

# BAPI-ActiveX-Control



HELP.BCFESDE8

**Release 4.6C**



## Copyright

© Copyright 2001 SAP AG. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Software-Produkte können Software-Komponenten auch anderer Software-Hersteller enthalten.

Microsoft<sup>®</sup>, WINDOWS<sup>®</sup>, NT<sup>®</sup>, EXCEL<sup>®</sup>, Word<sup>®</sup>, PowerPoint<sup>®</sup> und SQL Server<sup>®</sup> sind eingetragene Marken der Microsoft Corporation.

IBM<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, DB2/6000<sup>®</sup>, Parallel Sysplex<sup>®</sup>, MVS/ESA<sup>®</sup>, RS/6000<sup>®</sup>, AIX<sup>®</sup>, S/390<sup>®</sup>, AS/400<sup>®</sup>, OS/390<sup>®</sup> und OS/400<sup>®</sup> sind eingetragene Marken der IBM Corporation.

ORACLE<sup>®</sup> ist eine eingetragene Marke der ORACLE Corporation.

INFORMIX<sup>®</sup>-OnLine for SAP und Informix<sup>®</sup> Dynamic Server<sup>™</sup> sind eingetragene Marken der Informix Software Incorporated.

UNIX<sup>®</sup>, X/Open<sup>®</sup>, OSF/1<sup>®</sup> und Motif<sup>®</sup> sind eingetragene Marken der Open Group.

HTML, DHTML, XML, XHTML sind Marken oder eingetragene Marken des W3C<sup>®</sup>, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc.

JAVASCRIPT<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo und mySAP.com sind Marken oder eingetragene Marken der SAP AG in Deutschland und vielen anderen Ländern weltweit. Alle anderen Produkte sind Marken oder eingetragene Marken der jeweiligen Firmen.

## Symbole

Symbol	Bedeutung
	Achtung
	Beispiel
	Hinweis
	Empfehlung
	Syntax

## Inhalt

<b>BAPI-ActiveX-Control</b> .....	<b>5</b>
<b>BAPI-ActiveX-Control</b> .....	<b>6</b>
<b>Business-Objekte</b> .....	<b>9</b>
<b>BAPIs</b> .....	<b>10</b>
<b>Business Object Repository</b> .....	<b>11</b>
SAP-Business-Objekte suchen .....	12
Der BAPI Browser .....	13
<b>BAPI-ActiveX-Control: Benutzerleitfaden</b> .....	<b>14</b>
BAPI-ActiveX-Control-Objekt anlegen .....	15
Im R/3 anmelden .....	16
Benutzer- und System-Anmeldeinformationen vordefinieren .....	18
Business-Objekte anlegen .....	20
Daten an BAPIs übergeben .....	23
Von R/3 abmelden .....	25
Transaktionale RFCs mit BAPIs ausführen .....	26
Sammelobjekte anlegen .....	28
Tips zur Fehlersuche (Debugging) .....	29
Fehler und Ausnahmen handhaben .....	31
Beispielanwendungen .....	35
Beispiel: BAPIs aufrufen .....	36
Beispiel: Sammlungen von Business-Objekten handhaben .....	38
Objektbezug: BAPI-ActiveX-Control-Objekt .....	40
Objektbezug: Sammelobjekt .....	42
Objektbezug: Business-Objekt .....	44

## BAPI-ActiveX-Control

## BAPI-ActiveX-Control

### Definition

Das BAPI-ActiveX-Control ermöglicht es externen Anwendungen, durch den Aufruf von BAPIs (Business Application Programming Interface) über OLE-Automation Geschäftsfunktionen im R/3-System anzusprechen. Client-Programme greifen auf Proxy-Objekte zu, die lokale, vom BAPI-ActiveX-Control verwaltete Instanzen von SAP-Business-Objekten sind. Diese Proxy-Objekte entsprechen den echten SAP-Business-Objekten, die im Business Object Repository (BOR) des R/3 gespeichert sind.

Die Einzelheiten dieses Prozesses sind für das Client-Programm unsichtbar, da die mit Hilfe des BAPI-ActiveX-Controls aus dem R/3 exportierten Datenstrukturen in Objekten gekapselt sind. Jedoch ist der Zugriff auf Objekte einfach, da ihre Handhabung der lokaler Desktop-Projekte gleicht.

### Verwendung

Bevor Ihr Client-Programm mit Hilfe des BAPI-ActiveX-Controls SAP-Business-Objekte ansprechen kann, muß es eine lokale Instanz des BAPI-ActiveX-Controls anlegen und eine Objektvariable deklarieren. Dann kann es lokale Instanzen von Business-Objekten anfordern. Beim Anlegen auf dem Desktop gleichen lokale Instanzen von Business-Objekten ihre Schnittstellen dynamisch an die Schnittstellen der echten SAP-Business-Objekte an, die sie darstellen. Die Schnittstellen von Business-Objekten sind daher nur zur Laufzeit bekannt.

Selbst wenn Sie mit objektorientierter Programmierung nicht vertraut sind, ist das BAPI-ActiveX-Control einfach einzusetzen. Wenn Sie Sprachen wie Visual Basic kennen, werden Sie nicht einmal merken, daß Sie mit einem remoten R/3-System arbeiten. Client-Programme können in Visual Basic oder jeder anderen Sprache geschrieben werden, die OLE-Automation unterstützt.

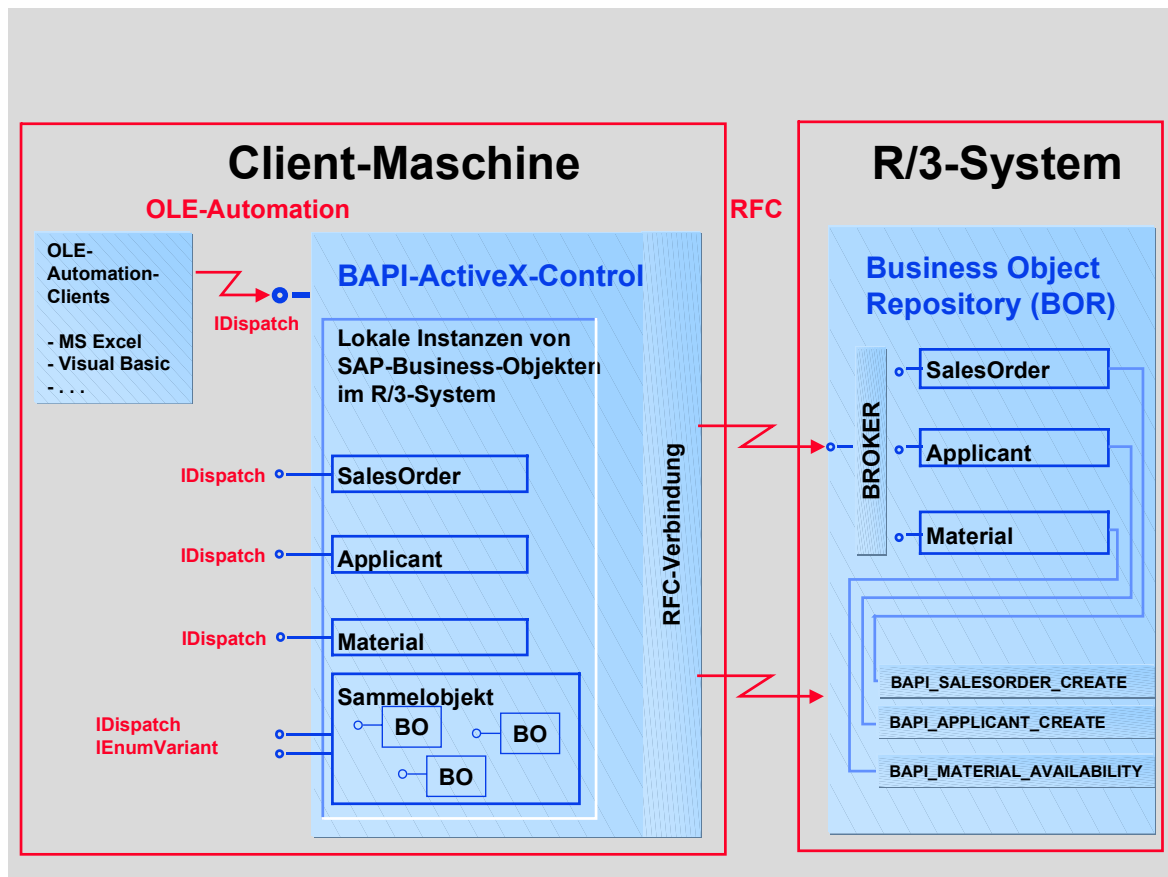


Zur Einführung in diese Technik finden Sie ein einfaches Beispiel in Visual Basic in dem entsprechenden Kapitel des [Tutorial: Kommunikationsschnittstellen \[Extern\]](#).

### Struktur

Client-Programme sprechen SAP-Business-Objekte an, indem sie OLE-Automation-Aufrufe an das BAPI-ActiveX-Control senden, das die Anforderungen über Remote Function Call (RFC) an das R/3-System weiterleitet (s. Grafik):

#### BAPI-ActiveX-Control



Wenn ein Client-Programm eine Methode zur Manipulation eines Business-Objekts aufruft, sucht das BAPI-ActiveX-Control nach einer geeigneten Methode:

- Ist die gerufene Methode ein BAPI, so ruft das BAPI-ActiveX-Control den entsprechenden Funktionsbaustein im R/3 direkt über das Function-ActiveX-Control.
- Ist die gerufene Methode kein BAPI, erfolgt der Zugriff über die BOR-Laufzeit.

Direkte OLE-Automation-Aufrufe von BAPIs sind schneller, da der beim Aufruf einer Methode aus dem BOR anfallenden Overhead wegfällt.

### Hierarchie lokaler Objekte

Da das BAPI-ActiveX-Control den Zugriff auf alle anderen Objekte verwaltet, muß das Client-Programm zuerst eine lokale Instanz des BAPI-ActiveX-Controls anlegen. Nur dann ist es möglich, auf Business-Objekte zuzugreifen und sie durch Lesen der Attribute und Aufruf der Methoden zu verarbeiten.

Das BAPI-ActiveX-Control verwaltet eine lokale Hierarchie der Business-Objekt-Instanzen. Dazu zählen:

- Business-Objekte
- Sammelobjekte

---

**BAPI-ActiveX-Control****Business-Objekte**

Sie legen Business-Objekte

- explizit im Client-Programm an.
- implizit durch das BAPI-ActiveX-Control als Rückgabewert beim Aufruf einer Methode an.

Wenn Sie ein Business-Objekt anlegen, gibt das Client-Programm - oder der Business Object Broker (BOB) - den benötigten Objekttyp an. Einmal angelegt reagiert jedes Business-Objekt nur auf die Attribute und Methoden seines Objekttyps, wie im BOR angegeben.

Business-Objekte sind nur solange aktiv, wie das BAPI-ActiveX-Control-Objekt, das sie angelegt hat, am R/3-System angemeldet bleibt. Wenn eine nicht beschriebene Ausnahme im R/3-System auftritt, kann die RFC-Verbindung abbrechen. Dann müssen Sie zu jedem Objekt die Verbindung erneut herstellen.

**Sammelobjekte**

Sammelobjekte sind eine Reihe von Business-Objekt-Instanzen, die Sie zusammen verarbeiten. Es gibt Methoden, die alle Objekte in einem Sammelobjekt nacheinander verarbeiten (Iteration).

**Integration**

Das BAPI-ActiveX-Control ist gegenwärtig unter Windows95 und WindowsNT implementiert.

## Business-Objekte

**Business-Objekte** sind echte Entitäten als Objekte im Modell eines Informationssystems dargestellt.

Business-Objekte **kapseln** sowohl Datenstrukturen als auch die auf die Daten angewandten Funktionen und verstecken ihre Komplexität vor anderen Objekten. Diese Kapselung von Daten und Funktionen erleichtert die Modifikation von Programmkomponenten, da Sie mit den entsprechenden Entitäten arbeiten können, ohne die Einzelheiten der Implementierung wissen zu müssen. So können Sie auch bestehende Funktionen wiederverwenden.

Client-Programme greifen auf Business-Objekte zu, indem Sie ihre **Attribute** lesen oder die **Methoden** aufrufen, die die **Schnittstelle** des Objekts bilden:

- Attribute

Attribute beschreiben die in einem Objekt gespeicherten Daten über eine Menge von Properties (Eigenschaften). Durch sie können die Datenstrukturen des Objekts direkt gelesen werden, aber Client-Programme können sie von außen nicht ändern.

- Methoden

Methoden dienen zur Kapselung der Datenstrukturen von Business-Objekten, und zu ihrer Verarbeitung. Beim Zugriff auf ein Objekt ruft das Client-Programm eine Methode mit Parametern auf und bekommt Rückgabeparameter zurückgeliefert.

- Schnittstelle

Die Schnittstelle ist eine Menge von Methoden, die zu einem Business-Objekt gehören; sie bestimmt, wie ein Objekt mit der Außenwelt kommuniziert.

Das Client-Programm definiert die zu verwendenden Objekttypen und legt zur Laufzeit Objektinstanzen dieser Objekttypen an.

SAP-Business-Objekte werden im Business Object Repository (BOR) des R/3 verwaltet.

## BAPIs

### BAPIs

BAPIs (Business-APIs) sind spezielle Methoden, die auf die im Business Object Repository (BOR) des R/3 abgelegten SAP-Business-Objekte angewendet werden, um bestimmte Geschäftsaufgaben durchzuführen, wie z.B. Erstellen eines Kundenauftrags oder Verfügbarkeitsprüfung eines Materials.

Im R/3 sind BAPIs als RFC-fähige Funktionsbausteine im Function Builder der ABAP Workbench abgelegt. Sie haben Standard-Geschäftsschnittstellen, über die externe Anwendungen mit Hilfe von SAP-Business-Objekten auf R/3-Prozesse, -Funktionen und -Daten zugreifen können.

Client-Programme, die mit BAPIs auf SAP-Business-Objekte zugreifen, können Teil des R/3-Systems, Teil eines externen Systems (z.B. ein Visual Basic-Frontend), Teil eines HTTP-Gateway oder Teil eines anderen R/3-Systems sein:

- Innerhalb von R/3 kann ein Client-Programm auf ein Business-Objekt zugreifen, indem es einen Methodenaufruf vom Business Object Broker (BOB) anfordert.  
Die in diesem Fall verwendete Methode kann ein BAPI sein, muß aber nicht.
- Ein externes Client-Programm kann ein BAPI direkt über das BAPI-ActiveX-Control aufrufen, indem es eine Instanz eines Business-Objekts anlegt.

In diesem Fall sucht das BAPI-ActiveX-Control nach einem geeigneten BAPI, um das Objekt zu verarbeiten:

- Gibt es für diesen Zweck ein BAPI, so ruft das BAPI-ActiveX-Control es direkt auf.
- Gibt es für diesen Zweck kein BAPI, erfolgt der Zugriff über das BOR.

Direkte OLE-Automation-Aufrufe von BAPIs sind schneller als der Zugriff über das BOR-Laufzeitsystem, da der beim Aufruf einer Methode aus dem BOB anfallende Overhead wegfällt.

## Business Object Repository

Das Business Object Repository (BOR) ist eine Entwicklungs- und Laufzeitumgebung im R/3-System, die alle SAP-Business-Objekttypen und ihre Methoden verwaltet. Zu den Business-Objekttypen gehören:

- Business-Objekte  
Business-Objekte sind Objekte wie "Kunde", "Material" oder "Buchungskreis". Sie liefern sowohl eine übergeordnete betriebswirtschaftliche Sicht auf das R/3-System als auch eine Programmierschnittstelle zum R/3-System..
- Technische Objekte  
Technische Objekte sind Texte, Hinweise, Arbeitspositionen und archivierte Belege, sowie Desktop-Objekte wie Texte, Grafiken und Tabellenkalkulationen. Desktop-Objekte können in verdichteter Form im BOR beschrieben werden.
- Metaobjekte  
Metaobjekte umfassen Objekttypen, Methoden, Attribute und Ereignisse. Jedes Objekt hat ein Attribut "Objekttyp", der sich auf das Metaobjekt bezieht, dem es zugeordnet ist. Die für ein bestimmtes Objekt verfügbaren Methoden, Attribute und Ereignisse leiten sich aus seinem Objekttyp ab.

Mit den Funktionen des BOR können Sie neue Objekttypen anlegen oder bestehende ändern. Um Objekte aus Ihrem eigenen System oder einem beliebigen externen System zu kapseln, erlaubt Ihnen das BOR auch die Definition eigener Objekttypen.

Der Kern des BOR-Laufzeitsystems ist der Business Object Broker (BOB). Client-Programme senden Anforderungen an den BOB, die

1. das angesprochene Laufzeitobjekt anlegen.
2. die korrekte Methodenimplementierung wählen.
3. die gefundene Methode aufrufen.
4. die Ergebnisse an den Aufrufer zurückliefern.
5. die Laufzeitobjekte zerstören.

Aufgerufene Methoden können auch selbst Methodenaufrufe anfordern.

Weitere Informationen zur Suche nach SAP-Business-Objekten im BOR finden Sie unter

[SAP-Business-Objekte suchen \[Seite 12\]](#)

## SAP-Business-Objekte suchen

## SAP-Business-Objekte suchen

SAP-Business-Objekte und ihre zugeordneten Methoden - BAPIs oder nicht - werden im Business Object Repository (BOR) des R/3 verwaltet und in einer Struktur gespeichert, die auf der Anwendungshierarchie basiert.

### Vorgehensweise

Um SAP-Business-Objekte im BOR zu suchen, verwenden Sie den Business Object Browser:

1. Melden Sie sich über den SAPGUI am R/3 an.  
Den SAPGUI benötigen Sie nur solange Sie programmieren. Sobald das Client-Programm läuft, ist er nicht mehr erforderlich (vorausgesetzt, die Business-Objekte, die Sie verwenden, senden keine Bilder).
2. Wählen Sie *Werkzeuge* → *ABAP Workbench* → *Überblick* → *Business Object Browser*
3. Klappen Sie die Knoten einer beliebigen Anwendung auf, bis Sie die Ebene erreichen, auf der Business-Objekte angebunden sind.
4. Wählen Sie ein Business-Objekt durch Doppelklick aus.
5. Klappen Sie den Knoten *Methoden* auf.  
Sie erhalten eine Liste der Methoden für das Objekt. Die Methoden, neben deren Namen auf der rechten Seite ein grünes Licht erscheint, sind BAPIs.
6. Wenn Sie nach BAPIs suchen, klicken Sie das grüne Licht neben dem Namen des gewünschten BAPI an.  
Sie sehen den Quellcode des BAPI-Funktionsbausteins.
7. Zur Anzeige der Parameterinformationen wählen Sie *Springen* → *Import/Exportparam.*
8. Zur Anzeige der einzelnen Felder einer Tabelle, zusammen mit Feldnamen, Feldtypen, und Feldlängen, klicken Sie zweimal auf eines der Referenzfelder.



Ehe Sie einen Objekttyp verwenden, müssen Sie sich vergewissern, daß er zum Einsatz freigegeben ist. Einige Objekttypen im BOR sind Modellierungsobjekte und daher noch nicht implementiert.

Wenn Sie nur nach den SAP-Business-Objekten suchen, für die BAPIs implementiert sind, verwenden Sie den BAPI Browser. Siehe auch:

[Der BAPI Browser \[Seite 13\]](#)

## Der BAPI Browser

Um nur nach den SAP-Business-Objekten zu suchen, für die BAPIs implementiert wurden, verwenden Sie den BAPI Browser. Der BAPI Browser ist eine BAPI-orientierte Sicht des Business Object Repository (BOR):

1. Melden Sie sich über den SAPGUI im R/3 an.
2. Wählen Sie *Werkzeuge* → *Business Framework* → *BAPI Browser*.
3. Klappen Sie die Knoten einer beliebigen Anwendung auf, bis Sie die Ebene erreichen, auf der Business-Objekte angebunden sind.
4. Wählen Sie ein Business-Objekt durch Doppelklick aus.
5. Klappen Sie den Knoten *Methoden* auf.

Sie erhalten eine Liste aller zum gewählten Business-Objekt implementierten BAPIs.

6. Um Parameterinformationen zu erhalten, klappen Sie den Knoten *Parameter* auf.
7. Um Dokumentation zu einem BAPI zu erhalten, klicken Sie auf die entsprechende Info-Ikone.

## BAPI-ActiveX-Control: Benutzerleitfaden

### Zweck

Dieser Prozeß listet die Aufgaben auf, die ein externes Client-Programm ausführen sollte, wenn es das BAPI-ActiveX-Control zum Zugriff auf SAP-Business-Objekte im R/3 verwendet und/oder die Datenstrukturen dieser Business-Objekte mit Hilfe von BAPIs verändert.

### Prozeßablauf

1. Legen Sie das BAPI-ActiveX-Control-Objekt an - eine Instanz des BAPI-ActiveX-Controls.
2. Stellen Sie eine Verbindung zum R/3-System her und melden Sie sich an.
3. Legen Sie eine oder mehrere Business-Objekt-Instanzen an.
4. Greifen Sie auf Business-Objekt-Properties zu und/oder rufen Sie BAPIs auf, um Business-Objekte zu verändern.
5. Geben Sie Business-Objekte frei.
6. Melden Sie sich vom R/3-System ab.
7. Geben Sie das BAPI-ActiveX-Control-Objekt frei.



Ein einfaches Beispielprogramm finden Sie im entsprechenden Kapitel des [Tutorial: Kommunikationsschnittstellen \[Extern\]](#).

## BAPI-ActiveX-Control-Objekt anlegen

### Voraussetzungen

Ehe Sie das BAPI-ActiveX-Control-Objekt im Client-Programm anlegen können, müssen Sie das BAPI-ActiveX-Control in Ihrem Visual Basic-Projekt aktivieren:

1. Wählen Sie *Project* → *Components*.
2. Markieren Sie *Controls*.
3. Markieren Sie *SAP BAPI Control*.

Das System fügt das BAPI-ActiveX-Control-Objekt in die Visual Basic-Toolbox ein.

### Vorgehensweise

Das BAPI-ActiveX-Control-Objekt können Sie auf eine der folgenden Arten anlegen:

- Ziehen Sie das BAPI-ActiveX-Control von der Visual Basic-Toolbox in das Projektformular.
- Geben Sie folgende Coding-Zeilen ein:

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

Im R/3 anmelden

## Im R/3 anmelden

### Voraussetzungen

Ehe Sie auf SAP-Business-Objekte zugreifen können und/oder BAPIs zum Verändern dieser Objekte aufrufen können, müssen Sie

1. eine Verbindung zum R/3-System herstellen.
2. sich am R/3-System anmelden.

Befindet sich die Verbindung nicht im Status "Angemeldet", so löst jeder Versuch, auf Business-Objekt-Properties zuzugreifen oder Methoden aufzurufen, eine Ausnahme aus.

### Vorgehensweise

#### Verbindung aufbauen

Sie können die Verbindung auf zwei verschiedene Arten aufbauen.

- Stellen Sie die Verbindung direkt her über das Logon-ActiveX-Control.  
Das Logon-ActiveX-Control kapselt den Prozeß des Verbindungsaufbaus und stellt ein Verbindungsobjekt in Form der Methode *NewConnection* des Logon-Objektes zur Verfügung.
- Stellen Sie die Verbindung implizit her über das BAPI-ActiveX-Control.  
Das Verbindungsobjekt des Logon-ActiveX-Controls ist eine Property des BAPI-ActiveX-Controls.

Keine dieser beiden Möglichkeiten garantiert jedoch, daß eine Verbindung zum R/3-System auch tatsächlich aufgebaut wurde.

Um den Status des Verbindungsobjekts und der R/3-Verbindung zu überprüfen, sehen Sie sich die Verbindungs-Property *IsConnected* an.

#### Am R/3 anmelden

Melden Sie sich am R/3-System mit der Methode *Logon* des Verbindungsobjekts an:

- Ist der Wert des zweiten Parameters der Methode *Logon* 'False' und wurden nicht alle Parameter, die Anmeldeinformationen enthalten, gesetzt, erscheint ein Dialogfenster, in dem der Benutzer die fehlenden Werte eingeben kann.
- Ist der Wert des zweiten Parameters der Methode *Logon* 'True' und wurden alle Parameter, die Anmeldeinformationen enthalten, gesetzt, ist es nicht erforderlich, dem Benutzer ein Dialogfenster anzuzeigen. In diesem Fall wird die Anmeldung direkt ausgeführt.



Sie können Anmeldeinformationen vordefinieren, indem Sie die Benutzer- und System-Properties des BAPI-ActiveX-Controls in der Visual Basic-Designumgebung pflegen.

#### Verbindung direkt herstellen über das Logon-ActiveX-Control

Im R/3 anmelden

Visual Basic-Coding	Kommentar
<pre>Set oLogonCtrl = CreateObject("SAP.LogonControl.1")</pre>	Legt das Logon-ActiveX-Control an
<pre>Set oBAPICtrl.Connection = oLogonControl.NewConnection</pre>	Legt eine neue Verbindung an und setzt sie im BAPI-ActiveX-Control. Damit ist nicht garantiert, daß die Verbindung zum R/3-System auch tatsächlich aufgebaut wurde.
<pre>OBAPICtrl.Connection.Client = "&lt;client&gt;" oBAPICtrl.Connection.User = "&lt;user name&gt;" oBAPICtrl.Connection.Language = "&lt;language&gt;" ...</pre>	Liefert Anmeldeinformationen für das R/3-System.

Verbindung implizit herstellen über das BAPI-ActiveX-Control

Visual Basic-Coding	Kommentar
<pre>OBAPICtrl.Connection.Client = "&lt;client&gt;" oBAPICtrl.Connection.User = "&lt;user name&gt;" oBAPICtrl.Connection.Language = "&lt;language&gt;" ...</pre>	Legt ein neues Verbindungsobjekt an. Liefert Anmeldeinformationen für das R/3-System.
<pre>OBAPICtrl.Connection.Logon (0, FALSE)</pre>	Öffnet die Verbindung zum R/3-System. Ein Dialogfenster erscheint, in dem der Benutzer die fehlenden Anmeldeparameter eingeben kann.

**Benutzer- und System-Anmeldeinformationen vordefinieren**

## Benutzer- und System-Anmeldeinformationen vordefinieren

### Vorgehensweise

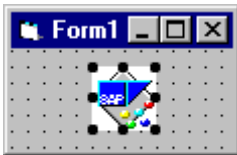
Um Benutzer- und System-Anmeldeinformationen vorzudefinieren, pflegen Sie die Properties des BAPI-ActiveX-Controls in der Visual Basic-Designumgebung.

Die nachfolgenden Abbildungen zeigen das Aussehen des BAPI-ActiveX-Control-Objekts in der Visual Basic-Toolbox und wie es nach dem Verschieben auf ein Formular aussieht.

#### BAPI-ActiveX-Control in der Visual Basic-Toolbox



#### BAPI-ActiveX-Control auf einem Visual Basic-Formular



Wenn Sie das BAPI-ActiveX-Control-Objekt auf ein Formular stellen, können Sie die Properties über die rechte Maustaste pflegen. Die nachfolgenden Abbildungen zeigen den Aufbau der Property-Seiten:

#### Benutzerseite

Benutzer- und System-Anmeldeinformationen vordefinieren

The screenshot shows the 'SAP BAPI Properties' dialog box with the 'User' tab selected. The 'System' tab is also visible. The 'R/3 User Defaults' section contains three input fields: 'Client' with the value '000', 'User' which is empty, and 'Language' with the value 'E'. At the bottom, there are three buttons: 'OK', 'Cancel', and 'Apply'.

Systemseite

The screenshot shows the 'SAP BAPI Properties' dialog box with the 'System' tab selected. The 'User' tab is also visible. The 'System' section has an input field for 'System' and a checkbox for 'GroupSelection'. The 'R/3 Server Entry' section contains four input fields: 'Application Server', 'Systemnumber' with the value '0', 'GroupName', and 'Message Server'. At the bottom, there are three buttons: 'OK', 'Cancel', and 'Apply'.

## Business-Objekte anlegen

# Business-Objekte anlegen

## Voraussetzungen

Ehe Sie lokale Instanzen von SAP-Business-Objekten anlegen können, müssen Sie das BAPI-ActiveX-Control-Objekt anlegen und sich am R/3-System anmelden.

## Vorgehensweise

Um lokale Instanzen von SAP-Business-Objekten anzulegen, verwenden Sie die Methode *GetSAPObject*.

*GetSAPObject* hat folgende Syntax:

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

*GetSAPObject* hat folgende Parameter:

- *ObjectType*

Dieser Parameter identifiziert ein SAP-Business-Objekt. Es handelt sich um eine Zeichenkette, die Groß- und Kleinschreibung unterscheidet und einem im Business Object Repository (BOR) definierten Objekttyp entspricht.

Sie können entweder den Namen des Business-Objekttyps oder seinen technischen Namen angeben. Der Business-Objekttyp *SalesOrder* hat beispielsweise den technischen Namen *BUS2032*. Beide Namen sind eindeutige Bezeichner desselben Business-Objekts.

- *ObjectKey1... ObjectKey10*

Der Parameter *ObjectKey* dient zur eindeutigen Kennzeichnung einer persistenten Instanz des SAP-Business-Objekts, zu dem Sie die Verbindung herstellen wollen.

Hat ein Objekttyp mehrere Schlüsselattribute, so entspricht jeder *ObjectKey*-Parameter einem Schlüsselattribut. Die Reihenfolge der *ObjectKey*-Parameter muß der Reihenfolge entsprechen, in der die Schlüsselattribute im BOR definiert sind.

Die *ObjectKey*-Parameter sind **optional**.

Sie müssen *ObjectKey*-Parameters übergeben, wenn

- Sie die Verbindung zu einer persistenten Instanz eines SAP-Business-Objekts aufbauen wollen.
- Sie eine **neue** Instanz eines persistenten Objekts anlegen wollen.
- Sie ein persistentes Objekt mit nur einer Instanz anlegen wollen.

In diesem Fall prüft das System zunächst, ob ein Business-Objekt mit dem angegebenen Schlüssel existiert. Wenn ja, wird eine Instanz des Business-Objekts angelegt und mit den aus der Datenbank gelesenen Daten initialisiert.

Sie müssen keine *ObjectKey*-Parameter angeben, wenn

- Sie eine **neue** Instanz eines Objekts anlegen wollen.
- Sie ein Objekt mit nur einer Instanz anlegen wollen.

## Verbindung zu einer persistenten Instanz eines Objekttyps aufnehmen



Legen Sie eine lokale Instanz des SAP-Business-Objekts *SalesOrder* mit der Verkaufsbelegnummer 784 an:

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

Das Objekt wird mit den aus der R/3-Datenbank gelesenen Daten initialisiert.

## Neue Instanz eines persistenten Objekts anlegen

In diesem Fall bedeutet **neu**, daß ein solches Objekt bisher in der R/3-Datenbank nicht existiert.

1. Legen Sie ein anonymes Business-Objekt an, ohne irgendwelche Schlüsselwerte anzugeben.
2. Bauen Sie das Objekt auf, indem Sie Properties setzen und Methoden aufrufen.
3. Führen Sie mit der für diesen Zweck verfügbaren Methode ein Commit des Objekts durch.

Diese Methode - in der Regel mit dem Namen *Post*, *Submit*, *Create* oder *Commit* - schreibt das Objekt in die R/3-Datenbank. Dabei erhält es einen Schlüssel, den Sie in Zukunft verwenden können, um sich auf das Objekt zu beziehen.



Legen Sie ein anonymes Business-Objekt des Typs *SalesOrder* an

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

Holen Sie Tabellen-/Strukturobjekte der exportierten Parameter und füllen Sie sie mit Daten

...

Führen Sie ein Commit des Business-Objekts *SalesOrder* durch und schreiben Sie es auf die R/3-Datenbank

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

Holen Sie den eindeutigen Schlüssel für die Instanz. In diesem Fall ist das die Verkaufsbelegnummer

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

## Objekttyp mit nur einer Instanz anlegen



Legen Sie eine Instanz des SAP-Business-Objekts *SCHEDULER* an.

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

**Business-Objekte anlegen**

Es gibt nur eine Instanz dieses Objekts und sie hat keinen Schlüssel.

## Daten an BAPIs übergeben

### Voraussetzungen

BAPIs erwarten eine große Anzahl strukturierter Datentypen und/oder Tabellen. Damit Sie beim Aufruf korrekt typisierte Parameter übergeben, bieten die R/3-Desktop-Integrationskomponenten zur Unterstützung eine Reihe von Objekten an, die Strukturen und Tabellen kapseln.

### Vorgehensweise

Um die Objekte anzulegen, mit denen Sie Daten auf einfache Weise an BAPIs übergeben können, verwenden Sie die Methode *DimAs*.

Die Methode *DimAs* hat folgende Syntax:

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

*DimAs* hat folgende Parameter:

- *BusinessObject*

Dies ist der Name des in der Anwendung zur Kennzeichnung des SAP-Business-Objekts verwendeten Objekts. Er wird von einem vorherigen Aufruf von *GetSAPObject* zurückgegeben.
- *Method*

Dies ist der Name des BAPI, zu dem der Parameter gehört. Die Zeichenkette unterscheidet Groß- und Kleinschreibung und muß einem der BAPIs für das SAP-Business-Objekt *Object* entsprechen, so wie es im BOR definiert ist.
- *Parameter*

Dies ist der Name des Parameters, dessen Struktur- oder Tabellenobjekt beschafft wird. Die Zeichenkette unterscheidet Groß- und Kleinschreibung und muß einem der Parameternamen des BAPI *Method* entsprechen, so wie es im BOR definiert ist.

Für jedes BAPI pflegt das BAPI-ActiveX-Control eine Sammlung von R/3-Tabellen, die durch ein Tabellensammelobjekt dargestellt werden. Um die Definition einer bestimmten Tabelle oder Struktur zu erhalten, rufen Sie die Methode *DimAs* auf und geben den Business-Objektyp, die BAPI-Methode und den Parameternamen an. Das System liefert dann das angeforderte Tabellen- oder Strukturobjekt zurück. Das Objekt verbirgt die von der nativen RFC-Schnittstelle übergebene interne Tabelle und handhabt die Daten, die im R/3-System gelesen oder dorthin geschrieben wurden. Es liefert außerdem eine vollständige, zweidimensionale Sicht der internen Tabelle.

Den erwarteten Typ finden Sie in der Parameterdefinition des angeforderten BAPI im Business Object Repository (BOR).



1. Holen Sie das Tabellenobjekt *OrderPartners* mit der Methode *DimAs*

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

2. Setzen Sie Werte des Parameters *OrderPartners*

### Daten an BAPIs übergeben

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

3. Rufen Sie die Methode auf

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



Aus Sicht des rufenden Programms müssen nur die **Exportparameter** mit *DimAs* **initialisiert** werden. Die Daten können dann in das beschaffte Tabellen- oder Strukturobjekt gestellt werden. Diese Parameter sind als Importparameter des BAPI-Funktionsbausteins definiert.

Sie können **nicht initialisierte** Objekte an Parameter übergeben, die Rückgabewerte erhalten.

## Von R/3 abmelden

### Voraussetzungen

Vor dem Abmelden vom R/3-System sollten Sie die Business-Objekte freigeben.

### Vorgehensweise

Um sich vom R/3-System abzumelden, rufen Sie die Abmeldemethode des Verbindungsobjekts auf und beenden die bestehende Verbindung mit folgenden Coding-Zeilen:

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

### Ergebnis

Nachdem Sie sich abgemeldet haben, werden nicht freigegebene Business-Objekte deaktiviert. Jeder Versuch, eine Methode aufzurufen oder auf eine Property zuzugreifen, löst eine Ausnahme aus.

## Transaktionale RFCs mit BAPIs ausführen

## Transaktionale RFCs mit BAPIs ausführen

BAPIs sollten, wie alle Funktionsbausteine, nur einmal im R/3-System ausgeführt werden. Daher müssen Client-Programme die Fähigkeit des BAPI-ActiveX-Controls ausnutzen, transaktionale RFCs auszuführen. Dazu müssen sie die Transaktions-IDs von BAPI-Aufrufen selbst verwalten.

Wenn während eines BAPI-Aufrufs ein Netzwerkfehler (CPI-C) auftritt, wird eine Ausnahme ausgelöst (Fehler 8793). Das Client-Programm kann dann natürlich den Aufruf wiederholen, aber dann besteht das Risiko, daß das BAPI ein zweites Mal ausgeführt wird:

- Tritt der Fehler beim **Rufen** eines BAPI auf, kann das Client-Programm die BAPI-Methode problemlos ein weiteres Mal aufrufen.
- Tritt der Fehler **während oder nach** dem BAPI-Aufruf auf, ist es möglich, daß das BAPI bereits ausgeführt wurde. In diesem Fall ist eine Wiederholung des Aufrufs nicht zu empfehlen.

Da jede Datenbanktransaktion oder Logische Arbeitseinheit (LUW, Logical Unit of Work) nur einen BAPI-Aufruf enthalten darf, benötigt jeder BAPI-Aufruf seine eigene Transaktions-ID. Um dies zu bewerkstelligen, bietet das BAPI-ActiveX-Control die folgenden Methoden und Properties:

- Methode *CreateTransactionID*

Diese Methode gibt eine im R/3-System angelegte Transaktions-ID zurück. Die Transaktions-ID muß bei den an den nächsten BAPI-Aufruf zu übergebenden Daten gespeichert werden.

- Property *TransactionID*

Diese Property enthält die im nächsten BAPI-Aufruf verwendete Transaktions-ID. Wird die Methode erfolgreich ausgeführt, löscht das BAPI-ActiveX-Control selbst die Transaktions-ID.

Tritt während oder nach dem BAPI-Aufruf ein Fehler auf, muß Ihr Client-Programm

1. die Verbindung zu einem späteren Zeitpunkt wieder aufnehmen.
2. die Daten wiederherstellen, die mit der 'alten', der Property *TransactionID* des BAPI-ActiveX-Controls zugeordneten Transaktions-ID gesichert wurden.
3. den Aufruf mit der 'alten' Transaktions-ID wiederholen.

Das Programm darf keine neue Transaktions-ID anlegen (mit *CreateTransactionID*), da sonst die Gefahr besteht, daß die BAPI-Methode ein zweites Mal ausgeführt wird.

Ist der Aufruf erfolgreich, so ist die Transaktion abgeschlossen.

4. die Transaktions-ID-Verwaltung aktualisieren.



Das folgende Visual Basic-Coding ruft die Methode *WriteData* des Business-Objekts *TestObject* auf. Die Methode *WriteData* hat nur einen Tabellenparameter namens *Data*.

Im Coding wird davon ausgegangen, daß das BAPI-ActiveX-Control-Objekt bereits angelegt wurde, und daß die Verbindung zum R/3-System besteht. Zudem fehlt im Coding die Überprüfung der Transaktions-ID-Verwaltung ebenso wie die

## Transaktionale RFCs mit BAPIs ausführen

Wiederholung bereits fehlgeschlagener Aufrufe, da das Client-Programm diese Schritte im Regelfall direkt nach dem Start ausführt, um sicherzustellen, daß alle BAPI-Aufrufe einmal ausgeführt wurden.

```
' Objektvariablen deklarieren

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

' Business-Objekt vom Typ 'TestObject' anlegen
    Set oTestObj = oBAPICtrl.GetSAPObject("TestObject")

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

' Tabelle mit Daten füllen
    :

' Transaktions-ID anlegen, die automatisch in der Property TransactionID
' des BAPI-Controls gespeichert wird
    strTID = oBAPICtrl.CreateTransactionID

' Transaktions-ID zusammen mit den Daten und anderen Informationen
' (Objekttyp, Methodenname), die für den Aufruf benötigt werden, speichern
    :

' BAPI transaktional aufrufen - die in der Property gespeicherte Transaktions-ID
' TransactionID wird für folgenden Aufruf verwendet
    oTestObj.WriteData Data:= otabData

' Erfolgreich
    Exit Sub

ErrorHandler: 'Analyze error, clean up
```

Weitere Informationen zu transaktionalen Remote Function Calls finden Sie unter [Transaktionale Remote Function Calls verwenden \[Extern\]](#).

## Sammelobjekte anlegen

## Sammelobjekte anlegen

Sammelobjekte werden nur mit Attributen oder Nicht-BAPI-Methoden von SAP-Business-Objekten verwendet. Es war notwendig, Sammelobjekte einzuführen, da einige Versionen von Visual Basic Schwierigkeiten mit der Handhabung von Arrays oder Folgen von OLE-Objekten haben. Um zu vermeiden, daß daraus Probleme entstehen, liefert das BAPI-ActiveX-Control Sammlungen von Business-Objekten immer in ein Sammelobjekt verpackt zurück. Analog dazu sind die erwarteten Parameter in einem Sammelobjekt verpackte Sammlungen von Business-Objekten.



Im folgenden Coding ist *oAppointments* ein von einem *SCHEDULE*-Objekt durch die Methode *AppointmentsGet* zurückgegebenes Sammelobjekt:

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

Im folgenden Coding ist *oParticipants* ein Sammelobjekt, das *USR01*-Objekte enthält und als Parameter an die Methode *Create* eines *APPOINTMNT*-Objekts übergeben wird. (Der Objekttyp für das Objekt wird **ohne E** geschrieben).

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

Die Methode *AppointmentsGet* ist kein BAPI, da Sammlungen von Business-Objekten bei der Programmierung mit BAPIs nicht verwendet werden.

Das Sammelobjekt unterstützt auch *For Each... Next*-Schleifen in Visual Basic.



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

## Tips zur Fehlersuche (Debugging)

### Voraussetzungen

Ehe Sie mit der Fehlersuche in Ihrer Anwendung beginnen, müssen Sie die Debugging-Traces einschalten. Standardmäßig sind alle Traces ausgeschaltet. Sie sollten auch nur während des eigentlichen Debugging-Prozesses eingeschaltet werden.

### Vorgehensweise

#### Trace des BAPI-ActiveX-Controls

Für diesen Zweck bietet das BAPI-ActiveX-Control die folgenden Properties:

- *LogFileName*

Verwenden Sie diese Property, um den Namen der Protokolldatei auf einen vom Standard abweichenden Wert zu setzen.

Wenn Sie keinen Wert setzen, schreibt das System die Trace-Dateien **dev\_bapi.trc** und **dev\_func.trc** in das aktuelle Verzeichnis:

- **dev\_bapi.trc** enthält Traces des BAPI-ActiveX-Controls.
- **dev\_func.trc** enthält Traces des Function-ActiveX-Controls.

Wenn Sie *LogFileName* auf einen vom Standard abweichenden Wert setzen, werden zwei Protokolldateien geschrieben: eine, deren Name der übergebenen Zeichenkette entspricht, und eine, deren Name aus dem extrahierten Pfad und **dev\_func.trc** besteht.

- *LogLevel*

Verwenden Sie diese Property, um die benötigte Trace-Komplexität anzugeben. Die zulässigen Werte liegen zwischen 1 und 9.

Wenn Sie *LogLevel* auf einen Wert größer 0 setzen, wird eine Form des Trace aktiviert. Je höher der Wert ist, desto detailliertere Trace-Informationen erhalten Sie.

#### Trace des RFC-Aufrufs

Wenn Sie einen Trace der Verbindungsaktivitäten auf RFC-Ebene durchführen wollen, setzen Sie die Property *TraceLevel* des Verbindungsobjekts, ehe Sie die Verbindung zum R/3-System herstellen.

Mögliche Werte für *TraceLevel* sind:

- 0 (Trace nicht erwünscht)
- 1 (Trace erwünscht)

Die Informationen über die Verbindungsaktivitäten werden in einer Datei mit Namen **rfc<n>\_<n>.trc** gespeichert (wobei n eine beliebige Zahl ist). Diese Datei finden Sie im aktuellen Standardverzeichnis.

Debugging-Traces sind besonders bei Anmeldeproblemen und bei Versionsproblemen auf RFC-Ebene hilfreich.

Tips zur Fehlersuche (Debugging)

## Fehler und Ausnahmen handhaben

Um sicherzustellen, daß jedes Auftreten eines Fehlers in den Protokolldateien aufgezeichnet wird, setzen Sie die Property *LogLevel* auf einen Wert größer 0.

Fehler	Vorgehensweise
Logon-Methode des Verbindungsobjekts schlägt fehl.	Prüfen Sie den Status des Verbindungsobjekts und der R/3-Verbindung. Verwenden Sie sich dazu die Property <i>IsConnected</i> des Sammelobjektes.
Ausnahme tritt während eines BAPI-Aufrufs auf (Ausnahme 8820).	Stellen Sie fest, ob einfach nur die Ausnahme-ID als Fehlerbeschreibung zurückgegeben werden sollte. Setzen Sie dazu die Property <i>BAPIExceptionCodes</i> des BAPI-ActiveX-Controls auf 'True'.
Remote Function Call (RFC)-Fehler tritt auf, entweder wenn ein BAPI eine Ausnahme auslöst oder wenn ein interner Fehler im BAPI-ActiveX-Control auftritt.	Verarbeiten Sie dies mit einem Error Handler.



Das folgende Coding ist ein Beispiel für einen in Visual Basic geschriebenen Error Handler:

```

On Error GoTo ErrorHandler

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

ErrorHandler:

If Err.Source = „SAP.BAPI“

    'dann trat ein Fehler im
    'SAP-Business-Objekt-Control auf
    'Handhabe ihn hier (Exit Sub, End,
    'Resume,...)

    Select Case Err.Number
        Case 8792
            ...
        Case...
            ...
    End Select
Else

    'ein anderer Typ Fehler trat auf
    'Handhabe ihn hier (Exit Sub, End,
    'Resume,...)

```

## Fehler und Ausnahmen handhaben

```

    ...
Endif
End Sub

```

## Ausnahmen des BAPI-ActiveX-Controls

Die für das BAPI-ActiveX-Control definierten Ausnahmen sind in der nachfolgenden Tabelle aufgezählt. Die Ausnahmenummer und eine Fehlerbeschreibung erhalten Sie vom Fehlerobjekt, das von Visual Basic gepflegt wird.

## Ausnahmen des BAPI-ActiveX-Controls

Fehler	Beschreibung
8792	<p>Interner RFC-Fehler.</p> <p>Fehlerschlüssel: RFC-Fehlerschlüssel. Beschreibung: RFC-Kurztextnachricht. CPIC-Status: CPIC-Fehlercode. Interner Zustand: RFC-interner Zustand.</p>
8793	<p>RFC-Ausnahme ausgelöst. Verbindung vom System unterbrochen.</p> <p>R/3-Fehlermeldung: Beschreibung des Fehlers. Fehlerschlüssel: RFC-Fehlerschlüssel. Beschreibung: RFC-Kurztextnachricht. CPIC-Status: CPIC-Fehlercode. Interner Zustand: RFC-interner Zustand.</p>
8794	(Nicht mehr verwendet.)
8795	<p>Laufzeit-Objekt des Typs <code>&lt;object type&gt;</code> mit persistentem Schlüssel <code>&lt;object key&gt;</code> konnte im Business Object Repository (BOR) nicht angelegt werden.</p> <p>Nachrichtenummer: Nachrichtenummer der T100-Nachricht im R/3-System. Arbeitsbereich: Arbeitsbereich der T100-Nachricht im R/3-System. R/3-Fehlermeldung: Beschreibung des Fehlers.</p> <p><b>Erklärung:</b> Ein temporärer Fehler, eine Anwendungsausnahme oder eine Ausnahme des Business Object Broker (BOB) trat im R/3-System auf.</p>
8796	<p>Persistenter Schlüssel des Laufzeitobjekts des Typs <code>&lt;object type&gt;</code> konnte nicht aus dem Business Object Repository (BOR) beschafft werden.</p> <p>Nachrichtenummer: Nachrichtenummer der T100-Nachricht im R/3-System. Arbeitsbereich: Arbeitsbereich der T100-Nachricht im R/3-System. R/3-Fehlermeldung: Beschreibung des Fehlers.</p> <p><b>Erklärung:</b> Ein temporärer Fehler, eine Anwendungsausnahme oder eine Ausnahme des Business Object Broker (BOB) trat im R/3-System auf.</p>

Fehler und Ausnahmen handhaben

8797	<p>Persistenter Schlüssel <b>&lt;object key&gt;</b> einer bestehenden Business-Objekt-Instanz vom Typ <b>&lt;object type&gt;</b> konnte im Business Object Repository (BOR) nicht gesetzt werden.</p> <p>Nachrichtenummer: Nachrichtenummer der T100-Nachricht im R/3-System. Arbeitsbereich: Abreitsbereich der T100-Nachricht im R/3-System. R/3-Fehlermeldung: Beschreibung des Fehlers.</p> <p><b>Erklärung:</b> Ein temporärer Fehler, eine Anwendungsausnahme oder eine Ausnahme des Business Object Broker (BOB) trat im R/3-System auf.</p>
8798	<p>Laufzeitobjekt vom Typ <b>&lt;object type&gt;</b> mit persistentem Schlüssel <b>&lt;object key&gt;</b> konnte im Business Object Repository (BOR) nicht freigegeben werden.</p> <p>Nachrichtenummer: Nachrichtenummer der T100-Nachricht im R/3-System. Arbeitsbereich: Abreitsbereich der T100-Nachricht im R/3-System. R/3-Fehlermeldung: Beschreibung des Fehlers.</p> <p><b>Erklärung:</b> Ein temporärer Fehler, eine Anwendungsausnahme oder eine Ausnahme des Business Object Broker (BOB) trat im R/3-System auf.</p>
8799	<p>Typinformationen des Business-Objekttyps <b>&lt;object type&gt;</b> konnten aus dem Business Object Repository (BOR) nicht beschafft werden.</p> <p>Nachrichtenummer: Nachrichtenummer der T100-Nachricht im R/3-System. Arbeitsbereich: Abreitsbereich der T100-Nachricht im R/3-System. R/3-Fehlermeldung: Beschreibung des Fehlers.</p> <p><b>Erklärung:</b> Ein temporärer Fehler, eine Anwendungsausnahme oder eine Ausnahme des Business Object Broker (BOB) trat im R/3-System auf.</p>
8800	<p>Während des Aufrufs des Objekttyps <b>&lt;method&gt;</b> trat ein Fehler im R/3-System auf.</p> <p>Nachrichtenummer: Nachrichtenummer der T100-Nachricht im R/3-System. Arbeitsbereich: Abreitsbereich der T100-Nachricht im R/3-System. R/3-Fehlermeldung: Beschreibung des Fehlers.</p> <p><b>Erklärung:</b> Ein temporärer Fehler, eine Anwendungsausnahme oder eine Ausnahme des Business Object Broker (BOB) trat im R/3-System auf.</p>
8801	<p>Objekttyp <b>&lt;method&gt;</b> ruft über RFC zurück ("callback"), aber dies ist nicht unterstützt.</p> <p><b>Erklärung:</b> Methodenimplementierung macht einen RFC-Rückruf zur Destination.</p>
8808	<p>Unerlaubter Zugriff auf <b>&lt;object type&gt;.&lt;method/property&gt;</b>.</p>
8809	<p>Keine Übereinstimmung bei Zugriff über Methode oder Property <b>&lt;object type&gt;.&lt;method/property&gt;</b>.</p>
8810	<p>Keine Verbindung zu einem SAP-System.</p>
8811	<p>Versuch, auf ein Objekt zuzugreifen, das nicht mehr mit einem R/3-System verbunden ist.</p>

## Fehler und Ausnahmen handhaben

8812	Das an den Funktionsbaustein übergebene Objekt ist kein initialisiertes Business-Objekt.
8813	Das an den Funktionsbaustein übergebene Objekt muß dieselbe Verbindung haben wie das Control.
8814	<method> ist kein BAPI.
8815	Parameter <parameter> der Methode <method> ist keine Struktur. Verwenden Sie elementare Datentypen.
8816	Logon-ActiveX-Control-Objekt oder Function-ActiveX-Control-Objekt oder Table-Factory-ActiveX-Control-Objekt konnte nicht angelegt werden.
8817	Fehler beim Zugriff auf persistenten Objektschlüsselparameter im BAPI <method>.
8818	Persistenter Schlüssel für Business-Objekt-Instanz vom Typ <object type> nicht gesetzt. BAPI <method> konnte nicht aufgerufen werden.
8819	Fehler beim Zugriff auf Parameter <parameter> oder zugeordnetes Sammelobjekt.
8820	Ausnahme im R/3-System ausgelöst während des Aufrufs von BAPI <method>. Ausnahme: Typ der aufgetretenen Ausnahme.
8821	Ungültige Typinformation für Objekttyp <object type>.

## Beispielanwendungen

Die nachfolgenden, in Visual Basic geschriebenen Anwendungen demonstrieren einen Großteil der Programmier Techniken, auf die Sie beim Einsatz des BAPI-ActiveX-Controls stoßen:

- [Beispiel: BAPIs aufrufen \[Seite 36\]](#)
- [Beispiel: Sammlungen von Business-Objekten handhaben \[Seite 38\]](#)

Um diese Beispielanwendungen ausführen zu können, müssen Sie alle notwendigen Controls - BAPI-ActiveX-Control, Logon-ActiveX-Control, Table-Factory-ActiveX-Control und Function-ActiveX-Control - auf Ihrer lokalen Maschine installiert und registriert haben. Dies wird automatisch durchgeführt, wenn Sie den SAP Desktop-SDK installieren, der auch das komplette Coding für die folgenden Zeilen enthält:

- [Beispiel: BAPIs aufrufen \[Seite 36\]](#):  
wdebapi.vbp, wdebapi.frm.
- [Beispiel: Sammlungen von Business-Objekten handhaben \[Seite 38\]](#):  
wdsched.vbp, wdsche1.frm, wdsche2.frm.



Weitere Informationen finden Sie auch im entsprechenden Kapitel des [Tutorial: Kommunikationsschnittstellen \[Extern\]](#).

**Beispiel: BAPIs aufrufen****Beispiel: BAPIs aufrufen**

Diese Visual Basic-Anwendung zeigt alle zum Aufruf eines BAPI erforderlichen Schritte.

Die Anwendung verwendet das Business-Objekt *SalesOrder*, das ein vertragliches Abkommen zwischen einer Verkaufsorganisation und einem Kunden in Bezug auf die Lieferung von Materialien oder die Bereitstellung von Dienstleistungen in definierter Menge zu definierten Preisen und in exakten Intervallen darstellt.

Die Anwendung ermöglicht es Ihnen, einen Kundenauftrag für ein bestimmtes Material anzulegen oder anzufragen, ob das Material verfügbar ist. Dazu muß das Business-Objekt *SalesOrder* im Business Object Repository (BOR) definiert sein.



Eine vollständige Beschreibung des Objekttyps *SalesOrder* finden Sie im Business Object Repository.

In diesem Beispiel wurde das BAPI-ActiveX-Control-Objekt angelegt, indem es zum Projekt hinzugefügt wurde, so daß es auf dem Formular sichtbar ist.

**Kundenauftrag im R/3 anlegen**

'Objektvariablen deklarieren:

```
Dim boOrder As Object          'Business-Objekt SalesOrder
Dim oPartners As Object 'Parameter OrderPartners der BAPI-Methode
Dim oHeader As Object         'Parameter OrderHeaderIn der BAPI-Methode
Dim oItemsIn As Object       'Parameter OrderItemsIn der BAPI-Methode
Dim oReturn As Object        'Parameter Return der BAPI-Methode
```

'Verbindung zum Business-Objekt *SalesOrder* herstellen

'(dies legt ein anonymes Objekt mit einem leeren Schlüsselfeld an):

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

'Struktur-/Tabellenobjekt holen:

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

'Header füllen:

```
oHeader.Value("DOC_TYPE") = "TA"          'Standardauftrag
oHeader.Value("SALES_ORG") = "0001" 'Verkaufsorganisation
oHeader.Value("DISTR_CHAN") = "01" 'Vertriebsweg
oHeader.Value("DIVISION") = "01"         'Sparte
oHeader.Value("PO_NUMBER") = ""         'Kundenbestellnummer
oHeader.Value("PRICE_DATE") = Now      'Datum
```

'Partner füllen:

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

---

**Beispiel: BAPIs aufrufen**

die den Auftrag bucht

```
oPartners.Value(1, "PARTN_NUMB") = "0000010096" 'Kundennummer
```

'Positionen füllen:

```
oItemsIn.Rows.Add
```

```
oItemsIn.Value(1, "REQ_QTY") = "0000000010000" 'Menge
```

```
oItemsIn.Value(1, "MATERIAL") = "BERLINER" 'Produkt-ID
```

```
oItemsIn.Value(1, "COND_VALUE") = "1432" 'Satz
```

'Methode aufrufen:

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

'Business-Objekte freigeben:

```
Set boOrder = Nothing
```

## Beispiel: Sammlungen von Business-Objekten handhaben

### Beispiel: Sammlungen von Business-Objekten handhaben

Diese Visual Basic-Anwendung beschafft die Terminplan-Informationen eines R/3-Benutzers und zeigt eine Zusammenfassung der Verabredungen dieses Benutzers an einem bestimmten Datum in einem Listenfenster an.

Für diese Anwendung benötigen Sie Objekte von vier verschiedenen Objekttypen:

#### Benötigte Objekttypen für Terminplan-Informationen

Business-Objekttyp	Beschreibung
USR01	SAP-Benutzer
SCHEDULER	Kalenderanwendung
SCHEDULE	Terminplan eines Benutzers
APPOINTMNT (ohne 'E' geschrieben)	Verabredung



Eine vollständige Beschreibung der einzelnen Objekttypen finden Sie im Business Object Repository.

Vom Objekttyp *SCHEDULER* gibt es nur eine Instanz. Seine Hauptfunktion besteht darin, *SCHEDULE*-Objekte anzulegen. (Im COM (Component Object Model)-Jargon: das Objekt vom Objekttyp *SCHEDULER* ist die Fabrik für Objekte vom Typ *SCHEDULE*.)

Während das BAPI-ActiveX-Control Instanzen von Business-Objekten anlegt, legt das *SCHEDULER*-Objekt Instanzen von *SCHEDULE*-Objekten an. Dies entspricht der MS-Anwendung Excel, die Tabellenkalkulationsobjekte anlegt.

Diese Fabrikkalenderobjekttypen sind ganz und gar das Ergebnis von Design-Entscheidungen. So liefert beispielsweise die Property *DominantOfToday* des Typs *SCHEDULE* Instanzen von *DOMINANT*-Objekten zurück.

#### Terminplan-Informationen eines R/3-Benutzers beschaffen

'Objektvariablen deklarieren:

```
Dim oBAPICtrl As Object           'BAPI-Control-Objekt
Dim oConnection As Object        'Verbindungsobjekt
Dim boSAPusr As Object           'SAP-Benutzer, dessen Verabredungen
                                 'benötigt werden
Dim boScheduler As Object        'Fabrikkalender für Terminpläne
Dim boSchedule As Object        'Terminplan des Benutzers boSAPusr
Dim oAppointments As Object     'Sammlung der Verabredungen
Dim datToDay as Date            'Datumsvariable
```

'BAPI-ActiveX-Control-Objekt anlegen:

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

Beispiel: Sammlungen von Business-Objekten handhaben

'Verbindungsobjekt holen:

```
oConnection = oBAPICtrl.Connection
```

'Benutzerspezifische Daten setzen:

```
oConnection.User = "MyUserName"
```

```
oConnection.Client = "000"
```

'Am R/3-System anmelden:

```
oConnection.Logon
```

'SAP-Benutzerobjekt holen:

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

'Kalenderanwendungsobjekt holen:

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

'Terminplan des SAP-Benutzers holen:

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

'Verabredungen des SAP-Benutzers holen:

```
datToDay = Date
```

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

'Sammlung der Verabredungen in Schleife verarbeiten und in einem Listenfenster anzeigen:

```
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
```

'Business-Objekte freigeben:

```
Set boSAPusr = Nothing
```

```
Set boScheduler = Nothing
```

```
Set boSchedule = Nothing
```

```
Set oAppointments = Nothing
```

'Abmelden:

```
oConnection.Logoff
```

'BAPI-ActiveX-Control-Objekt freigeben:

```
Set oBAPICtrl = Nothing
```

Objektbezug: BAPI-ActiveX-Control-Objekt

## Objektbezug: BAPI-ActiveX-Control-Objekt

### Beschreibung

Lokale Instanz des BAPI-ActiveX-Controls.

### Properties

- *Connection* (Object)

Setzt das Verbindungsobjekt, das alle notwendigen Informationen über den Status einer entfernten Verbindung zum R/3-System enthält, oder gibt es zurück; wird auch zum Aufbau einer Verbindung verwendet.

Beim Lesen wird das Verbindungsobjekt automatisch angelegt. Lese-/Schreibzugriff.
- *LogFileName* (String)

Setzt den Protokolldateinamen oder gibt ihn zurück.

Eigentlich gibt es zwei Protokolldateien:

  - **dev\_bapi.trc** für das BAPI-ActiveX-Control
  - **dev\_func.trc** für das Function-ActiveX-Control

Wenn kein Wert gesetzt ist, werden **dev\_bapi.trc** und **dev\_func.trc** in das aktuelle Verzeichnis geschrieben.
- *LogLevel* (Short)

Setzt die aktuelle Protokollebene oder gibt sie zurück.

Die Protokollebene hat einen Wertebereich von 0 (Trace deaktiviert) bis 9. Je höher der Wert, desto detaillierter sind die gelieferten Informationen.
- *BAPIExceptionCodes* (Bool)

Wenn in einem BAPI eine Ausnahme auftritt, legt die Property *BAPIExceptionCodes* fest, ob nur die Ausnahme-ID zurückgegeben werden soll oder ob als Fehlerbeschreibung im Ausnahmeobjekt zusätzliche Informationen geliefert werden sollen. In Visual Basic ist dies die Property *Description* des Objekts *Err*.

Dies bezieht sich nur auf die Ausnahme Nummer 8820 (Ausnahme aufgetreten während BAPI-Aufruf).
- *TransactionID* (String)

Setzt die aktuelle Transaktions-ID (TID) oder gibt sie zurück.

Die Property *TransactionID* wird durch den Aufruf der Methode *CreateTransactionID* gesetzt und nach dem erfolgreichen Ausführen des BAPI-Aufrufs gelöscht.

Um einen BAPI-Aufruf zu wiederholen, können Sie diese Property explizit auf die über die TID-Verwaltung beschaffte TID setzen.

### Methoden

Das BAPI-ActiveX-Control-Objekt hat folgende Methoden:

- *AboutBox()*

**Objektbezug: BAPI-ActiveX-Control-Objekt**

Zeigt das *About*-Fenster des Controls an.

- *Object CreateCollectionOfSAPObjects()*

Legt ein leeres Sammelobjekt an, in das Business-Objekte für eine spätere Verwendung als Parameter in einem Methodenaufruf hineingestellt werden können.

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

Legt eine lokale Instanz von Business-Objekten an.

Der Parameter *ObjectType* ist eine Zeichenkette, die Groß- und Kleinschreibung unterscheidet und dem Objekttyp eines im Business Object Repository (BOR) definierten Objekts entsprechen muß.

Die Parameter *ObjectKey* sind in der Regel optional. Wenn Sie allerdings eine lokale Instanz eines persistenten Business-Objekts anlegen, müssen Sie sie mit Werten versorgen, da sonst keine Objektschlüssel an den Funktionsbaustein übergeben werden.

Weitere Informationen finden Sie unter [Business-Objekte anlegen \[Seite 20\]](#).

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

Gibt ein bestimmtes Tabellen- oder Strukturobjekt zurück, das ein Parameter des gelieferten Methodennamens ist. Wenn das Objekt mit Daten gefüllt ist, kann es als Parameter an das geeignete BAPI übergeben werden. Alle Parameter sind Muß-Angaben.

Detaillierte Beschreibungen zu den einzelnen Parametern finden Sie unter [Daten an BAPIs übergeben \[Seite 23\]](#).

- *CreateTransactionID ( )*

Legt eine neue Transaktions-ID an und speichert den Wert in der Property *TransactionID*. Diese TID wird im nächsten BAPI-Aufruf verwendet.

Objektbezug: Sammelobjekt

## Objektbezug: Sammelobjekt

### Beschreibung

Sammlung lokaler Instanzen von SAP-Business-Objekten.

Sammelobjekte werden

- explizit mit der Methode *CreateCollectionOfSAPObjects* des BAPI-ActiveX-Control-Objekts angelegt.
- als ein Wert von einem BAPI zurückgegeben, das eine Folge von Business-Objekten zurückgibt.

Sammelobjekte werden als Parameter für Methoden benötigt, die Folgen von Objekten erwarten.

Neben den unten aufgeführten Properties und Methoden haben Sammelobjekte eine integrierte Iterationsunterstützung mit Hilfe von *For Each... Next*-Anweisungen in Visual Basic (COM's *\_NewEnum*-Methode).

### Properties

Sammelobjekte haben folgende Property:

- *Count*  
Gibt die Anzahl der Objekte in der Sammelobjekt zurück. Read-only.

### Methoden

Sammelobjekte haben folgende Methoden:

- *void Add(boObject)*  
Fügt ein Business-Objekt einem Sammelobjekt hinzu. Business-Objekte innerhalb eines Sammelobjekts haben normalerweise (aber nicht immer) denselben Objekttyp. So kann z.B. ein Client-Programm, das die Vorteile des Polymorphismus nützt, Sammelobjekte anlegen, die Objekte unterschiedlicher Typen enthalten.
- *Object Item(intIndex)*  
Standardmethode des Sammelobjekts. Gibt ein Business-Objekt aus dem Sammelobjekt zurück.  
Der Parameter *intIndex* ist eine Mußeingabe; er enthält die Indexnummer im Sammelobjekt. Der Indexbereich beginnt bei 0 und zählt -1.
- *void Remove(intIndex)*  
Entfernt ein Business-Objekt aus dem Sammelobjekt.  
Der Parameter *intIndex* ist eine Mußeingabe; er enthält die Indexnummer im Sammelobjekt.



Das folgende Visual Basic-Coding zeigt, wie ein Sammelobjekt verwendet wird:

'Sammelobjekt anlegen

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

'Objekt zum Sammelobjekt hinzufügen

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

'Sammelobjekt als Parameter in einer Methode verwenden

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

Objektbezug: Business-Objekt

## Objektbezug: Business-Objekt

### Beschreibung

Lokale Instanz eines SAP-Business-Objekts im R/3-System.

Wenn Sie eine lokale Instanz eines Business-Objekts anlegen, paßt sie sich automatisch der Schnittstelle des SAP-Business-Objekts an, das sie darstellt. Die Schnittstellen von Business-Objekten sind daher nur zur Laufzeit bekannt.

Weitere Informationen über die im R/3-System verfügbaren Objekttypen finden Sie im Business Object Repository (BOR).



Alle Properties sind gegenwärtig schreibgeschützt. Properties können nur über die Methode *Set* gesetzt werden (sofern implementiert).

Da alle Objekttypen die sogenannte *IFSAP*-Schnittstelle unterstützen, ist für jeden Objekttyp eine Mindestmenge an Properties und Methoden vorhanden.

### Properties

Business-Objekte haben mindestens die folgende Property:

- *Object ObjectType*  
Read-only. Gibt das Metaobjekt (Objekttypbeschreibung) des Business-Objekts zurück. Dies ist ein Objekt vom Typ *TOJTB*, das die Typinformationen enthält.

### Methoden

Business-Objekte haben mindestens folgende Methoden:

- *Display*  
Zeigt das Business-Objekt an.  
Wenn diese Methode für den Objekttyp implementiert ist, den Sie gerade betrachten, ruft sie normalerweise (aber nicht immer) eine Transaktion auf, die das Objekt auf einem SAPGUI-Bild anzeigt. Dies kann für Ihr Client-Programm vor- oder nachteilhaft sein.  
Wenn diese Methode für den Objekttyp, den Sie gerade betrachten, nicht implementiert ist, wird die Standardimplementierung von der *IFSAP*-Schnittstelle übernommen. Diese Standardimplementierung gibt den Schlüssel des Objekts zurück, was oft auch nicht besonders hilfreich ist.  
Ehe Sie die Methode *Display* auf ein Business-Objekt anwenden, sollten Sie ihre Implementierung für den Objekttyp prüfen, an dem Sie interessiert sind.
- *ExistenceCheck*  
Überprüft die Existenz eines Objekts in der R/3-Datenbank.  
Auch hier sollten Sie die Implementierung prüfen. Manche geben nichts zurück, andere geben einen Booleschen Wert zurück, wieder andere lösen eine Ausnahme aus, die im Client-Programm abgefangen werden muß.



Das Gerüst einer in Visual Basic geschriebenen Anwendung mit mehreren Property- und Methodenaufrufen finden Sie unter [Beispielanwendungen \[Seite 35\]](#).