

# BAPI-Programmierleitfaden (CA-BFA)



HELP.CABFABAPIG

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-Programmierleitfaden (CA-BFA)</b> .....	<b>5</b>
Einführung .....	6
Entwicklungsprozeß im Überblick.....	7
Analyse.....	9
Beschreibung des Szenarios .....	10
Definition des Szenarios im BAPI Explorer.....	22
Review des Szenarios.....	23
Design des BAPIs.....	24
Konventionen .....	25
Standardisierte BAPIs.....	26
Standardisierte Parameter.....	28
Design-Empfehlungen für Schnittstellen.....	30
Implementierung des BAPIs.....	31
Ablauf der Implementierung.....	32
Werkzeuge .....	34
Weiterführende Aspekte.....	36
Aktionen im ABAP Dictionary .....	37
Aktionen im Function Builder .....	40
Definition der Schnittstelle.....	41
Implementierung des Funktionsbausteins .....	43
Aktionen im BOR.....	54
Dokumentation des BAPIs .....	63
BAPI/ALE-Integration.....	71
Test und Freigabe .....	73
Test .....	74
Freigabe.....	77

## **BAPI-Programmierleitfaden (CA-BFA)**

Der vorliegende BAPI-Programmierleitfaden beschreibt den Entwicklungsprozeß für die Neuentwicklung von BAPIs. Darüberhinaus gelten die hier dargestellten Richtlinien auch für die Erweiterung und Modifikation bereits bestehender BAPIs.

## Einführung

# Einführung

## Business-Objekttypen und BAPIs

Zweck der **B**usiness **A**pplication **P**rogramming **I**nterfaces (BAPIs) ist es, den Zugriff auf SAP-Funktionalität über offizielle, stabile und dialogfreie Schnittstellen zu ermöglichen. Diese Schnittstellen sollen sowohl von anderen SAP-Komponenten als auch von externen Anwendungen, die von Kunden oder Complementary Software Partnern entwickelt werden, benutzt werden können.

BAPIs werden als API-Methoden von [SAP-Business-Objekttypen \[Extern\]](#) definiert. Diese Objekttypen werden wiederum im Rahmen des Business Framework-Ansatzes dazu verwendet, um eine objektbasierte Kommunikation zwischen Komponenten zu ermöglichen. Durch die Business-Objekte und ihre BAPIs wird es somit möglich, die Vorteile der Objektorientierung in der zentralen Informationsverarbeitung von Unternehmen zu nutzen. So wird beispielsweise die Wiederverwendung bestehender Funktionen und Daten, eine reibungslose technische Interoperabilität, ein flexibler Einsatz von Fremdkomponenten, etc. möglich. Mittels BAPIs können Anwendungen direkt auf die Anwendungsschicht des R/3-Systems zugreifen und als Client die betriebswirtschaftliche Logik des R/3-Systems nutzen. Dem Client bieten die BAPIs eine objektorientierte Sicht auf die Anwendungsobjekte an, um von Implementierungsdetails zu abstrahieren.

Die Entwicklung von BAPIs erfolgt stets über die Definition von Szenarien. Diese Szenarien werden verwendet, um systemübergreifende Geschäftsprozesse abzubilden und zu implementieren.

## Zielgruppe der Dokumentation

Diese Dokumentation beschreibt die Richtlinien, die SAP bei der Entwicklung und Implementierung von BAPIs einsetzt, um eine möglichst einheitliche BAPI-Entwicklung zu garantieren. Es wird somit gewährleistet, daß BAPIs für den Benutzer möglichst einfach verwendbar sind.

Der Programmierleitfaden richtet sich in an SAP-Entwickler, Partner und Kunden, die BAPIs implementieren.



Diese Dokumentation wurde für das Release 4.6C erstellt. Soweit nicht anders vermerkt, gelten diese Richtlinien jedoch auch bis einschließlich Release 4.0.

### Siehe auch:

[Terminologie](#)

[\[Extern\]Weiterführende Dokumentation zu BAPIs](#)

[\[Extern\]Änderungen zu Release 4.6C \[Extern\]](#)

## Entwicklungsprozeß im Überblick

Im folgenden wird der Entwicklungsprozeß für die Neuentwicklung von BAPIs beschrieben. Die dargestellten Richtlinien gelten jedoch auch für die Erweiterung und Modifikation bereits bestehender BAPIs.

Weitere Informationen zum Vorgehen bei Erweiterungen und Modifikationen erhalten Sie unter:

- [Kundenerweiterung von BAPIs \[Extern\]](#)
- [Modifikation von BAPIs \[Extern\]](#)
- [Weiterentwicklung freigegebener BAPIs durch SAP \[Extern\]](#)

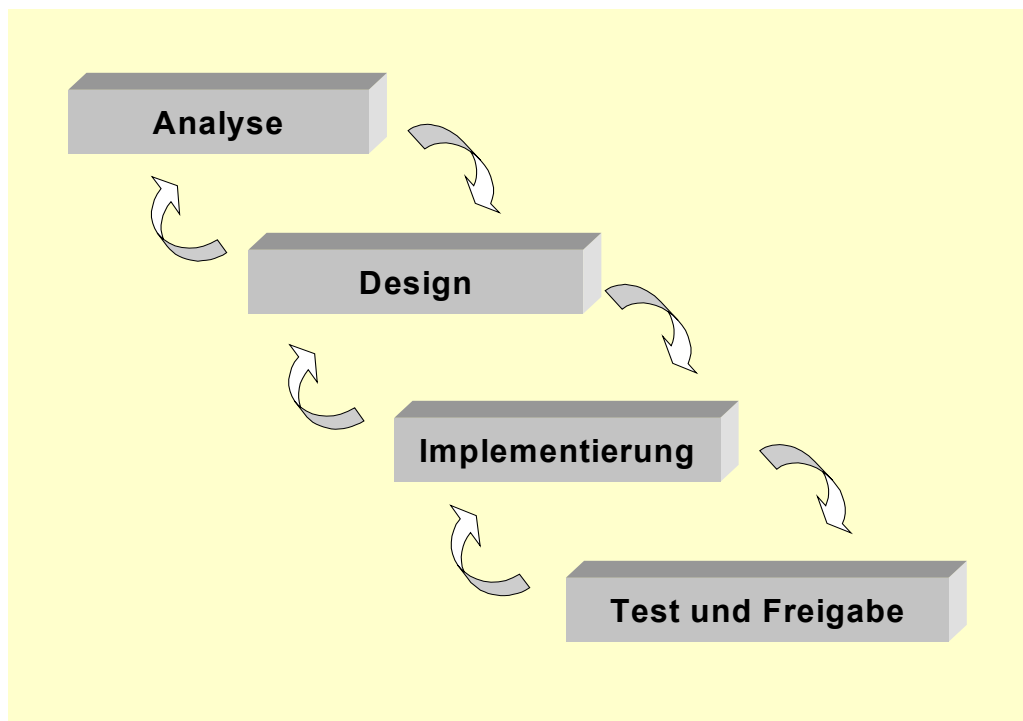
## Voraussetzungen

Um ein BAPI anhand dieses Programmierleitfadens zu implementieren, benötigen Sie folgende Kenntnisse:

- Grundkenntnisse über BAPIs.
- Kenntnisse der Programmiersprache ABAP.
- Allgemeine Kenntnisse des R/3-Systems.

## Ablauf

Der gesamte Entwicklungsprozeß läßt sich in die folgenden vier Phasen aufteilen:



## Entwicklungsprozeß im Überblick

1. **Analyse:**  
Bei der Analyse wird auf betriebswirtschaftlicher Ebene festgelegt, welcher Geschäftsprozeß mittels BAPIs realisiert werden soll. Dazu werden Szenarien definiert und anschließend wird identifiziert, welche Schritte innerhalb eines Szenarios extern (von anderen Komponenten) sichtbar sein sollen und also durch Business-Objekttypen und ihre BAPIs zu implementieren sind.
2. **Design:**  
Im Rahmen des Designs wird für jedes der an einem Szenario beteiligten BAPIs der Aufbau seiner Schnittstelle auf konzeptioneller Ebene definiert. Ziel des Designs sollte es sein, die Schnittstellen so kompakt und intuitiv zu gestalten, daß eine einfache Verwendung der BAPIs möglich ist. Gleichzeitig müssen die Schnittstellen auch allen technischen und formellen Richtlinien entsprechen.
3. **Implementierung:**  
Die Implementierung läßt sich in drei aufeinander aufbauende Bereiche aufteilen:
  - Definition der Datenstrukturen (inkl. Domänen und Datenelemente), welche den BAPI-Parametern zugrundeliegen.
  - Implementierung der Programmlogik in Funktionsbausteinen.
  - Definition des BAPIs im Business Object Repository (BOR), um einen objektorientierten Zugriff zu gewährleisten.
4. **Test und Freigabe:**  
Innerhalb des Testphase werden sowohl die Funktionalität und Verwendbarkeit eines BAPIs als auch seine Dokumentation getestet. Nach erfolgreich abgeschlossener Testphase kann das BAPI freigegeben werden.

## Organisatorische Unterstützung

Während des gesamten Entwicklungsprozesses leisten die **BFA Application Groups** der einzelnen Anwendungen in folgenden Punkten Unterstützung:

- Beim Design des Szenarios, der beteiligten Business-Objekttypen und der benötigten BAPIs
- Bei der technischen Realisierung
- Bei auftretenden Problemen

## Werkzeugunterstützung

Als zentrales Werkzeug für die BAPI-Entwicklung steht ab Release 4.6A der BAPI Explorer (Transaktion **BAPI**) zur Verfügung. Dieser ermöglicht es auf einfache Art, auf alle relevanten Informationen (sowohl auf Details und Dokumentation zu speziellen Business-Objekttypen und BAPIs als auch auf allgemeine Dokumentation im BAPI-Umfeld) zuzugreifen. Darüber hinaus integriert er alle für die BAPI-Entwicklung relevanten Tools und steuert über Projektformulare alle Phasen des Entwicklungsprozesses.

### Siehe auch:

Details zum [BAPI Explorer \[Extern\]](#)

## Analyse

Im Rahmen der Analyse müssen der Geschäftsprozeß und das diesen Prozeß implementierende Szenario konkret beschrieben und die im Szenario verwendeten [Business-Objekttypen \[Extern\]](#) und ihre BAPIs identifiziert werden. Die BAPIs sind dabei atomare Bausteine, mit deren Hilfe verschiedene Szenarien konstruiert werden können.

Es ergibt sich also folgende **zentrale Problemstellung** bei der Entwicklung von BAPIs: Einerseits sind BAPIs so zu gestalten, daß sie in verschiedenen Szenarien verwendet werden können, andererseits muß die Schnittstelle strukturiert und übersichtlich bleiben !!

Wichtig ist, daß in dieser Phase noch nicht festgelegt wird, wie die konkrete Signatur der einzelnen BAPIs aussehen soll. Es werden lediglich die für das zu realisierende Szenario benötigten BAPIs identifiziert und definiert, was diese BAPIs inhaltlich leisten müssen und welche Daten sie benötigen.

Während der Analyse können die folgenden drei Phasen unterschieden werden:

[Beschreibung des Szenarios](#)

[\[Seite 10\]Definition des Szenarios im BAPI Explorer](#)

[\[Seite 22\]Review des Szenarios](#)

[\[Seite 23\]](#)

## Beschreibung des Szenarios

# Beschreibung des Szenarios

## Der Geschäftsprozeß

Ein Geschäftsprozeß setzt einen betriebswirtschaftlichen Ablauf um und besteht aus einer Abfolge von Einzelfunktionen. Er sollte auf einer inhaltlichen Ebene beschrieben werden und ist unabhängig von jeglichen DV-technischen Details.



Kundenauftragserstellung in einer Filiale mit Bonitätsprüfung in der Zentrale.

Bei der Beschreibung des Geschäftsprozesses müssen folgende Teilaufgaben erledigt werden:

- Definition der Aufgabe und des Umfangs des Geschäftsprozesses.
- Identifizierung der einzelnen Schritte des Geschäftsprozesses. Dies kann beispielsweise über Aufstellen eines Prozeßmodells oder über Use Cases erfolgen.

## Das Szenario

Ein Szenario ist eine DV-technische Realisierung eines Geschäftsprozesses. Es beschreibt die Aufgabenverteilung und Interaktion zwischen den beteiligten Komponenten. Deshalb kann es mehrere Szenarien geben, die den gleichen Geschäftsprozeß umsetzen.



Kundenauftragserfassung und Bonitätsprüfung in verschiedenen R/3-Systemen (Zentrales Rechnungswesen, dezentraler Vertrieb).

Bei der Beschreibung des konkret zu implementierenden Szenarios sind folgende Aspekte zu beachten:

I  
d  
e  
n  
t  
i  
f  
i  
z  
i  
e  
r  
u  
n  
g  
d  
e  
r  
b  
e  
t  
e  
i  
l  
l  
i  
g  
t  
e  
n  
K  
o  
m  
p  
o  
n  
e  
n  
t  
e  
n  
u  
n  
d  
i  
h  
r  
e  
r  
j  
e  
w  
e  
i  
l  
l  
i  
g  
e  
n  
A  
u  
f  
g  
a  
b  
e  
n

**Beschreibung des Szenarios**

E  
r  
m  
i  
t  
t  
l  
u  
n  
g  
,  
o  
b  
ü  
b  
e  
r  
d  
a  
s  
S  
z  
e  
n  
a  
r  
i  
o  
A  
n  
w  
e  
n  
d  
u  
n  
g  
s  
s  
y  
s  
t  
e  
m  
e  
i  
n  
t  
e  
g  
r  
i  
e  
r  
t  
o  
d  
e  
r  
a  
l  
t  
e  
r  
n  
a  
t  
i

F  
e  
s  
t  
l  
e  
g  
u  
n  
g  
d  
e  
s  
l  
i  
n  
f  
o  
r  
m  
a  
t  
i  
o  
n  
s  
-  
u  
n  
d  
P  
r  
o  
z  
e  
ß  
f  
l  
u  
s  
s  
e  
s  
.

- Es wird festgelegt, welche Schritte komponentenübergreifend durchgeführt werden und welche Schritte innerhalb einer Komponente abzuhandeln sind.
- Insbesondere ist zu definieren, welche Daten zwischen welchen Komponenten ausgetauscht werden und wer diesen Austausch initiiert.
- Festlegung, in welcher Reihenfolge die einzelnen Schritte abgearbeitet werden.

**Beschreibung des Szenarios**

- Identifizierung der Schritte, die in einer Transaktion (LUW) zusammengefaßt werden.



**Beispiel:** Der Entwickler muß sich beispielsweise die Frage stellen, ob es sinnvoll ist, innerhalb einer LUW einen Kunden anzulegen und anschließend in der gleichen LUW einen Kundenauftrag zu erzeugen.

Siehe auch [Transaktionsmodell für die BAPI-Entwicklung \[Extern\]](#).

- Die Fehlerbehandlung muß wesentlich präziser und umfangreicher als bei lokalen Anwendungen sein.

F  
ü  
r  
d  
a  
s  
S  
z  
e  
n  
a  
r  
i  
o  
i  
s  
t  
z  
u  
e  
n  
t  
s  
c  
h  
e  
i  
d  
e  
n  
,  
o  
b  
d  
i  
e  
K  
o  
p  
p  
l  
u  
n  
g  
d  
e  
r  
S  
y  
s  
t  
e  
m  
e  
n  
g

d  
e  
r  
l

**Beschreibung des Szenarios**

- E  
s  
m  
ü  
s  
s  
e  
n  
a  
l  
l  
e  
P  
e  
r  
f  
o  
r  
m  
a  
n  
c  
e  
-  
k  
r  
i  
t  
i  
s  
c  
h  
e  
n  
S  
c  
h  
r  
i  
t  
t  
e  
i  
d  
e  
n  
t  
i  
f  
i  
z  
i  
e  
r  
t  
w  
e  
r  
d  
e  
n  
.

E  
s  
i  
s  
t  
z  
u  
b  
e  
r  
ü  
c  
k  
s  
i  
c  
h  
t  
i  
g  
e  
n  
,  
w  
e  
l  
c  
h  
e  
R  
/  
3  
-  
R  
e  
l  
e  
a  
s  
e  
s  
i  
n  
n  
e  
r  
h  
a  
l  
b  
d  
e  
s  
S  
z  
e  
n

**Beschreibung des Szenarios**

- F  
ü  
r  
j  
e  
d  
e  
s  
S  
z  
e  
n  
a  
r  
i  
o  
i  
s  
t  
e  
i  
n  
V  
e  
r  
a  
n  
t  
w  
o  
r  
t  
l  
i  
c  
h  
e  
r  
z  
u  
b  
e  
n  
e  
n  
n  
e  
n  
,  
d  
e  
r  
d  
i  
e  
K  
o  
r  
r  
e  
k  
t  
h  
e  
i  
t

## Die Business-Objekttypen und BAPIs

Jede an dem Szenario beteiligte Komponente muß Services anbieten, so daß die komponentenübergreifenden Schritte durchgeführt werden können. Es ist nun zu analysieren, wie für diese Services die Verantwortung auf Business-Objekttypen und ihre BAPIs verteilt werden kann. So wird beispielsweise für die externe Anlage eines Kundenauftrages im R/3-System das BAPI *CreateFromData* am Business-Objekttyp *SalesOrder* verwendet.



Für weitere Details siehe auch: [Business-Objekttypen \[Extern\]](#).

Für jede beteiligte Komponente sind zunächst die benötigten Business-Objekttypen zu ermitteln. Dabei sind folgende Aspekte zu beachten bzw. Frage zu beantworten:

- Kapselung der benötigten Funktionalität in Business-Objekttypen. Dabei geht es um eine geeignete Zerlegung des Gesamtsystems nach verschiedenen Verantwortlichkeiten. Die Zerlegung bzw. Kapselung soll dabei eindeutig und disjunkt erfolgen.
- Existieren bereits Business-Objekttypen für diese Verantwortlichkeiten?
- Gibt es bestehende Design-Pattern?

Es ist zu untersuchen, ob dieses oder ein ähnliches Problem bereits gelöst wurde.



**Beispiel:** Kopf-/Positionspattern, wie beispielsweise bei *SalesOrder*, *PurchaseOrder*, etc.

- Abgrenzung der Verantwortlichkeit gegenüber anderen Business-Objekttypen.
- Festlegung der Services, die aufgrund der Verantwortlichkeit zur Verfügung gestellt werden.



Für weitere Details siehe auch: [Designkriterien für Business-Objekte \[Extern\]](#).

Für jeden identifizierten Business-Objekttyp ist zu ermitteln, wie die ihm zugeordneten Services über BAPIs realisiert werden können.

Dabei sind folgende Aspekte zu beachten:

- Jeder zu erbringende Service wird durch ein oder mehrere BAPIs (= Methode am Business-Objekttyp) realisiert.
- BAPIs stellen allgemeine Funktionalität eines Business-Objekttyps zur Verfügung. Dazu sollten sie weitgehend unabhängig von einzelnen Szenarien und in verschiedenen Szenarien verwendbar sein.



Um die Verwendbarkeit eines BAPIs in verschiedenen Szenarien zu vereinfachen, sollte die BAPI-Signatur so aufgebaut sein, daß die Parameter und Felder des BAPIs nach Anwendungsfällen getrennt angeordnet sind, sofern dies möglich ist.

## Beschreibung des Szenarios

- Allerdings kann die Art des Szenarios Auswirkungen auf die Granularität der BAPIs haben. Es läßt sich die Integration von Anwendungssystemen von der Integration von alternativen Frontends unterscheiden:

### 1. Integration von Anwendungssystemen

- Die Integration von Anwendungssystemen wird typischerweise über eine Programm-zu-Programm-Kommunikation, lose asynchrone Kopplung und den Austausch größerer Datenmengen realisiert.
- Die Hauptanforderung an die in diesen Szenarien verwendeten Business-Objekttypen und BAPIs ist die Gewährleistung einer hohen Performance, die z.B. über eine Minimierung der Anzahl der Aufrufe erreicht wird.
- Die Konsequenz ist eine grobe Granularität der Anwendung, die durch umfangreichere BAPIs realisiert wird.
- **Beispiel:** Automatisches Anlegen eines Kundenauftrages durch ein Programm. Die Realisierung erfolgt über den Business-Objekttyp *SalesOrder* mit dem BAPI *CreateFromData*. Folglich wird der Auftrag zunächst vollständig im Sendesystem erzeugt und anschließend als Ganzes an das Empfängersystem geschickt.

### 2. Integration alternativer Frontends

- Szenarien mit alternativen Frontends stellen eine Mensch-Maschine-Kommunikation dar und können sowohl synchron als auch asynchron realisiert werden.
- Anforderungen an die Gestaltung der Business-Objekttypen und BAPIs sind die Gewährleistung von Flexibilität, Konfigurierbarkeit und eine Minimierung von Fehlersituationen.
- Die Konsequenz ist eine feinere Granularität der Anwendung, welche der Dialogverarbeitung im R/3 entsprechen sollte.
- **Beispiel:** Dialogbasiertes Anlegen eines Kundenauftrages im Internet. Die Realisierung erfolgt z.B. über die beiden Methoden *CreateFromData* und *AddItem* am Business-Objekttyp *SalesOrder*. Dabei legt die Methode *CreateFromData* in diesem Fall lediglich den Auftragskopf an, während mit Hilfe der Methode *AddItem* sukzessive Auftragspositionen hinzugefügt werden können.



#### SAP-intern:

Wo immer möglich, sollte ein BAPI so gestaltet werden, daß es für **beide** Verwendungszwecke verwendet werden kann. Das oberste Ziel sollte daher sein, für jeden Business-Objekttyp eine Reihe von standardisierten BAPIs zu entwickeln, die universell einsetzbar sind.

- Das Szenario muß so gestaltet werden, daß es möglich ist, sämtliche für einen BAPI-Aufruf vom R/3-System benötigten Informationen vorher durch andere BAPIs zu bekommen.
- Gibt es Customizing-abhängige BAPIs, so müssen BAPIs zur Verfügung gestellt werden, um diese Customizing-Einstellungen auszulesen.

## Beschreibung des Szenarios

  **SAP-intern**

Das Szenario, die Einbindung in das Objektmodell der Anwendung und das Design neuer BusinessObjekttypen wurde mit einem der BFA AG-Ansprechpartner abgesprochen.

Wurden während der Analyse noch nicht vorhandene Business-Objekttypen identifiziert, dann werden diese Objekttypen ausschließlich von den BFA AGs angelegt. Hierfür ist ein Antrag auf Neuanlage eines Business-Objekttyps im [BAPI Explorer \[Extern\]](#) zu stellen.

Bestehende Beispiele für BAPI-Szenarien finden Sie in der ALE-Geschäftsprozeß-Bibliothek unter [Überblick über ALE-Integrationsszenarien \[Extern\]](#).

---

**Definition des Szenarios im BAPI Explorer**

## Definition des Szenarios im BAPI Explorer

### Voraussetzungen

Das Szenario ist vollständig beschrieben.

### Ablauf

Definieren Sie ein Projektformular im BAPI Explorers wie folgt:

1. Starten Sie dazu den BAPI Explorer über die Transaktion **BAPI** im jeweiligen Entwicklungssystem (ab Release 4.6A).
2. Wählen Sie die Registerkarte *Projekt* aus und legen ein Projekt zur Implementierung neuer BAPIs an.
3. Anschließend führt Sie ein Projektformular durch den gesamten Erstellungsprozeß des BAPIs. Im ersten Abschnitt können dort die Grunddaten zum definierten Szenario angelegt werden.

#### Siehe auch:

Details zum [BAPI Explorer \[Extern\]](#)

## Review des Szenarios

Bevor das Szenario umgesetzt und mit einer konkreten Definition und Implementierung der BAPIs begonnen wird, sollte das Szenario einer Prüfung unterzogen werden. Zu diesem Review des BAPI-Szenarios sollten alle betroffenen Ansprechpartner und die Qualitätssicherung aus der relevanten Abteilung hinzugezogen werden.

### Funktionsumfang

Im Rahmen dieses Reviews sollten folgende Sachverhalte geprüft werden:

- Ist das Szenario in der geplanten Form sinnvoll ?
- Sind alle bei der Beschreibung des Szenarios zu erfüllenden Aufgaben korrekt erledigt worden ?
- Spielen die am Szenario beteiligten BAPIs reibungslos zusammen ?

Erst nach erfolgreichem Abschluß des Reviews sollte mit der Entwicklung der BAPIs fortgesetzt werden.

**Siehe auch:**

[Design des BAPIs \[Seite 24\]](#)

## Design des BAPIs

### Design des BAPIs

Nachdem in der Analyse das Szenario und die darin verwendeten Business-Objekttypen und BAPIs identifiziert wurden, erfolgt innerhalb der Design-Phase die Definition der BAPI-Signatur auf konzeptioneller Ebene. Dabei sollte eine Orientierung an inhaltlichen Sachverhalten erfolgen und eine Festlegung der Parameternamen und des Parameteraufbaus vorgenommen werden.

Da BAPIs betriebswirtschaftliche und keine technischen Schnittstellen darstellen, ergibt sich folgende **zentrale Forderung** an die Entwicklung von BAPIs:

Ein Verwender muß in einer externen Anwendung allein durch sein Anwendungswissen und die BAPI-Dokumentation einen BAPI-Aufruf korrekt implementieren können. Es dürfen keine R/3-Kenntnisse von ihm erwartet werden, so daß er das R/3-System als "Black Box" ansehen kann.

In dieser Phase ist auch zu entscheiden, ob das BAPI aufgrund seiner Funktionalität eine Instanzmethode oder Klassenmethode ist und ob es als ein [standardisiertes BAPI \[Seite 26\]](#) implementiert werden kann.

**Instanzmethoden** beziehen sich auf eine konkrete Instanz eines Business-Objekttyps; z.B. liefert das BAPI *SalesOrder.GetDetail* die Detaildaten zu genau einem (zu spezifizierenden) Kundenauftrag. Im BOR werden diese Methoden als **instanzabhängig** definiert.

**Klassenmethoden** beziehen sich auf keine konkrete Instanz eines Business-Objekttyps; z.B. liefert das BAPI *SalesOrder.GetList* eine Liste aller vorhandenen Kundenaufträge, die bestimmten Kriterien entsprechen. Zu den Klassenmethoden zählen auch standardisierte Create-Methoden. Eine Create-Methode legt eine neue Instanz an; z.B. legt das BAPI *SalesOrder.Create* einen neuen Kundenauftrag im R/3-System an. Im BOR werden diese Methoden als **instanzunabhängig** definiert.

Die konkreten Instanzen eines Business-Objekttyps werden über dessen Schlüsselfelder identifiziert, weshalb diese in diesem Zusammenhang eine besondere Rolle spielen. In der Design-Phase sollten bei instanzabhängigen BAPIs die Schlüssel als besondere Parameter gekennzeichnet werden. Für weitere Details siehe [Definition der Schnittstelle \[Seite 41\]](#).

Zu beachten ist beim Design der BAPI-Signatur darüber hinaus:

1. Es müssen bestimmte **Konventionen** beim Entwurf der Methode (des BAPIs) und der Parameter eingehalten werden. Siehe auch [Konventionen \[Seite 25\]](#).
2. Es gibt eine Reihe von **Standardmethoden und -parametern**, die bestimmte Grundfunktionen bereitstellen. Fallen die zu implementierenden BAPIs und deren Parameter unter diese Kategorie, so gibt es Vorschriften, wie diese gestaltet werden sollen. Einen Überblick erhalten Sie in [Standardisierte BAPIs \[Seite 26\]](#) und [Standardisierte Parameter \[Seite 28\]](#).
3. Damit die Signatur übersichtlich und leicht verwendbar wird, sollten [Design-Empfehlungen \[Seite 30\]](#) beachtet werden.

## Konventionen

Folgende Konventionen müssen auf der **Ebene der BAPI-Methode** eingehalten werden:

- Handelt es sich beim zu implementierenden BAPI um ein standardisiertes BAPI, dann sind die generischen Namen, wie z.B. *GetList*, *GetDetail*, etc. zu verwenden.
- Der Methodenname muß in englischer Sprache sein (maximal 30 Zeichen).
- Die einzelnen Namensteile eines BAPI-Namens werden durch den Wechsel von Groß- und Kleinschreibung identifiziert.  
**Beispiel:** *GetList*.
- Unterstriche („\_“) im BAPI-Namen sind verboten.
- Jedes BAPI besitzt einen Return-Parameter, welcher entweder ein Export-Parameter oder eine Export-Tabelle ist.
- Um Kundenerweiterungen des BAPIs zu ermöglichen, muß jedes BAPI einen *ExtensionIn*- und einen *ExtensionOut*-Parameter besitzen. Siehe auch: [Kundenerweiterung von BAPIs \[Extern\]](#).

**Siehe auch:** [Standardisierte BAPIs \[Seite 26\]](#)

Auf **Parameter-Ebene** gelten diese Konventionen:

- Werden standardisierte Parameter verwendet, dann müssen die für Standard-Parameter festgelegten Namen benutzt werden.
- BAPI-Parameternamen sollten möglichst aussagekräftig sein.  
Beispiele für schlecht gewählte Namen sind Abkürzungen oder technische Namen (z.B. "Flag", Tabellennamen, etc.).
- Die Parameter- und Feldnamen müssen in englischer Sprache mit maximal 30 Zeichen sein.
- Die einzelnen Namensteile eines Parameter-Namens werden durch den Wechsel von Groß- und Kleinschreibung getrennt, um größtmögliche Lesbarkeit zu erreichen.  
**Beispiel:** *CompanyCodeDetail*.
- Semantisch zusammengehörende Werte sollten zu strukturierten Parametern zusammengefaßt werden, anstatt mehrere skalare Parameter zu verwenden.
- Bei ISO-relevanten Feldern (Land, Sprache, Menge, Währung) sind zusätzliche Felder für ISO-Codes vorhanden.
- Allen Mengenfeldern müssen Mengeneinheitsfelder mitgegeben und allen Währungsbetragsfeldern müssen Währungsbezeichnungen mitgegeben werden.

**Siehe auch:** [Standardisierte Parameter \[Seite 28\]](#)

## Standardisierte BAPIs

# Standardisierte BAPIs

## Verwendung

Es gibt BAPIs, die für die meisten der SAP-Business-Objekttypen angelegt werden können, da sie bestimmte Grundfunktionen bereitstellen. Diese sollten nach Möglichkeit an allen Business-Objekttypen gleich implementiert werden. Standardisierte BAPIs erleichtern die Verwendung und Verhindern ein unübersichtliches Maß an verschiedenen BAPIs. Wenn immer es möglich ist, muß ein standardisiertes BAPI einem individuellen BAPI vorgezogen werden.

## Funktionsumfang

Es werden die folgenden standardisierten BAPIs bereitgestellt:

### Lesen der Instanzen von Business-Objekttypen

- **GetList ( )**  
Das BAPI *GetList( )* dient zur Selektion einer Menge von Objektschlüsselwerten z.B. Buchungskreisen oder Materialnummern und weiteren sinnvollen Werten. Das BAPI *GetList( )* ist eine Klassenmethode.  
Weitere Informationen finden Sie unter [Programmieren von GetList\(\)-BAPIs \[Extern\]](#).
- **GetDetail ( )**  
Mit dem BAPI *GetDetail( )* werden Detailinformationen zu einer Instanz eines Business-Objekttyps gesammelt und an das aufrufende Programm zurückgegeben. Die Instanz wird über ihren Schlüssel identifiziert. Das BAPI *GetDetail( )* ist eine Instanzmethode.  
Weitere Informationen finden Sie unter [Programmieren von GetDetail\(\)-BAPIs \[Extern\]](#).

### BAPIs zum Erstellen, Ändern oder Löschen von Instanzen eines Business-Objekttyps

Die folgenden BAPIs zum selben Objekttyp sind so zu programmieren, daß sie innerhalb einer Transaktion wiederholt aufrufbar sind. Wird zum Beispiel nach erfolgreichem Anlegen des Kundenauftrags 1, innerhalb der Transaktion ein weiterer Kundenauftrag 2 angelegt, so darf der zweite BAPI-Aufruf die Konsistenz des Kundenauftrags 2 nicht beeinflussen. Nach Abschluß der Transaktion mit *Commit Work* werden die beiden Aufträge auf der Datenbank konsistent verbucht.

- **Create ( ) oder CreateFromData ( )**  
Das BAPI *Create( )* oder *CreateFromData( )* legt eine Instanz zu einem SAP-Business-Objekttyp, z.B. eine Bestellung an. Diese BAPIs sind Klassenmethoden.  
Weitere Informationen finden Sie unter [Programmieren von Create\(\)-BAPIs \[Extern\]](#).
- **Change ( )**  
Mit dem BAPI *Change( )* wird eine bestehende Instanz eines SAP-Business-Objekttyps geändert, z.B. eine Bestellung. Das BAPI *Change( )* ist eine Instanzmethode.  
Weitere Informationen finden Sie unter [Programmieren von Change\(\)-BAPIs \[Extern\]](#).
- **Delete ( ) und Undelete ( )**  
Das BAPI *Delete( )* löscht eine komplette Instanz eines SAP-Business-Objekttyps von der Datenbank oder setzt eine Löschvormerkung.  
Das BAPI *Undelete( )* nimmt eine Löschvormerkung zurück. Diese BAPIs sind Instanzmethoden.  
Weitere Informationen finden Sie unter [Programmieren von Delete\(\)-BAPIs \[Extern\]](#).

## Standardisierte BAPIs

- **Cancel ( )**  
Im Gegensatz zum BAPI *Delete( )* wird mit dem BAPI *Cancel( )* eine Instanz eines Business-Objektyps storniert, d.h. die zu stornierende Instanz bleibt auf der Datenbank erhalten und als Storno wird eine zusätzliche Instanz erzeugt. Das BAPI *Cancel( )* ist eine Instanzmethode.  
Weitere Informationen finden Sie unter [Programmieren von Cancel\(\)-BAPIs \[Extern\]](#).
- **Add<Teilobjekt> ( ) und Remove<Teilobjekt> ( )**  
Das BAPI *Add<Teilobjekt>( )* fügt ein Teilobjekt zu einer existierenden Objektinstanz hinzu; entsprechend entfernt das BAPI *Remove<Teilobjekt>( )* ein Teilobjekt von einer Objektinstanz. Diese BAPIs sind Instanzmethoden.  
Weitere Informationen finden Sie unter [Programmieren von Methoden auf Teilobjekten \[Extern\]](#).

**BAPIs für die Massendatenverarbeitung**

Die obigen BAPIs zum Erstellen oder Verändern von Daten können auch für die Massenverarbeitung eingesetzt werden.

Weitere Informationen finden Sie unter [BAPIs für den Massendatentransfer \[Extern\]](#).

**BAPIs für die Replikation von Business-Objektinstanzen**

- **Replicate ( ) und SaveReplica ( )**  
Die BAPIs *Replicate( )* und *SaveReplica( )* werden als Methoden von replikationsfähigen Business-Objektypen implementiert. Sie ermöglichen es, bestimmte Instanzen eines Objekttyps zusätzlich auf ein oder mehreren anderen Systemen zur Verfügung zu stellen. Diese BAPIs werden vorwiegend für die Datenübertragung zwischen verteilten Systemen im Rahmen von Application Link Enabling (ALE) eingesetzt. Diese BAPIs sind Klassenmethoden.  
Weitere Informationen finden Sie unter [Programmieren von Replicate\(\)/SaveReplica\(\)-BAPIs \[Extern\]](#).

**Weitere, seltener verwendete standardisierte BAPIs**

- [Programmieren von GetStatus\(\)-BAPIs \[Extern\]](#)
- [Programmieren von ExistenceCheck\(\)-BAPIs \[Extern\]](#)

## Standardisierte Parameter

# Standardisierte Parameter

## Verwendung

Es gibt Parameter, die für verschiedene BAPIs angelegt werden können, da sie in allen BAPIs gleiche oder äquivalente Daten bereitstellen. Diese sollten nach Möglichkeit in allen BAPIs gleich implementiert werden.

## Funktionsumfang

### Adressenparameter

Für Adressenparameter sind bei BAPIs bestimmte Referenzstrukturen definiert. Diese Strukturen sollten Sie für die Verwendung in Ihrem BAPI kopieren, insbesondere dann, wenn der zugrundeliegende Objekttyp die zentrale Adreßverwaltung (ZAV) benutzt.

Weitere Informationen finden Sie unter [Adressenparameter \[Extern\]](#).

### Änderungsparameter

In BAPIs, die Änderungen auf der Datenbank hervorrufen, z.B. Change()- und Create()-BAPIs, muß man zwischen den Parameterfeldern unterscheiden können, die geänderte Werte enthalten und denen, die unverändert geblieben sind. Dies geschieht durch die Verwendung von standardisierten Änderungsparametern.

Weitere Informationen finden Sie unter [Änderungsparameter \[Extern\]](#).

### Extension-Parameter

Mit dem Parameter *ExtensionIn* bzw. *ExtensionOut* wird Kunden ein Mechanismus zur Verfügung gestellt, der die Erweiterung eines BAPIs ohne Modifikation ermöglicht.

Weitere Informationen finden Sie unter [Extension-Parameter \[Extern\]](#) und [Kundenerweiterung von BAPIs \[Extern\]](#).

### Return-Parameter

Jedes BAPI muß über einen Export-Parameter *Return* verfügen, mit dem Rückmeldungen an die aufrufende Anwendung zurückgegeben werden. Um Anwendungsprogrammierern eine einheitliche Fehlerbehandlung bei BAPI-Aufrufen zu ermöglichen, müssen alle Return-Parameter auf die gleiche, standardisierte Weise implementiert werden.

Weitere Informationen finden Sie unter [Return-Parameter \[Extern\]](#).

### Selektionsparameter

Standardisierte Selektionsparameter werden in BAPIs verwendet, mit denen nach bestimmten Instanzen eines Business-Objekttyps gesucht werden kann (z.B. in *GetList()*). Diese Parameter ermöglichen es dem Aufrufer des BAPIs, die entsprechenden Selektionskriterien anzugeben.

Weitere Informationen finden Sie unter [Selektionsparameter \[Extern\]](#).

### TestRun-Parameter

Der Parameter *TestRun* wird in den schreibenden BAPIs (*Create()* und *Change()*) verwendet, um vor dem tatsächlichen Anlegen der Objektinstanz auf der Datenbank die Angaben zur Instanz zu überprüfen. Das Anlegen der Objektinstanz wird dabei nur simuliert und die Daten werden nicht fortgeschrieben.

Weitere Informationen finden Sie unter [TestRun-Parameter \[Extern\]](#).

**Textübergabeparameter**

Für die Übergabe von Dokumentationstexten mittels eines BAPI (z.B. die Dokumentation zu einem Business-Objekttyp), müssen standardisierte Textübergabeparameter angelegt werden. Weitere Informationen finden Sie unter [Textübergabeparameter \[Extern\]](#).

## Design-Empfehlungen für Schnittstellen

## Design-Empfehlungen für Schnittstellen

Damit BAPIs intuitiv und leicht zu benutzen sind, sollten folgende Empfehlungen beim Design der Schnittstelle beachtet werden:

- Die Gruppierung der Daten zu Parametern sollte nach betriebswirtschaftlichen Gesichtspunkten erfolgen.
- Die Eingabeschnittstelle sollte gut strukturiert und intuitiv verständlich sein. Eine Strukturierung der Parameter ist in erster Linie nach Anwendungsfällen vorzunehmen.



**Beispiel:** Strukturierte Parameter sollten so aufgebaut sein, daß sie zunächst alle Felder, die für den einfachsten Anwendungsfall benötigt werden als einen logischen Block enthalten. Zusätzliche, für Spezialfälle benötigte Felder, sollten nach Anwendungsfällen gruppiert, an diesen "Grundblock" angehängt werden.

- Es dürfen keine für externe Verwender nicht interpretierbare Felder an der Schnittstelle erscheinen. Dazu muß versucht werden, von internen, R/3-spezifischen Details zu abstrahieren.
- Wo immer möglich, sollten **Werteilfen** (F4-Hilfen) an Parametern bzw. Parameterfeldern bereitgestellt werden. Dies ermöglicht dem Client auf einfache Weise, durch Aufruf des BAPIs *Helpvalues.GetList*, alle für ein Feld möglichen Eingabewerte zu ermitteln.  
**Siehe auch:** [Bereitstellung der Werteilfe \[Extern\]](#)
- Abhängigkeiten zwischen Datenfeldern untereinander und zwischen Parametern sind in der Dokumentation zu erfassen.
- Sämtliche von einem BAPI gelieferten Informationen sollten sinnvoll und ohne größere Umordnung der Daten weiterverarbeitet werden können. So sollten Export-Strukturen möglichst wieder als Input eines anderen BAPIs verwendet werden können.
- Schlüsselfeldern von *GetList*- und *GetDetail*-BAPIs sollten Textfelder mitgegeben werden.



Gibt es zu einem Schlüsselfeld mehrere Texte, so sollten diese Texte in einer separaten Tabelle abgelegt werden.

- Bei der Gestaltung der Schnittstelle ist der Verwendungszweck der BAPIs zu berücksichtigen:
  - Bei *BAPIs für Batch-Verarbeitung* richtet sich das Design in erster Linie nach Performance-Aspekten.
  - *BAPIs für alternative Front-Ends* benötigen eine flexiblere Schnittstelle, da sie i.d.R. in komplizierteren Abläufen verwendet werden.



Siehe auch: [Beispiel \[Extern\]](#) für das konzeptionelle Design eines BAPIs.

## Implementierung des BAPIs

Nachdem der Entwurf des zu implementierenden BAPIs auf konzeptioneller Ebene abgeschlossen ist, werden innerhalb der Implementierungsphase SAP-spezifische Sachverhalte relevant. So werden jetzt beispielsweise für die BAPI-Parameter die jeweiligen SAP-Datenstrukturen bestimmt und die Programmlogik realisiert.

Die nachfolgende Dokumentation gibt zunächst eine Übersicht über den Ablauf der Implementierung, in deren Rahmen die für BAPIs relevanten Komponenten und Werkzeuge vorgestellt werden. Anschließend werden die einzelnen Schritte detailliert beschrieben.

### **Siehe auch:**

[Ablauf der Implementierung \[Seite 32\]](#)

[Aktionen im ABAP Dictionary \[Seite 37\]](#)

[Aktionen im Function Builder \[Seite 40\]](#)

[Aktionen im BOR \[Seite 54\]](#)

[Dokumentation des BAPIs \[Seite 63\]](#)

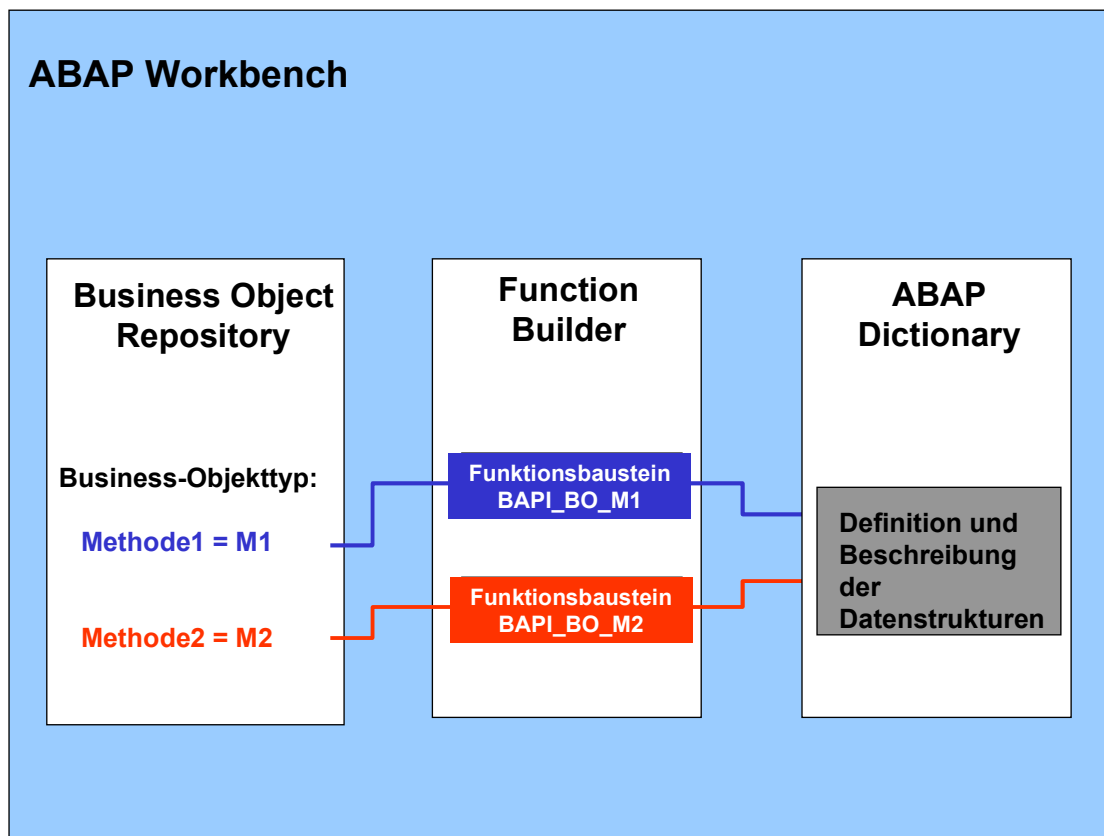
[BAPI/ALE-Integration \[Seite 71\]](#)

## Ablauf der Implementierung

## Ablauf der Implementierung

Ein BAPI wird im Business Object Repository (BOR) als API-Methode zu einem SAP-Business-Objekttyp definiert. Die Business-Objekttypen und ihre BAPIs werden im BOR angelegt und beschrieben. Ein BAPI wird standardmäßig als RFC-fähiger Funktionsbaustein implementiert. Dieser wird im Function Builder angelegt und dort beschrieben. Die Definitionen und Beschreibungen der Datenstrukturen, die ein BAPI verwendet, sind im ABAP Dictionary hinterlegt.

Diesen Zusammenhang verdeutlicht die folgende Grafik:



## Ablauf

Diese Strukturierung innerhalb der ABAP Workbench bedingt den Ablauf der Implementierung:

1. [Definition der Datenstrukturen \[Seite 37\]](#) (inkl. der Domänen und Datenelemente) im ABAP Dictionary.
2. [Implementierung des Funktionsbausteins \[Seite 40\]](#) im Function Builder
3. [Definition des Business-Objekttyps und seiner Methoden \[Seite 54\]](#) im BOR

**Ablauf der Implementierung**

Alle drei Phasen werden von sorgfältiger [Dokumentation \[Extern\]](#) begleitet, die in den zugehörigen Werkzeugen angelegt wird.



Alle notwendigen Arbeitsschritte können vom BAPI Explorer aus angesteuert werden.

**Siehe auch:**

[Werkzeuge \[Seite 34\]](#)

[Weiterführende Aspekte \[Seite 36\]](#)

## Werkzeuge

# Werkzeuge

Die wichtigsten Werkzeuge für die Entwicklung von BAPIs sind der BAPI Explorer, das ABAP Dictionary, der Function Builder und das Business Object Repository.

## Der BAPI Explorer

Seit Release **4.6A** ist der [BAPI Explorer \[Extern\]](#) das zentrale Tool für die BAPI-Entwicklung. Er kann mit dem Transaktionscode **BAPI** oder im *SAP Easy Access* über den Pfad *SAP Menü* → *Werkzeuge* → *Business Framework* → *BAPI Explorer* gestartet werden. Als zentraler Einstiegspunkt in die BAPI-Welt bietet er Zugriff auf alle relevanten Informationen und den direkten, kontextsensitiven Absprung in alle benötigten Tools/Transaktionen. Außerdem werden Sie mittels Projektformularen schrittweise durch den Entwicklungsprozeß geführt. Dabei erhalten Sie wichtige Hinweise in Form von Checkboxes und Informationen zu vorhandenen Werkzeugen. Ferner haben Sie die Möglichkeit, den Fortschritt Ihres Projektes zu protokollieren.

## Das ABAP Dictionary

Das ABAP Dictionary ermöglicht eine zentrale Beschreibung und Verwaltung aller im System verwendeten Datendefinitionen. Neu erfaßte oder geänderte Informationen werden allen Systemkomponenten automatisch zur Verfügung gestellt. Dies sorgt für Datenintegrität, Datenkonsistenz und Datensicherheit.



Weitere Informationen zum ABAP Dictionary finden Sie in der [Dokumentation zum ABAP Dictionary \[Extern\]](#)

## Der Function Builder

Mit dem Function Builder können innerhalb einer Funktionsgruppe Funktionsbausteine angelegt, implementiert, getestet und dokumentiert werden. Der Function Builder enthält eine Funktionsbibliothek, die als zentrale Ablage für alle Funktionsbausteine dient.



Weitere Informationen zum Function Builder finden Sie in der Dokumentation zum [Function Builder \[Extern\]](#).

## Das Business Object Repository (BOR)

Das Business Object Repository (BOR) beinhaltet die SAP-Business-Objekttypen und SAP-Interfacetypen sowie deren Komponenten wie z.B. Methoden, Attribute und Ereignisse. Im BOR wird ein BAPI als eine Methode eines SAP-Business-Objekttyps definiert. Im Zusammenhang mit den SAP-Business-Objekttypen und BAPIs übernimmt das BOR folgende Aufgaben:

- Es ermöglicht eine objektorientierte Sichtweise auf die Daten und Prozesse des Systems R/3. Der Zugriff auf R/3-Anwendungsfunktionen erfolgt über die Methoden (BAPIs) der SAP-Business-Objekttypen. Dabei werden die Implementierungsdetails gekapselt; nur die Schnittstellenfunktionalität der Methode ist nach außen hin sichtbar.

- Es ordnet die verschiedenen Schnittstellen entsprechend der Komponentenhierarchie. Dadurch ist ein schnelles und einfaches Auffinden der gesuchten Funktionalität möglich.
- Es übernimmt die Versionskontrolle für die BAPIs.  
Die Erweiterung einer BAPI-Schnittstelle durch zusätzliche Parameter wird im BOR dokumentiert. Dadurch ist jederzeit die Rekonstruktion einer früheren Schnittstellenversion möglich. Beim Anlegen eines BAPIs wird das Auslieferungsrelease der freigegebenen Version des BAPIs gespeichert. Das gleiche erfolgt beim Anlegen jedes Schnittstellenparameters.  
Der Funktionsbaustein, der einem BAPI zugrundeliegt, unterliegt der Versionskontrolle des Function Builders.
- Es garantiert eine stabile Schnittstelle.  
Wenn Änderungen an der BAPI-Schnittstelle im BOR vorgenommen werden, wird automatisch geprüft, ob diese Änderungen gegenüber den entsprechenden Entwicklungsobjekten im ABAP Dictionary syntaktisch kompatibel sind.



Weiterführende Informationen zum BOR finden Sie in der Dokumentation zum [Business Object Builder \[Extern\]](#).

### Ausblick

Zunächst müssen also die Namen, Parameter und Eigenschaften des BAPIs, sowie die Strukturen im ABAP Dictionary, auf denen das BAPI beruht, identifiziert werden. Erst wenn diese erforderlichen Informationen gegeben sind, kann das BAPI im Function Builder implementiert und die benötigten Programmierobjekte im BOR angelegt werden.

## Weiterführende Aspekte

## Weiterführende Aspekte

Beachten Sie bitte vor Einstieg in die Implementierungsphase auch die folgenden Aspekte:

**Freigeben von BAPIs**

Über die Freigabe wird die Verfügbarkeit des BAPIs als vollständig implementierte Methode eines Business-Objektyps bekannt gemacht. Darüber hinaus wird das BAPI so vor inkompatiblen Änderungen geschützt, denn alle Änderungen an einem freigegebenen BAPI werden automatisch im BOR und im ABAP Dictionary auf Kompatibilität geprüft und bei Inkompatibilität abgewiesen.



**SAP-intern**

Soll das BAPI in begründeten Fällen vorübergehend nur intern verwendet werden, also für den Kunden als nicht freigegeben gekennzeichnet sein, dann muß der Status des Funktionsbausteins auf *intern freigegeben* gesetzt werden.

Die Freigabe des BAPIs muß nach Absprache und in Zusammenarbeit mit den BFA AGs und der Qualitätssicherung der betroffenen Entwicklungsabteilung erfolgen.

**Erweiterung, Weiterentwicklung und Modifikation von BAPIs**

Soll kein neues BAPI angelegt, sondern ein bestehendes erweitert, weiterentwickelt oder modifiziert werden, dann beachten Sie die diesbezüglichen Anforderungen.

Weitere Informationen finden Sie unter [Weiterentwicklung, Modifikationen ... \[Extern\]](#)

**BAPIs für den Outbound-Fall**

Bis einschließlich Release 4.0 wurden BAPIs ausschließlich als Funktionsbausteine im gleichen System implementiert, in welchem sie definiert wurden.

Ab Release **4.5A** können BAPIs auch Schnittstellen beschreiben, die in einem Nicht-SAP System implementiert sind, jedoch vom R/3-System aus aufgerufen werden können.

Diese BAPIs werden als Outbound-BAPIs bezeichnet und werden im BOR als API-Methoden an SAP-Interfacetypen definiert. Bei [BAPIs für den Outbound-Fall \[Extern\]](#) sind demnach immer getrennte Systeme für Definition (Client) und Implementierung (Server) vorhanden.

## Aktionen im ABAP Dictionary

Während des ersten Schrittes der Implementierungsphase sind im ABAP Dictionary alle Datenstrukturen (inklusive Domänen und Datenelementen) zu definieren, welche für die Parameter des zu implementierenden BAPIs benötigt werden. Den Absprung in das ABAP Dictionary finden Sie im Projektformular des BAPI Explorers.

Folgende Konventionen sind dabei von Bedeutung:

### Konventionen für BAPI-Datenstrukturen

- Jeder Parameter muß sich auf eine Datenstruktur im ABAP Dictionary beziehen. Bei strukturierten Parametern ist dies immer die gesamte BAPI-Datenstruktur. Besteht der Parameter dagegen nur aus einem Feld, muß er auf ein Feld in einer BAPI-Datenstruktur verweisen.
- Für BAPIs müssen eigene Datenstrukturen angelegt werden, die unabhängig von den Datenstrukturen sind, die allgemein in der R/3-Anwendung verwendet werden.  
**Begründung:** Bei Freigabe des BAPIs werden die dem BAPI zugrundeliegenden Strukturen eingefroren und können nur noch beschränkt erweitert werden. Siehe auch [Weiterentwicklung freigegebener BAPIs \[Extern\]](#).



Bestehende, interne Strukturen können auf bestehende BAPI-Strukturen mit einem Feldzuweisungsbaustein abgebildet werden, der durch ein Mapping Tool (Transaktion BDBS) automatisch generiert wird.

Weitere Details finden Sie unter [Mapping zwischen Internen/Externen Datenformaten \[Extern\]](#)

- Alle Datenstrukturnamen müssen mit **<Namensraum>BAPI** beginnen.
- BAPI-Strukturnamen sollten möglichst aussagekräftig sein.
- Es dürfen keine APPENDs und INCLUDEs in BAPI-Datenstrukturen benutzt werden.  
**Begründung:** Die Verwendung von APPENDs und INCLUDEs kann zu [Inkompatibilitäten \[Extern\]](#) bei der Erweiterung der betroffenen BAPI-Datenstrukturen führen.
- Für Felder sollten möglichst bestehende, zentrale Datenelemente und Domänen verwendet werden.



Am [Beispiel \[Extern\]](#) des BAPIs *CompanyCode.GetDetail* wird gezeigt, wie die Datenstrukturen zu definieren sind.

## Aktionen im ABAP Dictionary

### Konventionen für Feldnamen

- Die Felder in Strukturen müssen leicht verständliche, englische Namen mit maximal 30 Zeichen besitzen.



#### SAP-intern

Verwenden Sie den Report BAPIFELD, um zu einer ABAP Dictionary-Struktur eine Liste von Vorschlägen für englische Feldnamen zu erzeugen. Eine Beschreibung, wie dieser Report zu verwenden ist, finden Sie in der Report-Dokumentation.

- Für jedes Datenelement sollte ein sinnvoller, englischer Defaultfeldname mit maximal 30 Zeichen hinterlegt werden.



#### SAP-intern

Der Defaultname wird in Zukunft immer von dem SAP-internen Report BAPIFELD vorgeschlagen.

**Vorgehensweise:** Aus dem Report BAPIFELD können Sie die noch zu bearbeitenden Datenelemente entnehmen. Pflegen Sie dann im ABAP Dictionary über *Definition* → *Defaultfeldname* den Defaultfeldnamen des Datenelements ein.

### Konventionen für Werthilfen

- An der Domäne sind evtl. Einzelwerte bzw. eine Wertetabelle für den geeigneten Einsatz der F4-Hilfe zu pflegen.
- Alle sinnvollen [Werthilfen \[Extern\]](#) (die über das Service-BAPI *Helpvalues.GetList* nutzbar sind) müssen an den Datenstrukturen/Datenelementen gepflegt werden. Dazu sind an den Feldern einer BAPI-Struktur Fremdschlüssel zu spezifizieren. Ist an der Domäne zu einem Feld eine Wertetabelle hinterlegt, ist ein Fremdschlüssel obligatorisch.

### Technische Konventionen

- Für alle Parameterfelder wird die interne Datendarstellung verwendet. Siehe auch [Mapping zwischen Internen/Externen Datenformaten \[Extern\]](#).
- Wird auf der Datenbank sowohl ein interner als auch ein externer Schlüssel abgelegt, muß an der BAPI-Schnittstelle immer der externe Schlüssel verwendet werden.
- Bei ISO-relevanten Feldern (Land, Sprache, Mengeneinheit, Währung) sind zusätzliche Felder für ISO-Codes vorhanden.
- Alle Währungsfelder verwenden die Domäne BAPICURR. In Spezialfällen kann auch die Domäne BAPICUREXT verwendet werden.
- Als Dezimalzeichen muß immer der Dezimalpunkt verwendet werden.

## Aktionen im ABAP Dictionary

- Alle Mengen- und Währungsbeträge haben den Dezimalpunkt an der korrekten Stelle.  
**Begründung:** Das BAPI hat stets eine einheitliche, kommagerechte Betragsdarstellung. Für die Konvertierung von Währungsbeträgen verwenden Sie deshalb die Funktionsbausteine  
BAPI\_CURRENCY\_CONV\_TO\_EXTERNAL und  
BAPI\_CURRENCY\_CONV\_TO\_INTERNAL.
- Bei Mengenfeldern muß das Referenzfeld/die Referenztabelle auf das Mengeneinheitensfeld korrekt gesetzt werden.
- Alle Datenelemente für Datumsfelder haben das Format YYYYMMDD.

## Konventionen für die BAPI/ALE-Integration

Damit aus einem BAPI ein IDoc für die asynchrone Kommunikation generiert werden kann, müssen folgende Bedingungen erfüllt sein:

- BAPI-Strukturnamen sollten nicht länger als 27 Zeichen sein, da ansonsten der automatisch generierte Name für das zugehörige Segment zu lang und somit eine manuelle Nachpflege notwendig wird.  
**Begründung:** Der Segmentname wird nach der Konvention <BAPI-Strukturname> + <dreistellige Zahl> generiert.
- Einzelne Felder in einer Datenstruktur dürfen maximal nur 250 Bytes lang sein.

## Konventionen für Kundenentwicklungen

- Beim Anlegen von Datenstrukturen (Domänen oder Datenelementen) durch IBUs, Partner und Kunden sind die von SAP vorgegebenen [Namensräume \[Extern\]](#) einzuhalten.

## Aktionen im Function Builder

### Aktionen im Function Builder

Nach Abschluß der Parameterdefinition wird der dem BAPI zugrundeliegende Funktionsbaustein im Function Builder (Transaktion SE37) implementiert. Der Absprung in die SE37 erfolgt wiederum innerhalb des Projektformulars im BAPI Explorer.

Beim [Anlegen des Funktionsbausteins \[Extern\]](#) sind folgende Regeln einzuhalten:

- Alle Funktionsbausteine entsprechen der folgenden Namenskonvention:  
**<Namensraum>BAPI\_<Business-Objekt>\_<Methode>.**  
Dabei sind maximal 30 Zeichen erlaubt.  
Bei Bedarf können Sie geeignet, jedoch im Einklang mit der obigen Konvention, abkürzen.  
Die Zuordnung zum Business-Objekttyp muß erkennbar bleiben.
- Alle BAPIs an einem SAP Business-Objekttyp sollten in einer Funktionsgruppe zusammengefaßt werden. Von dieser Regel sollte nur in Ausnahmefällen abgewichen werden.
- BAPIs an verschiedenen Objekttypen dürfen nicht zu einer Funktionsgruppe zusammengefaßt werden.

### Die nächsten Schritte

[Definition der Schnittstelle \[Seite 41\]](#)

[Implementierung des Funktionsbausteins \[Seite 43\]](#)

## Definition der Schnittstelle

Ein BAPI-Parameter muß sowohl im Funktionsbaustein als auch in der Methode zum Business-Objekttyp im BOR definiert werden. Die Definition muß in beiden Fällen, bis auf die Schlüsselfelder des Business-Objekttyps, identisch sein.

Die [Schlüsselfelder \[Extern\]](#) eines Business-Objekttyps dienen sowohl zur semantischen als auch der technischen Identifizierung einer Instanz. Die Schlüssel werden stets als Gesamtheit (Gesamtschlüssel) und nicht in Teilen verwendet. Dies hat wiederum sowohl semantische Gründe (Rolle des Schlüssels für die Identität einer Instanz) als auch technische Gründe (im BOR werden verschiedene Generierungen und Checks bei vorhandenen Schlüsselfeldern vorgenommen).

Bezieht sich ein BAPI auf eine konkrete Instanz des Business-Objekttyps, werden bei der Methode im BOR die Schlüsselfelder des Business-Objekts zur Identifizierung der Instanz sowie die anderen Parameter übergeben. Im BOR werden in diesem Fall die Schlüsselfelder nicht zusätzlich unter den Methodenparametern aufgeführt. Im Funktionsbaustein kann eine solche Unterscheidung nicht getroffen werden, weshalb dort die Schlüsselfelder als Parameter mit aufgeführt werden müssen.

Im Zusammenhang mit Schlüsselfeldern gelten daher für die Schnittstellen der Funktionsbausteine die folgenden Konventionen:

- Bei **Instanzmethode**n sind alle BOR-Schlüsselfelder obligatorische Importparameter im Funktionsbaustein. Kein BOR-Schlüsselfeld darf als Exportparameter im Funktionsbaustein definiert sein.
- Bei **Klassenmethode**n dürfen keine BOR-Schlüsselfelder als Exportparameter im Funktionsbaustein vorkommen (Ausnahme: Create-Methoden). Es dürfen auch keine BOR-Schlüsselfelder als Importparameter im Funktionsbaustein vorkommen.
- Bei **Create-Methoden** liegen alle BOR-Schlüsselfelder der Methode als Exportparameter im Funktionsbaustein vor. Es dürfen (wie bei allen Klassenmethoden) keine BOR-Schlüsselfelder als Importparameter im Funktionsbaustein vorkommen. Create-Methoden sind im BOR als instanzunabhängig klassifiziert.

Wenn laut obigen Richtlinien Schlüsselfelder als Parameter im Funktionsbaustein vorkommen, dann sind folgende Konventionen zwingend:

- Es wird der Gesamtschlüssel und keine Teilschlüssel verwendet.
- Für jedes Schlüsselfeld des Business-Objekttyps existiert ein eigener Parameter am Funktionsbaustein.
- Der Parameter am Funktionsbaustein und das BOR-Schlüsselfeld sind namensgleich.



In den vorangegangenen Erklärungen verstehen wir unter dem Schlüsselfeld einen Parameter, der zur Identifizierung einer Laufzeitinstanz (im OO-Sinne) dient und mit dem (einem) Schlüsselfeld des Business-Objekttyps namensgleich ist. Es ist aber natürlich möglich, durch andere Parameter die betriebswirtschaftlich notwendigen Informationen zur Verfügung zu stellen, die durch den Schlüssel repräsentiert werden.

## Definition der Schnittstelle

**Beispiel:** Bei einem BAPI *Create* am Business-Objektyp *CompanyCode* (Schlüssel *CompanyCodeID*) darf nach obigen Richtlinien kein Import-Parameter *CompanyCodeID* vorkommen. Der Name des anzulegenden Buchungskreises könnte aber statt dessen z.B. als Feld *ID* in einem Parameter *CompanyCodeData* übergeben werden.

Darüber hinaus müssen Funktionsbaustein-Schnittstellen zusätzlich diese Konventionen erfüllen:

- Für die Parameter *ExtensionIn* und *ExtensionOut* muß die Struktur BAPIPAREX verwendet werden.
- Dem [Return-Parameter \[Extern\]](#) ist die Struktur BAPIRET2 zugrunde zu legen.
- Der Funktionsbaustein muß RFC-fähig sein.
- Unterstriche ("\_") in Parameternamen sind nicht zulässig.



Im Function Builder ist nur Großschreibung möglich, so daß es hier keine Trennung der Namensteile der Parameter durch Wechsel der Groß- und Kleinschreibung gibt.

## Implementierung des Funktionsbausteins

Bei der Codierung des Funktionsbausteins müssen eine Reihe von BAPI-spezifischen Anforderungen eingehalten werden:

### Anforderungen an das Transaktionsmodell

- Ab Release 4.0 darf ein BAPI kein COMMIT WORK-Kommando mehr absetzen.  
**Begründung:** Die Transaktionssteuerung soll beim Aufrufer liegen. Mehrere BAPIs sollen innerhalb einer LUW kombiniert werden können. Weitere Informationen finden Sie unter [Transaktionsmodell für die BAPI-Entwicklung \[Extern\]](#).  
Darüber hinaus dürfen folgende Befehle nicht verwendet werden:
  - CALL TRANSACTION
  - SUBMIT REPORT
  - SUBMIT REPORT AND RETURN
- Datenbankveränderungen dürfen nur über die Verbuchung durchgeführt werden.  
**Begründung:** Der RFC führt einen impliziten Datenbank-Commit durch.
- Der globale Speicher darf nicht zur Übergabe von Werten verwendet werden.  
**Begründung:** Es sollen Seiteneffekte vermieden und eine Kapselung der Funktionalität unterstützt werden.
- Erfolgt nach einem BAPI-Aufruf kein **COMMIT WORK**, muß eine gesetzte Sperre explizit gelöscht werden.
- Bei allen Tabellen, die nicht über SAP-Sperren geschützt werden, muß die Reihenfolge der Fortschreibung immer gleich sein, um ein gegenseitiges Sperren (Deadlock) zu verhindern.

### Anforderungen durch die Remote-Fähigkeit von BAPIs

- Ein BAPI darf keine Ausgabe auf einem Bildschirm hervorrufen, d.h. es darf keine Listen, Abfragen oder Dialogfenster erzeugen. Dies gilt für das BAPI selbst und für alle Funktionsbausteine, die indirekt von dem BAPI aufgerufen werden.  
**Begründung:** Ein externer Client kann auf eine Bildschirmausgabe nicht reagieren. Im schlimmsten Fall kann es sogar zum Abbau der Verbindung im Client-Programm kommen.
- Jedes BAPI muß bei Bedarf eigene Berechtigungsprüfungen durchführen. Speziell sollten [Wertehilfen \[Extern\]](#) ebenfalls Berechtigungsprüfungen enthalten.

---

Implementierung des Funktionsbausteins

## Anforderungen an das Error-Handling

I  
m

B  
A  
P  
I  
w  
e  
r  
d  
e  
n  
k  
e  
i  
n  
e  
E  
x  
c  
e  
p  
t  
i  
o  
n  
s  
v  
e  
r  
w  
e  
n  
d  
e  
t.  
S  
t  
a  
t  
t  
d  
e  
s  
s  
e  
n  
w  
e  
r  
d  
e  
n  
a  
l  
l  
e  
F

h  
l  
e  
r

Implementierung des Funktionsbausteins

E  
i  
n  
B  
A  
P  
I  
d  
a  
r  
f  
k  
e  
i  
n  
e  
n  
P  
r  
o  
g  
r  
a  
m  
m  
a  
b  
b  
r  
u  
c  
h  
e  
r  
z  
e  
u  
g  
e  
n  
.  
E  
s  
d  
a  
r  
f  
n  
u  
r  
i  
m  
R  
e  
t  
u  
r  
n  
-  
P  
a  
r

D  
a  
s  
E  
r  
r  
o  
r-  
H  
a  
n  
d  
l  
i  
n  
g  
s  
p  
i  
e  
l  
t  
e  
i  
n  
e  
g  
r  
ö  
ß  
e  
r  
e  
R  
o  
l  
l  
e  
a  
l  
s  
b  
e  
i  
n  
o  
r  
m  
a  
l  
e  
n  
F  
u  
n  
k  
t  
i  
o  
n  
s

---

**Implementierung des Funktionsbausteins**

- Verwendete Fehlernummern dürfen nicht in ihrer Bedeutung verändert werden.
- Fehlermeldungen müssen für den Verwender in einer externen Anwendung verständlich sein. So sollten zum Beispiel keine Tabellennamen zurückgegeben werden.
- Aus der Fehlermeldung muß klar hervorgehen, ob sie betriebswirtschaftliche (inhaltliche) Gründe hat oder ob sie auf fehlenden Import-Daten beruht.

**Beispiel** für eine ungeeignete Fehlermeldung: "Mußfeld nicht gefüllt". Die Fehlermeldung muß hier klar spezifizieren, welche Mußfelder nicht gefüllt wurden.

## Anforderungen an die Performance

Implementierung des Funktionsbausteins

D  
a  
s  
B  
A  
P  
I  
s  
o  
ll  
p  
e  
rf  
o  
r  
m  
a  
n  
t  
g  
e  
s  
t  
a  
lt  
e  
t  
w  
e  
r  
d  
e  
n  
.  
D  
e  
s  
h  
a  
l  
b  
s  
o

ll  
t  
e  
n

D  
a  
d  
e  
r  
F  
u  
n  
k  
t  
i  
o  
n  
s  
b  
a  
u  
s  
t  
e  
i  
n  
,  
d  
e  
r  
e  
i  
n  
e  
m  
  
B  
A  
P  
I  
z  
u  
g  
r  
u  
n  
d  
e  
l  
i  
e  
g  
t,  
ü  
b  
e  
r  
R  
F  
C

## Implementierung des Funktionsbausteins

### **Besondere Handhabung großer Datenmengen**

Massendaten werden in der ABAP-Programmierung, bei der ein SAPGUI als Frontend dient, anders behandelt als bei der Programmierung auf externen Entwicklungsplattformen wie beispielsweise Visual Basic.

Werden innerhalb des Systems R/3 Massendaten gelesen, z.B. eine Liste mit sehr vielen Einträgen, so bleibt ein Großteil dieser Daten auf dem Anwendungs-Server. Es werden nur die Daten an das Frontend geschickt, die dort auch tatsächlich angezeigt werden. Bei der Programmierung mit Visual Basic dagegen werden in dieser Situation alle Daten vom Anwendungsserver an das Client-System übertragen. Dies hat eine stärkere Netzbelastung und höheren Speicherbedarf beim Client-System zur Folge.

Für den Fall, daß von Ihrem BAPI große Datenmengen gelesen werden sollen, geben Sie dem BAPI möglichst eine Begrenzung mit, so daß beispielsweise nur die ersten  $n$  Sätze gelesen werden. Alternativ oder zusätzlich hierzu könnte das BAPI dem aufrufenden Programm auch über eine Rückmeldung mitteilen, daß die Datenmenge zu groß ist und eine Nachselektion erfolgen sollte.

- Um die **Dauer von Sperren zu reduzieren**, sollten Sie Nummern nicht einzeln vergeben. Nutzen Sie stattdessen die gepufferte Vergabe, d.h. lesen Sie die Nummern in einen Puffer und vergeben Sie diese direkt aus dem Puffer.
- Sperrzeiten können auch vermindert werden, indem Sie Datenbankmanipulationen möglichst nahe am Aufruf des COMMIT WORK-Kommandos durchführen. Dadurch verhindern Sie, daß die Sperre zu lange steht und eventuell andere Prozesse blockiert.
- Je weniger selektiv oder spezifisch der (Teil-)Schlüssel eines geänderten Datensatzes ist, umso wahrscheinlicher ist es, daß viele BAPIs diesen Satz verändern und damit sperren. Zum Beispiel wird sich eine Statistik auf Werksebene sehr schlecht auf die Performance auswirken, wogegen eine Statistik auf Werks-/Materialebene für weitaus weniger BAPIs relevant sein wird und somit weniger Sperren hervorruft.
- Seien Sie sparsam mit lesenden Transaktionen, die von einem Datenbank-Commit abhängig sind (Committed Read), da eine lesende Transaktion auf die Abarbeitung des COMMIT WORK-Kommandos der ändernden Transaktion warten muß.

## Weitere Anforderungen

- Bei Tabellen, für die keine Eingabe erwartet wird, muß ein Refresh gemacht werden.
- Verzicht auf Get- und Set-Parameter.

**Begründung:** Es sollen Seiteneffekte vermieden und eine Kapselung der Funktionalität unterstützt werden.

- BAPIs sollten von Customizing-Einstellungen so unabhängig wie möglich sein. Insbesondere dürfen Customizing-relevante Werte nicht verändert werden.

---

**Implementierung des Funktionsbausteins**

- Es müssen Customer-Exits vorgesehen werden, um Kundenerweiterungen von BAPIs zu ermöglichen:

Jeder BAPI-Funktionsbaustein muß neben den durch die Anwendung vorgegebenen Customer-Exits zwei zusätzliche Exits enthalten.

Der erste Customer-Exit soll dem Kunden die Möglichkeit geben, alle an das BAPI übergebenen Daten zu prüfen. Speziell sollte eine Prüfung des Inhalts des ExtensionInParameter durchgeföhrt werden, bevor dieser in die Tabellenerweiterungen geschrieben wird.

Der zweite Customer-Exit wird zur Verfügung gestellt, damit der Kunde weitere Verarbeitungen (z.B. Schreiben zusätzlicher Tabellen) durchföhren kann. Hat der Kunde die Export-Parameter des BAPIs erweitert, sollte der zweite Customer-Exit zusätzlich dazu verwendet werden, den [ExtensionOut-Parameter \[Extern\]](#) zu füllen.

Siehe auch: [Kundenerweiterung von BAPIs \[Extern\]](#).

---

## Aktionen im BOR

### Aktionen im BOR

Nachdem der Funktionsbaustein vollständig implementiert wurde, müssen Sie jetzt das BAPI als Methode eines Business-Objektyps definieren. Neben der Ablage im zentralen Business Object Repository (BOR) ermöglichen Sie damit einen objektorientierten Zugriff auf das BAPI und einen möglichen Anschluß an das ALE-Verteilungsmodell (im asynchronen Fall).

Der Absprung in das BOR erfolgt wiederum innerhalb des Projektformulars im BAPI Explorer.

### Ablauf

Bei der Definition sind die folgenden Schritte auszuführen:

1 D  
. e  
n  
b  
e  
n  
ö  
t  
i  
g  
t  
e  
n  
S  
A  
P  
:-  
B  
u  
s  
i  
n  
e  
s  
s  
:-  
O  
b  
j  
e  
k  
t  
t  
v  
p  
i  
d  
e  
n  
t  
i  
f  
i  
z  
i  
e  
r  
e  
n  
[  
E  
x  
t  
e  
r  
n  
].

Aktionen im BOR

2 D  
. a  
s  
B  
A  
P  
l  
a  
l  
s  
A  
P  
l-  
M  
e  
t  
h  
o  
d  
e  
a  
n  
d  
e  
n  
B  
u  
s  
i  
n  
e  
s  
s  
-  
O  
b  
j  
e  
k  
t  
t  
y  
p  
a  
n  
h  
ä  
n  
g  
e  
n  
.  
H  
i  
e  
r  
f  
ü  
r  
s  
t  
e

## Aktionen im BOR

- Groß-/Kleinschreibung ist zu beachten (jedes neue Wort in Großbuchstaben). Siehe auch: [Beispiel \[Extern\]](#).
- Das Import- und Exportverhalten der Tabellenparameter ist korrekt im BOR gepflegt.  
**Begründung:** Im Gegensatz zum Funktionsbaustein kann man im BOR auch zwischen Import- und Export-Tabellen unterscheiden. Sie sollten daher die Standardoption *Import/Export* nur wählen, wenn die Tabelle tatsächlich importiert **und** exportiert wird.
- Der Return-Parameter ist stets mit *Export* definiert.

Aktionen im BOR

N  
e  
b  
e  
n  
d  
e  
r  
V  
e  
r  
e  
i  
n  
f  
a  
c  
h  
u  
n  
g  
d  
e  
r  
A  
r  
b  
e  
i  
t  
d  
u  
r  
c  
h  
d  
e  
n  
W  
i  
z  
a  
r  
d  
s  
t  
e  
l  
l  
t  
d  
i  
e  
s  
e  
r  
a  
u  
c  
h  
s  
i  
c

▪

Aktionen im BOR

-

▪

## Aktionen im BOR



Den Zusammenhang zwischen der Definition der BAPI-Methode im BOR und dem Funktionsbaustein demonstriert das folgende [Beispiel \[Extern\]](#).



Falls Sie den BAPI Wizard nicht verwenden können (z.B. bei späteren Änderungen am BAPI oder beim Überdefinieren einer Methode), bearbeiten Sie die Methode im BOR selbst. Dazu müssen Sie insbesondere die oben aufgeführten Punkte sicherstellen.

Bei Entwicklungen von IBUs, Kunden und Partnern sind **zusätzlich** folgende Punkte zu beachten:

- Lesen Sie die Dokumentation zu [Kundenerweiterung und -modifikation von BAPIs \[Extern\]](#).
- Als Kunde können Sie eigene BAPIs an eigenen Business-Objekttypen anlegen oder Sie definieren einen [Subtyp \[Extern\]](#) zu einem bestehenden SAP-Business-Objekttyp. Von der Definition neuer BAPIs an SAP-Business-Objekttypen wird dringend abgeraten, da dies eine Modifikation darstellt.
- Alle Entwicklungsobjekte (Business-Objekttypen, BAPIs, Parameter) sind in Ihrem eigenen [Namensraum \[Extern\]](#) anzulegen.

## Dokumentation des BAPIs

### Grundsatz

Die Qualität der Dokumentation eines BAPIs wird durch folgenden Grundsatz bestimmt:

Die Dokumentation muß so ausführlich sein, daß ein externer Entwickler, der den betriebswirtschaftlichen Hintergrund beherrscht, sich aber nicht mit dem R/3-System auskennt, das BAPI benutzen kann.

### Funktionsumfang

Die Dokumentation eines BAPIs umfaßt die folgenden vier Bereiche:

Dokumentation des BAPIs

1 D  
. i  
e  
D  
o  
k  
u  
m  
e  
n  
t  
a  
t  
i  
o  
n  
d  
e  
s  
S  
A  
P  
B  
u  
s  
i  
n  
e  
s  
s  
:  
Q  
b  
i  
e  
k  
t  
t  
y  
p  
s  
I  
E  
X  
t  
e  
r  
n  
l.

2 . D  
i  
e  
D  
o  
k  
u  
m  
e  
n  
t  
a  
t  
i  
o  
n  
d  
e  
r  
M  
e  
t  
h  
o  
d  
e  
l  
E  
x  
t  
e  
r  
n  
l  
s  
o  
l  
t  
e  
f  
o  
l  
g  
e  
n  
d  
e  
F  
r  
a  
g  
e  
n  
b  
e  
a  
n

**Dokumentation des BAPIs**

- Was ist die betriebswirtschaftliche Funktionalität des BAPIs und was sind dessen Einsatzmöglichkeiten?
- Was leisten die Funktionen des BAPIs im einzelnen?
- Liegen wichtige Einschränkungen vor, d.h. gibt es Funktionen, die man mit diesem BAPI nicht durchführen kann?
- Was ist bei der Verwendung dieser Methode besonders zu beachten (z.B. Berechtigungsprüfungen)?  
Welche weiteren Voraussetzungen gelten für das BAPI ?
- Welche Customizing-Abhängigkeiten existieren?
- Welche Abhängigkeiten existieren zwischen diesem BAPI und anderen BAPIs, und zwischen den einzelnen Methodenparametern?
- Handelt es sich um ein BAPI mit Pufferung?  
Falls ja, muß das BAPI als solches identifiziert werden.
- Enthält das BAPI ein COMMIT WORK-Kommando ?  
Falls dies der Fall ist, muß es explizit dokumentiert werden.

3 . D  
i  
e  
D  
o  
k  
u  
m  
e  
n  
t  
a  
t  
i  
o  
n  
n  
z  
u  
d  
e  
n  
B  
A  
P  
I  
P  
a  
r  
a  
m  
e  
t  
e  
r  
e  
n  
[  
E  
x  
t  
e  
r  
n  
]  
b  
e  
a  
n  
t  
w  
o  
r  
t  
e  
t  
f  
o  
|

e  
n  
d  
e

**Dokumentation des BAPIs**

- Was ist die Bedeutung des Parameters?
- Welches Feld eines Parameters muß gefüllt sein, d.h. welches sind die obligatorischen Felder?
- Welche Abhängigkeiten existieren zwischen den Feldern?  
Gibt es Parameterabhängigkeiten und Feldabhängigkeiten innerhalb einer Struktur ?
- Gibt es mögliche feste Werte und was ist deren Wirkung?
- Wie ist der Parameter vorbelegt?  
Alle Felder, die eventuell über das Customizing mit Default-Werten vorbelegt und daher schreibgeschützt sind, müssen entsprechend dokumentiert werden.
- Entspricht die Dokumentation des Return-Parameters den Richtlinien und enthält sie alle relevanten (bzw. die wichtigsten) Fehlermeldungen ?
- Wird bei einem Abbruch ausnahmsweise innerhalb des BAPIs ein Datenbank-Rollback durchgeführt ?  
Falls ja, muß dies explizit in der Dokumentation zum Return-Parameter beschrieben werden
- Wurden in der Dokumentation zu den Extension-Parametern (*ExtensionIn*, *ExtensionOut*) alle verfügbaren BAPI Table Extensions aufgelistet?

4 D  
.  
o  
k  
u  
m  
e  
n  
t  
a  
t  
i  
o  
n  
d  
e  
r  
D  
a  
t  
e  
n  
e  
l  
e  
m  
e  
n  
t  
e  
I  
E  
x  
t  
e  
r  
n  
]  
d  
i  
e  
d  
e  
n  
F  
e  
l  
d  
e  
r  
n  
i  
n  
B  
A  
P  
I  
P

---

**Dokumentation des BAPIs**

Anschließend ist darauf zu achten, daß die Funktionsbaustein- und Parameterdokumentation abschließend vom Dokumentationsentwickler *In aktiver Fassung* gesichert wurde, denn nur so gelangt die Dokumentation in den Arbeitsvorrat der Übersetzung.

## BAPI/ALE-Integration

BAPIs sind in das ALE-Kommunikationsmodell integriert. Die ALE-Kommunikation basiert auf Nachrichten (Messages) und erfolgt asynchron. ALE-Kommunikation ist die bevorzugte Integration für verteilte R/3-Systeme, z.B. zur Verteilung von Stammdaten.

Seit Release 4.0 sind **BAPIs die standardisierten Schnittstellen für ALE-gestützte Kommunikation**. Für BAPIs können die ALE-Services verwendet werden, z.B. asynchroner Aufruf des BAPIs, Nutzung des Verteilungsmodells, Monitoring und die Fehlerbearbeitung. Die für die ALE-Services benötigten IDoc-Typen sind aus BAPIs generierbar.

**Siehe auch:** [Nutzung der ALE Services \[Extern\]](#).

Bei der Verwendung eines BAPIs zur asynchronen Datenübertragung wird von der Anwendung im sendenden System anstelle des BAPIs die generierte ALE-IDoc-Schnittstelle aufgerufen. Die ALE-IDoc-Schnittstelle führt dabei die folgenden Schritte aus:

- Aufbau eines IDocs aus den BAPI-Daten
- Versenden des IDocs an das Zielsystem
- Empfangen des IDocs im Zielsystem, Aufbau der BAPI-Daten aus dem IDoc und Aufruf des BAPIs

Um die für die asynchrone Kommunikation neben dem BAPI benötigten Objekte (Nachrichtentyp, IDoc-Typ, Eingangs- und Ausgangsfunktionsbaustein) automatisch zu erstellen, steht Ihnen die Transaktion *BDBG* zur Verfügung.

**Siehe auch:** [BAPI-ALE-Schnittstelle pflegen \[Extern\]](#).

Ein BAPI sollte genau dann als asynchrone Schnittstelle implementiert werden, wenn mindestens eine der folgenden Bedingungen erfüllt ist:

- *Konsistente Datenbankänderungen in mehreren Systemen*  
Daten müssen sowohl auf dem lokalen System als auch auf einem remoten System konsistent fortgeschrieben werden.
- *Lockerung der Ankopplung*  
Eine Implementierung als synchrone Schnittstelle würde eine zu enge Kopplung zwischen dem Client- und dem Server-System darstellen. Bei Ausfall der Verbindung könnte das Client-System nicht mehr sinnvoll arbeiten.
- *Performance*  
Es handelt sich um eine Schnittstelle mit großem Datenvolumen oder es sollen zeitaufwendige Datenbankoperationen auf dem Server-System durchgeführt werden. Eine synchrone Schnittstelle kommt in dieser Situation aus Performance-Gründen nicht in Frage.

Aus der BAPI/ALE-Integration ergeben sich folgende Anforderungen:

- Für alle schreibenden BAPIs sind Nachrichtentypen angelegt und IDoc-Typen generiert.  
**Siehe auch:** [BAPI-ALE-Schnittstelle generieren \[Extern\]](#)
- Alle IDoc-Typen und die zugehörigen Segmente sind freigegeben.
- Die BAPI-Schnittstelle und die IDoc-Schnittstelle sind identisch. Beachten Sie insbesondere bei Änderungen am BAPI, daß der Nachrichtentyp, der IDoc-Typ und die Segmente nachgeneriert werden.



**BAPI/ALE-Integration**

Alle Export-Parameter des BAPIs mit Ausnahme des Return-Parameters werden ignoriert und nicht in den generierten IDoc-Typ aufgenommen.

**SAP-intern:**

Dient das BAPI der Integration zwischen verschiedenen SAP-Produkten, dann müssen die Schnittstelleninformationen den entsprechenden Systemen zur Verfügung gestellt werden. Dazu müssen die vom BAPI verwendeten Datenstrukturen (inklusive Datenelementen und Domänen) sowie gegebenenfalls die ALE-Schnittstelle in die Anwendungsbasis umziehen.

Sofern es sich um Applikationen mit eingeschränkter Verwendbarkeit handelt (Entkopplung, z.B. RW/LO und HR, teilweise NewDimension zu R/3), dann muß der BAPI-Funktionsbaustein explizit zur applikationsübergreifenden Verwendung freigegeben werden.

Wenden Sie sich dazu bitte an Ihren BFA-Ansprechpartner.

## Test und Freigabe

Um die Korrektheit der BAPIs, der Business-Objekttypen und der Dokumentation zu überprüfen, müssen spezielle Tests durchlaufen und ToDo-Checks ausgeführt werden. Ist die Testphase erfolgreich abgeschlossen worden, können Business-Objekttyp, BOR-Methoden und Funktionsbausteine freigegeben werden.

Weitere Informationen finden Sie unter

[Test \[Seite 74\]](#)

[Freigabe \[Seite 77\]](#)

## Test

## Test

Nachdem Sie den Funktionsbaustein, der Ihrem BAPI zugrundeliegt, implementiert und das BAPI als Methode eines SAP-Business-Objektyps im Business Object Repository (BOR) definiert haben, sollten Sie das BAPI testen.



Nehmen Sie die Tests zu Ihrem BAPI in enger Zusammenarbeit mit der Qualitätssicherung in Ihrer Entwicklungsabteilung vor.

## Testphase

Während der Testphase sind die folgenden Arbeitsschritte durchzuführen:

**Testen der Dokumentation**

Testen Sie die Verfügbarkeit und Verständlichkeit der Dokumentation, die zum Business-Objektyp, zum BAPI und zu jedem Schnittstellenparameter angelegt wurde. Da die Dokumentation von BAPIs entscheidend für die Verwendbarkeit der BAPIs ist, sollte dieser Test mit großer Sorgfalt durchgeführt werden.

**Test der syntaktischen Korrektheit eines BAPIs**

Im Rahmen dieses Testabschnitts sollte sichergestellt werden, daß das BAPI alle zuvor beschriebenen Konventionen erfüllt und daß alle Richtlinien für die BAPI-Entwicklung eingehalten wurden. Um die syntaktische Korrektheit zu gewährleisten, stehen Ihnen Prüfmöglichkeiten im BAPI Explorer und die BAPI ToDo zur Verfügung. Eine implizite Prüfung wird außerdem beim Einhängen des BAPIs über den BAPI/BOR Wizard und bei der Freigabe des BAPIs im BOR vorgenommen.



Um die syntaktische Korrektheit eines BAPIs im BAPI Explorer zu testen, positionieren Sie zunächst den Cursor im linken Teilfenster auf das zu testende BAPI. Wählen Sie anschließend im rechten Teilfenster die Registerkarte *Werkzeuge* aus und selektieren die Option *Konsistenzprüfungen*.

**Test der semantischen Korrektheit eines BAPIs**

Nachdem die syntaktische Korrektheit eines BAPIs sichergestellt ist, muß seine semantische Korrektheit überprüft, also der Funktions- und Integrationstest durchgeführt werden. Dazu haben Sie folgende Möglichkeiten:

**Testen des zugrundeliegenden Funktionsbausteins im Function Builder.**

Im Function Builder können Sie in einem Einzeltest die Parameter Ihres Funktionsbausteins mit entsprechenden Werten versorgen, um sicherzustellen, daß der Programmcode des Funktionsbausteins fehlerfrei abläuft.

Der Test im Function Builder besitzt allerdings die folgenden Nachteile:

- Die Tests führen zu keiner Manipulation der Datenbank, da kein COMMIT-Work ausgeführt werden kann.
- Die Tests sind nicht automatisierbar.
- Die Testdaten werden nicht in andere Systeme transportiert und gehen bei Änderungen am Funktionsbaustein verloren. Daher sind beispielsweise keine Regressionstests möglich.

**Testen des Funktionsbausteins mit dem Computer Aided Testing Tool (CATT)**

Mit Hilfe eines CATT-Testbausteins vom Typ **F** können Funktionsbausteine innerhalb des SAP Standard-Testwerkzeuges getestet werden. Die Verwendung des CATT hat im Vergleich zum Testen im Function Builder die folgenden Vorteile:

- Die Tests führen zu einer Manipulation der Datenbank, da automatisch ein Commit-Work ausgeführt wird.
- Die Tests können automatisiert werden.
- Die Testdaten werden zusammen mit dem Testbaustein transportiert und stehen somit in anderen Systemen und Releases für Regressionstests zur Verfügung.



Tests mit CATT sollten denen im Function Builder vorgezogen werden, da die Testabläufe erhalten bleiben und so optimal in den Qualitätssicherungsprozeß der SAP integriert werden können.

Weitere Informationen zur Verwendung des CATT für den Test von Funktionsbausteinen finden Sie unter [CATT: Verwendung von Funktionsbausteintests \[Extern\]](#).

 **Test der Aufrufbarkeit eines BAPIs in einer externen Anwendung**

Damit ein BAPI in einer externen Anwendung verwendbar ist, müssen folgende Bedingungen erfüllt sein:

- Das BAPI ist syntaktisch korrekt.  
Dies sollte durch den BAPI-Entwickler mit Hilfe des BAPI Explorer getestet werden.
- Die Verbindungstechnik funktioniert fehlerfrei.  
Dies sollte durch entsprechende Tests der Middleware-Abteilungen sichergestellt werden.

**Test** **Test der BAPIs auf unterschiedlichen Plattformen**

Die Plattformtests werden bei der Endmontage durchgeführt. Sie sind allerdings nur möglich, wenn die Tests zur semantischen Korrektheit eines BAPIs mit CATT durchgeführt wurden und die in diesem Zusammenhang angelegten CATT-Abläufe in der Endmontage wiederverwendet werden können.



Sollten Sie beim Testen Fehler in der Implementierung Ihres BAPIs finden, so korrigieren Sie diese und wiederholen Sie die Tests, bis Sie und die Qualitätssicherung Ihrer Entwicklungsabteilung mit der Implementierung und Dokumentation des BAPIs vollständig zufrieden sind.

## Freigabe

Nach erfolgreichem Abschluß aller Tests sind die BAPIs und alle zugehörigen Entwicklungsobjekte freizugeben, um sie für den Kunden verfügbar zu machen. Dies kann aber erst erfolgen, wenn die folgenden Bedingungen erfüllt sind:

- Die gesamte benötigte Dokumentation wurde erstellt.
- Es wurden keine Konsistenzfehler festgestellt.
- Es wurde eine Freigabeempfehlung durch den BFA AG-Ansprechpartner ausgesprochen.
- Es wurde eine Freigabeempfehlung durch den Quality Manager ausgesprochen.



Zur Bedeutung und zu den Konsequenzen der Freigabe eines BAPIs siehe auch [Weiterführende Aspekte \[Seite 36\]](#).

## Ablauf

Es sind die folgenden Arbeitsschritte auszuführen:

- [Freigabe des BAPI-Funktionsbausteins \[Extern\]](#) im Function Builder.
- [Freigabe des Business-Objektyps \[Extern\]](#) im BOR.
- [Freigabe des BAPIs \[Extern\]](#) als Methode im BOR.
- Bei potentiell schreibenden BAPIs erfolgt die Freigabe des IDocs und seiner Segmente.