

# SAP GUI Scripting User Guide



**Release 620**



## 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, JScript 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.

Copyright.....	2
 SAP GUI for Windows Scripting Support.....	4
 Running Scripts .....	4
 Recording a Script .....	5
 The Scripting Wizard.....	5
 JavaScript Engine for SAP GUI for Java.....	7
Availability and integration .....	7
Scripting window.....	7
Script recording modes .....	8
Event handlers.....	10
Concept of local and global scripting windows.....	11
Current scripting context .....	14
LiveConnect .....	14
 AppleScript support in SAP GUI for Java.....	18
 Script Editor application .....	18
 AppleScript dictionary for SAP GUI .....	18
 AppleScript menu and SAP GUI.....	23

## SAP GUI for Windows Scripting Support

SAP GUI for Windows comes with built-in support for recording and executing scripts.

### Running Scripts

There are 3 different ways to run a script:

- A script may be double-clicked on the desktop. In this case it will be executed by the Microsoft Windows Script Host.
- A script may be dropped on the SAP GUI window using Drag&Drop. It will then be executed using the Microsoft Script Control.
- A script may be run from the *Record and Playback* dialog, using the Microsoft Script Control:



This dialog is available from the *Customizing* menu of SAP GUI:



Using either the Microsoft Windows Script Host or the Microsoft Script Control, both Visual Basic Script (\*.vbs) and JScript (\*.js) files can be executed.

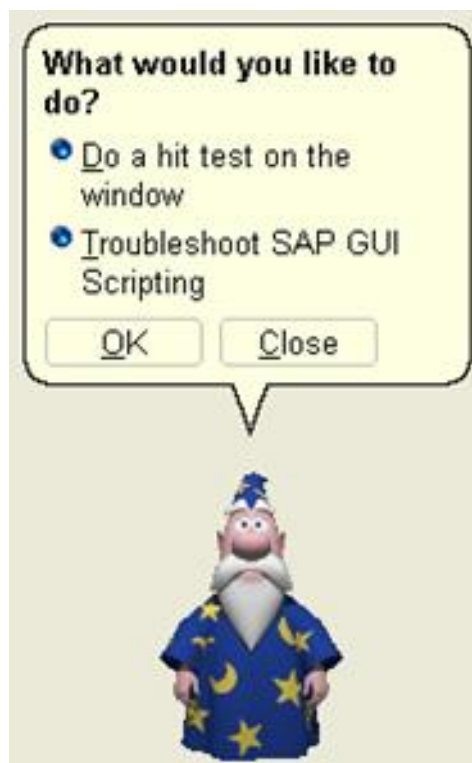
For your convenience the variables *application* of type *GuiApplication*, *connection* of type *GuiConnection* and *session* of type *GuiSession* are already predefined when a script is dropped onto the SAP GUI window or run from the *Playback and Record* dialog.

## Recording a Script

Clicking the *Record* button of the *Playback and Record* dialog shown above starts the record mode of SAP GUI in which all user actions are recorded into a Visual Basic Script. The file dialog opens when the recording is stopped using the *Stop* button, and you may save the script to the file system.

## The Scripting Wizard

Like the *Playback and Record* dialog, the Scripting Wizard is available from the *Customizing* menu by selecting the menu item *Script Development Tools*.



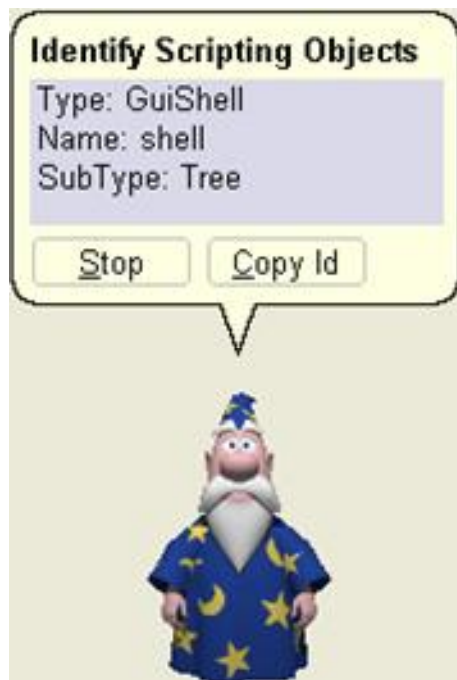
Using the troubleshooting option of the Wizard, you may be notified if any of the following problems have been detected:

- SAP GUI Scripting is disabled on the client PC by the end user. It should then be enabled in the Options dialog of SAP GUI.
- SAP GUI Scripting is disabled on the application server. The SAP system administrator may enable SAP GUI Scripting by setting the `sapgui/user_scripting` profile parameter.

- The current SAP system connection runs in low speed mode. This mode can be turned off in the properties dialog of the connection in SAPLogon.

The hit test mode of the Scripting Wizard may be used to identify scriptable objects within SAP GUI. When moving the mouse pointer over the SAP GUI window in hit test mode, the Wizard displays information about any scriptable object under the pointer.

You can copy the id of the object to the clipboard to use it in your own scripts. Please refer to the SAP GUI Scripting API documentation for a detailed explanation of ids and how to work with them.





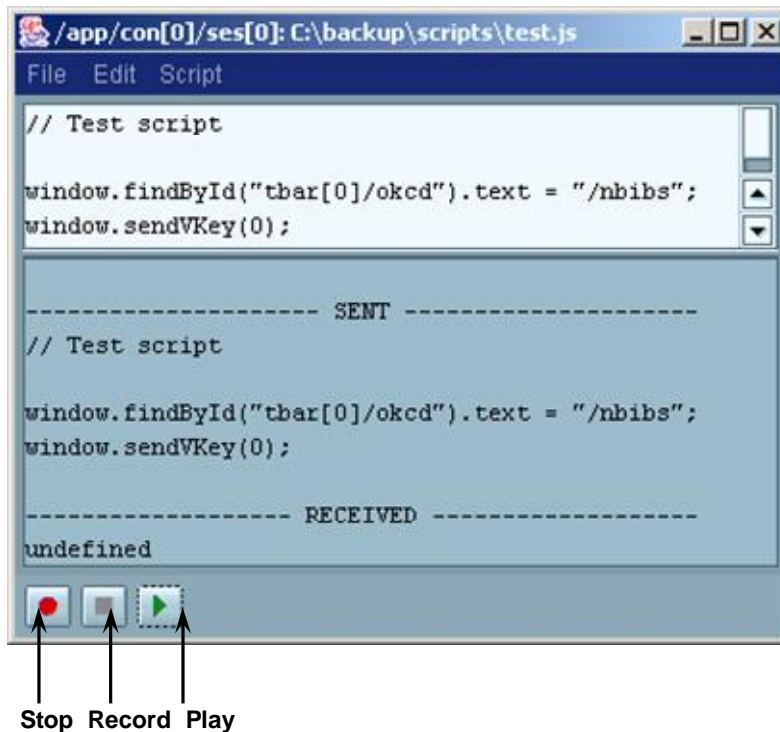
## JavaScript Engine for SAP GUI for Java

### Availability and integration

The JavaScript Engine is fully integrated into SAP GUI for the Java Environment (SAP GUI for Java) and therefore always available for scripting purposes during SAP GUI for Java runtime. To use the JavaScript Engine, open the scripting window from the '?' menu. It is also possible to run scripts by using the 'Scripts' menu from the menu bar in the main window.

### Scripting window

Since the JavaScript Engine is integrated in SAP GUI for Java and doesn't have its own user interface, it was necessary to implement a scripting window to be used for writing, recording and running JavaScript scripts within SAP GUI for Java.

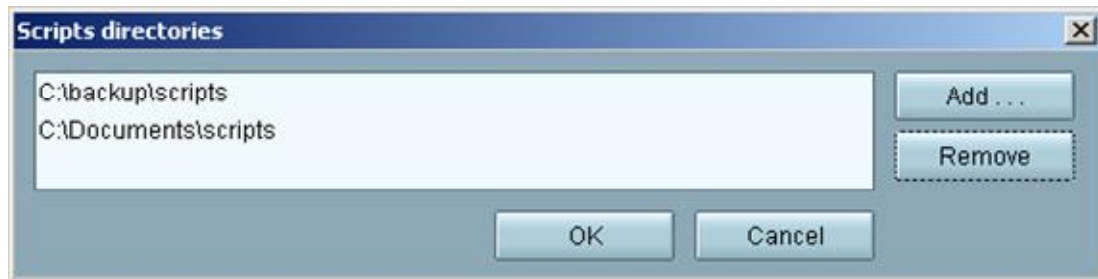


The scripting window is available from the '?' menu of each session. The predefined objects are *application*, *connection*, *session*, *window*, and *userarea*. This means that these objects can be addressed directly in a script without using an *application.findById(...)* call.

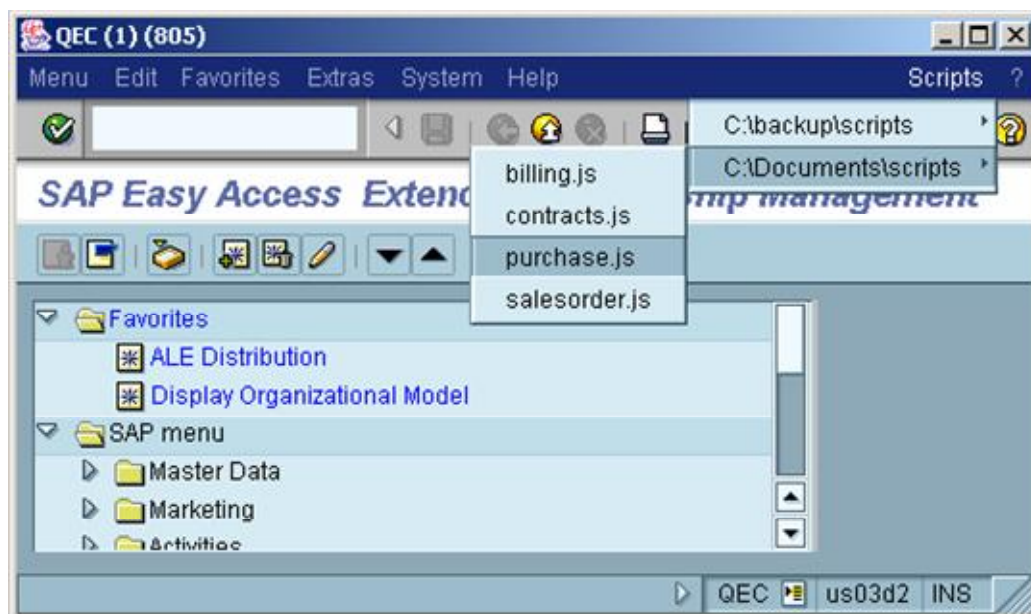
The scripting window consists of two parts: input and output. The title shows which *connection* and *session* (in this example these are first connection and first session) the scripting window belongs, and where the displayed script is located on the user's computer.

The user enters a script in the input part and clicks the 'Play' button. The JavaScript Engine processes the script and the result (if there is any) is written to the output part. Alternatively, the user can click the 'Record' button and afterwards continue working with SAP GUI for Java. All user actions will be recorded by SAP GUI for Java and displayed in the input part of the scripting window. After clicking the 'Stop' button, the recorded script can be saved by choosing 'File'-'>'Save'.

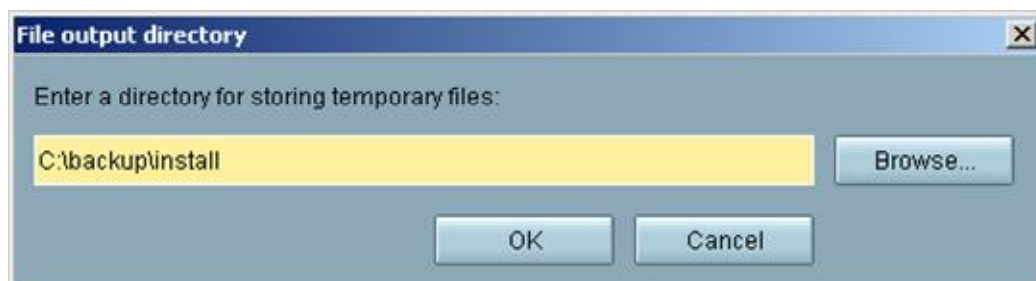
To make working with scripts more convenient, it is possible to set up directories where scripts may be stored. This feature is available from the 'File'->'Scripts directories...' menu.



If the user has any scripts stored in the scripting directories, they will be displayed per scripting directory under the 'Scripts' menu of the main window toolbar. Scripts can be run without reopening the scripting window simply by choosing them from the 'Scripts' menu:



By using the GuiUtils object from the Scripting API, it is possible to open and write temporary files. All files will be stored in the file output directory, which can be set by using the menu item 'File'->'File output directory...'. If not explicitly set, the Java user home directory is used by default.



## Script recording modes



SAP GUI for Java implements different recording modes for scripts. The scripting record mode can be set in the 'Options' section of the 'SAPGUI Preferences' dialog. This dialog is available from the 'Options'->'Preferences' menu command in the SAP GUI for Java logon window or from the '?' menu of the session:



In the global scripting window<sup>1</sup> all scripts are always recorded by using the *recording with absolute scripting id's* recording mode. That means that all script lines begin with the *application.findById()* call:

```
application.findById("/app/con[0]/ses[0]/wnd[0]/tbar[0]/okcd").text =
"/nbibs";
application.findById("/app/con[0]/ses[0]/wnd[0]").sendVKey(0);
application.findById("/app/con[0]/ses[0]/wnd[0]/usr").verticalScrollbar.position = 0;
application.findById("/app/con[0]/ses[0]/wnd[0]/usr").horizontalScrollbar.position = 0;
```

By default, the recording mode for the local scripting window is *recording with relative id's*. SAPGUI for Java makes use of these predefined variables in the local scripting context: *application*, *connection*, *session*, *window*, *userarea*. The above example will then appear as:

```
window.findById("tbar[0]/okcd").text = "/nbibs";
window.sendVKey(0);
userarea.verticalScrollbar.position = 0;
userarea.horizontalScrollbar.position = 0;
```

The third recording possibility in the local scripting context is *recording with common SAPGUI id's*. Using this recording mode, recorded scripts can be executed on SAP GUI for Windows. During recording in the common SAPGUI mode, SAP GUI for Java uses the predefined variables *application*, *connection* and *session*, which are also available and predefined in SAP GUI for Windows. If recorded in the common SAPGUI recording mode the script shown above will look like this:

```
session.findById("wnd[0]/tbar[0]/okcd").text = "/nbibs";
session.findById("wnd[0]").sendVKey(0);
session.findById("wnd[0]/usr").verticalScrollbar.position = 0;
session.findById("wnd[0]/usr").horizontalScrollbar.position = 0;
```

We recommend the *recording with relative id's* mode as the most convenient, with many predefined variables, and using the *recording with common SAPGUI id's* mode only for scripts, which are supposed to be run in both SAP GUI for Java and SAP GUI for Windows.

<sup>1</sup> See Chapter entitled 'Concept of local and global scripting windows'

## Event handlers

SAP GUI for Java defines global event handlers – functions, which are called if SAP GUI for Java fires certain types of events.

The following information describes all available event handlers. To use the event handlers, 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.

### **Function onEndRequest (session):**

This function is called immediately after the session is unlocked after server communication.

### **Function onSessionCreate (session):**

This function is called after a new session is created. The session is visualized in a new main window. This resembles the “/o” command that can be executed from the command field.

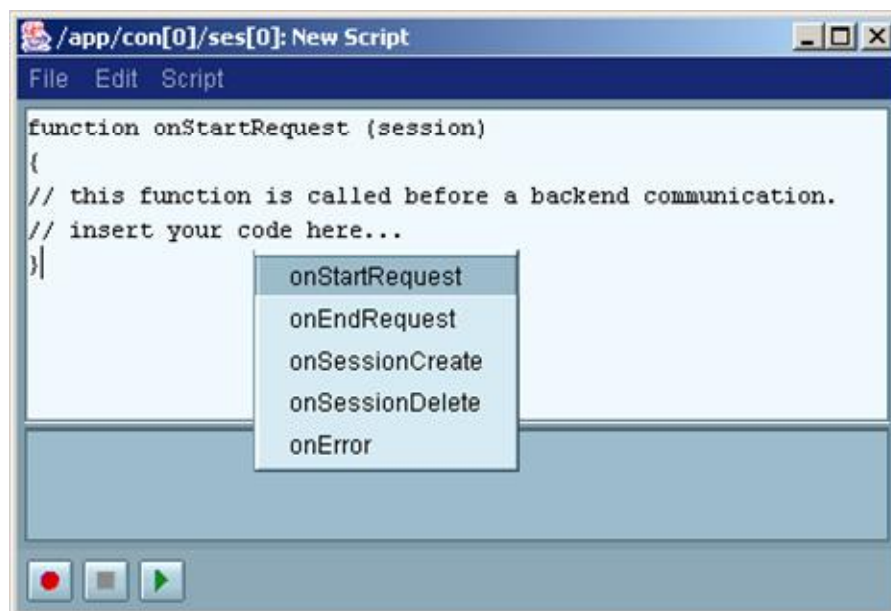
### **Function onSessionDelete (session):**

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

### **Function onError (session, errorid, desc1, desc2, desc3, desc4):**

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

To use a desired event handler, the user must open a scripting window and choose the ‘Insert’ command either from the ‘Edit’ or contextual menu. The contextual menu is made available by clicking the right mouse button on the input part of the scripting window (except for Macintosh, where it can be accessed by Control + click). After the event handler is displayed in the input part of the scripting window and code has been added, click the ‘Run’ button to execute the function. Once executed, the function is created in the current JavaScript context and will always be called when SAP GUI for Java fires the corresponding type of event.



To delete an event handler from the current JavaScript scope the user should choose this event handler from the 'Script'->'Destroy' menu.

## Concept of local and global scripting windows

In the context of JavaScript scripting there are two ways to write and execute scripts: using local and global scripting windows.

Irrespective of what kind of scripting window is used, a script addresses objects in a JavaScript context to access the objects from the SAP GUI for Java scripting component hierarchy. To define objects inside the JavaScript context means to define them in a so-called JavaScript Engine *scope*. A *scope* is a set of JavaScript objects. Execution of scripts requires a JavaScript scope for variable storage, and as a place to find standard objects such as a function. To be able to address objects they must be defined in the JavaScript scope. Once defined, the objects can be used by any script running in the same JavaScript scope.

The **global scripting window** is available from the '?' menu of the SAP GUI for Java logon window. The only predefined object is *application*. You have to use the *application.findById(...)* method to access objects from the scripting component hierarchy.

The **local scripting window** is available from the '?' menu of the session. Each running session has its own local scripting window. The predefined objects are *application*, *connection*, *session*, *window*, and *userarea*.

A disadvantage of global scripting is that all defined variables and objects share the same JavaScript scope. A running script, which changes the values of the objects, simply overwrites them. The next executing script then has to deal with the overwritten values of these objects. Consider the execution of the following 3 scripts:

Script 1:

```
session = application.findById("/app/con[0]/ses[0]");
window = session.getActiveWindow();
```

Script 2:

```
session = application.findById( "/app/con[0]/ses[1]" );  
window = session.getActiveWindow();
```

Script 3:

```
button = window.findById( "tbar[1]/btn[19]" );
```

If Script 2 is executed after Script 1, Script 3 will access the button in the window of the second session. If Script 1 is run after Script 2, Script 3 will access the button in the window of the first session. There is therefore no guarantee that after execution of a script the value of any object in the global scripting context will still be the same.

It is also difficult to run scripts for different sessions simultaneously inside the global scripting context. Since it is possible to execute only one script at a time, you would have to write all scripting instructions into one script and differentiate between the sessions in it.

One of the advantages of the local scripting window is the possibility to write and execute scripts in the context of a session. That way you may run scripts simultaneously in different sessions without any sort of collision. The local scripting approach is based on the local JavaScript scope, which is created for each session. All objects defined inside the local JavaScript scope are accessible only within the session to which that JavaScript scope belongs. There is also no danger of the values of the variables being overwritten. Taking the example of the three Scripts above, Script 1 and Script 3 will be executed in the local scripting window of first session, if the button of the first session is needed; Script 2 and Script 3 will be run in the local scripting window of the second session, if the button of the second session is needed.

Using a new JavaScript scope for each session makes it possible to define the most frequently used objects, which are indispensable for each script: *application*, *connection*, *session*, *window*, and *userarea*.

The advantage of predefined objects will become evident from the following script example. Consider two connections and three sessions belonging to the first connection. The purpose is to find a button on the main window of the second session of the first connection and to click it. Without predefined objects in a global scripting context the path to the button must be fully specified, such as:

```
var usr = application.findById( "/app/con[0]/ses[1]/wnd[0]/usr/" );  
usr.findById( "btnPB1" ).press();
```

In a local scripting context the script will consist of just a single line using the predefined object *userarea*:

```
userarea.findById( "btnPB1" ).press();
```

In addition to the simplicity of accessing objects through predefined objects, the predefined objects also make it possible to record a script for one session and execute it in another one. The benefit of the predefined *session* object is that connection and session indexes, which are necessary for accessing the current session, (e.g. *con[0]/ses[1]*), are omitted and the *session* object can be queried directly. This makes the recorded script session independent:

```
session.findById( "wnd[0]/usr/btnPB1" ).press();
```

To allow the user to control the execution of scripts SAP GUI for Java defines global event handlers – functions, which are called if SAP GUI for Java fires certain types of events.

When using event handlers in a global scripting context, you are required to program “if” or “switch/case” conditional instructions to differentiate between the objects belonging to

different sessions and connections and to prevent values from being overwritten. Consider the case of writing *onEndRequest* - event handlers for two sessions in the first connection using the local scripting context. The *onEndRequest* event handler is a function, which is called when the session is unlocked after server communication. You may just use the local scripting windows of the two sessions and place the following script into each of them:

```
function onEndRequest (session)
{
  var utils = application.utils;
  utils.showMessageBox("Info", "Session "+ session.getId()+
    " had server communication right now.",
    utils.MESSAGE_TYPE_INFORMATION,
    utils.MESSAGE_OPTION_OK);
  // do something for this session
  . . .
}
```

Suppose you use the *onEndRequest* - event handler in the global scripting context. You would have to create code like this in the global scripting window:

```
function onEndRequest (session)
{
  var utils = application.utils;

  // we need access to the first connection
  var connection = application.findById("/app/con[0]");
  var sessions = connection.getChildren();

  var sesId = session.getId();

  for (var i=0; i<sessions.getLength(); i++)
  {
    // check if the session belongs to the connection 0
    if (sessions.elementAt(i).getId() == sesId)
    {
      switch(sesId)
      {
        case "/app/con[0]/ses[0]":
          utils.showMessageBox("Info", "Session 0 " +
            "had server communication right now.",
            utils.MESSAGE_TYPE_INFORMATION,
            utils.MESSAGE_OPTION_OK);
          // do something for session 0
          . . .
          break;
        case "/app/con[0]/ses[1]":
          utils.showMessageBox("Info", "Session 1 "+
            "had server communication right now.",
            utils.MESSAGE_TYPE_INFORMATION,
            utils.MESSAGE_OPTION_OK);
          // do something for session 1
          . . .
          break;
      }
      return;
    }
  }
  utils.showMessageBox("Error", "For the connection 0 is session "+
    sesId + " unknown",
    utils.MESSAGE_TYPE_ERROR,
    utils.MESSAGE_OPTION_OK);
}
```

```
}
```

The second way is more complicated and if not implemented correctly, can cause errors that are difficult to find.

In summary, the advantages of local scripting are that it

- allows you to predefine essential scripting objects
- prevents overwriting object values
- allows you to execute session-dependent scripts simultaneously
- avoids complicated programming and errors

The global scripting window is convenient to use for session independent scripting purposes. You may also want to use the global scripting window to open first connection per script.

## Current scripting context

Each scripting window has its own JavaScript scope. All scripts, which are executed in the scripting window, share the same JavaScript scope. This means that all defined functions (event handlers) and variables are shared. For more information about the JavaScript Engine consult the JavaScript Engine documentation at <http://www.mozilla.org/rhino>.

Choose 'Script'-'>'Destroy'-'>'current Context' to remove all user defined objects and functions (event handlers) from the current JavaScript scope. You might want to use this feature to remove all defined variables and functions and continue to work in a new scripting context.

After closing a scripting window the scripting context is destroyed automatically. If any event handler is defined, the user will be asked whether the current scripting context should be destroyed. You might want to keep a scripting context to be able to execute scripts in it after reopening the scripting window.

## LiveConnect

LiveConnect is a technology that connects JavaScript and Java. You may use LiveConnect to create Java classes and call Java methods from within JavaScript. LiveConnect is fully supported by the JavaScript Engine used in SAPGUI for Java.

The following example illustrates the Java-to-JavaScript connectivity. Note that the class name must be fully qualified by its package. This is an example of custom naming of packages where *sample* must be a word other than *java*. i.e. "sample.javax.swing.JButton":

```
// create a new StringBuffer.  
  
var sb = new java.lang.StringBuffer();
```

```
// add some stuff to the buffer.
sb.append("hi, java!");

// print it to the java console.
java.lang.System.out.println(sb)
```

The following example allows you to choose a text file using the Java JFileChooser class, parses the chosen file and prints its content to the Java console. The entries in the text file must be separated by tabulators and line separators (carriage returns).

To test this example, export an Excel sheet into a text file. Then you may import the Excel sheet into a SAP R/3 table using SAPGUI for Java Scripting. This example reads the content of the text file which you may then write into your SAP R/3 table using the SAPGUI for Java scripting methods:

```
// open a file chooser
filePath = openFileChooser ();

// read string from file
text = readStringFromFile (filePath);

// parse file content
parseString (text);

// open a file chooser
function openFileChooser ()
{
    var filePath;

    try
    {
        chooser = new Packages.javax.swing.JFileChooser();
        retVal = chooser.showOpenDialog (null);

        if (retVal == Packages.javax.swing.JFileChooser.APPROVE_OPTION)
            filePath = chooser.getSelectedFile().getAbsolutePath();
        catch (e)
        {
            application.utils.showMessageDialog ("Error in openFileChooser",
                                                "Exception: " + e, 0, 0);
        }
        return filePath;
    }
}

// read a string from a file
function readStringFromFile (fileName)
{
    var result = "";
    var finished = false;

    try
    {
        // create new buffered reader
        bufReader = new java.io.BufferedReader(new
            java.io.FileReader(fileName));

        // loop until EOF
        while (!finished)
        {
            // read the next line
```

```
    line = bufReader.readLine();
    // if we are not finished
    if (line != null)
    {
        // add the line
        result = result + line + "/n";
    }
    else
    {
        // we are finished
        finished = true;
    }
}
// close buffered reader
bufReader.close();
}
catch (e)
{
    application.utils.showMessageDialog ("Error in readStringFromFile",
                                        "Exception: " + e, 0, 0);
}
// return file content
return result;
}

// parses a string
function parseString (text)
{
    try
    {
        // create new tokenizer
        parser = new java.util.StringTokenizer (text, "/n/r/t");
        while (parser.hasMoreTokens())
        {
            token = parser.nextToken();
            handleToken (token);
        }
    }
    catch (e)
    {
        application.utils.showMessageDialog ("Error in parseString",
                                            "Exception:" + e, 0, 0);
    }
}

// do something with the given token
function handleToken (token)
{
    java.lang.System.err.println ("Token: " + token);
}
```

Use the SAP GUI for Java scripting window to run these examples or to write your own. For more information about LiveConnect refer to the official Netscape documentation at <http://developer.netscape.com/docs/manuals/liveconnect.html>







## AppleScript support in SAP GUI for Java

The AppleScript support in SAP GUI for Java is available only on Mac OS X. AppleScript is a scripting language that allows you to control Macintosh computers without using the keyboard or mouse. With AppleScript, you can use a series of English-like instructions, known as a script, to control SAP GUI applications.



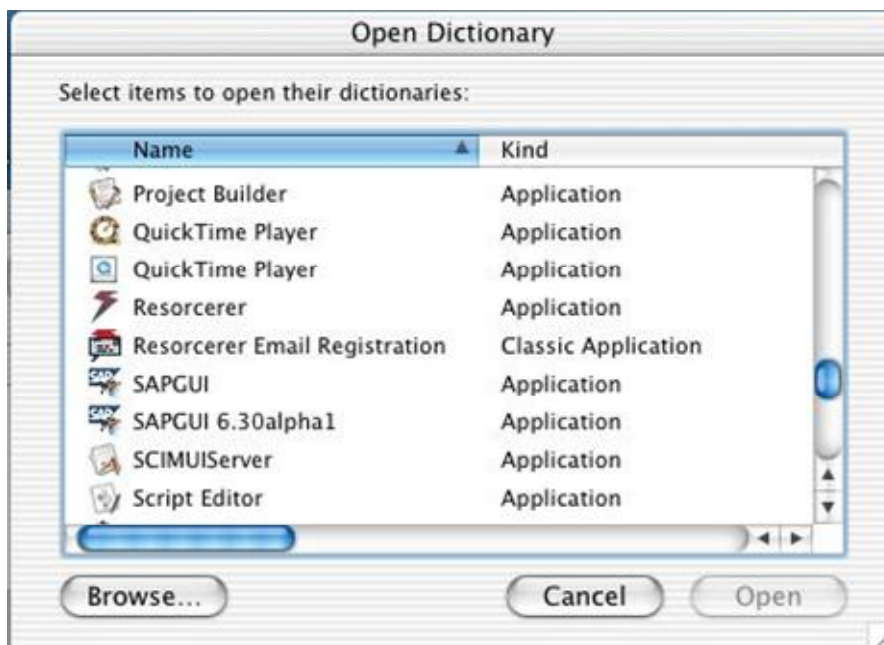
## Script Editor application

The Script Editor is the application that allows you to read, write, record, and save AppleScript scripts. It is located in the AppleScript folder in the Applications folder on your hard drive.



## AppleScript dictionary for SAP GUI

1. Every scriptable application must have a set of commands that it understands and can execute; this set is known as its dictionary. To open the SAP GUI dictionary, simply drag its icon onto your Script Editor and the dictionary appears. You can also choose File -> Open Dictionary... and select SAP GUI from the applications list in the dialog that appears:



The Script Editor will extract the scripting terminology from the chosen application and display it in a multi-paned browser window. The browser window has two panes separated by a movable divider. The scrollable pane on the left contains the SAPGUI terminology. Within the terminology there are classes, the scriptable application objects and events, which are the verbs or commands used to control the scriptable objects.

Entries selected in the left pane have their definitions and descriptions displayed in the right pane.

### Class Definitions

A class definition describes the properties and elements of a scriptable object. The following picture shows information about *table control* class:

For Script Editor 1.9:



For Script Editor 2.0:



Typically, a class definition begins with the term followed by a short description of the term on the same line.

This is followed by a list of elements or objects which belong to or comprise the scriptable object. A list of elements is not included in all class definitions as some scriptable objects do contain other scriptable objects while others do not.

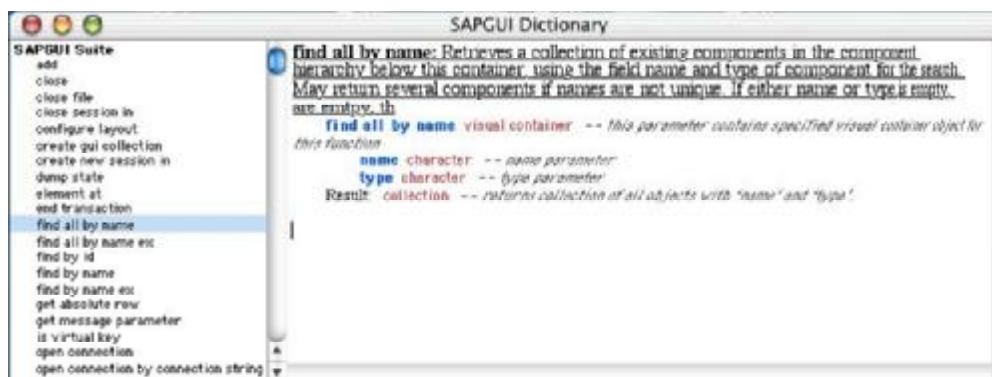
The class definition ends with a list of properties or qualities of the scriptable object and their values. A property entry begins with the name of the property, followed by:

- a description of the data format in which the property value is stored, such as: integer (number), point (a list of two numbers), string (textual data), boolean (true or false), and so on.
- an optional type indicator, [r/o], which indicates that the property has the 'read-only' attribute and cannot be changed.
- a short description of the property.

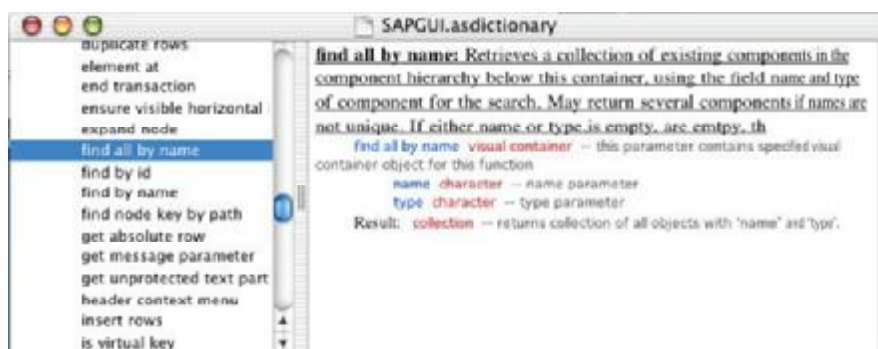
## Event Definitions

An event definition describes a verb or command used to target an action of a scriptable object. The following picture shows the screen shot, containing information about the *find all by name* event:

For Script Editor 1.9:



For Script Editor 2.0:



Typically, an event definition begins with the term followed by a short description of the term on the same line.

This is followed by a list of any parameters used during the execution of the command.

Each parameter entry begins with the name of the parameter followed by a description of the data format used with it, and ends with a short description of the parameter.

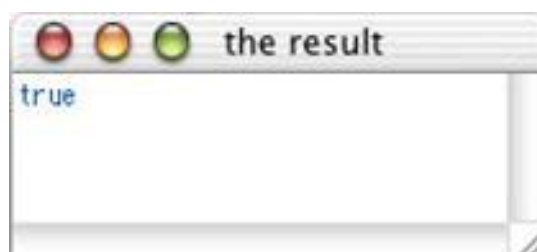


Running scripts using the Script Editor application

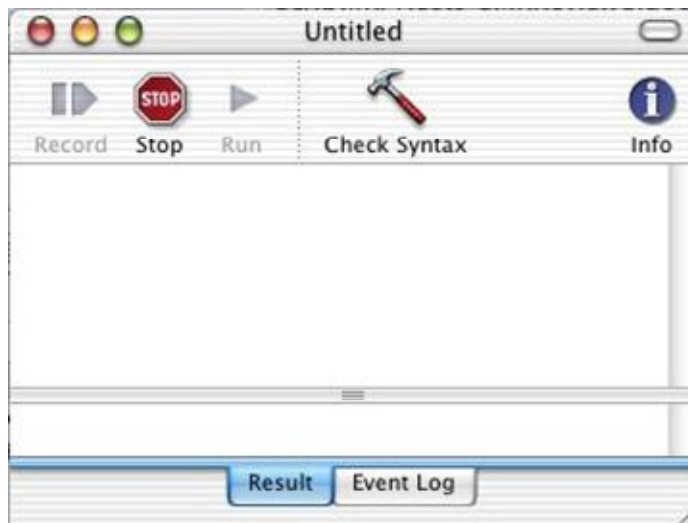
To run a script, click the Run button on the script window toolbar or press Command-R. If the script has not been checked for syntax errors, the check will take place before the script is executed.

To view the result, if any, of the last action of the script:

For Script Editor 1.9: Press Command+L key or choose Controls -> Show Result to open the Result window:



For Script Editor 2.0: Click the Result tab at the bottom of the script window. The script result will be displayed in the bottom section of the script window:

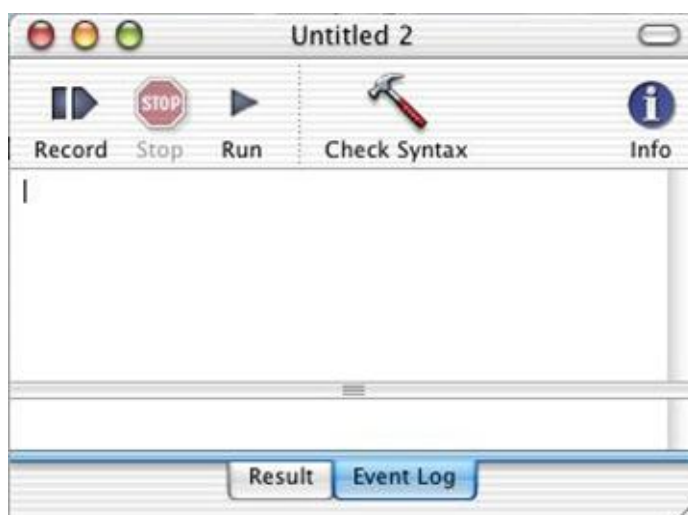


To view the actions of a script and their results:

For Script Editor 1.9: Choose Controls -> 'Open Event Log' or press Command + E key to open the Event Log window:



For Script Editor 2.0: Click the Event Log tab at the bottom of the script window before running the script. The lower pane will display an ongoing log of each script action and its result during the execution of the script. The Event Log provides the means to check if your script is performing its actions as expected or if it requires more editing.



If you are running a script from the Script Editor, you can press Command-period or click the Stop button to stop running the script.

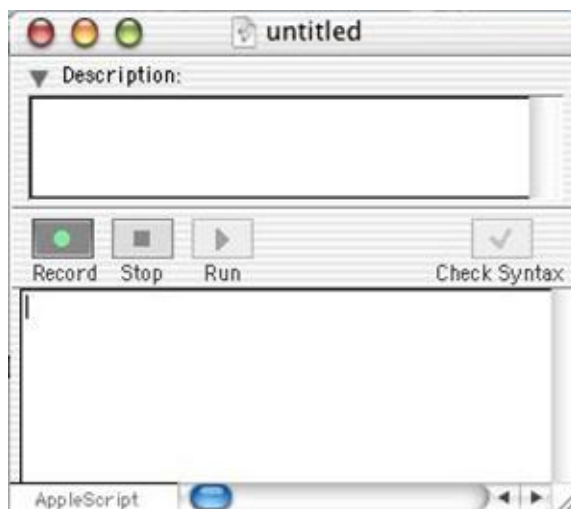


#### Recording scripts using the Script Editor application

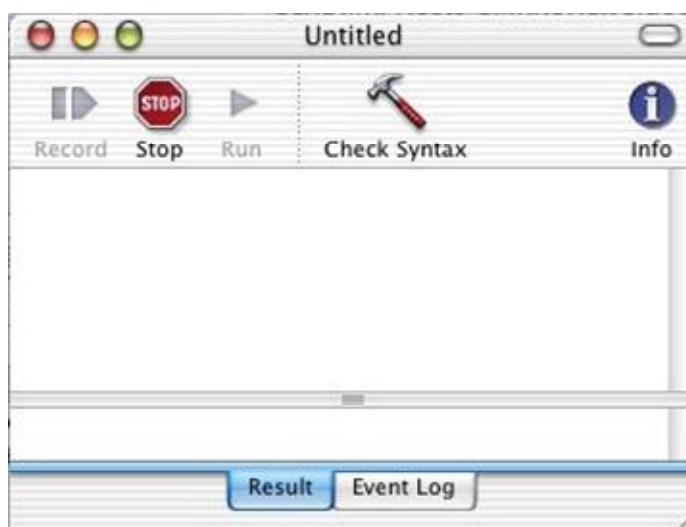
SAP GUI is a recordable application. This means that you can record all user actions in SAP GUI and then study in the Script Editor the scripts that SAP GUI creates. Since the application is "writing" the script, you will get a better idea of the commands it understands and the way they should be scripted.

To activate recording AppleScript for an action open the Script Editor and press the Record button.

For Script Editor 1.9 you can also use the Command+D shortcut:



For Script Editor 2.0 you can also use the Command+Shift+R shortcut:



Now switch to the SAP GUI application and perform some basic tasks. For example, open any transaction, change the text inside the field for this transaction, click the checkbox inside the window, save these changes, or switch to another transaction. Switch back to the Script Editor and press the Stop button. In the Script Editor window you will see the AppleScript commands which were recorded when you performed the SAP GUI tasks.

## Commenting in AppleScript

There are two methods for commenting in AppleScript. One is simply to precede the text to be commented with two hyphens (--), which disables everything else on that single line of a script, like this:

```
-- send virtual key 0 to main window "/app/con[0]/ses[0]/wnd[0]"
```

The second method, mostly used for multiple lines of code, is to enclose it like this: (\*commented text here \*). The example below shows a sample script where the last three lines have been commented:


```
tell application "SAPGUI"
    activate
    set txt of OKCode field "/app/con[0]/ses[0]/wnd[0]/tbar[0]/okcd" to
    "bibs"
    set txt of text field "/app/con[0]/ses[0]/wnd[0]/usr/txtRSYST-
    BNAME" to "demo"
    set txt of password field "/app/con[0]/ses[0]/wnd[0]/usr/pwdRSYST-
    BCODE" to "demo"
    (*
    set focus text field "/app/con[0]/ses[0]/wnd[0]/usr/txtRSYST-LANGU"
    send virtual key 0 to main window "/app/con[0]/ses[0]/wnd[0]"
    press button "/app/con[0]/ses[0]/wnd[0]/tbar[1]/btn[19]"
    *)
end tell
```




## AppleScript menu and SAP GUI

Mac OS X users can use an **AppleScript menu** for fast accessing and running scripts for the SAP GUI application.

To install the **AppleScript menu** go to the AppleScript folder in Applications and open the

**Script Menu.menu** application. The Mac OS X system adds a new element () to the system toolbar.

To create the SAP GUI element for **Script menu** choose the **Open Scripts Folder** command

from the  menu. The Mac OS X system switches to the Finder application and opens the Scripting folder (by default it is the Home/Library/Scripts folder). Create a new SAP GUI folder inside this folder.

If you want to make your script accessible from the Script menu, save your AppleScript in the SAP GUI folder, which can be found in Home/Library/Scripts. The SAP GUI AppleScript must be saved as a compiled script, as the Script menu works only with compiled scripts. To save your script as compiled, in the Script Editor application choose File->Save As... The **Save** dialog appears. Choose the **compiled script** option in the **File Format** pop-up menu.

The following example shows the SAP GUI menu with three scripts: Connect to SMK server, SAP GUI AppleScript Demo and SAP GUI open connection.

