






























SAP GUI Scripting API for the Windows and Java Platforms







Release 620


















Copyright	8
Icons	9
Typographic Conventions.....	9
 Scripting API for SAP GUI	10
Purpose	10
Integration.....	10
Features	10
Requirements	10
Operating System	10
SAP System.....	11
SAP GUI	11
 SAP GUI Runtime Hierarchy.....	11
 Runtime hierarchy overview	11
Top level administrative objects	11
Top level user interface objects.....	13
 SAP GUI Object Model	15
 Basic Interfaces	15
 GuiComponent.....	15
 GuiVComponent	17
 GuiVContainer	19
 GuiContainer.....	20
 Administrative Objects.....	21
 GuiApplication.....	21
Event Handlers	26
 GuiConnection	27
 GuiSession	28
Event Handlers	31

 Simple Visual Objects	36
 GuiBox	36
 GuiButton	36
 GuiRadioButton	36
 GuiCheckBox	37
 GuiLabel	37
 GuiTextField	38
 GuiPasswordField	39
 GuiCTextField	39
 GuiComboBox	40
 GuiOkCodeField	41
 GuiStatusBar	41
 Visual Container Objects	44
 GuiFrameWindow	44
 GuiMainWindow	46
 GuiModalWindow	48
 GuiUserArea	49
 GuiSimpleContainer	49
 GuiScrollContainer	50
 GuiTitlebar	50
 GuiToolbar	50

	GuiMenubar	51
	GuiMenu	51
	GuiContextMenu 	51
	GuiCustomControl	52
	GuiContainerShell.....	52
	GuiDockShell	52
	GuiShell	52
	GuiGOSShell	53
	GuiDialogShell	54
	GuiTabStrip.....	54
	GuiTab	55
	GuiTableControl.....	56
	GuiTableColumn.....	58
	GuiTableRow	58
	Controls	60
	GuiCtrlGridView	60
	GuiCtrlCalendar	68
	GuiCtrlWebViewer2D 	69
	GuiCtrlPicture	71
	GuiCtrlToolbar	73
	GuiCtrlTextEdit	75

	GuiCtrlOfficeIntegration 	76
	GuiCtrlTree	77
	GuiCtrlHTMLViewer	84
	GuiSplitterShell	85
	Graphics Controls	86
	GuiCtrlBarChart (Under Construction)	86
	GuiCtrlNetChart	87
	GuiCtrlChart	87
	GuiCtrlColorSelector	89
	Application Controls	90
	APOGrid (Under construction) 	90
	Utility Classes	95
	GuiSessionInfo	95
	GuiUtils	96
	GuiScrollbar	98
	Collections	99
	GuiComponentCollection	99
	GuiCollection	100
	Platform and Language Dependencies	102
	Accessing the Runtime Hierarchy	102
	SAP GUI for Windows	102

Attaching to a running SAPlogon process.....	102
Creating a SAP GUI instance using the scripting component.....	102
 SAP GUI for Java	102
JavaScript Engine	102
AppleScript Engine.....	103
 Event Handlers in JavaScript.....	103
 SAP GUI for Java using AppleScript.....	105
 Executing Scripts.....	105
 SAP GUI for Windows	105
 Appendix	106
 Technical Background - Connection Strings.....	106
Simple Connection Strings.....	106
SAP Routers.....	107
Message Servers and Logon Groups	107
Symbolic System Names	108
 DumpState Collection Format.....	108
 Examples.....	110
 Attaching to a SAPlogon instance	110
 Traversing the SAP GUI Runtime Hierarchy.....	110
SAP GUI for Windows using Visual Basic.....	110
SAP GUI for Java using Javascript	111
 Logging SAP GUI response times.....	112
 Recording user interaction.....	113
 Pre-setting values in the SAP GUI input history.....	116
 Q&A.....	118
Why is New Visual Design not available if I start SAP GUI for Windows using CreateObject?.....	118
Why do I receive control errors when running or recording a script?.....	119

Does SAP GUI for Windows behave differently when recording or playing back a script?	119
Where do I get the latest version of the documentation?	119
How do I report problems to SAP?	120
 SAP Notes on Scripting	121

Copyright

© Copyright 2003 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®, VBScript 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 IBM Corporation.

UNIX®, X/Open®, OSF/1®, and Motif® are registered trademarks of the Open Group. Citrix®, the Citrix logo, ICA®, Program Neighborhood®, MetaFrame®, WinFrame®, VideoFrame®, MultiWin® and other Citrix product names referenced herein are trademarks of Citrix Systems, Inc.

Apple, the Apple logo, AppleScript, AppleTalk, AppleWorks, Finder, LaserWriter, Mac, Macintosh, and PowerBook are trademarks of Apple Computer, Inc., registered in the United States and other countries.







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, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPHIRE, Management Cockpit, mySAP, mySAP.com, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. MarketSet and Enterprise Buyer are jointly owned trademarks of SAP Markets and Commerce One. All other product and service names mentioned are the trademarks of their respective owners.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax
	Not supported in SAP GUI for Java

Typographic Conventions

Type Style	Description
<i>Example text</i>	Words or characters that appear on the screen. These include field names, screen titles, push buttons as well as menu names, paths and options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, titles of graphics and tables
EXAMPLE TEXT	Names of elements in the system. These include report names, program names, transaction codes, table names, and individual key words of a programming language, when surrounded by body text, for example, SELECT and INCLUDE.
Example text	Screen output. This includes file and directory names and their paths, messages, names of variables and parameters, source code as well as names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Pointed brackets indicate that you replace these words and characters with appropriate entries.
EXAMPLE TEXT	Keys on the keyboard, for example, function keys (such as F2) or the ENTER key



Scripting API for SAP GUI

Purpose

Ever since the release of SAP system version 4.6C, there has been only very limited support for emulating user interaction with a SAP system. Existing technologies, such as ITOLE or Guilib, connect to the SAP system at the protocol level and have never been able to emulate the behavior of the compound controls introduced with 4.6C.

For this reason, applications relying on emulating user input worked only on the decreasing number of transactions using only standard dynpro elements.

Examples of affected applications are:

- Automatic testing of SAP functionality
- Customized front end applications replacing the SAP GUI
- Tools to customize applications on the SAP GUI level → GuiXT
- E-Learning applications that guide a user through SAP transactions

Integration

Many of the available SAP GUI controls were designed exclusively with user interaction in mind. As their business functionality is closely coupled with the user interface they cannot be instantiated outside the SAP GUI in a batch-like fashion.

We therefore decided not to add the business functionality of the SAP GUI controls to a low-level integration component such as Guilib. Instead the controls run within the SAP GUI, which itself exposes a new interface allowing the automation of tasks.

Features

We developed an object model representing the SAP GUI at runtime as a hierarchy of objects. Most of these expose an interface to an element of the user interface. These interfaces can be used to perform all the actions a user could do with the given element. In addition we offer outgoing interfaces through which an external application can receive notifications about events occurring within the SAP GUI.

Available uses for the scripting component include

- Listening to the actions a user performs in the SAP GUI and record them as a script
- Running a script that emulates user interaction
- Logging the SAP system information, such as response time

Requirements

Operating System

The Scripting interface is available on

- Windows 98, Windows 2000, Windows XP
- Linux, AIX, Solaris, HP-UX, Tru64
- OS/2

- Mac OS 9, Mac OS X

SAP System

Scripting support is available for the 3.1I, 4.0B, 4.5B, 4.6B, 4.6C, 4.6D, 6.10, and 6.20 releases and all subsequent releases.

For the releases 3.1I to 6.10 ABAP support packages and the SAP kernel patches are available to add the support, while they are already part of 6.20 and later releases. Note 480149 lists the required patch levels.

SAP GUI

The scripting interface can be installed with the SAP GUI release 6.20 and later releases.



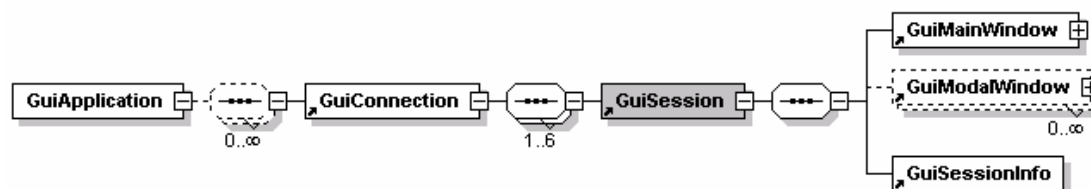
SAP GUI Runtime Hierarchy



Runtime hierarchy overview

Top level administrative objects

All objects defined in the scripting component's object model are available at runtime as members of a hierarchical tree with the root object being *GuiApplication*.



GuiApplication represents the process in which the SAP system activity takes place. Because of this there should always be only one *GuiApplication* object within a process. The children of *GuiApplication* are all the connections of class *GuiConnection* to the SAP systems available for scripting¹.

Connections are opened manually from the SAPLogon dialog or using the *openConnection* and *openConnectionByConnectionString* methods of *GuiApplication* from a script. As soon as a connection has been established a first session is created as a child of the connection. Up to 5 additional sessions can be created. Again, this can be done manually using the 'Create Session' menu item or toolbar button, or from a script using the *CreateSession* method of *GuiSession*.

While *GuiApplication* represents the overall SAP GUI application, a *GuiSession* represents a specific task being performed. For any given session there is exactly one transaction currently executed, and most tasks performed in SAP GUI can be performed within the context of one *GuiSession*.

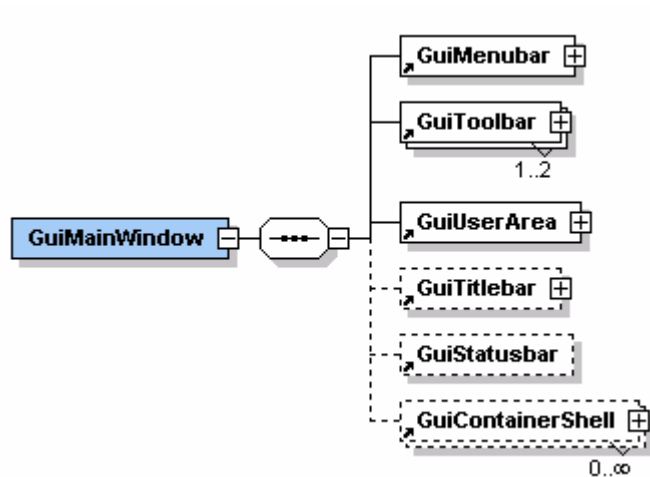
¹ See the chapter entitled 'Security considerations' for connections that are not scripting-enabled.

These considerations determine the event model of the scripting component. A user's interactions are best recorded or logged in the context of one session, therefore the *GuiSession* exposes an outgoing interface that allows an application to listen to the user interaction.

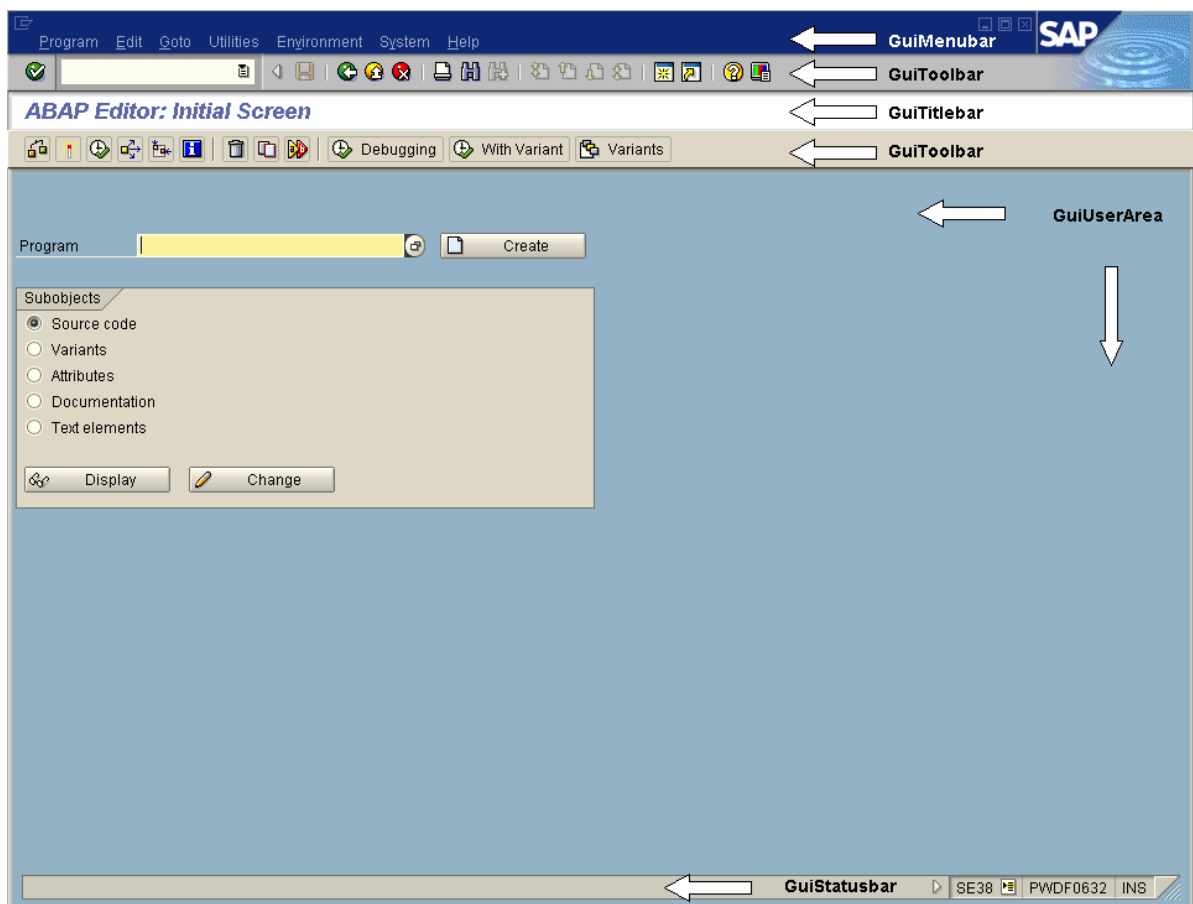
On the other hand, the *GuiApplication* exposes an outgoing interface that raises administrative events, for example when a session is created or destroyed.


Top level user interface objects

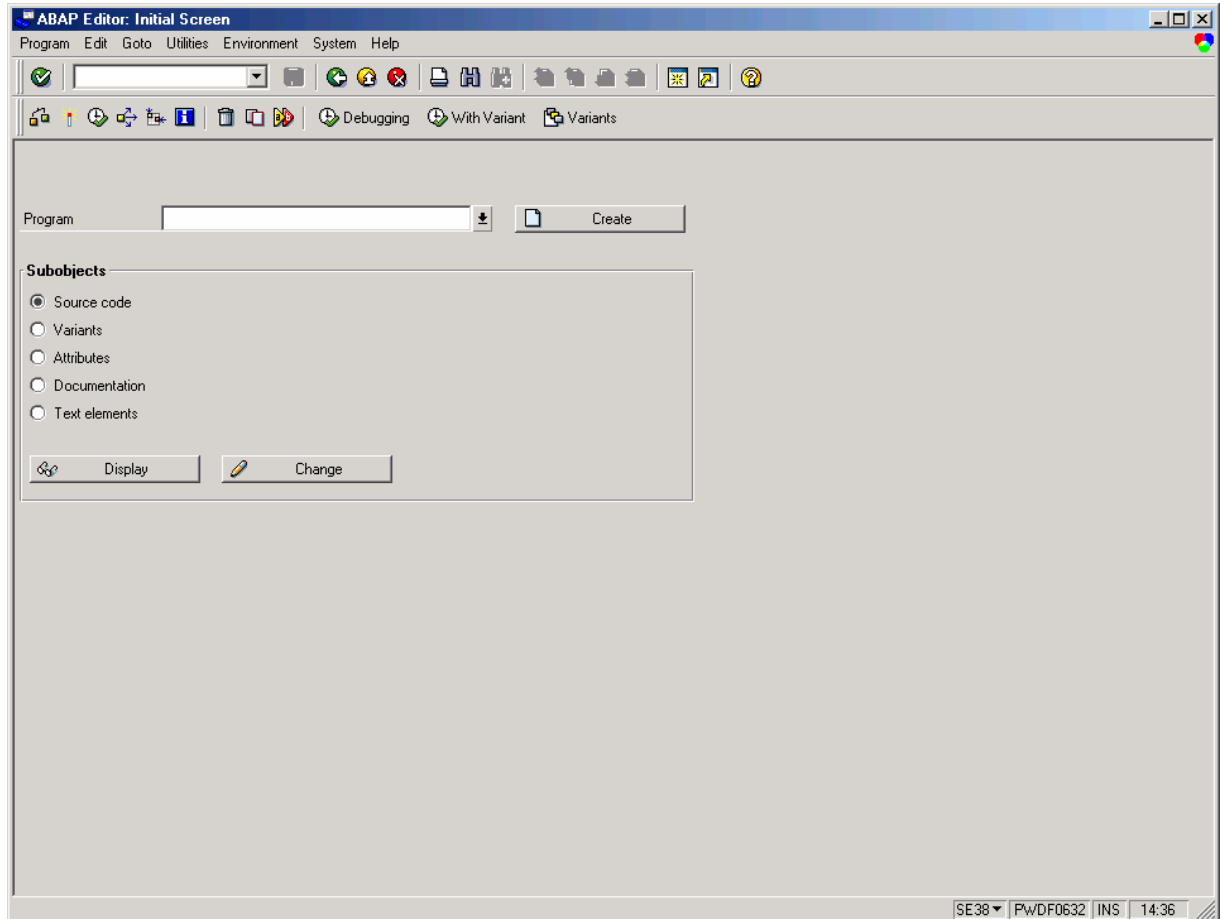
A session usually has a *GuiMainWindow* as its child. This window is the starting point for user interaction with a session.



The children of the *GuiMainWindow* are easily identified straightforward.



 Please note that the availability of some of these objects depends on the design mode used. The titlebar, for example, is available only in New Visual Design, not in classic mode, as can be seen in the following screenshot.





SAP GUI Object Model

There are some basic interfaces that most of the objects support, so that anybody accessing the objects can safely assume that certain properties or methods are available.



Basic Interfaces



GuiComponent

With the exception of some helper objects, all objects support this interface. It is designed to allow generic programming, meaning you can work with objects without knowing their exact type. *GuiComponent* is an abstract class.

Property id As String (Read only)

An object's *id* is created as a unique textual identifier for the object. This is achieved through a URL-like formatting, starting at the *GuiApplication* object and drilling down to the respective object.

For example, the *id* of the text field in transaction se38 is

```
/app/con[0]/ses[0]/wnd[0]/usr/ctxtRS38M-PROGRAMM.
```

It is created through first taking the type specific prefix for the given component, which in this case is 'ctxt'. The name of the field is then appended, if one exists. For dynpro fields the name is the field name defined in screen painter, in this case 'RS38M-PROGRAM'.

In many cases, especially if there is no name available for an object, an index has to be appended for uniqueness.



Example

Objects of type *GuiShell* do not have a name. So whenever there are several objects of this type on the same level in the hierarchy, a one-dimensional index is appended for uniqueness. On the other hand, list and step-loop screens are set up in a two-dimensional grid, so the index string added to the *id* will have two components as well.

Finally, the *id* of the parent object is added as a prefix, separated by the '/' character.

Using these fully qualified *ids* has one significant drawback: the index of both the connection and session become part of the *id*. If the *id* given above is used to search for the text field the search will only be successful if the session to which the text field belongs is again the first session of the first connection.

This problem can be circumvented using relative *ids*. A relative *id* does not start with '/'. Instead it begins with the type prefix of the child of the object on which the search is started. Therefore it is usually preferable to truncate the first part of the *id* up to the session index and work with a relative *id*, such as

```
wnd[0]/usr/ctxtRS38M-PROGRAMM.
```

This relative id is still unique with respect to a given session, but it is independent of the number of sessions or connections currently open.

Property type As String (Read only)

The *type* information of *GuiComponent* can be used to determine which properties and methods an object supports. The value of the *type* string is the name of the type taken from this documentation².

Property typeAsNumber As Long (Read only)

While the *type* property is a string value, the *typeAsNumber* property is a long value that can alternatively be used to identify an object's type³. It was added for better performance in methods such as *FindByIdEx*.

GuiUnknown:	-1
GuiComponent:	0
GuiVComponent:	1
GuiVContainer:	2
GuiApplication:	10
GuiConnection:	11
GuiSession:	12
GuiFrameWindow:	20
GuiMainWindow:	21
GuiModalWindow:	22
GuiLabel:	30
GuiTextField:	31
GuiCTextField:	32
GuiPasswordField:	33
GuiComboBox:	34
GuiOkCodeField:	35
GuiButton:	40
GuiRadioButton:	41
GuiCheckBox:	42
GuiCustomControl:	50
GuiContainerShell:	51
GuiBox:	62
GuiContainer:	70
GuiSimpleContainer:	71
GuiScrollContainer:	72
GuiUserArea:	74
GuiTableControl:	80
GuiTableColumn:	81
GuiTableRow:	82
GuiTabStrip:	90
GuiTab:	91
GuiScrollbar:	100
GuiToolbar:	101
GuiTitlebar:	102
GuiStatusbar:	103
GuiMenu:	110
GuiMenubar:	111
GuiCollection:	120
GuiSessionInfo:	121

² In SAP GUI for Windows you can find all the possible values in the *GuiComponentType* enumeration in the type library in *sapfewse.ocx*.

³ As for the *type* property, these values can be taken from the type library in *sapfewse.ocx*.

GuiShell:	122
GuiGOSShell:	123
GuiSplitterShell:	124
GuiDialogShell:	125
GuiDockShell:	126
GuiContextMenu	127

Property containerType As Boolean (Read only)

This property is *TRUE*, if the object is a container and therefore has the *children* property.

Property name As String (Read only)

The *name* property is especially useful when working with simple scripts that only access dynpro fields. In that case a field can be found using its name and type information, which is easier to read than a possibly very long id. However, there is no guarantee that there are no two objects with the same name and type in a given dynpro.



Property parent As GuiComponent (Read only)

The *parent* of an object is one level higher in the runtime hierarchy. An object is always in the *children* collection of its parent.



GuiVComponent

The *GuiVComponent* interface is exposed by all visual objects, such as windows, buttons or text fields. Like *GuiComponent*, it is an abstract interface. Any object supporting the *GuiVComponent* interface also exposes the *GuiComponent* interface.

Property text as String

The value of this property very much depends on the type of the object on which it is called. This is obvious for text fields or menu items. On the other hand this property is empty for toolbar buttons and is the class id for shells.

You can read the *text* property of a label, but you can't change it, whereas you can only set the *text* property of a password field, but not read it.

Property tooltip As String (Read only)

The *tooltip* contains a text which is designed to help a user understand the meaning of a given text field or button.

Property changeable As Boolean (Read only)

An object is changeable if it is neither disabled nor read-only.

Property modified As Boolean (Read only)

An object is modified if its state has been changed by the user and this change has not yet been sent to the SAP system.

Property `iconName` as String (Read only)

If the object has been assigned an icon, then this property is the name of the icon, otherwise it is an empty string.

Function `dumpState (innerObject As String) As GuiCollection`

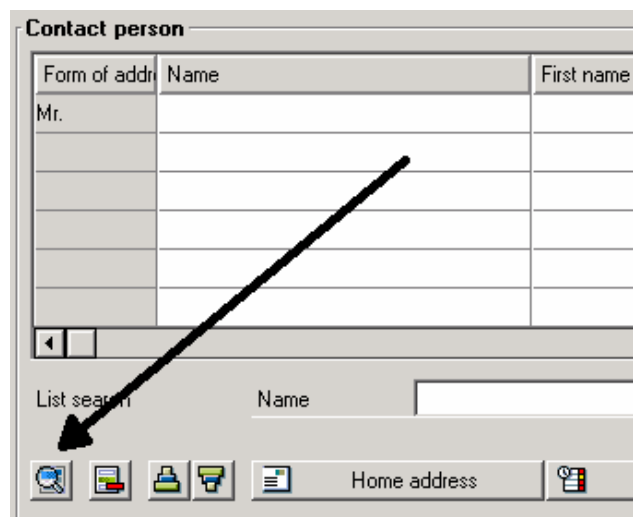
This function dumps the state of the object. The parameter *innerObject* may be used to specify for which internal object the data should be dumped. Only the most complex components, such as the `GuiCtrlGridView`, support this parameter. All other components always dump their full state. All components that support this parameter have in common that they return general information about the control's state if the parameter "innerObject" contains an empty string.

The format of the returned collection is described in the chapter "DumpState Collection Format" of the appendix. The available values for the *innerObject* parameter are specified as part of the class description for those components that support it.

Function `setFocus`

This function can be used to set the focus onto an object. If a user interacts with SAP GUI, it moves the focus whenever the interaction is with a new object. Interacting with an object through the scripting component does not change the focus. There are some cases in which the SAP application explicitly checks for the focus and behaves differently depending on the focussed object.

Pressing this button to display the details of an entry in the table control will only succeed if the focus has already been set on the respective entry. The button is indicated in the following screenshot.



Function `Visualize (on As Boolean, innerObject As String = "") As Boolean`

Calling this method of a component will display a red frame around the specified component if the parameter *on* is *true*. The frame will be removed if *on* is *false*. Some components such as `GuiCtrlGridView` support displaying the frame around inner

objects, such as cells. The format of the *innerObject* string is the same as for the *dumpState* method.



GuiVContainer

An object exposes the *GuiVContainer* interface if it is both visible and can have children. It will then also expose *GuiComponent* and *GuiVComponent*. Examples of this interface are windows and subscreens, toolbars or controls having children, such as the splitter control.

Property children As GuiComponentCollection⁴ (Read only)

This collection contains all the direct children of an object.

Parameter Name	Type	Reference type	Default value	Option	Pass	Short text	Long
ENVIRONMENT	TYPE	C		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Variable name for environment query ('EN')	
FILENAME	TYPE	C		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	File name for existence ('FE') and length ('FL')	
QUERY	TYPE	C		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Query command : 'CD' 'FE' 'FL' 'EN' 'WI' 'WS' ...	<input checked="" type="checkbox"/>
WINID	TYPE	C		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Window ID for window query ('WI')	
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		
				<input type="checkbox"/>	<input type="checkbox"/>		

In this example the user area has 4 direct children: one label, two text fields and a tab strip. The other visible objects are descendants of the tab strip.

Function findById (id As String, raise As Boolean = true) As GuiComponent

Any object exposing the *GuiVContainer* interface can search through its descendants for a given *id*. If the parameter is a fully qualified id, the function will first check if the container object's id is a prefix of the parameter id. If that is the case then this prefix is truncated.

So, unless there are several sessions or connections in use, the following calls are equivalent:

```
Set Comp =
UserArea.findById("/app/con[0]/ses[0]/wnd[0]/usr/txtHEADER-
FBFOOTLINE")
```

⁴ See the chapter entitled 'Collection interfaces' for a description of *GuiComponentCollection*

```
Set Comp = UserArea.findById("txtHEADER-FBFOOTLINE")
```

The result of a call to this function is a *GuiComponent*, therefore its *type* and *id* can be accessed.

If no descendant with the given *id* can be found and the parameter *raise* is set to *True* then the function raises an exception. If *raise* is set to *False* then the function returns a *Null/Nothing* object.

Function **findByName (name As String, type As String) As GuiComponent**

Unlike *findById*, this function does not guarantee a unique result. It will simply return the first descendant matching both the *name* and *type* parameters.

It is possible to replace the above call of *findById* using *findByName* like this:

```
Set Comp = UserArea.findByName("HEADER-FBFOOTLINE",  
"GuiTextField")
```

This is a more natural description of the object than the complex id, but it only makes sense on dynpro objects as most other objects do not have a meaningful name.

If no descendant with matching *name* and *type* can be found, the function raises an exception.

Function **findByNameEx (name As String, type As long) As GuiComponent**

This function has been introduced for performance reasons. The parameter *type* is not a string as in *findByName* but rather a number taken from *GuiComponent.typeAsNumber*.

Function **findAllByName (name As String, type As String) As GuiComponentCollection**

Function **findAllByNameEx (name As String, type As long) As GuiComponentCollection**

The methods *findByName* and *findByNameEx* return only the first object with matching name and type. There may however be several matching objects, which will be returned as members of a collection when *findAllByName* or *findAllByNameEx* are used.



GuiContainer

This interface resembles *GuiVContainer*. The only difference is that it is not intended for visual objects but rather administrative objects such as connections or sessions. Objects exposing this interface will therefore support *GuiComponent* but not *GuiVComponent*.

Property **children As GuiComponentCollection⁵ (Read only)**

This collection contains all direct children of the object.

Function **findById (id As String, raise As Boolean = true) As GuiComponent**

⁵ See the chapter 'Collection interfaces' for a description of *GuiComponentCollection*

Any object exposing the *GuiContainer* interface can search through its descendants for a given *id*. If the parameter is a fully qualified *id*, the function will first check if the container object's id is a prefix of the parameter id. If that is the case, this prefix is truncated.

If no descendant with the given *id* can be found the function raises an exception unless the optional parameter *raise* is set to *False*.



Administrative Objects

The objects in this section have no visual representation. They provide access to the low level administrative data of SAP GUI.



GuiApplication

The *GuiApplication* represents the process in which all SAP GUI activity takes place. If the scripting component is accessed by attaching to a SAPlogon process, then *GuiApplication* will represent SAPlogon. *GuiApplication* is a creatable class. However, there must be only one component of this type in any process.

Supported base interfaces: *GuiComponent*, *GuiContainer*

Type prefix: app

Name: app

Property connections As GuiComponentCollection⁶ (Read only)

This property is another name for the *Children* property. It has been added for better readability as all the children of *GuiApplication* are connections.

Property majorVersion As Long (Read only)

Version of the SAP GUI release, for example '6.20'.



Property revision As Long (Read only)

Revision of the SAP GUI release. In SAP GUI for Windows this is the compilation number.



Property patchLevel As Long (Read only)

Patchlevel of SAP GUI.

Property minorVersion As Long (Read only)

⁶ See the chapter entitled 'Collection interfaces' for a description of *GuiComponentCollection*

Build number of the scripting component.

Property newVisualDesign As Boolean (Read only)

- **False: Classic mode**

Display Profile Parameter Attributes

Param. Name: sapgui/user_scripting

Short description(Engl): Enable or disable user scripting on the frontend.

Appl. area: GUI Frontend

ParameterTyp: Logical value

Changes allowed: Change permitted

Valid for oper. system: All operating systems

DynamicallySwitchable:

Same on all servers:

Check module:

Dflt value: FALSE

ProfileVal: FALSE

Current value: FALSE

Created by: COHRS 10.01.2002 15:09:36

Changed by: BISCHOFJ 22.01.2002 16:40:39

- **True: New Visual Design**

Display Profile Parameter Attributes

Param. Name: sapgui/user_scripting

Short description(Engl): Enable or disable user scripting on the frontend.

Appl. area: GUI Frontend

ParameterTyp: Logical value

Changes allowed: Change permitted

Valid for oper. system: All operating systems

DynamicallySwitchable:

Same on all servers:

Check module:

Dflt value: FALSE

ProfileVal: FALSE

Current value: FALSE

Created by: COHRS 10.01.2002 15:09:36

Changed by: BISCHOFJ 22.01.2002 16:40:39

Property utils As GuiUtils⁷ (Read only)

This property returns a global GuiUtils object.

⁷ See the chapter entitled 'GuiUtils' for a description of GuiUtils.



Property historyEnabled As Boolean

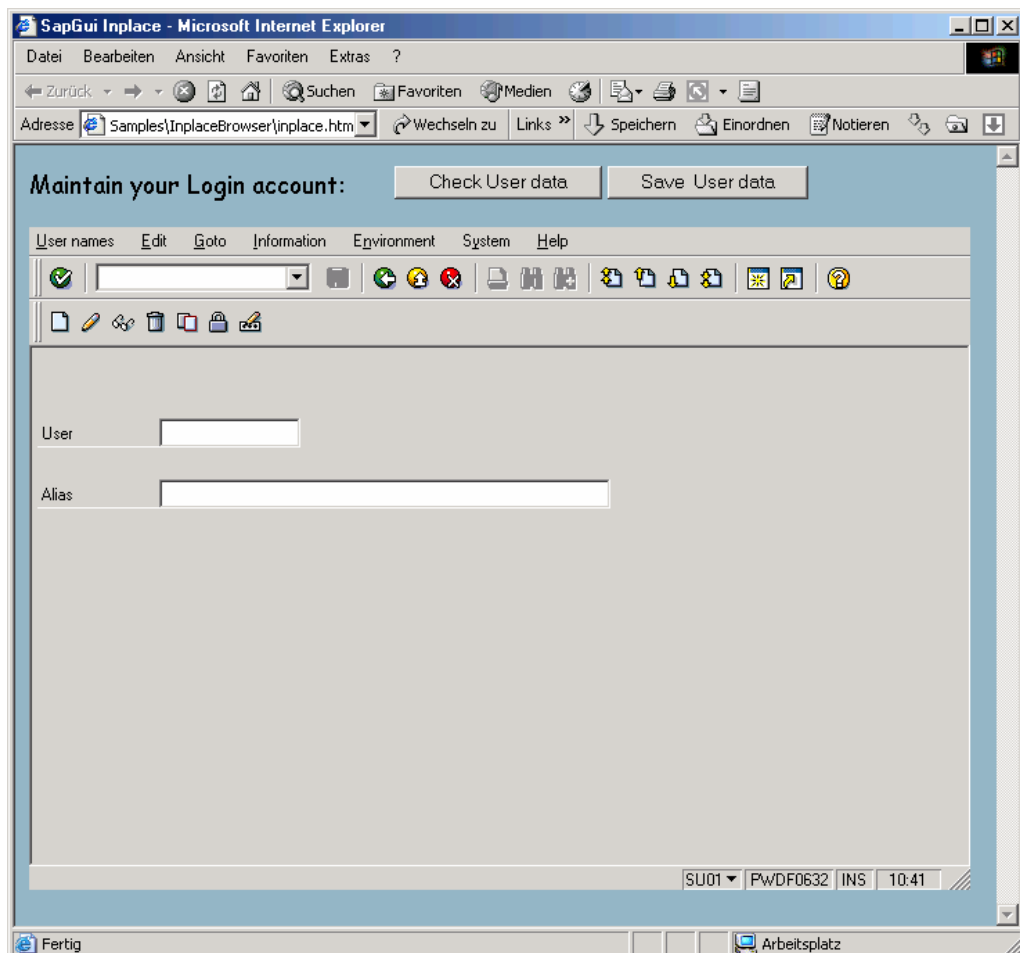
The local history function can be enabled or disabled using this property. Disabling it will significantly improve the performance of SAP GUI, which may be crucial during load tests, for example.

Function openConnection (descriptionString As String, sync As Boolean = False, raise As Boolean = True) As GuiConnection

The parameter *connectString* should contain one of the descriptions displayed in SAPLogon, for example "XYZ [PUBLIC]".

If you want to create a new SAP GUI instance and place it within your application you may add the suffix "/INPLACE".

In the following example the scripting component has been placed on an HTML page and the connection was opened in the *pageLoad* event handler using the "/INPLACE" switch.



This function will raise the exception `E_ACCESSDENIED` if the scripting support has been disabled by the administrator or the user.

When opening connections manually SAP GUI executes the request asynchronously, so that the SAPLogon dialog remains responsive after requesting a new connection.

This behaviour is also the default for SAP GUI Scripting. In the Scripting context it means that the call to *openConnection* may return before the new connection has

been opened. A side effect of this is that when opening a connection fails SAP GUI displays an error popup that can not be handled from the script.

This problem can be solved by setting the *sync* parameter to *True*. Then the call to *openConnection* will not return until a connection has been established, or an error has been detected. If *sync* is set to *True* and an error occurs an exception is raised, unless the parameter *raise* is set to *False*.

Function *openConnectionByConnectionString* (connectionData As String, sync as Boolean, raise As Boolean) As GuiConnection

The parameter *connectionData* is the connection string for the SAP server, for example *"/R/ALR/G/SPACE"*. A detailed discussion of connection strings is available in the appendix of this document. See the description of the *openConnection* method for a discussion of the *sync* and *raise* parameters.

Function *createGuiCollection* As GuiCollection⁸

Some functions accept collections as parameters. This function creates a collection object that is independent of the scripting language used, such as VBScript or JavaScript.

Function *addHistoryEntry* (fieldname As String, value As String) As Boolean



SAP GUI for Windows has an input history functionality, which displays for text fields the entries made in the past as a suggestion.

With this function, an entry can be added to the history database so that it will be available the next time the end user accesses the text field with the given field name. For a given text field, the field name can be determined by positioning the cursor on the field and pressing F1. A new window is then opened for the Performance Assistant, which has a Technical Information button. Using this will display the dialog below. The field name is displayed at the bottom of the dialog as 'Screen field'.

The functions returns *True* if the entry was added successfully.

⁸ See the chapter entitled 'Collection interfaces' for a description of *GuiCollection*

Technical Information	
Screen data	
Program name	SAPMS38L
Screen number	1009
GUI data	
Program name	SAPMS38L
Status	1009
Field data	
Struct.	RS38L
Field name	NAME
Data element	RS38L_FNAM
DE supplement	0
Parameter ID	LIB
Field description for batch input	
Screen field	RS38L - NAME
<input checked="" type="checkbox"/> Navigate <input type="checkbox"/>	

Function dropHistory As Boolean

Calling this function will delete all entries from the input history. The function returns *True* if the history data have been deleted successfully.

Function registerROT As Boolean

Accessing the SAPGUI entry in the Running Object Table from a C++ application may fail unless the interface is registered with a strong reference. This is not required when using Visual Basic or scripting languages.

The reference must be released using *revokeRot* before shutting down the Scripting component. Failing to do so will lead to undefined behaviour.

Most applications do not need to use this method, and shouldn't call it.

Function revokeROT

This method must be called before shutting down the Scripting component if *registerROT* was used.

Event Handlers⁹

Event createSession (newSession As GuiSession)

This event is raised whenever a new session is created, irrespective of whether of the session being created manually, from ABAP or by a script. The event is only raised for a session if the scripting support has been enabled for the corresponding backend. The following script attaches itself to the SAPlogon process and displays a pop-up whenever a new session is created.

```
Dim objSapGui
Set objSapGui = GetObject("SAPGUI")

Dim objScriptingEngine
Set objScriptingEngine = objSapGui.GetScriptingEngine
WScript.ConnectObject objScriptingEngine, "Engine_"

Dim Waiting
Waiting = 1

Do While (Waiting = 1)
    WScript.Sleep(100)
Loop

Set objScriptingEngine = Nothing
Set objSapGui = Nothing

Sub Engine_CreateSession(ByVal Session)
    Dim result
    result = MsgBox("Session created", vbOKCancel)
    If result = vbCancel then
        Waiting = 0
    End If
End Sub
```

Event destroySession (session As GuiSession)

This event is raised before a session is destroyed¹⁰. This can be done either by closing the main window manually, or by calling the *closeSession* method of *GuiConnection*.

You can handle this event from VBScript by adding the following procedure to the sample script on previous page:

```
Sub Engine_DestroySession(ByVal Session)
    Dim result
    result = MsgBox("Session destroyed", vbOKCancel)
    If result = vbCancel then
        Waiting = 0
    End If
End Sub
```

⁹ This description is valid for Visual Basic Script. See the chapter "Event Handlers in JavaScript" for a discussion how to use events in JavaScript in SAP GUI for Java and JScript in SAP GUI for Windows.

¹⁰ This event is **not** raised when running the SAP Workplace, because it might cause dead locks. Please use the *destroy* event of *GuiSession* instead

Event error (eventId As Long, desc1 As String, desc2 As String, desc3 As String, desc4 As String)

An *error* event is currently only raised, if the wrapper library required to access a SAP GUI ActiveX control from a script is not available. This event is also available on the *GuiSession* in which the error occurred.

**GuiConnection**

A *GuiConnection* represents the connection between SAP GUI and an application server. Connections can be opened from *SAPLogon* or from *GuiApplication*'s *openConnection* and *openConnectionByConnectionString* methods.

It is possible to connect to an application server from ABAP using the following line:

```
CALL FUNCTION func DESTINATION dest.
```

However, this connection is implemented as a re-direction between the two application servers involved. There will therefore be no new *GuiConnection* object available and the existing object will not reflect the server switch.

Supported base interfaces: *GuiComponent*, *GuiContainer*

Type prefix: con

Name: con[n]

Property sessions As GuiComponentCollection¹¹ (Read only)

This property is another name for the *children* property. It was added for better readability as all the children of *GuiConnection* are sessions. Accessing either the *children* property or the *sessions* property can cause the exception *Gui_Err_Scripting_Disabled_Srv* (624) to be raised if the respective application server has not enabled the scripting support.

Property disabledByServer As Boolean (Read only)

This property is set to *True* if the scripting support has not been enabled for the application server¹².

**Property description As String (Read only)**

This description is only available if the connection was started either from *SAPLogon* or using *GuiApplication.OpenConnection*. In both cases the description can then be used when calling the *OpenConnection* method to play back a script on the same system.

¹¹ See the chapter entitled 'Collection interfaces' for a description of *GuiComponentCollection*

¹² See the chapter entitled 'Security considerations' for connections that are not scripting-enabled

Property connectionString As String (Read only)

This property contains the connection string defining the backend connection. It is more difficult to read, but it doesn't rely on the SAPLogon entries.

Function closeSession (sessionId As String)

A session can be closed by calling this method of the connection. Closing the last session of a connection will close the connection, too.

**GuiSession**

The *GuiSession* provides the context in which a user performs a certain task such as working with a transaction. It is therefore the access point for applications which record a user's actions regarding a specific task or play back those actions.

GuiSession is self-contained in that ids within the context of a session remain valid independently of other connections or sessions being open at the same time. Usually an external application will first determine with which session to interact. Once that is clear, the application will work more or less exclusively on that session.

Traversing the object hierarchy from the *GuiApplication* to the user interface elements, it is the session among whose children the highest level visible objects can be found. In contrast to objects like buttons or text fields, the session remains valid until the corresponding main window has been closed, whereas buttons, for example, are destroyed during each server communication¹³.

Supported base interfaces: *GuiComponent*, *GuiContainer*

Type prefix: ses

Name: ses[n]

Property activeWindow As GuiFrameWindow (Read only)

All windows can be found in the *children* collection of *GuiSession*. However, most of the time an application will access the currently activated window of the session, as that is the window with which a user will probably interact. This property is intended as a shortcut to this window.

Property info As GuiSessionInfo (Read only)

GuiSessionInfo is described in the 'Utility classes' chapter. It contains technical information about the current connection, the login data, the running SAP application and more.

¹³ See the chapter 'Object lifetime considerations' for a discussion of the different lifetimes of objects in scripts



Property record as Boolean

Setting this property to *True* enables the recording mode of the session. In this mode changes to elements of the user interface are recorded within SAP GUI and sent to a recording application using the *Change* event described later. This property is not supported by SAP GUI for Java.

Some elements of the user interface may behave differently in record mode than during playback or manual interaction.

- The F4 help dialog is always displayed as a modal window.
- Drag & Drop is disabled.



Property recordFile As String

A simple way to record a script it to set the *recordFile* property to a valid filename and then enable the *record* property. A Visual Basic Script file of the given name will be created in the SapWorkDir on the client PC.

This property only accepts simple filenames without path information.



Property testToolMode As Long

During internal tests some aspects of the user interface proved to be difficult to handle with test tools using the scripting component to automate SAP GUI. For this reason a special mode has been added in which the following changes are administered.

- While success (S), warning (W) and error (E) messages are always displayed in the statusbar, information (I) and abort (A) messages are displayed as pop-up windows unless *testToolMode* is set.
- The update mode of the application server is changed to immediate mode for the connection.
- System messages are ignored so that they do not interrupt the recording or playback of scripts.

The test tool mode requires one of the following versions of the SAP kernel:

- 6.20 Patch level 29 and all following kernel versions
- 4.6D Patch level 1208, see note 511310.

Currently only the following values are allowed for this property:

- 0: Disable *testToolMode*
- 1: Enable *testToolMode*



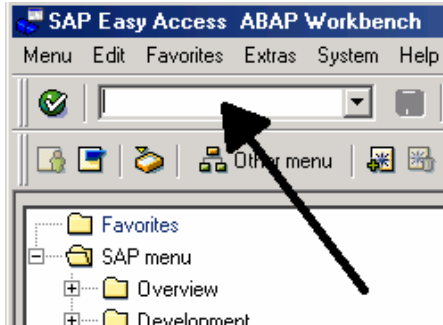
Property busy As Boolean

While SAP GUI is waiting for data from the server, any Scripting call will not return, which blocks the executing thread. This may not be acceptable for advanced applications.

A way to prevent this is to check the *busy* property of the session. If this property is *True*, then a subsequent Scripting call would have to wait for the server communication to be finished.

Function `sendCommand` (command as String)

Using this function you can execute any command string, which could otherwise be entered in the command field combo box indicated below.



Function `createSession`

This function opens a new session, which is then visualized by a new main window. This resembles the `/o` command that can be executed from the command field.

Function `startTransaction` (transaction As String)

Calling this function with parameter `"xyz"` has the same effect as `SendCommand("/nxyz")`.

Function `endTransaction`

Calling this function has the same effect as `SendCommand("/n")`.

Function `getVKeyDescription` (vKey As Long) As String

When a script is recorded, it will often contain `sendVKey(n)` calls, where *n* is a number. The method `getVKeyDescription` translates these numbers into a readable text. For example the number 0 is translated into the text "Enter".

Function `findByPosition` (x As Long, y As Long, raise As Boolean = True)

As GuiCollection

This method can be used to do a hittest on a SAP GUI session. The parameters *x* and *y* should be given in screen coordinates. If no component is found an exception is raised unless *raise* is set to *False*. In that case a *Null/Nothing* object is returned.

If a component is found, the collection contains 2 strings: The *Id* of the component and the description string for the inner object, which is only set for some complex components such as *GuiCtrlGridView*.

Event Handlers¹⁴

Event `startRequest` (session As `GuiSession`)

The *startRequest* event is raised before the session is locked during server communication. At this point user input can be checked before it is sent to the server. It is not possible to prevent server communication from this event.

Event `endRequest` (session As `GuiSession`)

endRequest is called immediately after the session is unlocked after server communication.

Event `error` (session As `GuiSession`, `eventId` As Long,

`desc1` As String, `desc2` As String, `desc3` As String, `desc4` As String)

An *error* event is currently only raised, if the wrapper library required to access a SAP GUI ActiveX control from a script is not available. *error* events from all sessions are also available at the *GuiApplication*.

Event `change` (session As `GuiSession`, `component` As `GuiComponent`, `commandArray` As Object)

In record mode, the session collects changes to elements of the user interface and sends these changes to a listening application whenever server communication is about to start or if the record mode is turned off. The *change* events are raised immediately before the *startRequest* event. There is at least one event for every modified element in the recorded session.



Only changes made at the SAP GUI level are recorded. Transactions may preset some of the entry fields with values from parameters stored in the SAP system. If these data are not changed in SAP GUI, they will not be recorded. This may cause problems during playback of scripts, if the entry fields are preset with different values.

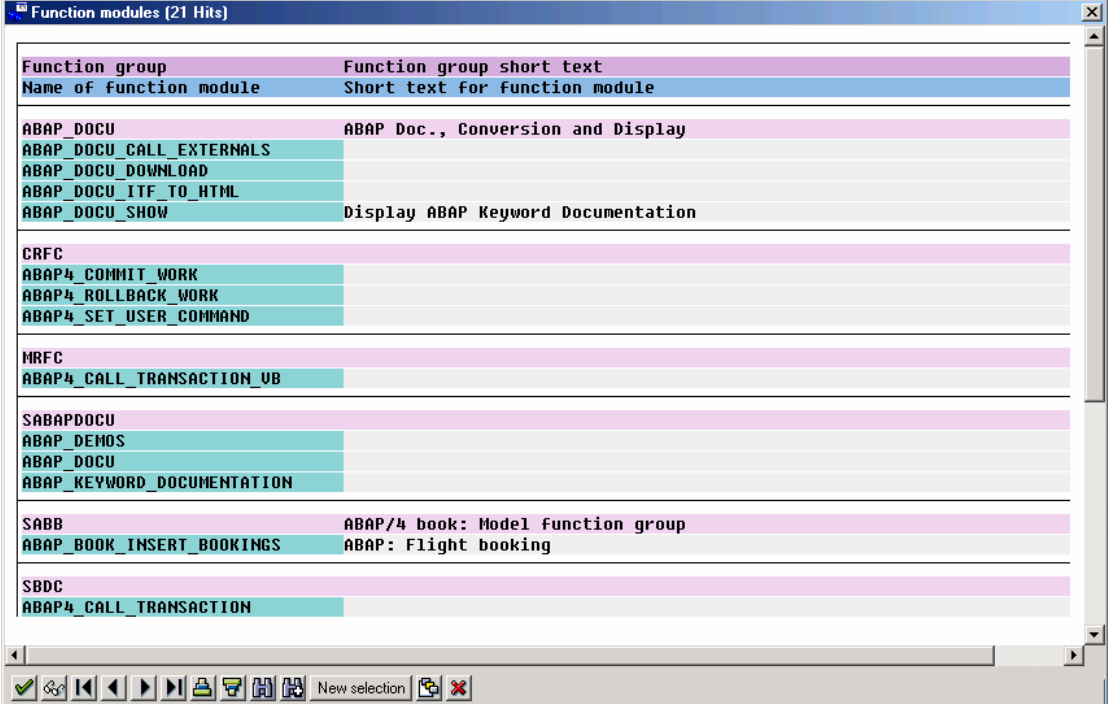
If any of the following techniques is used in a transaction, the user should manually modify all the entries he wants to see recorded:

- Usage of SAP parameters
- Variants
- Hold Data, from the menu System → User Profile

Playback of the changes will only work, if the order of the calls is the same as during recording.

¹⁴ This description is valid for Visual Basic Script. See the chapter “Event Handlers in JavaScript” for a discussion how to use events in JavaScript in SAP GUI for Java and JScript in SAP GUI for Windows.

Each event represents one line of script code. The *Component* parameter specifies the object on which to invoke a method or property. Therefore the first thing to record is *Component.id* for later use with *findById*. The recorder may however also decide to record other properties of *Component*. If, for example, a line in a table control or list is selected, it may be prudent not to record the position of the line, but rather the values in it. That way, a script can be generated that is more robust with respect to changes in the number, and therefore in the position, of lines.



Function group	Function group short text
Name of function module	Short text for function module
ABAP_DOCU	ABAP Doc., Conversion and Display
ABAP_DOCU_CALL_EXTERNALS	
ABAP_DOCU_DOWNLOAD	
ABAP_DOCU_ITF_TO_HTML	
ABAP_DOCU_SHOW	Display ABAP Keyword Documentation
CRFC	
ABAP4_COMMIT_WORK	
ABAP4_ROLLBACK_WORK	
ABAP4_SET_USER_COMMAND	
MRFC	
ABAP4_CALL_TRANSACTION_UB	
SABAPDOCU	
ABAP_DEMOS	
ABAP_DOCU	
ABAP_KEYWORD_DOCUMENTATION	
SABB	ABAP/4 book: Model function group
ABAP_BOOK_INSERT_BOOKINGS	ABAP: Flight booking
SBDC	
ABAP4_CALL_TRANSACTION	

If new function modules have been added, selecting a line from the list might return the wrong function module.

The *CommandArray* consists of method or property names and parameters. There will usually be only one line in the array.

Type	Method/Property name	Parameters
„SP“	„Text“	„Hello World“

This sets the parameter *Text* to value “Hello World”.

Type	Method/Property name	Parameters
„SP“	„RecordMode“	True

This sets the parameter *RecordMode* to the Boolean value *True*. It is up to the recorder to generate a script line with a valid textual representation of Boolean values, such as “true”, “True” or “TRUE” for example.

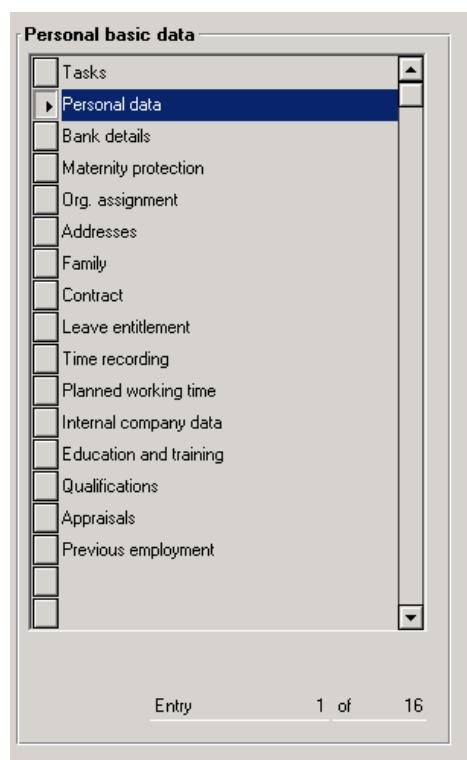
Type	Method/Property name	Parameters
„SP“	„TestToolMode“	0

This sets the parameter *TestToolMode* to value 0.

Type	Method/Property name	Parameters
„M“	„Resize“	96 32 False

The method *Resize* is called with three parameters. In this case the third member of the *CommandArray* is an array with 3 elements.

There are cases in which the *CommandArray* contains more than one line.



If a row is selected in this table control, two entries are added to the generated *Change* event's *CommandArray* parameter.

Type	Method/Property name	Parameters
„M“	„getAbsoluteRow“	1
“SP”	“selected”	True

The script code required to select this line should then look like this:

```
Session.findById("wnd[0]/usr/tblSAPMBIBSTC537").
    getAbsoluteRow("1").selected = "True"
```

Event contextMenu (session As GuiSession, component As GuiVComponent)

The contextMenu event is fired when SAP GUI is about to display a context menu. There are currently the following limitations:

- Only context menus of controls of type *GuiShell* are supported.
- The event is not fired for “cached” context menus, which are not retrieved from the server when being opened.

Event destroy (session As GuiSession)

This event is raised before a session is destroyed.

**Event automationFCCode (session As GuiSession, fcode As String)**

The event is only fired when using the SAP Workplace. It notifies the listener that SAP GUI executes a function code that was set by the Workplace framework.

Event hit (session As GuiSession, component As GuiComponent, innerObject As String)

The *hit* event is only raised when *elementVisualizationMode* is set to *True*, which turns on the hit test mode of SAP GUI. If in this mode a SAP GUI component is identified, the *hit* event is raised. The parameters of this event are

- The session on which the component was hit
- The component that was hit
- A description of an inner object of the component if an inner object was hit



Simple Visual Objects

Simple visual objects are those that are not containers, which means they have no children.



GuiBox

A *GuiBox* is a simple frame with a name. The items inside the frame are not children of the box.

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: box

Name: The fieldname taken from the SAP data dictionary.



GuiButton

GuiButton represents all push buttons that are on dynpros, the toolbar or in table controls.

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: btn

Name: The fieldname taken from the SAP data dictionary. There is one exception: for tabstrip buttons, it is the button id set in screen painter that is taken from the SAP data dictionary.

Function press

This emulates manually pressing a button. Pressing a button will always cause server communication to occur, rendering all references to elements below the window level invalid. The following code will therefore fail:

```
Set TextField = session.findById(".../txtF1")
session.findById(".../btnPB5").press
TextField.text = "Hello"
```



GuiRadioButton

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: rad

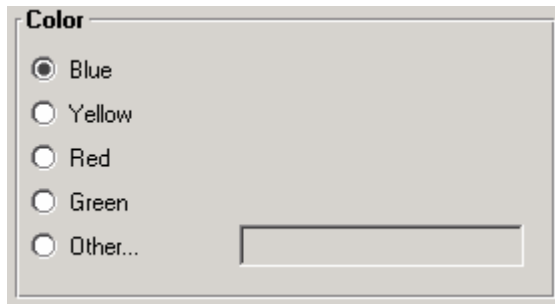
Name: Fieldname taken from the SAP data dictionary.

Property selected As Boolean (Read only)

This property is *True* if the radio button is selected.

Function select

Selecting a radio button automatically deselects all the other buttons within that group. This may cause a server roundtrip, depending on the definition of the button in the screen painter.



In this example selecting the radio button 'Other...' causes the SAP server to enable the text field.



GuiCheckBox

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: chk

Name: The fieldname taken from the SAP data dictionary.

Property selected As Boolean

Like radio buttons, checking a checkbox can cause server communication, depending on the ABAP screen painter definition.



GuiLabel

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: lbl

Name: The fieldname taken from the SAP data dictionary.

Property maxLength As Long (Read only)

The maximum text length of a label is counted in code units. On non-Unicode clients these are equivalent to bytes.

Property numerical As Boolean (Read only)

This flag is *True* if the label may only contain numbers.

Property caretPosition As Long

Setting the caret position within a label is possible even though it is not visualized as a caret by SAP GUI. However, the position is transmitted to the server, so ABAP application logic may depend on this position.

Property highlighted As Boolean (Read only)

Field types		
Normal	Hollenbach	
Normal right-justified	4530	
Highlighted	Hollenbach	
Scrollable	Schneider G...	
With check text	543	Omikron Delta
With scrollable check txt	34	Pump 200 kW, suction capa...

This is an example of both a highlighted *GuiLabel* on the left and a highlighted *GuiTextField* on the right. The *highlighted* property is defined in the data dictionary.



GuiTextField

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: txt

Name: The fieldname taken from the SAP data dictionary.

Property maxLength As Long (Read only)

The maximum length of text that can be written in a text field is counted in code units. On non-Unicode clients these are equivalent to bytes.

Property numerical As Boolean (Read only)

If this flag is set only numbers and special characters may be written into the text field.

Property caretPosition As Long

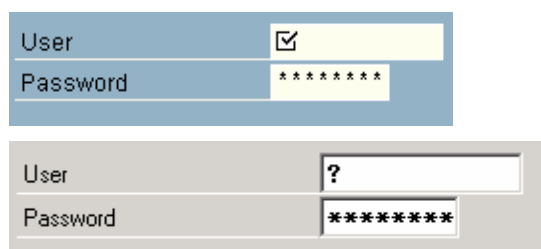
The position of the caret within a text field may be checked by the ABAP application to determine which word the caret is in. Among other things this is useful for context sensitive help.

Property highlighted As Boolean (Read only)

See *GuiLabel* for an example.

Property required as Boolean (Read only)

The following example shows the required text field for the user name, in both New Visual Design and in classic mode.



GuiPasswordField

The only difference between *GuiTextField* and *GuiPasswordField* is that the *Text* property can not be read for a password field. The returned text is always empty.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiTextField*

Type prefix: pwd

Name: The fieldname taken from the SAP data dictionary.



GuiCTextField

If the cursor is set into a text field of type *GuiCTextField* a combo box button is displayed to the right of the text field. Pressing this button is equivalent to pressing the F4 key. The button is not represented in the scripting object model as a separate object; it is considered to be part of the text field.

There are no other differences between *GuiTextField* and *GuiCTextField*.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiTextField*

Type prefix: ctxt

Name: Fieldname taken from the SAP data dictionary.



This is an example of *GuiCTextField* type text field, where the upper field has the focus. Please note that the button is only displayed when the corresponding input field has the focus.



GuiComboBox

The *GuiComboBox* looks somewhat similar to *GuiCTextField*, but has a completely different implementation. While pressing the combo box button of a *GuiCTextField* will open a new dynpro or control in which a selection can be made, *GuiComboBox* retrieves all possible choices on initialization from the server, so the selection is done solely on the client.

A combo box can be configured to send a function code to the server when the selection changes, which will invalidate references to all visible elements below the window level.

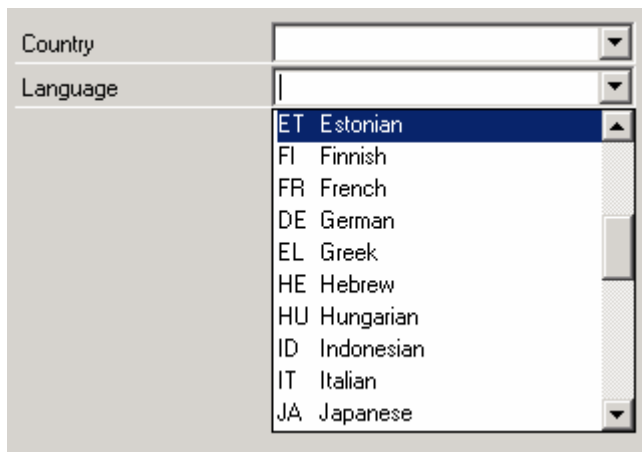
Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: cmb

Name: The fieldname taken from the SAP data dictionary.

Property entries As *GuiCollection*¹⁵ (Read only)

All members of this collection are of **GuiComboBoxEntry** type and have just two properties, *key* and *value*, both of type String. The key data can be displayed in SAP GUI by setting the 'Show keys...' check box on the option dialog's expert page.



In this example the first column contains the *key* property and the second column contains the *value* property.

Property key As String

This is the key of the currently selected item. You can change this item by setting the *key* property to a new value.

Property value As String

This is the value of the currently selected item. You can change this item by setting the *value* property to a new value.

Property required As Boolean (Read only)

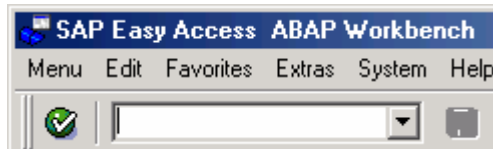
¹⁵ See the chapter entitled 'Collection interfaces' for a description of *GuiCollection*

If the *required* flag is set for a combo box then the empty entry is not selectable from the list.



GuiOkCodeField

The *GuiOkCodeField* is placed on the upper toolbar of the main window. It is a combo box into which commands can be entered. Setting the text of *GuiOkCodeField* will not execute the command until server communication is started, for example by emulating the *Enter* key (VKey 0).



Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: okcd

Name: Not available

Property opened As Boolean

In New Visual Design the *GuiOkCodeField* can be collapsed using the arrow button to the right of it.

Open:



Closed:



GuiStatusbar

GuiStatusbar represents the message displaying part of the statusbar on the bottom of the SAP GUI window. It does not include the system and login information displayed in the rightmost area of the statusbar as these are available from the *GuiSessionInfo* object.

The *text* property of the *GuiStatusbar* can not be changed.

Supported base interfaces: *GuiComponent*, *GuiVComponent*

Type prefix: sbar

Name: Not available.

Property `messageType As String` (Read only)

This property may have any of the following values:

Value	Description
S	Success
W	Warning
E	Error
A	Abort
I	Information

Property `messageId As String` (Read only)

This is the name of the message class used in the ABAP `message` call.

Property `messageNumber As String` (Read only)

This is the name of the message number used in the ABAP `message` call. It will usually be a number, but this is not enforced by the system.

Property `messageParameter (index As Long) As String` (Read only)

These are the values of the parameters used to expand the placeholders in the message text definition in the data dictionary. The `text` property of the `GuiStatusbar` already contains the expanded text of the message. A maximum of 8 parameter values can be provided in the ABAP coding, so `index` should be in the range from 0 to 7.

The ABAP language line

```
message e319(01) with 'test1' 'test2' 'test3' 'test4'.
```

will result in the following property values:

```
Text           = E:           test1 test2 test3 test4
Type           = E
Id             = 01
Number        = 319
Parameter 0    = test1
Parameter 1    = test2
Parameter 2    = test3
Parameter 3    = test4
Parameter 4    =
Parameter 5    =
Parameter 6    =
Parameter 7    =
as Popup       = False
```

The message 319 in message class 01 is defined as ' & & & &', with '&' being a placeholder.

Property messageAsPopup As Boolean (Read only)

Some messages may be displayed not only on the statusbar but also as a pop-up window. In such cases, this property is set to *True* so that a script knows it has to close a pop-up to continue.



Visual Container Objects

Visual containers are objects that are part of the user interface and have other visual objects as children.



GuiFrameWindow

A *GuiFrameWindow* is a high level visual object in the runtime hierarchy. It can be either the main window or a modal popup window. See the *GuiMainWindow* and *GuiModalWindow* sections for examples. *GuiFrameWindow* itself is an abstract interface.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: wnd

Name: wnd[n]



Property **iconic** As Boolean (Read only)

This property is *True* if the window is iconified. It is possible to execute script commands on an iconified window, but there may be undefined results, especially when controls are involved, as these may have invalid size settings.

Property **workingPaneWidth** As Long (Read only)

This is the width of the working pane in character metric. The working pane is the area between the toolbars in the upper area of the window and the statusbar at the bottom of the window.

Property **workingPaneHeight** As Long (Read only)

This is the height of the working pane in character metric.



Property **handle** As Long (Read only)

This is the handle of the underlying window in Microsoft Windows.



Property **systemFocus** As GuiVComponent (Read only)

The *systemFocus* specifies the component that the SAP system is currently seeing as being focussed. This value is only valid for dynpro elements and might therefore differ from the focus as seen on the frontend.



Property **guiFocus** As GuiVComponent (Read only)

The *systemFocus* only supports dynpro elements. To receive information about the currently focussed ActiveX control you can access the *guiFocus* property.

Property **elementVisualizationMode As Boolean**

When *elementVisualizationMode* is enabled, a hit test can be performed on SAP GUI by moving the cursor over the window. The *hit* event of the session is fired when a component was found at the mouse position.



Function **iconify**

This will set a window to the iconified state. It is not possible to iconify a specific window of a session; both the main window and all existing modals will be iconified.



Function **restore**

This will restore a window from its iconified state. It is not possible to restore a specific window of a session; both the main window and all existing modals will be restored.

Function **showMessageBox (title As String, Text As String, msgIcon As Long, msgType As Long) As Long**

This method shows the message box modal to the *GuiFrameWindow*. The *title* and *text* parameters set the title and text of the message box. The *msgIcon* parameter sets the icon to be used for the message box and should be set to one of the *MESSAGE_TYPE_** constants¹⁶. *msgType* sets the buttons available on the message box and should be set to one of the *MESSAGE_OPTION** constants. The return value will be one of the *MESSAGE_RESULT_** values.



Function **maximize**

This will maximize a window. It is not possible to maximize a modal window; it is always the main window which will be maximized.

Function **close**

The function attempts to close the window. Trying to close the last main window of a session will not succeed immediately; the dialog 'Do you really want to log off?' will be displayed first.

Function **isVKeyAllowed (VKey As Integer) As Boolean**

This function returns *True* if the virtual key *VKey* is currently available. The *VKeys* are defined in the menu painter.

¹⁶ All message box constants are defined in the object *GuiUtils*. See the 'GuiUtils' chapter.

Function sendVKey (VKey As Integer)

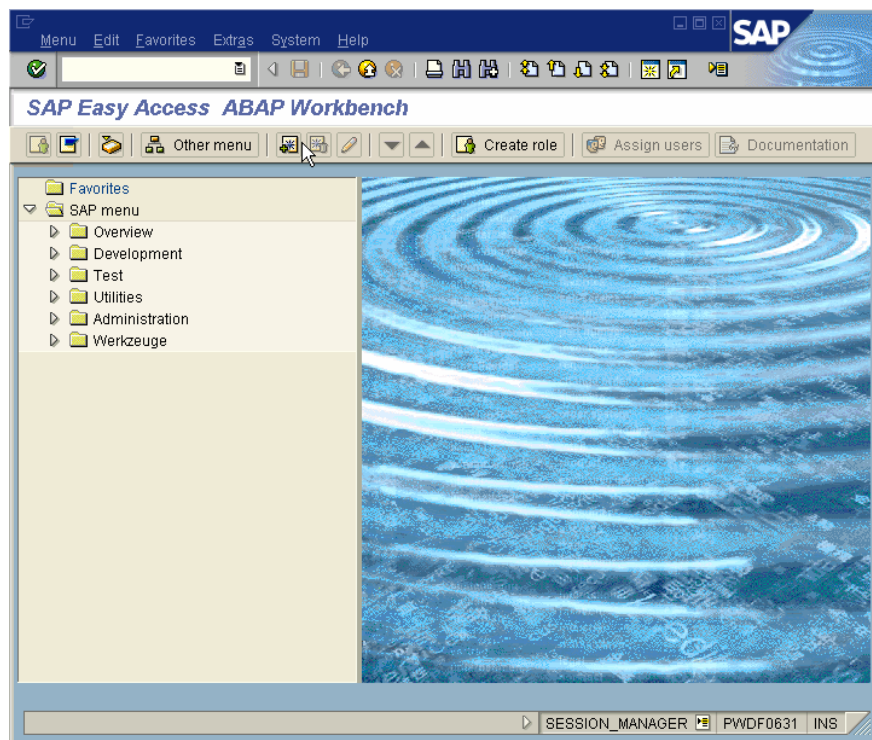
The virtual key *VKey* is executed on the window. The *VKeys* are defined in the menu painter.

Function hardCopy (name As String) As String

This function dumps a hardcopy of the window as a bitmap file to disk. The parameter is the name of the. If the function succeeds, then the return value will be the fully qualified path of the file. If no path information is given, then the file will be written to the SapWorkDir.

**GuiMainWindow**

This window represents the main window of a SAP GUI session.



Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*, *GuiFrameWindow*

Type prefix: wnd

Name: wnd[n]

Property toolbarVisible As Boolean

Property statusBarVisible As Boolean



Property buttonbarVisible As Boolean



Property titlebarVisible As Boolean



When SAP GUI is integrated into other applications, the toolbars, the status- and the titlebar may not be required for operating transactions. You may then save space by not displaying them, if you set the respective property to *False*.

Function `resizeWorkingPane` (width As Long, height As Long, throwOnFail As Boolean)

The `resizeWorkingPane` function will resize the window so that the available working area has the given width and height in character metric. A script may fail during playback if the size of the window differs from the size during recording. This becomes obvious when you compare the two screen shots below:

Type/variant
One-Line Table
One-Line Table with Toolbar
One-Line Table - ALV Grid Control Standard Call
One-Line Table with Threshold Values
One-Line Table with Total
One-Line Table with Total and Subtotal
One-Line List
One-Line List with Button Bar
One-Line List - ALV Standard Call
One-Line List with Top of Page
Design of the header information (top of page)
One-Line List with Threshold Values
One-Line List with Total
One-Line List with Total and Subtotal
Single-line list with several totals lines and heading levels
Single-line list with hierarchical headings
Single-line list with inserted lines
Two-Line List
Three-Line List
Hierarchical Sequential List (one line)

If the user selected 'Two-Line List' while recording a script and this line is not available during playback because the size of the window has been decreased, then the selection will fail.

Type/variant
One-Line Table
One-Line Table with Toolbar
One-Line Table - ALV Grid Control Standard Call
One-Line Table with Threshold Values
One-Line Table with Total
One-Line Table with Total and Subtotal
One-Line List
One-Line List with Button Bar
One-Line List - ALV Standard Call
One-Line List with Top of Page
Design of the header information (top of page)
One-Line List with Threshold Values
One-Line List with Total
One-Line List with Total and Subtotal
Single-line list with several totals lines and heading levels
Single-line list with hierarchical headings
Single-line list with inserted lines

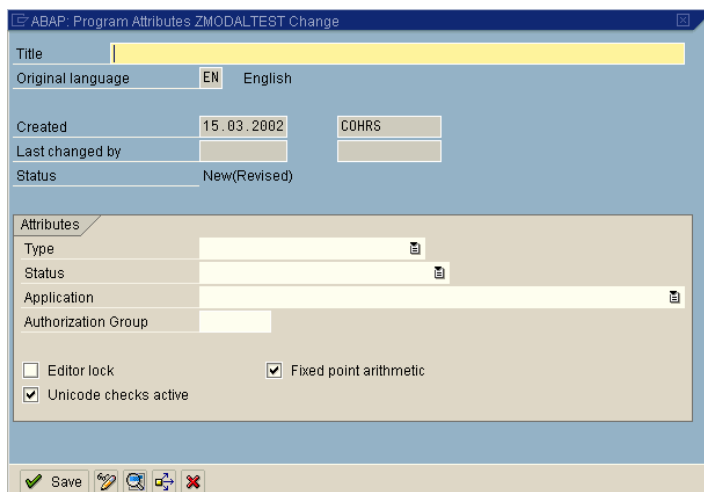
This is because only the visible lines of the list are available on the client. Scrolling always requires a new set of data to be read from the server. This may not be the case for other elements of the user interface. For example, a line of the Grid View which will be discussed later is accessible even if it is not currently visible on the client.

The *throwOnFail* parameter has been added for use in the SAP GUI for Java because some window managers may not support a program driven resize of a window.



GuiModalWindow

A *GuiModalWindow* is a dialog pop-up.



Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*, *GuiFrameWindow*

Type prefix: wnd

Name: wnd[n]



GuiUserArea

The *GuiUserArea* comprises the area between the toolbar and statusbar for windows of *GuiMainWindow* type and the area between the titlebar and toolbar for modal windows, and may also be limited by docker controls.

The standard dynpro elements can be found only in this area, with the exception of buttons, which are also found in the toolbars.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: usr

Name: usr

Property `horizontalScrollbar As GuiScrollbar`¹⁷

The user area is defined to be scrollable even if the scrollbars are not always visible.

Property `verticalScrollbar As GuiScrollbar`¹⁸

The user area is defined to be scrollable even if the scrollbars are not always visible.

Function `findByLabel (label As String, type as Long)`



A very simple method for finding an object is to search by specifying the text of the respective label.

Field types	
Normal	Hollenbach
Normal right-justified	4530
Highlighted	Hollenbach

In this case the upper text field can be found by searching for a *GuiTextField* with the label text 'Normal' describing it. This match is done through searching for a label with the given text and then checking if there is a matching object on the same line. This should be only used for simple dynpros.

Function `selectContextMenuItem (key as String)`

This function emulates the selection of an item from the user area's context menu.



GuiSimpleContainer

This container represents non-scrollable subscreens. It does not have any functionality apart from to the inherited interfaces.

¹⁷ See the chapter 'Utility Classes' for details

¹⁸ See the chapter 'Utility Classes' for details

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: sub

Name: The name of a *GuiSimpleContainer* is generated from the data dictionary settings.



GuiScrollContainer

This container represents scrollable subscreens. A subscreen may be scrollable without actually having a scrollbar, because the existence of a scrollbar depends on the amount of data displayed and the size of the *GuiUserArea*.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: ssub

Name: The name of a *GuiScrollContainer* is generated from the data dictionary settings.

Property horizontalScrollbar As GuiScrollbar¹⁹

Property verticalScrollbar As GuiScrollbar²⁰



GuiTitlebar

The titlebar is only displayed and exposed as a separate object in New Visual Design mode.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: titl

Name: titl

In some transactions the titlebar may contain objects of *GuiGosShell* type.



GuiToolbar

Every *GuiFrameWindow* has a *GuiToolbar*. The *GuiMainWindow* has two toolbars unless the second has been turned off by the ABAP application. The upper toolbar is the system toolbar, while the second toolbar is the application toolbar.

The children of a *GuiToolbar* are buttons. The indexes for toolbar buttons are determined by the virtual key values defined for the button.

¹⁹ See the chapter 'Utility Classes' for details

²⁰ See the chapter 'Utility Classes' for details

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: tbar

Name: tbar



GuiMenubar

Only the main window has a menubar. The children of the menubar are menus.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: mbar

Name: mbar



GuiMenu

A *GuiMenu* may have other *GuiMenu* objects as children.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: menu

Name: Text of the menu item. If the item does not have a text, which is the case for separators, then the name is the last part of the id, *menu[n]*.

Function select

Select the menu.



GuiContextMenu



A *GuiContextMenu* may have other *GuiContextMenu* objects as children.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*, *GuiMenu*

Type prefix: mnu

In contrast to the *GuiMenu* class the *Name* property of this class is the function code that is sent to the system when the menu item is selected.



GuiCustomControl

The *GuiCustomControl* is a wrapper object that is used to place ActiveX controls onto dynpro screens. While *GuiCustomControl* is a dynpro element itself, its children are of *GuiContainerShell* type, which is a container for controls.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: cntl

Name: Fieldname taken from the SAP data dictionary.



GuiContainerShell

A *GuiContainerShell* is a wrapper for a set of *GuiShell* objects.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: shellcont

Name: The name is the last part of the id, *shellcont[n]*.



GuiDockShell

A *GuiDockShell* is a wrapper for the docker control. It is represented by a *GuiContainerShell*.



GuiShell

GuiShell is an abstract object whose interface is supported by all the controls.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: shell

Name: The name is the last part of the id, *shell[n]*.

Property subType As String

Additional type information to identify the control represented by the shell, for example *Picture*, *TextEdit*, *GridView*...



Property handle As Long

The window handle of the control that is connected to the *GuiShell*.



Property currentContextMenu As GuiContextMenu

This property is only set when a context menu is available at the shell object.

Function selectContextMenuItem (functionCode As String)

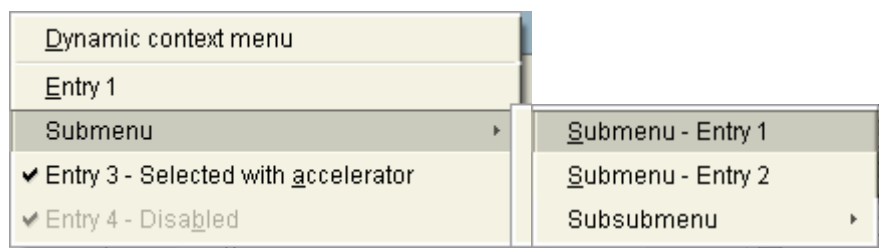
Select an item from the control's context menu.



Function selectContextMenuItemByText (text As String)

In some cases executing a context menu item may fail because the key is generated dynamically by the SAP server application whenever the transaction is started. The value required for selecting the menu item may then differ from the value that was recorded. This problem can be solved by selecting menu items by text instead of key. When items from submenus are selected, the texts of the menu items in the hierarchy should be concatenated with the pipe character '|' between two entries.

In the following example the text to use as parameter of this function would be "Submenu|Submenu - Entry 1".



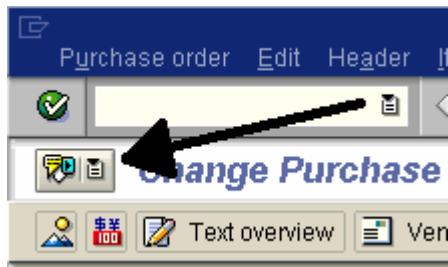
Function selectContextMenuItemByPosition (positionDesc As String)

One disadvantage of selecting menu items by text is that the parameter values in the script depend on the language settings of the SAP server connection. This can be avoided by selecting the entry by position rather than by text. The parameter value required to select the menu item above is '3|0'. Please note that the pipe symbol '|' is used to separate the position values.



GuiGOSShell

The GuiGosShell is only available in New Visual Design mode, for example in transaction me22.



It is a child of the titlebar and will contain another shell, in this example a toolbar control.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

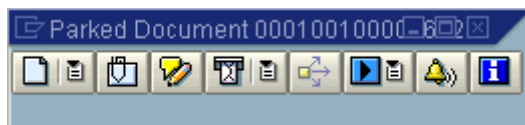
Type prefix: shell

Name: The name is the last part of the id, *shellcont[n]*.



GuiDialogShell

The *GuiDialogShell* is an external window that is used as a container for other shells, for example a toolbar.



Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: shellcont

Name: The last part of the id, *shellcont[n]*.

Function close

This method closes the external window.



GuiTabStrip

A tab strip is a container whose children are of type *GuiTab*.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: tabs

Name: Fieldname taken from the SAP data dictionary.

The screenshot shows a window with four tabs: 'Period closing', 'General data', 'Investment management', and 'Desig...'. The 'General data' tab is active. The form contains the following fields:

Applicant no.		Request date	
Tel.no.		Department	
Person respons.		Start of work	
Telephone no.		Date work ends	
Estimated costs		<input type="checkbox"/> Work permit	
Selection group			

The children of the tab strip are the tabs. While all tabs are available at any given time, only the children of the selected tab exist in the object hierarchy for server driven tab strips. So in this example, the text field 'Applicant no.' can only be found if the tab labelled 'General data' has been selected.

In some transactions there are local tabs strips where all tabs are available without further server access being required.

Property `leftTab` As `GuiTab` (Read only)

This is the left most tab whose caption is visible. In the example above it is the one with text 'Period closing'. The *leftTab* property can be changed by calling the *ScrollToLeft* method of a different *GuiTab*, as described below.

Property `selectedTab` As `GuiTab` (Read only)

The selected tab is the one whose descendants are currently visualized, in the example above it is the 'General data' tab. The selected tab has exactly one child, which is a *GuiScrollContainer*.



GuiTab

The *GuiTab* objects are the children of a *GuiTabStrip* object.

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: tabp

Name: Id of the tab's button taken from SAP data dictionary.

Function `scrollToLeft`

scrollToLeft shifts the tabs so that a certain tab becomes the *leftTab* of the tab strip.

Function select

This function sets the tab to be the tab strip's selected tab. Changing the selected tab of a tab strip may cause server communication.



GuiTableControl

The table control is a standard dynpro element, in contrast to the *GuiCtrlGridView*, which looks similar.

P...	U..	BA	Account	Description	VA	Tax	Amount
001	50		0000276000	Cash discount received	V1	4,50-	30,00-
002	25		0000000010	Soesel & Partner Gmb...		0,00	1.150,00-
003	40		0000160099	Vendor-obligatory		0,00	0,00
004	50	0004	0000113101	DeuBa (outgoing cheq...		0,00	1.115,50-
005	50		0000154000	Input tax (FRG)	V1	0,00	4,50-
006	50		0000276000	Cash discount received	V1	4,50-	30,00-
007	25		0000000010	Soesel & Partner Gmb...		0,00	1.150,00-
008	40		0000160099	Vendor-obligatory		0,00	0,00
009	50	0004	0000113101	DeuBa (outgoing cheq...		0,00	1.115,50-
010	50		0000154000	Input tax (FRG)	V1	0,00	0,45-
011	50		0000276000	Cash discount received	V1	0,45-	30,00-
012	25		0000000010	Soesel & Partner Gmb...		0,00	1.150,00

Supported base interfaces: *GuiComponent*, *GuiVComponent*, *GuiVContainer*

Type prefix: tbl

Name: Fieldname taken from the SAP data dictionary.

Property columns As *GuiCollection*²¹ (Read only)

The members of this collection are of *GuiTableColumn* type. Therefore they do not support properties like *id* or *name*.

Property rows As *GuiCollection*²² (Read only)

The members of this collection are of *GuiTableRow* type. Indexing starts with 0 for the first visible row, independent of the current position of the horizontal scrollbar. After scrolling, a different row will have the index 0.

Property colSelectMode As *GuiTableSelectionType* (Read only)

Property rowSelectMode As *GuiTableSelectionType* (Read only)

²¹ See the 'Collection interfaces' chapter for a description of *GuiCollection*

²² See the 'Collection interfaces' chapter for a description of *GuiCollection*

There are three different modes for selecting columns or rows, which are defined in the enumeration type *GuiTableSelectionType*.

NO_SELECTION	0	No selection possible.
SINGLE_SELECTION	1	One column/row can be selected.
MULTIPLE_INTERVAL_SELECTION	2	Several columns/rows can be selected

Property `horizontalScrollbar` As `GuiScrollbar`²³

The horizontal scrollbar of the table control.

Property `verticalScrollbar` As `GuiScrollbar`²⁴

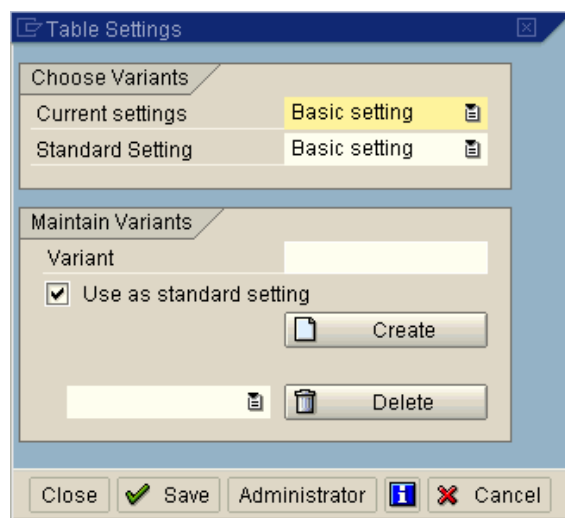
The vertical scrollbar of the table control.

Function `reorderTable` (permutation As String)

The parameter *permutation* describes a new ordering of the columns. For example "1 3 2" will move column 3 to second position.

Function `configureLayout`

In the configuration dialog the layout of the table can be changed. This dialog is a *GuiModalWindow*.



Function `getAbsoluteRow` (index As Long) As `GuiTableRow`

Unlike the *rows* collection, the indexing supported by this function does not reset the index after scrolling, but counts the rows starting with the first row with respect to the first scroll position. If the selected row is not currently visible then an exception is raised.

²³ See the 'Utility Classes' chapter for details

²⁴ See the 'Utility Classes' chapter for details



GuiTableColumn

GuiTableColumn extends *GuiComponentCollection*²⁵ with the following properties and methods.

Property title As String (Read only)

This is the caption of the column. In the example above the script

```
MsgBox session.findById
("wnd[0]/usr/tblSAPMBIBSTC535").columns(3).title
```

displays the text 'Account'.

Property fixed As Boolean (Read only)

Some columns may be fixed, which means that they will not be scrolled with the rest of the columns.

Property selected As Boolean

In the following example the column 'User' is selected:

User	TC...	Terminal	
Adam	SE38	p05722	▲
Maier	BIBS	hw1432	▼
Siegfried	AW45	p02312	
Newman	QR23	hs1321	
Goldberg	SA38	w04722	
Paul	TK10	hs3462	
Engel	UR32	q02312	
Reichhard	GR12	sk1231	

Property width As Long

The width of a column can be changed.



GuiTableRow

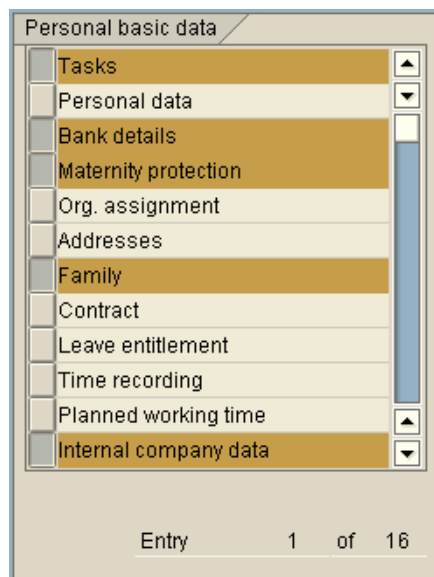
GuiTableRow extends *GuiComponentCollection*²⁶ with the following property.

²⁵ See the chapter 'Collection interfaces' chapter for a description of *GuiComponentCollection*

²⁶ See the 'Collection interfaces' chapter for a description of *GuiComponentCollection*

Property selected As Boolean

In this example multiple lines are selected.





Controls

The classes in this chapter represent the controls that were introduced in SAP Release 4.6C. All of these classes extend the *GuiShell* class.



GuiCtrlGridView

The grid view is similar to the dynpro table control, but significantly more powerful.

ID	No.	Flight date	FlgtPrice	Curr.	Plane
A	17	03.05.2001	94.966	ITL	727-2
AA	17	04.05.2001	949,66	USD	747-4
AA	17	05.05.2001	949,66	USD	747-4
AA	17	06.05.2001	949,66	USD	737-2
AA	17	07.05.2001	949,66	USD	747-4
AA	17	08.05.2001	949,66	USD	A330-
AA	17	09.05.2001	949,66	USD	747-4
AA	17	10.05.2001	949,66	USD	737-2
AA	26	03.05.2001	1.421,91	USD	A310-

Type library: GridViewScripting.dll

Property subType As String (Read only)

This property has the constant value "GridView".

Property currentCellRow As Long

The row index of the current cell ranges from 0 to the number of rows less 1, with -1 being the index of the title row.

Property currentCellColumn As String

The string identifying a column is the field name defined in the SAP data dictionary. In the example above the identifiers are named CARRID, CONNID, FLDATE, PRICE etc.

Property selectedColumns As GuiCollection

The selected columns are available as a collection of strings like the *currentCellColumn* string. Setting this property can raise an exception, if the new collection contains an invalid column identifier.

Property selectedRows As String

The string is a comma separated list of row index numbers or index ranges, such as "1,2,4-8,10".

Setting this property to an invalid string or a string containing invalid row indices will raise an exception.

Property selectedCells As GuiCollection

The collection of selected cells contains strings, each of which has the format "<index of the row>,<column identifier>", such as "0,CARRID". Trying to set this property to an invalid value will raise an exception.

Property columnCount As Long (Read only)

This property represents the number of columns in the control.

Property rowCount As Long (Read only)

This property represents the number of rows in the control.

Property frozenColumnCount As Long (Read only)

This property represents the number of columns that are excluded from horizontal scrolling.

Property firstVisibleRow As Long

This is the index of the first visible row in the grid. Setting this property to an invalid row index will raise an exception.

Property firstVisibleColumn As String

This property represents the first visible column of the scrollable area of the grid view. Fixed columns are ignored. So the *firstVisibleColumn* property in the example on the previous page would be "PRICE". Setting the property to an invalid column identifier will raise an exception.

Property columnOrder As GuiCollection

This collection contains all the column identifiers in the order in which they are currently displayed. Passing an invalid column identifier to this property will raise an exception.

Property toolbarButtonCount As Long (Read only)

The number of toolbar buttons including separators.

Property selectionMode As String (Read only)

Possible values are:

"RowsAndColumns": Only rows and columns can be selected. Individual rectangular areas of cells are not allowed.

"ListboxSingle": Only one single row can be selected.

"ListboxMultiple": One or more rows can be selected.

“Free”: Any kind of selection can be made.

Property visibleRowCount As Long (Read only)

Retrieves the number of visible rows of the grid.

Function clearSelection

Calling *clearSelection* removes all row, column and cell selections.

Function doubleClick (row As Long, column As String)

This function emulates a mouse double click on a given cell if the parameters are valid and raises an exception otherwise.

Function doubleClickCurrentCell

This function emulates a mouse double click on the current cell.

Function click (row As Long, column As String)

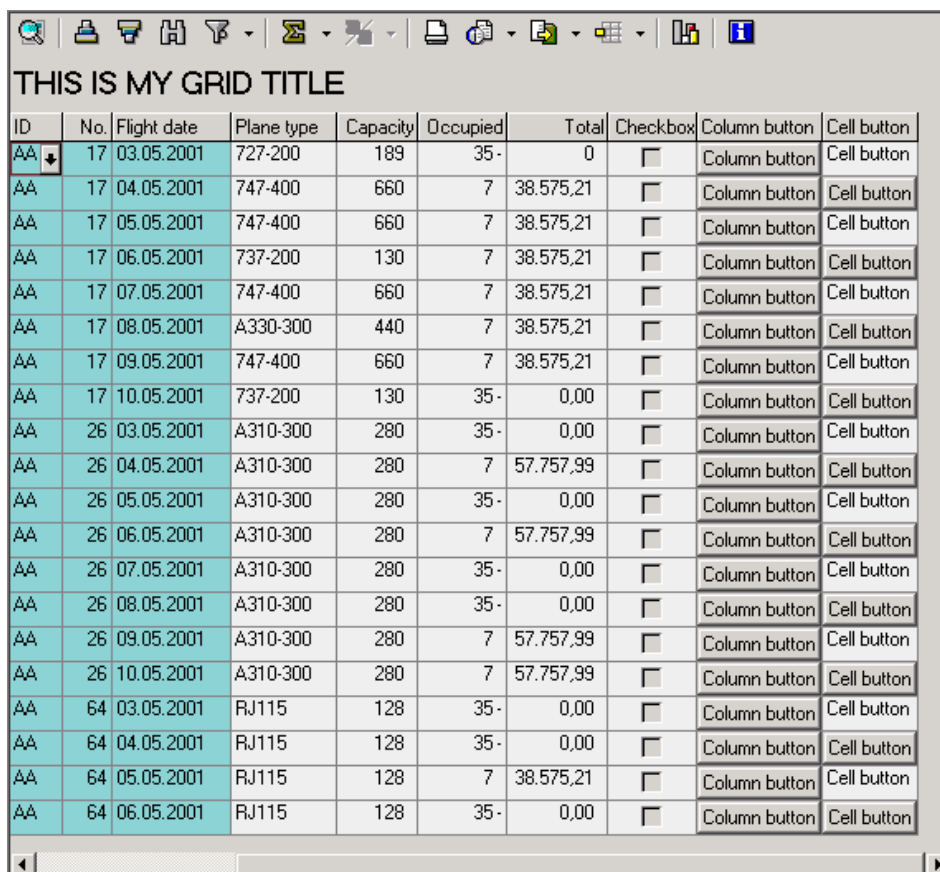
This function emulates a mouse click on a given cell if the parameters are valid and raises an exception otherwise.

Function clickCurrentCell

This function emulates a mouse click on the current cell.

Function pressButton (row As Long, column As String)

This function emulates pressing a button placed in a given cell. It will raise an exception if the cell does not contain a button, or does not even exist.



ID	No.	Flight date	Plane type	Capacity	Occupied	Total	Checkbox	Column button	Cell button
AA	17	03.05.2001	727-200	189	35-	0	<input type="checkbox"/>	Column button	Cell button
AA	17	04.05.2001	747-400	660	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	17	05.05.2001	747-400	660	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	17	06.05.2001	737-200	130	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	17	07.05.2001	747-400	660	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	17	08.05.2001	A330-300	440	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	17	09.05.2001	747-400	660	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	17	10.05.2001	737-200	130	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	26	03.05.2001	A310-300	280	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	26	04.05.2001	A310-300	280	7	57.757,99	<input type="checkbox"/>	Column button	Cell button
AA	26	05.05.2001	A310-300	280	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	26	06.05.2001	A310-300	280	7	57.757,99	<input type="checkbox"/>	Column button	Cell button
AA	26	07.05.2001	A310-300	280	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	26	08.05.2001	A310-300	280	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	26	09.05.2001	A310-300	280	7	57.757,99	<input type="checkbox"/>	Column button	Cell button
AA	26	10.05.2001	A310-300	280	7	57.757,99	<input type="checkbox"/>	Column button	Cell button
AA	64	03.05.2001	RJ115	128	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	64	04.05.2001	RJ115	128	35-	0,00	<input type="checkbox"/>	Column button	Cell button
AA	64	05.05.2001	RJ115	128	7	38.575,21	<input type="checkbox"/>	Column button	Cell button
AA	64	06.05.2001	RJ115	128	35-	0,00	<input type="checkbox"/>	Column button	Cell button

Function `pressButtonCurrentCell`

This function emulates pressing a button placed in the current cell. It will raise an exception if the cell does not contain a button.

Function `pressColumnHeader (column As String)`

This function emulates a mouse click on the header of the column if the parameter identifies a valid column and raises an exception otherwise.

Function `pressF1`

This emulates pressing the F1 key while the focus is on the grid view.

Function `pressF4`

This emulates pressing the F4 key.

Function `pressEnter`

This emulates pressing the *Enter* key.

Function `contextMenu`

Calling *contextMenu* emulates the context menu request.

Function pressToolBarButton (id As String)

This function emulates clicking a button in the grid view's toolbar.

Function pressToolBarContextButton (id As String)

This emulates opening the context menu of the grid view's toolbar.

Function selectToolBarMenuItem (id As String)

This function emulates the selection of an item from the context menu of the grid view's toolbar. The parameter should be the function code of the item.

Function pressTotalRow (row As Long, column As String)

Pressing the total row button indicated in the screenshot expands or condenses the grouped rows. If the selected cell is not a total row cell an exception is raised.

ID	No.	Flight date	FlgtPrice	Currency	Plane type	Me
LH	402	09.05.2001	2.469,12	DEM	RJ115	
LH	402	10.05.2001	2.469,12		RJ115	
LH	454	03.05.2001	2.469,12		DC-10-10	
LH	454	04.05.2001	2.469,12		DC-10-10	
LH	454	05.05.2001	2.469,12		DC-10-10	
			46.913,28	DEM		
AA	17	03.05.2001	94.966	ITL	727-200	
_A	17	03.05.2001	120.000-		727-200	
			25.034-	ITL		
AA	17	04.05.2001	949,66	USD	747-400	
AA	17	05.05.2001	949,66		747-400	
AA	17	06.05.2001	949,66		737-200	

Function pressTotalRowCurrentCell

This function differs from *pressTotalRow* only in that it tries to press the expansion button on the current cell.

Property title As String (Read only)

This property represents title of the grid control. In the above screen the title is "My titlebar".

Function setColumnWidth (column As String, width As Long)

The width of a column can be set using this function. The width is given in characters. For proportional fonts this refers to the width of an average character. Depending on the contents of the cell more or less characters may fit in the column. If the parameter is invalid an exception is raised.

Function setCurrentCell (row As Long, column As String)

If *row* and *column* identify a valid cell, this cell becomes the current cell. Otherwise, an exception is raised.

Function **modifyCell** (*row As Long, column As String, value As String*)

If *row* and *column* identify a valid editable cell and *value* has a valid type for this cell, then the value of the cell is changed. Otherwise, an exception is raised.

Function **modifyCheckBox** (*row As Long, column As String, checked As Boolean*)

If *row* and *column* identify a valid editable cell containing a checkbox, then the value of the cell is changed. Otherwise, an exception is raised.

Function **moveRows** (*fromRow As Long, toRow As Long, destRow As Long*)

The rows with an index greater than or equal to *fromRow* up to an index less than or equal to *toRow* are moved to the position of *destRow*.

Passing invalid index values as parameters raises an exception.

Function **insertRows** (*rows As String*)

The parameter *rows* is a comma separated text of indices or index ranges, for example "3,5-8,14,15". For any single index, a new row will be added at the given index, moving the old row one line down. If a range of indexes is inserted then all the new lines are inserted as one block, before any of the old lines. The entries must be ordered and not overlap, otherwise, an exception is raised.

Example:

0	Value A
1	Value B

If *rows* is "0,1", then the resulting table would be:

0	
1	Value A
2	
3	Value B

If, on the other hand, *rows* is "0-1", then the resulting table is:

0	
1	
2	Value A
3	Value B

Function **deleteRows** (*rows As String*)

The parameter *rows* is a comma separated string of indices or index ranges, for example "3,5-8,14,15". The entries must be ordered and not overlap, otherwise an exception is raised.

Function duplicateRows (rows As String)

The parameter *rows* is a comma separated string of indices or index ranges, for example "3,5-8,14,15". For any single index a copy of the row will be inserted at the given index. If a range of indexes is duplicated then all the new lines are inserted as one block, before the old lines. The entries must be ordered and not overlap, otherwise an exception is raised.

Example:

0	Value A
1	Value B

If *rows* is "0,1" then the resulting table would be:

0	Value A
1	Value A
2	Value B
3	Value B

If on the other hand *rows* is "0-1" then the resulting table is:

0	Value A
1	Value B
2	Value A
3	Value B

Function modified

Calling *modified* notifies the server that the data in the grid view has been modified.

Function currentCellMoved

This function notifies the server that a different cell has been made the current cell. It must be called whenever the current cell is changed.

Function selectionChanged

This function notifies the server that the selection has changed.

Function getColumnTitles (column As String) As GuiCollection

This function returns a collection of strings that are used to display the title of a column. The control chooses the appropriate title according to the width of the column.

Function getDisplayedColumnName (column As String) As String

This function returns the title of the column that is currently displayed. This text is one of the values of the collection returned from the function "getColumnTitles".

Function getColumnTooltip (column As String) As String

The tooltip of a column contains a text which is designed to help the user understands the meaning of the column.

Function getToolBarButtonId (buttonPos As Long) As String

Returns the ID of the specified toolbar button, as defined in the ABAP data dictionary.

Function getToolBarButtonIcon (buttonPos As Long) As String

Returns the name of the icon of the specified toolbar button.

Function getToolBarButtonType (buttonPos As Long) As String

Returns the type of the specified toolbar button. Possible values are:
"Button", "ButtonAndMenu", "Menu", "Separator", "Group", "CheckBox"

Function getToolBarButtonEnabled (buttonPos As Long) As Boolean

Indicates if the button can be pressed.

Function getToolBarButtonText (buttonPos As Long) As String

Returns the text of the specified toolbar button.

Function getToolBarButtonChecked (buttonPos As Long) As Boolean

Returns *True* if the button is currently checked (pressed).

Function getToolBarButtonTooltip (buttonPos As Long) As String

Returns the tooltip of the specified toolbar button.

Function selectAll

This function selects the whole grid content (i.e. all rows and all columns).

Function selectColumn (column As String)

This function adds the specified column to the collection of the selected columns.

Function deselectColumn (column As String)

This function removes the specified column from the collection of the selected columns.

Function getCellValue (row As Long, column As String) as String

This function returns the value of the specified cell.

Function getCellChangeable (row As Long, column As String) as Boolean

This function returns *True* if the specified cell is changeable.

Function getCellType (row As Long, column As String) as String

This function returns the type of the specified cell. Possible values are:

"Normal", "Button", "Checkbox", "ValueList"

Function getCellCheckBoxChecked (row As Long, column As String) as Boolean

Returns *True* if the checkbox at the specified position is checked. Throws an exception if there is no checkbox in the specified cell.

**Function dumpState (innerObject As String) As GuiCollection**

The GridView accepts the following values as "innerObject":

"Toolbar": The returned GuiCollection contains information about the grid's toolbar.

"Cell(*row,column*)": The values of "row" and "column" must identify a cell of the grid (e.g. "Cell(2,PRICE)"). The returned GuiCollection contains information about the referenced cell.

**GuiCtrlCalendar**

The calendar control can be used to select single dates or periods of time.

	WN	MO	TU	WE	TH	FR	SA	SU	▲
	5	28	29	30	31	1	2	3	
FEB 2002	6	4	5	6	7	8	9	10	
	7	11	12	13	14	15	16	17	
	8	18	19	20	21	22	23	24	
	9	25	26	27	28	1	2	3	
MAR 2002	10	4	5	6	7	8	9	10	
	11	11	12	13	14	15	16	17	
	12	18	19	20	21	22	23	24	
	13	25	26	27	28	29	30	31	
APR 2002	14	1	2	3	4	5	6	7	
	15	8	9	10	11	12	13	14	
	16	15	16	17	18	19	20	21	
	17	22	23	24	25	26	27	28	
Y 2002	18	29	30	1	2	3	4	5	
	19	6	7	8	9	10	11	12	
	20	13	14	15	16	17	18	19	▼

Type library: SapCalenScripting.dll

Property subType As String (Read only)

This property has the constant value "Calendar".

Property focusDate As String

The currently focussed date (identified by the focus border; see picture above) in the calendar control is available in the format “YYYYMMDD”. In this example it is “20020320”

Property firstVisibleDate As String

This is the earliest date visible in the calendar control. In the example above the value would be “20020228”.

Property selectionInterval As String

The interval is represented by two concatenated date strings separated by a comma.

	2001		JAN 2002					FEB 2002					MAR 2002					APR 2002					MAY 2002					JUN 2002				
WN	51	52	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25					
MO	17	24	31	7	14	21	28	4	11	18	25	4	11	18	25	1	8	15	22	29	6	13	20	27	3	10	17					
TU	18	25	1	8	15	22	29	5	12	19	26	5	12	19	26	2	9	16	23	30	7	14	21	28	4	11	18					
WE	19	26	2	9	16	23	30	6	13	20	27	6	13	20	27	3	10	17	24	1	8	15	22	29	5	12	19					
TH	20	27	3	10	17	24	31	7	14	21	28	7	14	21	28	4	11	18	25	2	9	16	23	30	6	13	20					
FR	21	28	4	11	18	25	1	8	15	22	1	8	15	22	29	5	12	19	26	3	10	17	24	31	7	14	21					
SA	22	29	5	12	19	26	2	9	16	23	2	9	16	23	30	6	13	20	27	4	11	18	25	1	8	15	22					
SU	23	30	6	13	20	27	3	10	17	24	3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23					

Depending on the order in which the user has selected the dates, the value of *selectionInterval* is either “20020318,20020322” or “20020322,20020318”.

Property calcInterval As String (Read only)

This property has the same format as the *selectionInterval* property. It describes the time interval that is currently stored locally in the calendar control.

Function contextMenu(contextMenuId As Long, reserved1, reserved2, reserved2, reserved4)

Calling this function opens a context menu.

contextMenuId: indicates the cell type of the cell in which the context menu was opened:

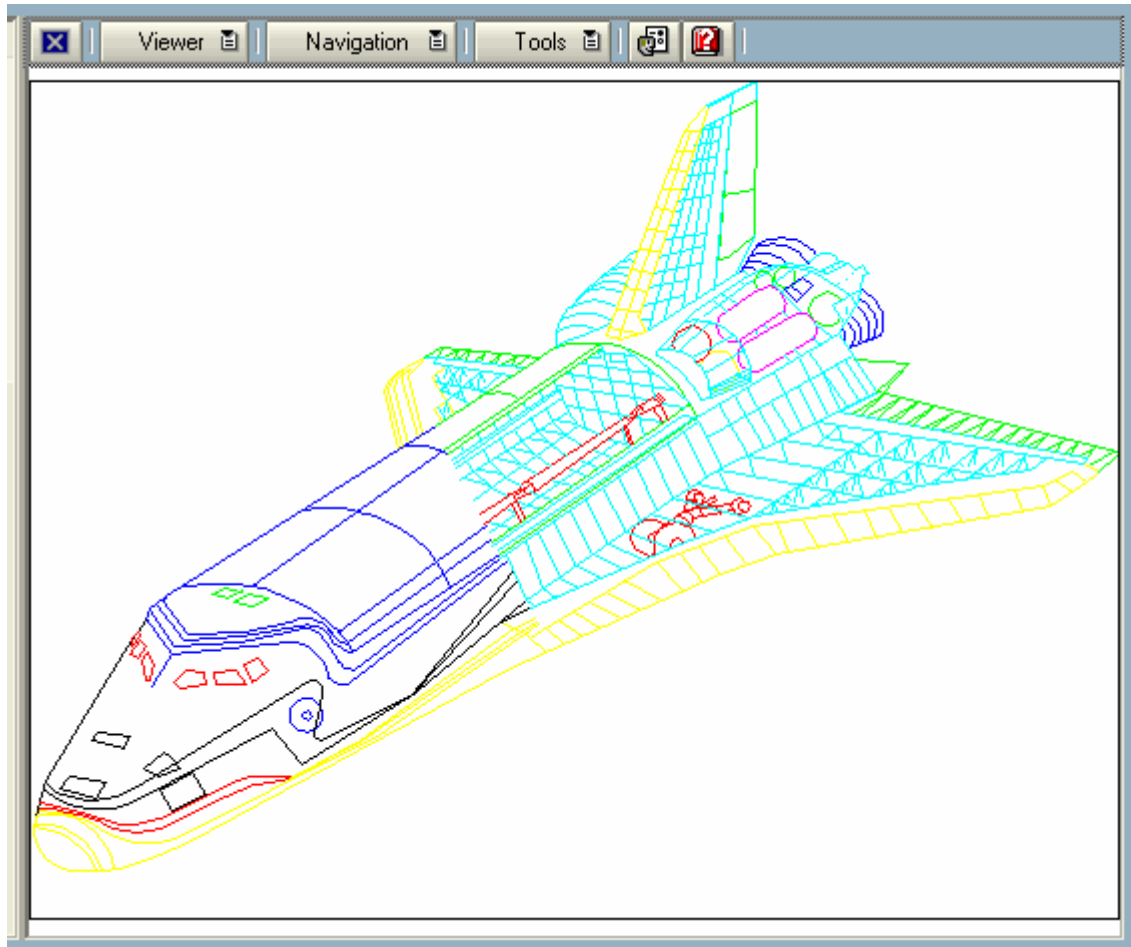
Value	Cell type	Description
0	Date	Invocation on a cell with a single date
1	Weekday	Invocation on a button for a certain day of the week.
2	Week	Invocation on a button for a specific week.

reserved1, reserved2, reserved2, reserved4: These parameters are reserved for future use. Their type is as yet unspecified. When calling this method use an empty string or the number 0.

**GuiCtrlWebViewer2D**

The Webviewer2D control is used to view 2-dimensional graphic images in the SAP system. Presently Webviewer2D supports 22 available image types. The user can carry out redlining

over the loaded image. The scripting wrapper for this control records all user actions during the redlining process and reproduces the same actions when the recorded script is replayed.



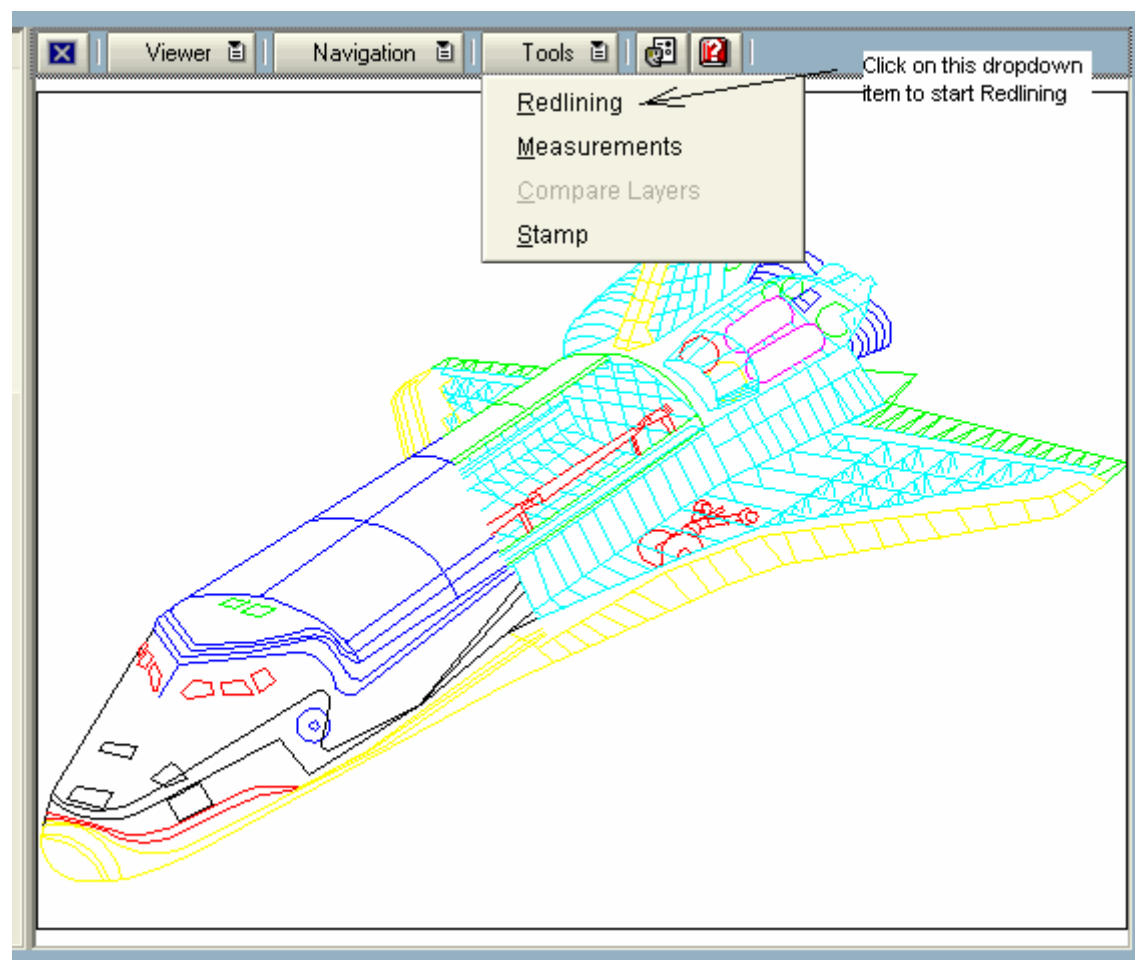
Type library: EAI2DScripting.dll

Property subType As String

This property has the constant value "EAI2D".

Property annotationEnabled As Long

The value of this property is set to 1 when redlining is started. The wrapper control starts recording user actions as soon as this property is set to value 1. The following screen shot shows how to start redlining:



Property annotationMode As Integer

During redlining, the selected redlining mode is stored in this property.

Property redliningStream As String

This property stores the redlining layer as BLOB (Binary large data object). During recording, the whole BLOB is copied into the generated script.

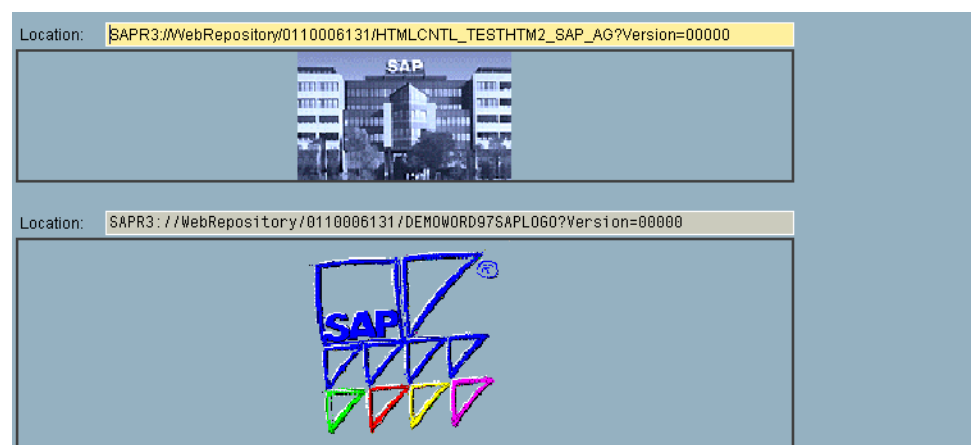


GuiCtrlPicture

Type library: SapImageScripting.dll

Property subType As String

This property has the constant value "Picture"



The picture control displays a picture on a SAP GUI screen.

Function click

This function emulates a single mouse click on a picture.

Function doubleClick

This function emulates a double click on a picture.

Function clickPictureArea (x As Long, y As Long)

The function emulates a click on a given position. The coordinates should be given in pixels with respect to the original picture file. They may differ from the pixel coordinates of the displayed picture because of scaling.

Function doubleClickPictureArea(x As Long, y As Long)

The function emulates a double click on a given position. The coordinates should be given in pixels with respect to the original picture file. They may differ from the pixel coordinates of the displayed picture because of scaling.

Function clickControlArea(x As Long, y As Long)

The function emulates a click on a given position. The coordinates should be given in pixels with respect to the picture control as it is displayed on the screen.

Function doubleClickControlArea(x As Long, y As Long)

The function emulates a double click on a given position. The coordinates should be given in pixels with respect to the picture control as it is displayed on the screen.

Function contextMenu(x As Long, y As Long)

The function opens a context menu on the given position. The coordinates should be given in pixels with respect to the picture control as it is displayed on the screen.

Property displayMode As String (Read only)

Possible values of this property are:

”Normal”: This value indicated that the picture is shown in its original size. If the picture’s size is larger than the size of the control, the control provides scrollbars. If the picture’s size is smaller than the size of the control, the picture is shown in the upper left corner of the control.

”Stretch”: The picture is resized in a way that it always occupies the complete area of the control.

”Fit”: The picture is resized on way that it fits into the control area without havin the need to show scrollbars. In contrast to ”Strech” the mode ”Fit” preserves the ratio of width and height of the picture.

”NormalCenter”: Like ”Normal” except that the picture is not shown in the upper left corner but in the center of the control.

”FitCenter”: Like ”Fit” except that the picture is not shown in the upper left corner but in the center of the control.

Property icon As String (Read only)

Returns the SAPGUI icon code (e.g. ”@01@”) of the displayed icon. If no icon is displayed, the property contains an empty string.

Property url As String (Read only)

Returns the URL of the displayed picture. If an icon is displayed (see property ”icon”), the property contains an empty string. Depending in the application that used the control the URL may contain temporary URL parts (e.g. UUIDs).

**GuiCtrlToolbar**

Type library: SapToolbScripting.dll

Property subType As String (Read only)

This property has the constant value ”Toolbar”.

Property buttonCount As Long (Read only)

The number of toolbar buttons including separators.

Function pressButton (id As String)**Preconditions:**

- Button is enabled
- *id* is a valid identifier for the toolbar button.

This function emulates pressing the button with the given id.

Function pressContextButton (id As String)

Preconditions:

- Button is enabled
- *id* is a valid identifier for the toolbar context button.

This function emulates pressing the context button with the given id.

Function selectMenuItem (id As String)**Preconditions:**

- *id* is a valid identifier for the toolbar menu item.

This function emulates selecting the menu item with the given id.

Function selectMenuItemByText (menuItemText As String)**Preconditions:**

- *menuItemText* is a valid menu item text for the toolbar menu item.

This function emulates selecting the menu item by menu item text.

Function getMenuItemIdFromPosition (position As Long) As String

This function returns the identifier of the menu item with index *Position*.

Function getButtonId (buttonPos As Long) As String

Returns the ID of the specified toolbar button.

Function getButtonIcon(buttonPos As Long) As String

Returns the name of the icon of the specified toolbar button.

Function getButtonType (buttonPos As Long) As String

Returns the type of the specified toolbar button. Possible values are:

"Button", "ButtonAndMenu", "Menu", "Separator", "Group", "CheckBox"

Function getButtonEnabled (buttonPos As Long) As Boolean

Indicates if the button can be pressed.

Function getButtonText (buttonPos As Long) As String

Returns the text of the specified toolbar button.

Function getButtonChecked (buttonPos As Long) As Boolean

Returns if the button is currently checked (pressed).

Function getButtonTooltip (buttonPos As Long) As String

Returns the tooltip of the specified toolbar button.



GuiCtrlTextEdit

The TextEdit control is a multiline edit control offering a number of possible benefits. With regard to scripting, the possibility of protecting text parts against editing by the user is especially useful.

Type library: TextEditScripting.dll

Property subType As String

This property has the constant value "TextEdit".

Property firstVisibleLine As Long

The first visible line is visualized at the top border of the control.

Property selectionIndexStart As Long (Read only)

Retrieves the absolute, zero based character index of the starting point from the visually selected text range, i.e. the position, where the selection begins. Note that a selection can be degenerated, i.e. *selectionIndexStart* is equal to *selectionIndexEnd*.

Property selectionIndexEnd As Long (Read only)

Retrieves the absolute, zero based character index of the ending point from the visually selected text range, i.e. the position where the selection ends. Note that a selection can be degenerated, i.e. *selectionIndexStart* is equal to *selectionIndexEnd*.

Property text As String

The value of this property represents the text contained in the control. Note that setting the text is denied if the control is in read only mode or a certain text part is protected.

Property numberOfUnprotectedTextParts As Long (Read only)

The number of unprotected text parts which are contained.

Function contextMenu

Calling *ContextMenu* emulates the context menu request.

Function doubleClick

This function emulates a mouse double click. For setting the selection, the function *setSelectionIndexes* can be called in advance.

Function getUnprotectedTextPart (part As Long) As String

This function retrieves the content of an unprotected text part using the zero based index *part*.

Function modifiedStatusChanged (status as Boolean)

This function emulates the change of the modified status.

Function pressF1

This function emulates pressing the F1 key on the keyboard.

Function pressF4

This function emulates pressing the F4 key on the keyboard.

Function setSelectionIndexes (start As Long, end As Long)

This function sets the visually selected text range. *start* and *end* are absolute, zero based character indexes. *start* corresponds to the position where the selection begins and *end* is the position of the first character following the selection. Note that setting *start* equal to *end* results in setting the cursor on this position.

Function setUnprotectedTextPart (part As Long, text As String) As Boolean

This function assigns the content of *text* to the unprotected text part with zero based index *part*. The function returns *True* if it was possible to perform the assignment. Otherwise, *False* is returned.

Function singleFileDropped (fileName as String)

This function emulates the drop of a single file with the directory path *fileName*.

Function MultipleFilesDropped (FileNames As GuiCollection)

Emulate a Drag&Drop operation, in which several files are dropped on the textedit control. The collection contains for each file the fully qualified file name as a string.

**GuiCtrlOfficeIntegration**

Type library: SapSdccScripting.dll

Property subType As String

This property has the constant value "OfficeIntegration".

Function setDocument (index As Long, document As String)

This function replaces or adds a new document with the specified index. The parameter *document* is the base64-representation of the binary document.

Function removeContent (name As String)

This function removes the content of a table in the table collection. The parameter *name* is the name of the table.

Function appendRow (name As String, row As String)

This function appends a new row to a table specified by the parameter *name* in the table collection. The parameter *row* is the base64 representation of the binary row.

Function customEvent (cookie As Long, eventName As String, paramCount As Long, par1 As Variant, par2 As Variant, par3 As Variant, par4 As Variant, par5 As Variant, par6 As Variant, par7 As Variant, par8 As Variant, par9 As Variant, par10 As Variant, par11 As Variant, par12 As Variant)

This function sends the custom event *eventName* to the server. The document specified by the parameter *cookie* is the source.

Function closeDocument (cookie As Long, everChanged As Boolean, changedAfterSave As Boolean)

This function sends the close event of the document specified by the parameter *cookie* to the server.

Function saveDocument (cookie As Long, changed As Boolean)

This function sends the save event of the document specified by the parameter *cookie* to the server.

**GuiCtrlTree**

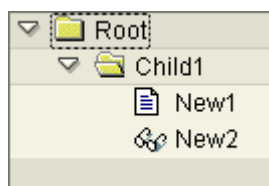
Type library: WdtTreeScripting.dll

Property subType As String

This property has the constant value "Tree".

Tree Control Types

The Tree Control supports three Tree Types:

1) Simple Tree**2) List Tree**

Objekte			
Dynpros			
0100 MUELLER	✓		Comment to Dynpro 100
0200 HARRYHIRSCH	✗		Comment to Dynpro 200
Programme			
SAPTROX1			Comment to SAPTROX1
SAPTRIXTROX			Comment to SAPTRIXTROX

The List Tree can have a Header

Hierarchy Header		List Header	
Objekte			
Dynpros			
Mask 1	✓	0100 MUELLER	Comment to Dy
Mask 2	✗	0200 HARRYHIRSCH	Comment to Dy
Programme			

3) Column Tree

Hierarchy & Header	Column2	Column3
Root Col. 1	Root Col. 2	Root Col. 3
Child1 Col. 1	Child1 Col. 2	Child1 Col. 3
New1 Col. 1		New1 Col. 3
New2 Col. 1	New2 Col. 2	New2 Col. 3

Tree Control Selection

The selection behaviour of a Tree Control instance is set once at the time of creation.

Node Selection Mode:

SingleNodeSelection: Only one node can be selected.

MultipleNodeSelection: Several nodes may be selected.

Airline/Flug-N...	Preis	W..	Fl.-Typ
64	3.595,83 U...		Note
AZ	57.556.450 ITL		Note
555	2.521.414 ITL		Note
788	18.672.115 ITL		Note
789	18.672.115 ITL		Note

Item Selection

If Item Selection is enabled, then just one item can be selected. Otherwise, only complete nodes can be selected. Item Selection is not available for the Simple Tree.

Airline/Flug-N...	Preis	W..	Fl.-Typ
▶ 64	3.595,83	U...	Note
▼ AZ	57.556.450	ITL	Note
▶ 555	2.521.414	ITL	Note
▶ 788	18.672.115	ITL	Note
▶ 789	18.672.115	ITL	Note

Column Selection

If Column Selection is enabled, one or more columns can be selected.

Airline/Flug-N...	Preis	W..	Fl.-Typ
▶ 64	3.595,83	U...	Note
▼ AZ	57.556.450	ITL	Note
▶ 555	2.521.414	ITL	Note
▶ 788	18.672.115	ITL	Note
▶ 789	18.672.115	ITL	Note

Node Selection, Item Selection and Column selection are mutually exclusive. For example, selecting an Item removes a Node Selection automatically.

Exceptions

Setting properties and calling methods can cause exceptions. Exceptions occur if the parameters describe non-existent objects or if the preconditions of the property or method are violated.

Property `selectedNode As String`

Precondition: Node Selection Mode is **SingleNodeSelection**

This is the key of the currently selected node. Selecting a node removes other selections (that is Column Selection and Item Selection).

Property `topNode As String`

This property influences the vertical scrolling of the Tree Control. *topNode* contains the key of the node that is located on the upper edge of the Tree Control. Setting a node x as top node is only possible if there are enough visible nodes below x to fill the display area of the Tree Control.

Property `hierarchyHeaderWidth As Long`

Precondition: Tree is a **Column Tree** or a **List Tree with Header**

The width of the Hierarchy Header in pixels.

Property `columnOrder As GuiCollection`²⁷

Preconditions:

- Tree is a **Column Tree**.

²⁷ See the 'Collection interfaces' chapter for a description of `GuiCollection`

- Column Order can be changed.

The property is used for working with a sequence of columns.

The name of each column in the Column Tree must occur exactly once.

Function `getTreeType As Long`

The returned number has the following meaning:

- 0 : Simple tree
- 1 : List tree
- 2 : Column tree

Function `getSelectionMode As Long`

- 0: Single Node
- 1: Multiple Node
- 2: Single Item
- 3: Multiple Item

Function `doubleClickNode (nodeKey As String)`

This function emulates double clicking a node.

Note: If Item Selection is enabled, double clicking a node can only be performed by double clicking on the Folder/Leaf Symbol of the node.

Function `defaultContextMenu`

This method requests a context menu for the whole Tree Control.

Function `nodeContextMenu (nodeKey As String)`

This method requests a context menu for a node.

Function `pressButton (nodeKey As String, itemName As String)`

Preconditions:

- Tree is a Column Tree or a List Tree.
- Item Selection is enabled.
- Item (NodeKey, ItemName) is a button.
- Button is enabled.

This method emulates pressing a button.

Function `changeCheckbox (nodeKey As String, itemName As String, checked As Boolean)`

Preconditions:

- Tree is a Column Tree or a List Tree.
- Item Selection is enabled.

- Item (NodeKey, ItemName) is a checkbox.
- Checkbox is enabled.

This method emulates changing a checkbox state.

Function pressHeader (headerName As String)

Precondition: Tree is a Column Tree or a List Tree with Header.

This method emulates clicking a header.

Function headerContextMenu (headerName As String)

Precondition: Tree is a Column Tree or a List Tree with Header.

This method requests a context menu for a header.

Function itemContextMenu (nodeKey As String, itemName As String)

Preconditions:

- Tree is a Column Tree or a List Tree.
- Item Selection is enabled.

This method requests a context menu for an item.

Function doubleClickItem (nodeKey As String, itemName As String)

Preconditions:

- Tree is a Column Tree or a List Tree.
- Item Selection is enabled.
- Item (NodeKey, ItemName) is a text.

This function emulates double clicking on a text item.

Function clickLink (nodeKey As String, itemName As String)

Preconditions:

- Tree is a Column Tree or a List Tree.
- Item Selection is enabled.
- Item (NodeKey, ItemName) is a link.

This function emulates triggering a link.

Function selectItem (nodeKey As String, itemName As String)

Preconditions:

- Tree is a Column Tree or a List Tree.
- Item Selection is enabled.

This function emulates the selection of an item. This selection removes all other selections.

Function selectNode (nodeKey As String)**Preconditions:**

- Node Selection Mode is *MultipleNodeSelection*.

The node with the key *nodeKey* is added to the Node Selection.

Function unselectNode (nodeKey As String)**Preconditions:**

- Node Selection Mode is *MultipleNodeSelection*.

The node with the key *nodeKey* is removed from the Node Selection.

Function unselectAll

All selections are removed.

Function collapseNode (nodeKey As String)

This function closes the node with the key *nodeKey*.

Function expandNode (nodeKey As String)

This function expands the node with the key *nodeKey*.

Function setColumnWidth (columnName As String, width As Long)

Precondition: Tree is a Column Tree.

This function sets the width of a column in pixels.

Function selectColumn (columnName As String)**Preconditions:**

- Tree is a Column Tree.
- Column Selection is enabled.

This function adds a column to the column selection. A node or item selection is removed.

Function unselectColumn (columnName As String)**Preconditions:**

- Tree is a Column Tree.
- Column Selection is enabled.

This function removes a column from the column selection.

Function pressKey (key As String)

This method emulates pressing a key.

Possible values for Key are: F1, F4, Delete, Insert, Enter, Cut, Copy, and Paste

Function ensureVisibleHorizontalItem (nodeKey As String, itemName As String)

Precondition: Tree is a Column Tree or a List Tree.

This function scrolls the Tree horizontally until the Item is visible.

Function getNodeKeyByPath (path As String) As String

Key of the node specified by the given path description.

Function getNodeTextByPath (path As String) As String

The text of a node defined by the given path is returned.

Function getNodeTextByKey (key As String) As String

This function returns the text of the node specified by the given key.

Function getItemText (key As String, name As String)

This function returns the text of the item specified by the *key* and *name* parameters.

Function getNodeChildrenCountByPath (path As String) As Integer

This function returns the number of children of the node given by the *path* parameter.

Function getNodesCol As GuiCollection

The collection contains the node keys of all the nodes in the tree.

Function getSubNodesCol (key as String) As GuiCollection

Collection of the keys of all subnodes of the node specified by the given key.

Function getColumnHeaders As GuiCollection

Collection of the titles of the columns.

Function getColumnNames As GuiCollection

Returns a collection of the column names.

Function getColumnCol (columnName As String) As GuiCollection

The keys of all the items in the given column.

Function getParent (key as String) As String

Key of the parent node of the node specified by the given key.

Function getNodePathByKey (key as String) As String

Given a node key, the path is retrieved (e.g. 2\1\2).

Function getCheckBoxState (nodeKey As String, itemName As String) As Long

Retrieves the CheckBox state (1 = Checked, 0 = Unchecked).

Function getItemType (nodeKey As String, itemName As String) As Long

Retrieves the column tree item type:

```
trvTreeStructureHierarchy = 0,  
trvTreeStructureImage = 1,  
trvTreeStructureText = 2,  
trvTreeStructureBool = 3,  
trvTreeStructureButton = 4,  
trvTreeStructureLink = 5
```

**GuiCtrlHTMLViewer**

Type library: SapHtmlScripting.dll

Property subType As String

This property has the constant value "HTMLViewer"

Function sapEvent (frameName As String, postData As String, url As String)

This function submits an HTML form to the backend.

frameName: this is the name of the frame in which the HTML form that has been submitted lives.

postData: contains the form data when a submit is made using the POST method.

url: This is the URL which is submitted to the backend. The protocol name for the URL string is "sapevent:". This is followed by the name of the event as defined in the Action Property of the HTML form which is called.

If the form is to be submitted using the GET method, the data is appended to the event name in the usual http URL fashion, for example:

```
sapEvent("Frame1","", "sapevent:SUBMIT_FORM_AS_GET_METHOD?FirstName=John&LastName=Smith");
```

In this case, *postData* is always an empty string.

If the form is to be submitted using the POST method, the data is transported in the *postData* parameter:

```
sapEvent("Frame1","FirstName=John&LastName=Smith","sapevent:SUBMIT_FORM_AS_POST_METHOD");
```

Function contextMenu

Calling *contextMenu* emulates the context menu request. Note that this function applies only to context menus provided by the backend, not to the local context menu, which is generated by the HTML control.



GuiSplitterShell

Type library: SapSplitScripting.dll

Property subType As String

This property has the constant value "Splitter"



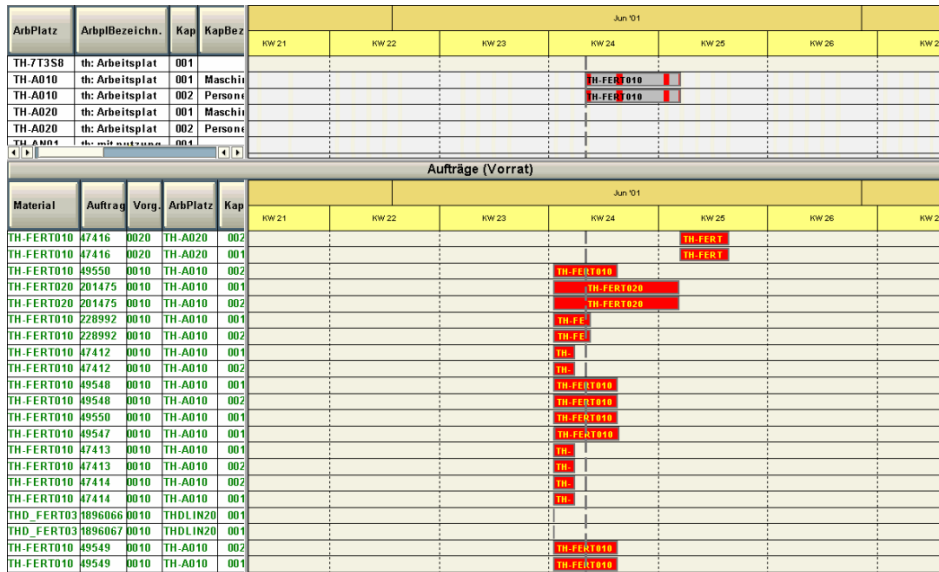
Graphics Controls

All of the classes in this chapter extend the *GuiShell* class.



GuiCtrlBarChart (Under Construction)

The bar chart control is a powerful tool to display and modify time scale diagrams.



Type library: sapbarcScripting.dll

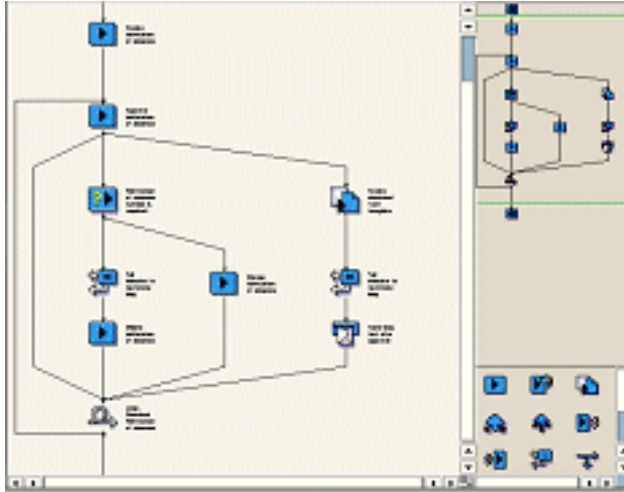
Property subType As String

This property has the constant value "sapbarc"



GuiCtrlNetChart

The net chart control is a powerful tool to display and modify entity relationship diagrams.



Type library: sapnetzScripting.dll

Property subType As String

This property has the constant value "sapnetz"

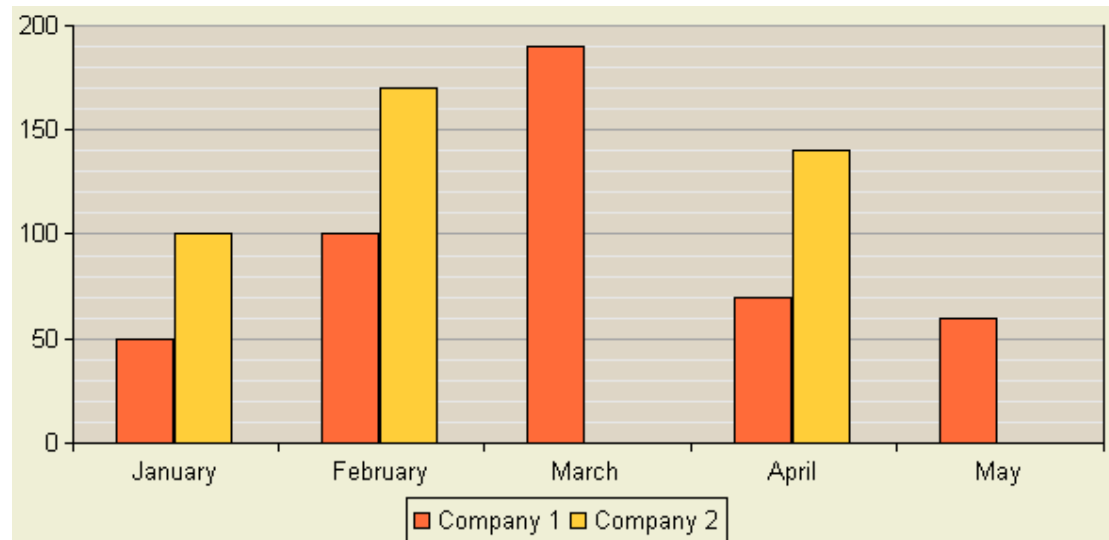
Function SendData

This function emulates the output of each action triggered at the control side. The result of the action is sent to the backend.

It's currently not possible to select – deselect single objects at the client-side and to replay/script these "local" actions.



GuiCtrlChart



Type library: chartScripting.dll

Property subType As String

This property has the constant value "Chart".

Function valueChange (series As Long, point As Long, xValue As String, yValue As String, dataChange As Boolean, id As String, zValue As String, changeFlag As long)

The parameters have the following meaning:

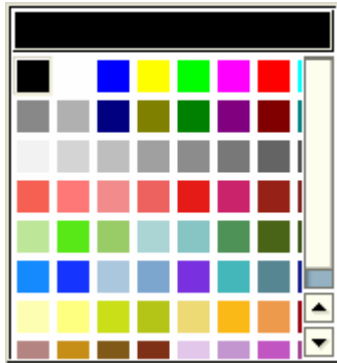
- series: Number of the data set
- point: Number of the point within the data set
- xValue: The new x value
- yValue: The new y value
- dataChange: True, if a value was changed using the DataPoint property page
- id: Object id within the framework. May be used instead of the pair series/point.
- zValue: The new z value
- changeFlag: Bitset that specifies which data were changed:
 - Bit 0: x value
 - Bit 1: y value
 - Bit 2: x value, and the new value is a point in time
 - Bit 3: y value, and the new value is a point in time
 - Bit 4: z value

If the new value is a point in time, it should be set using a string of the format "mm/dd/yyyy hh:mm:ss".



GuiCtrlColorSelector

The color selector displays a set of colors for selection.



Type library: sapselScripting.dll

Property subType As String

This property has the constant value "ColorSelector"

Function ChangeSelection (Index As Long)

This function emulates the user's selection of the color at the given index position.



Application Controls

All of the classes in this chapter extend the *GuiShell* class.



APOGrid (Under construction)



	Eir ...	W 07.2003	W 08.2003	W 09.2003	W 10.2003
Consensl ...	ST				
Corrected ...	ST				
Corrected ...	ST				
Prognose	ST	32.891	4.213.421	321	334
Manual Ac ...	ST				
Revenue	EUR				
Sales For ...	ST				
Sales His ...	ST				
VMI Promc ...	ST				

Type library: ApoGridScripting.dll

Property subType As String

This property has the constant value "ApoGrid".

The columns and rows are identified by their position starting with zero:

$0 \leq \text{row} < \text{RowCount}$

$0 \leq \text{column} < \text{ColumnCount}$

After a drill-down the rows are re-numbered so that the row number of any given row may change.

Scrolling horizontally does not affect the number of a column.

Method `getCellValue (column As Integer, row As Integer) As String`

Get the content of the cell at the specified column and row position.

Method `setCellValue (column As Integer, row As Integer, value As String)`

Set the value at the given position. This will send the text to the backend, as if the user had left the cell using TAB.

Method `paste (cellValues As GuiCollection, columnCount As Integer)`

Paste the values from the collection to the currently selected cell(s). The layout of the data in the collection is defined by *columnCount*. If more than one cell is selected, then the collection must only contain one value.

Example 1:

Value in clipboard: "V1"

Selected cells in grid:

Grid after paste operation:

	V1	V1		
	V1	V1		

Example 2:

Values in clipboard:

V1	V2	V3
V4	V5	V6

Selected cell in grid:

Grid after paste operation:

	V1	V2	V3	
	V4	V5	V6	

Method cut

The currently selected cell(s) are marked for cutting. They will be cleared during the next call to *Paste* within the APO Grid or Microsoft Excel.

Method cancelCut

Cancel the previous cut operation.

Method pressEnter

Validate a cell after setting the cell value.

Method selectRow (row As Integer)

Select the entire row.

Method `deselectRow` (row As Integer)

Unselect the entire row.

Property `selectedRows` As String (Read Only)

String that identifies the selected rows. Items are separated by “,”. Ranges are represented using a “-”.

Example: The rows 0 to 5, row 10 and the rows 20 to 100 are selected. The result string is “0-5,10,20-100”.

Method `selectColumn` (column As Integer)

Select the entire column.

Method `deselectColumn` (column As Integer)

Unselect the entire column.

Property `selectedColumns` As String (Read Only)

String that identifies the selected columns. Items are separated by “,”. Ranges are represented using a “-”.

Example: The columns 0 to 5, column 10 and the columns 20 to 100 are selected. The result string is “0-5,10,20-100”.

Method `selectCell` (column As Integer, row As Integer)

Select the cell at specified column and row position.

Method `deselectCell` (column As Integer, row As Integer)

Unselect the cell at specified column and row position.

Property `selectedCells` As GuiCollection (Read Only)

The collection contains items of type string. Each item represents a selected cell. The row and column is separated by “,”.

Example: The rectangle from (column 2, row 3) up to (column 4, row 4) is selected. Now the GuiCollection returned by "selectedCells" contains the following 6 items:

2,3

3,3

4,3

2,4

3,4

4,4.

Method `selectAll`

Select all entries in the grid.

Method clearSelection

Deselect all selected columns, rows or cells.

Property firstVisibleRow As Integer

Get or set the first visible row that is not fixed. May cause scrolling.

Property firstVisibleColumn As Integer

Get or set the first visible column that is not fixed. May cause scrolling.

Property visibleColumnCount As Integer

The number of columns that is displayed and not fixed.

Property columnCount As Integer (Read Only)

Get the column count of the grid.

Property rowCount As Integer (Read Only)

Get the row count of the grid.

Property fixedColumnsLeft As Integer (Read Only)

The number of fixed columns at the left side of the grid.

Property fixedColumnsRight As Integer (Read Only)

The number of fixed columns at the right side of the grid.

Property fixedRowsTop As Integer (Read Only)

The number of fixed rows at the top of the grid.

Property fixedRowsBottom As Integer (Read Only)

The number of the fixed rows at the bottom of the grid.

Property visibleRowCount As Integer (Read only)

Retrieves the number of visible rows of the grid. Partially visible rows are included in this value. Fixed rows on the top and on the bottom are not included.

Method getCellChangeable (column As Integer, row As Integer) As Boolean

Returns if the cell at the specified column and row position is "active", i.e. text can be entered.

Method getCellFormat (column As Integer, row As Integer) As String

Get the format of the cell at the specified column and row position.

Possible return values are:

- unset, when no format is specified
- "string", for free value format
- "integer", for integer value format
- "float.1", for float value format with 1 decimal digit.
- "float.2", for float value format with 2 decimal digits.
- "float.3", for float value format with 3 decimal digits.
- "float.4", for float value format with 4 decimal digits.
- "float.5", for float value format with 5 decimal digits.
- "float.6", for float value format with 6 decimal digits.
- "dateMMDDYYYY", for date value format with 2 digits for month, 2 digits for day and 4 digits for year.
- "dateDDMMYYYY", for date value format with 2 digits for day, 2 digits for month and 4 digits for year.

Method getCellTooltip (column As Integer, row As Integer) As String

Get the tooltip of the cell at the specified column and row position.

Method doubleClickCell (column As Integer, row As Integer)

Emulate a double click on the given cell.

Method contextMenu (column As Integer, row As Integer)

Display the context menu on the given cell.



Utility Classes



GuiSessionInfo

GuiSessionInfo is a member of all *GuiSession* objects. It makes available technical information about the session. Some of its properties are displayed in the right corner of the SAP GUI status line.

Property `messageServer` As String (Read only)

The message server information is available only if the session belongs to a connection which was started using load balancing.

Property `group` As String (Read only)

The login group information is available only if the session belongs to a connection which was started using load balancing.

Property `systemName` As String (Read only)

This is the name of the SAP system.

Property `systemNumber` As Long (Read only)

The system number is set only if the session belongs to a connection that was started without load balancing, by specifying an application server.

Property `applicationServer` As String (Read only)

The name of the application server is set only if the session belongs to a connection that was started without load balancing, by specifying an application server.

Property `sessionNumber` As Long (Read only)

The number of the session is also displayed in SAP GUI on the statusbar.

Property `client` As String (Read only)

The client selected on the login screen.

Property `user` As String (Read only)

The SAP name of the user logged into the system.

Property `language` As String (Read only)

The language specified on the login screen.

Property codepage As Long (Read only)

The codepage specified in SAPlogon in the properties of the connection.

Property transaction As String (Read only)

The transaction that is currently being executed.

Property program As String (Read only)

The name of the source program that is currently being executed.

Property screenNumber As Long (Read only)

The number of the screen currently displayed.

Property responseTime As Long (Read only)

This is the time that is spent on network communication from the moment data are sent to the server to the moment the server response arrives. The unit is milliseconds.

Property interpretationTime As Long (Read only)

The interpretation time begins after the data have arrived from the server. It comprises the parsing of the data and distribution to the SAP GUI elements. The unit is milliseconds.

Property flushes As Long (Read only)

The property *flushes* counts the number of flushes in the automation queue during server communication.

Property roundTrips As Long (Read only)

Before SAP GUI sends data to the server it locks the user interface. In many cases it will not unlock the interface once data arrive from the server, but instead will send a new request to the server immediately. Controls in particular use this technology to load the data they need for visualization. The count of these token switches between SAP GUI and the server is the *roundTrips* property.

**GuiUtils**

The *GuiUtils* class provides a script with basic file and message functionality. This is useful when writing a script in JavaScript, for example, as this language does not come with built-in file access functions.

Property MESSAGE_TYPE_INFORMATION As Long (Read only) == 0

Constant value to be used when calling the *showMessageBox* method. Using this value will display the letter 'i' as the message box icon.

Property MESSAGE_TYPE_QUESTION As Long (Read only) == 1

Constant value to be used when calling the *showMessageBox* method. Using this value will display a question mark as the message box icon.

Property MESSAGE_TYPE_WARNING As Long (Read only) == 2

Constant value to be used when calling the *showMessageBox* method. Using this value will display an exclamation mark as the message box icon.

Property MESSAGE_TYPE_ERROR As Long (Read only) == 3

Constant value to be used when calling the *showMessageBox* method. Using this value will display a stop sign as the message box icon.

Property MESSAGE_TYPE_PLAIN As Long (Read only) == 4

Constant value to be used when calling the *showMessageBox* method. Using this value will display no message box icon.

Property MESSAGE_OPTION_OK As Long (Read only) == 0

Constant value to be used when calling the *showMessageBox* method. Using this value will display an 'OK' button only.

Property MESSAGE_OPTION_YESNO As Long (Read only) == 1

Constant value to be used when calling the *showMessageBox* method. Using this value will display a 'Yes' button and a 'No' button.

Property MESSAGE_OPTION_OKCANCEL As Long (Read only) == 2

Constant value to be used when calling the *showMessageBox* method. Using this value will display an 'OK' button and a 'Cancel' button.

Property MESSAGE_RESULT_CANCEL As Long (Read only) == 0

Constant value to be used as a return value by the *showMessageBox* method. This value is returned when the 'Cancel' button has been pressed.

Property MESSAGE_RESULT_OK As Long (Read only) == 1

Constant value to be used as a return value by the *showMessageBox* method. This value is returned when the 'OK' button has been pressed.

Property MESSAGE_RESULT_YES As Long (Read only) == 2

Constant value to be used as a return value by the *showMessageBox* method. This value is returned when the 'Yes' button has been pressed.

Property MESSAGE_RESULT_NO As Long (Read only) == 3

Constant value to be used as a return value by the *showMessageBox* method. This value is returned when the 'No' button has been pressed.

Function showMessageBox (title As String, text As String, msgType As Long, msgOption As Long) As Long

Shows a message box. *title* and *text* set the title and text of the message box. *msgIcon* sets the icon to be used for the message box and should be set to one of the *MESSAGE_TYPE_** constants described above. *msgType* sets the buttons available on the message box and should be set to one of the *MESSAGE_OPTION** constants. The return value will be one of the *MESSAGE_RESULT_** values.

Function openFile (name As String) As Long

name is the name of the text file to be created. For security reasons this name must not contain any path information. The file will be created in the SapWorkDir for SAP GUI for Windows and in the file output directory in SAP GUI for Java. The return value is a handle to the file and is required for the methods which follow.

Function write (handle As Long, text As String)**Function writeLine (handle As Long, text As String)**

These functions write text to an open file. The *writeLine* function adds a new line character to the text.

Function closeFile (handle As Long)

This function closes a file that was opened using *openFile*.

**GuiScrollbar**

The *GuiScrollbar* class is a utility class used for example in *GuiScrollContainer* or *GuiTableControl*.

Property maximum As Long (Read only)

This is the maximum position of the scrollbar.

Property minimum As Long (Read only)

This is the minimum position of the scrollbar.

Property position As Long

The *position* of the thumb of the scrollbar can be set to values from *minimum* to *maximum*.

Property pageSize As Long (Read only)

When the user scrolls down a page, *position* will be increased by the value of *pageSize*.



Collections



GuiComponentCollection²⁸

The *GuiComponentCollection* is used for collections elements such as the *children* property of containers. Each element of the collection is an extension of *GuiComponent*.

Property type As String (Read only)

The value is 'GuiCollection'.

Property typeAsNumber As Long (Read only)

The value is 120.

Property count As Long (Read only)



The number of elements in the collection. This property is used implicitly from Visual Basic applications.

Property length As Long (Read only)

The number of elements in the collection.

Property newEnum As Object (Read only)



This property is used implicitly from Visual Basic applications.

Function elementAt (index As Long) As GuiComponent

This function returns the member in the collection at position *index*, where *index* may range from 0 to *count*-1. If no member can be found for the given index, the exception *Gui_Err_Enumerator_Index* (614) is raised.

Function item (index As Long) As GuiComponent



This function returns the member in the collection at position *index*, where *index* may range from 0 to *count*-1. It has been added for compatibility with Microsoft Visual Basic collections. If no member can be found for the given index the exception *Gui_Err_Enumerator_Index* (614) is raised.

²⁸ In SAP GUI for Java using JavaScript *GuiCollection* is used in place of *GuiComponentCollection*.



GuiCollection

GuiCollection is similar to *GuiComponentCollection*, but its members are not necessarily extensions of *GuiComponent*.

It can be used to pass a collection as a parameter to functions of scriptable objects. An object of this class is created by calling the *CreateGuiCollection* function of the *GuiApplication*.

Property type As String (Read only)

The value is 'GuiCollection'.

Property typeAsNumber As Long (Read only)

The value is 120.



Property count As Long (Read only)

The number of elements in the collection. This property has been added for compatibility with Microsoft Visual Basic collections.

Property length As Long (Read only)

The number of elements in the collection.



Property newEnum As Object (Read only)

It has been added for compatibility with Microsoft Visual Basic collections.

Function elementAt (Index As Long) As Any

This function returns the member in the collection at position *index*, where *index* may range from 0 to *count*-1. If no member can be found for the given index, an exception is raised.



Function item (Index As Long) As Any

This function returns the member in the collection at position *index*, where *index* may range from 0 to *count*-1. It has been added for compatibility with Microsoft Visual Basic collections. If no member can be found for the given index, an exception is raised.

Function add (item As Any)

After a *GuiCollection* has been created, items can be added by calling the *add* function.



Platform and Language Dependencies



Accessing the Runtime Hierarchy



SAP GUI for Windows

At runtime, all SAP GUI windows started from the same SAPlogon process become part of a unified object hierarchy.

The hierarchy can be accessed using different approaches. An application can either attach itself to a running SAPlogon process or create a new SAP GUI instance.

Attaching to a running SAPlogon process

When SAPlogon is started, it registers a surrogate object *SapGuiAuto* in the Running Object Table (ROT) as "SAPGUI". This object returns the interface to the scripting component's highest level object using the *GetScriptingEngine* function.

```
Set SapGuiAuto = GetObject("SAPGUI")
Set Application = SapGuiAuto.GetScriptingEngine
```

If no connections are open at this point, then the application object is the only available object. As soon as the user either connects to an SAP system or a script calls the *openConnection* or *openConnectionByConnectionString* functions, the connection becomes accessible as a child of the application.

Creating a SAP GUI instance using the scripting component

When no SAPlogon is available, the scripting component can be used to create a new instance of the *GuiApplication*. This instance is then not part of a SAPlogon process but rather of the process executing the script, so all connections are closed at the end of the script.

```
Set Application = CreateObject("Sapgui.ScriptingCtrl.1")
```



SAP GUI for Java

JavaScript Engine

The JavaScript Engine is fully integrated in the SAP GUI for the Java Environment (SAPGUI for Java) and therefore it is always available for user scripting purposes during the SAP GUI for Java runtime. To use the JavaScript Engine in a *local scripting context* one has to open the scripting window from the “?” menu of the running session. The available predefined objects are *application*, *connection*, *session*, *window*, and *userarea*. These objects have to be queried to access the SAP GUI for Java component hierarchy. To use the JavaScript Engine in a *global scripting context* the scripting window is available from the “?” menu of the logon window. The only predefined object of the SAP GUI for Java component hierarchy is *application*. For more information about local and global scripting windows refer to the “Scripting Hosts for SAP GUI” documentation.

AppleScript Engine

The AppleScript support is available in SAPGUI for Java on Mac OS X. To work with AppleScript you need to use the Script Editor application. This application is installed with the Mac OS X system and can be found in the “Applications” folder and “Apple Script” sub-folder. In your script there is only one predefined object – *GuiApplication*. Mac OS X developers can use the AppleScript support for communication with the SAPGUI for Java application. SAPGUI is a scriptable application. If you click the “RECORD” button in the Script Editor application, all user action in SAPGUI for Java will be stored in the recording window of Script Editor.



Event Handlers in JavaScript

SAP GUI for Java and SAP GUI for Windows using JScript define global event handlers – functions, which are called, if SAP GUI fires certain types of events.

The following describes all available event handlers. To use the event handlers, you have to implement the functions using exactly the same syntax and parameters for each of them as shown below:

Function onStartRequest (session)

This function is called before the session is locked during server communication. At this point the user input can be checked before it is sent to the server. It is not possible to prevent the server communication from this event handler. The following script shows an example of using this event handler:

```
//This function is called before server communication
function onStartRequest (session)
{
    var text = userarea.findById("txtF1").getText();

    //put text into the textfield before sending the data to the server
    if (text == "")
    {
        userarea.findById("txtF1").text = "before server communication";
    }
}
```

Function onEndRequest (session)

This function is called immediately after the session is unlocked after server communication. The following script shows an example of using this event handler (which is similar to the function *onStartRequest*):

```
//This function is called after server communication
function onEndRequest (session)
{
    var text = userarea.findById("txtF1").getText();

    //do something with text
    . . .
}
```

Function onSessionCreate (session)

This function is called after a new session is created. The session is visualized in a new main window.

```
function onSessionCreate (session)
{
    //show messagebox with session id:
    window.showMessageDialog("Information", "Session "+
        session.getId() + " is created",
        application.utils.MESSAGE_TYPE_PLAIN,
        application.utils.MESSAGE_OPTION_OK);
}
```

Function onSessionDelete (session)

This function is called after the session is destroyed. The main window, which visualized this session, is closed.

```
function onSessionDelete (session)
{
    //show messagebox with session id:
    window.showMessageDialog("Information", "Session "+
        session.getId() + " is deleted",
        application.utils.MESSAGE_TYPE_PLAIN,
        application.utils.MESSAGE_OPTION_OK);
}
```

Function onError (session, errorid, desc1, desc2, desc3, desc4)

This function is called if a runtime error occurs during the execution of a script.

```
function onError (session, errorid, desc1, desc2, desc3, desc4)
{
```



```
    window.showMessageDialog("Error", "Runtime error has occurred in "+
        "the session "+session.getId(),
        application.utils.MESSAGE_TYPE_ERROR,
        application.utils.MESSAGE_OPTION_OK);
}
```



SAP GUI for Java using AppleScript

The event-handler function is not available in AppleScript, The current AppleScript model does not provide any mechanism for writing an external event-handler function in your script.



Executing Scripts



SAP GUI for Windows

The SAP GUI scripting interface exposes COM automation interfaces. It does not come with a built-in script engine.

The examples in this document were written in Microsoft Visual Basic Script and can be executed using the Microsoft Windows Script Host, which is available for download from the Microsoft web pages. Alternatively, the Microsoft Script Control can be instantiated in external applications to run scripts that access the scripting interface of SAP GUI.



Visual Basic Script can be misused to create 'Virus' like scripts, therefore the installation of the script host may not be accepted under all circumstances.

Of course, the interface does not necessarily need to be accessed from textual scripts. Any programming language capable of invoking automation calls can be used.

A script can also be dropped onto a SAP GUI window. SAP GUI will then execute this script using the Microsoft Script Control. In this case, the object *session* of *GuiSession* type can be used in the script. It represents the session to which the drop target window belongs.



Appendix



Object Life Time Considerations

Each time data is sent and subsequently received from the server all elements of the SAP GUI window are invalidated and recreated. A script that references some of these elements after they have become invalid will fail.

It is therefore recommended that a script should only keep references to *GuiFrameWindow* objects or objects higher in the hierarchy than *GuiFrameWindow* for a limited period of time. These references will remain valid and can be used to find all other elements using *findById*.



Technical Background - Connection Strings

Connection String is a technical term used within SAP GUI. A connection string describes a connection address for a destination, e.g. an SAP system's application server, similar to an Internet URL describes a location for a web page.

Simple Connection Strings

In its simplest form, a connection string contains an IP address and a port number. This information is sufficient for SAP GUI to open a direct TCP connection to a destination, e.g. an application server. IP address and port number are marked with the prefixes '/H/' (for host) and '/S/' (for service). Note that the port number for an SAP application server is by convention 3200 plus the two-digit SAP system number.

Example for a simple connection string with an application server's IP address (172.16.64.17) and port number (3200):

```
/H/172.16.64.17/S/3200
```

If your network environment supports DNS (Domain Name Services), a hostname can be used instead of the IP address in all kinds of connection strings. (This requires a correct DNS configuration on the client, e.g. via the hosts file).

Example with an application server's hostname (iwdf8997.wdf.sap-ag.de) and port number (3200):

```
/H/iwdf8997.wdf.sap-ag.de/S/3200
```

If your network environment supports symbolic service names for well-known ports, the symbolic service name can be used instead of the port number in all kinds of connection strings. (This requires a correct service configuration on the client, e.g. in the services file). Note that SAP application server ports are by convention named 'sapdp<SID>', where <SID> is the SAP system id.

Example with host name (iwdf8997.wdf.sap-ag.de) and symbolic service name (sapdpIWD):

```
/H/iwdf8997.wdf.sap-ag.de/S/sapdpIWD
```

Simple connection strings need not be resolved by the SAP GUI application. Resolution of host names and symbolic service names is done by the operating system's network layer.

SAP Routers

In a WAN (Wide Area Network) environment, SAP routers are used to make connections to remote SAP systems that cannot be reached with a direct TCP connection. Passwords may be used for each SAP router to control access.

In order to make a connection, the client is responsible for providing the complete route to the destination, possibly including a chain of several SAP routers. Path information is not provided by the routers. (Strictly speaking, a SAP router is actually better described as an application level proxy with password capabilities and strict source routing).

The address for each router is specified by a simple connection string (with the router's host name and port number), optionally followed by '/P/' and the router password. The path from the current location to the destination is described by concatenating all router addresses, followed by the address of the destination SAP system. Thus, a connection string with SAP routers generally has the form <router 1><router 2>...<router n><destination>.

Example with two routers (gate.acme.com, port 3299, and gate.sap.com, port 3298), the first using a password (secret), for a connection to the application server iwdf8997.sap.com, port 3200):

```
/H/gate.acme.com/S/3299/P/secret/H/gate.sap.com/S/3298/H/iwdf8997.sap
.com/S/3200
<----- 1st router -----><---- 2nd router ----><-----
app_server ----->
```

Connection strings including SAP routers are passed to SAP GUI's communication layer and resolved step by step by the routers on the path. If host names and symbolic service names are used, each router must have access to correct network configuration information to resolve them.

Message Servers and Logon Groups

For load balancing purposes, application servers from one SAP system are usually configured in logon groups, where each group serves a particular kind of user. The application servers in each group are assigned to users by a least-heavily-loaded strategy. This load balancing is done by message servers. Each SAP system has exactly one message server, which can be reached via TCP on a specific message server port.

Care should be taken that the application server's port number is not confused with the message server's port number. Although the message server's host name may in small installations often be identical to the hostname of an application server, the port number is always different. Symbolic service names for message servers by convention have the form 'sapms<SID>', where <SID> is the SAP system id.

Message server and group information can be used to address a SAP system in a connection string. The address of the message server is specified as a combination of message server host name, message server port and group name. This information is marked with the prefixes '/M/' (message server host name), '/S/' (message server port) and '/G/' (logon group).

Example with message server (hostname alrmain, port number 4253) and logon group (SPACE):

```
/M/alrmain.wdf.sap-ag.de/S/4253/G/SPACE
```

Connection strings with message servers are resolved by SAP GUI by contacting the message server and retrieving the (simple) connection string of an application server for the specified group. This requires network access to the message server at the time the address is resolved.

SAP router connection strings may be used in combination with message server connection strings simply by specifying the router address before the message server address. The router is then used for contacting the message server as well as for contacting the resolved application server.

Symbolic System Names

The most user-friendly form of connection string addresses an SAP system only by its symbolic name (per convention, the system id) and the logon group name. These information are marked with the prefixes '/R/' (for the symbolic SAP system name) and '/G/' (for the logon group name).

Example with SAP system (ALR) and logon group (SPACE):

```
/R/ALR/G/SPACE
```

Connection strings with symbolic system names are resolved by SAP GUI by looking up the symbolic SAP system name in the Message Server List (a text file containing a mapping between symbolic system names and message server addresses) and replacing the /R/ part of the connection string with the resulting message server address. The result is a complete message server connection string which is then further resolved as explained above.

Formal syntax

For the technically interested reader, the following BNF grammar formally describes the syntax of connection strings:

```
<connection string> := [<router prefix>]<local>
<local>             := <simple>|<message server>|<symbolic>
<simple>            := "/H/"<host>"/S/"<service>
  <host>            := <hostname>|<ipaddr>
    <hostname>      := (any DNS hostname)
    <ipaddr>        := (any IP address, in dotted decimal form)
  <service>         := <servicename>|<port number>
    <servicename>   := (any IP service name)
    <port number>   := (any decimal number)
<messageserver>    := "/M/"<host>"/S/"<service>"/G/"<group>
  <group>           := (any ASCII string not containing '/')
<symbolic>         := "/R/"<system>"/G/"<group>
  <system>          := (any ASCII string not containing '/')
<router prefix>    := <router>*
  <router>          := "/H/"<host>"/S/"<service>["/P/"<password>]
  <password>       := (any ASCII string not containing '/')
```



DumpState Collection Format

The *DumpState* method returns a hierarchy of collections of type *GuiCollection*, which is three levels deep.

- The top (first) level collection contains a second level collection for every property that is to be dumped.
- The second level collection contains the complete information for one property. There is a third level collection for every sub-expression that might be required to access inner objects.

- Finally, the third level collection contains the OpCode, the property or method name, the parameter values and depending on the OpCode the return value to be checked.

The following OpCodes are used:

- GPR: Get property and compare return value.
- MR: Execute method and compare return value.
- GP: Get property and execute the next entry in the second level collection on the result.
- M: Execute the method and then execute the next entry in the second level collection on the result.

For example the calls

```
control.ItemCount = 42
```

```
control.GetItemValue(3, 2) = "MyText"
```

```
control.GetItem("2","3").Property1.MethodY("XYZ").Text = "ABC"
```

result in three entries of the top level collection:

First entry:

OpCode	Name	Parameter1/ Property-Value	Parameter2	Parameter...
GPR	ItemCount	42		

Second entry:

OpCode	Name	Parameter1	Parameter2	Parameter3/ Property- Value
MR	GetItemValue	3	2	MyText

Third entry:

OpCode	Name	Parameter1	Parameter2	Parameter...
M	GetItem	2	3	
GP	Property1			
M	MethodY	XYZ		
GPR	Text	ABC		

As you can see in this example, for calls that contain return values (MR, GPR) the last value in the third level collection is the return value.



Examples



Attaching to a SAPlogon instance

```
Function Attach
  Dim GuiAuto
  On Error Resume Next
  Set GuiAuto = GetObject("SAPGUI")
  On Error Goto 0

  If GuiAuto Is Nothing THEN
    MsgBox "Please start SAPlogon"
    Set Attach = Nothing
    Exit Function
  Else
    Dim oApp
    On Error Resume Next
    Set oApp = GuiAuto.GetScriptingEngine
    On Error Goto 0

    If oApp Is Nothing Then
      MsgBox "Scripting disabled"
      Set Attach = Nothing
      Exit Function
    Else
      Set Attach = oApp
    End If
  End If
End Function
```

This code fragment demonstrates how to obtain a reference to the *GuiApplication* object.



Traversing the SAP GUI Runtime Hierarchy

SAP GUI for Windows using Visual Basic

```
Sub ListChildren(oParent, oTS, nIdent)
  Dim strId
  Dim strName
  Dim strType

  If Not oParent Is Nothing Then
    strId = oParent.Id
    strName = oParent.Name
    strType = oParent.Type
  End If

  Dim i
  For i = 1 To nIdent
```

```

        oTS.Write("  ")
    Next

    oTS.WriteLine("Id = " + strId _
                  + " Name = " + strName _
                  + " Type = " + strType)

On Error Resume Next
Dim oChildren
Set oChildren = oParent.Children
On Error Goto 0

If IsObject(oChildren) Then
    Dim oChild
    For Each oChild in oChildren
        ListChildren oChild, oTS, nIdent + 1
    Next
End If

End Sub

Dim Application
Set Application = Attach

If Not Application Is Nothing Then
    Dim oFS
    Set oFS = CreateObject("Scripting.FileSystemObject")

    Dim oTS
    Set oTS = oFS.CreateTextFile("C:\dump.txt", True)
    oTS.WriteLine("Starting dump...")

    ListChildren Application, oTS, 1

    oTS.WriteLine("End of dump.")
    oTS.Close
    Set Application = Nothing
End If

```

SAP GUI for Java using Javascript

The runtime hierarchy can be traversed recursively as all objects support the basic *GuiComponent* properties. This code offers an insight into the available objects and their respective types. The *Attach* function was taken from the previous example.

```

dumpString = "";

// dump from session node
dump (session);

// print result string
application.utils.showMessageBox ("Result", dumpString,
    application.utils.MESSAGE_TYPE_PLAIN,
    application.utils.MESSAGE_OPTION_OK);

// this function dumps all children recursively
function dump (component)
{
    // concate component to dump string

```

```

        concat (component);

    try
    {
        var col = component.children;

        for (var x = 0; x < col.length; x++)
        {
            child = col.elementAt (x);
            dump (child);
        }
    }
    catch (e)
    {
        // component is a leaf...
    }
}

// this function concatenates a component id to the dump string
function concat (component)
{
    dumpString +=component.id + "\n";
}

```



Logging SAP GUI response times

This script uses the *startRequest* and *endRequest* events on the application and sessions to record performance data for up to 5 sessions in parallel.

Option Explicit

```

Dim GuiAuto
Set GuiAuto = GetObject("SAPGUI")

Dim Application
Set Application = GuiAuto.GetScriptingEngine

Dim arrSessions(5, 6)

Dim cSessions
cSessions = 0

Dim i
i = 0

Dim Connection
For Each Connection In Application.Children
    IF Connection.DisabledByServer = FALSE Then
        Dim Session
        For Each Session In Connection.Children
            If cSessions < 5 Then
                Dim SessionInfo
                Set SessionInfo = Session.Info
                cSessions = cSessions + 1
                ' Systemname
                arrSessions(cSessions - 1, 0) =
                    SessionInfo.SystemName
                ' Request count
            }
        }
    }
}

```



```

arrSessions(cSessions - 1, 1) = 0
' Roundtrips
arrSessions(cSessions - 1, 2) = 0
' Flushes
arrSessions(cSessions - 1, 3) = 0
' Accumulated ResponseTime
arrSessions(cSessions - 1, 4) = 0
' Session Object
Set arrSessions(cSessions - 1, 5) = Session
WScript.ConnectObject Session, "Session_"
' Session Id
arrSessions(cSessions - 1, 6) = Session.Id
End If
Next
END IF
Next

MsgBox "Stop listening", vbOkOnly, "Listening to " & CStr(cSessions)
& " sessions..."

Dim strOutput

For i = 0 To cSessions - 1
  If arrSessions(i, 1) = 0 Then
    arrSessions(i, 1) = 1
  End If

  strOutput = strOutput & _
    "System: " & arrSessions(i, 0) & _
    " Roundtrips: " & CStr(arrSessions(i, 2)) & _
    " Flushes: " & CStr(arrSessions(i, 3)) & _
    " Acc. Response Time: " & CStr(arrSessions(i, 4)) & _
    " Avg. Response Time: " & CStr(arrSessions(i, 4) /
      arrSessions(i, 1)) & _
    vbCrLf
Next

MsgBox strOutput, vbOkOnly, "Results"

Sub Session_EndRequest(Session)
  For i = 0 To cSessions - 1
    If arrSessions(i, 6) = Session.Id Then
      arrSessions(i, 1) = arrSessions(i, 1) + 1
      arrSessions(i, 2) = arrSessions(i, 2) +
        Session.Info.RoundTrips
      arrSessions(i, 3) = arrSessions(i, 3) + Session.Info.Flushes
      arrSessions(i, 4) = arrSessions(i, 4) +
        (Session.Info.ResponseTime / 1000)
    End If
  Next
End Sub

```



Recording user interaction

This script records all modifications currently made to a screen and dumps them to a file on the desktop, which can later be used to redo the modifications.

The example below makes use of the *session* object, which is only available if a script has been dropped onto a SAP GUI window. The previous examples showed how to get a reference to a session without using *session*.

```
Dim oFS
Dim oTS

Function GetCallPrefix(Control)
    Dim test

    If Not Control Is Nothing Then
        Dim strId
        strId = "N/A"
        On Error Resume Next
        strId = Control.Id
        On Error GoTo 0

        Dim vPos
        vPos = InStr(strId, "ses[")
        If vPos > 0 Then
            vPos = InStr(vPos, strId, "]")
            If vPos > 0 Then
                strId = Mid(strId, vPos + 2)
            End If
        End If

        test = "session.findById("""
        test = test + strId + """)."
    Else
        test = "session."
    End If
    GetCallPrefix = test
End Function

Sub Session_Change(Control, CommandArray)
    Dim callString
    Dim i
    Dim lower
    Dim PropObj
    Dim coll
    Dim collItem
    Dim ArrayIndex
    Dim MaxArrayIndex
    Dim elemArray
    Dim indexCommandArray

    ' first step: analyze if parameters are arrays
    ArrayIndex = 0
    MaxArrayIndex = 0
    For Each elemArray In CommandArray
        For i = (lower + 2) To UBound(elemArray)
            If VarType(elemArray(i)) = vbObject Then
                ArrayIndex = ArrayIndex + 1

                Set coll = elemArray(i)
                oTS.WriteLine("set coll" + CStr(ArrayIndex) + " =
                    Engine.CreateGuiCollection")

                For Each collItem In coll
```

```

        oTS.WriteLine("coll" + CStr(ArrayIndex) + ".Add
"" + -
        CStr(collItem) + """)
        Next
    End If
Next
Next

' second step generate method call
MaxArrayIndex = ArrayIndex
ArrayIndex = 0
callString = ""
indexCommandArray = -1

For Each elemArray In CommandArray
    indexCommandArray = indexCommandArray + 1

    lower = LBound(elemArray)
    If Len(callString) > 0 Then
        callString = callString + "."
    End If
    callString = callString + elemArray(lower + 1)

    Dim ParamValue
    For i = (lower + 2) To UBound(elemArray)
        If VarType(elemArray(i)) = vbBoolean Then
            If elemArray(i) = True Then
                ParamValue = ""True""
            Else
                ParamValue = ""False""
            End If
        Else
            If VarType(elemArray(i)) = vbObject Then
                ArrayIndex = ArrayIndex + 1
                ParamValue = "coll" + CStr(ArrayIndex)
            Else
                ParamValue = "" + CStr(elemArray(i)) + ""
            End If
        End If
    End If

    Select Case elemArray(lower)
        Case "M"
            If i = (lower + 2) Then
                If indexCommandArray < UBound(CommandArray)
                    Then
                        callString = callString + "("
                Else
                    callString = callString + " "
                End If
            Else
                callString = callString + ","
            End If
            callString = callString + ParamValue
            If (UBound(elemArray) = i) Then ' last parameter?
                If indexCommandArray < UBound(CommandArray)
                    Then
                        callString = callString + ")"
                    End If
            End If
        Case "SP"
            callString = callString + " = " + ParamValue
    End Select
End For

```

```

        Case "GP"
            ' no parameter for properties => JScript
        End Select
    Next
Next

callString = GetCallPrefix(Control) + callString
oTS.WriteLine(callString)

' third step: clear all arrays
For ArrayIndex = 1 To MaxArrayIndex
    oTS.WriteLine("set coll" + CStr(ArrayIndex) + " = nothing")
Next

End Sub

Set SapGuiAuto = GetObject("SAPGUI")
Set SapApplication = SapGuiAuto.GetScriptingEngine
Set SapSession = SapApplication.Children(0).Children(0)
WScript.ConnectObject SapSession, "Session_"

Set oFS = CreateObject("Scripting.FileSystemObject")

Set objShell = WScript.CreateObject("WScript.Shell")
strPath = objShell.SpecialFolders("Desktop")

Set oTS = oFS.CreateTextFile(strPath & "\sapsnap.vbs", True)
SapSession.Record = True

SapSession.Record = False

oTS.Close

Set SapSession = Nothing
Set SapApplication = Nothing

```



Pre-setting values in the SAP GUI input history

The input history data file is protected so that it can not be accessed from outside SAP GUI. However, it may be useful to pre-set the history database to a set of often-used values. The following script deletes all entries from the local history database and then adds new ones:

```

Dim Application
Set Application = Attach

If Not Application Is Nothing Then
    Dim bResult
    MsgBox "Drop history data", vbOkOnly, "History script"
    bResult = Application.DropHistory
    MsgBox "Add new entries", vbOkOnly, "History script"
' Write history for se37 text field
    bResult = Application.AddHistoryEntry ("RS38L-NAME", "Asterix")
    bResult = Application.AddHistoryEntry ("RS38L-NAME", "Obelix")
    bResult = Application.AddHistoryEntry ("RS38L-NAME", "Idefix")
End If

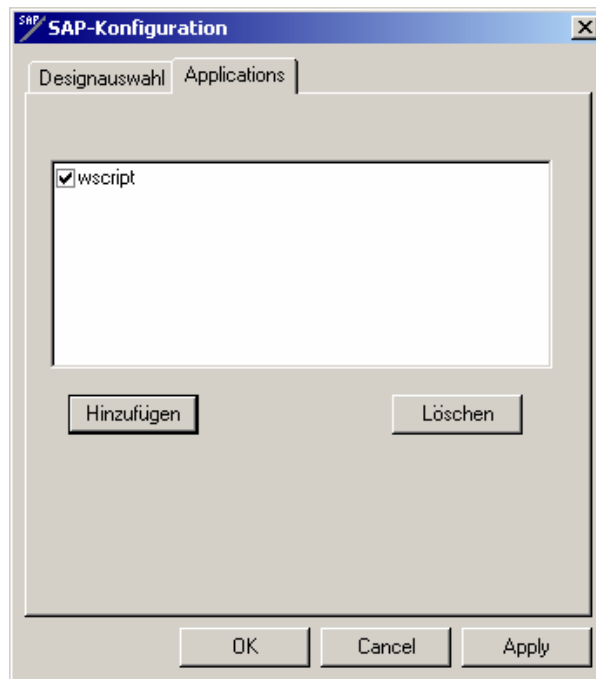
```

The function *Attach* was taken from the first example.



Why is New Visual Design not available if I start SAP GUI for Windows using CreateObject?

The New Visual Design is configured on a per process base. If you create a *GuiApplication* in your own process rather than attaching to a running SAPLogon, the name of your process's executable must be added to the list of applications in the SAP configuration dialog in the control panel.



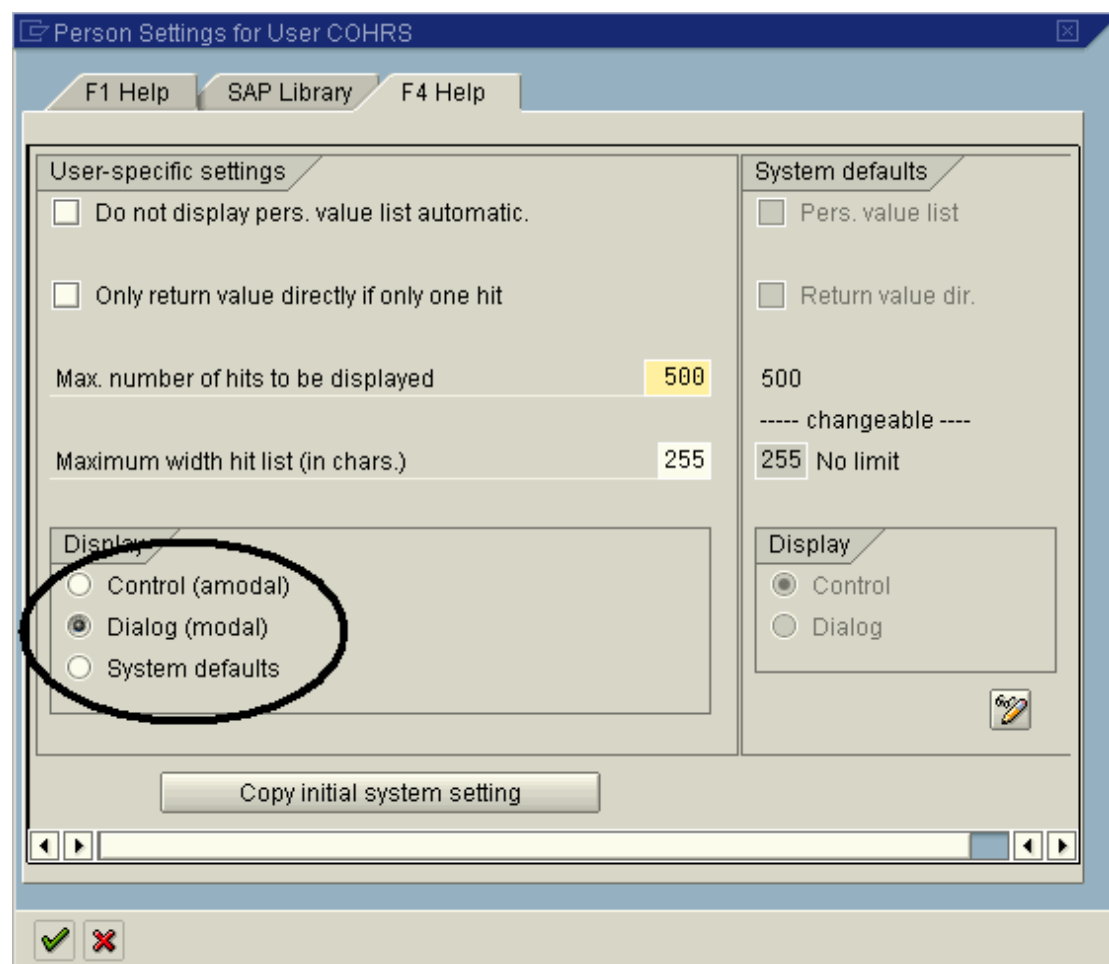
In this example the executable `wscript.exe` has been added to the list so that SAP GUI windows created from Visual Basic scripts are displayed using New Visual Design.

This information may also be set automatically in the registry. In branch `HKCU\Software\SAP\General\` a key with the name *Applications* should be created if it does not already exist.

To enable New Visual Design for `wscript.exe`, a new *wscript* key has to be created under the *Applications* key. Then a string value named 'Enjoy' should be added to the *wscript* key with the value 'On'.

Why do I receive control errors when running or recording a script?

SAP GUI Scripting does not support the non-modal version of the F4 help. This setting can be changed in the settings dialog available from the SAP GUI help menu.



On the tab labelled *F4 Help*, the *Display* value should be set to *Dialog (modal)*.

Does SAP GUI for Windows behave differently when recording or playing back a script?

There are some minor differences in the behaviour of SAP GUI for Windows when a script is recorded or played back. These are listed in SAP Note 587202.

Where do I get the latest version of the documentation?

The latest version of all the SAP GUI Scripting documentation and samples is available at the SAP service market place.

- Go to service.sap.com/sapgui

- Click on the link 'SAP GUI Scripting' in the tree on the right side of the page

How do I report problems to SAP?

Please create a message in SAP's support system OSS, and place it on the BC-FES-SCR component.



SAP Notes on Scripting

- 480149: Describes the ABAP and kernel patch level requirements
- 587202: Limitations of SAP GUI Scripting
- 548788: Creating trace files of SAP GUI Scripting problems to be send to SAP
- 527737: Composite SAP note on SAP GUI Scripting
- 612454: SAP GUI Scripting: Status and Lifetime
- 619459: SAP GUI Scripting support of SAP applications