* and write to the R/3 database

```
boOrder.CreateFromData OrderHeaderIn:=oHeader, _
                      OrderPartners:=oPartners, _
                      OrderItemsIn:=oItemsIn, _
                      Return:=oReturn
```

Get unique key for instance. In this case, it is the sales document number

```
OrderNumber.Caption = boOrder.SalesDocument
```

Creating an Object Type With Only One Instance

Create an instance of the SAP business object *SCHEDULER*.

```
Set boScheduler = oBAPICtrl.GetSAPObject("SCHEDULER")
```

There is only one instance of this object and it has no key.

Passing Data to BAPIs

Prerequisites

BAPIs expect a large number of structured data types and/or tables. To help you pass properly typed parameters when calling them, the R/3 desktop integration components therefore provide a set of objects encapsulating structures and tables.

Procedure

To create the objects for passing data easily to BAPIs, use the *DimAs* method.

The *DimAs* method has the following syntax:

```
Object.DimAs(Object BusinessObject, String Method, String Parameter)
```

DimAs has the following parameters:

- *BusinessObject*

  This is the name of the object used in the application to identify the SAP business object. It is returned from a previous call to *GetSAPObject*.

- *Method*

  This is the name of the BAPI to which the parameter belongs. It is a case-sensitive string and must match one of the BAPIs for the SAP business object *Object*, as defined in the BOR.

- *Parameter*

  This is the name of the parameter whose structure or table object is retrieved. It is a case-sensitive string and must match one of the parameter names of the BAPI *Method*, as defined in the BOR.

For each BAPI, the BAPI ActiveX control maintains a collection of R/3 tables represented by a tables collection object. To get the definition of a specific table or structure, call the *DimAs* method and specify the business object type, the BAPI method and the parameter name. This returns the table or structure object requested. The object hides the internal table passed from the native RFC interface and handles the data read from or written to the R/3 System. It also provides a full two-dimensional view of the internal table.

You can find the expected type in the parameter definition of the requested BAPI in the Business Object Repository (BOR).

1. Get the *OrderPartners* table object with the *DimAs* method

```
   Set oPartners = oBAPICtrl.DimAs(boOrder, "CreateFromData",
"OrderPartners")
```

2. Set values of parameter *OrderPartners*

```
   oPartners.Rows.Add
   oPartners.Value(1, "PARTN_ROLE") = "AG"
   oPartners.Value(1, "PARTN_NUMB") = "0000010096"
```

3. Call the method

```
boOrder.CreateFromData OrderHeaderIn:=oHeader, _
                      OrderPartners:=oPartners, _
                      OrderItemsIn:=oItemsIn, _
                      Return:=oReturn
```

From the point of view of the calling program, only **export parameters** have to be **initialized** with *DimAs*. The data can then be filled in the retrieved table object or structure object. These parameters are defined as import parameters of the BAPI function module.

You can pass **uninitialized** objects to parameters that receive return values. These parameters are defined as export parameters of the BAPI function module.

Logging Off From R/3

Prerequisites

Before logging off from the R/3 System, you should release the business objects.

Procedure

To log off the R/3 System, call the connection object's Logoff method and close the established connection with the following lines of code:

```
oBAPICtrl.Connection.Logoff
Set sapConnection = Nothing
```

Result

Once you have logged off, unreleased business objects are disabled. Any attempt to call a method or access a property raises an exception.

Performing Transactional RFCs With BAPIs

BAPIs, like all function modules, should be executed only once in the R/3 System. For this reason, client programs must exploit the transactional RFC capability of the BAPI ActiveX control by managing the transaction IDs of BAPI calls themselves.

If a network (CPI-C) error occurs during the course of a BAPI call, an exception is raised (error 8793). The client program can, of course, repeat the call, but there is a risk that the BAPI may be executed a second time:

- If the error occurs when **calling** a BAPI, the client program can safely call the BAPI method again
- If the error occurs **during or after** the BAPI call, the BAPI may already have been executed, so repeating the call is not recommended

Since each database transaction or Logical Unit of Work (LUW) can contain only one BAPI call, each BAPI call needs its own transaction ID. To enable this, the BAPI ActiveX control provides the following methods and properties:

- *CreateTransactionID* method

  This method returns a transaction ID created in the R/3 System. The transaction ID has to be stored together with the data passed to the next BAPI call.

- *TransactionID* property

  This property contains the transaction ID used in the next BAPI call. If the method is successfully executed, the BAPI ActiveX control itself deletes the transaction ID.

If an error occurs during or after the BAPI call, your client program must

1. Reconnect later.
2. Restore the data saved with the 'old' transaction ID assigned to the *TransactionID* property of the BAPI ActiveX control.
3. Repeat the call, using the 'old' transaction ID.

   The program must not create a new transaction ID (with *CreateTransactionID*), because the BAPI method could then be executed a second time.

   If the call is successful, the transaction is complete.
4. Update transaction ID management.

The following Visual Basic code calls the *WriteData* method of the business object *TestObject*. The *WriteData* method has just one table parameter called *Data*.

The code assumes that the BAPI ActiveX control object has already been created and that a connection to R/3 has been established. Also, the code does not include looking up the transaction ID management and repeating calls that have already failed, because a client program normally takes these steps immediately after startup to ensure that all BAPI calls have been performed once.

```
'Declare object variables
```

```
Dim oTestObj As Object
Dim otabData As Object
Dim strTID As String

'Create a business object of type 'TestObject'

   Set oTestObj = oBAPICtrl.GetSAPObject("TestObject")

'Get the 'Data' table object

   Set otabData = oBAPICtrl.DimAs(oTestObj,"WriteData","Data")

'Fill table with data

   :

'Create transactinID, which is automatically stored in the property TransactionID
'of the BAPI control

   strTID = oBAPICtrl.CreateTransactionID

'Save transaction ID together with data and other information (object type,
'method name) needed to make call

   :

'Call BAPI transactionally - the transaction ID stored in the property
' TransactionID is taken for the following call

   oTestObj.WriteData  Data:= otabData

'Succeeded

   Exit Sub

ErrorHandler: 'Analyze error, clean up
```

For further information on transactional Remote Function Calls, see Using Transactional Remote Function Calls [Ext.].

Creating Collection Objects

Collection objects are used only with attributes or non-BAPI methods of SAP business objects. It was necessary to introduce collection objects, because some versions of Visual Basic have difficulty in handling arrays or sequences of OLE objects. To avoid problems arising from this, the BAPI ActiveX control always returns collections of business objects wrapped in a collection object. Similarly, the parameters it expects are collections of business objects wrapped in a collection object.

In the following code, *oAppointments* is a collection object returned from a *SCHEDULE* object by the *AppointmentsGet* method:

```
Set oAppointments = boSchedule.AppointmentsGet(datToDay,
datToDay)
```

In the following code, *oParticipants* is a collection object that contains *USR01* objects and is passed as a parameter to the *Create* method of an *APPOINTMNT* object. (The object type for the appointment object is written **without** *E*).

```
boAppointment.Create datFrom, datTo, strType, strRoom, "",
                     strNote, strVisibility, strTyp,
datFrom, oParticipants
```

The *AppointmentsGet* method is not a BAPI, since collections of business objects are not used when programming with BAPIs

The collection object also supports *For Each... Next* loops in Visual Basic.

```
For Each boAppointment In oAppointments
    lstAppointments.AddItem(boAppointment.Room)
Next
```

Debugging Hints

Prerequisites

Before debugging your application, you must switch on debugging traces. By default, all traces are switched off and should only be switched on during the actual debugging process.

Procedure

Tracing the BAPI ActiveX Control

For this purpose, the BAPI ActiveX control provides the following properties:

- *LogFileName*

  You use this property to set the the log file name to a value other than the default.

  If you do not set a value, the trace files **dev_bapi.trc** and **dev_func.trc** are written to the current directory:

  dev_bapi.trc contains traces of the BAPI ActiveX control.

  dev_func.trc contains traces of the Function ActiveX control.

  If you set *LogFileName* to a value other than the default, one log file that matches the passed string and one that consists of the extracted path and the **dev_func.trc** are written.

- *LogLevel*

  You use this property to specify the complexity of trace required. Its possible value range is from 0 to 9.

  If you set *LogLevel* to a value greater than 0, some form of trace is enabled. The higher the value, the more detailed the trace information you get.

Tracing the RFC Call

If you want to perform a trace of connection activity at RFC level, set the *TraceLevel* property of the connection object before connecting to the R/3 System.

Possible values of *TraceLevel* are:

- 0 (trace not requested)
- 1 (trace requested)

The connection activity information is logged in a file called **rfc<n>_<n>.trc** (where n is any number). This file is located in the current default directory.

Debugging traces are particularly helpful for logon problems and version problems at RFC level.

Handling Errors and Exceptions

To ensure that every occurrence of an error is recorded in the log files, set the *LogLevel* property to a value greater 0.

Error	Procedure
Logon method of connection object fails.	Check status of connection object and R/3 connection by using the *IsConnected* property of the connection object.
Exception occurs during a BAPI call (exception 8820).	Determine whether just the exception ID should be returned as error description by setting the *BAPIExceptionCodes* property of the BAPI ActiveX control to 'True'.
Remote Function Call (RFC) error occurs either when a BAPI raises an exception or when an internal error occurs in the BAPI ActiveX control.	Process this with an error handler.

The following code is an example of an error handler written in Visual Basic:

```
On Error GoTo ErrorHandler

   Set boOrder = oBAPICtrl.GetSAPObject("SalesOrder")
   Set oPartners = oBAPICtrl.DimAs(boOrder, "Simulate",
"OrderPartners")
...
Exit Sub

ErrorHandler:

If Err.Source = „SAP.BAPI"
                           'then an error occurred in the
                           'SAP Business Object Control
                           'handle it here (Exit Sub, End,
                           'Resume,...)
   Select Case Err.Number
      Case 8792
        ...
      Case...
        ...
   End Select
Else
                           'another type of error occurred
                           'handle it here (Exit Sub, End,
                           'Resume,...)
       ...
Endif
```

```
        End Sub
```

BAPI ActiveX Control Exceptions

The exceptions defined for the BAPI ActiveX control are listed in the following table. You can get the exception number and an error description from the error object maintained by Visual Basic.

BAPI ActiveX Control Exceptions

Error	Description
8792	Internal RFC error. Error key: RFC error key. Description: RFC short text message. CPIC status: CPIC error code. Internal state: RFC internal state.
8793	RFC exception raised. Connection closed by the system. R/3 error message: Description of the error. Error key: RFC error key. Description: RFC short text message. CPIC status: CPIC error code. Internal state: RFC internal state.
8794	(No longer used.)
8795	Unable to create runtime object of type **`<object type>`** with persistent key **`<object key>`** in Business Object Repository (BOR). Message number: Message number of T100 message in the R/3 System. Work area: Work area of T100 message in the R/3 System. R/3 error message: Description of the error. **Explanation:** A temporary error, an application exception or a Business Object Broker (BOB) exception has occurred in the R/3 System.
8796	Unable to retrieve persistent key of runtime object of type **`<object type>`** from Business Object Repository (BOR). Message number: Message number of T100 message in the R/3 System. Work area: Work area of T100 message in the R/3 System. R/3 error message: Description of the error. **Explanation:** A temporary error, an application exception or a Business Object Broker (BOB) exception has occurred in the R/3 System.

8797	Unable to set persistent key **`<object key>`** of an existing business object instance of type **`<object type>`** in Business Object Repository (BOR). Message number: Message number of T100 message in the R/3 System. Work area: Work area of T100 message in the R/3 System. R/3 error message: Description of the error. **Explanation:** A temporary error, an application exception or a Business Object Broker (BOB) exception has occurred in the R/3 System.
8798	Unable to release runtime object of type **`<object type>`** with persistent key **`<object key>`** in Business Object Repository (BOR). Message number: Message number of T100 message in the R/3 System. Work area: Work area of T100 message in the R/3 System. R/3 error message: Description of the error. **Explanation:** A temporary error, an application exception or a Business Object Broker (BOB) exception has occurred in the R/3 System.
8799	Unable to retrieve type information of business object type **`<object type>`** from Business Object Repository (BOR). Message number: Message number of T100 message in the R/3 System. Work area: Work area of T100 message in the R/3 System. R/3 error message: Description of the error. **Explanation:** A temporary error, an application exception or a Business Object Broker (BOB) exception has occurred in the R/3 System.
8800	Error occurred in R/3 System while calling the object type **`<method>`**. Message number: Message number of T100 message in the R/3 System. Work area: Work area of T100 message in the R/3 System. R/3 error message: Description of the error. **Explanation:** A temporary error, an application exception or a Business Object Broker (BOB) exception has occurred in the R/3 System.
8801	Object type **`<method>`** calls back via RFC, but this is not supported. **Explanation:** Method implementation makes an RFC call back to destination.
8808	Unauthorized access of member **`<object type>.<method/property>`**.
8809	Method or property access mismatch for **`<object type>.<method/property>`**.
8810	No connection to an SAP system.
8811	Attempt to access an object no longer connected to an R/3 System.
8812	Object passed to function is not an initialized business object.
8813	Object passed to function must have the same connection as the control.
8814	**`<method>`** is not a BAPI.

8815	Parameter **`<parameter>`** of method **`<method>`** is not a structure. Use elementary data types.
8816	Creation of Logon ActiveX control object, or Function ActiveX control object, or Table Factory ActiveX control object, failed.
8817	Error accessing persistent object key parameter in BAPI **`<method>`**.
8818	Persistent key for business object instance of type **`<object type>`** not set. Unable to call BAPI **`<method>`**.
8819	Error accessing parameter **`<parameter>`** or associated collection object.
8820	Exception raised in the R/3 System while calling BAPI **`<method>`**. Exception: Type of exception that occurred.
8821	Invalid type information for object type **`<object type>`**.

Example Applications

The following example applications, written in Visual Basic, demonstrate most of the programming techniques you encounter with the BAPI ActiveX control:

- Example: Calling BAPIs [Page 34]
- Example: Handling Collections of Business Objects [Page 36]

To run these example applications, all the necessary controls - BAPI ActiveX control, Logon ActiveX control, Table Factory ActiveX control and Function ActiveX control - have to be installed and registered on your local machine. This is performed automatically when you install the SAP Desktop SDK, which also provides complete code in the following files:

- For Example: Calling BAPIs [Page 34]:

  wdobapi.vbp, wdobapi.frm.

- For Example: Handling Collections of Business Objects [Page 36]:

  wdosched.vbp, wdosche1.frm, wdosche2.frm.

See also the corresponding unit of Tutorial: Communication Interfaces [Ext.].

Example: Calling BAPIs

This Visual Basic application demonstrates all the necessary steps for calling a BAPI.

The application uses the *SalesOrder* business object, which is a contractual arrangement between a sales organization and a customer concerning the delivery of materials or the rendering of services in defined quantities at defined prices and at precise intervals.

The application allows you to create a sales order for a specific material, or enquire whether the material is available. To do this, the *SalesOrder* business object has to be defined in the Business Object Repository (BOR).

For a complete description of the *SalesOrder* object type, look in the Business Object Repository.

In this example, the BAPI ActiveX control object was created by adding it to the project, so that it is visible on the form.

Creating a sales order in R/3

'Declare object variables:

```
Dim boOrder As Object         'Business object SalesOrder
Dim oPartners As Object 'Parameter OrderPartners of BAPI method
Dim oHeader As Object         'Parameter OrderHeaderIn of BAPI method
Dim oItemsIn As Object  'Parameter OrderItemsIn of BAPI method
Dim oReturn As Object         'Parameter Return of BAPI method
```

'Connect to business object *SalesOrder*
'(this creates an anonymous object with an empty key field):

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

'Get structure/table objects:

```
Set oPartners = oBAPICtrl.DimAs(boOrder, "CreateFromData",
"OrderPartners")
Set oHeader = oBAPICtrl.DimAs(boOrder, "CreateFromData",
"OrderHeaderIn")
Set oItemsIn = oBAPICtrl.DimAs(boOrder, "CreateFromData",
"OrderItemsIn")
```

'Fill header:

```
oHeader.Value("DOC_TYPE") = "TA"          'Standard order
oHeader.Value("SALES_ORG") = "0001" 'Sales organization
oHeader.Value("DISTR_CHAN") = "01"        'Sales channel
oHeader.Value("DIVISION") = "01"          'Division
oHeader.Value("PO_NUMBER") = ""           'Customer purchase order
number
oHeader.Value("PRICE_DATE") = Now         'Date
```

'Fill partners:

```
oPartners.Rows.Add
oPartners.Value(1, "PARTN_ROLE") = "AG"        'PartnerRoll: Person
```

```
posting the order
oPartners.Value(1, "PARTN_NUMB") = "0000010096"'Customer number
```

'Fill items:

```
oItemsIn.Rows.Add
oItemsIn.Value(1,"REQ_QTY") = "0000000010000"   'Quantity
oItemsIn.Value(1,"MATERIAL") = "BERLINER"       'Product ID
oItemsIn.Value(1,"COND_VALUE") = "1432"         'Rate
```

'Call the method:

```
boOrder.CreateFromData OrderHeaderIn:=oHeader, _
                       OrderPartners:=oPartners, _
                       OrderItemsIn:=oItemsIn, _
                       Return:=oReturn
```

'Free the business objects:

```
Set boOrder = Nothing
```

Example: Handling Collections of Business Objects

This Visual Basic application retrieves the calendar information of an R/3 user and displays a summary of his/her appointments on a given date in a list box.

For this application, you need objects from four different object types:

Object types needed for calendar information

Business object type	Description
USR01	SAP user
SCHEDULER	Calendar application
SCHEDULE	Calendar of a user
APPOINTMNT (written without 'E')	Appointment

For a complete description of each object type, look in the Business Object Repository.

There is only one instance of the *SCHEDULER* object type. Its main function is to create *SCHEDULE* objects. (In COM terms, the object with the object type *SCHEDULER* is the factory for objects of the type *SCHEDULE*.)

While the BAPI ActiveX control creates instances of business objects, the *SCHEDULER* object creates instances of *SCHEDULE* objects. This is similar to the MS application Excel which creates spreadsheet objects.

These factory object types are purely the result of design decisions. For example, the *DominantOfToday* property of the *SCHEDULE* type returns instances of *DOMINANT* objects.

Retrieving the calendar information of an R/3 user

'Declare object variables:

```
Dim oBAPICtrl As Object             'BAPI control object
Dim oConnection As Object           'Connection object
Dim boSAPusr As Object              'SAP user whose appointments
                                    'are required
Dim boScheduler As Object           'Factory for schedules
Dim boSchedule As Object            'Schedule of the user boSAPusr
Dim oAppointments As Object         'Collection of appointments
Dim datToDay as Date                'A date variable
```

'Create the BAPI ActiveX control object:

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

'Get the connection object:

```
oConnection = oBAPICtrl.Connection
```

'Set user-specific data:

```
oConnection.User = “MyUserName”
oConnection.Client = “000”
```

'Log on to the R/3 System:

```
oConnection.Logon
```

'Get the SAP user object:

```
Set boSAPusr = oBAPICtrl.GetSAPObject("USR01", "Plattner")
```

'Get the calendar application object:

```
Set boScheduler = oBAPICtrl.GetSAPObject("SCHEDULER")
```

'Get the calendar of the SAP user:

```
Set boSchedule = boScheduler.ScheduleGet ("SCHEDULE", boSAPusr)
```

'Get the appointments of the SAP user:

```
datToDay = Date
Set oAppointments = boSchedule.AppointmentsGet(datToDay,datToDay)
```

'Loop through appointments collection and display in a list box:

```
Dim str as String
For Each boAppointment In oAppointments
         str = Format(boAppointment.TimeFrom, "hh:mm "
         str = str & Format(boAppointment.TimeTo, "hh:mm ")
         str = str & boAppointment.Room & " "
         str = str & boAppointment.Note
         lstAppointments.AddItem(str)
Next
```

'Release business objects:

```
Set boSAPusr = Nothing
Set boScheduler = Nothing
Set boSchedule = Nothing
Set oAppointments = Nothing
```

'Logoff:

```
oConnection.Logoff
```

'Release BAPI ActiveX control object:

```
Set oBAPICtrl = Nothing
```

Object Reference: BAPI ActiveX Control Object

Description

Local instance of the BAPI ActiveX control.

Properties

- *Connection* (Object)

  Returns or sets the connection object, which holds all the necessary information about the status of a remote connection to the R/3 System, and is also used to establish a connection.

  When read, the connection object is created automatically. Read/write access.

- *LogFileName* (String)

  Returns or sets the log file name.

  There are actually two log files:

  – **dev_bapi.trc** for the BAPI ActiveX control

  – **dev_func.trc** for the Function ActiveX control

  If no value is set, **dev_bapi.trc** and **dev_func.trc** are written to the current directory.

- *LogLevel* (Short)

  Returns or sets the current log level.

  The log level has a value range from 0 (trace disabled) to 9. The higher the value, the more detailed information you get.

- *BAPIExceptionCodes* (Bool)

  If an exception occurs in a BAPI, the *BAPIExceptionCodes* property determines whether only the exception ID is to be returned, or whether additional information is to be provided as an error description in the exception object. In Visual Basic, this is the *Description* property of the *Err* object.

  This applies only to exception number 8820 (exception occurred during BAPI call).

- *TransactionID* (String)

  Returns or sets the current transaction ID (TID).

  The *TransactionID* property is set by calling the *CreateTransactionID* method and then deleted after the BAPI call has been executed successfully.
  To repeat a BAPI call, you can set this property explicitly with the TID retrieved from TID management.

Methods

The BAPI ActiveX control object has the following methods:

- ***AboutBox()***

  Displays the control's *About* box.

- ***Object CreateCollectionOfSAPObjects()***

  Creates an empty collection object to which business objects may be added for subsequent use as parameters in a method call.

- ***Object GetSAPObject(String ObjectType, Variant ObjectKey1,... Variant ObjectKey10)***

  Creates local instances of business objects.

  The *ObjectType* parameter is a case-sensitive string that matches the object type of an object defined in the Business Object Repository (BOR).

  The *ObjectKey* parameters are usually optional. However, you have to supply them if you are creating a local instance of a persistent business object. Otherwise, no object keys are passed to the function.

  See also Creating Business Objects [Page 20].

- ***Object DimAs(Object BusinessObject, String Method, String Parameter)***

  Returns a specific table or structure object, which is a parameter of the provided method name. When this object is filled with data, it can be passed as a parameter to the appropriate BAPI. All the parameters are mandatory.

  For a detailed description of each parameter, see Passing Data to BAPIs [Page 22].

- ***CreateTransactionID ( )***

  Creates a new transaction ID and stores the value in the property TransactionID. This TID is used in the next BAPI call.

Object Reference: Collection Object

Description

Collections of local instances of SAP business objects.

Collection objects are:

- Created explicitly with the *CreateCollectionOfSAPObjects* method of the BAPI ActiveX control object
- Returned as a value from a BAPI that returns sequences of business objects

Collection objects are needed as parameters for methods which expect sequences of objects.

Besides the properties and methods listed below, collection objects have built-in support for iteration via *For Each... Next* statements in Visual Basic (COM's *_NewEnum* method).

Properties

Collection objects have the following properties:

- *Count*

  Returns the number of objects in the collection list. Read-only.

Methods

Collection objects have the following methods:

- *void Add(boObject)*

  Adds a business object to the collection object. Business objects within a collection object are generally (but not always) of the same object type. For example, a client program that takes advantage of polymorphism can build collections of objects containing objects of different object types.

  - *Object Item(intIndex)*

  Default method of the collection object. Returns a business object from the collection object.

  The *intIndex* parameter is required and is the index number in the collection. The index range is from 0 to count -1.

- *void Remove(intIndex)*

  Removes a business object from the collection object.

  The *intIndex* parameter is required and is the index number in the collection.

  The following Visual Basic code demonstrates how the collection object is used:

  'Create a collection object

```
Set oParticipants = oBAPICtrl.CreateCollectionOfSAPObjects()
```

'Add an object to the collection

```
oParticipants.Add(boWhoAmI)
```

'Use the collection object as a parameter in a method

```
boAppointment.Create datFrom, datTo, strType, strRoom, "",
                     strNote, strVisibility, strTyp, datFrom,
oParticipants
```

Object Reference: Business Object

Description

Local instance of an SAP business object in the R/3 System.

When you create a local instance of a business object, it automatically adapts to the interface of the SAP business object it represents. The interfaces of business objects are thus only known at runtime.

For information on the object types available in the R/3 System, look in the Business Object Repository (BOR).

All properties are currently read-only. Properties can only be set by the *Set* method (when implemented).

Since all object types support the so-called *IFSAP* interface, there is a minimal set of properties and methods available for each object type.

Properties

Business objects have at least the following property:

- *Object ObjectType*

  Read-only. Returns the meta-object (object type description) from the business object. This is an object of type *TOJTB* which contains the type information.

Methods

Business objects have at least the following methods:

- *Display*

  Displays the business object.

  If the method has been implemented for the object type you are looking at, it usually (but not always) calls a transaction which displays the object on a SAPgui screen. This may or may not be useful for your client program.

  If the method has not been implemented for the object type you are looking at, the default implementation is inherited from the *IFSAP* interface. This default implementation returns the key from the object, which may also not be very helpful.

  Before using the *Display* method on a business object, check its implementation for the object type which is of interest to you.

- *ExistenceCheck*

  Checks the existence of the object in the R/3 database.

  Here again, you should check the implementation. Some return nothing, some return a boolean, some raise an exception which must be handled in the client program.

For a skeleton of an application written in Visual Basic with several property and method calls, see Example Applications [Page 33].