

SAP Automation RFC and BAPI Interfaces (BC-FES-AIT)



HELP.BCFESDE5

Release 4.6C



Copyright

© Copyright 2001 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft®, WINDOWS®, NT®, EXCEL®, Word®, PowerPoint® and SQL Server® are registered trademarks of Microsoft Corporation.

IBM®, DB2®, OS/2®, DB2/6000®, Parallel Sysplex®, MVS/ESA®, RS/6000®, AIX®, S/390®, AS/400®, OS/390®, and OS/400® are registered trademarks of IBM Corporation.

ORACLE® is a registered trademark of ORACLE Corporation.

INFORMIX®-OnLine for SAP and Informix® Dynamic Server™ are registered trademarks of Informix Software Incorporated.

UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group.

HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA® is a registered trademark of Sun Microsystems, Inc.

JAVASCRIPT® is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Tip

Contents

SAP Automation RFC and BAPI Interfaces (BC-FES-AIT).....	15
The SAP Assistant Product.....	16
System Requirements.....	17
What's New in Release 4.6A?.....	18
What's New in Release 4.6B?.....	19
What's New in Release 4.6C?.....	20
SAP Assistant Architecture.....	21
SAP Assistant Internal Components	22
Relationship Between SAP Assistant Components	24
The SAP Assistant Screen	25
Working with RFC Functions	27
Working with Business Objects and BAPIs	28
Using the SAP Assistant	29
Defining Destination Systems	30
Logging Onto an R/3 System	33
Switching to a Different System	34
Using Multiple SAP Assistant Windows	35
Browser Views	36
Changing Views	37
Specifying a Filter for RFCs and Function Groups	38
Changing Filter.....	39
Searching for Objects or Functions.....	40
Browsing Business Objects and RFCs	41
Displaying Function Parameters and their Details.....	42
Displaying Key Information for Business Objects	43
Displaying BAPI (Method) Parameters	44
Viewing R/3 Table and Structures	45
Working Offline: Using a Local Repository	46
Saving Data to a Local Repository	47
Reading from a Local Repository (Offline).....	49
Deleting a System From a Local Repository	50
Exporting Functions and Business Objects to Excel	51
Calling an RFC Function	52
Code Generation for Business Objects and RFC Functions	53
System Requirements for Code Generation.....	54
Generating Code.....	55
Navigating the BAPI Selector Screen.....	58
The C++ BAPI Proxy Classes.....	59
Introduction	60
The Business Object Proxy Classes	62
The Parameter Container Classes	64
The Reference Structure Type Classes	65
ABAP to Native Data Type Mapping	67
The CBoBase Class	68

GetName	69
GetType	70
GetKey	71
The CBoKey Class	72
GetKeyField	73
SetKeyField	74
The CBoTable Template Class	75
AppendToTable	77
ClearTable	78
CopyLineFrom	79
CopyLineTo	80
GetConstRow	81
GetLength	82
GetRow	83
GetRowCount	84
InsertIntoTable	85
RemoveRow	86
The Java BAPI Proxy Classes	87
Introduction	88
Using Java RFC packages	90
The Business Object Proxy Classes	92
The Parameter Container Classes	94
The ABAP Reference Structure Types	95
The Structure Parameter Classes	96
The Table Parameter Classes	97
appendRow	98
createEmptyRow	99
deleteAllRows	100
deleteRow	101
getRow	102
getRowCount	103
insertRow	104
updateRow	105
The Table Row Classes	106
ABAP to Native Data Type Mapping	107
The jboBase Class	108
getKey	109
setConnection	110
setKey	111
The jboKey Class	112
getKeyfield	113
getKeyfieldParameter	114
SetKeyfield (with keyfield name and value)	115
SetKeyfield (with ISimple interface)	116
BAPI Beans	117

Introduction.....	118
The BAPI Beans Information Classes	120
The jboBAPIDescriptor Class.....	121
The jboParameterContainerDescriptor Class	122
The jboConstructorDescriptor Class	123
The jboParameterDescriptor Class	124
The jboSimpleParameterDescriptor Class	125
The jboStructureParameterDescriptor Class	126
The jboTableParameterDescriptor Class	127
The jboKeyfieldDescriptor Class	128
Tracing Errors When Running the SAP Assistant.....	129
SAP Automation ActiveX (OCX) Controls.....	130
Summary of ActiveX Controls.....	131
Possible Uses for OCX Controls	134
Using the SAP Automation ActiveX Controls	135
Summary of Programming Tasks	136
Example Application with the Function Control	137
Variation using the Dynamic Calling Convention.....	139
Creating the Base-level Control	140
Connecting to an R/3 System	141
Performance and Debugging Tips	142
SAP Control Base Classes	144
SAP Standard Collection	145
SAP Named Collection	147
Using Collection Objects.....	148
SAP Data Object.....	149
Safe Arrays and Values	151
Font Objects.....	152
Data Types.....	153
SAP Automation ActiveX Controls.....	154
The Function Control.....	155
Introduction	156
Function Control Object Hierarchy	157
Function Control	158
Using the Function Control	159
Requesting Functions	160
Adding a Function.....	161
Setting Parameter Values.....	162
Viewing Table Objects	163
Using Parameter and Structure Objects.....	164
Using Named Argument Calling Conventions	165
Functions Collection Object	167
Functions Collection Properties	168
Functions Collection Methods	169
Connection Object	170
Connecting through a Logon Control.....	171

Setting the Connection Implicitly	172
Function Object.....	173
Dynamic Function Call.....	174
Function Properties.....	175
Function Property: Tables.....	176
Function Methods	177
Exports Collection Object.....	178
Exports Collection Properties	179
Exports Collection Methods	180
Imports Collection Object.....	181
Imports Collection Properties	182
Imports Collection Methods	183
Structure Object	184
Structure Properties.....	185
Structure Property: ColumnSAPType.....	186
Structure Methods	187
Parameter Object.....	188
Parameter Properties.....	189
Parameter Methods	190
The Transaction Control.....	191
Transaction Control Object Hierarchy.....	192
Using the Transaction Control	193
What's New in Release 4.6A?	196
Transactions Collection Object	197
Transactions Collection Properties	198
Transactions Collection Methods	199
Transaction Object.....	200
Transaction Object Properties	201
Transaction Object Methods.....	203
Screens Collection Object.....	204
Screens Collection Properties	205
Screens Collection Methods	206
Screen Object	208
Screen Object Properties.....	209
Screen Object Methods	210
Fields Collection Object	211
Fields Collection Properties	212
Fields Collection Methods.....	213
Field Object	214
Field Object Properties	215
The Table Tree Control.....	216
Introduction	217
Table Tree Object Hierarchy	218
Basic Concept.....	219
Table Tree Object	220

Table Tree Properties	221
Table Tree Property: Events	224
Table Tree Property: DragDrop	226
Table Tree Methods.....	227
Table Tree Events	228
Table Tree Event: DuplicatedKey.....	231
Table Tree Event: NodeInsert	232
Table Tree Event: NodeRemove.....	233
Table Tree Event: DragSourceFill	234
Table Tree Event: DropEnter	235
Table Tree Event: Drop	236
Nodes Collection Object	237
Nodes Collection Properties	238
Nodes Collection Property: Item.....	239
Nodes Collection Methods.....	240
Nodes Collection Method: Remove.....	241
Nodes Collection Method: Add.....	242
Nodes Collection Method: AddEx.....	244
Node Object	245
Node Object Properties	246
Node Object Property: Key.....	248
Node Object Property: Type.....	249
Node Object Property: ForceExpander	250
Node Object Property: AllChildren	251
Node Object Methods	253
Node Object Method: SaveData, LoadData	254
Structures Collection Object	255
Structures Collection Properties	256
Structures Collection Properties: Item.....	257
Structures Collection Methods.....	258
Structures Collection Method: Add.....	259
Structures Collection Method: Remove.....	260
Structures Collection Method: Insert	261
Structure Object.....	262
Structure Object Properties	263
Structure Object Property: Type	264
Structure Object Property: Alignment.....	266
Structure Object Property: Hidden	267
Configuring the Tree	268
Connecting Tree Views and Table Objects	272
Drag and Drop with Tree Views.....	274
Design Environment Property Pages.....	275
Table Tree Property Page: General	276
Table Tree Property Page: Structure.....	278
Table Tree Property Page: Events	280

Table Tree Property Page: Fonts	281
Table Tree Property Page: Colors	282
Appearance for Different Configurations	283
Code Examples.....	285
Glossary.....	286
Item	287
Node	288
Root Control/Root	289
Root Node.....	290
Nothing.....	291
CreateObject.....	292
Dynamic Node Properties.....	293
Hierarchy Expander/Expander Symbol	294
Folder.....	295
Leaf.....	296
Level	297
Collection	298
Pre-Defined Images	299
The Table Factory Control	300
Introduction	302
Table Factory Object Hierarchy	303
Using the Table Factory Control	304
Table Factory Object.....	305
Table Factory Properties	306
Table Factory Methods	307
Table Factory Events.....	308
Tables Collection Object.....	309
Tables Collection Properties.....	310
Tables Collection Methods	311
Tables Collection Method: Remove	312
Tables Collection Method: Unload	313
Tables Collection Method: Item	314
Tables Collection Method: Add	315
Table Object.....	316
Table Object Properties	317
Table Property: Data	318
Table Object Methods.....	319
Table Method: CreateFromR3Repository	322
Table Method: CreateFromHandle.....	323
Table Method: CreateFromTable	324
Table Method: Create.....	325
Table Method: AttachHandle	326
Table Method: DetachHandle.....	327
Table Method: Refresh.....	328
Table Method: SelectToMatrix	329

Table Method: SelectMatrixToNumber.....	330
Table Method: SelectToVector.....	331
Table Method: SelectVectorToNumber.....	332
Table Method: BuildTiledRanges	333
Creating a Table Object.....	334
Using SelectTo* Methods	335
Displaying and Navigating Table Data	337
RFCTableParameter Object	338
RfcTableParameter Object Properties.....	339
Rows Collection Object.....	340
Rows Collection Properties.....	341
Rows Collection Property: Item	342
Rows Collection Methods	343
Rows Collection Method: Remove	344
Rows Collection Method: Add	345
Rows Collection Method: Insert.....	346
Row Object	347
Row Object Properties.....	348
Row Object Methods	349
Columns Collection Object.....	350
Columns Collection Properties	351
Defining a Key Column for a Table	352
Columns Collection Methods	353
Columns Collection Method: Remove	354
Columns Collection Method: Item	355
Columns Collection Method: Add	356
Columns Collection Method: Insert	357
Column Object	358
Column Object Properties	359
Column Object Property: Type	360
Column Object Methods	361
Ranges Collection Object	362
Ranges Collection Properties	363
Ranges Collection Methods.....	364
Ranges Collection Method: Add.....	365
Range Object	366
Range Object Properties	367
Boundaries of a Range Object	368
Range Object Methods	369
Views Collection Object	370
Views Collection Properties	371
Views Collection Methods.....	372
How to Connect Views to a Table	373
Matrix Object.....	374
Code Examples.....	375

First Steps.....	376
Accessing Table Data.....	377
Automatic Display	380
Using Dynamic Structures for a Table.....	381
Glossary.....	383
Table.....	384
Row.....	385
Column.....	386
Matrix	387
Range	388
The Table View Control.....	389
Introduction	390
Table View Control Object Hierarchy	391
Basic Concept.....	392
Using the Table View Control	393
Table View Object.....	396
Table View Properties.....	397
Table View Property: Selection	401
Table View Property: EnableProtection.....	402
Table View Property: Value	403
Table View Property: Cell	404
Table View Methods	405
Table View Events	406
Table View Event: DragSourceFill.....	410
Table View Event: DropEnter	411
Table View Event: Drop.....	412
Connecting Table Views and Table Objects.....	413
Drag and Drop with Table Views	416
Columns Collection Object.....	417
Columns Collection Properties	418
Columns Collection Methods	419
Columns Collection Methods: Add	420
Columns Collection Methods: Remove	421
Columns Collection Methods: Insert.....	422
Column Object	423
Column Object Properties	424
Column Object Properties: Type	426
Column Object Properties: Alignment	427
Column Object Methods	428
Rows Collection Object.....	429
Rows Collection Properties.....	430
Rows Collection Methods	431
Rows Collection Methods: Add	432
Rows Collection Methods: Remove	433
Rows Collection Methods: Insert.....	434

Row Object	435
Row Object Properties.....	436
Row Object Methods	437
Cell Object	438
Cell Object Properties.....	439
Cell Object Methods	440
Design Environment Property Pages.....	441
Table View Property Page: General.....	442
Table View Property Page: Flags	443
Table View Property Page: Events.....	444
Table View Property Page: Columns.....	445
Table View Property Page: Rows.....	446
Table View Property Page: Fonts.....	447
Table View Property Page: Colors	448
Glossary.....	449
Collection	450
Root Control, Root.....	451
Root Node.....	452
Nothing.....	453
CreateObject.....	454
Active Cell.....	455
Range	456
Named Object.....	457
The Logon Control	458
Introduction	459
Logon Control Object Hierarchy	460
Using the SAP Logon Control.....	461
Connecting to the R/3 System.....	462
Using the Logon Control to Connect to R/3	463
Logon Object.....	465
Logon Object Properties	466
Logon Property: Events	467
Logon Property: Caption	468
Logon Property: BackColor	469
Logon Property: hWnd.....	470
Logon Property: Enabled.....	471
Logon Property: Font.....	472
Logon Property: Parent	473
Logon Property: Default	474
Logon Property: ApplicationName.....	475
Logon Property: System.....	476
Logon Property: ApplicationServer.....	477
Logon Property: SystemNumber	478
Logon Property: MessageServer.....	479
Logon Property: GroupName	480

Logon Property: TraceLevel	481
Logon Property: RFCWithDialog	482
Logon Property: Client	483
Logon Property: User	484
Logon Property: Language	485
Logon Object Methods	486
Logon Method: Enable3D	487
Logon Method: NewConnection	488
Logon Object Events	489
Connection Object	490
Connection Object Properties	491
Connection Property: IsConnected	492
Connection Object Methods	493
Connection Method: Logon	494
Connection Method: Logoff	495
Using Logon Controls in Design Mode	496
Code Examples	498
Connecting Directly with the Logon Control	499
Logging on Silently	500
Glossary	501
Connection	502
Password	503
Nothing	504
CreateObject	505
DCOM Connector-compatible Components	506
The SAPBrowser Control	507
Properties	508
Methods	509
AddBAPIAppObject	510
AddBAPIOBJECT	511
AddBAPISearchObject	512
AddRFCFunctionGroups	513
AddRFCFunctions	514
AddSearchRFCFunctionGroups	515
AddSearchRFCFunctions	516
CallFunction	517
ClearAll	518
ClearBAPITab	519
ClearRFCTab	520
ClearSearchTab	521
DeleteObject	522
EnableBAPITab	523
EnableRFCTab	524
EnableSearchTab	525
GetSelectedObject	526

HidePropertyWindow	527
IsApplicationArea	528
IsBAPI	529
IsBusinessObject	530
IsFunction	531
IsMethod	532
IsRFC	533
IsSearch	534
Refresh	535
ShowPropertyWindow	536
StartPrint	537
Undo	538
Example	539

SAP Automation RFC and BAPI Interfaces (BC-FES-AIT)

One of the methods for integrating an external PC application to R/3 is by using the RFC interfaces of R/3, also called the RFC channel.

Using the RFC interfaces external applications access R/3 by making remote calls to R/3 functions.

The [SAP Automation suite \[Ext.\]](#) offers several products that are based on the RFC interfaces. These products use the following technologies:

- Making Remote Function Calls (RFCs) directly to invoke SAP function modules
- Using Business APIs (BAPIs) to access and work with SAP business objects
- Using transactions for batch input
- Exchanging asynchronous messages with R/3 using IDoc interface technology

See the discussion in of these technologies in the [SAP Automation Help \[Ext.\]](#).

The SAP Automation suite includes several ActiveX controls (previously called OCX controls) for making remote calls to R/3 functions.

SAP Automation also includes a stand alone product, [SAP Assistant \[Page 16\]](#), which you can use for viewing RFCs and BAPIs, and for calling RFC function modules.

The SAP Assistant product uses several of the ActiveX controls/components. These components are also available separately for use directly in PC applications.

If you are looking to use any of those products by itself, read its documentation here, under the [SAP Automation ActiveX \(OCX\) Controls \[Page 130\]](#) topic.

The SAP Assistant Product

The SAP Assistant Product

The **SAP Assistant** is a stand-alone executable program providing an online tool for users wishing to access R/3 business objects and Remote Function Calls (RFC) metadata information from outside of R/3.

Using the SAP Assistant you can perform a number of tasks related to SAP business objects and RFCs:

- Browse (view and search) metadata information of SAP RFC function modules and of business object and their BAPIs (Business APIs, which are the methods of the business objects)
- Call RFCs and use their functionality directly online
- Use a BAPI Wizard for generating either Java or C++ classes, which you can then use for creating and manipulating SAP business objects using their BAPIs.

Note that the browsing functionality of the SAP assistant shows RFC and BAPI metadata only, meaning that it give information about the various parameters of an RFC or a BAPI, but it does not show table data from R/3. For example, browsing allows you to view the *GeneralLedgerAccount* business object, its fields and methods. It does not show actual General Ledger Account record information.

Only by calling RFC function modules or BAPIs can you access R/3 table and field data.

System Requirements

To use the SAP Assistant you need the following:

- Windows NT 4.0, Windows 95, or Windows 98 operating system
- The SAP DCOM Connector installed
- SAP R/3 Release 3.0D or higher

Restrictions

- You cannot use the SAP Assistant to call an RFC that invokes a SAPGUI screen, be it a dialog or a window. You can determine if SAP Assistant fails as a result of calling such an RFC, by using Transaction SE37 in R/3 and checking if the RFC invokes a SAPGUI screen.
- The BAPI wizard can generate C++ code for Windows NT or HP UNIX. To generate code for other platforms you need to create your own templates.

What's New in Release 4.6A?

What's New in Release 4.6A?

- [Using a local repository database \[Page 46\]](#) is simpler.
- Local repository file type is now *.sdb*.
- The R/3 Logon dialog (invoked when you log online) allows you to [define new destination systems \[Page 30\]](#), which you can then use to log on. It also allows you to change or delete existing destination definitions.
- [Table Browser feature \[Page 45\]](#) allows you to view the metadata of an R/3 table or structure.
- Context menu (menu you invoke with the right-mouse-button) is available at all three tabs of the *BAPI/RFC detail pane*. The context menu provides a shortcut to the most commonly used actions on the object you are viewing.
- The [filter that is used \[Page 38\]](#) when you are in *RFC functions* or *function groups* view is displayed at the top of the hierarchy at the RFC tab.
- You can mark items for [code generation \[Page 53\]](#) from the [search tab \[Page 40\]](#) too.
- You can [save data into a local repository \[Page 47\]](#) from any tab in the *BAPI/RFC detail pane*, including the [search tab \[Page 40\]](#).
- You can [run an RFC \[Page 52\]](#) from any of the tabs in the *BAPI/RFC detail pane*, including the search tab.

(It is possible to run the underlying RFC for a BAPI directly from the BAPI tab at the *BAPI/RFC detail pane*.)
- String specification for [searching business objects \[Page 40\]](#) is not case sensitive and it allows using wild cards.
- When [running an RFC function \[Page 52\]](#), specifying import parameters and looking at the resulting export parameters is done at the same dialog.
- You can [run an RFC \[Page 52\]](#) from a [local database \(repository\) \[Page 46\]](#). SAP Assistant will log on to the R/3 system to actually run the function.
- A [Trace option \[Page 129\]](#) allows you to save log of errors occurring when running SAP Assistant.
- The BAPI Wizard can now [generate code \[Page 53\]](#) for HP UNIX.

What's New in Release 4.6B?

- The SAP Assistant's menus have been reorganized to make the product more intuitive.
- The BAPI Wizard has been redesigned to make it easier to use.

What's New in Release 4.6C?

What's New in Release 4.6C?

A new menu item allows you to create a blank XML document template for use with IDocs. The menu item is: *File* → *Blank XML doc. for IDoc Type*. However, this feature is useful mainly in the context of using XML documents instead of IDocs, which is the purpose of the [IDoc Connector for XML product \[Ext.\]](#). See the [documentation of this feature in the IDoc Connector for XML product \[Ext.\]](#).

New Code Generator Features:

- The Code Generator can now generate code for RFC functions (in addition to business objects)
- The Code generator now allows you to generate code for COM-compatible languages, such as Visual Basic, using the code generator that comes with the SAP DCOM Connector. You use the SAP Assistant code generator feature to invoke this code generator.
- The user interface for generating code changed again to simplify its use and to accommodate the above features.

For details, see the topics under [Code Generation for Business Objects and RFC Functions \[Page 53\]](#).

SAP Assistant Architecture

SAP Assistant Internal Components

SAP Assistant Internal Components

Internally, the SAP Assistant is based on several other tools that are part of the [SAP Automation suite \[Ext.\]](#).

The following table describes the products that are included as part of the SAP Assistant package. Note that these products are also available as tools you can install and use separately.

Tool	Type	Function	Language(s)
DCOM Connector Logon Component [Ext.]	COM Server	<p>Helps programs using the SAP DCOM Connector in handling the connection parameters of COM objects created for the DCOM Connector.</p> <p>The DCOM Connector Logon Component provides a Logon dialog with which you can get the necessary connection parameters from an end user. It also allows the end user to define destination systems.</p> <p>The DCOM Connector Logon Component also allows you to easily copy connection parameters into a DCOM Connector COM object.</p>	Any COM/DCOM-compliant application
Repository Services [Ext.]	COM Server	<p>Provides read access to the metadata of business objects and RFC function modules in an R/3 system to COM-compliant programs and applications.</p> <p>Also allows you to save a copy of the metadata in a local database. Using a local database provides local caching mechanism to speed up access to the metadata. It also enables offline access to the metadata.</p>	Any DCOM/COM-compliant application
Repository Browser (Also called the SAP Browser Control [Page 507])	ActiveX Control (OCX)	<p>A control that can be hosted by any ActiveX container. It consists of a window with two panes for browsing SAP BAPI and RFC metadata information.</p> <p>Allows online calling of RFC functions from within the control.</p> <p>Also exposes several methods to enable the container application to control and automate metadata browsing.</p> <p>Allows you to export properly formatted metadata information to MS Excel.</p>	Any DCOM/COM-compliant application

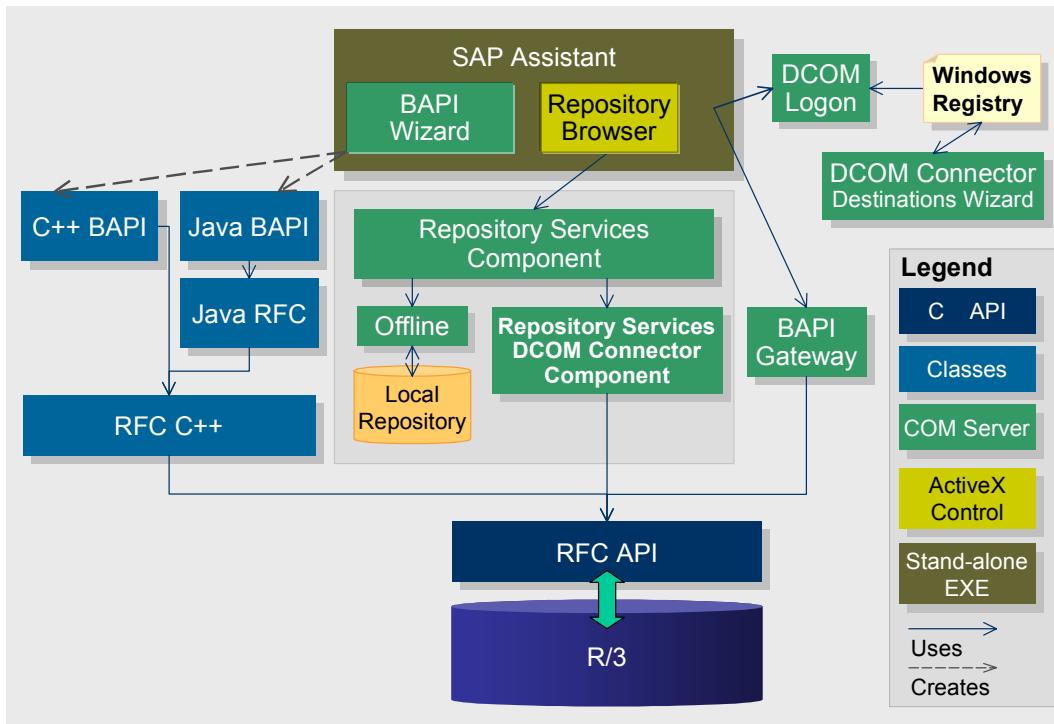
SAP Assistant Internal Components

<u>BAPI Gateway [Ext.]</u>	COM Server	<p>Allows you to dynamically call BAPIs and RFCS through the DCOM Connector, that is, it allows you to determine at run time which BAPIs or RFCS you call.</p> <p>You use the BAPI gateway in conjunction with the SAP Automation <u>Repository Services component [Ext.]</u> to obtains the metadata for the BAPIs or RFCS you wish to call at run time.</p>	Any COM/DCOM-compliant application
--	------------	---	------------------------------------

Relationship Between SAP Assistant Components

Relationship Between SAP Assistant Components

The following diagram shows the SAP Assistant, the tools that it is based on, and the class libraries it is capable of producing. The diagram illustrates the relationship between these products.

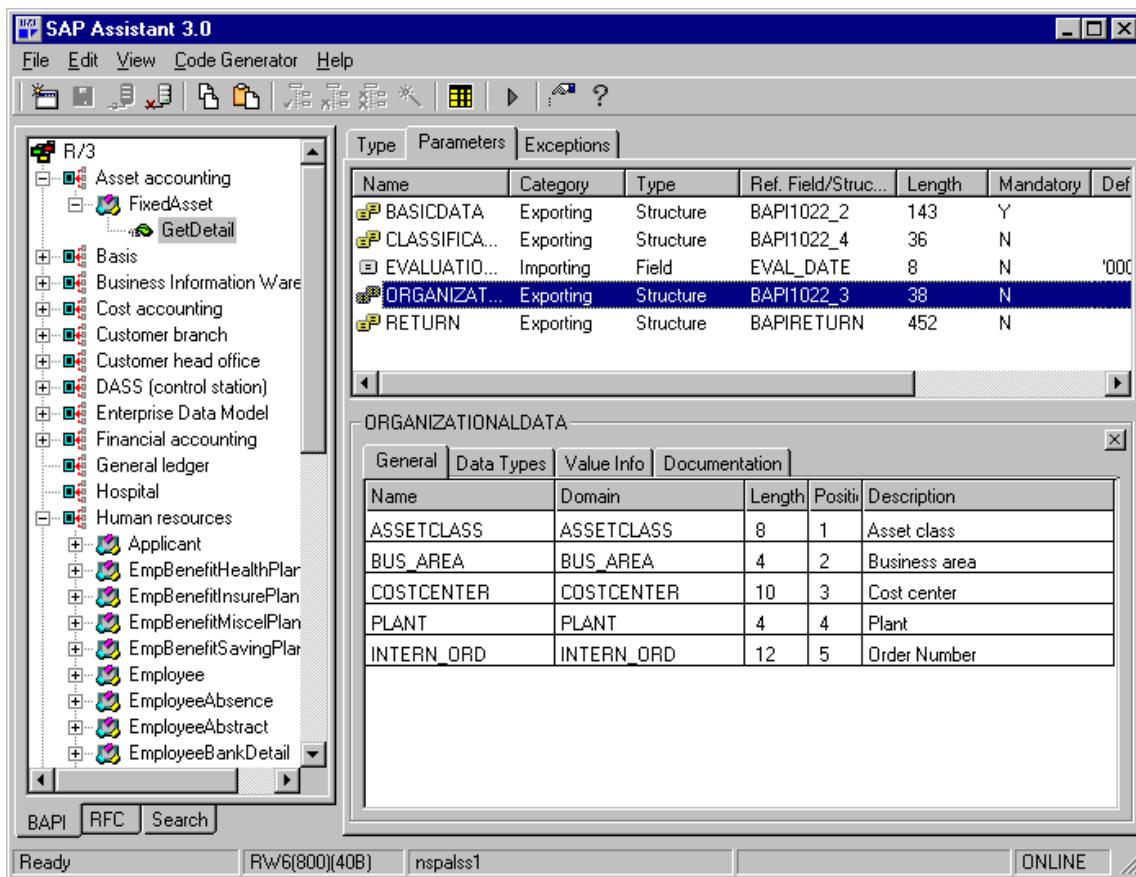


The SAP Assistant Screen

The SAP Assistant Screen

The SAP Assistant screen area consists of the Browser window, which uses the [SAP Browser control \[Page 507\]](#) to display the RFC and BAPI information.

The following is the screen of the SAP Assistant:



SAP Assistant Panes

As seen in the above screen, the SAP Assistant window is composed of three frames or panes. (which are the panes of the Browser within it):

- The left-hand side pane of the Browser window allows you to choose between BAPI, RFC, or Search views, by choosing one of three tabs. This pane displays a list of BAPIs or RFCs that exist on the system to which you are connected. The SAP Assistant sets this pane to "Function Group" for 3.0X systems since there are no Application Hierarchy or Business Objects defined for SAP R/3 3.0X releases. The Search tab allows you to search for a specific program, function, or function group. We refer to this pane as the *BAPI/RFC tree pane*.

The SAP Assistant Screen

- The right-hand side pane changes based on your selection in the BAPI/RFC tree pane: it shows the details of the BAPI or RFC you have selected in the tree pane. We refer to this pane as the *BAPI/RFC detail pane*.
- Once you display a BAPI or an RFC and see its parameters in the BAPI/RFC detail pane you can double-click on any of the parameters or other details items to view the properties of this parameter. This opens a third frame at the bottom of the BAPI/RFC detail pane. This third frame is called the *Properties pane*. Because the information is unavailable, the SAP Assistant shows nothing in the field description and field dictionary type parameter properties for users running 3.0X Releases of the SAP R/3 System.

There are four tabs in the Properties pane:

Tab	Description
<i>General Info</i>	Shows basic information, such as the name, internal name, decimal position, and description of the field of the parameter displayed in the heading row.
<i>Data Types</i>	Shows the ABAP Type and the Dictionary Type of the parameter displayed in the heading row.
<i>Value Info</i>	Shows the check table of the parameter displayed in the heading row. If you are connected in online mode (you are logged onto an R/3 system), you can display the values in the check table by clicking on the check table name in the Value Info pane. The Check Table field in the Value Info tab then becomes a drop-down list containing the check table values. Also shows whether the parameter has fixed value.
<i>Documentation</i>	Provides a long text description of the selected item. A search function is also available in this tab.

SAP Assistant Status Line

The status line in the SAP Assistant screen indicates the following:

- The type of connection you have: whether it is online (logged on to an R/3 system) or [offline \[Page 46\]](#) (you are reading data from a local repository, which is a local database file)
- The system you are connected to (regardless of whether you are connected in online or offline mode)
- The application server of the system you are connected to
- The local repository database filename, if you are using a local repository

Working with RFC Functions

The SAP Assistant lets you work with R/3 function modules (RFC functions) or R/3 business objects (BAPIs). When working with RFC functions, the SAP Assistant lets you:

- display a list of remote functions or function groups in the R/3 System.
- [display information on a function \[Page 42\]](#) (function or parameter description)
- [call the function \[Page 52\]](#), and see the results
- [export function information to Microsoft Excel \[Page 51\]](#)

To work with RFC functions or function groups, [switch to one of the RFC views \[Page 37\]](#).

You can also [search for RFC functions and function group at the search tab \[Page 40\]](#) and then work with them from this tab.

Working with Business Objects and BAPIs

Working with Business Objects and BAPIs

The SAP Assistant lets you work with R/3 function modules (RFC functions) or R/3 business objects (BAPIs). When working with business objects and BAPIs the SAP Assistant allows you to:

- display a list of business objects in the R/3 System
- [display the key fields of a business object \[Page 43\]](#)
- [display the details of the methods \(BAPIs\) for a business object \[Page 44\]](#)
- [call the underlying RFC function \[Page 52\] of a BAPI, and see the results](#)
- [export business object and BAPIs metadata information to Microsoft Excel \[Page 51\]](#)
- [generate code for programming BAPIs in Java or C++ \[Page 53\]](#)

To work with business objects and their BAPIs, [you must be in one of the BAPI views \[Page 37\]](#).

You can also [search for business objects at the search tab \[Page 40\]](#) and then work with them from this tab.

Using the SAP Assistant

Procedure

1. Start the SAP Assistant program.
2. Connect to the R/3 System by either choosing the Logon button or choosing *File → Logon R/3 System*. The *R/3 Logon* dialog appears.
3. If no destination system name is listed at the *Destination* field of the *R/3 Logon* dialog, [set up a destination definition \[Page 30\]](#) for all the R/3 Systems you wish to log onto. You must define at least one destination to log onto.
4. [Log onto the R/3 System \[Page 33\]](#) you wish to use.
5. The SAP Assistant displays metadata of business objects in an application hierarchy view.

You can [change the view \[Page 37\]](#) to show any of the other three views (RFC functions, for example).

If you change to one of the RFC views, we recommend that you [set up a filter \[Page 38\]](#). Using no filter when browsing RFCs results in thousands of RFCs, which can take a long time to appear in the browser window.

6. [Browse business objects or RFC functions \[Page 41\]](#) in the BAPI or RFC tab.
7. If you wish to save all or some of the data into a [local repository \[Page 46\]](#), that is, if you wish to be able to access this same data next time from a database file on your hard disk, mark and save the data.
8. Exit the SAP Assistant program by choosing *File → Exit*.

From either online mode or when using a local repository file, you can perform the following tasks:

- [Browse business objects or RFC functions \[Page 41\]](#).
- [Use a filter to narrow down the list of viewed RFC functions or function groups \[Page 38\]](#).
- [Search for a particular business object or RFC \[Page 40\]](#).
- [Call an RFC function module \[Page 52\]](#). Since BAPIs are implemented internally as RFC function modules, you can perform a task similar to calling a BAPI by calling its underlying function module.
- [Generate Java or C++ code for using BAPIs \[Page 53\]](#).

Defining Destination Systems

Defining Destination Systems

Use

When working with the SAP Assistant you log onto an R/3 System to get its BAPI or RFC metadata.

Before you can log onto any R/3 System through the SAP Assistant, you must set up a destination definition for that system.

Destination definitions are stored in the Windows Registry.

You may set up destination definitions for some or for all of the systems you are going to access by using the SAP DCOM Connector. You can do so before invoking the SAP Assistant, or you can do so while using the SAP Assistant. However, you should define a destination system you wish to use before logging on.

For the details of how to use the SAP DCOM Connector to define destination system, see the DCOM Connector documentation.

You can also define destinations in SAP Assistant.

Procedure

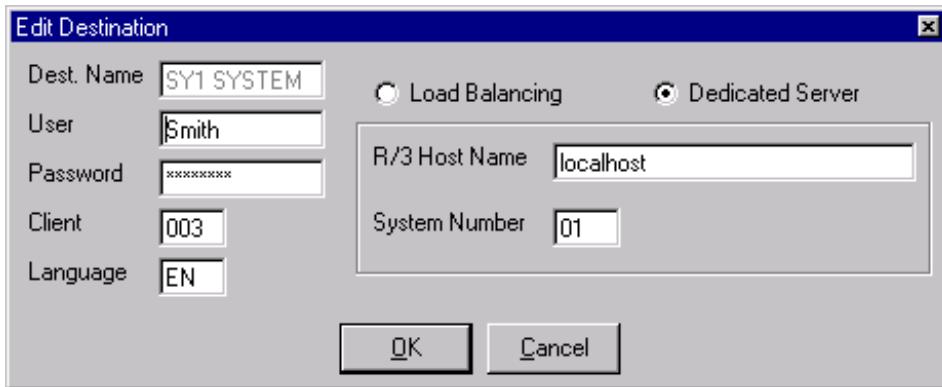
1. Logon online by choosing *File → Logon R/3 System*.
2. Choose the Administration tab.
3. Perform one of the following tasks:

Task	Action
Add a new destination	Choose <i>Add Destination</i> . Enter a name for the destination definition at the <i>Dest. Name</i> . This is a string of up to 25 characters describing the destination system.
Edit an existing destination	Select the destination name from the <i>Destinations</i> list, and choose <i>Edit Destination</i> .

The *Edit Destination* dialog appears, allowing you to define a new destination or to change an existing destination definition.

The following screen shows an example of the *Edit Destination* dialog for editing an existing destination:

Defining Destination Systems



4. Choose between *Load Balancing* and *Dedicated Server*.

When using a dedicated server you specify the actual R/3 application server to use when logging on. When using load balancing you specify a message server which then selects the least busy application server for you to log onto.

5. Once you choose between *Load Balancing* and *Dedicated Server*, enter data as required for logging onto the appropriate system:

Mode	Field	Enter
<i>Load Balancing</i>	<i>Message Server</i>	The name of the computer acting as the message server. This can be in a format similar to: "hs0020.mynetwork.mycompany.net" or it can be an IP address. You may prefix the computer name with a router name in the following format: /H/router-name/H/computer-name
	<i>R/3 Sys Name</i>	The three-character system name
	<i>Group</i>	Group name, such as PUBLIC
<i>Dedicated Server</i>	<i>R/3 Host Name</i>	The computer name of the application server to use. Use the same format as when specifying the computer name for the <i>Message Server</i> , including the option to use a router prefix.
	<i>System Number</i>	R/3 system number

6. You can enter your *user* (user name), *password*, *client*, and *language* to use when logging onto the system you have defined above.

Result

If you enter the *user* (user name), *password*, *client*, and *language* information at the *Administration* tab of the *R/3 Logon* dialog, they are kept as part of the permanent definition of

Defining Destination Systems

the destination system. You will be able to use them whenever you log into this R/3 system using this destination definition.

As an alternative, you can specify the *user ID*, *password*, *client*, and *language* at the *Logon* tab of the *R/3 Logon* dialog, which will be valid for the current logon only.

Logging Onto an R/3 System

Use

Before you can work with most of the functions of the SAP Assistant, you must log onto the R/3 system you wish to use at least once.

If you have previously downloaded data from an R/3 system and saved it into a local repository, you can [work offline \[Page 46\]](#) with this data, meaning that you can work with the data in the local repository, instead of from the live R/3 system.

Prerequisite

Before you can log onto an R/3 system, you must [define it as a destination system \[Page 30\]](#).

Procedure

1. Choose the *Logon Online* button (or choose *File* → *Logon R/3 System*). The *R/3 Logon* dialog appears.
2. Choose the *Logon* tab.
3. Select a destination at the *Destination* field. You can do so by either selecting from the drop-down-list at the *Destination* field or from the *Show Recent Connections* list (The right-pointing arrow icon). If the *Destination* field is empty, no destination had been defined for you to use. You must first [define at least one destination system \[Page 30\]](#).
4. If the destination definition includes all the necessary information for the *Client*, *UserID*, and *Language* fields, you can use them to log on, by simply choosing *OK*.

Enter data into any of the fields that are missing information or for any of the fields you wish to override.

Result

If no errors occur, you are connected to the specified R/3 system. SAP Assistant downloads the BAPI or RFC metadata from the specified system and client. This may take some time, depending on the amount of BAPIs and RFC that exist on that system and client.

If downloading RFC data takes a very long time, consider the [filter you are using \[Page 38\]](#). Using no filter when viewing RFCs results in thousands of RFCs.

Switching to a Different System

Switching to a Different System

Use

You can only read data from one system at a time. This means that before you can log into a system for the sake of reading data from it, you must log off from the system you were previously logged onto.

This means that before switching systems in the following situations, you must first log off from the current system:

- To switch from one system to another in online mode
- To switch from one system to another in offline mode
- To switch from being connected offline to being connected to an R/3 system in online mode

You do **not** need to log off from the current system in the following situations:

- Connecting offline (logging onto a local repository) while being connected online for the purpose of saving data into the local repository. As the matter of fact, connecting first to an online system and then opening the target local repository is the correct procedure for saving data into a local repository. The reason you do not have to log off the local repository in this case is because you are not reading data from the local repository, you are only writing into it.
- You also do not need to log off from a system you are using when exiting SAP Assistant. SAP Assistant logs you off automatically when you exit.

Procedure

1. Log off the current system, by choosing the *Logoff* button, or by choosing *File → Logoff*.
2. [Log onto the desired R/3 system \[Page 33\]](#) or [Open the desired local repository \[Page 49\]](#).

Using Multiple SAP Assistant Windows

Use

With each SAP Assistant window you can work with a single R/3 system/client pair. Even when [working with a local repository \[Page 46\]](#), you can only work with one system at a time.

Opening another SAP Assistant window allows you to work with multiple systems or local repositories simultaneously.

Procedure

Choose the *New Browser Window* button or choose *File → New Window*.

Result

A new SAP Assistant window opens, allowing you to log into a different system or local repository. When the new window opens, you are not logged into the system or local repository with which you are working in the previous window.

Browser Views

Browser Views

At the BAPI/RFC Tree pane of the Browser window you can only see one of the following views at a time:

Tab	View	Displays
BAPI	Applications hierarchies	<p>All of the application hierarchies in the system and their BAPIs. This consists of:</p> <ul style="list-style-type: none"> • a list of all the application areas for which there are BAPIs • each of the application areas expands to show all of its business objects • each of the business objects expands to show the list of its methods (BAPIs).
	Business objects	<p>All the business objects in the system. This includes all the BAPIs of each of the business objects. Unlike the Application hierarchy view, the business objects are listed individually, and not sorted by application area.</p>
RFC	Function groups	<p>All of the function groups in the system and their RFC functions. This consists of the groups of RFCs, and under each group name, the list of the RFCs belonging to that group.</p>
	Functions	<p>All of the RFCs in the system. Unlike the Function group view, the RFCs in this view are listed individually, not sorted by function group.</p>

When you view BAPIs you cannot view RFCs and the opposite is also true.

The default view when you log onto an R/3 system is the Application hierarchies view.

Activities

You can [change the view after logging on \[Page 37\]](#).

Changing Views

Use

When you log onto an R/3 system or open a local repository file, the Browser windows shows the application hierarchies in that system.

After logging on you can switch to any of the other [views \[Page 36\]](#):

- Business objects
- Function groups
- Functions

Procedure

- To switch to another view in the same tab, mark the top of the hierarchy and use the Context menu (right-mouse menu).

For example, to switch to viewing individual BAPIs from Application Hierarchies view, select the R/3 line at the top of the hierarchy, click the right mouse button, and choose *View Business Objects*.

- To switch to a view in another tab (an RFC view from a BAPI view and back):

Choose the desired view from the *View* menu. For example, to view Function Groups, choose *View → RFC → Function Groups*.

The following table summarizes your choices for views:

To specify	Choose
BAPI view	One of these object types: Application Hierarchies Business Objects
RFC view	One of these object types: Function Groups Functions

The tab at the BAPI/RFC Tree pane changes according to your selection (for example, if you chose Function Groups, the RFC tab becomes active).

Specifying a Filter for RFCs and Function Groups

Use

You can specify a filter to use next time you display RFC Function Groups or Functions.

Procedure

1. Choose *View → Options*.
2. Choose the *Filter* tab (the default).
3. Specify the filter selection criteria in either the *Function Groups* or *Functions (RFC)* fields. (Note that filtering BAPIs is not available yet).

Result

The filter is applied next time you change to one of the RFC views. It stays in effect until you change it.

If you set the filter before logging on, then the data downloaded and displayed initially is filtered accordingly.

However, if you change the filter after connecting and after the data is already displayed in one of the RFC views, then you must refresh the view or [change the filter \[Page 39\]](#).

To refresh the view with the new filter, choose either *View → RFC → Functions* or *View → RFC → Function Groups*.

The filter that is in effect is shown at the top of the hierarchy in the RFC tab.

Changing Filter

Use

To change the filter definition while already in one of the RFC views.

Procedure

1. In the *RFC* tab, with one of the *RFC* views active invoke the context menu (right-mouse-button).
2. Specify the new filter string at the *RFC Filter* dialog.
3. Choose *OK*.

Searching for Objects or Functions

Searching for Objects or Functions

Use

Using the Search tab in the BAPI/RFC Tree pane you can search for a specific business object, RFC function, or function group.

Procedure

1. Choose the *Search* tab.
2. Select *BO* (for business object), *RFC* (for RFC function), or *Fn Groups* (for function groups).
3. Enter the search text.
 - In all searches case is not important.
 - You can use a wildcard with the search (see below).
4. Choose *Search*.

Using Wildcards with the Search

The *Search* tab allows you to use wildcards ("*"). For example, searching for a business object using the search string "company*" finds the *Company* and *CompanyCode* business objects.

You can use a trailing wild card as in the above example when searching for RFCs and business objects.

When searching for RFCs you can also use a wildcard prefix. For example, to find all the RFCs that contain the word "system" anywhere in their name, specify *system*. The prefix wildcard feature is not available when searching for business objects.

Result

The items that satisfy the search criteria are displayed at the Search tab.

You can now perform the following tasks with the RFCs or business objects found in the search:

- [Run an RFC function \[Page 52\]](#)
- [Mark for code generation \[Page 53\]](#)
- [Save to a local repository file \[Page 47\]](#)

Browsing Business Objects and RFCs

Displaying Function Parameters and their Details

Procedure

Viewing the parameters of an RFC function is similar to [viewing the parameters of a single BAPI \[Page 44\]](#).

1. Find and select the function name in the *BAPI/RFC tree* pane. To do so, [switch to one of the RFC views \[Page 37\]](#) and look for the function in the hierarchy, or [use the Search tab \[Page 40\]](#), if you know at least a part of its name.
2. The *Type* tab of the *BAPI/RFC tree* pane displays general details of the function. Choose the *Parameters* tab to list all parameters for the selected function.
3. Double-click on the name of a parameter to invoke the *Properties* pane, in which you can view all the details of a single parameter.

Once the *Properties* pane opens, you only need to click on a parameter name to view its details in the *Properties* pane.

Displaying Key Information for Business Objects

Use

The Key of a BAPI is a field or a set of fields which together identify a unique record in the R/3 table. For example, the key fields for the GeneralLedgerAccount business object are the COMPANYCODE and the GLACCT, which is the G/L account number.

This procedure describes how to get a list of the key field(s) of a business object.

Procedure

4. Find and select the business object name in the *BAPI/RFC tree pane*. To do so, [switch to one of the BAPI views \[Page 37\]](#) and look for the business object in the hierarchy under its application hierarchy, or [use the Search tab \[Page 40\]](#), if you know at least the first part of the business object name.
5. The *General* tab of the *BAPI/RFC tree pane* displays general details of the business object, such as its internal name. Choose the *Key* tab to list all the key fields for the selected business object.
6. Double-click on the name of a key field to invoke the *Properties* pane, in which you can view all the details of the key field.

Once the *Properties* pane opens, you only need to click on a key name to view its details in the *Properties* pane.

Displaying BAPI (Method) Parameters

Displaying BAPI (Method) Parameters

Procedure

Viewing the parameters of a business object is similar to [viewing the parameters of an RFC function \[Page 42\]](#).

7. Find and select the BAPI name in the *BAPI/RFC tree* pane.

To do so, [switch to one of the BAPI views \[Page 37\]](#) and look for the BAPI in the hierarchy under its business object. You can also [use the Search tab \[Page 40\]](#), to look for the parent business object, if you know at least the beginning of its name.

8. The *Type* tab of the *BAPI/RFC tree* pane displays general details of the BAPI, such as its internal name. Choose the *Parameters* tab to list all parameters for the selected BAPI.
9. Double-click on the name of a parameter to invoke the *Properties* pane, in which you can view all the details of a single parameter.

Once the *Properties* pane opens, you only need to click on a parameter name to view its details in the *Properties* pane.

Viewing R/3 Table and Structures

Use

You can use the SAP Assistant table browser to view the metadata of an R/3 table or an R/3 structure.

Prerequisites

You must know the name of the table or structure.

Note that viewing a table or structure is not related to browsing BAPIs or RFCs, and therefore it is not related to any activity in the BAP/RFC browsing area (in the three panes of the browser)

Procedure

1. Choose *View → Table/Structure Browser* or choose the *Table/Structure Metadata Browser* icon.
2. Enter the name of the table or structure at the *Table/Structure Name* field.
3. Choose *Display*.

Result

The *Table/Structure Browser* dialog displays information about the various fields of the R/3 table or structure.

You can change the size of the columns in the *Table/Structure Browser* dialog.

Dismiss the dialog by choosing *Close*.

Working Offline: Using a Local Repository

Working Offline: Using a Local Repository

Purpose

Since BAPI and RFC metadata does not change often, you may be able to work with BAPI and RFC metadata which you have previously downloaded from an actual R/3 system. This allows you to work offline, meaning that you work with data in a local repository. This speeds your access to the data.

You can create multiple local repositories, each containing a different set of metadata.

A SAP Assistant local repository file uses the file type *SDB*.

Process Flow

To work with a local repository you must first save some metadata into a local repository file. You copy this data from an R/3 system.

1. You must first [log onto the R/3 system \[Page 33\]](#) whose data you wish to save.
2. [Save data into a local repository file \[Page 47\]](#).
3. [Open the local repository of your choice to read data from it \[Page 49\]](#). Before reading data from the local repository, close the connection to the R/3 system (log off from the online connection).

You may [delete items from a local repository \[Page 50\]](#).

You may also delete a local repository file, by deleting the SDB file in Windows.

Saving Data to a Local Repository

Use

To create a copy of the RFC or BAPI metadata from an R/3 system, you may save it into a local repository, which resides in a local file on your hard disk.

You may elect to save all the RFCs or business objects of a single system into one local repository or you can save a subset of this data. The smallest item you can save is a single RFC or a single business object.

You may save metadata from multiple R/3 systems in a single local repository file. If you combine data from multiple systems in one local repository file, you will have to choose which system to "log" onto when using the local repository (you can only read from one system at a time).

Procedure

1. [Log onto the live R/3 system \[Page 33\]](#) from which you wish to copy metadata.
2. Find the data you wish to save. You can save data from the BAPI, the RFC, or the Search pane.
3. Mark the portion of the data you wish to save.

You may save a subset or all of the data, for example, you can save all or some of the BAPIs in a system as follows:

To Save:	Mark:
All of the business objects and BAPIs in the system	The line at the top, titled "R/3"
All the business objects and BAPIs that belong to one application hierarchy (an R/3 application module, for example, Asset Accounting)	The name of the application hierarchy
A single Business object with all its BAPIs	The name of the business object

Note that you cannot save a single BAPI of a business object.

4. Open the local repository file you wish to save into. You can create a new file or you can save into an existing file:

To Save Into	Action
A new local repository file	Choose <i>File</i> → <i>New File</i> The <i>New Local Repository</i> dialog appears: Specify or Browse to the directory you wish to store the file at. Specify a file name.

Saving Data to a Local Repository

An existing local repository	Choose <i>File → Open File</i> The <i>Open Local Repository</i> dialog appears: Browse to the directory where the file is stored and mark the file name. Choose <i>Open</i> .
------------------------------	---

You can skip this step if you already have a local repository file open, and you wish to save the data into that file.

5. Choose *File → Save Selected Item*. The data you have selected is saved into the local repository file.

Saving Additional Data to an Open Local Repository

The local repository file remains open until you:

- Close it, by logging off from it
- Open another file
- Exit SAP Assistant

While it remains open you can save additional data into the same file without having to reopen it.

You can save additional data from the same system by selecting it and choosing *Save Selected Item*.

You can also save data from other R/3 systems to the same file. To do so:

1. Log off from the online connection to the first system.
2. Log onto the next system from which you wish to save data.
3. Choose *Save Selected Item*.

Reading from a Local Repository (Offline)

Use

To read BAPI or RFC metadata from a local repository file, instead from a live R/3 system, you can log onto a system in Offline mode.

Prerequisites

- You must have previously [saved the desired metadata from an R/3 system \[Page 47\]](#).
- You must also be logged off from any live R/3 connection or from any other local repository you are working with. To disconnect from the other system or local repository choose the *Logoff* button or choose *File → Logoff*.

Procedure

1. Choose *File → Open File*.
2. Browse for the local repository file you wish to work with at the *Open Local Repository* dialog. Select the desired file, and choose *Open*.
3. If the local repository file contains data from more than one system, then you need to specify which of them you wish to work with. The *Available SAP Systems* dialog displays the list of system whose data is included in the local repository you have selected. Select one of these systems.
4. Choose *Open*.

Result

The SAP Assistant displays the metadata that exists in the selected system/client pair in the selected local repository file. Remember that this does not necessarily include all the metadata that exists in the original R/3 system. It only includes the metadata that had been saved into the local repository file.

Deleting a System From a Local Repository

Deleting a System From a Local Repository

Use

You may delete all the metadata that belongs to one R/3 system from a local repository file.

If you wish to delete all the data in a local repository file, you can delete the SDB file itself.

Prerequisites

You must be logged off from any live R/3 connection or from any other local repository you are working with. To disconnect from the other system or local repository choose the *Logoff* button or choose *File → Logoff*.

Procedure

5. Choose *File → Open File*.
6. Browse for the local repository file you wish to delete data from at the *Open Local Repository* dialog. Select the desired file, and choose *Open*.
7. At the *Available SAP Systems* dialog select the system whose data you wish to delete.
8. Choose *Delete*.
9. Confirm the deletion at the SAP Assistant dialog box.

Exporting Functions and Business Objects to Excel

Use

You can download the data of a business object, a whole application hierarchy, a function, or a function group to an Excel spreadsheet.

You can download data from an R/3 system you are logged into or from a local repository file.

Prerequisites

[Connect to the R/3 system \[Page 33\]](#) or [open the local repository file \[Page 49\]](#) you wish to download data from.

Procedure

1. Select the data you wish to download.
2. Choose *View → View Selected Item in MS Excel*.

Result

SAP Assistant opens an Excel spreadsheet and downloads the selected business object, application hierarchy, function, or function group details. You can now save the spreadsheet and perform any other Excel operation on the data in it.

Note that you cannot download a single BAPI. If you select a BAPI, the data for the business objects and its other BAPIs is downloaded.

Calling an RFC Function

Calling an RFC Function

Use

To make a remote function call from within SAP Assistant.

Prerequisites

You must be either [logged on to an R/3 system \[Page 33\]](#), or you must be [connected to a local repository file for reading \[Page 49\]](#).

The function you wish to run must be displayed at one of the tabs at the *BAPI/RFC Tree* pane (either the BAPI, RFC, or Search tabs).

Procedure

1. In the *BAPI/RFC tree pane* find and select the RFC function name or the name of the BAPI whose underlying RFC function you wish to run.
2. Choose *File → Run Selected RFC/BAPI*.
To run an RFC you must be logged on. If you are not logged on, the SAP Assistant, asks if you wish to log on using the last logon parameters. If you choose Yes, SAP Assistant logs onto the system you last logged onto.
3. Enter the parameters necessary for running the RFC.
4. Choose *Run* to execute the call.
5. When the call returns, you can check the RETURN parameter. This parameter contains error messages, if any occurred. The value of the various exporting parameters contain the data returned from the call. If an exporting parameter is a single field its data appears at the value column. If the returned value is a structure or a table, click the table icon to see its fields and their values.

Code Generation for Business Objects and RFC Functions

Use

To make it easier for you to program external applications that use R/3 business objects and remote function calls, the SAP Assistant can generate code for:

- Business objects

The code generated for business objects comprises proxy classes for individual business objects along with their methods. The code generator allows you to choose which R/3 business objects and which of their methods to generate code for. It even allows you to specify which of the parameters of each of the methods to include in the generated code.

- RFC functions

The code generated for RFC functions comprises RFC proxy classes for individual RFC functions. The code generator allows you to choose the functions to generate code for. It then generates code for the selected function(s) with all of their parameters.

The SAP Assistant can generate code for object-oriented languages such as C++ and Java. You can also create templates and macros to enable code generation for another programming language.

After generating the code with the Sap Assistant, you can use the generated code to create your application, creating instances of the objects defined in the generated code's classes.

For example, for the business object *SaleOrder*, the code generated for Java is the Java class *jboSalesOrder* and for C++ the class *CBoSalesOrder*. These proxy classes, in turn, can be instantiated to represent a *SalesOrder* instance that exists in R/3.

In addition, these classes have member function that you can implement to make remote calls to the BAPI methods of the business object. For example, from the business object *SalesOrder*, you can make remote calls to such BAPI methods as *SalesOrder.Createfromdata*, *SalesOrder.Getlist*, and *SalesOrder.Getdetail*.

See Also

See the topic [Generating Code \[Page 55\]](#) for the detailed steps of generating code. Also see [The C++ BAPI Proxy Classes \[Page 59\]](#) and [The Java BAPI Proxy Classes \[Page 87\]](#) topics for the details of the generated proxy classes.

System Requirements for Code Generation

System Requirements for Code Generation

Requirements for Generating the Code

To generate the code for the BAPI proxy classes you need the following software:

- Windows NT 4.0 (Service Pack 3 or higher), Windows 95, Windows 98, or Windows 2000
- SAP Assistant installed and running

Requirements for Using the Generated Proxy Classes

For the Windows Version:

To program using the generated code for C++ on Windows you need:

- Windows NT 4.0 (Service Pack 3 or higher) or Windows 2000
- Visual Studio 98 (Visual C++ 6.0) with Visual Studio Service Pack 3

For the HP UNIX Version:

You can program with the C++ code generated for business objects and BAPIs on HP Unix. For this you need:

- The UNIX operation system (The code was tested on HP-UX 10.20).
- The HP aC++ compiler (the compiler is included in the C/C++ SoftBench Solutions). The code was tested on SoftBench Solutions D.06.20.

Programming with the code for RFC functions with the UNIX platform is not supported.

Generating Code

Use

To specify the parameters for generating code, and to generate code with the SAP Assistant.

You can specify items such as which business objects or RFC functions to generate code for, which target language to generate the code for, where to place the generated files, and which code generator to use.

Prerequisites

The code generator of the Sap Assistant gets the metadata for the specified business objects or functions from a specific R/3 system. You must therefore be logged onto that R/3 system.

Alternatively, you can use a [local repository \[Page 46\]](#), in which you have previously saved the metadata of the business objects or functions you wish to generate code for.

It may be more efficient to use the local repository for code generation, for example, if you are generating code for many objects or functions, or if the network connection between the application's computer and the R/3 server is slow.

Procedure

1. Display the appropriate view to see either business objects or RFC functions.
2. For each business object or RFC function you wish to generate code for:
 - a. Select a single business object or RFC function.
 - b. Choose *Code Generator* → *Mark For Code Generation*.
The icon next to the marked item changes to an arrow.
You can mark in this way multiple business objects along with multiple RFC functions.
You can switch between RFC and business objects views until you complete your selection of items for code generation.
3. When you are done selecting business objects and functions, choose *Code Generator* → *Generate Code*.
4. The *BAPI Wizard - Introduction* screen appears. Check the box *skip this screen in future*, if you wish. Choose *Next*.
5. The *BAPI Wizard - Language* screen appears.

Enter data in the *BAPI Wizard - Language* screen as follows:

Field	Description
<i>Output Folder</i>	Enter the path of the file where you wish the generated files to be placed. The code generator uses the output directory you specify to create subdirectories in which to place the files belonging to the individual business objects or RFC functions. For example, for the business object <i>CompanyCode</i> , the BAPI Wizard would create a subdirectory <i>companycode</i> (if it did not already exist) to store all the files generated for that business object.

Generating Code

Language	Select the programming language in which the proxy classes are generated. Choose between C++ and Java. To use a Custom language, you or someone else must have created custom templates. Creating custom templates is not described in this document.
Generate DCOM Connector Proxies	<p>Checking this box uses a different code generator than the one used for the C++ or Java options above: it uses the code generator that comes with the SAP DCOM Connector product.</p> <p>Check this box only if you wish to generate code for VB or other COM-compatible languages.</p> <p>For C++ we recommend that you do not use this code generator (do not check this box).</p>
Template and Macro	<p>SAP Assistant points to the template and macro files of the appropriate language. It includes the full path to these files, using the default subdirectory for the template and macro files (where these files were originally installed). Normally you should take the path suggested by the code generator.</p> <p>However, if the SAP Assistant executable or the template or macro files have been moved to another directory, the path to the template and macro files may be invalid.</p> <p>It is highly recommended that you leave all of these files in their installed location. However if you do move any of these files, make sure that the path and file name specified in the <i>Template</i> and <i>Macro</i> fields are correct. You can choose the button next to the <i>Template</i> or <i>Macro</i> field to browse for the applicable file.</p>

6. Choose *Next*.
7. The *BAPI Wizard - BAPI Selector* screen appears. This screen allows you to pick and choose a subset of the methods and even a subset of method parameters for business objects. It also allows you to enter values for individual fields.
See [Navigating the BAPI Selector screen \[Page 58\]](#) for details of how to select items or enter values.
8. Choose *Next*. The *BAPI Wizard - Final Step* screen appears.
9. When you are satisfied with your selections and with the preset values for parameters and fields, choose *Finish* to start the code generation.
To examine the code, double-click the entry in the *List of Generated Files*.
10. To exit the BAPI Wizard, choose *Cancel*.

Result

The BAPI Wizard generates the code and displays the path and name of all the generated files.

See Also

[The C++ BAPI Proxy Classes \[Page 59\]](#), [The Java BAPI Proxy Classes \[Page 87\]](#)

Navigating the BAPI Selector Screen

Navigating the BAPI Selector Screen

The *BAPI Wizard - BAPI Selector* screen displays all the items you have selected in a hierarchy tree.

By default, the BAPI Wizard screen displays all the RFC functions, all the business objects and all of the methods of the business objects you have marked for code generation before invoking the BAPI Wizard.

Selecting Items

Use the mouse commands in the following table to navigate through the tree to select functions, methods, parameters and fields for generation:

Task	Command
Expand a branch of the hierarchy tree	Click the plus sign (+) next to the branch, or click the right mouse button.
Collapse a branch of the hierarchy tree	Click the minus sign (-) next to the branch, or click the right mouse button.
Select an item	Select the checkbox next to an item, not the text, and click the left mouse button.
Deselect an item	Select the checkbox next to the item and click the left mouse button. Note that when you deselect a branch, the BAPI Wizard automatically deselects all subsidiary objects.

Entering Values

Select a field in the tree for which you want to enter a default value. The information for the field's data type and length automatically appears.

Enter the default value in the *Value* field. The text must conform to your compiler's syntax.

You can enter values for multiple fields in this manner.

The C++ BAPI Proxy Classes

[Introduction \[Page 60\]](#)

[The Business Object Proxy Classes \[Page 62\]](#)

[The Parameter Container Classes \[Page 64\]](#)

[The Reference Structure Type Classes \[Page 65\]](#)

[ABAP to Native Data Type Mapping \[Page 67\]](#)

[The CBoBase Class \[Page 68\]](#)

[The CBoKey Class \[Page 72\]](#)

[The CBoTable Template Class \[Page 75\]](#)

Introduction

Introduction

The C++ BAPI Proxy Classes are C++ classes generated using the BAPI Wizard component of the SAP Assistant, utilizing R/3 Business Object Repository metadata. The BAPI Proxy Classes enable C++ desktop application programmers to create proxy objects that correspond to and communicate with already and newly created business objects that exist in R/3 database. For example, when a desktop application gathers data for a job applicant and intends to store the collected data in R/3, the desktop application can simply use the C++ BAPI proxy class for the business object *Applicant* to create a proxy object, fill the parameters and call the appropriate method exposed by the business object for creating a new instance of *Applicant* in R/3 database. Once the new *Applicant* instance is created in R/3, the proxy object that runs on the desktop application can be used to correspond with the business object in R/3 for data retrieval and modification. Each business object that exists in R/3 is identified by its *key*, consisting of one or more fields(*keyfields*). A proxy object that runs in a desktop application can establish itself as the proxy object of a particular instance of business object in R/3 by identifying itself using the same key as the intended instance in R/3.

The classes produced by the BAPI Wizard for each business object fall in several categories, and the following topics discuss the class categories in more detail.

To use the proxy classes, the application program must compile with the header files from the RFC C++ Class Library (`crfc*.h`) and the RFC Library and link with these 2 libraries (`rfcclass.lib` and `librfc32.lib` respectively), which are also part of the SAP Automation package.

Classes Generated for each Business Object

The following list shows the files/classes generated for R/3 Business Object *ProfitCenter* by the BAPI Wizard:

Class	File	Category
<code>CBoProfitCenter</code>	<code>cboprofitcenter.h/.cpp</code>	(1)
<code>CBoProfitCenter::CGetlistParams</code>	"	(2)
<code>CBoProfitCenter::CGetdetailParams</code>	"	(2)
<code>CBoBapi0002_3</code>	<code>cbobapi0002_3.h/.cpp</code>	(3)
<code>CBoBapi0015_1</code>	<code>cbobapi0015_1.h/.cpp</code>	(3)
<code>CBoBapi0015_2</code>	<code>cbobapi0015_2.h/.cpp</code>	(3)
<code>CBoBapireturn</code>	<code>cbobapireturn.h/.cpp</code>	(3)

The 'Category' column in the above table lists the class category each class belongs to. The numbers denote the following:

Category	Denotes
(1)	The Business Object Proxy Classes [Page 62]
(2)	The Parameter Container Classes [Page 64]
(3)	The Reference Structure Type Classes [Page 65]

The Business Object Proxy Classes

The *Business Object Proxy Classes* category contains classes that are used as proxy classes for the business objects themselves. For example, for the business object *ProfitCenter*, there is a C++ class `CBoProfitCenter` that can be used as proxy class. In the class `CBoProfitCenter`, there are member functions defined and implemented to execute the call to the business object methods of the same name. For example, the method in the Business Object Repository `ProfitCenter.Getdetail` can be called using `CBoProfitCenter::Getdetail()`. Likewise, `PurchaseReqItem.Getlist` can be called using `CBoPurchaseReqItem::Getlist()`.

The Business Object Proxy Classes are defined in `cbo<business_object_name>.h` and are implemented in `cbo<business_object_name>.cpp`.

BAPI Method Categories

There are 3 categories of BAPI methods: factory, instance and class.

- Factory methods create new instances of the business object in R/3 database. The application program using the C++ BAPI Proxy Classes cannot directly call the factory method simply because they are not exposed by the classes in the category of Business Object Proxy Classes. To create new instances of R/3 business objects and to construct proxy objects to represent the newly created instances, please read the section „Construction of Proxy Objects“ on this page.
- Instance methods access a particular instance of an R/3 business object. The application program must have a constructed proxy object in order to call instances methods. Refere to „Construction of Proxy Objects“ section on this page. Examples of this category of BAPI methods include „`Getdetail`“ and „`Getstatus`“, which returns the detail information and status of a particular instance of business object, respectively.
- Class methods are used to obtain information regarding the objects of the same class. In the case of R/3 business objects, the class methods obtain information regarding the instances of a given business object, such as a list of profit centers obtained using `ProfitCenter.Getlist`. The class method `ProfitCenter.Getlist` can be called using `CBoProfitCenter::Getlist()` and the application program does not need to construct a proxy object in order to call this function, since the proxy member functions are declared `static` in the classes of the category Business Object Proxy Classes.

Business Object Identification Key

Most of the business objects in R/3 database are identified using a unique identification key, and therefore the corresponding proxy object running in the desktop application should also hold the identification key. The application program using a proxy object must use the key when calling BAPI methods that operate on specific instances of business objects. For example, when a desktop application program needs to call `ProfitCenter.Getdetail` to obtain detail information on a specific profit center, then the identification key of the proxy object must be correctly set to indicate the particular profit center whose detail information is desired. The key is implemented using the `CBoKey` class, defined in `cboglobal.h` and implemented in `cboglobal.cpp`.

Each identification key consists of one or more fields, called *keyfields*. The content of individual keyfield can be read or written using getter and setter functions in the `CBoKey`

The Business Object Proxy Classes

class. The entire key is used for identifying a particular instance of a business object in R/3, and therefore all keyfields need to be set correctly.

All of the business object proxy classes are derived from class `CBoBase`, which holds attributes common to all business object proxy classes such as business object name, type, and the identification key. The application program uses the `CBoBase::GetKey()` function to access the `CBoKey` attribute in any business object proxy object, and the `CBoKey` class offers functions to set and retrieve the values of individual fields in the identification key itself.

Construction of Proxy Objects

The construction of a C++ proxy object for a business object is done using the constructors in the classes of the Business Object Proxy Classes category. There are 2 general ways of constructing a proxy object:

1. Using a constructor that takes a parameter container class object as argument: this type of constructor internally makes a BAPI factory method call to R/3 to create a new instance of the business object. By doing so, the newly constructed proxy object becomes the proxy object for the newly created business object in R/3. Care must be taken that the parameter container object to be passed as argument be filled with appropriate input data.
2. Using a constructor that does not take a parameter container class object as argument: this type of constructor is used to construct an empty proxy object first, then the application program can use the `CBoBase::GetKey()` function to set the identification key for the purpose of designating the newly constructed proxy object to represent the R/3 business object instance holding the same identification key.

See also:

[The CBoBase Class \[Page 68\]](#)

[The CBoKey Class \[Page 72\]](#)

[The Parameter Container Classes \[Page 64\]](#)

The Parameter Container Classes

The Parameter Container Classes

In the C++ business object proxy class, for each member function that interfaces with a business object method, there is a corresponding parameter container class that contains the parameters used for calling the method. For example, `CBoProfitCenter::Getdetail()` has a class `CBoProfitCenter::CBoGetdetailParams` that contains all the defined parameters for the method `ProfitCenter.Getdetail`. Using the member functions in this class, the application program sets the parameter values in preparation for a method call, and retrieves the returned parameter values after the method call.

The parameter container class contains 3 types of parameters: simple, structure and table parameters.

- **Simple parameters:** each simple parameter has a setter and getter function for setting and getting the value of the simple parameter. The setter function name has the format `Set<parameter_name>(...)`, and the getter function name has the format `Get<parameter_name>()`. `parameter_name` itself has the format of beginning with an upper-case character and with the rest of the characters in lower-case.
- **Structure and table parameters:** unlike simple parameters, structure and table parameters have only the getter function, without the accompanying setter function. The getter function name has identical format as the the getter function name for simple parameters. The getter functions for structure and table parameters return a reference to an object which is the proxy object of the structure or table parameters. Then, the application program can use the setter and getter functions of the individual fields in the structure or table parameters to access the individual fields. For description of the proxy classes for structure and table parameters, please refer to [The Reference Structure Type Classes \[Page 65\]](#).
- **Table parameters:** the proxy classes for table parameters use the template class `CBoTable` in conjunction with the reference structure type proxy classes for accessing individual rows using index. Please refer to [The CBoTable Template Class \[Page 75\]](#).

The classes of this category are defined in `cbo<business_object_name>.h` and are implemented in `cbo<business_object_name>.cpp`, in the same file where the parent business object proxy classes are defined.

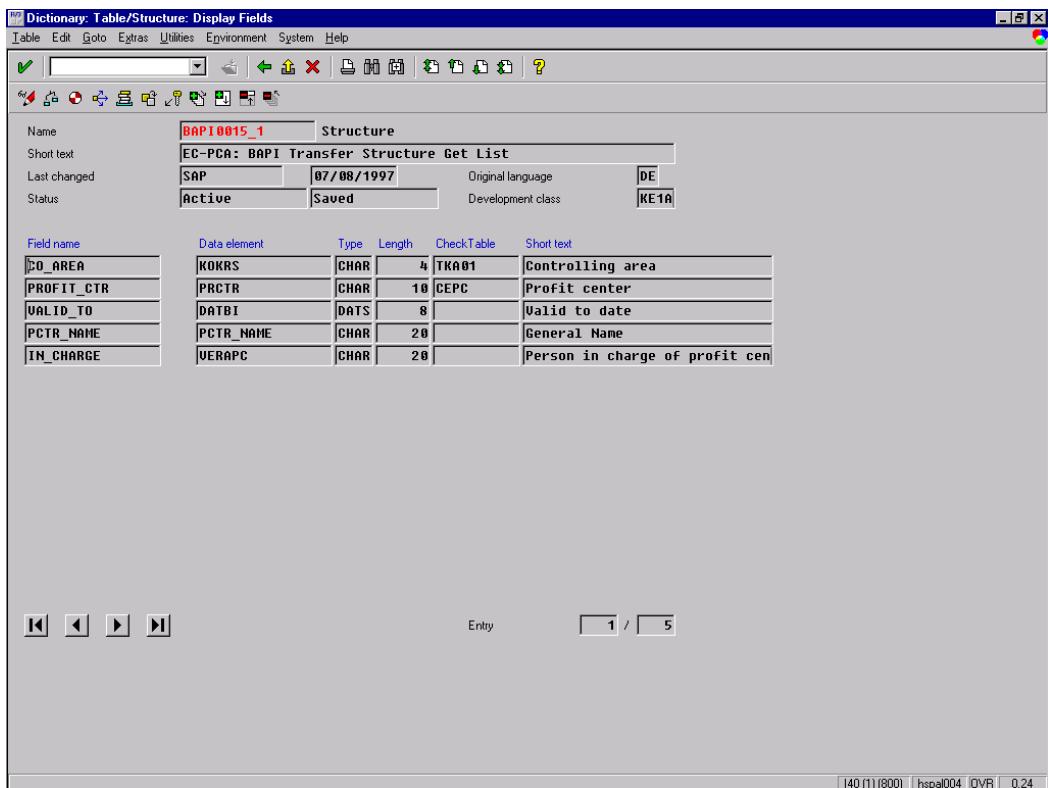
See also:

[The Business Object Proxy Classes \[Page 62\]](#)

The Reference Structure Type Classes

The Reference Structure Type Classes

This category of classes are proxy classes for the reference structure types defined in the ABAP Dictionary and can be viewed using R/3 Transaction SE11. Examples of reference structure types abound, and some well-used ones are BAPIORDERS, BAPIITEMIN, BAPIEBAN, BAPI0015_1 and BAPI0002_3. These reference structure types contain fields of various ABAP data types (see [ABAP to Native Data Type Mapping \[Page 67\]](#)), and are used for structure parameters and records of table parameters of BAPI methods.



Naming convention for C++ proxy classes of reference structure types

The C++ proxy classes for these reference structure types offer setter and getter functions for writing to and reading from the fields in the reference structure, respectively, and are named using the following the convention. The naming convention dictates that the names are prefixed with the characters „CBo“ and followed by the name of the reference structure type itself, with the first character in upper case and the rest in lower case. The names of the proxy classes corresponding to the above-mentioned reference structure types would be CBoBapiorders, CBoBapitemin, CBoBapieban, CBoBapi0015_1, and CBoBapi0002_3, respectively.

Naming convention for getter and setter functions of fields

The naming convention for the setter and getter functions for the fields dictates that the setter function name be prefixed with the characters „Set“ and the getter function name be prefixed

The Reference Structure Type Classes

with the characters „Get“. The naming convention for the field names deletes all the underscore characters („_“) and converts all characters to lower-case, except for the characters that immediately follow the deleted underscore characters. For example, the setter function name for field ORDER_DATE would be `SetOrderDate()`; the getter function name for field FAX_NUM_1 would be `GetFaxNum1()`.

Native data types returned by field getter functions

The native data types returned by field getter functions are listed in section [ABAP to Native Data Type Mapping \[Page 67\]](#).

Native data types required by field setter functions

The native data types used by field setter functions in as argument are also listed in the table in section [ABAP to Native Data Type Mapping \[Page 67\]](#). However, for `string` objects, the argument for field setter function is `const string&`, to save time by passing by reference.

ABAP to Native Data Type Mapping

ABAP Data Type	C++ Native Data Type
CHAR	string (of Standard C++ Library)
INT4	int
INT2	int
INT1	int
NUMC	string
PACK	double
LANG	string
CURR	double
CUKY	string
DATS	string
UNIT	string
TIMS	string
DEC	double
QUAN	double
ACCP	string
CLNT	string
FLTP	double
RAW	string (see note), void*

Note: the `RAW` data type is mapped to the `string` class of the Standard C++ Library, but care must be taken when using this object. When the application program calls a field getter function (`Get<fieldname>()`) on a field of `RAW` type and retrieves a `string` object, the `string` object encapsulates a character array of length equal to the ABAP-defined length of the field. The character array encapsulated inside the `string` object may contain `null` character in the beginning or the middle of the array, therefore this array cannot be treated as a normal character string. Using `cout` of the `iostream` class would output a truncated character string, terminated by the first null character. Also, when the application program calls a field setter function (`Set<fieldname>()`), the argument passed is `void*`, a pointer that points to the raw byte array to be copied to the intended field.

The CBoBase Class

This class is defined in *cbobase.h*.

This class is the base class from which all Business Object Proxy Classes are derived. This class encapsulates the common attributes and functionality needed in the Business Object Proxy Classes, such as business object name and type, and the identification key. This class offers functions that return these common attributes.

Construction

CBoBase

It is discouraged to use this constructor directly by the application program.

Operations

[GetName \[Page 69\]](#)

Returns the name of the business object.

[GetType \[Page 70\]](#)

Returns the type of the business object.

[GetKey \[Page 71\]](#)

Returns reference to the identification key (`CBoKey`) attribute. This reference can then be used to set or get keyfield values.

GetName

Purpose

Returns the name of the business object.

Syntax

```
const string& GetName();
```

Parameters

None.

Return Value

Returns a reference to a `string` object containing the name of the business object.

Description

This function returns the name of the business object.

GetType

GetType

Purpose

Returns the type of the business object.

Syntax

```
const string& GetType();
```

Parameters

None.

Return Value

Returns a reference to a `string` object containing the type of the business object.

Description

This function returns the type of the business object.

GetKey

Purpose

Returns reference to the identification key (`CBoKey`) attribute. This reference can then be used to set or get keyfield values.

Syntax

```
CBoKey& GetKey();
```

Parameters

None.

Return Value

Returns a reference to the `CBoKey` attribute.

The CBoKey Class

This class is defined in *cboglobal.h*.

This class encapsulates the identification key for an instance of a business object. It offers functions for retrieving and setting the values of individual keyfields.

Construction

CBoKey

The constructor does not construct a usable object.

Operations

[GetKeyField \[Page 73\]](#)

Returns the value of the specified keyfield.

[SetKeyField \[Page 74\]](#)

Sets the value of the specified keyfield.

GetKeyField

Purpose

Returns the value of the specified keyfield.

Syntax

```
string GetKeyField(const string& strKeyFieldName);
```

Parameters

strKeyFieldName: Name of the parameter.

Return Value

Returns a `string` object containing the value of the specified keyfield.

Description

This function returns the value of the specified keyfield. If the specified keyfield does not exist, then the string returned is a null string.

Related Information

[SetKeyField \[Page 74\]](#)

SetKeyField

SetKeyField

Purpose

Sets the value of the specified keyfield.

Syntax

```
bool SetKeyField(const string& strKeyFieldName,  
                  const string& strValue);
```

Parameters

strKeyFieldName: Name of the keyfield.

strValue: Value to be set to the specified keyfields.

Return Value

Returns a true indicating that the keyfield was set to given value; returns false if given keyfield was not found.

Description

This function returns the value of the specified keyfield. If the specified keyfield does not exist, then the string returned is a null string.

Related Information

[GetKeyField \[Page 73\]](#)

The CBoTable Template Class

This class is defined in *cboglobal.h*.

This class is used in conjunction with the reference structure type proxy classes (see [The Reference Structure Type Classes \[Page 65\]](#)) for accessing the rows and their fields in table parameters. The reference structure type proxy classes are used as template arguments for CBoTable template class. The resulting class, for example, CBoTable<CBoBapi0015_1>, then offers the functions necessary to access the rows in the table parameter of type BAPI0015_1, and setter and getter functions to access the fields.

Example:

The following example shows how the application program uses a table parameter proxy class to access the rows and fields of a table parameter. Note that this example uses the reference structure type CBoBapi0015_1, which describes a profit center. The table parameter itself is called *Profitcenters*, and the function

CBoProfitCenter::CGetlistParams::GetProfitcenters() returns a reference to an object of the table parameter proxy class CBoTable<CBoBapi0015_1>. And, the CBoTable<CBoBapi0015_1> object offers the GetRow() and GetConstRow() functions for the application program to obtain access to a specified row object, which is in turn, a CBoBapi0015_1 object.

```
CBoProfitCenter::CGetlistParams GetlistParams;
CBoTable<CBoBapi0015_1>& Profitcenters;
Profitcenters = GetlistParams.GetProfitcenters();
for(int i=0; i < Profitcenters.GetRowCount(); i++)
{
    string strOut;
    CBoBapi0015_1& ThisRecord = Profitcenters.GetRow(i);

    // or use 'const' when only reading from fields
    // const CBoBapi0015_1& ThisRecord = Profitcenters.GetConstRow(i);

    strOut = ThisRecord.GetCoArea();
    cout << strOut << endl;

    strOut = ThisRecord.GetProfitCtr();
    cout << strOut << endl;

    strOut = ThisRecord.GetValidTo();
    cout << strOut << endl;

    strOut = ThisRecord.GetPctrName();
    cout << strOut << endl;

    strOut = ThisRecord.GetInCharge();
    cout << strOut << endl;
}
```

Construction

CBoTable

The CBoTable Template Class

The application program does not need to construct an object of this class.

Operations

[AppendToTable \[Page 77\]](#)

Appends a new, initialized row to the table parameter, and returns a reference to a reference structure type proxy object that represents the newly appended row.

[ClearTable \[Page 78\]](#)

Deletes all rows in the embedded table parameter.

[CopyLineFrom \[Page 79\]](#)

Copies the content of a specified memory area to the specified row in a table parameter.

[CopyLineTo \[Page 80\]](#)

Copies the content of the specified row in a table parameter to a specified memory area.

[GetConstRow \[Page 81\]](#)

Returns a ,const' reference to an object of the reference structure type proxy class that represents the desired table parameter row specified by the index.

[GetLength \[Page 82\]](#)

Returns the number of bytes in each row of the embedded table parameter.

[GetRow \[Page 83\]](#)

Returns a reference to an object of the reference structure type proxy class that represents the desired table parameter row specified by the index.

[GetRowCount \[Page 84\]](#)

Returns the number of rows currently in the embedded table parameter.

[InsertIntoTable \[Page 85\]](#)

Returns a reference to an object of the reference structure type proxy class.

[RemoveRow \[Page 86\]](#)

Removes the specified row.

AppendToTable

Purpose

Appends a new, initialized row to the table parameter, and returns a reference to a reference structure type proxy object that represents the newly appended row.

Syntax

```
AType& AppendToTable();
```

Parameters

None.

Return Value

Returns a reference to an object of the reference structure type proxy class. *AType* denotes the reference structure type proxy class.

Exceptions

Throw const char* for failure to append a memory block by RFC Library.

Description

This function appends a row to the embedded table parameter and initializes all the fields within the row. Then it returns a reference to an object of the reference structure type proxy class. For example, if the table parameter's reference structure type is `BAPI0015_1`, then the returned reference is a reference to a `CBoBapi0015_1`.

Related Information

[The Reference Structure Type Classes \[Page 65\]](#)

[GetConstRow \[Page 81\]](#)

[GetRow \[Page 83\]](#)

[RemoveRow \[Page 86\]](#)

ClearTable

ClearTable

Purpose

Deletes all rows in the embedded table parameter.

Syntax

```
void ClearTable();
```

Parameters

None.

Return Value

None.

CopyLineFrom

Purpose

Copies the content of the specified row in a table parameter to a specified memory area.

Syntax

```
void CopyLineFrom(int nIndex, void *pDestination);
```

Parameters

nIndex: a zero-based integer specifying the desired row.

pSource: a void pointer pointing to the memory area containing source data.

Exceptions

Throws const char* when RFC Library fails to copy line.

Description

The application programmer must make sure that the source data is valid.

Related Information

[CopyLineTo \[Page 80\]](#)

CopyLineTo

Purpose

Copies the content of the specified row in a table parameter to a specified memory area.

Syntax

```
void CopyLineTo(int nIndex, void *pDestination);
```

Parameters

nIndex: a zero-based integer specifying the desired row.

pDestination: a void pointer pointing to the destination memory area.

Exceptions

Throws const char* when RFC Library fails to copy line.

Description

The application programmer must make sure that the destination block of memory is large enough to hold a table row.

Related Information

[CopyLineFrom \[Page 79\]](#)

GetConstRow

Purpose

Returns a ,const‘ reference to an object of the reference structure type proxy class that represents the desired table parameter row specified by the index.

Syntax

```
const AType& GetConstRow(int nIndex) const;
```

Parameters

nIndex: a zero-based integer specifying the desired row.

Return Value

Returns a ,const‘ reference to an object of the reference structure type proxy class that represents the desired row. *AType* denotes the reference structure type proxy class.

Exceptions

Throws const char* when the specified integer index is larger than the total number of rows in the table.

Description

This function returns a ,const‘ reference to an object of the reference structure type proxy class that represents the table parameter row specified by the integer index. For example, if the table parameter’s reference structure type is BAPI0015_1, then the returned reference is a reference to a CBoBapi0015_1.

Once this ,const‘ reference is obtained, the application program will not be able to use the field setter functions to modify the contents of the fields.

Related Information

[AppendToTable \[Page 77\]](#)

[GetRow \[Page 83\]](#)

[RemoveRow \[Page 86\]](#)

GetLength

Purpose

Returns the number of bytes in each row of the embedded table parameter.

Syntax

```
int GetLength();
```

Parameters

None.

Return Value

Returns an integer indicating the length.

GetRow

Purpose

Returns a reference to an object of the reference structure type proxy class that represents the desired table parameter row specified by the index.

Syntax

```
AType& GetConstRow(int nIndex);
```

Parameters

nIndex: a zero-based integer specifying the desired row.

Return Value

Returns a reference to an object of the reference structure type proxy class that represents the desired row. *AType* denotes the reference structure type proxy class.

Exceptions

Throws const char* when the specified integer index is larger than the total number of rows in the table.

Description

This function returns a reference to an object of the reference structure type proxy class that represents the table parameter row specified by the integer index. For example, if the table parameter's reference structure type is `BAPI0015_1`, then the returned reference is a reference to a `CBoBapi0015_1`.

Once this reference is obtained, the application program will be able to use both the field setter and getter functions to modify the contents of the fields.

Related Information

[AppendToTable \[Page 77\]](#)

[GetConstRow \[Page 81\]](#)

[RemoveRow \[Page 86\]](#)

GetRowCount

GetRowCount

Purpose

Returns the number of rows currently in the embedded table parameter.

Syntax

```
int GetRowCount();
```

Parameters

None.

Return Value

Returns an integer indicating the number of rows.

InsertIntoTable

Purpose

Inserts a new, initialized row to the table parameter, and returns a reference to a reference structure type proxy object that represents the newly inserted row.

Syntax

```
AType& InsertIntoTable(int nIndex);
```

Parameters

nIndex: an integer that represents an index into the table parameter where the new row is to be inserted.

Return Value

Returns a reference to an object of the reference structure type proxy class. *AType* denotes the reference structure type proxy class.

Exceptions

Throw const char* for failure to insert a memory block by RFC Library.

Description

This function inserts a row to the embedded table parameter and initializes all the fields within the row. Then it returns a reference to an object of the reference structure type proxy class. For example, if the table parameter's reference structure type is `BAPI0015_1`, then the returned reference is a reference to a `CBoBapi0015_1`.

Related Information

[The Reference Structure Type Classes \[Page 65\]](#)

[AppendToTable \[Page 77\]](#)

RemoveRow

RemoveRow

Purpose

Removes the specified row.

Syntax

```
void RemoveRow(int nIndex);
```

Parameters

nIndex: a zero-based integer specifying the desired row.

Return Value

None.

Exceptions

Throws const char* when the specified integer index is larger than the total number of rows in the table.

Related Information

[AppendToTable \[Page 77\]](#)

[GetConstRow \[Page 81\]](#)

[GetRow \[Page 83\]](#)

The Java BAPI Proxy Classes

[Introduction \[Page 88\]](#)

[Using Java RFC packages \[Page 90\]](#)

[The Business Object Proxy Classes \[Page 92\]](#)

[The Parameter Container Classes \[Page 94\]](#)

[The ABAP Reference Structure Types \[Page 95\]](#)

[The Structure Parameter Classes \[Page 96\]](#)

[The Table Parameter Classes \[Page 97\]](#)

[The Table Row Classes \[Page 106\]](#)

[ABAP to Native Data Type Mapping \[Page 107\]](#)

[The jboBase Class \[Page 108\]](#)

[The jboKey Class \[Page 112\]](#)

[BAPI Beans \[Page 117\]](#)

Introduction

Introduction

The Java BAPI Proxy Classes are Java classes generated using the BAPI Wizard component of the SAP Assistant, utilizing R/3 Business Object Repository metadata. The BAPI Proxy Classes enable Java application and applet programmers to create proxy objects that correspond to and communicate with already and newly created business objects that exist in R/3 database. For example, when an applet or a web-based application gathers data for a job applicant and intends to store the collected data in R/3, the applet or application can simply use the Java BAPI proxy class for the business object *Applicant* to create a proxy object, fill the parameters and call the appropriate method exposed by the business object for creating a new instance of *Applicant* in R/3 database. Once the new *Applicant* instance is created in R/3, the proxy object that runs on the desktop application can be used to correspond with the business object in R/3 for data retrieval and modification. Each business object that exists in R/3 is identified by its *key*, consisting of one or more fields(*keyfields*). A proxy object that runs in a desktop application can establish itself as the proxy object of a particular instance of business object in R/3 by identifying itself using the same key as the intended instance in R/3.

The classes produced by the BAPI Wizard for each business object fall in several categories, and the following topics discuss the class categories in more detail.

To use the proxy classes, the application program must compile with the class files from the Java RFC classes (packages `com.sap.rfc` and `com.sap.rfc.exception`), which are part of the SAP Automation package.

Classes Generated for each Business Object

The following list shows the files/classes generated for R/3 Business Object *ProfitCenter* by the BAPI Wizard:

Class	File	Category
<code>jboProfitCenter</code>	<code>jboprofitcenter.java</code>	(1)
<code>jboProfitCenterGetlistParams</code>	<code>jboprofitcentergetlistparams.java</code>	(2)
<code>jboProfitCenterGetdetailParams</code>	<code>jboprofitcentergetdetailparams.java</code>	(2)
<code>jboBapi0002_3Structure</code>	<code>jboBapi0002_3Structure.java</code>	(3)
<code>jboBapi0015_1Table</code>	<code>jboBapi0015_1Table.java</code>	(4)
<code>jboBapi0015_1TableRow</code>	<code>jboBapi0015_1TableRow.java</code>	(5)
<code>jboBapi0015_2Structure</code>	<code>jboBapi0015_2Structure.java</code>	(3)
<code>jboBapireturnStructure</code>	<code>jboBapireturnStrucure.java</code>	(3)

The 'Category' column in the above table lists the class category each class belongs to. The numbers denote the following:

Category	Denotes
(1)	The Business Object Proxy Classes [Page 92]
(2)	The Parameter Container Classes [Page 94]
(3)	The Structure Parameter Classes [Page 96]

Introduction

(4)	The Table Parameter Classes [Page 97]
(5)	The Table Row Classes [Page 106]

Using Java RFC packages

In order to use Java BAPI Proxy Classes, the application programmer needs to be familiar with some aspects of the Java RFC package: `com.sap.rfc` and `com.sap.rfc.exception`. The application programmer needs to provide `IRfcConnection` objects for logging on to R/3 so that the proxy objects can communicate with their corresponding instances.

The following function shows sample code that attempts to log on to an R/3 system. Note the use of the classes in the `com.sap.rfc` and `com.sap.rfc.exception`, such as `MiddlewareInfo`, `FactoryManager`, `JRfcBaseRuntimeException`, `userInfo`, `connectInfo`, `JRfcRfcConnectionException`, etc.

```

//  

// connect  

//  

public static boolean connect()  

{  

    //cleanup previous connection  

    cleanUp();  

    try  

    {  

        // create a middlewareInfo object with rfcHost name  

        MiddlewareInfo mdInfo  

            = new MiddlewareInfo();  

        mdInfo.setMiddlewareType  

            (MiddlewareInfo.middlewareTypeOrbix);  

        mdInfo.setOrbServerName(rfcHost);  

        // let the global factory manager use the middlewareInfo  

        // to create all object factories by binding to the server  

        facMan = FactoryManager.getSingleInstance();  

        if(facMan != null)  

            facMan.setMiddlewareInfo(mdInfo);  

    }  

    catch (JRfcBaseRuntimeException je)  

    {  

        System.out.println("Bind to object failed.\n"  

            + "Unexpected JRFC runtime exception:\n" + je.toString());  

        return false;  

    }  

    connection = logon();  

    if( connection == null) return false;  

    return true;  

}  

//  

// logon  

//  

// This function attempts to log on to an R/3 system. It assumes  

// that the following static attributes are initialized:  

//  

//     String rfcHost  = null,  

//           client   = null,  

//           user     = null,

```

Using Java RFC packages

```
//          password = null,
//          r3Host    = null,
//          r3SysNo  = null,
//          language = null;
//
private static IRfcConnection logon()
{
    UserInfo userInfo = new UserInfo();
    ConnectInfo connectInfo = new ConnectInfo();
    IRfcConnection connection = null;

    userInfo.setClient(client);
    userInfo.setUserName(user);
    userInfo.setPassword(password);
    userInfo.setLanguage(language);

    connectInfo.setRfcMode(ConnectInfo.RFC_MODE_VERSION_3);
    connectInfo.setDestination("xxx");
    connectInfo.setHostName(r3Host);
    connectInfo.setSystemNo((short)Integer.parseInt(r3SysNo));
    connectInfo.setLoadBalancing(false);
    connectInfo.setCheckAuthorization(true);

    try
    {
        connection = facMan.getRfcConnectionFactory()
            .createRfcConnection(connectInfo, userInfo);
        connection.open();
    }
    catch (JRfcRfcConnectionException re)
    {
        System.out.println("Unexpected RfcError while opening
                           connection.\n" + re.toString());
        return null;
    }
    catch (JRfcBaseRuntimeException je)
    {
        System.out.println("Unexpected JRFC runtime exception:\n"
                           + " while opening connection.\n"
                           + je.toString());
        return null;
    }
    return connection;
}
```

The Business Object Proxy Classes

The *Business Object Proxy Classes* category contains classes that are used as proxy classes for the business objects themselves. For example, for the business object *ProfitCenter*, there is a Java class `jboProfitCenter` that can be used as proxy class. In the class `jboProfitCenter`, there are member functions defined and implemented to execute the call to the business object methods of the same name. For example, the method in the Business Object Repository `ProfitCenter.Getdetail` can be called using `jboProfitCenter.getdetail()`. Likewise, `PurchaseReqItem.Getlist` can be called using `jboPurchaseReqItem.getlist()`.

The Business Object Proxy Classes are defined in `cbo<business_object_name>.h` and are implemented in `cbo<business_object_name>.cpp`.

BAPI Method Categories

There are 3 categories of BAPI methods: factory, instance and class.

- **Factory** methods create new instances of the business object in R/3 database. The application program using the Java BAPI Proxy Classes cannot directly call the factory method simply because they are not exposed by the classes in the category of Business Object Proxy Classes. To create new instances of R/3 business objects and to construct proxy objects to represent the newly created instances, please read the section „Construction of Proxy Objects“ on this page.
- **Instance** methods access a particular instance of an R/3 business object. The application program must have a constructed proxy object in order to call instances methods. Refere to „Construction of Proxy Objects“ section on this page. Examples of this category of BAPI methods include „Getdetail“ and „Getstatus“, which returns the detail information and status of a particular instance of business object, respectively.
- **Class** methods are used to obtain information regarding the objects of the same class. In the case of R/3 business objects, the class methods obtain information regarding the instances of a given business object, such as a list of profit centers obtained using `ProfitCenter.Getlist`. The class method `ProfitCenter.Getlist` can be called using `jboProfitCenter.getlist()` and the application program does not need to construct a proxy object in order to call this function, since the proxy member functions are declared `static` in the classes of the category Business Object Proxy Classes.

Business Object Identification Key

Most of the business objects in R/3 database are identified using a unique identification key, and therefore the corresponding proxy object running in the desktop application should also hold the identification key. The application program using a proxy object must use the key when calling BAPI methods that operate on specific instances of business objects. For example, when a desktop application program needs to call `ProfitCenter.Getdetail` to obtain detail information on a specific profit center, then the identification key of the proxy object must be correctly set to indicate the particular profit center whose detail information is desired. The key is implemented using the `jboKey` class, defined in `jboKey.h`.

Each identification key consists of one or more fields, called *keyfields*. The content of individual keyfield can be read or written using getter and setter functions in the `jboKey`

The Business Object Proxy Classes

class. The entire key is used for identifying a particular instance of a business object in R/3, and therefore all keyfields need to be set correctly.

All of the business object proxy classes are derived from class `CBoBase`, which holds attributes common to all business object proxy classes such as business object name, type, and the identification key. The application program uses the `jboBase.getKey()` function to access the `jboKey` attribute in any business object proxy object, and the `jboKey` class offers functions to set and retrieve the values of individual fields in the identification key itself.

Construction of Proxy Objects

The construction of a Java proxy object for a business object is done using the constructors in the classes of the Business Object Proxy Classes category. There are 2 general ways of constructing a proxy object:

1. Using a constructor that takes a parameter container class object as argument: this type of constructor internally makes a BAPI factory method call to R/3 to create a new instance of the business object. By doing so, the newly constructed proxy object becomes the proxy object for the newly created business object in R/3. Care must be taken that the parameter container object to be passed as argument be filled with appropriate input data.
2. Using a constructor that does not take a parameter container class object as argument: this type of constructor is used to construct an empty proxy object first, then the application program can use the `jboBase.getKey()` function to set the identification key for the purpose of designating the newly constructed proxy object to represent the R/3 business object instance holding the same identification key.

See also:

[The `jboBase` Class \[Page 108\]](#)

[The `jboKey` Class \[Page 112\]](#)

The Parameter Container Classes

The Parameter Container Classes

In the Java business object proxy class, for each member function that interfaces with a business object method (BAPI), there is a corresponding parameter container class that contains the parameters used for calling that BAPI method. For example, `jboProfitCenter.getList()` has a class `jboProfitCenterGetlistParams` that contains all the defined parameters for the method `ProfitCenter.Getlist`. Using the member functions in this class, the application program sets the parameter values in preparation for a method call, and retrieves the returned parameter values after the method call.

See an example of a parameter container class for the BAPI method `ProfitCenter.Getlist` in the Java BAPI HTML Reference (in [Com.sap.bapi.profitcenter.jboprofitcentergetlistparams.html](#)).

The parameter container class contains 3 types of parameters: simple, structure and table parameters.

- **Simple parameters:** each simple parameter has a setter and getter function for setting and getting the value of the simple parameter. The setter function name has the format `set<parameter_name>(...)`, and the getter function name has the format `get<parameter_name>()`. `parameter_name` itself has the format that begins with an upper-case character and carries the rest of the characters in lower-case. Each simple parameter proxy is of a Java native type that corresponds to the ABAP data type of the simple parameter itself. For mapping of ABAP data types to Java native data types and vice versa, please refer to [ABAP to Native Data Type Mapping \[Page 107\]](#).
- **Structure parameters:** The getter function of the structure parameter returns a structure parameter proxy object. Likewise, the setter function takes, as argument, a structure parameter proxy object, and stores that into the parameter container object. The naming convention for the setter and getter functions of structure parameters are identical to that for the simple parameters. Please refer to [The Structure Parameter Classes \[Page 96\]](#).
- **Table parameters:** The getter function of the table parameter returns a table parameter proxy object. Likewise, the setter function takes, as argument, a table parameter proxy object, and stores that into the parameter container object. The application program then uses the member functions offered by the table parameter proxy object to access individual `table row` proxy objects, which in turn, offers setter and getter functions for the `table fields`. The naming convention for the setter and getter functions of table parameters are identical to that for the simple parameters. Please refer to [The Table Parameter Classes \[Page 97\]](#) and [The Table Row Classes \[Page 106\]](#).

See an example of a table parameter proxy class in the Java BAPI HTML Reference (in [Com.sap.bapi.purchasereqitem.jbobapiebantable.html](#)).

See an example of a table row proxy class for accessing individual fields of a table row in the Java BAPI HTML Reference (in [Com.sap.bapi.purchasereqitem.jbobapiebantablerow.html](#)).

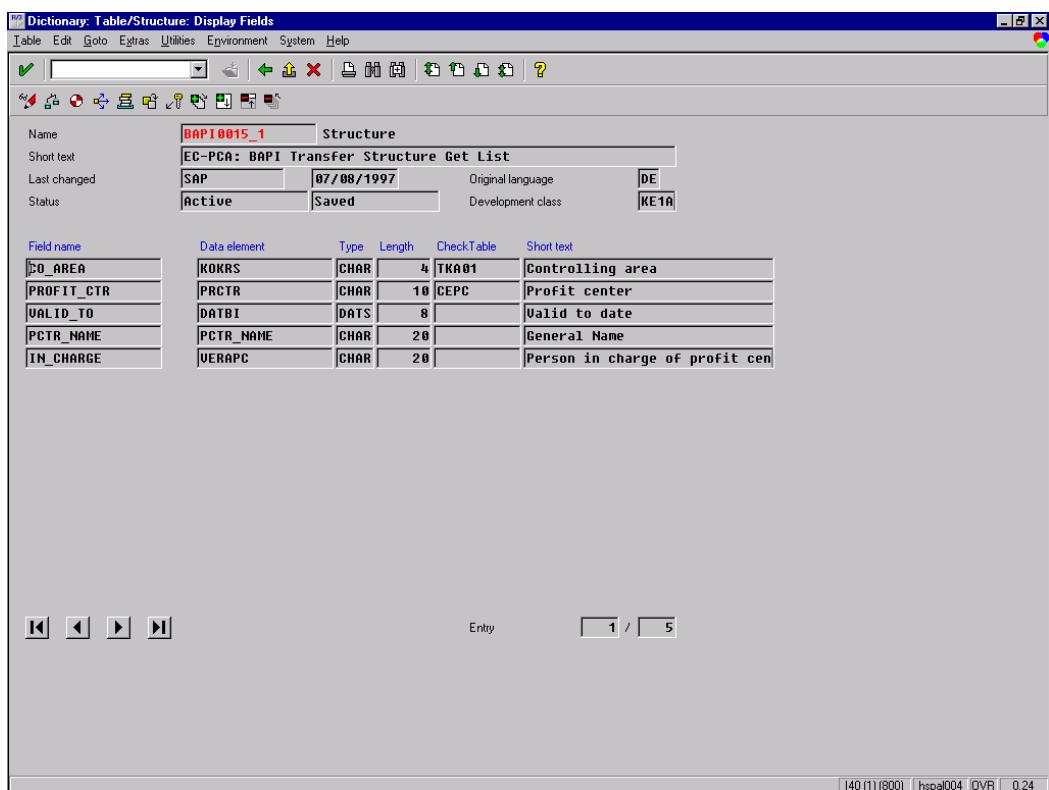
The classes of this category are defined and implemented in `jbo<business_object_name><method_name>Params.java`, where `<business_object_name>` is the name of the business object, and `<method_name>` is the name of the method with the first character in upper-case and the rest in lower-case.

The ABAP Reference Structure Types

This category of classes are proxy classes for the reference structure types defined in the ABAP Dictionary and can be viewed using R/3 Transaction SE11. Examples of reference structure types abound, and some well-used ones are BAPIORDERS, BAPIITEMIN, BAPIEBAN, BAPI0015_1 and BAPI0002_3. These reference structure types contain definition of fields: field ABAP data type ([ABAP to Native Data Type Mapping \[Page 107\]](#)), field offset, length, position and decimal places.

In the BAPI Wizard-generated parameter proxy source files, [The Structure Parameter Classes \[Page 96\]](#) and [The Table Parameter Classes \[Page 97\]](#) are based on the ABAP reference structure types. These classes provide type-safe access to individual fields of the structure or table parameters.

The following diagram shows the R/3 transaction SE11 displaying the definition of reference structure type BAPI0015_1.



The Structure Parameter Classes

The Structure Parameter Classes

The structure parameter proxy classes are proxy classes for structure parameters of BAPI methods. They offer setter and getter functions to write to and read from individual fields in the structure parameter, respectively. A field setter function takes data of a Java native type as arguments and sets its corresponding field to that data value. A field getter function returns data value of Java native type, converted from the ABAP data type of the corresponding field of the structure parameter.

See an example of a structure parameter proxy class for a structure parameter of the reference structure type `BAPI0015_2` in the Java BAPI HTML Reference (in Com.sap.bapi.profitcenter.jbobapi0015_2structure.html)

Naming convention

The naming convention of the setter and getter of fields is as follows:

- The setter function name has the format `set<field_name>(....)`, and the getter function name has the format `get<field_name>()`.
- `field_name` itself follows the convention deletes all the underscore characters („_“) and converts all characters to lower-case, except for the characters that immediately follow the deleted underscore characters.
- For example, the setter function name for field `ORDER_DATE` would be `SetOrderDate()`; the getter function name for field `FAX_NUM_1` would be `GetFaxNum1()`.

The Table Parameter Classes

The table parameter classes are wrapper classes of table parameters of BAPI methods. These classes offer functions to operate at the row-level, without functions to access individual fields of the rows. To access fields within any row, the classes of this category offer functions to access a proxy object of any given row, and that row proxy object is then used for accessing the individual fields.

See an example of a table parameter proxy class in the Java BAPI HTML Reference (in *Com.sap.bapi.purchasereqitem.jbobapiebantable.html*).

Construction

jbo<reference_structure_type>Table

The application program has no need to construct objects of this class

Operations

[appendRow \[Page 98\]](#)

Appends a row to the table parameter.

[createEmptyRow \[Page 99\]](#)

Returns a table row proxy object for an empty row.

[deleteAllRows \[Page 100\]](#)

Deletes all rows from the table parameter.

[deleteRow \[Page 101\]](#)

Deletes specified row.

[getRow \[Page 102\]](#)

Returns a table row proxy object for the specified row.

[getRowCount \[Page 103\]](#)

Returns the number of rows in the embedded table parameter.

[insertRow \[Page 104\]](#)

Inserts a row to the table parameter at the given indexed position.

[updateRow \[Page 105\]](#)

Updates an existing row in the table parameter at the given indexed position.

appendRow

appendRow

Purpose

Appends a row to the table parameter.

Syntax

```
void appendRow(jbo<RefStructType>TableRow tableRow);
```

Parameters

tableRow: a table row proxy object. This object is of type `jbo<RefStructType>TableRow` where `<RefStructType>` is the ABAP reference structure type defined in R/3 transaction SE11.

Return Value

None.

Exceptions

Throws `JRfcRemoteException`.

Related Information

[The Table Row Classes \[Page 106\]](#)

createEmptyRow

Purpose

Returns a table row proxy object for an empty row.

Syntax

```
jbo<RefStructType>TableRow createEmptyRow();
```

Parameters

None.

Return Value

A table row proxy object of type `jbo<RefStructType>TableRow` where `<RefStructType>` is the reference structure type.

Exceptions

Throw `JRfcRemoteException`.

Description

The created empty row that is returned to the caller is not appended to or inserted into the table parameter.

Related Information

[The Table Row Classes \[Page 106\]](#)

deleteAllRows

deleteAllRows

Purpose

Deletes all rows from the table parameter.

Syntax

```
void deleteAllRows();
```

Parameters

None.

Return Value

None.

Exceptions

Throw JRfcRemoteException.

deleteRow

Purpose

Deletes specified row.

Syntax

```
void deleteRow(int index);
```

Parameters

index: an integer index that specifies the row to be deleted.

Return Value

None.

Exceptions

Throws JRfcRemoteException.

Description

The index must point to a valid row.

Related Information

[The Table Parameter Classes \[Page 97\]](#)

[The Table Row Classes \[Page 106\]](#)

getRow

getRow

Purpose

Returns a table row proxy object for the specified row.

Syntax

```
jbo<RefStructType>TableRow getRow(int index);
```

Parameters

index: an integer index that specifies the row to be accessed.

Return Value

A table row proxy object of type `jbo<RefStructType>TableRow` where `<RefStructType>` is the reference structure type.

Exceptions

Throws `JRfcRemoteException`.

Description

The index must point to a valid row.

Related Information

[The Table Parameter Classes \[Page 97\]](#)

[The Table Row Classes \[Page 106\]](#)

getRowCount

Purpose

Returns the number of rows in the embedded table parameter.

Syntax

```
int getCount();
```

Parameters

None.

Return Value

An integer value representing the number of rows.

Exceptions

Throws JRfcRemoteException.

insertRow

insertRow

Purpose

Inserts a row to the table parameter at the given indexed position.

Syntax

```
void insertRow(int index, jbo<RefStructType>TableRow tableRow);
```

Parameters

index: an integer indicating the position where the new row is to be inserted.

tableRow: a table row proxy object. This object is of type `jbo<RefStructType>TableRow` where `<RefStructType>` is the ABAP reference structure type defined in R/3 transaction SE11.

Return Value

None.

Exceptions

Throw `JRfcRemoteException`.

Description

The index must be valid. That is, smaller than the total number of rows in the embedded table parameter.

Related Information

[The Table Parameter Classes \[Page 97\]](#)

[The Table Row Classes \[Page 106\]](#)

updateRow

Purpose

Updates an existing row in the table parameter at the given indexed position.

Syntax

```
void insertRow(int index, jbo<RefStructType>TableRow tableRow);
```

Parameters

index: an integer indicating the position where the new row is to be inserted.

tableRow: a table row proxy object. This object is of type `jbo<RefStructType>TableRow` where `<RefStructType>` is the ABAP reference structure type defined in R/3 transaction SE11.

Return Value

None.

Exceptions

Throw `JRfcRemoteException`.

Description

The index must be valid. That is, smaller than the total number of rows in the embedded table parameter.

Related Information

[The Table Row Classes \[Page 106\]](#)

The Table Row Classes

The table row classes work very much the same way as the structure parameter classes. Please refer to [The Structure Parameter Classes \[Page 96\]](#) for description of naming conventions of the field setter and getter functions for accessing the individual fields.

See an example of a table row proxy class for accessing individual fields of a table row in the Java BAPI HTML Reference (in *Com.sap.bapi.purchasereqitem.jbobapiebantablerow.html*).

ABAP to Native Data Type Mapping

ABAP Data Type	C++ Native Data Type
CHAR	java.lang.String
INT4	java.math.BigInteger
INT2	java.math.BigInteger
INT1	java.math.BigInteger
NUMC	java.math.BigInteger
PACK	java.math.BigDecimal
LANG	java.lang.String
CURR	java.math.BigDecimal
CUKY	java.lang.String
DATS	java.lang.String
UNIT	java.lang.String
TIMS	java.lang.String
DEC	java.math.BigDecimal
QUAN	java.math.BigDecimal
ACCP	java.lang.String
CLNT	java.lang.String
FLTP	java.math.BigDecimal
RAW	byte[]

The jboBase Class

This class is defined in *jboBase.java*.

This class is the base class from which all business object proxy classes are derived. This class encapsulates attributes and functionality that are common to all business object proxy classes, such as the object identification key, name, and type of the business object instances they represented by the proxy.

Construction

jboBase

The objects of this class are not intended to be directly instantiated by the application program.

Operations

[getKey \[Page 109\]](#)

Returns `jboKey` attribute for the business object proxy.

[setConnection \[Page 110\]](#)

Sets the `com.sap.rfc.IRfcConnection` object for this business object proxy.

[setKey \[Page 111\]](#)

sets the `jboKey` attribute to the `jboKey` value in argument.

getKey

Purpose

Returns the `jboKey` attribute for the business object proxy.

Syntax

```
jboKey getKey();
```

Parameters

None.

Return Value

The `jboKey` attribute that contains the identification key of this business object.

setConnection

setConnection

Purpose

Sets the `com.sap.rfc.IRfcConnection` object for this business object proxy.

Syntax

```
void setConnection(IRfcConnection connection);
```

Parameters

connection: the `IRfcConnection` object to be used for this business object proxy for communicating with its corresponding instance in R/3.

setKey

Purpose

sets the `jboKey` attribute to the the `jboKey` value in argument.

Syntax

```
void setKey(jboKey newObjectKey);
```

Parameters

`newObjectKey`: a `jboKey` object to replace the existing one in the business object proxy.

Return Value

None.

The jboKey Class

The jboKey Class

This class is defined and implemented in *jboKey.java*.

This class encapsulates the identification key used for identifying instances of R/3 business objects, and implements the logic for accessing the values of the keyfields.

Constructor

jboKey

Objects of this class are not intended to be instantiated by the application program

Operations

[getKeyfield \[Page 113\]](#)

Returns the value of the specified keyfield.

[getKeyfieldParameter \[Page 114\]](#)

Returns the `ISimple` interface to the embedded simple parameter object in the keyfield object specified by the keyfield name.

[setKeyfield\(with keyfield name and value\) \[Page 115\]](#)

Sets the keyfield specified by the keyfield name with the value specified.

[setKeyfield\(with ISimple interface\) \[Page 116\]](#)

Sets the embedded simple parameter object with the one whose `ISimple` interface is given.

getKeyfield

Purpose

Returns the value of the specified keyfield.

Syntax

```
String getKeyfield(String keyfieldName);
```

Parameters

keyfieldName: name of the keyfield whose value is of interest.

Return Value

A `String` object containing the value of the desired keyfield object.

getKeyfieldParameter

getKeyfieldParameter

Purpose

Returns the `ISimple` interface to the embedded simple parameter object in the keyfield object specified by the keyfield name.

Syntax

```
ISimple getKeyfieldParameter(String keyfieldName);
```

Parameters

keyfieldName: name of the keyfield whose value is of interest.

Return Value

A `com.sap.rfc.ISimple` interface to the desired keyfield object.

Description

This function returns the `com.sap.rfc.ISimple` interface to the keyfield object.

SetKeyfield (with keyfield name and value)

Purpose

Sets the keyfield specified by the keyfield name with the value specified.

Syntax

```
void setKeyfield(String keyfieldName, String keyfieldValue);
```

Parameters

keyfieldName: name of the keyfield whose value is of interest.

keyfieldValue: value to set to the specified keyfield.

Return Value

None.

SetKeyfield (with ISimple interface)

SetKeyfield (with ISimple interface)

Purpose

Sets the embedded simple parameter object with the one whose `ISimple` interface is given.

Syntax

```
void setKeyfield(ISimple keyfield);
```

Parameters

keyfield: a `com.sap.rfc.ISimple` interface to a simple parameter object that would be set as the embedded simple parameter object for this keyfield.

Return Value

None.

BAPI Beans

[Introduction \[Page 118\]](#)

[The BAPI Beans Information Classes \[Page 120\]](#)

Introduction

Introduction

This document is intended for Java programmers familiar with the concepts of JavaBeans and component software, and the use of the Core Reflection API.

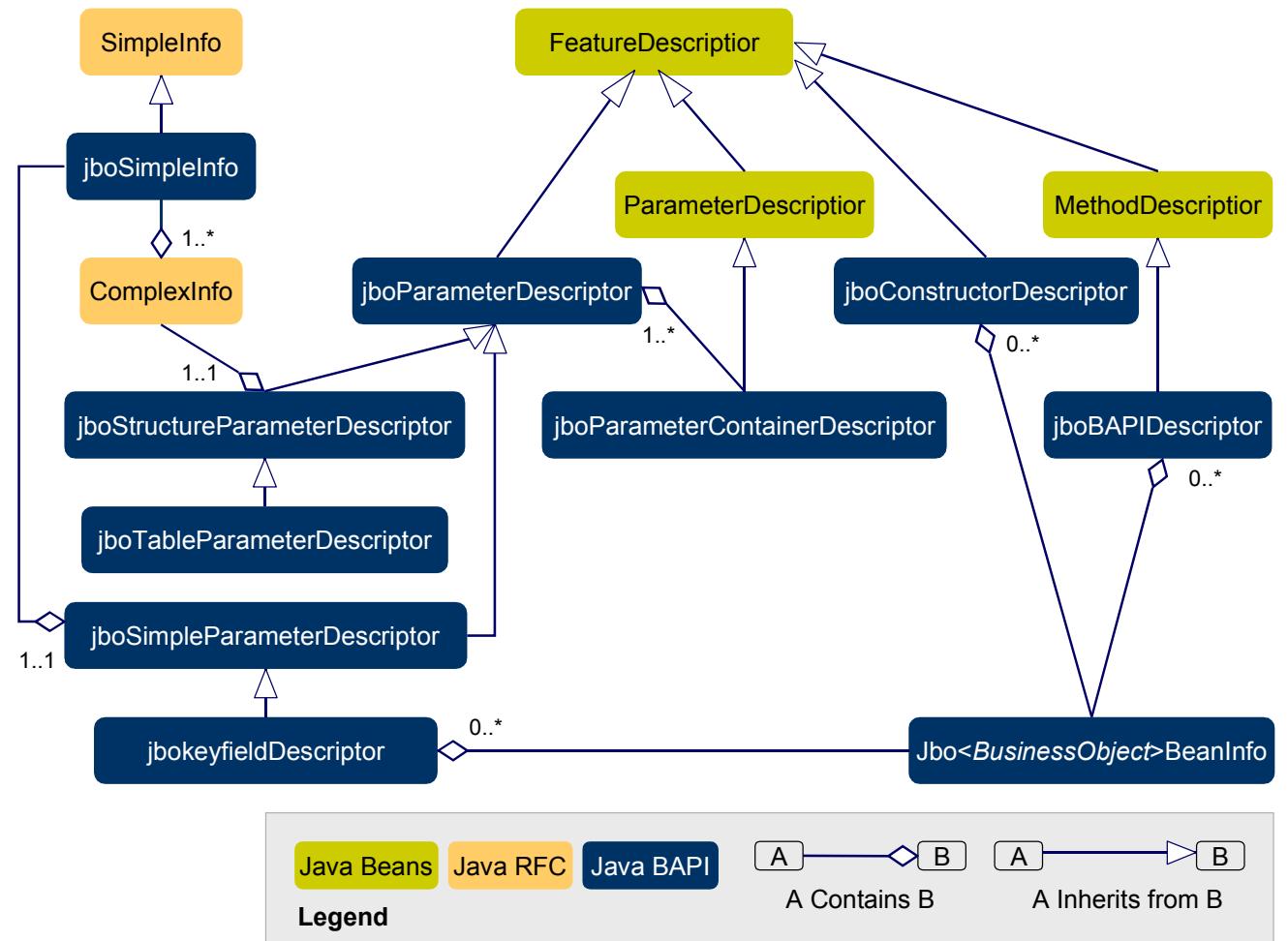
The Java BAPI Class Library consists of both proxy classes for the business object instances that exist in the R/3 database, and wrapper classes that help make the use of parameters easier when making BAPI calls. You can use the proxy classes of the Java BAPI Class Library (along with their associated parameter wrapper classes), as JavaBeans, ready-to-use components that form the building blocks of applications.

For example, you could place both the `jboSalesOrder` proxy class and the `jboSalesOrderBeanInfo` JavaBean information class, and all of the required subclasses into the JAR file, `jboSalesOrder.jar`, and use them in any JavaBean container application.

To fully understand how to use BAPI Beans and the associated descriptor classes, please see the sample programs in the Java BAPI Class Library in the SAP Automation kit.

The BAPI Beans Class Hierarchy

The BAPI Beans Class Hierarchy



The BAPI Beans Information Classes

Each BAPI Bean information class is based on a class in the Java Class Library. The Java Class Library implements Java so that each business object's proxy class has a corresponding JavaBeans information class. For example, the proxy class for the business object *SalesOrder* is *jboSales_Order*, and the corresponding JavaBeans information class is *jboSalesOrderBeanInfo* (the *jboSalesOrderBeanInfo* class is derived from the *com.sap.bapi.jboBeanInfo* class). The SAP R/3 System uses the information class as the gateway to obtain detailed information about a BAPI Bean, including its properties, methods, and features.

Unlike classes in the Java Class Library, BAPI Beans have no explicit properties. SAP has implemented BAPI Beans so that a key consisting of several keyfields identifies each business object instance, and the methods required are the BAPI methods themselves. SAP also has implemented factory methods, such as those responsible for creating new instances of business objects, as BAPI Beans. In addition, all of the simple, structure, import, export, and table parameters associated with BAPI methods are implemented as BAPI Bean features, and all of the constructors, keyfields, methods and parameters are described by JavaBean feature descriptor classes. These classes are described below:

[The *jboBAPIDescriptor* Class \[Page 121\]](#)

[The *jboParameterContainerDescriptor* Class \[Page 122\]](#)

[The *jboConstructorDescriptor* Class \[Page 123\]](#)

[The *jboParameterDescriptor* Class \[Page 124\]](#)

[The *jboSimpleParameterDescriptor* Class \[Page 125\]](#)

[The *jboStructureParameterDescriptor* Class \[Page 126\]](#)

[The *jboTableParameterDescriptor* Class \[Page 127\]](#)

[The *jboKeyfieldDescriptor* Class \[Page 128\]](#)

Each member of these classes offers the following methods:

- `getBAPIDescriptors`
- `getBeanDescriptor`
- `GetConstructorDescriptors`
- `GetKeyfieldDescriptors`
- `GetMethodDescriptors`

For more information on these methods, please see the Java BAPI HTML Reference.

The jboBAPIDescriptor Class

The `jboBAPIDescriptor` class is derived from the `java.lang.reflect.MethodDescriptor` class and offers application programs access to all the information available for the selected method. Each object belonging to the `jboBAPIDescriptor` class describes a BAPI Bean method, and each of these methods encapsulates a separate BAPI call. An application program can invoke a method either separately or by using the `java.lang.reflect.MethodDescriptor` superclass.

The `jboBAPIDescriptor` class offers the following methods:

- `getMethodType`
- `getParameterContainerDescriptor`
- `isClassMethod`
- `isInstanceMethod`

For more information on these methods, please see the Java BAPI HTML Reference.

The `jboParameterContainerDescriptor` Class

The `jboParameterContainerDescriptor` class is derived from `java.beans.ParameterDescriptor`. Each object in the `jboParameterContainerDescriptor` class describes a container class for parameters. Each of these container classes is, in turn, associated with a BAPI method and a BAPI Bean constructor. An application program can use the `jboParameterContainerDescriptor` class to obtain a list of objects that describes each parameter needed for making BAPI calls. This class also offers application programs the ability to set parameter values indirectly via the Core Reflection API.

The `jboParameterContainerDescriptor` class offers the following methods: `getBAPIParameters` and `getParameterContainerClass`. For more information on these methods, please see the Java BAPI HTML Reference.

The jboConstructorDescriptor Class

The jboConstructorDescriptor Class

BAPI Beans constructors are considered to be features; therefore, the `jboConstructorDescriptor` class is derived from `java.beans.FeatureDescriptor`. An application program can use objects belonging to this class to retrieve information about BAPI Bean constructors. In addition, an application program can use the Core Reflection API to invoke the constructors to fill automatically the BAPI parameters needed to construct BAPI Beans.

This class offers the following methods:

- `getConstructor`
- `getParameterContainerDescriptor`
- `getParameterDescriptors`

For more information on these methods, please see the Java BAPI HTML Reference.

The `jboParameterDescriptor` Class

`jboParameterDescriptor` is the superclass of all other BAPI Beans parameter descriptor classes. Since SAP considers BAPI parameters to be BAPI Bean features, this class is derived from `java.beans.FeatureDescriptor`. Applications cannot directly use objects belonging to this class: they can only use the derived parameter descriptor classes identified elsewhere in this document. However, an application program can use the methods in this class to find the type of the parameter object being described. In addition, a program can use the Core Reflection API with this superclass to set values automatically to these parameters.

The `jboParameterDescriptor` class offers the following methods:

- `getParameterClass`
- `getParameterType`
- `isSimple`
- `isStructure`
- `isTable`

For more information on these methods, please see the Java BAPI HTML Reference.

The jboSimpleParameterDescriptor Class

The `jboSimpleParameterDescriptor` class describes simple parameter objects, which are derived from `com.sap.bapi.jboParameterDescriptor`. Application programs can use this class to retrieve information about the described simple parameter (contained in `com.sap.bapi.jboSimpleInfo` class), and can set values to this parameter using the Core Reflection API.

This class offers one method: `getSimpleInfo`. For more information on this method, please see the Java BAPI HTML Reference.

The `jboStructureParameterDescriptor` Class

The `jboStructureParameterDescriptor` class describes structure parameter objects. This class is derived from `com.sap.bapi.jboParameterDescriptor`. Application programs can use this class to retrieve information about a structure parameter and the fields defined in the `com.sap.rfc.ComplexInfo` class. In addition, programs can use this class to set values to the specified structure parameter using the Core Reflection API.

This class offers these two methods: `getComplexInfo` and `getReferenceStructureType`. For more information on these methods, please see the Java BAPI HTML Reference.

The jboTableParameterDescriptor Class

The `jboTableParameterDescriptor` class describes table parameter objects. This class is derived from `com.sap.bapi.jboStructureParameterDescriptor`. Application programs can use this class to retrieve information about a table parameter and all of the fields contained in `com.sap.rfc.ComplexInfo` class. In addition, applications can use the `jboTableParameterDescriptor` class to set values to a described table parameter by using the Core Reflection API.

This class offers no methods.

The `jboKeyfieldDescriptor` Class

All items belonging to the `jboKeyfieldDescriptor` class describe keyfield objects. This class is derived from the `com.sap.bapi.jboSimpleParameterDescriptor`. Application programs can use the `jboKeyfieldDescriptor` class to retrieve information about any keyfield described in the `com.sap.bapi.jboSimpleInfo` class. Applications can then use this information to set values to the keyfield by using the Core Reflection API.

Tracing Errors When Running the SAP Assistant

Activating a Log File

Use

If you encounter errors when running the SAP Assistant, you can activate the Trace option, which then produces a log file if errors occur.

Since the SAP Assistant is using several SAP Automation components, it is helpful to the SAP Assistant development team to see the resulting log file for tracing the error to the component producing the error.

Procedure

1. Choose *View* → *Options*.
2. Choose the *Trace* tab.
3. Choose between the Log Errors Only and the Log Errors and Messages. Also specify the name of the log file and its path.

Result

If errors occur, report them through SAPNet, and attach the log file produced by the trace tool.

Checking Version and Location of Underlying Components and DLLs

Use

In addition to activating the log file, you can verify that the major underlying components and DLLs for the SAP Assistant exist, and that they are of the correct version.

Procedure

1. Choose *Help* → *About*.
2. View the list of components and DLLs in the edit box in the *About SAP Assistant* dialog.

The list includes the DLL or component name, followed by a colon, followed the full path to the component or DLL file, then followed by a version number.
If the line for the component (or the DLL) does not include those details, you may be missing the component. Try re-installing the SAP Assistant.
3. Also copy the text within the text box and include it in your error report in SAPNet.

SAP Automation ActiveX (OCX) Controls

SAP Automation ActiveX (OCX) Controls

The SAP Automation ActiveX controls discussed here use Remote Function Call (RFC) to execute calls to the R/3 System. These controls provide objects that allow the programmer to manage function calls from desktop applications for working with RFC functions, R/3 transactions, and R/3 table data.

Summary of ActiveX Controls

Summary of ActiveX Controls

The SAP Automation OCX controls run on Windows 95 and Windows NT.

The following table describes some of the SAP Automation OCX controls that are documented here. Follow the link to see the detailed documentation of each control.

Tool	Function
Logon Control [Page 462]	<p>Creates a connection object that enables COM-compliant programs to log onto R/3.</p> <p>The Logon method of the connection object establishes a connection to R/3. It provides the option to suppress the display of the standard SAP Logon dialog box to the user, so that you could provide your own logon dialog (for example, if you wish to specify the logon system or account programmatically.)</p>
Function Control [Page 155].	Provides objects (functions collection, function, logon, tables, table, exports, export, imports, import, parameter types, and so on) to call function modules (RFCs) on an R/3 System.
Table Factory Control [Page 300] (Also called Table Control)	Works with the Function control to manage tables attached to Function objects. In addition, the Table Factory encapsulates Table objects for easier access by the client application.
Table View Control [Page 389].	Allows on screen viewing of an internal table (see Table Control) from R/3's RFC library in a spreadsheet format.
Table Tree Control [Page 216]	Allows displaying and management of tables (see Table Control) that contain hierarchically structured data (trees of parent nodes and their children) visible. Table Trees allow the programmer and user to manage tables containing directory trees.
Transaction Component (Also called Transaction Control [Page 191])	<p>Provides screen and field object management, so a user can remotely call R/3 transactions or use them in programs.</p> <p>Exposes R/3 batch input capability (BDC) to COM-compliant programs and applications. This means that the external program can send input field values to an R/3 screen, but output field values are not returned.</p> <p>Makes using batch input easier to COM-compliant programs by eliminating the need to populate the fields of the BDC table, which is a prerequisite to using the standard R/3 batch input method.</p>

Summary of ActiveX Controls

BAPI ActiveX Control [Ext.]	<p>Allows you to develop external client/server applications (with R/3 as a server) that access business functions in the SAP System by calling BAPIs (Business APIs) through OLE Automation.</p> <p>Achieves this by allowing you to create (on the client) local instances of business objects, which act as proxy objects for the business object in the SAP system.</p> <p>The meta information required from the R/3 System is retrieved dynamically at runtime.</p>
---	---

SAP DCOM Connector-based Components

The following SAP Automation components use the SAP DCOM Connector:

Tool	Function
DCOM Connector Logon Component [Ext.]	<p>Helps programs using the SAP DCOM Connector in handling the connection parameters of COM objects created by the DCOM Connector.</p> <p>The DCOM Connector Logon Component provides a Logon dialog with which you can get the necessary connection parameters from an end user.</p> <p>The DCOM Connector Logon Component also allows you to easily copy connection parameters into a DCOM Connector COM object.</p>
Repository Services [Ext.]	<p>Provides read access to the metadata of business objects and RFC function modules in an R/3 system to COM-compliant programs and applications.</p> <p>Also allows you to save a copy of the metadata in a local database and then access the data offline.</p>
Repository Browser (Also called the SAP Browser Control [Page 507])	<p>A control that can be hosted by any ActiveX container. It consists of a window with two panes for browsing SAP BAPI and RFC metadata information. (See the SAP Assistant screen [Page 25]: it uses the SAP Browser control)</p> <p>Allows online calling of RFC functions from within the control.</p> <p>Also exposes several methods to enable the container application to control and automate metadata browsing.</p> <p>Allows you to export properly formatted metadata information to MS Excel.</p>
BAPI Gateway [Ext.]	<p>Allows you to dynamically call BAPIs and RFCs through the DCOM Connector, that is, it allows you to determine at run time which BAPIs or RFCs you call.</p> <p>Eliminates the need to use the SAP DCOM Connector wizard for creating the BAPI component.</p> <p>Use the SAP Automation Repository Services component [Ext.] to obtain the metadata for the BAPIs or RFCs you wish to call at run time.</p>

Relationship Between the Controls and SAP Assistant

Older versions of SAP Assistant have used several of the ActiveX controls described in the first table above.

Summary of ActiveX Controls**SAP Assistant Uses the SAP DCOM Connector**

The SAP Assistant product currently uses the SAP DCOM Connector, and it therefore now uses the above DCOM Connector-based components.

The SAP Assistant product uses the Repository Services component to read metadata of RFCS and BAPIs from the SAP System. It uses the SAP Browser control to display the BAPIs and RFCS and their details. Both the Repository Services component and the SAP Browser Control are now using the SAP DCOM Connector.

The SAP Assistant uses the BAPI Gateway and the DCOM Connector Logon Component in place of the following controls:

DCOM-Compatible Component (Currently Used by SAP Assistant)	Replaces ActiveX Control(s) In SAP Assistant
BAPI Gateway [Ext.]	Function Control, BAPI ActiveX Control
DCOM Connector Logon Component [Ext.]	Logon Control

If you are developing new applications that use DCOM and the SAP DCOM Connector, we recommend that you, too, use these components.

Possible Uses for OCX Controls**Possible Uses for OCX Controls**

The following table lists some possible applications of OCX controls.

Program	Possible Use	Server	Client
MS Excel 5	Upload planning data, download report data.	✓	✓
MS Project 4	Control purchasing schedules.	✓	✓
Visio	View business process flows.	✓	✓
MS Word 6	Use spellcheck.	✓	
Powerbuilder MS Access 2 Borland Delphi	Build database front-ends, use programming language of choice.		✓ ✓ ✓

Using the SAP Automation ActiveX Controls

Summary of Programming Tasks

Summary of Programming Tasks

Your program's overall goal is to call a function in the R/3 System, sending data as input to the function and receiving data as return values. In the Function Control, sending and return parameters are presented as further objects contained in the Function object.

Any application using OLE Automation must perform the programming tasks listed below. You should perform the first three steps for both methods of remote call access:

1. Create the base-object - the component itself.
2. Supply connection and logon information.
3. Open a connection to the R/3 System and log the user on.

If you are making **non-dynamic** remote function calls, continue as follows:

4. Request a Function object for the R/3 function you want.
5. Set the export and table parameter values.
6. Make the remote call.
7. Get the return values from the import and table parameters.

If you are making **dynamic** remote function calls, do not add the function to the functions collection, but write the function call like a native (local) VB function:

4. Create table or structure objects to place data into to pass data with the table parameters or structure parameters.
5. Invoke the function from the functions collection object.
6. Pass parameters to the named argument such as:

`Xfunc (X1: = 5, X2: = nVar, X3: = objVar)`

where X_1 , X_2 , and X_3 are the argument names in the function interface.

Parameters can be either simple variables or object variables.

The following sections show how to program these tasks.

[Example Application with the Function Control \[Page 137\]](#)

[Variation using the Dynamic Calling Convention \[Page 139\]](#)

Example Application with the Function Control

Example Application with the Function Control

This example application demonstrates almost all the objects discussed and several of their properties and methods. The application gets a list of customers from the R/3 System and prints attributes for each customer (for example, name and ZIP code). The function interface is:

RFC_CUSTOMER_GET	IMPORT		Name (NAME1)		
			IMPORT		<i>Customer-Number (KUNNR)</i>
			TABLES		<i>CUSTOMER_T (RFCKNA1)</i>

This function uses selection criteria (name and customer number) to retrieves a set of customers, as in the SQL query:

```
select * from CUSTOMER_T where NAME1=Name
and KUNNR=Customer Number
```

The table has the following structure (all fields are of type RFC_CHAR):

Customer Table Structure:

- KUNNR
- ANRED
- NAME1
- PFACH
- STRAS
- PSTLZ
- ORT01
- TELF1
- TELFX

The following example accesses the remote function by adding Function objects to the Functions collection object:

Declare object variables:

```
Dim Functions as Object
Dim GetCustomers as Object
Dim Customers as Object
```

Create the Function control (that is, the high-level Functions collection):

```
Set Functions = CreateObject ("SAP.Functions")
```

Indicate what R/3 System you want to log on to:

```
Functions.Connection.Destination = "B20"
```

Set the rest of Connection object values:

Example Application with the Function Control

.....

Log on to the R/3 System:

```
Functions.Connection.Logon  
if Functions.Connection.Logon (0, True) <> True then  
    MsgBox "Cannot logon!"  
End If
```

Retrieve the Function object (the Connection object must be set up before Function objects can be created):

```
Set GetCustomers = Functions.Add("RFC_CUSTOMER_GET")
```

Set the export parameters (here, get all customers whose names start with J):

```
GetCustomers.Exports("NAME1") = "J*"  
GetCustomers.Exports("KUNNR") = "*"
```

Call the function (if the result is false, then display a message):

```
If GetCustomers.Call = True then
```

There are two ways of accessing the table:

```
Set Customers = GetCustomers.Tables(1)  
Set Customers = GetCustomers.Tables("Customers")  
Print Customers (Customers.RowCount, "KUNNR")  
Print Customers (Customers.RowCount, "NAME1")
```

```
Else
```

```
    MsgBox "Call Failed! error: " + GetCustomers.Exception
```

```
End If
```

```
Functions.Connection.Logoff
```

A variation of this code can be used for dynamic calls. See [Variation using the Dynamic Calling Convention \[Page 139\]](#).

Variation using the Dynamic Calling Convention

Variation using the Dynamic Calling Convention

Although dynamic calling looks different from non-dynamic calling, it performs the same function. The code is the same as that shown in [Example Application with the Function Control \[Page 137\]](#) until the logon process is complete. Then, you call the function with parameter “NAME1”:

```
Functions.RFC_CUSTOMER_GET (Exception, NAME1:="J*", KUNNR:="*",  
CUSTOMER_T:=Customers)
```

The customers table retrieved by the function is copied into the Customers variable. This technique makes the code a little easier to read, but takes slightly more time to process.

Creating the Base-level Control

Controls are usually made up of collections of objects. For example, the Function control contains Function objects, and the Transaction control contains Transaction objects. The highest level of a control - the base-level control - is either an actual collection object (like the Functions collection object) or a single control object (like the Table Factory object).

Controls also maintain some properties or objects common to all their sub-objects. Examples are the R/3 Connection (an object shared by all Functions or Transactions in a collection) or the Count property (the number of objects in a collection).

To create the base-level control, you use the *CreateObject* function and a fixed request string. This string specifies the kind of control (i.e. "SAP.TableFactory.1") and causes creation of the relevant base-level object. For example:

```
transactionsOCX = CreateObject("SAP.Transactions")
```

For the SAP OCX controls, the fixed strings are:

CreateObject String

OCX control	Highest (base-level) object	Fixed String
Function control	Functions collection object	"SAP.Functions"
Transaction control	Transactions collection object	"SAP.Transactions"
Logon control	Logon object	"SAP.LogonControl"
Table Factory control	Table Factory object	"SAP.TableFactory"
Table View control	Table View object	"SAP.TableViewControl"
Table Tree control	Table Tree object	"SAP.TableTreeControl"

For more information, see [SAP Control Base Classes \[Page 144\]](#).

Connecting to an R/3 System

- For applications using the SAP DCOM Connector, use the SAP Automation DCOM Connector Logon Component to handle the connection to R/3 systems. Using this method requires only that you fill out the logon properties of the COM object. You do not need to actively establish a connection to R/3. The connection is established automatically whenever needed based on the logon properties of the COM object.

To learn more on how to obtain the data for the logon properties, see the help for the [SAP Automation DCOM Connector Logon Component \[Ext.\]](#).
- COM-compliant programs who do not use the SAP DCOM Connector can use the Logon Control, documented here. Note that the Logon Control is an older component in the [SAP Automation suite of products \[Ext.\]](#).

Performance and Debugging Tips

Performance and Debugging Tips

The following sections describe techniques you can use to improve the performance and reduce debugging time for your applications.

Avoiding Unnecessary Object Creation

Application objects and all collection objects are created dynamically when your code makes a reference to one of them. These objects are temporary and are destroyed when no more references to them exist. As a result, multiple accesses to one of these objects can affect performance. To avoid this problem, assign the object to an object variable of your own and then make the accesses.



You can improve the code

```
GetFunct.Exports("P1")
GetFunct.Exports("P2")
GetFunct.Exports("P3")
GetFunct.Exports("P4")
GetFunct.Exports("P5")
```

by changing it to

```
set MyExports = GetFunct.Exports
MyExports("P1")
MyExports("P2")
MyExports("P3")
MyExports("P4")
MyExports("P5")
```

In the first section of code, five temporary collection objects are created, and each is destroyed after a single statement. In the second section, only one temporary object is created. (This object is not destroyed after the first statement, because the other statements still refer to it.)

Tracing RFC Calls

You can request a trace of connection activity as the RFC call executes. The TraceLevel property in the Connection object lets you specify tracing. Possible values are 0 (tracing not requested) and 1 (tracing requested).

The activity information is logged in a file "RFC<random number>.TRC" located in the active default directory.

The Functions collection and Transactions collection objects also provide logging functionality with the LogFileName and LogLevel properties.

Note on Embedded Property Calls

When coding your application, you should be aware that client languages may execute certain kinds of statements differently. Of particular importance is whether or not your language performs cascaded evaluation in the form of statements like:

```
MyFunct.Exports("P1").Value("F1")
```

This statement requires the language to first call the Exports property for MyFunct. When a Structure object is returned, you call the Value property on the Structure object.

Some languages do not perform full evaluation of statements. They evaluate the first call (the Exports property), but do not evaluate the returned value (a Parameter or Structure object) to call the next property (Value method) on it. As a result, you get the wrong object returned, and eventually a runtime error.

The Excel macro language executes the above statement correctly, but Visual Basic 3.0 does not. However, almost all interpreters fail to evaluate statements correctly when a default function is left implicit:

```
GetFunct.Exports("P1")("F1")
```

Note on Default Property Calls

Some languages do not evaluate the default value property. In this case, the default value property must be explicitly specified:

```
MyFunct.Exports("P1")      ' Does not work.'
```

```
MyFunct.Exports.Item ("P1") ' OK'
```

SAP Control Base Classes

SAP Control Base Classes

SAP Standard Collection

All active OLE controls containing collections within an SAP system are implemented according to common conventions. As long as a collection is not explicitly declared as a non-standard collection, the following description applies to the collection. However, not all properties and methods mentioned below are available with every collection.

The lower bound index for all collections is 1.

For hints on using collection objects, see [Using Collection Objects \[Page 148\]](#).

Standard Collection Properties

Name	Parameter	Type	Description
Count	void	Long	Returns the number of objects stored in the collection.
Item	Variant valIndex	Object	Returns an object according to <i>valIndex</i> . Item is always the default property.

Standard Collection Methods

Name	Parameter	Return Type	Description
Add	Object dependent	Object	Adds a new object and returns the new object.
Insert	Variant valIndex Object dependent	Object	Inserts a new object at position <i>valIndex</i> and returns the new object.
Remove	Variant valIndex	Boolean	Removes the object at position <i>valIndex</i> .
RemoveAll	void	Boolean	Removes all objects from the collection.
UnLoad	Variant valIndex	Object	Unloads the object at position <i>valIndex</i> .

Detailed Description

Object Item(Variant valIndex)

The item method returns an object from the collection. The parameter *valIndex* identifies the position of the object to be returned. The type of this parameter depends on the object. It may describe the position where the object can be found, either as a simple integer value, or as a string value (as described in [Named Collections \[Page 147\]](#)), or in any object-dependent variant data type. In the following sections, the valid types are described for each collection.

Object Add (...)

The parameters for the Add method depend on the object. These parameters are used to initialize the new object. Add always returns the new object.

SAP Standard Collection

Object Insert(Variant valIndex,...)

The parameters for the Insert method depend on the object. These parameters are used to initialize the new object. Insert always returns the new object. The first parameter of the Insert methods always describes the position where to insert the new object (the new object is always inserted in front of the position described by valIndex). The type of this parameter is object-dependent. It may describe the position where to insert the new object, either as a simple integer value, or as a string value (as described in [Named Collections \[Page 147\]](#)), or as an Object which is already part of the collection. Nevertheless, the indexing parameter always has the same meaning as the indexing parameter of the default property [Item \[Page 287\]](#).

Boolean Remove(Variant valIndex)

This method removes an object from its collection. The parameter valIndex identifies the position of the object to be returned. The type of this parameter depends on the object. It may describe the position where to insert the new object, either as a simple integer value, or as a string value (as described in [Named Collections \[Page 147\]](#)). Nevertheless, the indexing parameter always has the same meaning as the indexing parameter of the default property [Item \[Page 287\]](#).



When removing an object from the collection, the object becomes **invalid**. Any further attempts to work on the object return an Invalid Object Exception. Use UnLoad if the object should be removed from the collection for further use.

Object UnLoad(Variant valIndex)

This method unloads an object from its collection. The parameter valIndex identifies the position of the object to be returned. The type of this parameter depends on the object. It may describe the position where to unload the object, either as a simple integer value, or as a string value (as described in [Named Collections \[Page 147\]](#)). Nevertheless, the indexing parameter always has the same meaning as the indexing parameter of the default property [Item \[Page 287\]](#).

SAP Named Collection

Named collections are derived from [SAP Standard Collections \[Page 145\]](#) and may always work with strings as indexing parameters for methods like Item, Insert, Remove and UnLoad. Objects within a named collection always have a Name property which stores the indexing name. The name describing an object in a named collection does not have to be unique. If a name is used frequently, Item, Insert, Remove and UnLoad always use the first object with the given name.

Further features of named collections are dynamic properties, created as a result of the objects' names. Instead of invoking the Item property, an object may also be returned if the name of the object is used as property.



```
Dim oObj as Object  
' Add a new empty object  
Set oObj = NamedCollectionObject.Add ()  
' Assign name  
oObj.Name = "ItemB"  
' Add object and pass name as parameter.  
Set oObj = NamedCollectionObject.Add ("ItemC")  
' Insert object in prior to object "ItemB"  
Set oObj = NamedCollectionObject.Insert ("ItemB", "ItemA")  
  
' Accessing the object  
' Retrieve second object through index  
Set oObj = NamedCollectionObject.Item(2)  
' Retrieve second object through name  
Set oObj = NamedCollectionObject.Item("ItemB")  
' Retrieve second object through dynamic property  
Set oObj = NamedCollectionObject.ItemB
```

For more information about collection objects, see [Using Collection Objects \[Page 148\]](#).

Using Collection Objects

Using Collection Objects

The SAP Assistant defines several *collection* object types. Collection objects gather all objects of a given type into a list. For example, all Function objects for a given control are gathered together in the Functions collection object representing that control.

A collection object provides list-oriented functions for accessing list objects, adding and removing from the list, looping through the list, and so on.



Loop through a Rows collection object:

```
For Each Customer in Customers.Rows  
    print Customer ("NAME")  
Next Customer
```

SAP Data Object

The SAP Data Object is a special object for purposes of data transport. This object is used in [drag and drop \[Page 274\]](#) and clipboard operations. The SAP Data Object implements an automation and an IDataObject interface. The IDataObject interface is a standard OLE interface for data object manipulation. The automation interface displays the following methods:

Name	Parameters	Return Type	Description	
GetData	Long Variant	cfFormat vaData	void	Retrieves data from the data object in the specified format
SetData	Long Variant	cfFormat vaData	void	Stores data in the data object in the specified format
IsFormatAvailable	Long	cfFormat	Boolean	Returns TRUE if the data object contains data in the specified format



SetData may specify a format which is not initially cached in the data object. Since the data object does not interpret the data in any way, the data may be of any clipboard format. The variant data type must be any data type which is transportable to other processes. Therefore, object may not be stored in the data object.

Valid formats are: char, short, long, String, Date, Time, Boolean and safe arrays of these types.



```

Sub DragSourceFill(DataObject As Object)
    Dim cfFormat As Long
    Dim Data As String

    cfFormat = RegisterClipboardFormat("MyClipFormat");
    Data = "This is my personal string"
    DataObject.SetData(cfFormat,Data)

Sub End

Sub Drop(DataObject As Object)
    Dim cfFormat As Long
    Dim Data As String

```

SAP Data Object

```
cfFormat = RegisterClipboardFormat("MyClipFormat");  
if DataObject.IsFormatAvailable(cfFormat) then  
    DataObject.GetData(cfFormat,Data)  
    MsgBox(Data)  
  
end if  
Sub End
```

Safe Arrays and Values

If an SAP object returns data as a safe array, the object always has the property Data. This property usually returns a two-dimensional safe array with a lower bound index of 1 for each dimension. An example of a Data property would be the entire content of a table or the entire content of a row or column in a table. Single date like the content of a cell in a table is always returned as a Variant. The corresponding property is always called Value. If an object implements a Value property, this property is the default property.

Font Objects

Font Objects

Font Properties:

Table Caption

Name	Type
Name	String
Size	Currency
Bold	Boolean
Italic	Boolean
Underline	Boolean
StrikeThrough	Boolean
Weight	Short
CharSet	Short

For more information, see the VBA help on font objects.

Data Types

The following are the available data types:

Table Caption

Types in Help File	VBA Data Type	C++ Data Type
Char	Byte (By Val)	unsigned char
Short	Integer (By Val)	short
Long	Long (By Val)	long
String	String (By Val)	BSTR
Boolean	Bool (By Val)	VT_BOOL
Object	Object	IDispatch *
Short*	Integer	short*
Long*	Long	long*
String*	String	BSTR*
Boolean*	Bool	VT_BOOL*
void	Method does not return a value	void
Array of type	(1, n) of <i>Type</i>	VT_ARRAY VT_ <i>Type</i>

SAP Automation ActiveX Controls

SAP Automation ActiveX Controls

The Function Control

The Function control is an OCX control that makes remote function calls to the R/3 System. It handles parameters comfortably, makes the calling of functions easy and passes results back to the OLE client (VB, VBA, and others) rapidly.

Introduction

[Introduction \[Page 156\]](#)

[Function Control Object Hierarchy \[Page 157\]](#)

Control and Object Reference

[Function Control \[Page 158\]](#)

[Functions Collection Object \[Page 167\]](#)

[Function Object \[Page 173\]](#)

[Exports Collection Object \[Page 178\]](#)

[Imports Collection Object \[Page 181\]](#)

[Structure Object \[Page 184\]](#)

[Parameter Object \[Page 188\]](#)

Introduction

Introduction

In the R/3 System, a function is a unit of ABAP code, including any associated tables, export parameters, or import parameters. The *Function control* is an OCX control that makes remote calls to deliver R/3 System functionality to external programs.

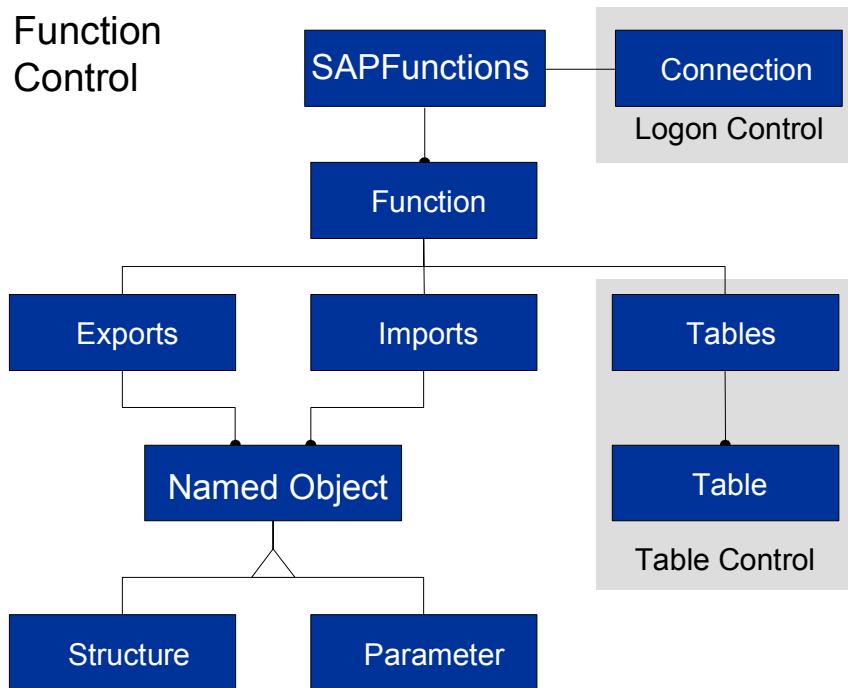
If you want to call a function in the R/3 System, you use the Function control to set up the necessary environment. The Function control acts as a container for a Function collection object and its Function objects. Therefore, before you create other types of objects, first create a Functions collection object.

The Function control retrieves information about the function to be called from the R/3 System. You then set parameter values for the function using normal OLE automation techniques.

When it is time to call the function, the control assembles all the data needed by the RFC library and initiates the call. After the call, return parameters are unpacked and returned to the calling program.

Function Control Object Hierarchy

The object hierarchy for the Function control is as follows:



Function Control

To use a Function control, you must:

- create the Function control

To do this, call `CreateObject("SAP.Functions")`. A Functions collection object is created automatically.

- establish a connection to an R/3 System

To establish a connection, you must create a Connection object. You can do this either directly from the Functions collection object, or using a Logon control. Once the Connection object is created, you can use it to logon to the R/3 System.

- add a Function object to the list (or call the function immediately)

Using the Function Control

After creating the Function control, you have to tell it about its connection to the R/3 System. Then, you can start adding Function objects and calling RFCs. The Function control base-class is the Functions collection object. You add and remove functions by using the *Add* and *Remove* methods.

Once a function has been added, you can assign parameter values and call it. The *Call* method of the Function object returns a Boolean value that tells you whether the call executed with no problems. Use the *Exception* property of the Function object to get more information on any errors. If you have problems calling RFCs, consult the *LogFileName* and *LogLevel* properties of the Function control's base-class (the Functions collection object). These properties provide more information for trouble-shooting.

The following code creates the Function object, sets a connection, sets parameters, and calls the function.

```
' Create the component.  
Dim functions As Object  
  
Set functions = CreateObject ("SAP.Functions")  
  
' Set the connection.  
  
Set functions.Connection = conn  
  
' Add (retrieve) function from R/3.  
func = Functions.Add ("RFC_CUSTOMER_GET")  
  
' Set a parameter.  
  
func.Exports ("NAME1") = "ACME Steel"  
  
' Call the function.  
  
If Func.Call <> True then  
    MsgBox "Call Failed. Exception " + func.exception  
End If
```

For more information, see:

[Requesting Functions \[Page 160\]](#)

[Adding a Function \[Page 161\]](#)

[Setting Parameter Values \[Page 162\]](#)

[Viewing Table Objects \[Page 163\]](#)

[Using Parameter and Structure Objects \[Page 164\]](#)

[Using Named Argument Calling Conventions \[Page 165\]](#)

Requesting Functions

Requesting Functions

You must get a Functions collection object in order to access all other objects. Each Functions collection contains one Connection object. If your application needs to access data from multiple R/3 Systems, you must create a separate Function control for each system. Each of these components has its own Connection object.

Adding a Function

Both ABAP function modules and their separate parameters are represented by the object types Function, Parameter, Structure, and Table. You must get the Function object explicitly in order to be able to access the other object types. The Function control adds each function requested to the Functions collection.

Setting Parameter Values

Setting Parameter Values

Before calling an RFC function, you must set export and table parameter values. Depending on the ABAP function definition, you may not have to use all the formal variables specified in the interface. If you don't specify a value for a parameter, it is not sent with the call.

- To set values for import/export parameters defined as simple fields, use the Parameter object and its property functions.
- For import/export parameters defined as structures, use the Structure object and its properties to access individual fields.
- To set values for table parameters, use the Table, Rows and Row objects (and their properties) to access individual table rows and fields.

If the control cannot convert between ABAP data types and your variable data types, it sends an error message as soon as you fill the parameter.

For information about handling export and import parameters, see [Using Parameter and Structure Objects \[Page 164\]](#).

For information about handling table parameters, see [Viewing Table Objects \[Page 163\]](#).

Handling parameter objects

RFC objects “depend” on the parent object that contains them. If you assign the object to an object variable, the variable shares the object with the containing parent object.



After the statement

```
Set ObjVar = FunctionsOCX ("RFC_PING")
```

`ObjVar` and `FunctionsOCX ("RFC_PING")` point to the same object. This is important because if you remove the object (or its parent) from the relevant collection, you invalidate the contents of the object variable.

In this example, if you remove `Funct1` from its Functions collection, the variable `ObjVar` becomes invalid.

There are two exceptions to this rule:

- Table objects that have been detached (“unloaded”) from their parent Function objects, making the Table control the Table’s new parent. For more information, see [Viewing Table Objects \[Page 163\]](#).
- Collection objects that have been assigned to object variables.

Both of these become independent of the containing Function object. By contrast, you cannot detach Export/Import objects in this way. In addition, Table objects that are simply assigned (rather than unloaded) to a variable also remain shared.

Table and Structure objects are always independent of the Function objects when a remote function is called directly because there is no Function object created externally.

Viewing Table Objects

When setting values for table parameters:

- To access fields in a row, use the Table.Item and Row.Value properties.
- To process table rows as units, use the Table.Rows property to loop through all rows, or the Table.Item property for direct access to a single row.

You use the Row object to retrieve column information in the Table object. Do not attempt to use the Structure object methods on table rows; the Structure object is only for use with objects from the Exports and Imports collections.



If you assign a row to a object variable, and then delete either the Table object from the Tables collection object, or the row from the table (using the any of the RemoveRow, DeleteTable, or FreeTable methods), you invalidate the contents of the target object variable.

To avoid this problem, detach Table objects from the Function object. The table component provides the methods to unload and set for detaching and reattaching tables. A Table object unloaded to an object variable is no longer shared with the Function object. However, Table objects that are simply assigned (rather than unloaded) to a variable remain shared.

Using Parameter and Structure Objects

If you want to access fields in an export/import parameter defined as a structure, use the following:

1. Assign the parameter to an object variable:

```
set StructObj = MyFunct.Exports("Employee_Struct")    OR  
Set FieldObj   = MyFunct.Exports("Company_Number")
```

2. Use either Parameter or Structure functions on the variable, depending on whether the parameter is defined as a field or structure:

```
StructObj.Value("Name") = "Smith" OR  
FieldObj.Value = "1234"
```

Structure objects are provided only to perform operations on export and import parameters. Do not attempt to use the methods and properties for this object type with table rows.

Structure and Parameter objects are fundamentally different from the other object types. They are not maintained in their own collection lists, and do not occur in any other object type as a property or method. As a result, expressions of the form

```
MyFunction.Structure.Value("field-name") OR  
MyFunction.Parameter.Value
```

are not valid and result in runtime errors.

To get information on a parameter's structure definition, use the online Assistant.

Using Named Argument Calling Conventions

Using Named Argument Calling Conventions

Since named argument calling conventions do not have qualifiers to distinguish the import parameters from the export parameters like the calling conventions used in ABAP, you must be aware which is the import parameter and which is the export parameter. From a caller's point of view, you can use either variables or constants for exporting parameters. However, you can only use variables for importing parameters so that the variables can store the returned data.

From the caller's point of view, if the exporting parameter is a structure and you want to pass data to this parameter, you must first create a Structure object using the `CreateStructure` method of the Functions collection and fill data in the Structure object. If the import parameter is a structure, you can pass any variable to receive the returned Structure object. There is no need to create Structure objects yourself.

If you only want to retrieve data in table parameters, you can use any variable. There is no need to create Table objects yourself. The remote Function object creates the Table objects and stores them in your variables. If you want to pass data to the table parameters, you must first create a Table object in the table component and assign data to the able object. Then, you can pass the table object to the table parameter.

The following example illustrates these two scenarios.



For the remote function interface:

```
xFunc

  Importing   IP      LIKE    TP-IP
              SIP     LIKE    SX STRUCTURE SX
  Exporting   EP      LIKE    TP-EP
              SEP     LIKE    SY STRUCTURE SY
  Tables      TP      STRUCTURESZ
```

If you only want to retrieve data, the following VBA code illustrates the calling statement.

Call the `xFunc` remote function:

```
R3.xFunc IP:= 1, SEP:= objStruct, EP:= nVar, TP:=objTable
```

where `objStruct`, `nVar`, and `objTable` can be uninitialized.

If you want to pass data to the table parameter and the export parameter with the structure type, the following VBA code illustrates the calling statement:

Create a Structure object with structure type "SX":

```
set objMyStruct = R3.CreateStructure("SX")
```

Create a Table object with table structure type "SZ":

```
set objTable = R3.CreateTable("SZ")
```

Fill in data for `objMyStruct` and `objTable` here:

```
...
```

Using Named Argument Calling Conventions

Call the xFunc remote function:

```
R3.xFunc IP:=1, SIP:= objMyStruct, EP:= nVar, SEP:= objStruct,  
TP:=objTable
```

Functions Collection Object

The Functions collection object manages the resources needed to make function calls to an R/3 System. These resources include Function objects and a single Connection object. You must get a Connection object and log onto the R/3 System before requesting new Function objects.

The Functions collection object is implemented as an SAP standard collection object. For information on SAP collection objects, see: [SAP Standard Collection \[Page 145\]](#).

[Properties \[Page 168\]](#)

[Methods \[Page 169\]](#)

Functions Collection Properties

The Functions collection object has the following properties:

Functions Collection Properties

Name	Parameters	Return Type	Description
Connection		VT_DISPATCH	Returns or sets the Connection object. If no Connection object exists, a new one is created. See the Connection Object [Page 490] . Read/write.
Count		VT_I4	Number of Function objects in the list. Read-only.
LogFileName		VT_BSTR	The log file name. Read/write.
LogLevel	Level	VT_I4	Returns or sets the current log-level. Level is a VT_I2 with the values 0-9 (where 0 means no log information, and 9 means full information). Read/write.

For information on SAP collection objects, see: [SAP Standard Collection \[Page 145\]](#).

Functions Collection Methods

The Functions collection object has the following methods:

Functions Collection Methods

Name	Parameters	Return Type	Description
Add	FunctionName	VT_DISPATCH or VT_EMPTY	Creates a Function object and adds it to the collection. If successful, returns the Function object interface. The 'FunctionName' parameter is required and is a string (VT_BSTR).
Item	Index	VT_DISPATCH	Returns a Function object from the collection. The 'Index' parameter is required and may be a string (VT_BSTR) or an integer (VT_I4).
Remove	Index	VT_EMPTY	Removes a Function object from the collection. Uses an indexing argument in the same way as the Item method.
RemoveAll		VT_EMPTY	Removes all Function objects from the collection.

For information on SAP collection objects, see: [SAP Standard Collection \[Page 145\]](#).

Connection Object

Connection Object

The Function control uses a Connection object to connect to the R/3 System. Only one Connection object can be associated with a Functions collection. You must log on with the Connection before any new Function objects can be requested.

You can create the Connection object through the Functions collection, or you can acquire it from a Logon control. The [Logon control \[Page 458\]](#) provides a Connection object through its [NewConnection \[Page 488\]](#) method.

Connection objects can be shared by multiple Function controls. Every Connection object continues to exist as long as there is any control that refers to it.

The following code examples illustrate two ways to create Connection objects:

- [Connecting through a Logon Control \[Page 171\]](#)
- [Setting the Connection Implicitly \[Page 172\]](#)

To create functions and call them in an R/3 System, you first have to establish a connection. (Function definitions are retrieved from the R/3 System.)

Connecting through a Logon Control

The following code shows how to set a Connection object directly through the Logon control.

Code	Comment
<code>Set LogonOCX = CreateObject ("SAP.LogonControl")</code>	Creates a Logon control.
<code>Set fns.Connection = LogonOCX.NewConnection</code>	Creates a new Connection object and sets it in the Functions collection object.
<code>Set fns.Connection.User = "Csmith"</code>	Provides logon information to the R/3 System.
<code>...</code>	
<code>fns.Connection.Logon (0,True)</code>	Silent logon.

Setting the Connection Implicitly

The following code shows how to set a Connection object implicitly from inside the Functions collection.

Code	Comment
<code>Set Conn = RfcObj.Connection</code>	Gets the Connection object using the Connection method of the Functions collection.
<code>Conn.User = "CPIC"</code>	Assigns a user name.
<code>Conn.Password = "test"</code>	Assigns a password.
<code>...</code>	
<code>Conn.Logon(0, False)</code>	Opens the connection.

Function Object

A Function object calls an RFC function, bundles parameters and makes results available. You can call a function in two ways, with the Call method or the [Dynamic Function Call \[Page 174\]](#) method.

[Properties \[Page 175\]](#)

[Methods \[Page 177\]](#)

Dynamic Function Call

Dynamic function calls allow you to treat R/3 function modules as if they were methods of a Function collection. For example:

```
result = <FunctionOCX>.RFC_CUSTOMER_GET (Name1 = "JOHN")
```

Creation of a Function object for the function is delayed until the R/3 System is accessed for function information. If the function RFC_CUSTOMER_GET exists, the Function object is created and executed.

One positional parameter is allowed. The first parameter can be a string that holds the exception message after the call.

Function Properties

The Function object has the following properties:

Function Object Properties

Name	Parameters	Return Type	Description
Exception		VT_BSTR	If an exception occurs during the call, this returns a string containing the exception text.
Exports	Index	VT_DISPATCH	Returns an Exports collection object (Exports Collection Object [Page 178]) that contains a list of export parameters used in the function. If the index is left empty, the whole collection is returned. Otherwise, the parameter indicated by number or string will be returned. Read-only.
Imports	Index	VT_DISPATCH	Returns an Imports collection object (Imports Collection Object [Page 181]) that contains a list of import parameters used in the function. The index can be empty. If the index is left empty, the whole collection is returned. Otherwise, the parameter indicated by number or string will be returned. Read-only.
Name		VT_BSTR	Returns an ABAP function name. This is the default property. Read-only.
Parent		VT_DISPATCH	The Functions Collection Object [Page 167] that contains this function.
Tables [Page 176]		VT_DISPATCH	Returns a Tables collection object that contains a list of ABAP internal tables used in the function. See the Table Collection Object [Page 309] topic in the Table Factory Control.
Description		VT_BSTR	Returns the documentation for the function module as found in the R/3 System.

Function Property: Tables

Function objects must have all their Table objects attached at all times. If the associated Table objects are unloaded (live outside of the Function objects), the Function objects will create new Table objects for themselves. Whenever the existing Table objects are replaced from outside, the Function object will remove and replace these objects.

When you get a Tables collection object or a Table object from a Function object, you receive a dispatch pointer to that object directly from the Table Factory control. Treat these pointers (objects) exactly as described for the corresponding part of the [SAP Table Factory \[Page\] 300](#).



Code	Comment
<pre>Set customers = MyFunct.tables.Item ("CUSTOMER_T")</pre>	<p>Here, you attach the dispatch pointer to the Customers object, then access the Tables property of the Function object. Inside the Table object, access the item "Customer T" with the Item method.</p>

The Tables collection is a direct connection to the Table object in the Table Factory control. This means that it supports all properties and methods of the Tables object in the Tables collection object.



Code	Comment
<pre>Set Table = function.Tables ("TAB1")</pre>	<p>Here, when the "item" is implicit, the Item property is invoked by default.</p>

Function Methods

The Function object has the following methods:

Function Object Methods

Name	Type	Description
Call	VT_BOOL	Calls the ABAP function. If successful, returns True. Packs the RFC parameters and sends them to R/3.

Exports Collection Object

Exports Collection Object

The Exports collection object maintains the export parameters for a Function object. An export parameter can either be a [Parameter Object \[Page 188\]](#) or a [Structure Object \[Page 184\]](#).

[Properties \[Page 179\]](#)

[Methods \[Page 180\]](#)

Exports Collection Properties

The Exports collection object has the following properties:

Exports Collection Properties

Name	Parameters	Return Type	Description
Parent		VT_DISPATCH	Returns the Function object (See Function Object [Page 173]) that owns this Exports collection object. Read-only
Item	Index	VT_DISPATCH	Returns an export parameter from the collection. The 'Index' parameter is required and may be a string (VT_BSTR) or an integer (VT_I4).
Count		VT_I4	Number of the export parameters in the list. Read-only.

Exports Collection Methods

The Exports collection object has the following methods:

Exports Collection Methods

Name	Parameters	Return Type	Description
Remove	Index	VT_EMPTY	Destroys and removes the specified parameter from the collection. The 'Index' parameter is required and may be a string (VT_BSTR) or an integer (VT_I4).
RemoveAll		VT_EMPTY	Removes all parameters from the collection.
Unload	Index	VT_DISPATCH	Like the Remove method, except that the item is not destroyed. After unloading, the item is no longer a member of the collection, but continues to exist for other purposes. Unload returns the unloaded object.

Imports Collection Object

The Imports collection object maintains the import parameters for a Function object. An import parameter can either be a [Parameter object \[Page 188\]](#) or a [Structure object \[Page 184\]](#).

[Properties \[Page 182\]](#)

[Methods \[Page 183\]](#)

Imports Collection Properties

The Imports collection object has the following properties:

Imports Collection Properties

Name	Parameters	Return Type	Description
Parent		VT_DISPATCH	Returns the Function object (See Function Object [Page 173]) that owns this Imports collection object. Read-only
Item	Index	VT_DISPATCH	Returns an import parameter from the collection. The 'Index' parameter is required and may be a string (VT_BSTR) or an integer (VT_I4).
Count		VT_I4	Number of the import parameters in the list. Read-only.

Imports Collection Methods

The Imports collection object has the following methods:

Imports Collection Methods

Name	Parameters	Return Type	Description
Remove	Index	VT_EMPTY	Destroys and removes the specified parameter from the collection. The 'index' parameter is required and may be a string (VT_BSTR) or an integer (VT_I4).
RemoveAll		VT_EMPTY	Removes all parameters from the collection.
Unload	Index	VT_DISPATCH	Like the Remove method, except that the item is not destroyed. After unloading, the item is no longer a member of the collection, but continues to exist for other purposes. Unload returns the unloaded object.

Structure Object

Structure Object

The Structure object contains a ABAP structure, as provided by the R/3 System.

[Properties \[Page 185\]](#)

[Methods \[Page 187\]](#)

Structure Properties

The Structure object has the following properties:

Structure Object Properties

Name	Parameters	Return Type	Description
ColumnCount		VT_I2	Returns the number of columns in the Structure object. Read-only.
Type		VT_BSTR	Returns the name of the Structure object. Read-only.
Function		VT_DISPATCH	Returns the Function object that owns the Structure object (or nothing if there is no owner). Read-only.
Name		VT_BSTR	Returns ABAP name for the structure. Read-only.
Width		VT_I2	Returns the structure width. Read-only.
ColumnName	Index	VT_BSTR	Gets column name at pos. <i>index</i>
Value	Member	VT_VARIANT	Sets and returns a value for a given member. This is the default property. The required 'member' parameter is a string (VT_BSTR)
ColumnOffset	Index	VT_I2	Returns the first byte for the column (from the start of the structure). Read-only.
ColumnLength	Index	VT_I2	Returns the length of the column in bytes. Read-only.
ColumnSAPType [Page 186]	Index	VT_I4	Returns the SAP internal type for the column. Read-only.
ColumnDecimals	Index	VT_I2	Returns the number of decimal places to the right of the decimal point (for numeric columns only). Read-only.

Structure Property: ColumnSAPType

Structure Property: ColumnSAPType

The ColumnSAPType property returns the internal type for the column. Possible values are:

Enum Name	Enum Value	Meaning
RfcTypeChar	0	String
RfcTypeDate	1	Date
RfcTypeBCD	2	BCD
RfcTypeTime	3	Time
RfcTypeHex	4	Binary
RfcTypeNum	6	Numeric
RfcTypeFloat	7	Float
RfcTypeLong	8	Long
RfcTypeShort	9	Short
RfcTypeByte	10	Byte

Structure Methods

The Structure object has the following methods:

Structure Object Methods

Name	Return Type	Description
IsStructure	VT_BOOL	Helps determine whether a named object is a parameter or a structure. (For a Structure object, returns TRUE.)
Clear	VT_EMPTY	Initializes the Structure object to default values.

Parameter Object

Parameter Object

The Parameter object contains an RFC parameter. Parameter objects are always contained in an [Exports collection \[Page 178\]](#) or [Imports collection \[Page 181\]](#).

[Properties \[Page 189\]](#)

[Methods \[Page 190\]](#)

Parameter Properties

The Parameter object has the following properties:

Parameter Object Properties

Name	Return Type	Description
Function	VT_DISPATCH	Returns the Function object that owns this Parameter object. Read-only
Length	VT_I4	Returns the byte length of the data.
Name	VT_BSTR	Returns the parameter name.
SAPType	VT_I2	Returns the ABAP data type of the parameter.
Type	VT_BSTR	Returns the RFC type.
Value	VT_VARIANT	Sets or returns the value of a parameter. This property is the default.
Description	VT_BSTR	Returns the parameter description documented in the R/3 System.

Parameter Methods

Parameter Methods

The Parameter object has the following methods:

Parameter Object Methods

Name	Type	Description
IsStructure	VT_BOOL	Helps determine whether a named object is a parameter or a structure. Returns always FALSE.
Clear	VT_EMPTY	Initializes the object to default values.

The Transaction Control

Purpose

The Transaction control is an OCX control that allows you to execute R/3 Batch Input transactions from external programs.

You can use the Transaction control to enter data into R/3 by executing R/3 transactions.

Implementation Considerations

The Transaction control can be used in COM-compliant applications, such as those programmed in Visual Basic, C++, and so on.

Features

The Transaction control provides an object oriented view of the R/3 transactions, in that it uses a [hierarchy of transactions, screens, and fields collections and objects \[Page 192\]](#). You use these objects to enter data into R/3 transactions that accept batch input.

The Transaction control simplifies the use of batch input transactions from external programs in that it eliminates the need to populate the fields of the BDC table and then send the BDC table to the R/3 system. With the Transaction control you need only assign values to fields on R/3 screens. The Transaction control takes care of the data transfer.

Constraints

The Transaction control only enables transaction calls in batch input mode. External programs can send screen values to the transaction, but the interface does not return output field values.

For complete transaction execution (with data transfer in both directions), use the [SAP Automation GUI Library \[Ext.\]](#).

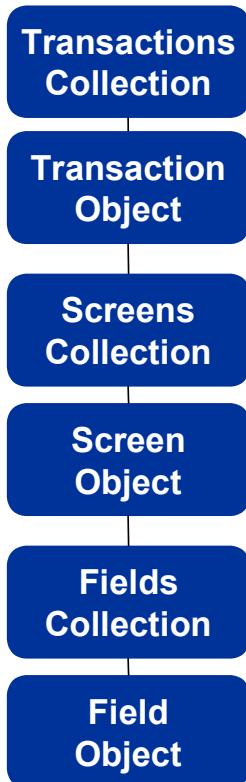
Activities

To use the Transaction control, you must be very familiar with the R/3 transaction you are calling: you need to know the sequence of screens it contains, and for each screen you need to know the fields it contains and what values those fields can take.

Using the Transaction control you create a transaction and all of its screens, and then you assign values to every field you wish use for data entry. You then call the transaction to perform the batch input.

Transaction Control Object Hierarchy

The highest level object in the Transaction control is a Transactions collection object. The Transactions collection object contains Transaction objects, each of which represents a single transaction you wish to execute. The Transaction collection maintains a single R/3 connection for all the Transaction objects it contains. Each transaction object contains a Screens collection, representing the set of screens associated with the transaction. The Screens collection contains the individual screens as Screen objects. Every Screen object contains a Fields collection representing the set of fields on that screen. Every Field object allows you to define the value you wish to enter into that field. The following diagram summarizes the hierarchy of Transaction control objects.



Using the Transaction Control

Using R/3 Transactions Interactively

R/3 transactions are composed of a sequence of screens, into which a user can enter data. An end user enters data into fields on these screens, then the user chooses a button or menu option that possibly leads to another screen. After entering data at the last screen in the sequence, the transaction ends.

Using R/3 Transactions with the Transaction Control

When using the Transaction control, you are using R/3 transactions in Batch Input mode. Note that Batch Input mode is different from the online mode. Not all transactions that are available in online mode are available in Batch Input mode.

When working with the Transaction control you need to specify the transaction you are using, its screens, and the fields you are sending values for. You need to specify similar parameters to those you specify when transferring data into R/3 with Batch Input methods. For example, to enter a value into a field, you need to provide field name and field value. To execute a function, you need to use the constant BDC_OKCODE, representing the command field on the screen.

An easy way to obtain the code of a transaction, its screens and all the necessary fields is by recording a Batch Input transaction (Transaction SHDB). Using transaction SHDB you record the desired transaction while going through its screens and entering data as an end user. At the end of the transaction (which ends the recording) the transaction, the screens and the fields you have used appear in a tree hierarchy.

Procedure

1. Create the Transactions collection.
2. Create a transaction object for every transaction you are going to use. Do so by using the Add method of the Transaction collection.
3. Create the screens of the transaction as objects of the Transaction control. Create a screen by using the Add method of the Screens collection. You then specify the program name and number as properties of the individual Screen object.
4. Add each of the fields that you wish to use for data entry to the Fields collection, and also specify the field name and its value by using the Name and Value properties of the Field object.
5. Set the Connection object: Use the SAP Automation [Logon control \[Page 458\]](#) to create a Connection object, and assign this connection object to the Connection property of the Transactions collection. This allows the Transactions collection to take care of the connection to R/3 for executing the transaction.
6. Call the transaction by using the Call method of the Transaction object.

Note that the Transaction control always calls the transaction in batch input mode. This means that the external program can send input field values to an R/3 screen, but output field values are not returned. For complete transaction execution, with data transfer in both directions, use the [SAP Automation GUI Library \[Ext.\]](#).

Also note that no GUI is displayed when you use the Transaction control.

Using the Transaction Control

Example

```
' Create a Transactions collection:  
Set transOCX = CreateObject("SAP.Transactions.1")  
  
' Add a Transaction object for the transaction:  
Set trans = transOCX.Add("SE11", "DOMAIN")  
  
' Add screens and fields:  
Set Screen = trans.Screens.Add  
  
Screen.Program = "SAPMSRD0"  
Screen.Number = "0100"  
  
Create the Fields collection.  
Set Fields = Screen.Fields  
  
Fields.Add "RSRD1-OBJNAME", "ZTST"  
Fields.Add "RSRD1-DOMA", "X"  
Fields.Add "BDC_OKCODE", "=ADD"  
  
Set Screen = trans.Screens.Add  
Screen.Program = "SAPMSD01"  
Screen.Number = "0100"  
  
Set Fields = Screen.Fields  
  
Text = "Testing at " + Str$(Hour(Now)) + ":" + Str$(Minute(Now))  
Fields.Add "DD01V-DDTEXT", Text  
Fields.Add "DD01V-DATATYPE", "CHAR"  
Fields.Add "DD01V-LENG", "10"  
Fields.Add "BDC_OKCODE", "/11"  
  
Set Screen = trans.Screens.Add  
Screen.Program = "SAPLSTRW"  
Screen.Number = "0100"  
Screen.Fields.Add "BDC_OKCODE", "/9"  
  
' Use the Logon Control to create a Connection object.  
' Assign the Connection object to the Connection property  
' of the Transaction collection:  
  
Dim conn as object  
Dim locx as object  
  
set locx = CreateObject ("SAP.LogonControl.1")  
set conn = locx.NewConnection  
set transOCX.Connection = conn
```

Using the Transaction Control

```
' Call the transaction:  
  
if trans.Call <> true then  
    MsgBox "Call failed..."  
End If
```

See Also

For more details on using Batch Input programming, see the topic [Data Transfer \[Ext.\]](#) in the [BC - Basis Programming Interfaces \[Ext.\]](#) section of the R/3 Library documentation.

What's New in Release 4.6A?

What's New in Release 4.6A?

The following are the changes to the Transaction Control in Release 4.6A:

- The underlying RFC call for the Call method of the Transaction control has changed from RFC_CALL_TRANSACTION to ABAP_CALL_TRANSACTION. This architectural change of the Transaction control ensures that:
 - Authorization is properly checked with every transaction call
 - The proper return codes are returned to all transactions you call

This change in the underlying RFC call applies to Transaction Control of releases 4.5A or later.

This functionality is available for release 4.0B with a Hot Package. Refer to SAPNet R/3 Note number 146433 for details and for the path to the Hot Package on sepervX.

- The Call method of the [Transactions collection \[Page 197\]](#) is no longer available. Note that you call a transaction by using the [Call method of the Transaction object \[Page 203\]](#), instead.
- A new property had been added to the [Transaction object \[Page 201\]](#): the Subrc property.
- The Message [property of the Transaction object \[Page 201\]](#) had been replaced by the Messages property.

Transactions Collection Object

The Transactions collection object allows you to list and view the Transaction objects currently in use. To create a Transactions collection, use the Visual Basic statement:

```
CreateObject("SAP.Transactions")
```

The Transaction collection maintains a single R/3 connection for all the Transaction objects contained in the collection.

[Properties \[Page 198\]](#)

[Methods \[Page 199\]](#)

Transactions Collection Properties

Transactions Collection Properties

Property Name	Return Type	Access Type	Description
Count	Long	Read-only	Returns the number of Transactions objects in the collection list.
Connection	Object	Read/write	Returns or sets the connection used to log on to the R/3 System.
LogFileName	String	Read/write	The log file name.
LogLevel	Integer	Read/write	Returns or sets the current log level. Level has values 0-9 (where 0 means no log information, and 9 means full information).

Transactions Collection Methods

Method Name	In Parameters	Return Type	Description
	Param	Type	
Add	Transaction Code	String	Variant Creates a Transaction object and adds it to the collection. If successful, returns the Transaction object. The TransactionCode parameter is the transaction name defined in the R/3 System and is required. The UserIdentifier parameter can be any string used to identify the set of Transaction objects with the same transaction code. This parameter is required.
	UserIdentifier	Variant	
Item	Transaction Code	String	ITransaction Returns a Transaction object from the collection. The parameters used here are defined the same as for the Add method. The Item method is the default method for the Transactions collection object.
	UserIdentifier	Variant	
Remove	Transaction Code	Variant	Boolean Removes a Transaction object from the collection. Any references to this removed Transaction object are invalid. The parameters here are the same as for the Add method, except that the TransactionCode parameter can also be an Index (numeric) when the UserIdentifier is omitted.
	UserIdentifier	Variant	
Remove All	(None)		(None) Removes all Transaction objects from the collection.

Transaction Object

Transaction Object

The Transaction object encapsulates one R/3 transaction: the screens and screen fields relevant to that transaction.

The Transaction object manages a set of Screen objects and defines the sequences that are used when calling an R/3 System transaction.

[Properties \[Page 201\]](#)

[Methods \[Page 203\]](#)

Transaction Object Properties

Property Name	Return Type	Access Type	Description
Messages	Table	Read – only	Returns the numbers of all transaction messages. The returned table has the same structure as the BDCMSGCOLL table in R/3.
Screens	IS screens	Read – only	Returns a Screens collection object containing a list of Screen objects.
TransactionName	String	Read – only	Returns the name of the Transaction object.
UpdateMode	String	Read - write	The UpdateMode determines the type of update for a transaction call and can have the following values:
		'A'	Asynchronous update
		'S'	Synchronous update (the default)
		'N'	No display
TransactionID	String	Read – only	Returns the UserIdentifier parameter specified when the Transaction object was created (using the Add method for the Transaction collection).

Transaction Object Properties

Subrc	Lo ng	Re ad – on ly	Returns the sy-subrc value of the call transaction statement in ABAP.
--------------	----------	---------------------------	---

Transaction Object Methods

Method Name	In Parameters	Return Type	Description
Call	(None)	Boolean	Calls the R/3 System transaction using the screen sequences defined in the Transaction object. If call transaction is complete, returns True. Use the Message property to determine whether the call transaction is successful.
Screens	Index	Variant	IScreen Returns a Screen object from the Screens collection associated with the transaction. The Screen parameter is optional. The Index parameter is required. If the Screen parameter is omitted, the Index parameter specifies the order of the Screen object. Otherwise, it specifies the program name.
	Screen	Variant	

Screens Collection Object

Screens Collection Object

The Screens collection object contains a list of all the active Screen objects that belong to the transaction being executed.

[Properties \[Page 205\]](#)

[Methods \[Page 206\]](#)

Screens Collection Properties

Property Name	Return Type	Access Type	Description
Count	Long	Read-only	Returns the number of Screen objects in the list.
Parent	Object	Read-only	Returns the parent Transaction object for the Screens collection object.

Screens Collection Methods

Screens Collection Methods

Method Name	In Parameter(s)	Return Type	Description
Add	(None)	IScreen	Creates a Screen object in the Screens collection and returns the Screen object if successful.
Item	Index	Variant	Object Returns a Screen object from the collection. The ScreenNumber parameter is optional. The Index parameter is required. If the ScreenNumber parameter is omitted, the Index parameter specifies the order of the Screen object. Otherwise, it specifies the program name. The Item method is the default member for the Screens collection object.
	Screen Number	Variant	
Remove	Index	Variant	Removes a single Screen object from the collection. Any references to the removed Screen object are invalid. The parameters are defined the same as for the Item method, except that the Index parameter can also be a Screen object, if the ScreenNumber parameter is omitted.
	Screen Number	Variant	
Move	FromIndex	Variant	IScreen Moves the Screen object from its current position (specified by either FromIndex or FromScreen) to the position ToIndex. The Screen object moved is returned.
	FromScreen	Variant	
	ToIndex	Long	
GetIndex	ProgramName	String	Returns the index for the Screen object in the collection. The screen is identified by ProgramName and ScreenNumber (as given in the R/3 System).
	Screen Number	String	
Insert	Index	Long	Inserts the screen Item into the collection at position Index. Returns the inserted

Screens Collection Methods

	Item	Ob jec t		object.
--	------	----------------	--	---------

Screen Object

Screen Object

The Screen object manages a set of Field objects and can have only a subset of fields that are used in a screen in the R/3 System.

[Properties \[Page 209\]](#)

[Methods \[Page 210\]](#)

Screen Object Properties

Property Name	Return Type	Access Type	Description
Number	String	Read-only	Returns the screen number of the Screen object.
Program	String	Read-only	Returns the program name with which the Screen object is associated.
Parent	Object	Read-only	Returns the Screens collection that is the parent of this screen.
Fields	Object	Read-only	Returns the Fields collection that includes the fields on this screen.

Screen Object Methods

Screen Object Methods

Method Name	Parameter(s)	Return Type	Description
Fields	Index	Object	Returns a Fields collection object containing a list of Field objects. Read-only. The Index parameter is required and may be a string or an integer.

Fields Collection Object

The Fields collection object contains the Field objects that are being used by the current Screen object. It is used for viewing, adding, and deleting items in the Fields collection.

[Properties \[Page 212\]](#)

[Methods \[Page 213\]](#)

Fields Collection Properties

Fields Collection Properties

Property Name	Return Type	Description
Count	Long	Returns the number of Field objects in the list. Read-only.
Parent	IScreen	Returns the parent Screen object for the Fields collection object. Read-only.

Fields Collection Methods

Meth od Name	Paramet ers	Retur n Type	Description	
Add	FieldName	Vari ant	IField	Creates and returns a Field object to the collection if successful. Initializes the value of the field to the <i>FieldValue</i> parameter.
	FieldValue	Vari ant		If the <i>FieldValue</i> is omitted, returns a Field object without a value. If both parameters are omitted, returns an unnamed, empty Field object.
Item	Index	Vari ant	IField	Returns a Field object from the collection. The Index parameter is required and can be either a field name or an index in the collection. The Item method is the default member for the Fields collection object.
Rem ove	Index	Vari ant	Boolea n	Removes a single Field object from the collection. The Index parameter is required and can be either a field name or an index in the collection. Any references to the removed Field object are invalid.

Field Object

Field Object

The Field object represents user interface elements in the SAPGUI. These elements are usually used for data entry. The Field Object has properties but no methods:

[Properties \[Page 215\]](#)

Field Object Properties

Property Name	Return Type	Access Type	Description
Name	String	Read–write	Sets or returns the name of the Field object.
Parent	IFields	Read–only	Returns the Fields collection object containing the Field object.
Value	Variant	Read–write	Sets or returns a value of the Field object. This is the default property.

The Table Tree Control

This section contains the following topics:

Introduction

[Introduction \[Page 217\]](#)

[Table Tree Object Hierarchy \[Page 218\]](#)

[Basic Concept \[Page 219\]](#)

Control and Object Reference

[Table Tree Object \[Page 220\]](#)

[Nodes Collection Object \[Page 237\]](#)

[Node Object \[Page 245\]](#)

[Structures Collection Object \[Page 255\]](#)

[Structure Object \[Page 262\]](#)

[Design Environment Property Pages \[Page 275\]](#)

Programming Guide

[Connecting Tree Views and Table Objects \[Page 272\]](#)

[Configuring the Tree \[Page 268\]](#)

[Drag and Drop with Tree Views \[Page 274\]](#)

[Appearance for Different Configurations \[Page 283\]](#)

[Pre-Defined Images \[Page 299\]](#)

[Code Examples \[Page 285\]](#)

[Glossary \[Page 286\]](#)

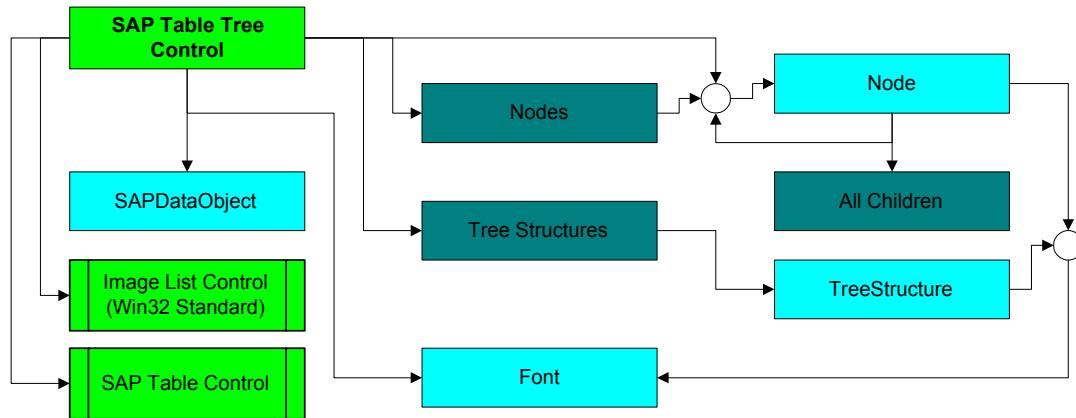
Introduction

The SAP Table Tree control is an R/3-aware active OLE control that simplifies the handling of hierarchical data. Using OLE Automation technology, the SAP Table Tree control can work either as a standalone control or in combination with other SAP controls like the [SAP Table Factory \[Page 300\]](#).

The Table Tree control can be used independently of any R/3 System. In fact, the control can run in any active control container as an advanced active OLE control. However, to get full R/3-aware functionality with the Table Tree control, you must use it together with the [SAP Table Factory \[Page 300\]](#) control. Combining both controls lets you handle hierarchical R/3 data easily and quickly.

Table Tree Object Hierarchy

Table Tree Object Hierarchy



Basic Concept

The SAP Table Tree control is a Windows-active OLE control that can work in various scenarios. The user can specify the control's appearance and data in two ways:

- in [design mode \[Page 275\]](#) (using different property pages)
- at runtime using OLE automation calls

In either case, the structure of a node in the tree (i.e., the properties the node has) can be defined by the user. The user can then store and display any information within a node of the tree. (See also [Configuring the Tree \[Page 268\]](#)).

The Table Tree control supports several features:

- [drag-and-drop operations \[Page 274\]](#) on demand
A default implementation for drag and drop transfers data in a pre-defined format, yet the use of your own transfer schemes is also supported. A set of events enables the user to react immediately on state changes of the control.
- persistent storage methods for handling static R/3 data
- R/3-aware execution

The SAP Table Tree control can perform as an R/3-aware control when you connect it to the [Views \[Page 370\]](#) collection of a Table object. Table objects are part of the [SAP Table Factory \[Page 300\]](#) control. Connecting these controls provides highly automated procedures for displaying and navigating through hierarchical data obtained from an R/3 System.

Table Tree Object

The SAP Table Tree object is the highest object in the Table Tree control's hierarchy. Therefore it is also called [root object \[Page 289\]](#). In Visual Basic program, you obtain a Table Tree object by calling `CreateObject("SAP.TableTreeControl")`. In design mode, you can insert a Table Tree control directly into a form from a toolbar.

[Properties \[Page 221\]](#)

[Methods \[Page 227\]](#)

[Events \[Page 228\]](#)

Table Tree Properties

Table Tree Properties

The Table Tree object has the following properties:

Table Tree Properties

Name	Type	Description
Events [Page 224]	Long	Enables or disables events fired by the control.
Nodes [Page 237]	Object	Returns a Collection [Page 298] of Node [Page 245] objects. These nodes are the root nodes of the tree control. Read-only.
Indentation	Short	Width of hierarchy expander in pixels. (See also the Width and Alignment properties of the Structure [Page 262] object.)
Structures [Page 255]	Object	Returns a collection [Page 298] of Structure [Page 262] objects. The Structure objects represent node properties: the set of properties applies to every node in the tree. (See Configuring the Tree [Page 268] .)
HideSelection	Boolean	Indicates whether the selected Node [Page 288] or Item [Page 287] should be highlighted when the control does not own the focus.
ImageList	Object	A list of images used for displaying image items in the tree. The image list is the standard Windows ImageList object and is only available on 32 bit architectures (Windows 95 or Windows NT 3.51 or later). If no image list object is assigned, images are taken from a pre-defined image pool [Page 299] . (See also Structure [Page 262] object and Node [Page 245] object.)
LabelEdit	Boolean	Indicates whether single items of a node object may be edited (not implemented yet).
PathSeparator	String	Separation character or string used in the FullPath property of the Node [Page 245] objects.

Table Tree Properties

Scrollbars	CScrollBarType	Indicates the kind of scrollbars. The Scrollbars property does not force scrollbars immediately. If the correct values are set, the scrollbars appear automatically when the display size requires them. Otherwise scrollbars are not available. Possible values are:	
		ScrollTypeNone = 0	No scrollbars.
		ScrollTypeHorizontal = 1	Vertical scrollbar visible when required.
		ScrollTypeVertical = 2	Horizontal scrollbar visible when required.
		ScrollTypeBoth = 3	Both scrollbars visible when required.
Sorted	Boolean	Indicates whether the tree should be sorted. Sorting is always done when the first property item in a node is a text item. (See also Structure [Page 255] object).	
SelectedNode	Object	Returns the currently selected Node [Page 245] object. If no node is selected, Nothing [Page 291] is returned. Read-only.	
DragDrop [Page 226]	CDragDropType	Enables and disables drag-and-drop operations.	
Font	Object	Font [Page 152] object used as default font.	
ScreenUpdate	Boolean	Enables or disables screen updating. This property is useful if many nodes in the tree are being modified, inserted or deleted. Changing ScreenUpdate from FALSE to TRUE always forces a screen update, no matter whether data was changed or not.	
BorderStyle	short	Sets the style of border. Possible value are:	
		Enable3D = 0	Draws a 3 D frame
		Simpleframe = 1	Draws a simple frame

Table Tree Properties

SelectMode	CTreeSelectMode	Sets the selection mode for single nodes and items	
		trvTreeModeLineSingle = 0	Only entire nodes may be selected
		trvTreeModeLineMultiple = 1	Not implemented yet
		trvTreeModelItemSingle = 2	Single items in a node may be selected
		trvTreeModelItemMultiple = 3	Not implemented yet
TableInsertRt	Object	Sets the parent node to insert a connected SAP Table object [Page 220] . (See also Connecting Tree Views and Table Objects [Page 272] .)	

Table Tree Property: Events

Table Tree Property: Events

You can use the Events property to enable or disable events or groups of events. Disabling events can improve performance, especially for operations that manipulate large chunks of data. Events may also be turned on and off temporarily.

Note that one event cannot be disabled: the DuplicatedKey event is always fired and cannot be turned off.

The following values are available for the Events property:

Value	Description
DisableAllTreeEvents = 0	Disable all events
EnableTableCreate = 1	Fire event after associated table is created. (Not implemented yet)
EnableTableClear = 2	Fire event after associated table is deleted (Not implemented yet)
EnableTreeDataChange = 4	Not implemented yet.
EnableNodeInsert = 8	Fire NodeInsert [Page 232] event after a node is inserted.
EnableNodeRemove = 16	Fire NodeRemove [Page 233] event after a node is removed.
EnableBeforeInput = 32	Fire BeforeLabelEdit event before user input.
EnableAfterInput = 64	Fire AfterLabelEdit event after user input.
EnableCollapse = 128	Fire Collapse event before a node is collapsed.
EnableExpand = 256	Fire Expand event before a node is expanded.
EnableClicks = 512	Fire Click, DblClick, ItemClick and ItemDblClick events.
EnableKeyboardEvents = 1024	Fire KeyUp and KeyDown events.
EnableSelectionEvent = 2048	Fire SelChange event.
EnableDragDropEvents = 4096	Fire DragSourceFill [Page 234] , DragComplete, DropEnter [Page 235] and Drop [Page 236] events.
EnableTreeError = 16384	Fire Error event after an error occurred.
EnableStandardTreeEvents	Enable summary of standard events.
EnableAllTreeEvents = 32767	Enable all events.

All values may be combined through **and** or **or** operations.



' Enable NodeInsert and NodeRemove event

Table Tree Property: Events

```
MyControl.Events = MyControl.Events or EnableOnNodeInsert or _  
EnableOnNodeRemove  
  
' Disable NodeInsert and NodeRemove event  
MyControl.Events = MyControl.Events and not _  
(EnableOnNodeInsert or EnableOnNodeRemove)
```

Table Tree Property: DragDrop

Table Tree Property: DragDrop

Purpose

Enables and disables drag-and-drop operations.

Return Value

type CDragDropType

Description

The following DragDrop values are possible:

Value	Description
DragDropModeDisable = 0	Turns off all default drag-and-drop operations.
DragFolders = 1	Only folders [Page 295] are sensitive for dragging. If a folder is dragged, the leaves [Page 296] in the folder are part of the drag-and-drop operation
DragLeafs = 2	Only leaves are sensitive for dragging.
DragAll = 3	Folders and leaves may be dragged.
DropFolders = 4	Only folders from other Table Tree controls may be dropped.
DropLeafs = 8	Only leaves [Page 296] from other Table Tree controls may be dropped
DropAll = 12	Folders [Page 295] and leaves [Page 296] from other Table Tree controls may be dropped.
DragDropFolders = 5	Same as DragFolder and DropFolder.
DragDropLeafs = 10	Same as DragLeafs and DropLeafs.
DragDropAll = 15	Same as DragAll and DropAll.



These values may *not* be combined by **and** or **or** operations. Also, you can implement drag-and-drop event handlers to override the control's default behavior completely. In these cases, it is the responsibility of the application to implement a reasonable drag-and-drop scenario.

See also [Drag and Drop with Tree Views \[Page 274\]](#).

Table Tree Methods

Table Tree Methods

The Table Tree object has the following methods:

Table Tree Methods

Name	Parameter(s)	Return Type	Description
IsKey	String szkey	Boolean	Returns TRUE, if szKey is the key of any Node [Page 288] in the table. (See also the Key property of the Node [Page 245] object).
AboutBox	void	void	Displays the AboutBox dialog.

Table Tree Events

The SAP Table Tree control fires several events in order to inform the container about state changes, user interaction and drag-and-drop operations. To enable or disable these events, use the Table Tree object's [Events \[Page 224\]](#) property.

The Table Tree events are:

Table Tree Events

Name	Parameters	Description
Click	void	A mouse click occurred within the control's client area.
DblClick	void	A double click occurred within the control's client area.
KeyDown	short* KeyCode short ShiftState	A key was pressed. The virtual key code is passed in KeyCode, the current state of the shift key in ShiftState. KeyCode may be modified within the event-handling routine.
KeyUp	short* KeyCode short ShiftState	A key was released. The virtual key code is passed in KeyCode, the current state of the shift key in ShiftState. KeyCode may be modified within the event-handling routine.
Collapse	Object Node	The Node [Page 245] passed in as parameter <i>Node</i> is being collapsed. This event is fired immediately before the node is collapsed.
Expand	Object Node	The node passed in as parameter <i>Node</i> is being expanded. This event is fired immediately before the node is expanded. It is legal to add children to the node in the Expand event. The added children are displayed properly afterwards.
NodeClick	Object Node	A click occurred on the node passed in as parameter <i>Node</i> .
ItemClick	Object Node Long Index	A click occurred on an Item [Page 287] object within the Node [Page 288] passed in as parameter <i>Node</i> . The parameter <i>Index</i> may be used to access the item data using the <i>Value</i> property of the Node [Page 245] object. (See also the <i>SelectMode</i> property and Structure [Page 262] object.)

Table Tree Events

ItemCbIClick	Object Node Long Index	A double click occurred on Item [Page 287] within the node passed in as parameter Node. The parameter Index may be used to access the item data through the Value property of the node object. (See also the SelectMode property and Structure [Page 262] object.)
ItemGotFocus	Object Node Long Index	The Item [Page 287] with index Index in Node received the input focus. (See also the SelectMode property and the Structure [Page 262] object.)
ItemLostFocus	Object Node Long Index	The Item [Page 287] with index Index in Node lost the input focus. (See also the SelectMode property and Structure [Page 262] object.)
SelChange	Object Node Long Index	The selected Node [Page 288] or Item [Page 287] has changed. The new selected node and item are passed in as parameter Node and Index. Index indicates the selected item within the node. If Index = -1, no node is selected. Otherwise the parameter Index may be used to access the item's data through the Value property of the Node [Page 245] object. (See also the SelectMode property and Structure [Page 262] object).
DuplicatedKey [Page 231]	Object Node String* NewKey String OldKey Short* Handled	An already-existing key is being inserted or assigned.
BeforeLabelEdit	Object Node Long Index Short* Cancel	The user has begun to edit an Item [Page 287] in Node [Page 288] Node. Index indicates which item within node. Index may be used to access the item's data through the Value property of Node, or to retrieve the item's description from the Structures [Page 255] collection. If Cancel is set to TRUE, the edit operation is postponed. (See also the SelectMode property and Structure [Page 262] object.)

Table Tree Events

AfterLabelEdit	Object Node Long Index Short* Cancel String NewString	The editing of an Item [Page 287] has completed. The new text is passed in as NewString and may be modified by the container. Index may be used to access the item's data through the Value property of the Node object, or to retrieve the item description from the Structures [Page 255] collection. If Cancel is set to TRUE, the entire edit operation is being postponed.
NodeInsert [Page 232]	Object Node	The node Node is being inserted.
NodeRemove [Page 233]	Object Node	The node Node is being removed.
DragSourceFill [Page 234]	Object DataObject Object Node Short* Handled	A drag-and-drop operation is starting with the node Node. DataObject is an SAP Data Object [Page 149] and may be filled with any format.
DropComplete	Object Node Long Effect	A drag-and-drop operation originally started on Node has been successfully completed. Effect describes the type of drop. It is up to the application to consider the Effect.
DropEnter [Page 235]	Object DataObject Long KeyState Long* Effect Short* Handled	During a Drag and Drop [Page 274] operation, the mouse pointer was moved into the client area of the control. (See also SAP Data Object [Page 149])
Drop [Page 236]	Object Node Object DataObject Long* Effect Short* Handled	A drop has occurred on the control's client area on node Node. (See also SAP Data Object [Page 149])

Table Tree Event: DuplicatedKey

Purpose

Event-notification that an already-existing key is being inserted or assigned.

Syntax

The DuplicatedKey event-handler has the syntax:

```
void DuplicatedKey(Object Node, String *NewKey,  
                    String OldKey, Short *Handled)
```

Parameters

- *Node*: [node \[Page 288\]](#) to which the key should be attached
- *OldKey*: key that caused the DuplicatedKey event
- *NewKey*: return parameter for filling in a new key
- *Handled*: return parameter that tells the container the event has already been handled

Description

This event is fired after the user tries to:

- insert a [Node \[Page 245\]](#) object with a key that already exists
- change a node's [Key \[Page 248\]](#) property to an already-existing value

There are several possible reactions to the DuplicatedKey event. Your event-handler could remove the [Node \[Page 288\]](#) by invoking the node's Remove method. Or the handler could change the node's [Key \[Page 248\]](#) property to a new value (by returning the new value in the parameter *NewKey*). However, you may *not* modify the Key property of the Node object. This would not lead to any change of the key.

In any case, you must set the *Handled* flag to TRUE to inform the control that the event has been handled. If you don't do this, an exception is raised. You can also return an empty string in *NewKey*. This removes the node from the key map but *not* from the control. Afterwards, the node is not accessible via keys anymore, until a valid key is assigned. If the event-handling routine provides a new key that already exists, a further DuplicatedKey event is fired.



Improper handling of the DuplicatedKey event may lead to an endless loop. This would be the case if no valid key is provided and the Handled flag is set to TRUE.

Table Tree Event: NodeInsert

Table Tree Event: NodeInsert

Purpose

Event-notification that a node has been inserted.

Syntax

The NodeInsert event-handler has the syntax:

```
void NodeInsert(Object Node)
```

Parameters

- *Node*: [node \[Page 288\]](#) being inserted

Description

The NodeInsert event is fired after the node *Node* was successfully inserted into the corresponding [Nodes \[Page 237\]](#) collection. All properties and methods of the node including the Remove method are valid at that time. This is a convenient time to perform application-dependent validation of the data stored in the node. It is possible to add additional information, to attach the node to a new parent, or to remove the node. (See also [Configuring the Tree \[Page 268\]](#)).

Table Tree Event: NodeRemove

Purpose

Event-notification that a node has been removed.

Syntax

The NodeRemove event-handler has the syntax:

```
void NodeRemove (Object Node)
```

Parameters

- *Node*: [node \[Page 288\]](#) being removed

Description

The NodeRemove event is fired immediately before the node *Node* is removed from the corresponding [Nodes \[Page 237\]](#) collection. This is a convenient time to invalidate application-dependent relations to other object.



It is not possible to cancel the Remove method in order to avoid removing the object. Furthermore all children have already been removed from the Children collection (Children property of the Node object) and the connection to the parent object has been released and returns [Nothing \[Page 291\]](#).

Table Tree Event: DragSourceFill

Purpose

Event-notification that a drag-and-drop operation has begun.

Syntax

The DragSourceFill event-handler has the syntax:

```
void DragSourceFill(Object DataObject, Object Node, Short *Handled)
```

Parameters

- *DataObject*: Data object containing the data being dragged
- *Node*: [node \[Page 288\]](#) where the operation is beginning
- *Handled*: parameter telling whether the container has handled the event

Description

The DragSourceFill event is fired when the user starts a [drag-and-drop operation \[Page 274\]](#) on node *Node*. The Data object to be used for drag and drop is passed in as *DataObject* and represents a [SAP Data Object \[Page 149\]](#).

If you code an event-handler for this event, your code can fill *DataObject* with the appropriate data, and set the *Handled* flag to TRUE. In this case, any default event processing by the control is disabled.

If the event-handler for this event does not set *Handled* to TRUE, the control performs default event-handling. This includes adding the selected data to the data object using the formats CF_TEXT, 'SAPTreeNodeStream' and 'SAPTreeItemStream'.

Any data stored in the data object using these formats is overwritten (see also [Logon Object \[Page 465\]](#)).

Table Tree Event: DropEnter

Table Tree Event: DropEnter

Purpose

Event-notification that the user has dragged data into a target control.

Syntax

The DropEnter event-handler has the syntax:

```
void DropEnter(Object DataObject, Long KeyState, Long * Effect, Short *Handled)
```

Parameters

- *DataObject*: Data object containing the data being dragged
- *KeyState*: (See the Visual Basic documentation.)
- *Effect*: Return parameter telling what kind of drop is allowed
- *Handled*: Parameter telling whether the container has handled the event

Description

The DropEnter event is fired when the mouse pointer is moved into the client area of a control during a [drag-and-drop operation \[Page 274\]](#). The *DataObject* passed to the DropEnter event handler is a [SAP Data Object \[Page 149\]](#) that was originally filled in the [DragSourceFill \[Page 234\]](#) event. (The source control may be either a Table Tree or a Table View control.)

By calling the method *IsFormatAvailable* on the *DataObject*, your event handler can determine whether the *DataObject* contains acceptable information or not. If the *DataObject* is acceptable, set the *Effect* parameter to one of the following values in order to change the cursor accordingly:

Value	Description	Cursor
DROPEFFECT_NONE = 0	Drop is not possible	
DROPEFFECT_COPY = 1	Drop of copy is possible	
DROPEFFECT_MOVE = 2	Move of DataObject is possible	

Set *Handled* to TRUE to prevent the control's default drag-and-drop handling from being invoked.

Table Tree Event: Drop

Purpose

Event-notification that the user has dropped data into a target control.

Syntax

The Drop event-handler has the syntax:

```
void Drop(Object Node, Object DataObject, Long * Effect, Short  
*Handled)
```

Parameters

- *Node*: target node into which data is being dropped
- *DataObject*: data object containing the data being dropped
- *Effect*: parameter telling what kind of drop is being performed
- *Handled*: parameter telling whether the container has handled the event

Description

The Drop event is fired if the control's client area is the drop target of a [drag-and-drop operation \[Page 274\]](#).

Use *DataObject*'s *IsFormatAvailable* and *GetData* methods to retrieve the data stored in *DataObject*. This data was originally filled in the [DragSourceFill \[Page 234\]](#) event and may be stored in any format. (The source control may be either a Table Tree or a Table View control.)

In most cases, you should then set *Handled* to TRUE. Otherwise the control's default drag-and-drop processing is executed.

The *Effect* parameter indicates whether the *DataObject* should be copied, moved or whether a link should be established.

The *Effect* parameter is initially set in the *DropEnter* event-handler, then passed in to *Drop* event-handler, and subsequently on to the *DropComplete* event-handler fired by the drop source control.

Nodes Collection Object

The Nodes collection is a collection of [Node \[Page 245\]](#) objects. It is implemented as a [standard collection \[Page 145\]](#) like many other SAP [collections \[Page 298\]](#) controls.

The Nodes collection may contain as many nodes as desired.

[Properties \[Page 238\]](#)

[Methods \[Page 240\]](#)

Nodes Collection Properties

Nodes Collection Properties

The Nodes collection object has the following properties:

Nodes Collection Properties

Name	Parameter	Type	Description
Count		Long	Returns the number of objects in the collection. Read-only.
Item [Page 239]	Variant valIndex	Object	Retrieves a Node [Page 245] from the Nodes collection.

Nodes Collection Property: Item

Purpose

Retrieves a Node object from a Nodes collection.

Syntax

The Item property has the syntax:

```
Object Item(Variant vaIndex)
```

Description

The Item property retrieves a [Node \[Page 245\]](#) object from the [Nodes \[Page 237\]](#) collection (and is the default property for Nodes collection objects). Set the parameter *valIndex* to describe the element you want returned. The following variant data types are legal types for *valIndex*:

- Any string data type

The [Node \[Page 245\]](#) object with a [Key \[Page 248\]](#) property equal to *valIndex* is returned. Since Key is a unique value within the entire control, the returned Node object need not be a member of the Nodes collection for which the method was called: the returned node may be any node within the entire control. If the key is not valid, *valIndex* is converted to an integer value that is used as index into the collection. This leads either to the desired node, or to one of two exceptions:

- Bad Index Exception (if *valIndex* can be converted to an integer value)
- Type Mismatch Exception (if *valIndex* could not be converted to an integer)
- Any type convertible to an Integer

The parameter *vaWhat* is converted to an integer value and used as index in the collection.

Nodes Collection Methods

The Nodes collection object has the following methods:

Nodes Collection Methods

Name	Parameters	Return Type	Description
Remove [Page 241]	Variant vaWhat	Boolean	Removes a Node [Page 245] object from the collection.
RemoveAll	void	void	Removes all nodes from the collection.
Add [Page 242]	Variant vaRelative CTreeAddType Relationship String Key Variant Text Long Image Long SelectedImage	Object	Adds and returns a new Node object to the collection.
AddEx [Page 244]	Variant vaRelative CTreeAddType Relationship String Key Variant Text Long Image Long SelectedImage CTreeNodeType Type	Object	Adds and returns a new node object with a given node type to the collection.

Nodes Collection Method: Remove

Nodes Collection Method: Remove

Purpose

Remove a Node item from a Nodes collection.

Syntax

The Remove method has the syntax:

```
Boolean Remove (Variant vaWhat)
```

Description

Removes a [Node \[Page 245\]](#) object from the [Nodes \[Page 237\]](#) collection. The parameter *vaWhat* describes the element to be removed. Legal variant types for *vaWhat* are Object or any data type that can be converted into an integer value. If *vaWhat* has type Object, the corresponding object is removed from the collection. Otherwise *vaWhat* is converted to an integer and used to remove the corresponding node. If the node has any children, all children are removed prior to removing the node.



When removing a node from the Nodes collection, the Node object becomes invalid. Any further attempt to work on the object returns an Invalid Object Exception.

Nodes Collection Method: Add

Nodes Collection Method: Add

Purpose

Adds a new Node object to the Nodes collection.

Syntax

The Add method has the syntax:

```
Object Add(Variant vaRelative, CTreeAddType Relationship, String Key,  
Variant Text, Long Image, Long SelectedImage)
```

Parameters

- *vaRelative*: (type Variant)

Indicates either an already existing node object or [Nothing \[Page 291\]](#) or an empty variant. A node may be referenced by an object expression containing the node or a string expression containing the key for the desired node.

The relationship between the new node and *vaRelative* is described in the parameter *Relationship*. The combination of *vaRelative* and *Relationship* defines the position of the new node to be inserted. If *vaRelative* is empty or Nothing, the new node is inserted as a root node. *Relationship* may not be equal to TreeAddFirstChild or, in this case, TreeAddLastChild.

- *Relationship*: (type CTreeAddType)

Describes the relationship between *vaRelative* and the new node. Possible value are:

- (TreeAddLastSibling = 1) Add as last sibling to *vaRelative*
- (TreeAddNextSibling = 2) Add as next sibling to *vaRelative*
- (TreeAddPreviousSibling = 3) Add as previous sibling to *vaRelative*
- (TreeAddFirstChild = 4) Add as first child to *vaRelative*
- (TreeAddFirstSibling = 5) Add as first sibling to *vaRelative*
- (TreeAddLastChild = 6) Add as last child to *vaRelative*

- *Key*: (type String)

A unique [Key \[Page 248\]](#) value used to identify the [Node \[Page 245\]](#) object. This value may later be used to retrieve the node through the Nodes collection's Item property. A valid value for *Key* is also an empty string. This prevents the key from being inserted into the key map.

- *Text*: (type Variant)

Describes initialization data for the new node to be inserted. There are two valid variant data types for this parameter:

- Any data type that can be converted to a string

The first [Item \[Page 287\]](#) of type text is initialized with *Text*. (See also [Structure Object \[Page 262\]](#)).

Nodes Collection Method: Add

- Object

The object pointed to by *Text* **must** be a [Node \[Page 245\]](#) object. All [items \[Page 287\]](#) with the same name and data type in both nodes (that is, in the *Text* node and the newly created node) are copied from *Text* to the new node. The item need not be a node from the same control, but it must be a node from the same application.
- *Image*: (type Long)

Indicates the image to be used for the first image typed item. If *Image* equals -1, pre-defined images for [folders \[Page 295\]](#) and [leafs \[Page 296\]](#) are used. Otherwise, *Image* is an index in the *ImageList* property of the Table Tree object. (See also the properties for [The Table Tree Control \[Page 216\]](#).)
- *SelectedImage*: (type Long)

(Not yet supported.)

Return Value

type Object

Description

This method adds a new [Node \[Page 245\]](#) object to the [Nodes \[Page 237\]](#) collection.

Note that adding a new node does not necessarily mean the node becomes visible. If any sibling of the node is visible, the new node will also be visible if possible. (See also the node's [Type \[Page 249\]](#) property and the [root's \[Page 289\]](#) Type property). If you want the node to be visible, call *EnsureVisible* on the new node object.

The node's Type property is initialized with *trvNodeTypeFolder*. If you want a node of type *trvNodeTypeLeaf* inserted, call [AddEx \[Page 244\]](#) or modify the Type property of the new node object.

Nodes Collection Method: AddEx

Nodes Collection Method: AddEx

Purpose

Adds a new Node object to the Nodes collection, and sets the node's type.

Syntax

The Add method has the syntax:

```
Object AddEx (Variant vaRelative,  
              CTreeAddType Relationship,  
              String Key,  
              Variant Text,  
              Long Image,  
              Long SelectedImage,  
              CTreeNodeType Type)
```

Description

This method is the same as the Nodes collection's [Add \[Page 242\]](#) method, except that you can pass the type of the new node to the method as additional parameter.

Node Object

The Node object is a single object that represents a single entry in the tree. A node may consist of as many single structure [items \[Page 287\]](#) as desired. (See also [Structure Object \[Page 262\]](#)).

[Properties \[Page 246\]](#)

[Methods \[Page 253\]](#)

Node Object Properties

Node Object Properties

The Node object has the following properties:

Node Object Properties

Name	Parameter	Type	Description
Expanded		Boolean	Indicates whether the node is currently expanded. If the Expanded property is assigned TRUE, the node is expanded as if the user had clicked on the expander symbol.
Index		Visible	Returns the index for the node within the Nodes [Page 237] collection. Read-only.
Selected		Long	Indicates whether the node is currently selected. May also be written.
Visible		Boolean	Indicates whether the node is currently visible. Read-only. (see also EnsureVisible)
Key [Page 248]		String	Unique identifying key value or an empty string.
Parent		Object	Returns the parent node object or Nothing [Page 291] if the node is a root node [Page 290] . This property may also be written in order to move a node to a different location in the tree.
LastSibling		Object	Returns the last node at the sibling level, or Nothing [Page 291] when invoked on an empty collection. Read-only.
FirstSibling		Object	Returns the first node at the sibling level, or Nothing [Page 291] when invoked on an empty collection. Read-only.
Next		Object	Returns the next node at the sibling level, or Nothing [Page 291] when invoked on the last node in a collection. Read-only.
Previous		Object	Returns the previous node at the sibling level, or Nothing [Page 291] when invoked on the first node in a collection. Read-only.
Child		Object	Returns the first child node, or Nothing [Page 291] if the node does not have any children. Read-only.

Node Object Properties

Children		Object	Returns a Nodes [Page 237] collection containing all the child nodes at the next level. Even though a node may not have any children, a valid object is always returned. If the node does not have any children, the Count property for the Children collection returns a value of 0. Read-only.
Type [Page 264]		CNodeType	Indicates the type of the node.
Font		Object	Font [Page 152] object used for all text items in the node.
ForceExpander [Page 250]		Boolean	If TRUE, the node is forced to display an expander symbol, no matter whether the Children collection is empty or not.
AllChildren [Page 251]		Collection	Enumerator with an IEnumVARIANT interface.
FullPath		String	Returns the full path of the object consisting of the name and the names of all parents. Each name is separated by the value assigned to the PathSeparator property of the Table Tree Object [Page 220]
Data		Array of Variant	Returns a safe array containing the data of all items [Page 287] within the node.
Value	Long <i>Idx</i>	Variant	Returns the value of a single item of the node. The data of a single item may also be accessed by Dynamic Node Properties [Page 293] created at runtime.

Node Object Property: Key

Node Object Property: Key

Purpose

Identifies a Node object uniquely with the Table Tree control.

Return Value

type String

Description

Every node can be identified by the value of its Key property. The Key value is unique not merely within a single Nodes collection but throughout the entire control. Use a key to retrieve a node from the control by calling the [Item \[Page 287\]](#) method of the [Nodes \[Page 237\]](#) collection.

If you attempt to insert a node with a key that already exists, or to change a key to an already existing value, a [DuplicatedKey \[Page 231\]](#) event is fired.

Nevertheless, if you are not interested in the key values, you may assign an empty string as key value to a node. This removes the node from the key map, but keeps the node itself within the control. All operations not involving a key may still be performed.

You can assign the empty string as Key value to as many nodes as desired.

Node Object Property: Type

Purpose

Indicates the type of the Node object.

Return Value

type CNodeType

Description

The Type property can return four possible values:

Value	Description	Default Icon
trvNodeTypeFolder = 0	The node is a folder [Page 295]	  Text
trvNodeTypeLeaf = 1	The node is a leaf [Page 296] node	 Text
trvNodeTypeHidden = 2	The node is hidden	Node is not displayed
trvNodeTypeDisabled = 4	The node is disabled	  Text

Different value may be combined by **and** or **or** operations.



' Hide a node

```
Node.Type = Node.Type or trvNodeTypeHidden
```

' Show a node

```
Node.Type = Node.Type and (not trvNodeTypeHidden)
```



Since trvNodeTypeFolder equals 0, and the node may also be hidden or disabled, it is not possible to determine whether a node is a folder by comparing the Type property with trvNodeTypeFolder. The proper method is to check whether a node is a leaf. For example:

```
' Check for Folder
if Node.Type and trvNodeTypeLeaf <> trvNodeTypeLeaf then.....
' Check for Leaf
if Node.Type and trvNodeTypeLeaf = trvNodeTypeLeaf then...
```

Node Object Property: ForceExpander

Node Object Property: ForceExpander

Purpose

Forces display of an expander symbol for a node.

Return Value

type Boolean

Description

Set the ForceExpander property to TRUE, if you want to force the node to display an expander symbol, regardless of whether the node has child nodes or not.

ForceExpander is intended for use with a special performance-improvement scenario. An application may find it desirable to delay filling a tree. In this case, it does not fill entire trees but rather only particular folders, as requested by the user. When requested, the filling can be done by the Expand event handler. It is thus possible to add all children for the expanding node at the last moment.

However, this technique would not work when a node that had no Children, since in this case, the node would normally be displayed without an [Expander symbol \[Page 294\]](#). As a result, it would not be possible to expand the node, and thus also impossible to fill the folder.

In this situation, the application can set the ForceExpander property to TRUE in order to allow the user to expand the corresponding node.

Node Object Property: AllChildren

Purpose

Returns a collection object containing all descendants for a node.

Return Value

type Collection

Description

The Allchildren property returns a collection object. This property stands for an IEnumVARIANT interface that lets the control user reference the descendants of the node, regardless of whether the node is expanded, enabled, visible, or hidden.



```
Dim Node As Object
Set Node = Tree.Nodes.Item(1);
for each node in Node.AllChildren
node.Type = node.Type and not NodeTypeHidden
next node
C++:
LPENUMVARIANT lpEnum;
HRESULT hr;
hr = pNode->QueryInterface(IID_IEnumVARIANT, (LPVOID FAR *) &
lpEnum);
if (SUCCEEDED(hr))
{
    VARIANT vaChildNode;
    lpEnum->Reset();
    while (hr == NOERROR)
    {
        VariantInit(&vaChildNode);
        hr = lpEnum->Next(1, &vaChildNode, NULL);
        if (hr == NOERROR)
        {
            hr = VariantChangeType(&vaChildNode,
&vaChildNode, VARIANT_NOPROPERTY, VT_DISPATCH);
            if (SUCCEEDED(hr))
            {

```

Node Object Property: AllChildren

```
LPDISPATCH lpChildNode;
lpChildNode = V_DISPATCH(&vaChildNode);
// work on the automation interface
...
}

VariantClear(&vaChildNode);
}

lpEnum->Releas();
}
```

Node Object Methods

The Node object has the following methods:

Node Object Methods

Name	Parameter	Return Type	Description
EnsureVisible		Boolean	Forces the node to be visible, expanding all parent nodes if necessary.
Remove		void	Removes the node from the parent Nodes [Page 237] collection. For root nodes [Page 290] , the node is removed from the Table Tree object.
SaveData [Page 254]	String DocName String StreamName	Boolean	Saves the data for the node and all descendant nodes.
LoadData [Page 254]	String DocName String StreamName	Boolean	Loads the data for the node and all descendant nodes.

Node Object Method: SaveData, LoadData

Node Object Method: SaveData, LoadData

Purpose

Saves or loads all data belonging to the node (and its descendants) to or from a file.

Syntax

The SaveData method has the syntax:

```
SaveData (String DocName, String StreamName)
```

The LoadData method has the syntax:

```
LoadData (String DocName, String StreamName)
```

Return Value

type Boolean

Description

These two methods save or load all data of the node (and of its descendants) to or from a file.

The parameters specify the compound document in *DocName* and a stream in *StreamName*. If the document does not exist, a document is created. If the stream does not exist, a stream is created, otherwise the existing stream is overwritten.

For these methods, the compound document is a file, and the stream contains data for a single tree (single node and its descendants). For multiple trees, different stream names are used to store different sets of nodes (or other data) in the compound document. For more information on compound documents, see the OLE2 references.

When loading a node's data with LoadData, the stream need not necessarily be created by a node with the same [structure \[Page 262\]](#) as the node to which the data is being loaded.

During loading, the same mechanism that is used with [Drag and Drop with Tree Views \[Page 274\]](#) works. Any property item in the node being loaded is compared with the item from the loading stream. If the Name and the [Type \[Page 264\]](#) for the target item are the same as those from the loading stream, the target item is assigned the data from the stream. Otherwise the target item remains untouched.

Structures Collection Object

The Structures collection is a [named collection \[Page 147\]](#) of [structure \[Page 262\]](#) objects. Within this named collection the name of an object must be unique. The Structures collection and the Structure object are used to define the content and appearance of the tree control. (See also [Configuring the Tree \[Page 268\]](#) and [Design Mode \[Page 275\]](#)).

[Properties \[Page 256\]](#)

[Methods \[Page 258\]](#)

Structures Collection Properties

The Structures collection object has the following properties:

Structures Collection Properties

Name	Parameter	Type	Description
Count		Long	Returns the number of objects in the collection. Read-only.
Item [Page 257]	Variant valIndex	Object	Returns the Structure object indexed by <i>valIndex</i> . Read-only.
TableKeyIndex		long	Defines the column index in a connected SAP Table object, for the nodes' Key property. (See also Connecting Tree Views and Table Objects [Page 272])
TableForceExpanderIndex		long	Defines the column index in a connected SAP Table object, for the nodes' ForceExpander property. (See also Connecting Tree Views and Table Objects [Page 272])
TableIsFolderIndex		long	Defines the column index in a connected SAP Table object, for the nodes' Type property. If the value is 0, the type is assumed to be a leaf [Page 296] , otherwise it is a folder [Page 295] . (See also Connecting Tree Views and Table Objects [Page 272])
TableExpandedIndex		long	Defines the column index in a connected SAP Table object, for the nodes' Expanded property. (See also Connecting Tree Views and Table Objects [Page 272])

Structures Collection Properties: Item

Purpose

Retrieves a Structure object from a Structures collection object.

Syntax

The Item property has the syntax:

```
Object Item(Variant vaIndex)
```

Description

The Item property returns a [structure \[Page 262\]](#) object. The parameter *val/index* identifies the Structure object to be returned. This may either be the Index or the Name of the Structure object. A further method for retrieving an item is to invoke the item's name directly as a property. (See also [Named Collection \[Page 147\]](#)).



```
Dim Structures As Object  
Set Structures = Tree.Structures  
Structures.Add("ItemA")  
Structures.Add("ItemB")  
Structures.ItemA.Type = TreeStructureText
```

Structures Collection Methods

The Structures collection object has the following methods:

Structures Collection Methods

Name	Parameter	Return Type	Description
Add [Page 259]	Variant vaWhat	Object	Adds a structure [Page 262] object to the collection and returns the new object.
Remove [Page 260]	Variant valIndex	Boolean	Remove the Structure object indexed by <i>valIndex</i> . The meaning of <i>valIndex</i> is the same as with the Item [Page 257] property.
Insert [Page 261]	Variant valIndex Variant vaWhat	Object	Inserts a new structure object in the collection and returns the new object.

Structures Collection Method: Add

Purpose

Adds a Structure object to a Structures collection object.

Syntax

The Add method has the syntax:

```
Object Add(Variant vaWhat)
```

Description

Add may be used in two different variants. If you set *vaWhat* to a string, the Name property of the new Structure object is initialized with that parameter. Otherwise, set the parameter to VT_EMPTY to create a new default-initialized object. The Name property of the object should be initialized in any case, because this name is used to access the [items \[Page 287\]](#) within a [node \[Page 245\]](#) object.

Structures Collection Method: Remove

Structures Collection Method: Remove

Purpose

Removes a Structure object from a Structures collection object.

Syntax

The Remove method has the syntax:

```
Object Remove (Variant vaIndex)
```

Description

Removes the Structure object indexed by *valndex*. The meaning of *valndex* is similar to that of the [Item \[Page 257\]](#) property.

It is **not** possible to remove a Structure object with Type *trvTreeStructureHierarchy*. In order to remove a hierarchy-typed item, the Table Tree object's [Type \[Page 264\]](#) property must be set to *trvTreeTypeLeafs*.

Structures Collection Method: Insert

Purpose

Inserts a Structure object into a Structures collection object.

Syntax

The Insert method has the syntax:

```
Object Insert(Variant vaIndex, Variant vaWhat)
```

Description

Inserts a new structure object into the collection. The *vaWhat* parameter has the same meaning as in the [Add \[Page 259\]](#) method. The *valIndex* parameter is an index describing the position where the object should be inserted. Valndext has the same meaning as the *valIndex* parameter for the [Item \[Page 257\]](#) property.

Structure Object

Structure Object

The Structure object is used to define the content and appearance of the Table Tree control. For example, you can define the kind of data stored in a node, which parts of the data are visible, and how the node appears on screen. Specifically, a Structure object defines a single property item in a tree node. All nodes in the tree have the same set of property items.

Further Table Tree features are described in:

- [Structure Object Properties \[Page 263\]](#)
- [Configuring the Tree \[Page 268\]](#)
- [Connecting Tree Views and Table Objects \[Page 272\]](#)
- [Drag and Drop with Tree Views \[Page 274\]](#)

Structure Object Properties

The Structure object has the following properties:

Structure Object Properties

Name	Type/Parameter	Description
Type [Page 264]	CTreeStructureType	Defines the type of the Structure item.
Alignment [Page 266]	CTreeStructureAlign	Defines the alignment of the structure item.
Width	Short	The width of the structure item in pixel. Only necessary if the Alignment property is not trvTreeAlignAuto.
Name	String	The name of the item. (See also Configuring the Tree [Page 268]).
Hidden [Page 267]	Boolean	Indicates whether the structure item should be displayed.
Index	Long	Returns the index of the object in the structures [Page 255] collection. Read-only.
TableIndex	Long	Returns the index of the corresponding column in the SAP Table object [Page 300] , if the control was added to the SAP Table object's Views collection [Page 370] (see also Connecting Tree Views and Table Objects [Page 272]).
HasFocusRect	Boolean	Indicates whether the structure item contributes to the focus rectangle drawn around the selected Node [Page 288] and items [Page 287] .
ExpandOnDblClick	Boolean	Indicates whether the node is expanded if the item is double-clicked.

Structure Object Property: Type

Purpose

Specifies the type of a Structure object.

Return Value

type CTreeStructureType.

Description

There are three valid values for the Type property:

Type property values

Value	Description	Default Appearance
trvTreeStructureHierarc hy = 0	Defines a hierarchy item. This item does not store any data but returns the Level [Page 297] of a Node [Page 288] . This type may only be used as first structure object in a Structures [Page 255] collection. The name of the item is always forced to be 'Level'. The corresponding node's Level property is always read-only.	 for non-expanded folders [Page 295]  for expanded folders
TreeStructureImage = 1	Defines an image item. A node may contain as many different image items as desired. The corresponding dynamic node property [Page 293] is read/write and may store any variant data which may be converted to an Integer. This value is used as index in the ImageList or may be -1 for the default icons.	

Structure Object Property: Type

TreeStructureText = 2	Defines a text item. A node may contain as many different text items as desired. The corresponding dynamic node property [Page 293] is read/write and may store any variant data which may be converted to a String. Text items are displayed with the font assigned to the node object's font property. If this property is not assigned, the font of the root object [Page 289] is used.	Text <i>Text</i>	Default text depending on the chosen font and type of a node.
-----------------------	--	-----------------------------	---

Structure Object Property: Alignment

Structure Object Property: Alignment

Purpose

Specifies the alignment of a Structure object.

Return Value

type CTreeStructureAlign.

Description

There are four valid values for the Alignment property:

Alignment property values

Value	Description
trvTreeAlignLeft = 0	The item's alignment is left-justified. For hierarchy items, the expander [Page 294] symbol is left-aligned within the node's level.
trvTreeAlignCenter = 1	The item is centered. For hierarchy items, the expander symbol is centered within the node's level.
trvTreeAlignRight = 2	The item's alignment is right-justified. For hierarchy items, the expander symbol is right-aligned within the node's level.
trvTreeAlignAuto = 3	The item is aligned according to the size of the item. For hierarchy and image items, the alignment depends on the width of the corresponding bitmap or icon. For text items, the alignment depends on the text length and the selected font.

Structure Object Property: Hidden

Purpose

Specifies the alignment of a Structure object.

Return Value

type Boolean.

Description

Hidden items are most convenient for storing data that is associated with a node, but should not influence the visual appearance of the node. This avoids additional maps or tables where associated information is stored.

An example might be the hierarchically-ordered staff of a company. While navigating through the hierarchy, perhaps only the staff member's name is of interest. However, after selecting the node, all the information about the staff member is desired. This could be defined over several Structure items: the Structure containing the 'Name' item with a Hidden property set to FALSE. The Structure(s) containing all other information have the Hidden property set to TRUE.

(See also [Configuring the Tree \[Page 268\]](#))

Configuring the Tree

Configuring the Tree

The SAP Table Tree control features a dynamic node structure that lets users store any data desired in a [Node object \[Page 245\]](#).

Node objects can also have any number of properties (called [Item \[Page 287\]](#)s) that characterize the node. These properties constitute the node's internal structure: all nodes in a control have the same structure. (That is, they all have the same set of properties).

The node structure is defined by a series of [structure objects \[Page 262\]](#) maintained in the [Structures \[Page 255\]](#) collection. Each Structure object corresponds to a single property item: for each item, a property is created in each Node object. The name of the Structure object is the same as the name of the property in each node.

Three property items are fixed (pre-defined) for all nodes, and any others are dynamically created. You create a [dynamic node property \[Page 293\]](#) for the control by adding a Structure object to the Structures collection.

The fixed properties are Level, Image, and Name: every tree node has a hierarchy level, a bit image, and a text name. The fixed properties are automatically contained in the Structures collection.

In order to maintain all desired information in a node, you can define some property items as hidden. Hidden items are not displayed: the node is only used as a storage location for the data. Thus it is not necessary to maintain additional tables or arrays to store data associated with a node. This is particularly useful with programming languages like Microsoft Visual Basic where pointers are not part of the language definition.

This configuration also plays an important role if your application uses any drag-and-drop, persistent storage, or clipboard operations. For example, when dragging a node from one control to another, all source control properties having the same Name and [Type \[Page 264\]](#) as items in the destination control will be transported to the destination control.

A similar case applies when the compound stream created by a call to SaveData on a [node \[Page 245\]](#) is used to insert new nodes into a different control. In this case too, all items with the same Name and Type in the target control are fed by data from the stream.



The following VB 4.0 example illustrates the relationship between the configuration of the [Structures \[Page 255\]](#) collection, the data stored in a [node \[Page 245\]](#) object, and the appearance of the tree. The example here maintains a company's personnel hierarchy in a table tree. (The complete example is found under STAFF.VBP and STAFF.FRM on the distribution disks.)

In design mode, SAP Table Tree controls with the name 'StaffTree' were placed on a form and five structure items defined as in the following table.

Structures Collection

Item Index	Item Name	Item Type	Hidden	Alignment
1	Level	trvTreeStructureHierarchy	FALSE	Auto
2	Image	trvTreeStructureImage	FALSE	Auto
3	Name	trvTreeStructureText	FALSE	Auto

Configuring the Tree

4	EmployeeID	trvTreeStructureText	TRUE	don't care
5	Department	trvTreeStructureText	TRUE	don't care

While initializing the control, several nodes are inserted in the control.

Initialization code :

```

Private Sub Form_Load()
Dim RootFolder As Object
Dim Folder As Object
Dim Leaf As Object
Set RootFolder = StaffTree.Nodes.AddEx(_
, trvTreeAddFirstSibling, "StaffHierarchy", _
"My Company", -1, -1, NodeTypeFolder)
RootFolder.EnsureVisible
Set Folder = StaffTree.Nodes.AddEx(_
RootFolder, trvTreeAddFirstChild, "Sales", "Sales", _
-1, -1, trvNodeTypeFolder)
Set Leaf = Folder.Children.AddEx(_
Folder, trvTreeAddFirstChild, "", "Peter", _
-1, -1, trvNodeTypeLeaf)
        ' Set an unique key as combination of name and
        department
Leaf.Key = Leaf.Parent.Key & Leaf.Name
Set Leaf = Folder.Children.AddEx(_
Folder, trvTreeAddFirstChild, "", "Frank", _
-1, -1, trvNodeTypeLeaf)
        ' Set an unique key as combination of name and
        department
Leaf.Key = Leaf.Parent.Key & Leaf.Name
Set Leaf = Folder.Children.AddEx(_
Folder, trvTreeAddFirstChild, "", "Paula", _
-1, -1, trvNodeTypeLeaf)
        ' Set an unique key as combination of name and
        department
Leaf.Key = Leaf.Parent.Key & Leaf.Name
End Sub

```

Configuring the Tree

The *Department* and *EmployeeID* is supplied within the [Table Tree Event: NodeInsert \[Page 232\]](#) event handler:

```

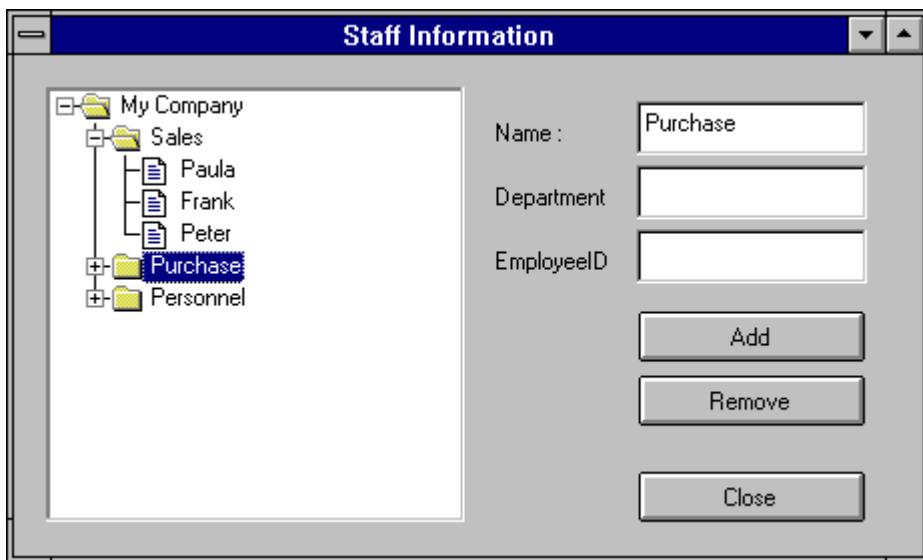
Private Sub StaffTree_NodeInsert(ByVal NewNode As Object)
    Dim Node As Object
    Dim MinID As Integer
    Dim NumPos As Integer
    Dim ActID As Integer
    MinID = 0

    If (Not NewNode.Parent Is Nothing) And _
        ((NewNode.Type And trvNodeTypeLeaf) =
        trvNodeTypeLeaf) Then
        NewNode.Department = NewNode.Parent.Name
        For Each Node In NewNode.Parent.AllChildren
            NumPos = InStr(1, Node.EmployeeID, "#")
            If NumPos > 0 Then
                ActID = Val(Mid$(Node.EmployeeID, NumPos +
                1, Len(Node.EmployeeID)))
                If ActID > MinID Then
                    MinID = ActID
                End If
            End If
        Next Node
        NewNode.EmployeeID = NewNode.Parent.Name & "#" & (MinID + 1)
    End If
End Sub

```

By defining a Structure item 'EmployeeID' and 'Department', the new [dynamic properties \[Page 293\]](#) are parts of the [node \[Page 245\]](#) object. Because the [Hidden \[Page 267\]](#) property of these [items \[Page 287\]](#) equals TRUE, the resulting tree is displayed:

Configuring the Tree



Connecting Tree Views and Table Objects

The SAP Table Tree control becomes an R/3-aware control by connecting it to a SAP Table object. A Table object is part of the Table Factory control.

The connection is defined through the [Structures \[Page 255\]](#) TableIndex properties and the [structure \[Page 262\]](#) TableIndex property. These properties define the column from which the associated values are taken. If a Table Tree object is added to the [Views \[Page 370\]](#) collection of a Table object, a node is inserted for each row in the table. All necessary information is taken from the table to initialize the new nodes.

Special treatment is reserved for the first Structure item's TableIndex. This property is used as a row index in the Table object to define the parent of a node. Consequently, the content of the Table object has to define a completely consistent subtree and may not add any nodes elsewhere.

In addition, the application must define the node that indicates where to insert the new subtree through the Table Tree object's TableInsertRoot property. If this property equals nothing, the subtree is inserted into the root. Otherwise this property must point to an already existing node. In this case, this node is used as parent for the subtree.

Unlike the connection between a Table View control and a Table object, the connection to the Table Tree is only temporary. After the new nodes are inserted, the Table Tree control cancels the connection.



```
Dim oStructs As Object

Set oStructs = oTree.Structs
Set oTree.TableInsertRoot = Nothing
oStructs.TableKeyIndex = 1
oStructs.TableForceExpanderIndex = 2
oStructs.Item(1).TableIndex = 3
oStructs.Item(3).TableIndex = 4

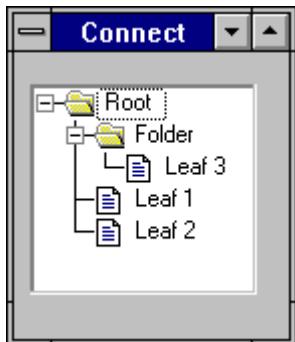
oTab.Views.Add oStructs.Object
...
```

Table Object's Content

“KeyRoot”	1	0	“Root”
“KeyFo1”	1	1	“Folder”
“KeyLe1”	0	1	“Leaf1”
“KeyLe2”	0	1	“Leaf2”
“KeyLe3”	0	2	“Leaf3”

Connecting Tree Views and Table Objects

Tree's appearance:



Drag and Drop with Tree Views

The SAP Table Tree control implements several standard drag-and-drop scenarios. The drag-and-drop behavior is defined through the [DragDrop \[Page 226\]](#) property of the Table Tree object.

Drag and Drop may be disabled, dragging may be enabled for [folders \[Page 295\]](#) and/or [leafs \[Page 296\]](#), dropping may be enabled for folder and/or leafs, or both drag and drop may be enabled for folders and / or leafs. If folders are drag-and-dropped, all sub-folders (that is, all children and grandchildren) are part of the data transport in a control specific format. Additionally all item data for the drag source node is stored in the data transport object in CF_TEXT format.

If drag and drop happens between two SAP Table Tree controls, the control-specific formats ensure enhanced data transfers. All items defined with the same Name and [Type \[Page 249\]](#) in the source control and in the destination control are transported to the destination control. This happens even if the sequence of the items in both controls is different. If the destination control has items that are not part of the data transport object, these items are created with default values.

Drag and Drop operations within one control support move and copy operations. Drag and drop between two controls supports only copy operations as the default implementation.

The data transport scenarios mentioned also work across process boundaries. If the default drag-and-drop implementation is not sufficient, you can program event handlers for the [DragSourceFill \[Page 234\]](#), [DragComplete](#), [DropEnter \[Page 235\]](#) and [Drop \[Page 236\]](#) events that satisfy more sophisticated demands.

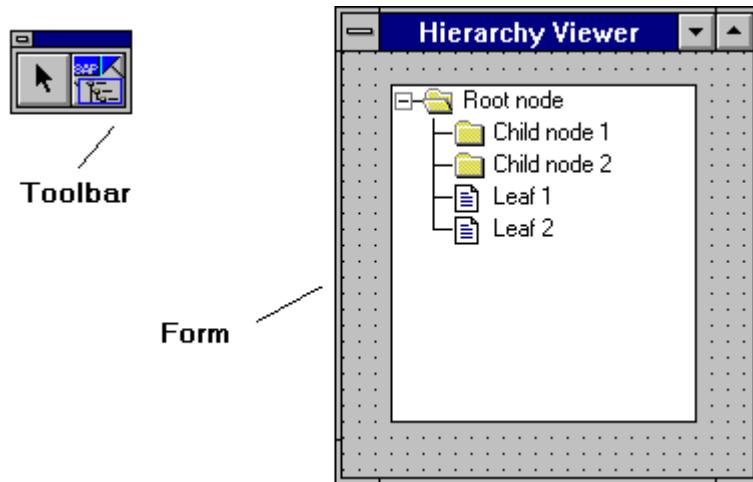
The compound stream created by a call to the SaveData method on a Node object may be used to insert new nodes into a different control. Again, all items with the same Name and Type in the control being loaded are fed by data from the stream.

Design Environment Property Pages

Most container programs and development environments that are OLE-aware support a design environment for designing forms and dialogs interactively. Within this environment, a toolbar window offers a choice of control that the user can place on the form.



The SAP Table Tree control is indicated by . A typical view of the design environment is shown below.



The control's appearance depends on the configuration of the SAP Table Tree control, just as the properties available depend on the container. Some containers add additional properties like Visible, Default, Parent or Cancel. These additional properties are described in the container's manual.

The container also determines whether property are displayed as in Microsoft Visual Basic, or whether they are only available through property pages at the design time. Nevertheless, almost every control container supports a right mouse button menu with entries that invoke the control's property pages. The SAP Table Tree control supports five property pages :

[Table Tree Property Page: General \[Page 276\]](#)

[Table Tree Property Page: Structure \[Page 278\]](#)

[Table Tree Property Page: Events \[Page 280\]](#)

[Table Tree Property Page: Fonts \[Page 281\]](#)

[Table Tree Property Page: Colors \[Page 282\]](#)

[Appearance for Different Configurations \[Page 283\]](#)

Table Tree Property Page: General**Table Tree Property Page: General**

Property	Description	
Structure Items	Enter the number of items in a tree node. Valid values are 1..255 (see also Configuring the Tree [Page 268]).	
Hierarchy Ident	Enter the width of the hierarchy item in pixels. This value is only used if the Alignment [Page 266] property of the hierarchy item is not set to TreeAlignAuto.	
Selection Mode	Choose the selection mode:	
	Single column, line selection	Only an entire node may obtain the focus. All items with an property HasFocusRect set to TRUE are displayed with the window highlight color as a background color.
	Multiple column, line selection	(Not yet supported)
	Single column, item selection	Single items may be selected. Each item with the HasFocusRect property set to TRUE may catch the focus. Only the corresponding item is displayed with the window highlight color as background color. Selection may be moved by using the left and right key.
	Multiple column, item selection	(Not yet supported)
Control Type	Choose the general appearance of the tree:	
	Show folders	Only folders [Page 295] are displayed in the tree. Nevertheless, a node may also contain leaves.
	Show leafs	Only leaves [Page 296] are displayed. (See also Appearance for Different Configurations [Page 283]).
	Show folders and items	Leaves and folders are displayed. Folders are always displayed prior to any leaf [Page 296] .

Table Tree Property Page: General

DragDropMode	Choose the desired mode (see DragDrop [Page 226] property).
---------------------	---

Table Tree Property Page: Structure**Table Tree Property Page: Structure**

The property page Structure defines single Structure items (see also [Configuring the Tree \[Page 268\]](#)). Within this property page the number of Structure items defined in the [Table Tree Property Page: General \[Page 276\]](#) may be configured.

Property	Description
Item	Choose the item you want to configure.
Name	Enter a unique name for the item. A dynamic node property [Page 293] is generated for this name for every node [Page 245] object during runtime. The name may also be empty. Then the item is only accessible through the Data property of the Node object with an appropriate index.
Width	Enter the number of pixels to be used for displaying the item. This value is only used as long as Alignment is not set to Auto.
Hidden	Set Hidden to FALSE if the item should be displayed.
Focus	Set Focus to TRUE if the item should take on the focus. (See also SelectMode property of Table Tree object [Page 220] .)
Expand OnDblClick	Set ExpandOnDblClick to TRUE if the node should be expanded or collapsed when the user invokes a double-click on the item.

Table Tree Property Page: Structure

Alignment	Choose the desired alignment type for the item. If Auto is not used, the item is displayed using the number of pixels defined in the Width property while painting.	
Type	Choose the desired item type:	
	Hierarchy	The item is the hierarchy item and drawn as an hierarchy expander [Page 294] . This type is only disabled but may not be modified or assigned. The hierarchy item may be removed by changing the type property of the Table Tree object to 'Show leafs' in the Table Tree Property Page: General [Page 276] Dialog.
	Image	The item is an image. The corresponding dynamic node property [Page 293] is an index in the ImageList property of the Table Tree object [Page 220] .
	Text	The item is from type text. The corresponding dynamic node property is converted to a VT_VSTR value and displayed when the node is painted.
TableColumnIndex	Enter the column index of a connected SAP Table control. (See also Connecting Tree Views and Table Objects [Page 272])	



The hierarchy type is only allowed for the first [Item \[Page 287\]](#). If a hierarchy item is used, the name is forced to 'Level'. The according node creates a dynamic 'Level' property which returns the level of the according node within the entire tree. This property is always read-only. It is not possible to define more than one item of type hierarchy. If the hierarchy type is omitted by modifying the Table Tree object's [Type \[Page 249\]](#) property, the general behavior of the control changes (see [Appearance for Different Configurations \[Page 283\]](#)).

Table Tree Property Page: Events

Table Tree Property Page: Events

This property page is used to enable or disable events fired to the container at runtime. The main aspect of disabling events is for performance reasons. More sophisticated scenarios will change this property through the Events property of the Table Tree object on demand during runtime.

Table Tree Property Page: Fonts

This property page allows the user to define the default font [\[Page 152\]](#) used by the control. Usually this is the default font assigned to the entire form by the control container.

Table Tree Property Page: Colors**Table Tree Property Page: Colors**

Information about property page colors will be available in further releases.

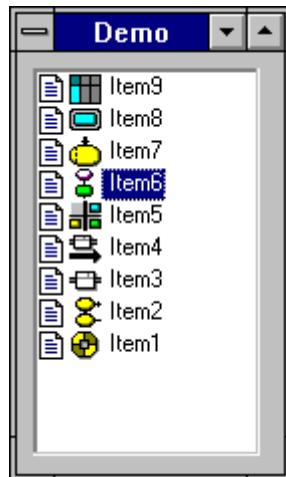
Appearance for Different Configurations

The SAP Table Tree control may not only be used as a tree control: it can also act as an image list or a check list (not implemented yet). This results when you define a [structure \[Page 262\]](#) in the [Structures \[Page 255\]](#) collection that has no hierarchy item. You omit hierarchy [items \[Page 287\]](#) by defining the control type as 'Leafs only' in design mode or by assigning the Type property a value of TreeTypeLeafs. All methods and properties for the control work similarly, with the exception of the Add and AddEx methods of the Nodes collection.

Without having defined a hierarchy typed item, it is not possible to insert any child nodes and nodes of type [Folder \[Page 295\]](#). The functionality is reduced to one Nodes collection (the Table Tree object's [Nodes collection \[Page 237\]](#)) that contains only [leafs \[Page 296\]](#).

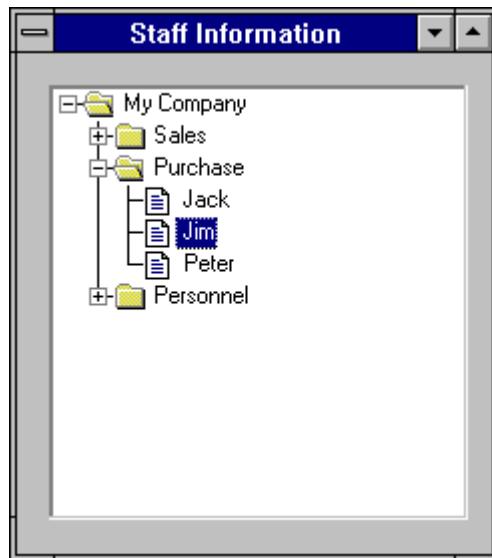


A control configured without hierarchy typed item, two image typed items and a text item would appear as follows:

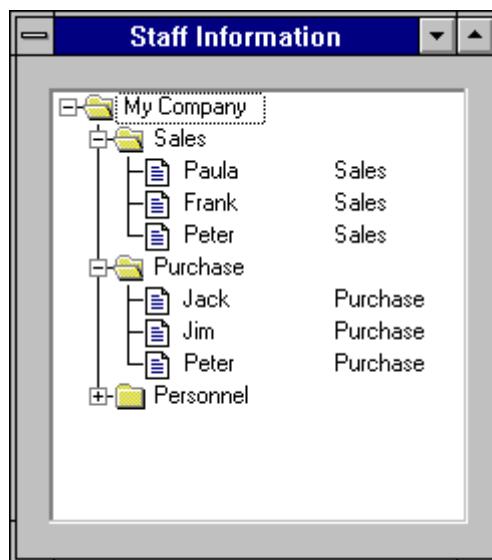


A control configured with hierarchy typed item, one image typed item and a text item would appear as follows:

Appearance for Different Configurations



A control configured with hierarchy typed item, one image typed item, two text items and single item selection would appear as follows:



Code Examples

You can find Visual Basic examples of how to use the Table Tree OCX in your installation directory. Look for the SAPGUI directory, and within it, find the subdirectories:

`rfcsdk/ocx/demo`

Within the `rfcsdk/ocx/demo` directory, you find Visual Basic project files whose forms demonstrate the following techniques:

Visual Basic Examples

Technique	VB Project Name
First Steps	<code>1ststept.vbp</code>
Item Definition	<code>staff.vbp</code>
Duplicated Key	<code>DupKey.vbp</code>
Drag and Drop	<code>DragDrop.vbp</code>
Table Connection	<code>Connect.vbp</code>

Glossary

The following glossary terms are described:

[Item \[Page 287\]](#)

[Node \[Page 288\]](#)

[Root Control/Root \[Page 289\]](#)

[Root Node \[Page 290\]](#)

[Nothing \[Page 291\]](#)

[CreateObject \[Page 292\]](#)

[Dynamic Node Properties \[Page 293\]](#)

[Hierarchy Expander/Expander Symbol \[Page 294\]](#)

[Folder \[Page 295\]](#)

[Leaf \[Page 296\]](#)

[Level \[Page 297\]](#)

[Collection \[Page 298\]](#)

Item

An item (or property item) describes a single characteristic of a [node \[Page 245\]](#) object. Items are defined by [structure \[Page 262\]](#) objects in the [Structures \[Page 255\]](#) collection. Since the collection of Structure objects is the same for all nodes in a tree, all nodes have the same set of characteristic properties.

Some property items are pre-defined (for example, Level, Image, and Name), but all others are dynamically defined. A node may contain up to 255 items, which allows a highly flexible configuration of the SAP Table Tree control. The entire appearance of the control and the data stored in the control is configured by the [Structures \[Page 255\]](#) collection.

Node

Node

A Node object is single entry in the tree. A node's internal structure is defined by its property [items \[Page 287\]](#). Property items contain the data and description for each node. (See [Configuring the Tree \[Page 268\]](#)).

Root Control/Root

The root control is the highest level object in the object hierarchy. It is accessible through the Table Tree object returned by [CreateObject \[Page 292\]](#).

Root Node

Root Node

A root node is a [node \[Page 245\]](#) located at the highest level of the tree's hierarchy. A root node has no parent node and is part of the root control's (i.e., the Table Tree object's) [Nodes \[Page 237\]](#) collection.

Nothing

VBA key word for an empty object. This value is the same as NULL in C++ or nil in Pascal.

CreateObject

CreateObject

VBA function to create an OLE object. See VBA help for more information

Dynamic Node Properties

Dynamic Node Properties represent properties that are created dynamically at runtime. These properties are created for each node, as specified by the Structure objects that define each property. These properties are not to be seen in an object browser and have to be defined by the user.

See also:

- [Configuring the Tree \[Page 268\]](#)
- [Structures \[Page 255\]](#) collection object
- [structure \[Page 262\]](#) object

Hierarchy Expander/Expander Symbol

Hierarchy Expander/Expander Symbol

The hierarchy expander looks like  for non-expanded folders and  for expanded folders. If a node does not have any children, no expander is drawn unless the ForceExpander property is set to TRUE.

Folder

A folder is a node object which is able to contain further nodes in its Children collection. The default icons for a folder are as follows:

Closed folder	Expanded property equals FALSE.	
Open folder	Expanded property equals TRUE.	
Disabled folder	TreeNodeDisabled bit is set in the node's Type [Page 249] property.	

Leaf

A leaf is a node object which is not able to contain further nodes in its Children collection.
The default icons for a leaf are as follows:

Enabled leaf	TreeNodeDisabled bit is cleared in the node's set in the node's Type [Page 249] property.	
Disabled leaf	TreeNodeDisabled bit is set in the node's Type property.	

Level

The Level is a node property (item) that is automatically created if a [structure \[Page 262\]](#) object of type TreeStructureHierarchy exists in the [Structures \[Page 255\]](#) collection. The Level property is always read-only and returns the hierarchy level of a node as a long integer. A root node's Level equals 0.

Collection

Collection

A collection is a summary of objects. A collection usually supports methods like Item, Add, Insert and Remove (see [SAP Standard Collection \[Page 145\]](#)). You can iterate through collection objects by using `For ... Each` loops in Visual Basic or the IEnumVARIANT interface in C++ (see [AllChildren \[Page 251\]](#)).

Pre-Defined Images

The following images are stored in the SAP Table Tree control with the index listed on the right hand side. If no image list is assigned to the ImageList property of the Table Tree object, these images are displayed. Which image is to be displayed in each node is determined in the node's dynamic image typed properties.



The Table Factory Control

The Table Factory Control

This section contains the following topics:

Programming with the Table Factory

[Introduction \[Page 302\]](#)

[Table Factory Object Hierarchy \[Page 303\]](#)

Control and Object Reference

[Table Factory Object \[Page 305\]](#)

[Tables Collection Object \[Page 309\]](#)

[Table Object \[Page 316\]](#)

[RFCTableParameter Object \[Page 338\]](#)

[Rows Collection Object \[Page 340\]](#)

[Row Object \[Page 347\]](#)

[Columns Collection Object \[Page 350\]](#)

[Column Object \[Page 358\]](#)

[Ranges Collection Object \[Page 362\]](#)

[Range Object \[Page 366\]](#)

[Views Collection Object \[Page 370\]](#)

[Matrix Object \[Page 374\]](#)

Programming Guide

[How to Connect Views to a Table \[Page 373\]](#)

[Creating a Table Object \[Page 334\]](#)

[Using SelectTo* Methods \[Page 335\]](#)

[Displaying and Navigating Table Data \[Page 337\]](#)

[Code Examples \[Page 375\]](#)

[Glossary \[Page 383\]](#)

Introduction

Introduction

The SAP Table Factory control encapsulates internal table (ITAB) handling as provided by the RFC Library. It eases the way the desktop programmer can use internal tables.

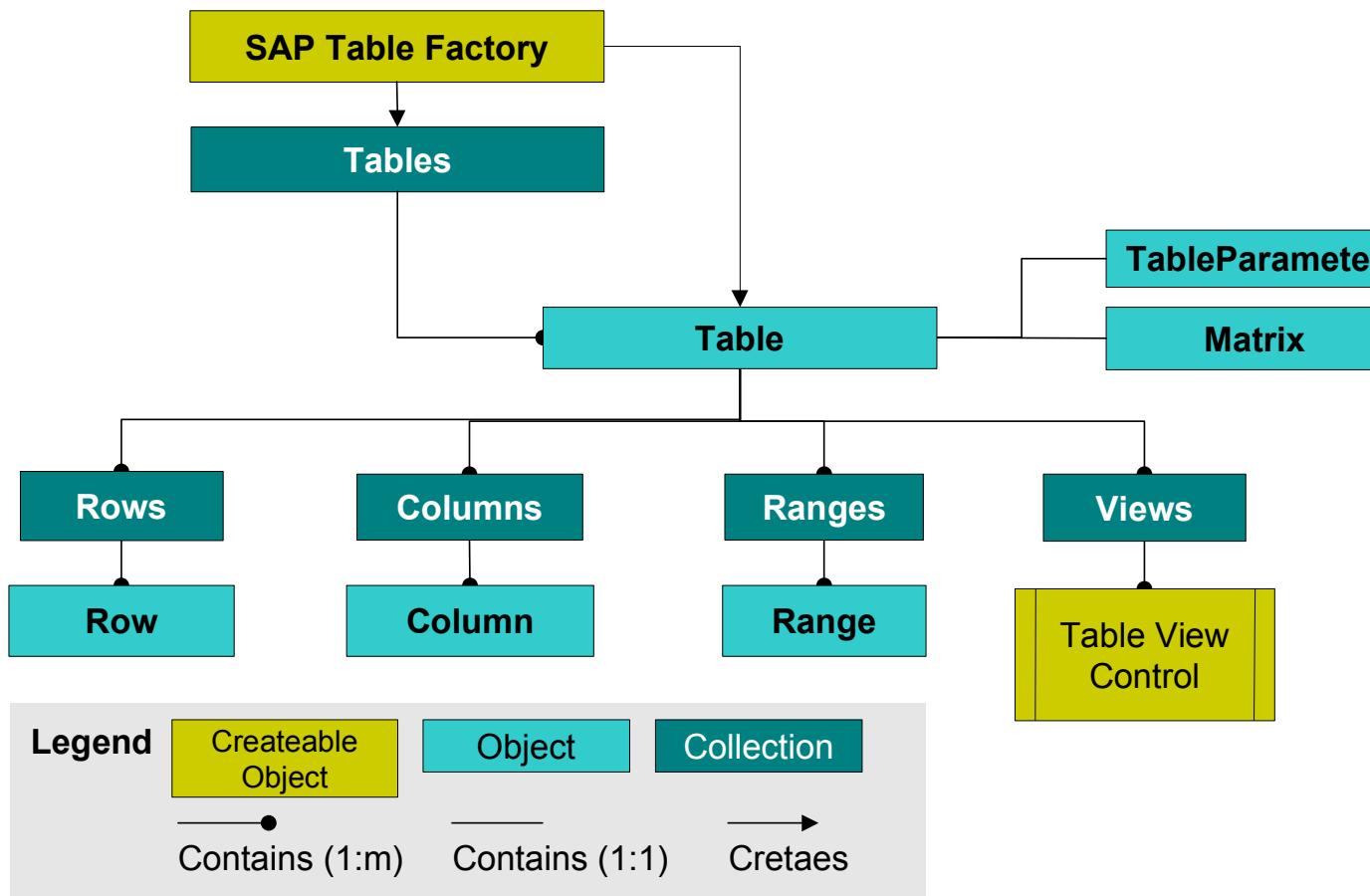
The SAP Table Factory is designed to work optimally with Visual Basic 4.0 (3.0), VBA and C++ through provided wrapper classes. Future releases will also support dual interface in the 32-Bit version.

This version of the component can be created by adding custom controls in Visual Basic (VB) 4.0 and Visual C (VC) 4.0 or by the following line of code:

```
Dim oTableFactoryCtrl as Object  
Set oTableFactoryCtrl = CreateObject ("SAP.TableFactory.1")
```

This code creates the highest level object in the control, the Table Factory object.

Table Factory Object Hierarchy



Using the Table Factory Control

Each Function object owns a collection of R/3 tables. The Table Factory control gives you easy access to these SAP internal tables. If used from within the Function control, the Table Factory is invisible to the user. The Table Factory control provides a Tables collection object that is used as a property in the Function object. When you access the function's Tables collection, Visual BasicTM gets a "dispatch pointer" to the Table control. In the previous example, the *RFC_CUSTOMER_GET* call returns an internal table with the name "CUSTOMER_T". To get the last row of that table and display the ZIP code, for example, use the following code:

```
' Get table.  
  
Set customers = func.Tables ("CUSTOMER_T")  
' Get zipcode of last row.  
  
zipCode = customers (customers.RowCount, "PSTLZ")
```

Table Factory Object

The Table Factory control object is the highest object in the control's hierarchy. You obtain a Table Factory object by calling [CreateObject \[Page 292\]](#) in VBA or by inserting a Table Factory control directly into a form from a toolbar.

[Properties \[Page 306\]](#)

[Methods \[Page 307\]](#)

[Events \[Page 308\]](#)

Table Factory Properties

Table Factory Properties

The Table Factory object has the following properties:

Table Factory Properties

Name	Type	Description
Events	Long	Enables or disables events [Page 308] fired by the Table Factory control.

Table Factory Methods

Table Factory Methods

The Table Factory object has the following methods:

Table Factory Methods

Name	Return Type	Description
NewTable	Object	Creates and returns a new Table [Page 316] object.
NewTables	Object	Creates and returns a new Tables [Page 309] collection object.
AboutBox		Displays the AboutBox dialog.
NewStructure	Object	Creates and returns a new object of type Structure [Page 184] .

Table Factory Events

Table Factory Events

The Table Factory object has the following events:

Name	Parameter(s)	Description
Error	Object	An error occurred in one of the objects.

Events or groups of events may be enabled or disabled by setting bits in the events string returned by the Events property. Disabling events may lead to improved performance, especially for operations that manipulate large chunks of data. Events may also be turned on and off temporarily. The following values are available for the events property:

ttaDisableAllTableEvents = 0	Disable all events
ttaEnableOnTableCreate = 1	Fire event after associated table is created
ttaEnableOnTableClear = 2	Fire event after associated table is cleared
ttaEnableOnTableDelete = 4	Fire event after associated table is deleted
ttaEnableOnTableDataChange = 8	Fire event after any data has changed
ttaEnableOnRowInsert = 16	Fire event after a row is inserted
ttaEnableOnRowRemove = 32	Fire event after a row is removed
ttaEnableOnColumnInsert = 32	Fire event after a column is inserted
ttaEnableOnColumnRemove = 64	Fire event after a column is removed
ttaEnableAllTableEvents = 32767	Enable all events

All values may be combined through **and** or **or** operations.



```

Rem Enable TableCreate and TableInsert event
MyControl.Events = MyControl.Events or _
    ttaEnableOnTableCreate or ttaEnableOnTableDelete

Rem Disable RowInsert and RowRemove event
MyControl.Events = MyControl.Events and not _
    (ttaEnableOnRowInsert or ttaEnableOnRowRemove)

```

Tables Collection Object

The Tables collection is a collection of Table objects. It is implemented as a standard collection like many other SAP active control collections.

[Properties \[Page 310\]](#)

[Methods \[Page 311\]](#)

Tables Collection Properties

The Tables collection object has the following properties:

Tables Collection Properties

Name	Parameters	Type	Description
Count		Long	Returns the number of objects in the collection. Read-only.
Item [Page 287]	VARIANT vaWhichItem	DISPATCH	Retrieves a Table [Page 316] object from the table collection. Read-only.

Tables Collection Methods

The Tables collection object has the following properties:

Tables Collection Methods

Name	Parameters	Description
Remove [Page 312]	VARIANT vaWhat	Removes a Table [Page 316] object from the collection.
RemoveAll		Removes all Table objects from the collection.
Add [Page 315]	VARIANT vaWhat	Adds a new table to the collection.
Insert	VARIANT vaWhichItem VARIANT vaWhat	Inserts a Table object at a given position in the collection.
Unload [Page 313]	VARIANT vaWhat	Unloads a table from the collection.

Tables Collection Method: Remove

Purpose

Removes a table from the Tables collection.

Syntax

The Remove method has the syntax:

Remove (VARIANT vaWhat)

Description

The parameter vaWhat describes the element to be removed. Legal variant types for vaWhat are VT_DISPATCH, VT_BSTR or any data type that can be converted to an integer value.

If vaWhat has type VT_DISPATCH, the corresponding object is searched in the collection and removed. If vaWhat has type VT_BSTR, the first table with that name is removed. Otherwise vaWhat is converted to an index and the corresponding table is removed.



When removing a table from the Tables collection, the Table object becomes invalid. Any further attempt to work on the object returns an invalid object exception. Use the [Unload \[Page 313\]](#) method to remove a Table object without invalidating it.

Tables Collection Method: Unload

Purpose

Unloads a table from the Tables collection.

Syntax

The Unload method has the syntax:

Unload (VARIANT vaWhat)

Description

The parameter vaWhat describes the element to be unloaded. The return value is the unloaded Table object. This method enables the programmer to remove a table from the Tables collection without invalidating it.

Tables Collection Method: Item

Tables Collection Method: Item

Purpose

Retrieves a table from the Tables collection.

Syntax

The Item method has the syntax:

Item (VARIANT vaWhichItem)

Description

The parameter vaWhichItem describes the element to be returned. The following variant data types for arWhichItem are legal:

Type	Description
VT_BSTR, VT_LPCSTR, VT_LPWSTR VT_BSTR *, VT_LPCST *, VT_LPWSTR *	The first table with name vaWhatItem is returned. If the name is not found, vaWhichItem is converted to an integer value which is used as index in the collection.
Any type convertible to a VT_I4	The parameter vaWhichItem is converted to an integer value and used as index in the collection.

Tables Collection Method: Add

Purpose

Adds a new table to the collection.

Syntax

The Add method has the syntax:

Add (VARIANT vaWhat)

Description

Set the type for the vaWhat parameter to:

- VT_EMPTY, to create a new table
- VT_BSTR, to create a new table with the parameter name vaWhat
- VT_DISPATCH, to add the Table object to the end of the collection

Table Object

Table Object

The Table object is the key object in the SAP Table Factory. It maintains an internal table (ITAB) passed in from the native RFC interface. The Table object actually handles data read from or written to the R/3 System, providing a full two-dimensional view of this data.

The same Table object can contain different internal tables, as long as they have the same structure. (See [AttachHandle \[Page 326\]](#) and [DetachHandle \[Page 327\]](#)).

Available information on Table objects:

[Properties \[Page 317\]](#)

[Methods \[Page 319\]](#)

Other programming information:

[Creating a Table Object \[Page 334\]](#)

[Using SelectTo* Methods \[Page 335\]](#)

[Displaying and Navigating Table Data \[Page 337\]](#)

Table Object Properties

Table Object Properties

A Table object has the following properties:

Table Object Properties

Name	Parameters	Type	Description
RowCount		Long	Returns the number of Rows contained in the table. Read-only.
ColumnCount		Long	Returns the number of Columns contained in the table's Columns collection. Read-only.
RfcParameter		Object	Returns a RFCTableParameter [Page 338] object. Read-only
Rows		Object	Returns a Row [Page 347] object. Read-only
Columns		Object	Returns a Column [Page 358] object. Read Only
Ranges		Object	Returns an object of type Ranges [Page 366] . Read-only.
Views		Object	Returns an object of type Views. Read-only
Data [Page 318]		Array of VARIANT	Returns the Table object in the form of an array.
Value	long RowIndex VARIANT ColumnIndex	VARIANT	Access a single value in the table.

Table Property: Data

Table Property: Data

Purpose

Accesses (sets or gets) Table object data as if it were an array.

Description

The Data property copies data quickly between an array and a Table object. For example, this property can be used to fill an Excel Range or as a return value of a Visual Basic function. Used as a get operation, the Data property gets the data from a Table object and returns it in safe-array form:

```
MyArray = MyTable.Data
```

Used as a set operation, the Data property takes data from an array and loads it in the Table object:

```
MyTable.Data = MyArray
```

If you use the Data property as a set operation, and the provided array contains more rows than the table actually has, the table is resized automatically. The table is always filled starting in cell (1,1). If you want to start with a different left upper corner, use the [Range \[Page\] 366](#) object.



If you copy an array to a Table object, and the array has more columns than the table, Column objects will not be added automatically to the Columns collection.

Table Object Methods

Table Object Methods

A Table object has the following methods:

Table Object Methods

Name	Parameter(s)	Return Type	Description
CreateFromR3Repository [Page 322]	VARIANT R3Connection String RepositoryEntry String ParameterName	Boolean	This methods defines a column structure for a Table object by reading all information from the R/3 repository.
CreateFromHandle [Page 323]	VARIANT R3Connection Long TableHandle String RepositoryEntry String ParameterName	Boolean	This methods defines a column structure for a Table object by reading all information from the R/3 Repository and associating the table handle with it.
CreateFromTable [Page 324]	Object Table	Boolean	This method defines a column structure for a Table object by copying the column structure of the input Table object.
Create [Page 325]	String ParameterName Long TableLineLength	Boolean	This method defines a single-column structure for a Table object. The column has width TableLineLength.
AttachHandle [Page 326]	Long TableHandle	Boolean	This method attaches a TableHandle (H_ITAB) to a Table object.
DetachHandle [Page 327]		Long	This method detaches the TableHandle from the Table object. The Table object structure is still valid.
Refresh [Page 328]	void	void	Forces an internal state update.

Table Object Methods

FreeTable	void	void	Frees all data in the Table.
DeleteTable	void	void	Deletes the table data and structure.
SelectToMatrix [Page 329]	VARIANT RowVector Long RowAssocIndex VARIANT ColumnVector Long ColumnAssocIndex Long DataAssocIndex	Object	This method returns a Matrix [Page 374] object containing a two-dimensional view of one column of the table.
SelectMatrixToNumber [Page 330]	VARIANT RowVector Long RowAssocIndex VARIANT ColumnVector Long ColumnAssocIndex Long DataAssocIndex Long Decimals	Object	This method returns a Matrix [Page 374] object containing a two-dimensional view of one column of the table. It also tries to convert the result into numbers.
SelectToVector [Page 331]	VARIANT RowVector Long RowAssocIndex Long DataAssocIndex	Object	This method returns a Matrix [Page 374] object using one input dimension.

Table Object Methods

SelectVectorToNumber [Page 332]	VARIANT RowVector Long RowAssocIndex Long DataAssocIndex Long Decimals	Object	This method returns a Matrix [Page 374] object using one input dimension. It also tries to convert the result into numbers.
BuildTiledRanges [Page 333]	Long Size	Object	Returns a Range [Page 362] collection whose ranges together completely span the table.

Table Method: CreateFromR3Repository

Table Method: CreateFromR3Repository

Purpose

Defines the column structure for a Table object using R/3 Repository information.

Syntax

The CreateFromR3Repository method has the syntax:

```
CreateFromR3Repository (R3Connection, RepositoryEntry, ParameterName)
```

Parameters

- *R3Connection*: Can be a Connection object as provided by the [Logon control \[Page 458\]](#), or an RfcHandle as used by the native RFC API.
- *RepositoryEntry*: Name of the structure in the SAP R/3 Repository.
- *ParameterName*: Formal parameter name declared for the table in the function module interface (in the R/3 Function Library). This name is used to identify the table in a Tables collection.

Description

This method returns TRUE on success, otherwise FALSE.

See also [Creating a Table Object \[Page 334\]](#).

Table Method: CreateFromHandle

Purpose

Defines the column structure for a Table object using R/3 Repository information, and associates an R/3 internal table handle with the object.

Syntax

The CreateFromHandle method has the syntax:

```
CreateFromHandle(R3Connection, TableHandle, RepositoryEntry,  
                  ParameterName)
```

Parameters

- *R3Connection*: Can be a Connection object as provided by the [Logon control \[Page 458\]](#), or an RfcHandle as used by the native RFC API.
- *TableHandle*: the ITAB handle as it used by the native RFC API.
- *RepositoryEntry*: Name of the structure in the SAP R/3 Repository.
- *ParameterName*: Formal parameter name declared for the table in the function module interface (in the R/3 Function Library). This name is used to identify the table in a Tables collection.

Description

This method:

- creates the column structure for a Table object by reading information from the R/3 repository
- associates the provided ITAB handle with the Table object

The CreateFromHandle method lets you use Table objects with internal tables provided by foreign applications. The method returns TRUE on success, otherwise FALSE.

See also [Creating a Table Object \[Page 334\]](#).

Table Method: CreateFromTable

Table Method: CreateFromTable

Purpose

Defines the column structure for a table by copying the structure of an input Table object.

Syntax

The CreateFromTable method has the syntax:

```
CreateFromTable(Table)
```

Description

Use this method to copy the input table's column structure and use it with the Table object. CreateFromTable returns TRUE on success, otherwise FALSE.

See also [Creating a Table Object \[Page 334\]](#).

Table Method: Create

Purpose

Defines a single-column structure for a Table object.

Syntax

The Create method has the syntax:

```
Create (TableName, TableLength)
```

Description

This method defines a column structure with a single column for the Table object. The TableLength parameter specifies the overall length of each table row. You can then add columns to the table object using the Add method in the Columns collection object.

The method returns TRUE on success, otherwise FALSE.

See also [Creating a Table Object \[Page 334\]](#).

Table Method: AttachHandle

Table Method: AttachHandle

Purpose

Attaches a table handle to the current Table object.

Syntax

The AttachHandle method has the syntax:

```
AttachHandle(TableHandle)
```

Description

All old data is lost and the old table is destroyed, but the old structure is used. This enables the use of the Table object for tables which are created by other libraries or applications.

Table Method: DetachHandle

Purpose

Detaches the ITAB handle from the Table object

Description

This method does the opposite of the AttachHandle method: invalidates all associated objects (like Rows) and returns the ITAB handle.

Table Method: Refresh

Table Method: Refresh

Purpose

Refreshes the internal status of a Table object.

Description

Use the Refresh method to force an update of a Table object's internal state. Normally, you call this method after making any RFC calls that update Table object data. The Refresh method is primarily useful if there is a view connected to the Table object, and the view must be notified of a data change, so it can update its display.

Table Method: SelectToMatrix

Purpose

Returns a Matrix object containing a two-dimensional view of one column of the table.

Syntax

The SelectToMatrix method has the syntax:

```
SelectToMatrix (RowVector, RowAssocIndex, ColumnVector,  
                ColumnAssocIndex, DataAssocIndex)
```

Parameters

- *RowVector*: Array of values for the first dimension of the returned matrix.
- *RowAssocIndex*: Index of the table column with which RowVector values are to be compared.
- *ColumnVector*: Array of values for the second dimension of the returned matrix.
- *ColumnAssocIndex*: Index of the table column with which the ColumnVector values are to be compared.
- *DataAssocIndex*: Index of the table column from which the desired data is to be taken.

Description

The SelectToMatrix method selects data from the Table object based on selection criteria applied to two columns of the table. For each row where the two columns match the selection criteria, data values from a third column are collected. The data collected is then returned as a two-dimensional Matrix object, using the selection criteria as values for the row and column dimensions.

For more information on this process, see [Using SelectTo* Methods \[Page 335\]](#).

Related Information

[SelectMatrixToNumber \[Page 330\]](#)

[SelectToVector \[Page 331\]](#)

[SelectVectorToNumber \[Page 332\]](#)

Table Method: SelectMatrixToNumber

Table Method: SelectMatrixToNumber

Purpose

Returns an Matrix object containing a two-dimensional view of one column of the table.

Syntax

The SelectMatrixToNumber method has the syntax:

```
SelectMatrixToNumber (RowVector, RowAssocIndex, ColumnVector,  
ColumnAssocIndex, DataAssocIndex, Decimals)
```

Parameters

- *RowVector*: Array of values for the first dimension of the returned matrix.
- *RowAssocIndex*: Index of the table column with which RowVector values are to be compared.
- *ColumnVector*: Array of values for the second dimension of the returned matrix.
- *ColumnAssocIndex*: Index of the table column with which the ColumnVector values are to be compared.
- *DataAssocIndex*: Index of the table column from which the desired data is to be taken.
- *Decimals*: Type of conversion to be used on the data. A value of 0 means a conversion to long; a value greater than 0 means a conversion to double.

Description

SelectMatrixToNumber is like [SelectToMatrix \[Page 329\]](#), except that it tries to convert the resulting values into integer and float numbers. If the original data is not numeric, this method produces meaningless data.

SelectMatrixToNumber is useful because the RFC Library tends to use TYP/C and most RFC tables use the NUM/C, writing out packed datatypes. On the front-end, however, integer and float values are more appropriate.

For more information about the selection process, see [Using SelectTo* Methods \[Page 335\]](#).

Related Information

[SelectToMatrix \[Page 329\]](#)

[SelectToVector \[Page 331\]](#)

[SelectVectorToNumber \[Page 332\]](#)

Table Method: SelectToVector

Purpose

Returns a Matrix object containing one vector of data from the Table object.

Syntax

The SelectToVector method has the syntax:

```
SelectToVector (InputData, InputAssocIndex, DataAssocIndex)
```

Parameters

- *InputData*: Array of values for the row or column dimension of the returned matrix
- *InputAssocIndex*: Index of the table column with which the InputData values are to be compared.
- *DataAssocIndex*: Index of the table column from which the desired data is to be taken.

Description

The difference between this method and the SelectToMatrix method is that the selection criteria are applied to only one column of the Table object.

The InputData parameter contains the selection criteria, however, in a two-dimensional SafeArray. This means the calling program should place the selection values in either a single row of InputData, or a single column. When you do this, the resulting matrix has:

- n-rows and one column, if InputData contains multiple row values for a column
- one row and n-columns, if InputData contains multiple column values for a row

The SelectToVector method is implemented in this way to provide compatibility with data structures imported and exported by many desktop applications such as Microsoft Excel.

If you want to apply two dimensions of selection criteria to a matrix selection, use [SelectToMatrix \[Page 329\]](#) or [SelectMatrixToNumber \[Page 330\]](#).

For more information on the selection process, see [Using SelectTo* Methods \[Page 335\]](#).

Related Information

[SelectToMatrix \[Page 329\]](#)

[SelectMatrixToNumber \[Page 330\]](#)

[SelectVectorToNumber \[Page 332\]](#)

Table Method: SelectVectorToNumber

Table Method: SelectVectorToNumber

Purpose

Returns an Matrix object containing a two-dimensional view of one column of the table.

Syntax

The SelectVectorToNumber method has the syntax:

```
SelectVectorToNumber (InputData, InputAssocIndex, DataAssocIndex,  
Decimals)
```

Parameters

- *InputData*: Array of values for the row or column dimension of the returned matrix
- *InputAssocIndex*: Index of the table column with which the InputData values are to be compared.
- *DataAssocIndex*: Index of the table column from which the desired data is to be taken.
- *Decimals*: Type of conversion to be used on the data. A value of 0 means a conversion to long; a value greater than 0 means a conversion to double.

Description

SelectVectorToNumber is like [SelectToVector \[Page 331\]](#), except that it tries to convert the resulting values into integer and float numbers. If the original data is not numeric, SelectVectorToNumber produces meaningless data.

This method is useful because the RFC Library tends to use TYPCL and most RFC tables use the NUMC, writing out packed datatypes. On the front-end, however, integer and float values are more appropriate.

For more information about the selection process generally, see [Using SelectTo* Methods \[Page 335\]](#).

Related Information

[SelectToVector \[Page 331\]](#)

[SelectToMatrix \[Page 329\]](#)

[SelectMatrixToNumber \[Page 330\]](#)

Table Method: BuildTiledRanges

Purpose

Creates a series of Range objects that together span all rows of a table.

Syntax

The BuildTiledRanges method has the syntax:

BuildTiledRanges (Size)

Parameters

Size: Number of rows to be placed in each range

Description

This method provides an easy way to build a collection of “tiled ranges” that together completely cover the table. This method is useful for adjusting table data access according to the displayed data in your user interface.

The Size parameter indicates the number of rows each resulting [Range \[Page 366\]](#) object is to contain.

Creating a Table Object

Creating a Table Object

You create a Table object by using either the Add method (in the Tables collection) or the NewTable method (in the TableFactory object). However, these methods do not define the structure for the table. You can define the structure in one of the following ways:

- By structuring the table explicitly:

Use the [Create \[Page 325\]](#) method. You provide the table name and the internal length of one row. In order to build a structure, just add new columns to the Table object. (You do this with the Columns collection)

- By using the R/3 System:

- Create a new table using method [CreateFromR3Repository \[Page 322\]](#).
- Create the structure for a given ITAB handle by using method [CreateFromHandle \[Page 323\]](#).

Both methods use an RFC connection to obtain the actual structure of a table in the R/3 System. The first parameter for these functions is the RFC connection: either a Connection object (as produced by the [SAP Logon Control \[Page 458\]](#)), or a native RFC handle (a parameter of type Long). C/C++ programmers can use the VisualRFCService Handle by using a VARIANT of type CY, with the application handle in the lower part and the Login ID in the upper part of that structure.

- By copying the structure from another Table object

Use the method [CreateFromTable \[Page 324\]](#) to do this.

After creating the Table's structure, you can always change it by adding, removing and changing the column description. However, these changes must never exceed the row length originally defined by the Create methods.

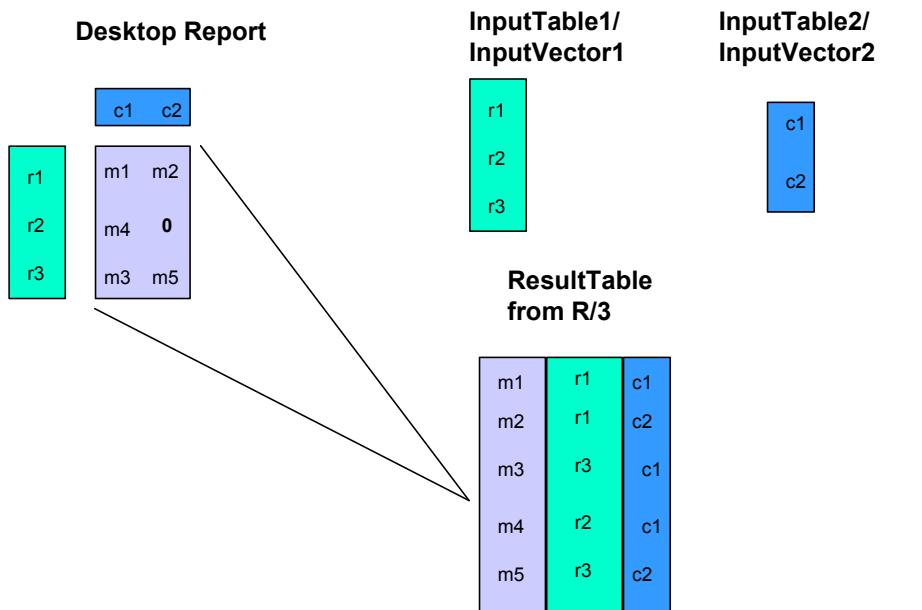
This dynamic behavior of the [Columns \[Page 350\]](#) allows RFC-enabled programs to use black box (generic) tables. An example is a function module that passes back information in a single-column table and provides information about the table's structure in a second table. This approach enables the use of RFC functions that can return different types of tables based on the values of input parameters.

Using SelectTo* Methods

The *SelectTo** methods provide a special view of the table (like a dejoin). These methods return all table data that satisfies value specifications for certain table dimensions.

In the example below, the *SelectTo* method returns a two-dimensional view of one column of a table. Suppose the desktop application requests data from a particular table column (the “desired-data” column), where the values to be selected depend on values in two other (“key-value”) columns of the table. The desktop program specifies the values in the “key-value” columns when calling the *SelectTo* method:

- the “key-value” columns are *InputVector1* and *InputVector2*
- the “ResultTable from R/3” shows the original table, as transferred from R/3
 - the “key-value” columns contain r-values {r1, r2, r3} and c-values {c1, c2}.
 - the “desired-data” column contains m-values {m1, m2, m3, m4, m5}
- the “Desktop Report” shows the two-dimensional matrix of m-values created by the *SelectTo* method



MatrixSelection with Rowvector= InputVector1, columnVector= InputVector2, dataColumn = 1

Calling programs can request data by specifying either one or two input vectors. To specify values using:

- two input vectors, use [SelectToMatrix \[Page 329\]](#) or [SelectMatrixToNumber \[Page 330\]](#)

Using SelectTo* Methods

- one input vector, use [SelectToVector \[Page 331\]](#) or [SelectVectorToNumber \[Page 332\]](#)

Displaying and Navigating Table Data

Table objects have many features. You can:

- Use the [Ranges \[Page 362\]](#) property to define special views over a Table object (for example, only the first 20 rows in a table, or one of the last two columns).
- Use additional controls together with the Table Factory to automate access to R/3 table structures or hierarchical data:
 - The [The Table View Control \[Page 389\]](#) provides functions for displaying and navigating R/3 table data by row, column or cell.
 - The [The Table Tree Control \[Page 216\]](#) provides special functions for displaying hierarchical data as packed into an R/3 table.

The table view and table tree controls provide R/3-specialized access routines when you connect them to the Table object's [Views \[Page 370\]](#) collection.

RFCTableParameter Object

The RfcTableParameter object manages all information needed for actually using a table in an RFC function call. (The table corresponds to the Table parameter in the RfcCall and RfcReceive RFC functions.)

Available information on RfcTableParameter objects:

[Properties \[Page 339\]](#)

The following example describes the code necessary for packing a Table object into the RfcCall. This code is used internally by the [function control \[Page 155\]](#).



```
Dim oParameter as Object
Dim Table as RFC_TABLE
Dim RfcRc as short
Dim hRfc as long
Set oParameter = oTable.RfcParameter
Table.Name = oParameter.Name
Table.Length = oParameter.NameLength
Table.Type = oParameter.Type
Table.itab = oParameter.TableHandle
RfcRc = RfcCall (hRFC, "MYRFCFUNCTION", 0, 0, Table)
```

RfcTableParameter Object Properties

The RfcTableParameter object has the following properties:

RfcTableParameter Object Properties

Name	Type/Parameter	Description
Name	String	Parameter name of the table. Read-only.
NameLength	Long	Length of the name. Read-only.
TableHandle	Long	ITAB handle of the table. Read-only.
RowLength	Long	Length of one row in the table. Read-only.
Type	Long	RFC Type of the table. Read-only.
TypeName	String	Description of the RFC Type. Read-only.

Rows Collection Object

Rows Collection Object

The Rows collection object is a collection of [Row \[Page 347\]](#) objects. The Rows collection is implemented as a standard collection like many other SAP active control collections. It eases the row access to the table data.

[Properties \[Page 341\]](#)

[Methods \[Page 343\]](#)

Rows Collection Properties

Rows Collection Properties

The Rows collection object has the following properties:

Rows Collection Properties

Name	Type	Description
Count	Long	Returns the number of objects in the collection. Read-only.
Item [Page 342]	VARIANT vaWhichItem	Retrieves a Row [Page 347] from the rows collection. Read-only.

Rows Collection Property: Item

Rows Collection Property: Item

Purpose

Retrieves a [Row \[Page 347\]](#) from the Rows collection.

Description

The parameter vaWhichItem describes the element to be returned. The following variant data types for arWhichItem are legal:

Type	Description
VT_BSTR, VT_LPCSTR, VT_LPWSTR VT_BSTR *, VT_LPCST *, VT_LPWSTR *	The Row object with the key value vaWhatItem is returned. If no key column is defined, vaWhichItem is converted to an integer value and used as index into the collection. See Defining a Key Column for a Table [Page 352] .
Any type convertible to a VT_I4.	The parameter vaWhichItem is converted to an integer value and used as index into the collection.

Rows Collection Methods

Rows Collection Methods

The Rows collection object has the following methods:

Rows Collection Methods

Name	Parameter	Description
Remove [Page 344]	VARIANT vaWhat	Removes a Row [Page 347] object from the collection.
RemoveAll		Removes all Row [Page 347] objects from the collection.
Add [Page 345]	VARIANT vaWhat	Adds a new Row [Page 347] object to the collection.
Insert [Page 346]	VARIANT valIndex	Inserts a new Row [Page 347] object at a given position to the collection.

Rows Collection Method: Remove

Rows Collection Method: Remove

Purpose

Removes a [Row \[Page 347\]](#) from the Rows collection.

Description

The parameter vaWhat describes the element to be removed. Legal variant types for vaWhat are VT_DISPATCH or any data type that may be converted into an integer value. If vaWhat has type VT_DISPATCH, the corresponding object is searched in the collection and removed. Otherwise vaWhat is converted to an index and the corresponding row is removed.



When removing a row from the rows collection, the Row object becomes invalid. Any further attempt to work on the object returns an invalid object exception.

Rows Collection Method: Add

Purpose

Adds a new row to the collection.

Description

The parameter vaWhat describes the element to be added. If vaWhat has type VT_EMPTY, a new [Row \[Page 347\]](#) is added. If vaWhat is of type VT_DISPATCH, a copy of the Row object is added to the end of the collection.

The return value is always a new [Row \[Page 347\]](#) object.

Rows Collection Method: Insert

Rows Collection Method: Insert

Purpose

Inserts an object into the collection at the position provided by valIndex.

Syntax

The Insert method has the syntax:

Insert (valIndex)

Description

The new Row is inserted at the position indicated by valIndex. All following Rows are automatically reindexed.

Row Object

The Row object is an object that represents a single line of the table. This object has the following properties:

[Properties \[Page 348\]](#)

[Methods \[Page 349\]](#)

Row Object Properties

Row Object Properties

The Row object has the following properties:

Row Object Properties

Name	Parameters	Type	Description
Index		Long	Actual index of the Row in the Table [Page 316] . Read-only.
Data		Array of VARIANT	Gets/sets the data (all column values) in the Row object. The data is returned as an array.
Value	VARIANT ColumnIndex	VARIANT	Returns a single value in the row. ColumnIndex is either an index or a name.

Row Object Methods

The Row object has the following methods:

Row Object Methods

Name	Parameter	Description
Clear		Clears the contents of the Row object. Any read accesses after clearing return undefined values.

Columns Collection Object

The Columns collection object is a named collection of [Column \[Page 358\]](#) objects. It is implemented as a standard collection like many other SAP active control collections. It eases the column access to the table data. The columns collection is also responsible for the definition of the actual structure of the table.

[Properties \[Page 351\]](#)

[Methods \[Page 353\]](#)

Columns Collection Properties

The Columns collection object has the following properties:

Columns Collection Properties

Name	Type	Description
Count	Long	Returns the number of objects in the collection. Read-only.
Item [Page 355]	VARIANT vaWhichItem	Retrieves a Column [Page 358] from the columns collection. Read-only.
KeyColumn	VARIANT	Defines a column of the table that contains key information. See Defining a Key Column for a Table [Page 352] .

Defining a Key Column for a Table

You can access a given table [Row \[Page 347\]](#) by calling the Item property with a key value as parameter. To enable key access, you must specify the column in which the key value should be found. Use the KeyColumn property to specify the column name or index of the column. The following restrictions apply to the column used as a key column:

- The column values must be of type String (RFC_CHAR).
- All values in the Column must be unique



```
Dim oTable as Object
Dim oRow as Object
Dim oCustRow as object
dim sCustomerId as string

REM oTable contains the following columns 'Name' and
'CustomerID', 'Street' ...

REM CustomerID is unique in the table
sCustomerId = '0815'

REM Good: Find a special row by using key access
oTable.Columns.KeyColumn = "CustomerID"

Set oCustRow = oTable.Rows.Item (sCustomerId)
oCustRow.Value(1) = .....

REM Bad: Normal search to find the data for the specified
CustomerID

For each oRow in oTable.Rows
    if sCustomerId = oRow.Value("CustomerID")
        Set oCustRow = oRow 'Remember Row object
        break
    end if

Next oRow
oCustRow.Value(1) = .....
```

Columns Collection Methods

The Columns collection object has the methods:

Columns Collection Methods

Name	Parameter	Description
Remove [Page 354]	VARIANT vaWhat	Removes a Column [Page 358] object from the collection.
RemoveAll		Removes all Column objects from the collection.
Add [Page 356]	VARIANT vaWhat	Adds a new Column object to the collection.
Insert [Page 357]	Long Index VARIANT vaWhat	Inserts a new Column object at a given position in the collection.

Columns Collection Method: Remove

Columns Collection Method: Remove

Purpose

Removes a [Column \[Page 358\]](#) object from the Columns collection.

Syntax

The Remove method has the syntax:

```
Remove (vaWhat)
```

Description

The parameter vaWhat describes the element to be removed. Legal variant types for vaWhat are VT_BSTR or any data type that can be converted to an integer value. If vaWhat has type VT_BSTR, the first object with that name in the collection is removed. Otherwise vaWhat is converted to an index and the corresponding column is removed.



When removing a column from the Columns collection, the Column object becomes invalid. Any further attempt to work on the object returns an invalid object exception.

Columns Collection Method: Item

Purpose

Retrieves a [Column \[Page 358\]](#) object from the Columns collection.

Syntax

The Item method has the syntax:

```
Item(vaWhichItem)
```

Description

The parameter vaWhichItem tells describes the element to be returned. The following variant data types for vaWhichItem are legal:

Type	Description
VT_BSTR, VT_LPCSTR, VT_LPWSTR VT_BSTR *, VT_LPCST *, VT_LPWSTR *	The column with the name vaWhichItem is returned. If the name is undefined, vaWhichItem is converted to an integer value and used as an index into the collection.
Any type convertible to a VT_I4.	The parameter vaWhichItem is converted to an integer value and used as index into the collection.

Columns Collection Method: Add

Columns Collection Method: Add

Purpose

Adds a new Column object to the collection.

Syntax

The Add method has the syntax:

```
Add(vaWhat)
```

Description

The Add method always returns a new [Column \[Page 358\]](#) object.

If the vaWhat parameter has type VT_EMPTY, a new column is added. If vaWhat is of type VT_DISPATCH, a copy of the Column object is added to the end of the collection.

Columns Collection Method: Insert

Purpose

Inserts a Column object into the collection.

Syntax

The Insert method has the syntax:

```
Insert(Index, vaWhat)
```

Description

This method inserts an object into the collection at the position provided by Index (a number or string). The vaWhat parameter is optional, but if provided, must be a Column object. In this case, the new column is inserted at the indicated index. All following Columns are then automatically reindexed.

Column Object

Column Object

The Column Object is responsible for access to second dimension of the Table object. It enables data access to columns and also specifies the actual structure definition of the table.

[Properties \[Page 359\]](#)

[Methods \[Page 361\]](#)

Column Object Properties

The Column object has the properties:

Column Object Properties

Name	Type/Parameter	Description
Index	Long	Index of the column in the Table [Page] 316 . Read-only.
Name	String	Name of the column.
Type	CRfcType [Page] 360	Data type of the column.
TypeName	String	Description name of the type. Read-only.
IntLength	Long	Width of the column.
Decimals	Long	Number of decimals.
Offset	Long	Internal start address of the column. Read-only.
Data	Array of VARIANT	Gets/sets the data (all row values) for the Column object. The data is returned as an array.
Value	Long RowIndex	Access a single value in the column.

Column Object Property: Type

Column Object Property: Type

The Type property has the following valid values:

Value	Description
RfcTypeChar = 0	The values are of type String (the default type).
RfcTypeDate = 1	The values are of type Date (special string format).
RfcTypeBCD = 2	The values are of BCD String (compressed packed data).
RfcTypeTime = 3	The values are of type Time (special string format).
RfcTypeHex = 4	The values are binary data.
RfcTypeNum = 6	The values are strings that only contain numbers.
RfcTypeFloat = 7	The values are of type float.
RfcTypeLong = 8	The values are of type long.
RfcTypeShort = 9	The values are of type short.
RfcTypeByte = 10	The values are the integers in the range 0-255.

Column Object Methods

The Column object has the methods:

Column Object Methods

Name	Parameter	Description
Clear		Clears the contents of Column. Any read accesses following a call to Clear will return undefined values.

Ranges Collection Object

The Ranges collection object is a collection of [Range \[Page 366\]](#) objects that provide direct access to parts of the [Table \[Page 316\]](#) object. It eases table manipulation by letting you define views of the table. This is useful, for example, when you only need the second and third column of a table, or only the rows actually displayed in your User Interface control.

[Properties \[Page 363\]](#)

[Methods \[Page 364\]](#)

Ranges Collection Properties

The Ranges collection object has the following properties:

Ranges Collection Properties

Name	Parameters	Type	Description
Count		Long	Returns the number of objects in the collection. Read-only.
Item	Long IWhichItem	DISPATCH	Retrieves a Range [Page 366] object from the Ranges collection. Read-only.

Ranges Collection Methods

The Ranges collection object has the following methods: [Ranges Collection Method: Add \[Page 365\]](#)

Ranges Collection Methods

Name	Parameter	Description
Remove	Long lWhat	Removes a Range [Page 366] object from the collection.
RemoveAll		Removes all Range [Page 366] objects from the collection.
Insert	Long lWhichItem VARIANT vaWhat	Inserts a new Range [Page 366] object at a given position to the collection.
Add [Page 365]	VARIANT vaWhat, Long LowerBound, Long UpperBound, Long LeftBound, Long RightBound	Adds a new Range [Page 366] object to the collection. All Parameters are optional.

Ranges Collection Method: Add

Purpose

Adds a new Range object to the collection.

Syntax

The Add method has the syntax:

```
Add(What, LowerBound, UpperBound, LeftBound, RightBound)
```

Description

This method adds a new Range object to the collection and returns the newly added object.

The What parameter is optional, and can be either a [Range \[Page 366\]](#) object or omitted. In the former case, the Range object is appended to the Ranges collection. In the latter case, a new Range object is added.

The other parameters are also optional and provide the bounds of the range. If they are missing, the new Range object uses the table boundaries. It can be resized afterwards by accessing the properties of the Range object.



```
Dim oTable as Object
Dim oRange as Object
REM Different usages of the Add method
REM 1. Standard
Set oRange = oTable.Ranges.Add
REM oRange contains the whole table
REM 2. Create a Range which contains the Rows 2 to 10
Set oRange = oTable.Ranges.Add (, 2, 10)
REM 3. Create a Range that contains the Columns 2 and 3
Set oRange = oTable.Ranges.Add (,,,2,3)
REM 4. Build a Range collection, in that each Range has 5 Rows
Dim oRanges as Object
Set oRanges = oTable.BuildTiledRange(5)
Set oRange = oRanges.Item(3)
```

Range Object

The Range object is a two-dimensional view of all or part of a Table object. It is like a Range object in Microsoft EXCEL.

[Properties \[Page 367\]](#)

[Methods \[Page 369\]](#)

Range Object Properties

The Range object has the properties:

Range Object Properties

Name	Type/Parameter	Description
LowerBound [Page 368]	Long	Lower index. Determines the starting row in the Table object.
UpperBound [Page 368]	Long	Upper index. Determines the ending row in the Table object.
LeftBound [Page 368]	Long	Left index. Determines the starting column in the Table object.
RightBound [Page 368]	Long	Right index. Determines the ending column in the Table object.
Data	Array of VARIANT	Gets/sets the data (all values) of the Range object. The data is returned as an array.
Value	Long RowIndex Long ColumnIndex	Returns a single value in the Range object.

Boundaries of a Range Object

The boundaries of a Range object can be set in two ways:

- by using the parameters of the [Add method \[Page 365\]](#) on a Ranges collection object
- by setting the bounds explicitly with the LowerBound, UpperBound, LeftBound, or RightBound properties

All boundaries are checked against the underlying Table object. An OLE exception is triggered if a bound exceeds the corresponding table dimension.



```
Dim oRange as Object
Dim oTable as Object
Rem 1. Set the Boundaries through the properties
Set oRange = oTable.Ranges.Add
oRange.LowerBound = 2
oRange.UpperBound = 4
oRange.LeftBound = 2
oRange.RightBound = 4
Rem 2. Set the Boundaries via the Ranges.Add method
Set oRange = oTable.Ranges.Add ( 2,4,2,4)
```

Range Object Methods

The Range object has the methods:

Range Object Methods

Name	Parameter	Description
Clear	void	Clears the contents of the Range. Any following read accesses return undefined values.

Views Collection Object

Views Collection Object

The Views collection object contains all views related to the Table object. The objects in a View collection are not “View objects”, but rather additional controls that are used together with the Table object.

[Properties \[Page 371\]](#)

[Methods \[Page 372\]](#)

Using Views with Table Objects

A view provides the ability to connect additional “table-related” controls to the Table object. These additional controls provide ways of accessing and displaying the data stored in the Table object.

Any object implementing the ISAPInternalLock and ISAPInternalViewNotification interface can be added to the Views collection. Currently, A table-related control is either a Table View control or a Table Tree control.

For more details, see:

- [The Table View Control \[Page 389\]](#)
- [The Table Tree Control \[Page 216\]](#)
- [How to Connect Views to a Table \[Page 373\]](#).

Views Collection Properties

The Views collection object has the following properties:

Views Collection Properties

Name	Type	Description
Count	Long	Returns the number of objects in the collection. Read-only.
Item	Long IWhichItem	Returns an view from the Views collection. Read-only.

Views Collection Methods

The Views collection object has the following methods:

Views Collection Methods

Name	Parameter	Description
Remove	Long lWhat	Removes a View object from the collection.
RemoveAll		Removes all View objects from the collection.
Add	Object oWhat	Adds a View to the collection.
Insert	Long lWhichItem Object oWhat	Inserts a new View at a given position to the collection.



```
Dim oTableView as Object REM SAP.TableView Object
Dim oTable as Object
oTable.Views.Add oTableView
oTable.Value (1,1) = "Testdata"
REM The String "Testdata" is automatically displayed in
TableView
REM If the OTableView is an object which lies on a Visual
Basic Form
REM the following code must be used
oTable.Views.Add oTableView.Object Rem access the dispatch
interface of the TableView
```

How to Connect Views to a Table

The ability to connect a view to the Table object eases the way a programmer can display data stored in the Table object. The view (a separate control) can be added directly to the Views collection of the Table object. Afterwards all data changes in the Table object are automatically displayed in all views stored in the Views collection.

Any object which implements the ISAPInternalLock and ISAPInternalViewNotification interface can be added to the Views collection. At the moment, the SAP Table View control and the SAP Tree View control can be used as views.

For additional information, see:

- [Connecting Tree Views and Table Objects \[Page 272\]](#)
- [Connecting Table Views and Table Objects \[Page 413\]](#)



```
Dim oTableView As Object REM SAP.TableView Object
Dim oTable As Object
oTable.Views.Add oTableView
oTable.Value (1,1) = "Testdata"
REM The String "Testdata" is automatically displayed in
REM TableView
REM If the oTableView is an object which lies on a Visual
Basic Form
REM the following code must be used
oTable.Views.Add oTableView.Object
```

Matrix Object

The Matrix object is the result of the [SelectTo* \[Page 335\]](#) methods provided in the [Table \[Page 316\]](#) object. The Matrix object has the following properties:

Matrix Object Properties

Name	Type	Description
RowCount	Long	Returns the number of rows in the matrix. Read-only.
ColumnCount	Long	Returns the number of columns in the matrix. Read-only.
IsRowVector	Boolean	Returns TRUE if the matrix contains only one row. Read-only.
IsColumnVector	Boolean	Returns TRUE if the matrix contains only one column. Read-only.
Data	Array of VARIANT	Gets/sets the data values in the Matrix object. The data is returned as an array.
Row	Long Index	Gets/sets the data values in one row of the matrix. The data is returned as an array.
Column	Long Index	Gets/sets the whole data of one column of the matrix. The data is returned as an array.
Value	Long RowIndex, Long ColumnIndex	Gets/sets the value of one matrix entry.

Code Examples

This section contains the following examples:

[First Steps \[Page 376\]](#)

[Accessing Table Data \[Page 377\]](#)

[Automatic Display \[Page 380\]](#)

[Using Dynamic Structures for a Table \[Page 381\]](#)

First Steps

```
Rem oConnection was obtained by the Logon control.  
Dim oTableFactoryCtrl as Object  
Dim oTable as Object  
Rem Create an instance of the Table Factory.  
Set oTableFactoryCtrl = CreateObject ("SAP.TableFactory.1")  
Rem Get a new Table object.  
Set oTable = oTableFactoryCtrl.NewTable ()  
if oConnection.IsConnected () = tloConnected then  
    oTable.CreateFromR3Repository (oConnection, "RYPBMHI39,  
"NODETAB")  
else  
    Set oTable = Nothing  
end if  
Rem Now you have a fully initialized Table object.  
Rem To display the column structure:  
Dim oColumn as Object  
For each oColumn in oTable.Columns  
    msgbox oColumn.Name  
    msgbox oColumn.TypeName  
next oColumn
```

Accessing Table Data

The following code demonstrates techniques for accessing the data stored in a Table object.

```
Rem This example assumes that we have a table with 10 Rows and 5
Columns.

Dim vVal as VARIANT
Dim i as Long
Dim j as Long

Rem You can access each entry direct in the Table object
vVal = oTable.Value (1,1) ' or vVal = oTable(1,1)
Rem Alternatively, the column index can be a string
vVal = oTable.Value (1,"KUNNR") ' or vVal = oTable(1,"KUNNR")
Rem Setting a value works the same way
oTable.Value (1,1) = vVal

Rem You can access each entry of a table row using the Row object
Dim oRow as Object
Set oRow = oTable.Rows.Item (1) ' or Set oRow = oTable.Rows(1)
vVal = oRow.Value (1) ' or vVal = oRow(1)
Rem Alternatively, the column index can be a string
vVal = oRow.Value ("KUNNR") ' or vVal = oRow("KUNNR")
Rem Setting a value works the same way
oRow.Value (1) = vVal

Rem You can access each entry in a table column using the Column
object.

Dim oColumn as Object
Set oColumn = oTable.Columns.Item(1) ' or Set oColumn=oTable.Columns(1)
vVal = oColumn.Value (1) ' or vVal = oColumn(1)
Rem Setting a value works the same
oColumn.Value (1) = vVal

Rem You can change the viewport to the table by using the Range object.
Rem   The following line of code builds a Ranges collection in which
Rem   each Range consists of 2 Rows of the Table
```

Accessing Table Data

```
oTable.BuildTiledRanges (2)

Rem You can access each entry in a table range using the Range object
Dim oRange as Object

Set oRange = oTable.Ranges.Item(1) ' or Set oRange=oTable.Ranges(1)
vVal = oRange.Value (1,1) ' or vVal = oRange(1,1)
Rem Setting a value works the same way
oRange.Value (1,1) = vVal

Rem In order to speed up access to all data, the Data property
Rem is provided for each of the object types to be accessed.
Rem The following lines of code provide the standard technique for
Rem accessing all data in the table.

Dim vVal as VARIANT(10,5)
Dim i as Long
Dim j as Long
For i = 1 To Table.RowCount
    For j = 1 To Table.ColumnCount
        vVal (i,j) = Table.Value(i,j)
    Next i
Next oRow

Rem By using the data property, the above code shrinks to one line:
Dim vVal as VARIANT
vVal = Table.Data

Rem Set values in the same way. Rows are added automatically,
Rem but columns are not.
Table.data = vVal

Rem The same works for a Row, a Column and a Range.

Dim oRow as Object
Dim oColumn as Object
Dim oRange as Object
Set oRow = oTable.Rows(1)
Set oColumn = oTable.Columns(2)
Set oRange = oTable.Ranges(3)
vVal = oRow.Data
```

Accessing Table Data

```
vVal = oColumn.Data  
vVal = oRange.Data  
oRow.Data = vVal  
oColumn.Data = vVal  
oRange.Data = vVal
```

Automatic Display

Automatic Display

Once a Table object and a Table View object are connected, all data changes made in one object are directly propagated to the other. This propagation can flow in both directions and happens automatically (no additional coding):

```
Rem oTableView1 is an SAP Table View object placed on the
Rem   Visual Basic Form
Rem oTable is a Table object
oTable.Views.Add oTableView1.Object ' Connect the grid with the table
oTable(1,1) = "Hallo" ' Hallo is automatically displayed by the grid
```

Using Dynamic Structures for a Table

Some SAP BAPI function modules (Business API) use generic table parameters and return the actual structure of the data table in a second table. The Table object is capable of using another interpretation of its contents by simply changing its column structure. The actual column structure is restricted to the row length of the dummy table. The following code demonstrates how you can achieve this:

```
Dim oFuncOCX as Object
Dim oFunc as Object
Dim oFieldTab as Object
Dim oDataTab as Object
Dim colObj as Object
Dim k as Long

Rem Create a Function control
Set oFuncOCX = CreateObject ("SAP.Functions")
Rem Add a BAPI to the Functions collection
Set oFunc = oFuncOCX.Add ("MC_RFC_BAPI_OIWID")
if not oFunc.Call then stop end if ' if error stop

Rem Access the table parameter
Set oFieldTab = oFunc.Tables("FIELDS")
Set oDataTab = oFunc.Tables("DATA")

Rem Clear the old column structure
For k = oDataTab.columnCount To 1 Step -1
    oDataTab.Columns.Remove k
Next

Rem Imprint the new structure
For k = 1 To oFieldTab.RowCount
    Set colObj = oDataTab.Columns.Add
    colObj.intlength = Val(oFieldTab.cell(k, 3))
    If InStr("FNPI", oFieldTab.cell(k, "type")) > 0 Then
        colObj.type = 7
    else
        colObj.type = 0
    End If
Next
```

Using Dynamic Structures for a Table

End If
Next

Glossary

[Table \[Page 384\]](#)

[Row \[Page 385\]](#)

[Column \[Page 386\]](#)

[Matrix \[Page 387\]](#)

[Range \[Page 388\]](#)

Table**Table**

The Table object represents (encapsulates) an internal table as it is provided by the RFC library.

Row

The Row object represents a single line of a Table object, e.g. the line operations of the RFC-ITAB handling.

Column

Column

The Column object represents a single column of the underlying Table object. There is no corresponding “direct handling” in the RFC library.

Matrix

The Matrix object represents a two-dimensional data array. It serves as a result of the SelectTo* methods provided by the Table object.

Range

Range

The Range object serves as a view of the underlying table, similar to Range objects in an Excel spreadsheet.

The Table View Control

This section contains the following topics:

Introduction

[Introduction \[Page 390\]](#)

[Table View Control Object Hierarchy \[Page 391\]](#)

[Basic Concept \[Page 392\]](#)

Control and Object Reference

[Table View Object \[Page 396\]](#)

[Columns Collection Object \[Page 417\]](#)

[Column Object \[Page 423\]](#)

[Rows Collection Object \[Page 429\]](#)

[Row Object \[Page 435\]](#)

[Cell Object \[Page 438\]](#)

[Design Environment Property Pages \[Page 441\]](#)

Programming Guide

[Connecting Table Views and Table Objects \[Page 413\]](#)

[Drag and Drop with Table Views \[Page 416\]](#)

[Glossary \[Page 449\]](#)

Introduction

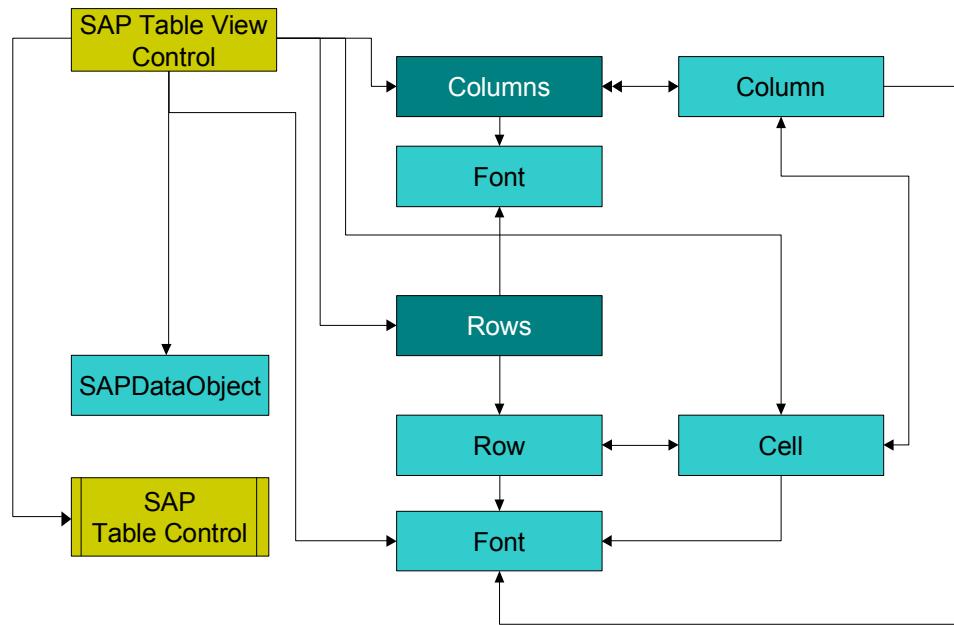
Introduction

The SAP Table View control is an R/3-aware active OLE control that simplifies the handling of table data. Using OLE Automation and OLE control technology, the SAP Table View control can either work as a standalone control or in combination with other SAP OLE controls like the [SAP Table Factory \[Page 300\]](#).

Using the SAP Table View control is not necessarily combined with the use of any R/3 system. It can run in every OLE control container as an advanced active OLE control. Nevertheless, the SAP Table View control exploits its full functionality in conjunction with the [SAP Table Factory \[Page 300\]](#). Combining both active OLE controls allows the user an easy and very quick way to navigate through hierarchical structures.

Table View Control Object Hierarchy

Table View Control Object Hierarchy



Basic Concept

Basic Concept

The SAP Table View control is a Windows ActiveX OLE control that may be used in various scenarios. It is up to the user of the control to define the appearance and data stored in the control. This is either done interactively in design mode (see [Design Environment Property Pages \[Page 441\]](#)), or at runtime through OLE Automation calls.

For this purpose, the object hierarchy of the Table View control is separated into columns, rows, and cells. This separation makes access to data and properties of the control easier. Different fonts may be assigned to each object. Different formatting is available for each column. Each column can also be combined with one of several data types. The user interface is changed according to these data types. For example, Boolean values are displayed as checkboxes and selection data types as drop-down combo boxes. Drag-and-drop clipboard operations are supported in a standard text-format-based implementation. Nevertheless, these operations may also be adapted easily to the application's needs.

Last but not least, the SAP Table View control can become an R/3-aware control through connection to the [Views \[Page 370\]](#) collection of a [SAP Table object \[Page 305\]](#). This results in an highly automated procedure for the display and navigation of table data obtained from an R/3 System or any other data source.

Using the Table View Control

Using the Table View Control

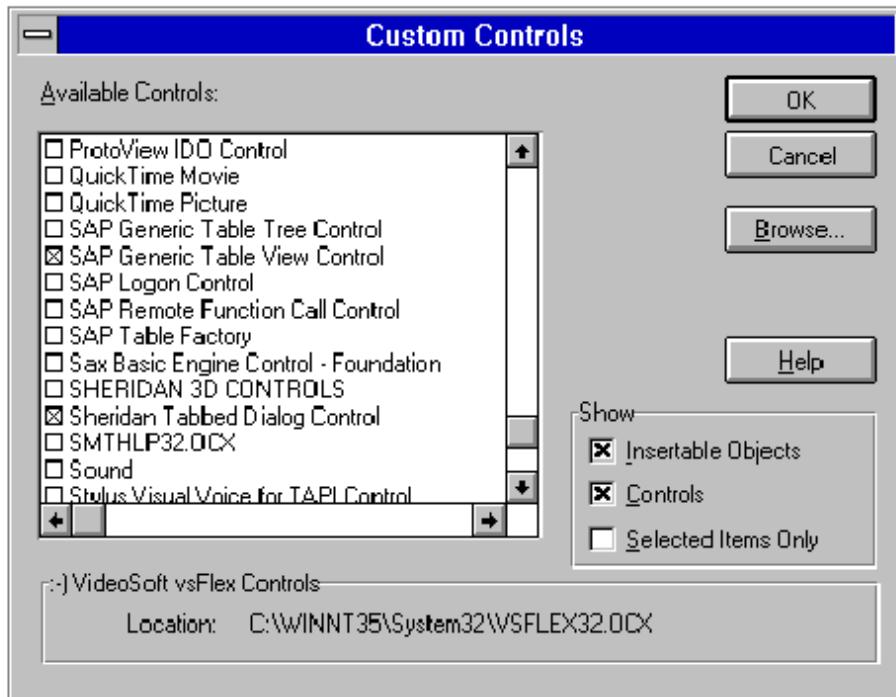
The Table View control provides views of tables that have been retrieved from R/3. It also allows you to edit the data shown in the view, and automatically gets the table contents updated. You get access to a Table View control by using a variable. This is easy in Visual Basic (Version >= 4.0).

To bind a table view to a table, get the table (such as Customers) and the view (such as SAPTableView), and then notify the table that it now has a view.



```
' Establish view-table connection.  
  
Customers.Views.Add SAPTableView.Object  
  
Customers.Refresh
```

Below is a complete procedure for displaying a table from Visual Basic. The first step is to drop a table view onto a form and set its name.



When you have retrieved the table via an RFC, connect it to the view. In Visual Basic, create a form, then select *SAP Generic Table View Control* in the *Custom Control* dialog.

Using the Table View Control



An icon is placed in the toolbox. Click on this icon, and then create the control in the area on the form where you want the table data displayed. Change the name of the new object (in the *Properties* window) to SAPTableView (the name is optional, as long as you use the same name in the script).

Create a text-entry field, and name it (for example) NameInput. Then, add a button and add the following code to it (the click callback function):

```

Private Sub Command1_Click()
    ' Create function component.

    Dim fns As Object
    Set fns = CreateObject("SAP.Functions.1")
    fns.logfilename = "c:\tmp\table+viewlog.txt"
    fns.loglevel = 8

    Dim conn As Object
    Set conn = fns.Connection
    conn.Client = "000"
    conn.Language = "E"
    conn.tracelevel = 6

    If conn.logon(0, 0) <> True Then
        MsgBox "Could not logon!"
    End If

    Dim Customers, Customer As Object
    Dim Result As Boolean

    ' Get the name to display from NameInput and call function ...
    Result = fns.RFC_CUSTOMER_GET(Exception, NAME1:=NameInput, KUNNR:="*",
    CUSTOMER_T:=Customers)

    If Result <> True Then
        MsgBox ("Call error: " + Exception)
        Exit Sub
    Else
        ' try to display table view.
        Customers.views.Add TheTableView.object
        Customers.Refresh
        MsgBox "Got " + Str$(Customers.RowCount) + " rows."
    End If
    Set fns = Nothing

```

Using the Table View Control

```
Set conn = Nothing  
End Sub
```

The resulting form should resemble the following:

Form1

	A	C	E	
116	0000003710	RJB Snack Food Divisior	100 Mission Ave	6
117	0000003720	RJB Health Foods Divisid	4321 Buffalo Rd	0
118	0000003800	Candid International Copi	345 Candid Dr	3
119	0000003810	Candid Mini Copiers Divis	9867 Midland Ave	1
120	0000003820	Candid Corporate Copier	765 Larson Dr	8
121	0000003891	Blip Pens Corporation	134 Ballpoint St	0
122	0000003892	Randy Maps Incorporated	345 Printit Dr	1
123	0000003893	PH Desktop Computers C	87 Ocean Dr.	7
124	0000003894	PH Monitor Corporation	4321 Callahan dr	8
125	0000003895	PH Keyboard Corporation	987 Type Dr	4
126	0000003910	Infix Co.	365 E. Evelyn Av.	9
127	0000003999	Johnson engineering	3460 West Bayshore	0
128	0000004000	Roberta Energy Ltd		7

Select names (SQL style):

Table View Object

The SAP Table View object is the highest object in the control's hierarchy, and therefore is also called the [root object \[Page 451\]](#). You obtain a Table View object by:

- calling `CreateObject ("SAP.TableViewControl")` in a Visual Basic application
- inserting a Table View control directly into a Visual Basic form from a toolbar

[Properties \[Page 397\]](#)

[Methods \[Page 405\]](#)

[Events \[Page 406\]](#)

Table View Properties

Table View Properties

The Table View object has the following properties:

Table View Properties

Name	Type	Description	
Events	Long	Enables or disables events fired by the control.	
BorderStyle	short	Sets the border style for the control. Possible values are:	
		Enable3D = 0	(Draws a 3D frame)
		Simpleframe = 1	(Draws a simple frame)
ForeColor	OleColor	Defines the table foreground color. This color is used for displaying text.	
BackColor	OleColor	Defines the table background color.	
Enabled	Boolean	Enables or disables the entire control.	
HWnd	Handle	Window handle of the control.	
Parent	Handle	Window handle of the parent window.	
Font	Object	Font object used as default font for displaying text.	
ActiveColumn	Long	Returns or sets the column index for the active cell [Page 455] .	
ActiveRow	Long	Returns or sets the row index for the active cell [Page 455] .	
Columns [Page 417]	Object	Collection of all Column objects in the Table View. Read-only.	
Rows [Page 429]	Object	Collection of all Row objects in the Table View. Read-only.	
ColumnCount	Long	Number of columns.	
RowCount	Long	Number of rows.	
FixedColumns	Long	Number of fixed columns. Fixed columns are columns which do not scroll.	
FixedRows	Long	Number of fixed rows. Fixed rows are rows which do not scroll.	

Table View Properties

Selection [Page 401]	Object	Returns or sets the current selection.	
HideSelection	Boolean	Indicates whether the selected range should be highlighted when the control does not own the input focus.	
ShowGridLines	Boolean	Indicates whether grid lines should be drawn.	
ShowHScrollbar	Boolean	Indicates whether a horizontal scrollbar should be attached to the control.	
ShowVScrollbar	Boolean	Indicates whether a vertical scrollbar should be attached to the control.	
ShowRowHeaders	Boolean	Indicates whether a row header should be displayed. (See also Header property of row [Page 435] object).	
ShowColHeaders	Boolean	Indicates whether a column header should be displayed (see also Header property of column [Page 423] object).	
EnableProtection [Page 402]	Boolean	Enables or disables the protection of cells, rows or columns.	
SelectMode	CSelModeType	Sets or returns the current selection mode. Possible values are:	
		tavSelModeDisable = 1	No selection is possible.
		tavSelModeCell = 2	Selection of single cells, rows and columns is possible.
		tavSelModeRow = 3	Only entire rows may be selected.
		tavSelModeCol = 4	Only entire columns may be selected.
DragDrop	CDragDropType	Sets or returns the current drag and drop mode. (See also DropEnter [Page 411] , Drop [Page 412] , DragSourceFill [Page 410] and DragComplete events.) Possible values are:	
		tavDragDropModeDisable = 1	Drag and Drop is disabled.

Table View Properties

		tavDragOnly = 2	The control may only be used as drag source.
		tavDropOnly = 3	The control may only be used as drop target.
		tavDragDropAll = 4	Full drag and drop is enabled.
Formula	Variant	Not implemented yet. Returns the same as Value.	
Value [Page 403]	Variant	Returns the data of a single cell.	
Data	Array of Variant	Sets or returns the entire data of the control in a two dimensional array. A single element of the array has type variant.	
Cell [Page 404]	Object	Returns a Cell object [Page 438] . Read-only.	
AutoConfig	CAutoConfig	Defines how the connection to a SAP Table object [Page 220] is established. Possible values are: (See also Connecting Table Views and Table Objects [Page 413] .)	
		tavAutoConfigDisabled = 0	The connection is defined by the TableIndex property of the column [Page 423] and row [Page 435] objects.
		tavAutoConfigRows = 1	The connection is defined by the TableIndex property of the column [Page 423] objects. Rows are automatically maintained by the connection.
		tavAutoConfigCols = 2	The connection is defined by the TableIndex property of the row [Page 435] objects. Columns are automatically maintained by the connection.

Table View Properties

		tavAutoConfigAll = 3	The connection is automatically maintained by the connection.
Table	Object	Returns the SAP Table object [Page 220] connected to the view. (See also Connecting Table Views and Table Objects [Page 413])	

Table View Property: Selection

Table View Property: Selection

Purpose

The Selection property returns or sets the current selection.

Description

The current selection is an object representing either a single cell or a [range \[Page 456\]](#) of cells. The SAP Table View control allows only one selected range. In either case, the selected object has properties and methods of its own:

Property	Type	Description
<default property>	String	Sets or returns the selection.
UpperBound	Long	Set or returns the uppermost selected row
LowerBound	Long	Set or returns the lowermost selected row
LeftBound	Long	Set or returns the leftmost selected column
RightBound	Long	Set or returns the rightmost selected column

Use the <default property> for the selected object to determine what part of the table was selected. The default property returns a string value that uses a special notation to specify the selection:

- Single cells: The column is identified by a character and the row by a number. A1 would be the first row in the first column. B5 would be the fifth row in the second column.
- Ranges: Codes for the left topmost and the right bottom-most cells are separated by a colon. A range consisting of all cells starting from the second row in the second column to the fifth row in the fourth column would be set or returned by a range string of 'B2:D5'.

The following method is also available with the selected object:

Method	Parameter	Description
Set	Long Lower Long Left Long Upper Long Right	Sets the current selection.

When you want to set all selection boundaries at once, the Set method is more efficient than making four calls to the boundary properties.

Table View Property: EnableProtection

Table View Property: EnableProtection

Purpose

Globally enables and disables modifications to the Table View.

Description

You can protect all or part of a Table View against modifications. If the user tries to modify a protected cell, an Error event is fired.

To turn on protection, set the EnableProtection property to TRUE.

To specify which [cell \[Page 438\]](#), [row \[Page 435\]](#), or [column \[Page 423\]](#) you want to protect, you must also set the Protection properties in each individual item. However, if you set the individual Protection properties without setting the global EnableProtection, all cells remain open for modifications.

Table View Property: Value

Table View Property: Value

Purpose

Sets or returns the data in a specified cell.

Syntax

The Value property has the syntax:

`Value (Long Row, Long Column)`

Description

The Value property sets or returns the data in the cell identified by Row and Column.

Table View Property: Cell

Purpose

Returns a [Cell Object \[Page 438\]](#) for given row and column values.

Syntax

The Cell property has the syntax:

```
Cell(Long Row, Long Column)
```

Description

This property returns the Cell object defined by *Row:Column*. This property is read-only.

Table View Methods

Table View Methods

The Table View object has the following methods:

Table View Methods

Name	Parameter(s)	Return Type	Description
AboutBox	void	void	Displays the AboutBox dialog.
CopyToClipboard	void	Boolean	Copies the current selection to the clipboard using CF_TEXT format.
PasteFromClipboard	void	Boolean	Pastes the content of the clipboard starting at the active cell. Returns TRUE if the paste operation was successful.
ClearSelection	void	void	Deletes data values for the current selection.
Clear	void	void	Deletes data values for the entire table.
ColumnAutoWidth	Long Column1 Long Column2	void	Adjusts the width of the columns specified by <i>Column1..Column2</i> automatically, as required by column contents.

Table View Events

Table View Events

The SAP Table View control fires several events in order to inform the container on state changes, user interaction and drag and drop operations. The events may be enabled or disabled through the Table View object's [Events property \[Page 397\]](#).

The events are:

SAP Table View Events

Name	Parameters	Description	
DblClick	void	A double-click occurred within the control's client area.	
KeyDown	short* KeyCode short ShiftState	A key was pressed. The virtual key code is passed in <i>KeyCode</i> , the current state of the shift key in <i>ShiftState</i> . <i>KeyCode</i> may be modified within the event-handling routine.	
KeyUp	short* KeyCode short ShiftState		A key was released. The virtual key code is passed in <i>KeyCode</i> , the current state of the shift key in <i>ShiftState</i> . <i>KeyCode</i> may be modified within the event-handling routine.
TableCreate	void	A connected SAP Table object [Page 220] has created its table.	
TableClear		A connected SAP Table object was cleared.	
DataChange	Long Row1 Long Row2 Long Column1 Long Column2 Variant vaData		The data in the range [Page 456] <i>Row1:Column1</i> to <i>Row2:Column2</i> has changed. The new data is passed in <i>vaData</i> . If only one cell has changed, <i>vaData</i> contains data of a simple variant type, if an entire range has changed, <i>vaData</i> contains an array of variant.

Table View Events

RowInsert	Object Row		A new row [Page 435] object was inserted. The new Row object is passed as parameter <i>Row</i> .
ColumnInsert	Object Column		A new column [Page 423] object is being inserted. The new Column object is passed in as parameter <i>Column</i> .
RowRemove	Object Row		The row [Page 435] object <i>Row</i> is being removed. It is not possible to cancel the remove operation within the remove event handler.
ColumnRemove	Object Column		The column [Page 423] object <i>Column</i> is removed. It is not possible to cancel the remove operation within the remove event handler.
BeforeInput	Object Cell		The user starts an input action in the cell <i>Cell</i> . The corresponding Row and Column objects and indices are accessible through the Row and Column property of the Cell [Page 438] object.
AfterInput	Object Cell Variant newValue		The user ends an input action in the cell <i>Cell</i> . The according Row and Column objects and indices are accessible through the Row and Column property of the Cell [Page 438] object. The new value for the cell is passed to the event handler in <i>NewValue</i> .
DragSourceFill [Page 410]	Object DataObject Short* Handled		A drag-and-drop [Page 416] operation is starting. <i>DataObject</i> is a SAP Data Object [Page 149] and may be filled with any format.
DropComplete	Long Effect		A drag-and-drop [Page 416] operation has been completed. <i>Effect</i> describes the type of drop done. It is up to the application to consider the <i>Effect</i> .

Table View Events

DropEnter [Page 411]	Object DataObject Long KeyState Long* Effect Short* Handled		During a drag-and-drop [Page 416] operation, the mouse pointer was moved into the client area of the control. (See also SAP Data Object [Page 149])
Drop [Page 412]	Object DataObject Long Row Long Column Long* Effect Short* Handled		A drop has occurred on the control's client area. (See also SAP Data Object [Page 149])
SelChange	Long RowLow Long ColumnLo w Long RowHigh Long ColumnHig h		The selection [Page 401] property has changed. The parameters <i>RowLow</i> , <i>ColumnLow</i> , <i>RowHigh</i> , <i>ColumnHigh</i> are the new selection. If only one cell is selected, the high values equal the low values.
Error	Short Number String* Description Long Scode String Source String HelpFile Long HelpConte xt Short* CancelDisp		This event is fired after an error has occurred. <i>Description</i> is a literal error description. <i>Scode</i> is the corresponding OLE error code or control specific error code. <i>Source</i> is the name of the control causing the error, <i>HelpFile</i> and <i>HelpContext</i> define where to find specific help on the error that occurred. <i>CancelDisp</i> must be set to TRUE if no error message box should appear. Setting <i>CancelDisp</i> to FALSE is very convenient during development. For release applications, this parameter should be set to TRUE in most cases.

Table View Events

If special errors should be treated in the error event handler, the *Scode* and the *Number* parameter are to be used as the describing element. The *Description* parameter is language-dependent and may vary with different versions of the control.

Table View Event: DragSourceFill

Purpose

Event-notification when a user starts a drag-and-drop operation.

Syntax

The DragSourceFill event has the syntax:

```
DragSourceFill(Object DataObject, Short *Handled)
```

Description

This event is fired at the source control when the user starts a [drag-and-drop \[Page 416\]](#) operation. The data object to be used for drag and drop is passed in to the event-handler as *DataObject* and represents a [SAP Data Object \[Page 149\]](#).

If you code an event-handler for this event, your code can fill *DataObject* with the appropriate data, and set the *Handled* flag to TRUE. In this case, any default event processing by the control is disabled.

If your event-handler does not set *Handled* to TRUE, the control performs default event-handling. This includes adding the selected data to the data object using the format CF_TEXT. Any data previously stored in the data object using this format is overwritten.

Table View Event: DropEnter

Table View Event: DropEnter

Purpose

Event-notification when a user drags a drag-and-drop object into the target control.

Syntax

The DropEnter event has the syntax:

```
DropEnter(Object DataObject, Long KeyState, Long * Effect, Short  
*Handled)
```

Description

The DropEnter event is fired when the mouse pointer is moved into the client area of a control during a [drag-and-drop \[Page 416\]](#) operation. The *DataObject* passed to the DropEnter event handler is a [SAP Data Object \[Page 149\]](#) that was originally filled in the [DragSourceFill \[Page 410\]](#) event. (The source control may be either a Table View or a Table Tree control.)

Your event-handler for DropEnter can call the IsFormatAvailable method (on the *DataObject*) to determine whether *DataObject* contains acceptable information or not. If the *DataObject* is acceptable, set the *Effect* parameter to one of the following values in order to change the cursor accordingly:

Value	Description	Cursor
DROPEFFECT_NONE = 0	Drop is not possible.	
DROPEFFECT_COPY = 1	Drop of copy is possible.	
DROPEFFECT_MOVE = 2	Move of DataObject is possible.	

Set the *Handled* parameter to TRUE if you want to disable the control's default drag and drop handling.

See your Visual Basic documentation for information on the contents of the *KeyState* parameter.

Table View Event: Drop

Purpose

Event-notification when a user drops a drag-and-drop object into the target control.

Syntax

The Drop event has the syntax:

```
Drop(Object DataObject, Long Row, Long Column, Long * Effect, Short  
*Handled)
```

Description

The Drop event is fired if the control's client area is the drop target of a [drag-and-drop \[Page 416\]](#) operation. The parameters *Row* and *Column* indicate the location in the table where the *DataObject* was dropped.

Invoke the *IsFormatAvailable* and *GetData* methods (on the *DataObject*) to retrieve any data stored in the *DataObject*. This data was originally filled in the [DragSourceFill \[Page 410\]](#) event and can be in any format. (The source control may be either a Table View or a Table Tree control.)

The *Effect* parameter indicates whether the *DataObject* should be copied, moved or whether a link to the data source object should be established (see also [DropEnter \[Page 411\]](#)event). The *Effect* parameter will subsequently be passed on to the drop source control when the DropComplete event is fired.

Set the *Handled* parameter to TRUE if you want the control's default drag and drop handling to be canceled.

Connecting Table Views and Table Objects

Although the SAP Table View control can be used alone, you must connect it to a Table object's Views collection to exploit its full functionality. Connecting the view control to the Table object means that data is transported automatically between both controls in both directions. Any change of data either by user interaction or through the control's automation is transferred to the Table object. The same scenarios work when data is changed in the Table object. Any formatting or data conversion done within the Table object is immediately reflected in the view control.

Connecting and Disconnecting the Controls

The connection is easily established. Every Table object exports a [Views \[Page 370\]](#) collection. This collection maintains all objects using the Table object as data source. By adding an instance of a Table View control to this View collection (in the Table object), the connection is established. The connection is released by removing the Table View object from the Table object's Views collection.



```

Dim oTableFactory As Object
Dim oTable As Object
  ' Create TableFactory and Object
  Set oTableFactory = CreateObject("SAP.TableFactory.1")
  Set oTable = oTableFactory.NewTable
  ' Establish view - connection
  oTable.Views.Add SAPTableView1.Object
  '... Do anything
  ' Remove view - connection
  oTable.Views.Remove(1)

```

How the Connection is Managed

The connection between the Table View and Table Factor controls is configured through:

- the AutoConfig property in the Table View object
- the TableIndex property in the [Row \[Page 435\]](#) and [Column \[Page 423\]](#) objects (in the Table View control)

The TableIndex properties in the Table View object are the key to the connection mechanism. These indexes tell which Table View row or column corresponds to which Table Factory row and column. Thus each Table View cell may address data in the Table object using its TableIndex properties. The AutoConfig property in the Table View object distinguishes between four different modes:

Value	Description
-------	-------------

Connecting Table Views and Table Objects

tvaAutoConfigDisabled = 0	The TableIndex of the Row [Page 435] and Column [Page 423] objects are provided and maintained by the application using the control. No automated mechanism works for these properties. If all TableIndex properties are set to 0, the behavior is the same as if there were no connection.
tvaAutoConfigRows = 1	The TableIndex of all Row objects are provided and maintained automatically through the connection. If rows are inserted or removed in the Table object, all TableIndex properties (in the relevant Table View Rows) are updated automatically. The relevant new or outdated Rows in the Table View are also automatically inserted or deleted.
tvaAutoConfigCols = 2	The TableIndex of all Column objects are provided and maintained automatically through the connection. If columns are inserted or removed in the Table object, all TableIndex properties (in the relevant Table View Columns) are updated. The relevant new or outdated Columns in the Table View are also automatically inserted or deleted.
tvaAutoConfigAll = 3	All TableIndex properties are provided and maintained automatically through the connection. All modifications in the Table object are reflected automatically for all Table View rows and columns.



The entire connection is driven by the [Row \[Page 435\]](#) object and [Column \[Page 423\]](#) object's TableIndex properties. Nevertheless it is possible to insert and remove rows and columns through the SAP Table View control's [Rows \[Page 429\]](#) and [Columns \[Page 417\]](#) collections. These rows and columns are not configured automatically in any case. This allows the application to insert or remove rows and columns from the Table View without any effect on the Table object. If the Table View object and the Table object need to be kept synchronous, the application should always work on the Table object.

How Events are Communicated

The following rules apply to the connection:

Event	Reaction
Data changed in the Table object	Corresponding cell data in the Table View object is changed immediately.
Data changed in the Table View object	Corresponding cell data in the Table object is changed immediately.
Row deleted in the Table object	Corresponding row is deleted in the Table View object.
Row deleted in the Table View object	No modification of the Table object.
Column deleted in the Table object	Corresponding column is deleted in the Table View object.

Connecting Table Views and Table Objects

Column deleted in the Table View object	No modification to the Table object.
Row inserted in the Table object	A new row is inserted in the Table View object if it is possible to determine the position. Otherwise a new row is added.
Row inserted in the view object	No modification to the Table object.
Column inserted in the Table object	A new column is inserted in the view object if it is possible to determine the position, otherwise a new column is added.
Column inserted in the Table View object	No modification to the Table object.
Table object cleared	All Table View rows with a connection to the Table object are cleared.
Table View object cleared	Each Table View cell with a connection to the Table object clears the corresponding value in the Table object.
Table object deleted	Each Table View row and each column with a connection to the Table object is removed.
Table View object deleted	No modification to the Table object.

Drag and Drop with Table Views

The SAP Table View control implements several standard drag and drop scenarios. The drag and drop behavior is defined through the DragDrop property of the Table View object.

Standard drag and drop may be disabled, only dragging may be enabled, only dropping may be enabled or both drag and drop is enabled. Standard drag-and-drop operations are always performed using CF_TEXT format. Standard drag-and-drop operations within one control support move and copy operations; drag and drop between two controls supports only copy operations as default implementation.

The data transport scenarios mentioned also work across process boundaries.

If the default drag and drop implementation is not sufficient, you can implement event handlers for the [DropEnter \[Page 411\]](#), [Drop \[Page 412\]](#), [DragSourceFill \[Page 410\]](#) and DragComplete events to fulfill more complicated requirements.

Columns Collection Object

The columns collection is a collection of [column \[Page 423\]](#) objects. It is implemented as a [named collection \[Page 147\]](#). The Columns collection object maintains the number and arrangement of columns, the column headings and column heading fonts.

[Properties \[Page 418\]](#)

[Methods \[Page 419\]](#)

Columns Collection Properties

The Columns collection object has the following properties:

Columns Collection Properties

Name	Parameter	Type	Description
Item	Variant valIndex	Object	Returns the object indexed by <i>valIndex</i> . If <i>valIndex</i> is from type VT_BSTR, the object is returned by name, otherwise the value is converted to an integer and used as index.
Count		Long	Returns or sets the number of objects in the collection. Legal values for this property range from 0 to 255.
Height		Short	Defines the height of the column headers. The height is measured in twips.
Font [Page 152]		Object	Defines the font used for the column headers.

Columns Collection Methods

The Columns collection object has the following methods:

Columns Collection Methods

Name	Parameters	Return Type	Description
Add [Page 420]	Variant vaWhat	Object	Adds an object to the collection.
Remove [Page 421]	Variant valIndex	Boolean	Remove an object from the collection.
Insert [Page 422]	Variant valIndex Variant vaWhat	Object	Inserts an object into the collection.

Columns Collection Methods: Add

Columns Collection Methods: Add

Purpose

Adds an object to the Columns collection.

Syntax

The Add method has the syntax:

```
Add (Variant vaWhat)
```

Returns

type Object

Description

The Add method adds an object to the Columns collection and returns the new object.

Legal types for the parameter **vaWhat** are:

Type	Description
Object that does not equal nothing [Page 453] .	An already-existing object is added. This object must be a Column [Page 423] object. This variant type is used to display the same column more than once.
Object that equals nothing, VT_EMPTY or VT_ERROR	A new Column object is created and added to the collection.

Columns Collection Methods: Remove

Purpose

Removes an object from the Columns collection.

Syntax

The Remove method has the syntax:

```
Remove (Variant vaIndex)
```

Returns

type Boolean

Description

Legal types for the parameter valIndex are :

Type	Description
Any type convertible to an integer.	The parameter <i>valIndex</i> is converted to an integer and used as index in the collection. The corresponding object is removed from the collection and marked as invalid. Any further attempt to work on this object leads to an 'Invalid object' exception.
String not convertible to an integer.	The parameter <i>valIndex</i> is used to search and remove the object with the corresponding name from the collection.
Object	The parameter <i>valIndex</i> is used to search and remove the objects which are the same as <i>valIndex</i> .



If *valIndex* is any type except Object, only one element is removed from the collection. If *valIndex* is of type Object, all columns addressed by *valIndex* are removed. This occurs if the object was inserted or added more often than once.

Columns Collection Methods: Insert

Columns Collection Methods: Insert

Purpose

Inserts an object into the Columns collection.

Syntax

The Insert method has the syntax:

```
Insert (Variant vaIndex, Variant vaWhat)
```

Returns

type Object

Description

This method inserts an object into the collection. Legal types for the parameter *valIndex* are:

Type	Description
Any type convertible to an integer	The parameter <i>valIndex</i> is converted to an integer and used as index in the collection. The object is inserted prior to the object found.
String that is not convertible to an integer	The parameter <i>valIndex</i> is used to search the object by name. Prior to this object the new object is inserted.
Object	The parameter <i>valIndex</i> is used to search the objects. Prior to this object the new object is inserted.

Legal types for the parameter *vaWhat* are the same as for the [Add \[Page\] 420](#) Method.



Using a *vaWhat* parameter that is a Column object is legal. A column may be inserted as many times as desired.

Column Object

The Column object is a [named object \[Page 457\]](#). It controls the behavior, arrangement and display of one column. Data may also be accessed through the column object.

[Properties \[Page 424\]](#)

[Methods \[Page 428\]](#)

Column Object Properties

Column Object Properties

The Column object has the following properties:

Column Properties

Name	Parameter	Type	Description
Type [Page 426]		CViewColType	Defines the type of column.
Header		String	Column heading text.
Length		Short	not implemented yet.
Protection		Boolean	Sets this property to TRUE to protect the entire column against user input and modifications. This flag is used only if the Table View object's EnableProtection [Page 402] property is set to TRUE.
Width		Short	Width of this column. The value is measured in average character width units.
Font [Page 152]		Object	Font used for this column.
Format		String	Defines the format for all cells in this column.
Data		Array of Variant	Returns the data of the entire column in a two-dimensional array. The first dimension always equals one, the second dimension equals the number of cells in the column.
Index		Long	Returns the index in the object's Columns [Page 417] collection. Read-only.
Visible		Boolean	Indicates whether the column should be visible. If this property is set to TRUE, the column becomes invisible by setting the columns width to 0.
Alignment [Page 427]		Short	Controls the alignment of the cells in this column.
TableIndex		Long	Returns the index of an associated column in a SAP Table object [Page 220] if a connection to a SAP Table object is established. (See Connecting Table Views and Table Objects [Page 413]).
Value	Long Row	Variant	Returns or sets the value for the cell with the corresponding <i>Row</i> index.
Cell	Long Row	Object	Returns a Cell [Page 438] object for the cell with the corresponding <i>Row</i> index.

Column Object Properties

Formula	Long Row	Variant	Same as value.
----------------	----------	---------	----------------

Column Object Properties: Type**Purpose**

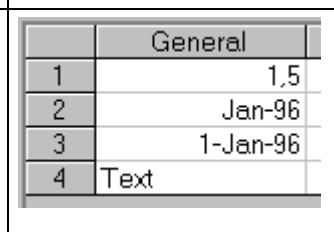
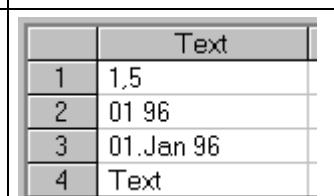
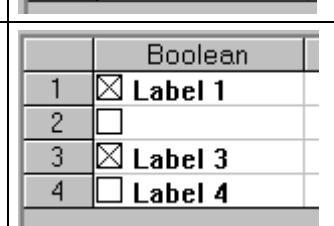
Specifies the type of a Column object.

Returns

type CviewColType

Description

Legal values for Type are :

Value	Description	Appearance
tavColumnGeneral = 1	General data type. The data is converted automatically to numeric, date or time values (if possible) and displayed accordingly. If the Alignment [Page 427] property is set to tavColumnAlignGeneral, the display is also justified automatically.	
tavColumnText = 2	All data is treated as text data. The display is aligned according to the Alignment property. Data is not converted. This is especially necessary if text data like '0001' should be stored in a cell.	
tavColumnBoolean = 3	All data is stored as Boolean. A check box is used for displaying and modifying data.	
tavColumnSelection = 4	not implemented yet	
tavColumnNumeric = 5	Same as tavColumnGeneral.	
tavColumnObject = 6	not implemented yet	
tavDate = 7	not implemented yet	
tavTime = 8	not implemented yet	

Column Object Properties: Alignment

Column Object Properties: Alignment

Purpose

Controls the alignment of the cells in the Column object.

Returns

type Short

Description

Legal values for Alignment are:

Value	Description
tavColumnAlignGeneral = 1	Text is aligned left justified, numerical, date and time values are aligned right justified.
tavColumnAlignLeft = 2	All output is aligned left justified.
tavColumnAlignCenter = 3	All output is centered.
tavColumnAlignRight = 4	All output is aligned right justified.
tavColumnAlignFill = 5	All output is aligned left justified and repeated until the entire column width is filled.

Column Object Methods

The Column object has the methods:

Column Methods

Name	Parameter	Type	Description
Clear	void	void	Clear the entire column.

Rows Collection Object

The Rows collection is a collection of [Row \[Page 435\]](#) objects. It is implemented as a [standard collection \[Page 145\]](#). The rows collection controls the number and arrangement of rows, row headings and row heading fonts.

[Properties \[Page 430\]](#)

[Methods \[Page 431\]](#)

Rows Collection Properties

Rows Collection Properties

The Rows collection object has the following properties:

Rows Collection Properties

Name	Parameters	Type	Description
Item	Variant valIndex	Object	Returns the object indexed by valIndex. <i>ValIndex</i> may be from any variant type convertible to an integer value.
Count		Long	Returns or set the number of objects in this collection. Legal values for this property are 0 to 16384.
Width		Short	Defines the width of the row headers in average character units.
Font [Page 152]		Object	Defines the font used for the row headers.

Rows Collection Methods

Rows Collection Methods

The Rows collection object has the following methods:

Rows Collection Methods

Name	Parameters	Return Type	Description
Add [Page 432]	Variant vaWhat	Object	Adds an object to the collection.
Remove [Page 433]	Variant valIndex	Boolean	Removes an object from the collection.
Insert [Page 434]	Long Index Variant vaWhat	Object	Inserts an object into the collection.

Rows Collection Methods: Add

Rows Collection Methods: Add

Purpose

Adds an object to the Rows collection

Syntax

The Add method has the syntax:

```
Add(Variant vaWhat)
```

Returns

type Object

Description

This method adds an object to the Rows collection and returns the object added. Legal types for the parameter *vaWhat* are :

Type	Description
Any variant data type convertible to an integer value	A new Column object is created and added to the collection.

Rows Collection Methods: Remove

Purpose

Removes an object from the Rows collection

Syntax

The Remove method has the syntax:

Remove (Variant vaIndex)

Returns

type Object

Description

Removes an object from the Rows collection. Legal types for the parameter *valIndex* are:

Type	Description
Any variant data type convertible to an integer value.	The parameter <i>valIndex</i> is converted to an integer and used as index in the collection. The corresponding object is removed from the collection and marked as invalid. Any further attempt to work on this object leads to an 'Invalid object' exception.
Object	The parameter <i>valIndex</i> is used to search and remove the objects which are the same as <i>valIndex</i> .



If *valIndex* is of type Object, the entire collection is searched for the object. This may be very ineffective for large tables. Since a [Rows \[Page 429\]](#) collection may hold the same object only once, it is better to remove a row by its index or to invoke the remove method on the row object.

```

` do not use this construction :

Rows.Remove (Row)
` remove by Row index :
Rows.Remove (Row.Index)
` remove by row remove method :

Row.Remove

```

Rows Collection Methods: Insert

Rows Collection Methods: Insert

Purpose

Inserts an object into the Rows collection

Syntax

The Insert method has the syntax:

```
Remove (Variant vaIndex, Variant vaWhat)
```

Returns

type Object

Description

Inserts an object into the collection. Legal types for the parameter *valIndex* are:

Type	Description
Any type convertible to an integer.	The parameter <i>valIndex</i> is converted to an integer and used as index in the collection. The object is inserted prior to the object found.
Object	The parameter <i>valIndex</i> is used to search the objects. Prior to this object the new object is inserted.

Legal types for the parameter *vaWhat* are the same as for the [Add \[Page 432\]](#) Method.



If *valIndex* is of type Object, the entire collection is searched for the object. This may be very ineffective. Since a Row collection may hold the same object only once, it is better to use the [Row \[Page 435\]](#) object's Index property.

```
Rows.Insert (Row.Index, )
```

Row Object

The Row object manages the behavior, arrangement and display of a single row. Data may also be accessed through the row object.

[Properties \[Page 436\]](#)

[Methods \[Page 437\]](#)

Row Object Properties

Row Object Properties

The Row object has the following properties:

Row Object Properties

Name	Parameters	Type	Description
Height		Short	Height of row. The value is measured in twips.
Header		String	Row heading text.
Protection		Boolean	Sets this property to TRUE to protect the entire row against user input and modifications. This flag is only used if the Table View object's [Page 451] EnableProtection [Page 402] property is set to TRUE.
Font [Page 152]		Object	Font used for this row.
Data		Array of Variant	Returns the data of the entire row in a two-dimensional array. The first dimension equals the number of cells in the row, the second dimension always equals one.
Index		Long	Returns the index in the object's Rows [Page 429] collection. Read-only.
TableIndex		Long	Returns the index of an associated row in a Table View Object [Page 396] , if a connection to a SAP Table object is established. (See Connecting Table Views and Table Objects [Page 413])
Value	Long Column	Variant	Returns or sets the value for the cell with the according <i>Column</i> index.
Formula	Long Column	Variant	Same as value.
Cell	Long Column	Object	Returns a Cell [Page 438] object for the cell with the according <i>Column</i> index.

Row Object Methods

The Row object has the following methods:

Row Object Methods

Name	Description
Clear	Clears the entire row.
Remove	Removes the row from its Rows [Page 429] collection.

Cell Object

The Cell object allows access data in a single cell. It is also possible to protect a single cell against changes or to change the appearance of the cell's font. Special properties for different types of cells are available (see also the [Type \[Page 426\]](#) property of [Column \[Page 423\]](#) object).

[Properties \[Page 439\]](#)

[Methods \[Page 440\]](#)

Cell Object Properties

The Cell object has the properties:

Cell Object Properties

Name	Type	Description
Protection	Boolean	Set this property to TRUE to protect the cell against user input and modifications. This flag is only used, if the Table View object's EnableProtection [Page 402] property is set to TRUE.
Font [Page 152]	Object	Font used for this cell.
Value	Variant	Returns or sets the value for the cell.
Formula	Variant	Same as value.
CheckLabel	String	Set or returns the label for the check box. Check boxes are only displayed for cells in columns of Type [Page 426] tvaColumnBoolean.
Row	Object	Returns the Row [Page 435] object for this cell. Read-only.
Column	Object	Returns the Column [Page 423] object for this cell. Read-only.

Cell Object Methods

Cell Object Methods

The Cell object has the following methods:

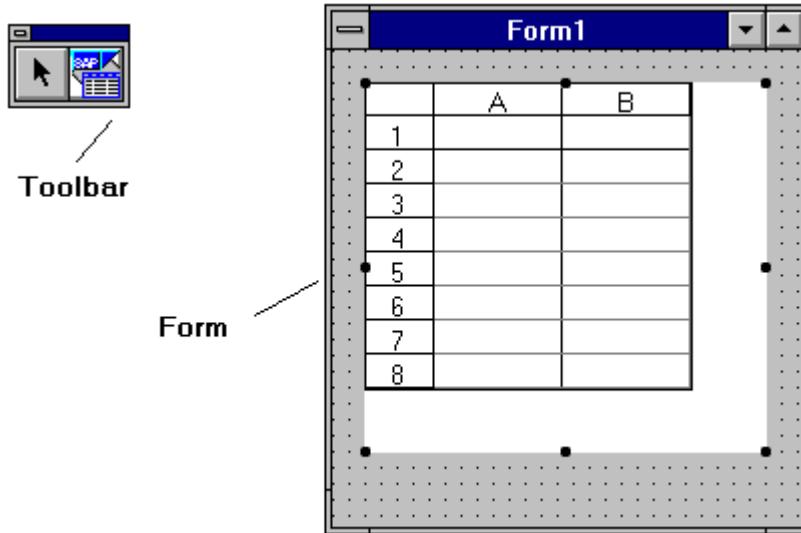
Cell Object Methods

Name	Description
Clear	Clears the cell.

Design Environment Property Pages

Most OLE control-aware container programs and development environments support a design environment that enables the user to interactively design forms and dialogs. Within this design environment, the user can choose controls from a toolbar window and place on the form.

The SAP Table View control is indicated by . A typical view on the design environment is shown below.



The control's appearance depends on its configuration, just as the available properties for the control depend on the container. Some containers add additional properties like Visible, Default, Parent or Cancel. These extended properties are described in the container's manual.

It is also up to the container whether a single property is displayed as in Microsoft Visual Basic, or whether the properties are only available through the property pages at design time. Nevertheless, every control container should support a right mouse button menu on the control with a menu entry properties, which invokes the control's property pages. The SAP Table View control supports seven property pages:

- [Table View Property Page: General \[Page 442\]](#)
- [Table View Property Page: Flags \[Page 443\]](#)
- [Table View Property Page: Events \[Page 444\]](#)
- [Table View Property Page: Columns \[Page 445\]](#)
- [Table View Property Page: Rows \[Page 446\]](#)
- [Table View Property Page: Fonts \[Page 447\]](#)
- [Table View Property Page: Colors \[Page 448\]](#)

Table View Property Page: General

Table View Property Page: General

Set the following properties just as you would in a Visual Basic program. For possible values of properties, see [Table View Properties \[Page 397\]](#).

Property	Description
Row Count	Enter the number of rows.
Column Count	Enter the number of rows.
Fixed Rows	Enter the number of fixed (non-scrollable) rows.
Fixed Columns	Enter the number of fixed (non-scrollable) columns.
DragDrop Mode	Choose the desired drag-and-drop mode.
Selection	Choose the desired SelectMode value.
Configuration	Choose the desired configuration for a connection to a SAP Table object [Page 413] .
Row Header Width	Enter the row header width in average character units.
Column Header Height	Enter the column header height in twips.

Table View Property Page: Flags

Set the following properties just as you would in a Visual Basic program. For possible values of properties, see [Table View Properties \[Page 397\]](#).

Property	Description
Vertical Scrollbar	Enable vertical scroll bars.
Horizontal Scrollbar	Enable horizontal scroll bars.
Protection	Set the EnableProtection [Page 402] property.
Show Grid Lines	Selected whether grid lines should be drawn.
HideSelection	Selected whether selection should be highlighted if the control does not own the input focus.
Show Row Headers	Select whether row headers should be displayed.
Show Column Headers	Select whether column headers should be displayed.

Table View Property Page: Events

This property page is used to enable or disable events fired to the container during runtime. The main reason to disable events is to improve performance. More sophisticated scenarios can set these properties dynamically (using the Events property of the Table View object at runtime).

For more information on enabling and disabling events, see:

[Table View Properties \[Page 397\]](#)

[Table View Events \[Page 406\]](#)

Table View Property Page: Columns

Table View Property Page: Columns

Set the following properties just as you would in a Visual Basic program. For more information, see:

[Column Object \[Page 423\]](#)

[Column Object Properties \[Page 424\]](#)

[Columns Collection Object \[Page 417\]](#)

Property	Description
Column	Select the desired column.
Header	Enter the column header.
TableIndex	Enter the index of the associated table column for a connection to a SAP Table object [Page 413] .
Width	Enter the width of the column in average character units.
Format	Enter the desired format string.
Name	Enter the name of the column. The Column object is a Named Object [Page 457] in the Columns collection. The Name property may be used to access the Column object through dynamic properties [Page 293] .
Alignment	Select the desired Alignment [Page 427] for this column.
Type	Select the Type [Page 426] for this column.
Visible	Set the column to visible or invisible
Protected	Mark the column as protected. (See also EnableProtection [Page 402])

Table View Property Page: Rows

Table View Property Page: Rows

Set the following properties just as you would in a Visual Basic program. For more information, see:

[Row Object \[Page 435\]](#)

[Row Object Properties \[Page 436\]](#)

[Rows Collection Object \[Page 429\]](#)

Property	Description
Row	Select the desired row.
Header	Enter the row header.
Height	Enter the height of the row in twips.
TableIndex	Enter the index of the associated table column for a connection to a SAP Table object [Page 413] .
Protected	Mark the row as protected. (See also EnableProtection [Page 402])

Table View Property Page: Fonts

This property page allows the user to define the default font [\[Page 152\]](#) used by the control. Usually this is the font assigned to the entire form by the control container.

Table View Property Page: Colors

Table View Property Page: Colors

Set the following properties just as you would in a Visual Basic program. For more information, see:

[Table View Properties \[Page 397\]](#)

Property	Description
ForeColor	Select the desired ForeColor [Page 397] , also used as standard text color.
BackColor	Select the desired ForeColor [Page 397] for the control.

Glossary

[Collection \[Page 450\]](#)

[Root Control, Root \[Page 451\]](#)

[Root Node \[Page 452\]](#)

[Nothing \[Page 453\]](#)

[CreateObject \[Page 454\]](#)

[Active Cell \[Page 455\]](#)

[Range \[Page 456\]](#)

[Named Object \[Page 457\]](#)

Collection

Collection

A collection object maintains a set of other objects that all have the same type. A collection usually supports methods like Item, Add, Insert and Remove (see [SAP Standard Collection \[Page 145\]](#)). You can iterate through a collection by using **For ... Each** loops in Visual Basic or the IEnumVARIANT interface in C++ (see [Node Object Property: AllChildren \[Page 251\]](#)).

Root Control, Root

The root control is the highest level object in the object hierarchy. It is accessible through the object returned by [CreateObject \[Page 454\]](#).

Root Node

Root Node

A root node is a [Node \[Page 245\]](#) located at the highest level of the tree's hierarchy. It has no parent node and is part of the Table View object's [Nodes \[Page 237\]](#) collection.

Nothing

VBA key word for an empty object. This value is the same as NULL in C++ or nil in Pascal.

CreateObject

CreateObject

VBA function to create an OLE object. See your Visual Basic documentation for complete information.

Active Cell

The active cell is the cell that receives user input if the SAP Table View control owns the input focus. Subsequent user input is done in this cell. The active cell is marked with a black simple frame.

Range

Range

A range is a rectangular area that groups cells together. A range is usually displayed using inverted colors.

Named Object

A Named Object is an object that exposes a property called Name of type String and is stored in a [Named Collection \[Page 147\]](#). Within the Named collection, the object is accessible through its name.

The Logon Control

This section contains the following topics:

Introduction

[Introduction \[Page 459\]](#)

[Logon Control Object Hierarchy \[Page 460\]](#)

[Using the SAP Logon Control \[Page 461\]](#)

Control and Object Reference

[Logon Object \[Page 465\]](#)

[Connection Object \[Page 490\]](#)

Programming Guide

[Code Examples \[Page 498\]](#)

[Glossary \[Page 501\]](#)

Introduction

The SAP Logon control is an ActiveX control that encapsulates the connection process as provided by the RFC Library. It eases the way the desktop programmer can connect a client application with R/3. The SAP Logon control is designed in such a way that it can be used optimally to work with Visual Basic 4.0, VBA and C++ through provided wrapper classes. Future releases will also support Dual Interface in the 32-Bit version.

This version of the component can be created via the Custom Control Adding in Visual Basic (VB) 4.0 and Visual C (VC) 4.0 or by the following line of code:

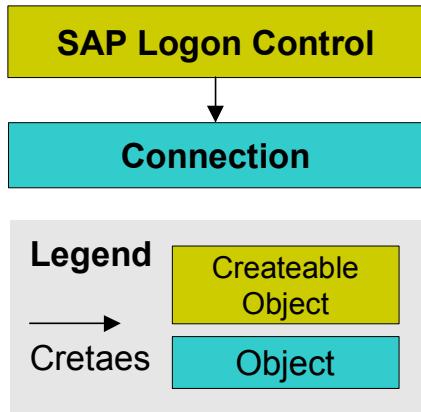
```
Dim oLogonCtrl as Object  
Set oLogonCtrl = CreateObject ("SAP.Logoncontrol.1")
```



Note that the Logon Control is being replaced by the [DCOM Connector Logon Component \[Ext.\]](#) for applications using the SAP DCOM Connector.

Logon Control Object Hierarchy

Logon Control Object Hierarchy



Using the SAP Logon Control

The SAP Logon control handles the remote access to an R/3 System. To access R/3, your program must do the following:

1. Obtain a Logon object.

Get this object by calling `CreateObject("SAP.Logoncontrol")` or by inserting a Logon control directly into a form from a toolbar.

2. Obtain a [Connection Object \[Page 490\]](#) object.

Get a Connection object by using the Logon object's [Logon Method: NewConnection \[Page 488\]](#) method or by handling a Click Event.

3. Call the Connection object's [Logon \[Page 494\]](#) method.

The Logon method tries to establish the connection immediately and returns TRUE if successful, otherwise FALSE. In the latter case, the [IsConnected \[Page 492\]](#) property provides detailed information about the source of the failure.

The following topics are available:

[Logon Object \[Page 465\]](#)

[Connection Object \[Page 490\]](#)

[Using Logon Controls in Design Mode \[Page 496\]](#)

[Code Examples \[Page 498\]](#)

[Connecting Directly with the Logon Control \[Page 499\]](#)

[Logging on Silently \[Page 500\]](#)

Connecting to the R/3 System

Connecting to the R/3 System

The Connection object is a property of the Function control and the Transaction control. It is created automatically when you request the relevant collection for either the Function or the Transaction control. The Logon control creates connections. If you have a Connection object obtained from another control or directly from the Logon control, you can set it in the Function or Transaction control.

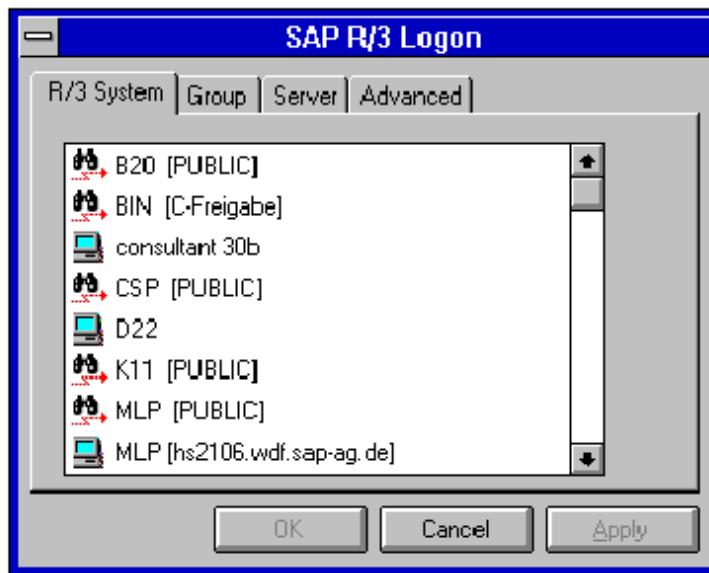
The Connection object's Logon method establishes the connection to the R/3 System. This method has a parameter that can suppress the dialog box when the user logs in. This parameter allows you to automatically log the user into a fixed account or provide your own logon dialogs.

To establish a connection, you must call the Logon method for your Connection object. For more information, see [Using the Logon Control to Connect to R/3 \[Page 463\]](#).

Using the Logon Control to Connect to R/3

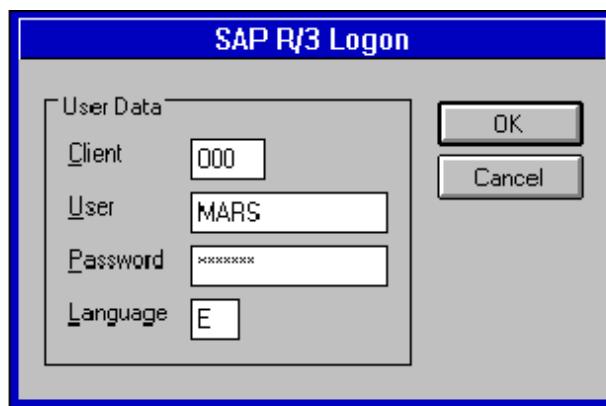
Most controls deal directly with R/3 and therefore need a connection to the application server. To get a connection, you need to create a Logon object and call the *NewConnection* method on that object. The result of this call is the connection object. To log on to the R/3 System, you use the Logon control. The example code generates the window shown below:

```
' Create the Control.  
  
Dim LogonControl As Object  
  
set LogonControl = CreateObject ("SAP.LogonControl.1")  
  
' Create the connection.  
  
Dim conn As Object  
  
set conn = LogonControl.NewConnection  
  
' Log on.  
  
if conn.Logon (0, True) <> True then  
    MsgBox "Cannot log on!"  
  
End If
```



You can set parameters for the logon process such as user name and password. See the <LOGON control SECTION> for details. Depending on the parameters you set in the Logon control, one or two dialog boxes are displayed that ask for the input needed to make the connection. One of these dialog boxes is:

Using the Logon Control to Connect to R/3



Logon Object

The SAP Logon object is the highest object in the Logon control hierarchy. It is obtained by calling CreateObject in Visual Basic or by inserting the control directly into a design-mode form from a toolbar.

The following topics are available:

[Properties \[Page 466\]](#)

[Methods \[Page 486\]](#)

[Events \[Page 489\]](#)

Logon Object Properties

Logon Object Properties

The Logon object has the following properties:

SAP Logon Properties

Name	Type	Description
Events [Page 467]	Long	Enables or disables events fired by the control.
Caption	String	Text on the button.
BackColor	Color	Background color of the button.
hWnd	HWND	Window handle of the button.
Enabled	Enabled	Enables the button.
Font	Font	Font of button text.
Parent	HWND	Window handle of Parent Window.
Default	Boolean	Button Default pushbutton.
ApplicationName	String	Name of the application that uses the Logon control, used to identify connections.
System	String	SAP R/3 System name.
ApplicationServer	String	Application Server of the R/3 System.
SystemNumber	Long	System number of the R/3 System.
MessageServer	String	Message Server of the R/3 system that is doing the load balancing.
GroupName	String	Name of the group of R/3 Application Servers you want to connect to.
TraceLevel	Long	Debug On/Off.
RFCWithDialog [Page 482]	Long	Enables support for RFC with SAPGUI (3.0C or later).
Client	String	Client in the R/3 System.
User	String	User of the R/3 System.
Language	String	Language you use want to use in the R/3 System.

Logon Property: Events

Purpose

Enables or disables [events \[Page 489\]](#) fired by the Logon object control.

Description

Events or groups of events can be enabled or disabled using the Events property. Disabling events may lead to improved performance, especially for operations with large chunks of data. Events may also be turned on and off temporarily. The following values are available for the Events property:

Events property values

tloDisableAllLogonEvents = 0	Disable all events.		
tloEnableOnClick = 1	Fire event after logon button is clicked.		
tloEnableOnLogoff = 2	Fire event after associated connection is disconnected.		
tloEnableOnError = 4	Fire event after an error has occurred.		
tloEnableOnCancel = 8	Fire event after the user has clicked the Cancel button in the logon dialog.		
tloEnableAllLogonEvents = 32767	Enable all events.		

All values may be combined through **and** or **or** operations.



```
Rem Enable OnClick and OnLogoff event
MyControl.Events = MyControl.Events or _
    EnableOnClick or EnableOnLogoff
Rem Disable OnCancel event
MyControl.Events = MyControl.Events and not _
    (EnableOnCancel)
```

Logon Property: Caption

Logon Property: Caption

Purpose

Gets or sets the button text.

Returns

type String.

Logon Property: BackColor

Logon Property: BackColor

Purpose

Gets or sets the background color of the button.

Returns

type Color.

Logon Property: hWnd

Logon Property: hWnd

Purpose

Gets or sets the window handle of the button.

Returns

type HWND.

Logon Property: Enabled

Logon Property: Enabled

Purpose

Enables the button.

Returns

type Enabled.

Logon Property: Font

Logon Property: Font

Purpose

Gets or sets the font of the button text.

Returns

type Font.

Logon Property: Parent

Purpose

Window handle of the parent window.

Returns

type HWND.

Logon Property: Default

Logon Property: Default

Purpose

Button default pushbutton.

Returns

type Boolean.

Logon Property: ApplicationName

Logon Property: ApplicationName

Purpose

Name of the application that uses the Logon control, used to identify connections.

Returns

type String.

Logon Property: System

Logon Property: System

Purpose

SAP R/3 System name.

Returns

type String.

Logon Property: ApplicationServer

Purpose

Application server of the R/3 System.

Returns

type String.

Logon Property: SystemNumber

Logon Property: SystemNumber

Purpose

System number of the R/3 System.

Returns

type Long.

Logon Property: MessageServer

Purpose

Message server for the R/3 System that is performing load balancing.

Returns

type String.

Logon Property: GroupName

Logon Property: GroupName

Purpose

Gets or sets the name of the group of R/3 application servers you want to connect to.

Returns

type String.

Logon Property: TraceLevel

Purpose

Sets debugging mode on or off.

Returns

type Long.

Logon Property: RFCWithDialog

Logon Property: RFCWithDialog

Purpose

Enables supports RFC with SAP GUI (3.0C or later).

Returns

type Long.

Description

Set this property to a non-zero value when you want to start the SAPGUI before making RFC calls. The running SAPGUI allows you to call RFC functions that display SAP screens.

You can only use this property for connecting against R/3 Systems with Release 3.0C or later.

Logon Property: Client

Purpose

Gets or sets the client in the R/3 System.

Returns

type String.

Logon Property: User

Logon Property: User

Purpose

Gets or sets the user of the R/3 System.

Returns

type String.

Logon Property: Language

Purpose

Gets or sets the language you want to use in the R/3 System.

Returns

type String.

Logon Object Methods

Logon Object Methods

The Logon object has the following methods:

SAP Logon Methods

Name	Return Type	Description
Enable3D [Page 487]	void	This method makes sure that the dialogs provided have a 3D look and feel.
NewConnection [Page 488]	Object	Returns and produces a new object of type Connection [Page 502] .
AboutBox		Displays the AboutBox dialog.

Logon Method: Enable3D

Purpose

This method makes sure that the provided dialogs have the 3D look and feel.

Parameters

None.

Description

This property is not needed in Visual Basic 4.0 or Visual C++ 4.0. It is needed in Excel 7.0 due to the fact that the subclassing mechanism does not work directly in that environment.

Logon Method: NewConnection

Logon Method: NewConnection

Purpose

Creates and returns a new object of type [Connection \[Page 502\]](#).

Parameters

type Object.

Description

The Connection object created handles the connection to the R/3 System. All property values are assigned to the newly-obtained Connection object.

Logon Object Events

The Logon object has the following events:

SAP Logon Control Events

Name	Parameters	Description
Error	Object Connection	An error occurred in one of the objects.
Click	Object Connection	The user has clicked on the logon button.
Logoff	Object Connection	The connection has been closed.
Cancel	Object Connection	During the logon for the connection, the user clicked on the Cancel button in the logon dialog.



All properties of the Logon object serve as the default assignment for a Connection object obtained by the method NewConnection or the Click event.

Connection Object

Connection Object

The Connection object holds all information about the status and the parameters of one connection to an R/3 System. All Connection object properties can either be set in the design environment using a [SAP Logon control \[Page 465\]](#) object or before the method [Logon \[Page 494\]](#) is called on the Connection object. After a successful logon has taken place, all connection parameters are set to read-only, due to the fact that an established connection cannot be changed. The property [IsConnected \[Page 492\]](#) displays the actual status of the connection.



Each R/3 user must use a password to logon to an R/3 System. The SAP Logon control will never save the password, it must be either provided by the client application (to do a silent logon) or it is prompted during logon. All password access is write-only.

The following topics are available:

[Properties \[Page 491\]](#)

[Methods \[Page 493\]](#)

Connection Object Properties

The Connection object has the following properties:

Connection Properties

Name	Type	Description
ApplicationName	String	Name of the application that uses the Logon control, used to identify connections.
System	String	SAP R/3 System name.
ApplicationServer	String	Application server of the R/3 System.
SystemNumber	Long	System number of the R/3 System.
MessageServer	String	Message Server of the R/3 System that is doing the load balancing.
GroupName	String	Name of the group of R/3 Application Servers you want to connect to.
TraceLevel	Long	Debug On/Off.
RFCWithDialog [Page 482]	Long	Support RFC with SAPGUI (3.0C or later).
Client	String	Client in the R/3 System.
User	String	User of the R/3 System.
Password	String	Password of the user. Write-only.
Language	String	Language you use want to use in R/3.
IsConnected [Page 492]	CRfcConnectionStatus	Current status of the Connection object. Read-only.

Connection Property: IsConnected

Connection Property: IsConnected

Purpose

Checks the current status of the Connection object and the R/3 connection.

Returns

type String.

Description

Use this property after the [Logon \[Page 494\]](#) method has returned FALSE. All valid values for the IsProperty property are given in the RfcConnectionStatus table:

RfcConnectionStatus values

tloRfcNotConnected= 0	The R/3 connection is not established, Logon was not called, or Logoff has been called.
tloRfcConnected = 1	The R/3 connection is established.
TloRfcConnectCancel= 2	The R/3 connection is not established due to the fact that the user pressed the Cancel Button during Logon.
tloRfcConnectParameterMissing = 4	The R/3 connection could not be established, due to the fact that a silent logon was attempted with connection parameters.
TloRfcConnectFailed = 8	The R/3 connection failed. Call LastError to display additional information.

Connection Object Methods

The Connection object has the following methods:

Name	Parameters	Description
Logon [Page 494]	HWND hWnd Boolean Silent	Establishes a connection to the R/3 System. Returns TRUE if successful, FALSE otherwise.
Logoff [Page 495]		Disconnects a connection from the R/3 object.
Reconnect		Tries to reestablish a connection to the R/3 System.
LastError		Displays a dialog which shows all information about the last RFC Error.
SystemInformation		Displays a dialog with all system information about the connected R/3 System.

Connection Method: Logon

Purpose

This method performs a remote logon to the R/3 System.

Syntax

The Logon method has the syntax:

Logon (HWND ParentWindow, Boolean Silent)

Returns

type Boolean.

Description

Set the Silent parameter to FALSE if you want the system to display a Logon dialog. If not, set all parameters in advance: the system will log the user on silently. If you set Silent to TRUE and omit logon parameter values, the method always fails.

If the method succeeds, it returns TRUE, otherwise FALSE. When FALSE, use the [IsConnected \[Page 492\]](#) property to determine the type of logon failure.

Connection Method: Logoff

Purpose

Disconnects an established RFC connection.

Returns

void.

Description

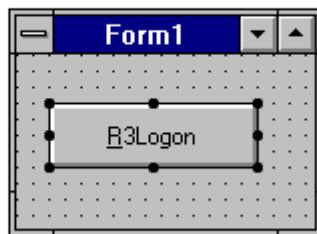
The Logoff method resets all Connection object properties to Read/Write mode. After this, all properties may be changed by the client program. Once disconnected, all function calls sent to the connection will fail.

Using Logon Controls in Design Mode

Using Logon Controls in Design Mode

The SAP Logon control supports most of its configuration in the Design Environment of Visual C++ 4.0 and Visual Basic 4.0.

The following picture shows the appearance of the SAP Logon control in Visual Basic 4.0 Toolbox and as it is placed on a form.



If you have placed the control in a form, you can edit its properties by using the right mouse button. The following two graphics show the layout of the Logon control property pages:

- User Page



- System Page

Using Logon Controls in Design Mode



Code Examples

Code Examples

Example code for establishing a connection is provided in the following examples:

[Connecting Directly with the Logon Control \[Page 499\]](#)

[Logging on Silently \[Page 500\]](#)

Connecting Directly with the Logon Control

This example shows the direct use of the Logon control in an application serving as an OLE Automation Client:

```
Dim oLogonCtrl as Object
Dim oConnection as Object
Rem ***Create the Logon Control
Set oLogonCtrl = CreateObject ("SAP.Logoncontrol.1")
Rem ***Get a connection object
Set oConnection = oLogonCtrl.NewConnection
Rem ***Try to connect to the R/3 System
if oConnection.Logon (Form.hWnd, FALSE) = FALSE then
    MsgBox "R/3 connection failed"
end if
Rem ***Now the connection is established: you can call your functions
```

You can also place connection code in an event subroutine:

```
Dim oConnection as Object
Private Sub SAPLogonControl1_Click()
Set oConnection = SAPLogonControl1.NewConnection
Rem ***Try to connect to the R/3 System
if oConnection.Logon (Form.hWnd, FALSE) = FALSE then
    MsgBox "R/3 connection failed"
    exit sub
end sub
Rem ***Now the connection is established: you can call your functions
```

Logging on Silently

This example shows how to use the Connection object properties so that the Connection object can connect to R/3 without displaying any dialog:

```
Dim g_oConnection as Object
Private Sub SAPLogonControl1_Click()

    Rem ***Save Connection object to your global variable
    Set g_oConnection = SAPLogonControl1.NewConnection

    Rem ***Either all parameters are set by the design environment (except
    Rem ***the password) or they are hardcoded, as in the following lines
    Rem ***The following lines depend on your R/3 environment
    Rem ***These are the system parameters
    g_oConnection.System ="R30"                      'Name of your R/3 System
    g_oConnection.ApplicationServer = "hs2001"        'Applic.Server of R/3
    System
    g_oConnection.SystemNumber = 0                   'SystemNumber of your R/3 System

    rem ***User specific data
    g_oConnection.User = "MyUserName"
    g_oConnection.Password = "secret"
    g_oConnection.Client = "000"
    g_oConnection.Language = "E"

    Rem ***Try to connect to the R/3 System
    if g_oConnection.Logon (Form.hWnd, TRUE) = FALSE then
        MsgBox "R/3 connection failed"
        exit sub
    end if
    end sub
    Rem ***Now the connection is established: you can call your functions
```

Glossary

[Connection \[Page 502\]](#)

[Password \[Page 503\]](#)

[Nothing \[Page 504\]](#)

[CreateObject \[Page 505\]](#)

Connection

Connection

The Connection object holds all information about the actual status of a remote connection to an R/3 System and is used to establish one.

Password

Each R/3 user must use the password to logon to an R/3 System. The SAP Logon control never saves the password: it must either be provided by the client application (in a silent logon) or the user will be prompted during logon. All password access is write-only.

Nothing**Nothing**

VBA key word for an empty object. This value is the same as NULL in C++ or nil in Pascal.

CreateObject

VBA function to create an OLE object. See VBA help for more information.

DCOM Connector-compatible Components

DCOM Connector-compatible Components

The SAPBrowser Control

The *SAPBrowser control* provides a window consisting of three panes for displaying SAP R/3 BAPI and RFC metadata information (metadata is information about data).

The [SAP Assistant product \[Page 16\]](#) uses the SAP Browser control for displaying BAPI and RFC metadata to its users. See the discussion of the [SAP Assistant screen \[Page 25\]](#) for a description of the Browser parts. Also see related topics in the SAP for an example of the implementation of the SAP Browser control in an application.

The SAPBrowser control also allows you to export properly formatted metadata information to MS Excel (any version), a search feature, and documentation. The SAPBrowser control exposes several methods to enable the container application to control and automate metadata browsing.

[Properties \[Page 508\]](#)

[Methods \[Page 509\]](#)

[Example \[Page 539\]](#)

Properties

Properties

This control supports all standard window properties.

Methods

[AddBAPIAppObject \[Page 510\]](#)
[AddBAPIObject \[Page 511\]](#)
[AddBAPISearchObject \[Page 512\]](#)
[AddRFCFunctions \[Page 514\]](#)
[AddRFCFunctionGroups \[Page 513\]](#)
[AddSearchRFCFunctions \[Page 516\]](#)
[AddSearchRFCFunctionGroups \[Page 515\]](#)
[CallFunction \[Page 517\]](#)
[ClearAll \[Page 518\]](#)
[ClearBAPITab \[Page 519\]](#)
[ClearRFCTab \[Page 520\]](#)
[ClearSearchTab \[Page 521\]](#)
[DeleteObject \[Page 522\]](#)
[EnableBAPITab \[Page 523\]](#)
[EnableRFCTab \[Page 524\]](#)
[EnableSearchTab \[Page 525\]](#)
[GetSelectedObject \[Page 526\]](#)
[HidePropertyWindow \[Page 527\]](#)
[IsApplicationArea \[Page 528\]](#)
[IsBAPI \[Page 529\]](#)
[IsBusinessObject \[Page 530\]](#)
[IsFunction \[Page 531\]](#)
[IsMethod \[Page 532\]](#)
[IsRFC \[Page 533\]](#)
[IsSearch \[Page 534\]](#)
[Refresh \[Page 535\]](#)
[ShowPropertyWindow \[Page 536\]](#)
[StartPrint \[Page 537\]](#)
[Undo \[Page 538\]](#)

AddBAPIAppObject

Use this method to add a BAPI application object to the BAPI pane.

Syntax

AddBAPIAppObject(ApplicationHierarchy As Object)

Part

ApplicationHierarchy

Description

Required. Repository Service Application Hierarchy Object

AddBAPIObject

Use this method to add a BusinessObject to the BAPI pane.

Syntax

AddBAPIObject(ByVal BusinessObject As Object)

Part

BusinessObject

Description

Required. Repository Service Business Object

AddBAPISearchObject

AddBAPISearchObject

Use this method to populate the search pane with the searched for BusinessObject, if available.

Syntax

AddBAPISearchObject(ByVal BusinessObject As Object)

Part

BusinessObject

Description

Required. Repository Service Business Object

AddRFCFunctionGroups

Use this method to populate the RFC tab window with function groups.

Syntax

AddRFCFunctionGroups(ByVal FunctionGroup As Object)

Part

FunctionGroup

Description

Required. Repository Service Function Group object

AddRFCFunctions

AddRFCFunctions

Use this method to populate the RFC pane with function objects.

Syntax

AddRFCFunctions(ByVal Functions As Object)

Part

Functions

Description

Required. Repository Service Functions object (collection of Function object)

AddSearchRFCFunctionGroups

Use this function to populate the search pane with the searched for function groups, if available.

Syntax

AddSearchRFCFunctionGroups(ByVal FunctionGroup As Object)

Part

FunctionGroup

Description

Required. Repository Service Function Group object

AddSearchRFCFunctions

AddSearchRFCFunctions

Use this method to populate the search pane with the search for function, if available.

Syntax

AddSearchRFCFunctions(ByVal Functions As Object)

Part

Functions

Description

Required. Repository Service Functions object (collection of function objects)

CallFunction

Use this method to call the selected RFC function in the RFC tab window. It invokes a screen for entering import RFC function data.

Syntax

CallFunction()

ClearAll

ClearAll

Use this method to clear the BAPI, RFC, and search panes and the property window.

Syntax

ClearAll()

ClearBAPITab

Use this method to clear the BAPI tab window.

Syntax

ClearBAPITab()

ClearRFCTab

ClearRFCTab

Use this method to clear the RFC tab window.

Syntax

```
ClearRFCTab()
```

ClearSearchTab

Use this method to clear the Search pane.

Syntax

ClearSearchTab()

DeleteObject

DeleteObject

Use this method to delete an object. Methods and the root object cannot be deleted. Returns 0 if successful or non-zero.

Syntax

`DeleteObject (ByVal hobj As Long) as Long`

Part

`hobj`

Description

Required. Unique index of object to be deleted.

EnableBAPITab

Use this method to enable or disable the BAPI tab.

Syntax

EnableBAPITab(ByVal Enable As Boolean)

Part

Enable

Description

Required. True to enable, false to disable.

EnableRFCTab

Use this method to enable or disable the RFC tab.

Syntax

EnableRFCTab(ByVal Enable As Boolean)

Part

Enable

Description

Required. True to enable, false to disable.

EnableSearchTab

Use this method to enable or disable the Search tab.

Syntax

EnableSearchTab (ByVal Enable As Boolean)

Part

Enable

Description

Required. True to enable, false to disable.

GetSelectedObject

Use this method to return the IDespatch pointer for the selected object.

Syntax

```
GetSelectedObject (hobj As Long) As Object
```

Part

hobj

Description

Required. Long variable by ref. Function populates it with unique index. Later, this index may be used in the function DeleteObject to delete the object from view.

HidePropertyWindow

Use this method to hide the property window.

Syntax

`HidePropertyWindow()`

IsApplicationArea

IsApplicationArea

Use this method to return a boolean value indicating whether or not the selected object is an application area.

Syntax

```
IsApplicationArea()
```

IsBAPI

Use this method to return a boolean value indicating whether the active tab window pane contains a business application programming interface (BAPI).

Syntax

IsBAPI()

IsBusinessObject

IsBusinessObject

Use this method to return a boolean value indicating whether or not the selected object is a business object.

Syntax

```
IsBusinessObject()
```

IsFunction

Use this method to return a boolean value indicating whether or not the selected object is a function.

Syntax

IsFunction ()

IsMethod

IsMethod

Use this method to return a boolean value indicating whether or not the selected object is a method.

Syntax

IsMethod ()

IsRFC

Use this method to returns a boolean value indicating whether or not the active tab window pane contains a remote function call (RFC).

Syntax

IsRFC()

IsSearch

IsSearch

Use this method to return a boolean value indicating whether or not the active tab window is a search pane.

Syntax

IsSearch()

Refresh

Use this method to refresh the control window.

Syntax

Refresh()

ShowPropertyWindow

ShowPropertyWindow

Use this method to show the property details of the selected object, such as whether or not the object is a BAPI, RFC, parameter, etc. This function invokes a screen with details including documentation about the object.

Syntax

```
ShowPropertyWindow(ByVal hwnd as long)
```

Part

hwnd

Description

Required. Parent Window Handle

StartPrint

Use this method to export the metadata (pertaining to the selected object) to Excel. Excel must be installed to use this method. Returns “true” if successful.

Syntax

StartPrint(nPrintmode As Long)

Part

nPrintMode

Description

Default; always 1. Reserved for future use.

Undo

Undo

Use this method to return to the previous operation in the BAPI and RFC panes.

Syntax

`Undo()`

Example

Procedure

To run the following example, create a new Visual Basic project by following these steps:

1. Right-click on tool box and choose Components. The Components dialog box appears.
2. In the Control tab, search for SAPBrowser in the list.
3. Select the SAPBrowser check box and choose OK.
4. Drag (or double click) the SAPBrowser icon in the tool box to Form1.
5. Resize the control properly on Form1.
6. Create two command buttons and name them cmdBrowse and cmdProperty.
7. Double click anywhere on Form1 to see the Visual Basic code window.
8. Copy the code below and paste it into the code window.
9. Save and run the project. If all the required components are installed on your machine, you should see the R/3 logon screen when you run the project.

Code

```
Option Explicit

Private moRepObj As Object
Private moLogonObj As Object
Dim moConn As Object

Private Sub cmdBrowse_Click()
Dim oAppHiers As Object
Dim oAppHier As Object
Set moRepObj = CreateObject("SAP.RepositoryServicesonline.1")
If moRepObj Is Nothing Then
    MsgBox "Could not create Repository object", vbInformation
    Exit Sub
End If
Set moLogonObj = CreateObject("SAP.LogonControl.1")
If Not moLogonObj Is Nothing Then
    Set moConn = moLogonObj.NewConnection
    If Not moConn Is Nothing Then moConn.Logon
Else
    MsgBox "Could not create logon object", vbInformation
    Exit Sub
End If
```

Example

```
End If
moRepObj.Connection = moConn
SAPBrowse1.EnableBAPITab True
Set oAppHiers = moRepObj.ApplicationHierarchies()
If SAPBrowse1.Connect(moRepObj) Then
    For Each oAppHier In oAppHiers
        SAPBrowse1.AddBAPIAppObject oAppHier
    Next
End If
End Sub
Private Sub cmdProperty()
    SAPBrowse1.ShowPropertyWindow Me.Hwnd
End Sub
Private Sub Form_Unload(Cancel As Integer)
    moConn.Logoff
End Sub
```