OCLC

SCRIPTING IN WINIBW3

Getting started

author:	Jürgen Hofmann, Frans Bouwmeester, Jian Zhang
business unit:	ITC
date:	18/11/2013
document name:	Scripting in WinIBW3.doc
version:	1.18

DOCUMENT HISTORY

Date	Version	Author	Remarks
10-10-2004	0.1	JH	Creation
19-10-2004	0.2	JH	Added application object model
13-12-2004	1.5	FB	Adapted description of FindTag.
21-12-2004	1.6	FB	Corrected description of PressButton.
08-03-2005	1.7	FB	Corrected description of Find and ReplaceAll. Added Replace
21-03-2005	1.8	JH	Corrected description of "Variable" property
12-07-2006	1.9	FB	Adapted for WinIBW 3.1. Added utility objects, and a section about XUL programming.
12-09-2007	1.10	FB	Added section about Preferences and Environment variables
15-7-2008	1.11	JZ	Updated for WinIBW 3.3.
22-12-2008	1.12	JZ	Updated for WinIBW 3.3.3.
06-11-2009	1.13	JZ	Updated for WinIBW 3.3.8.
25-05-2010	1.14	JZ	Updated for WinIBW 3.4.
07-04-2011	1.15	JZ	Updated for WinIBW 3.4.1.
10-11-2012	1.16	JZ	Updated for WinIBW 3.6
18-04-2013	1.17	JZ	Updated for WinIBW 3.6.1
18-11-2013	1.18	JZ	Updated for WinIBW 3.7

CONTENTS

1 Introduction

WinIBW3 provides scripting functionality for repeating cataloging tasks automatically. This document will help you get started with scripting in WinIBW3. It first explains what the WinIBW3 scripts are. Then it provides an overview of all the WinIBW3 functions\attributes which can be called\retrieved from WinIBW3 scripts, utility objects for file input and output, and XUL programming for dialog boxes. Last, it highlights the differences between VBScript and JavaScript, and the changes in WinIBW3 scripting comparing with WinIBW2.

This document is no general tutorial about JavaScript and VBScript. For this, excellent resources are available on the internet. See for example:

- A Beginner's Guide to JavaScript
- JavaScript for the Total Non-Programmer
- PageResource JavaScript tutorial

The intended audience for this document:

- OCLC developers responsible for creating general WinIBW3 standard scripts,
- system administrators responsible for creating site-specific WinBW3 standard scripts,
- WinIBW3 end user for editing or recording user scripts,
- writers of end user documentation,
- testers of the WinIBW3 product.

2 WinIBW scripts

In WinIBW3 there are two kinds of scripts: Standard scripts and User scripts.

2.1 Standard scripts

Standard scripts are the scripts provided by OCLC or site administrators and included in the WinIBW3 setup. They are created in JavaScript.

Standard scripts can be edited in any editor outside WinIBW3. However, Unicode characters can't be directly copied and pasted instead Unicode values have to be used (e.g. \u65B0).

A user of WinIBW3 may not change Standard scripts, but can invoke the functions defined in Standard scripts in the following ways:

- By means of the user interface;
- Via the Short-key assigned to a function;
- Via the function call 'callStdScriptFunction' from a user script, see section 3.1.18 about how to use it.

2.2 User scripts

User scripts are the scripts recorded or edited by a user of WinIBW3. They can be created in either JavaScript or VBScript, depending on the setting in the WinIBW3 setup when you use a WinIBW3 new version for the first time. However, the language for user scripts can be random switched between JavaScript and VBScript within WinIBW3. See the document "WinIBW3 Product Description.doc" about how to select the scripting language within WinIBW3.

User scripts can be edited in the script editor embedded within WinIBW3. This editor is Unicode support, where Unicode characters can be directly copied and pasted. See the document "WinIBW3 Product Description.doc" about how to how to use the WinIBW3 script editor.

The functions defined in user scripts can be invoked in the following ways:

- By means of the user interface;
- Via the Short-key assigned to a function.

2.3 Standard scripts versus User scripts

Standard scripts are processed by the Mozilla IDL and executed in the Mozilla engine, whereas User scripts (in both VBScript and JavaScript) are processed by the Microsoft IDL and executed in the Microsoft engine, i.e. they run in separate environments. Therefore, they can not interfere with each other. However, a function defined in standard scripts can be called from a user script of both VBScript and JavaScript.

Since the Microsoft IDL supports default value, the function parameter in user scripts of both VBScript and JavaScript can be optional. The Mozilla IDL does not work with default value. That is why there is no optional parameter in a JavaScript function of Standard scripts.

3 The WinIBW3 Application Object Model

In this section, all the attributes and functions of the WinIBW3 Application Object Model are presented with one table for each item. They are grouped according to an actual object. In each table, the first column lists the attribute or function name and the function parameters in case of a function. The second column lists the attribute type or the return type of the 'get' function, and the third column lists the attribute access type (readonly [get], writeonly [set] or readwrite [get/set]) in case of an attribute. Besides, for each item the interface for both VBScript and JavaScript are presented in each table. The interface for JavaScript is the same for both User Scripts and Standard Scripts in case that there are no optional parameters. If an interface has optional parameters, these optional parameters are only applicable for User Scripts (both VBScript and JavaScript). Moreover, the functionality for each item is explained at the last part of each table.

3.1 The `Application` Object

The Application object is a top level object and may be accessed as `Application` in VBScript and as `application` in JavaScript.

3.1.1 Attribute ActiveWindow

Name		type	access
VB	**ActiveWindow**	**ActiveWindow**	**[get]**
JS	**activeWindow**	**IActiveWindow**	**[get]**

Returns a reference to the currently active window. Refer to The ActiveWindow object for more details on the returned object.

3.1.2 Attribute Height

Name		type	access
VB	**Height**	**long**	**[get/set]**
JS	**height**	**int**	**[get/set]**

The height of the main window in pixels.

3.1.3 Attribute Width

Name		type	access
VB	**Width**	**Long**	**[get/set]**
JS	**width**	**int**	**[get/set]**

The width of the main window in pixels.

3.1.4 Attribute OverwriteMode

Name		type	access
VB	**OverwriteMode**	**Boolean**	**[get/set]**
JS	**overwriteMode**	**bool**	**[get/set]**

Indicates whether WinIBW is in overwrite mode (true) or in insert mode (false).

3.1.5 Attribute Language

Name		type	access
VB	**Language**	**String**	**[get]**
JS	**language**	**string**	**[get]**

Returns the currently active language. Return values can be “FR” = French, “DU” = German, “NE” = Dutch, “EN” = English and “TR” = Turkish.

3.1.6 Attribute IgnoredColor

Name		type	access
VB	**IgnoredColor**	**Long**	**[get/set]**
JS	**ignoredColor**	**int**	**[get/set]**

Specifies the colour used in the title edit control for ignored text.

In VBScript you may use predefined colour constants like vbRed, vbGreen, vbBlue etc. The colours are defined in reverse RGB fashion, i.e. you define blue, red, green in this order. In VBScript you may use a form of `CLng(“&h00C000”)` which means no blue, C0 green and no red.

In JavaScript you may use a constant like `0x00C000`.

3.1.7 Attribute ProtectedColor

Name		type	access
VB	**ProtectedColor**	**Long**	**[get/set]**
JS	**protectedColor**	**int**	**[get/set]**

Specifies the colour used in the title edit control for protected text.

In VBScript you may use predefined colour constants like vbRed, vbGreen, vbBlue etc. The colours are defined in reverse RGB fashion, i.e. you define blue, red, green in this order. In VBScript you may use a form of `CLng(“&h00C000”)` which means no blue, C0 green and no red.

In JavaScript you may use a constant like `0x00C000`.

3.1.8 Attribute ReceivedMessageOnly

Name		type	access
VB	**ReceivedMessageOnly**	**Boolean**	**[get]**
JS	**receivedMessageOnly**	**bool**	**[get]**

This property is set to true by WinIBW, if the last communication with the server only consisted of a message received by the server.

3.1.9 Attribute Windows

Name		type	access
VB	**Windows**	**WindowCollection**	**[get]**
JS	**windows**	**IWindowCollection**	**[get]**

A collection of WindowObject that resembles the currently open windows. Refer to The WindowsCollection object for more details on the return object.

3.1.10 Function Activate

Name	
VB	**Sub Activate**
JS	**void activate()**

Makes WinIBW the active window.

3.1.11 Function ActivateCommandLine

Name

```
VB  Sub ActivateCommandLine()
JS  void activateCommandLine ()
```

Moves focus to command line.

3.1.12 Function ActivateView

Name

```
VB  Sub ActivateView()
JS  void activateView()
```

Moves focus to title editor or short presentation list or other lists, depending on which one is on the window.

3.1.13 Function ActivateWindow

Name

```
VB  Sub ActivateWindow(windowID as long) as Boolean
JS  bool activateWindow(windowID: int)
```

Activates the window with the WindowID `windowID`. Returns True on success and False on failure (i.e. the window with the given ID does not exist).

3.1.14 Function NewWindow

Name

```
VB  Sub NewWindow() as long
JS  int newWindow()
```

Opens a new window and returns its windowId as long.

3.1.15 Function CloseWindow

Name

```
VB  Sub CloseWindow(windowID as long) as Boolean
JS  bool closeWindow(windowID: int)
```

Closes the window with the WindowID *windowID*. Returns True on success and False on failure (i.e. the window with the given ID does not exist).

3.1.16 Function Connect

Name

```
VB  Sub Connect(host as string, port as string) as Boolean
JS  bool connect(host: string, port: string)
```

Makes a connection to the CBS and LBS, and returns TRUE if the connection was established succesfully. *strHost* is the machine name or IP Address; *strPort* is the port to connect to. *strPort* may be a range of ports separated by a hyphen. This function opens a new window for the new connection, which is not closed when the connection fails.Example:

```
      Application.Connect(“chico.pica.nl”, “1024-1028”)
```

3.1.17 Function MessageBox

Name

```
VB  Sub MessageBox(title as string, message as string,
           iconName as string)
```

```
JS  void messageBox(title: string, message: string,
          iconName: string)
```

Displays a (Unicode capable) Message Box with title *title* and message *message*. *IconName* specifies the icon to be displayed; this may be “alert-icon”, “error-icon”, “message-icon” or “question-icon”.

3.1.18 Function CallStdScriptFunction

Name

```
VB  Sub CallStdScriptFunction(functionName as string)as Boolean
JS  bool callStdScriptFunction(functionName as string)
```

From user scripts, makes a call of a function defined in standard scripts. ***functionName*** **is the name of the function to be called. Return True on success and False on Failure (e.g. the function with the given name does not exist).**

3.1.19 Function PauseScript

Name

```
VB  Sub PauseScript()
JS  Void pauseScript()
```

Pauses script execution of the user script until the script is resumed by invoking “Resume” on the script menu. Standard scripts can not be paused.

3.1.20 Function AddSyntaxColor

Name

```
VB  Sub AddSyntaxColor(format as string, regex as string,
          color as long)
JS  void addSyntaxColor(format: string, regex: string,
          color: string)
```

This function adds a syntax colouring expression to WinIBW.

> *format* specifies one of the well-known formats (“UNM”, “D”, etc).
>
> *regex* specifies the regular expression that has to be matched to
> apply the colorization
>
> *color* specifies the colour used for the colorization

Please refer to the SetupStudio Users Manual for a in depth discussion of how to configure syntax colouring and a specification of the regular expression syntax used.

3.1.21 Function RemoveSyntaxColor

Name

```
VB  Sub RemoveSyntaxColor(format as string, regex as string)
JS  void removeSyntaxColor(format: string, regex: string)
```

Removes a syntax colouring expression previously added by a call to *AddSyntaxColor*. *format* and *regex* have to match the parameters that where used for the *AddSyntaxColor* call.
You may also specify an empty string for any of these parameters, which will match any format or any expression respectively.
Please refer to the SetupStudio Users Manual for a in depth discussion of how to configure syntax colouring and a specification of the regular expression syntax used.

3.1.22 Function ShellExecute

Name

```
VB  Sub ShellExecute(URL as string [, showCommand as short = 5]
         [, operation as string = "open"] [, parameters as string =
         ""])
JS  void shellExecute(URL: string, showCommand: short, operation:
         string, parameters: string)
```

performs an operation on the specified file, e.g. you can display web pages or print specified files.

URL is the URL of the file on which the operation should be performed. You may either specify a well-formed URL or an absolute path.

showCommand specifies how an application will be displayed when it is opened. The possible values are:

0	Hides the window and activates another window
3	Maximizes the specified window
5	Activates the window, displays it in its current size and position
6	Minimizes the specified window and activates the next top-level window
9	Activates and displays the window. If the window is minimized or maximized, it is restored to its original size and position.

In user scripts of both VBScript and JavaScript, *showCommand* is optional and defaults to 5.

operation specifies the verb that should be performed on the file. The following verbs are commonly used:

"edit"	Launches an editor and opens the document for editing
"explore"	Explores the folder specified by *strURL*
"find"	Initiates a search starting from the specified directory
"open"	Opens the file specified by the *strURL* parameter. The file can be an executable, a document, a folder or a web page.
"print"	Prints the document specified by *strURL*.

In user scripts of both VBScript and JavaScript, *operation* is optional and defaults to "open".

In user scripts of both VBScript and JavaScript, *parameters* is optional and defaults to an empty string. If the *URL* parameter specifies an executable, *parameters* will be passed to the executable.
VB example:

```
Application.ShellExecute "http://www.oclcpica.com"
```

Javascript examples:

```
application.shellExecute("http://www.oclcpica.com", 5, "open", "");

application.shellExecute(
   "file:///C:/Program%20Files/Microsoft%20Office/OFFICE11/WINWORD.EXE",
   5,
   "open",
   "c:\sample.txt");

application.shellExecute(
   "file:///c:/sample.doc",
   5,
   "open",
   "");
```

The last two exemples do exactly the same, when winword is the default application for opening .doc documents.

3.1.23 Function GetProfileInt

Name

```
VB  Sub GetProfileInt(section as string, entry as string,
           defaultValue as long) as long
JS  int getProfileInt(section: string, entry: string,
           defaultValue: int)
```

Retrieves the value of *entry* as long from the WinIBW3 preferences from the key *Section* for the current user. If the value is not present, *default* is returned.

3.1.24 Function GetProfileString

Name

```
VB  Sub GetProfileString(section as string, entry as string,
           defaultValue as string) as string
JS  string getProfileString(section: string, entry: string,
           defaultValue: string)
```

Retrieves the value of *entry* as string from the WinIBW3 preferences from the key *Section* for the current user. If the value is not present, *default* is returned.

3.1.25 Function WriteprofileInt

Name

```
VB  Sub WriteProfileInt(section as string, entry as string,
           value as long) as Boolean
JS  bool writeProfileInt(section: string, entry: string, value: int)
```

Stores the value of *entry* as long in the WinIBW3 preferences for the key *Section* for the current user.

3.1.26 Function WriteprofileString

Name

```
VB  Sub WriteProfileString(section as string, entry as string,
           value as string) as Boolean
JS  bool writeProfileString(section: string, entry: string,
           value: string)
```

Stores the value of *entry* as string in the WinIBW3 preferences for the key *Section* for the current user.

3.1.27 Function DownloadToFile

Name

```
VB  Sub downloadToFile(strSourceInURL as string,
           strDestination_localFile as string) as Boolean
JS  bool downloadToFile(strSourceInURL: string,
           strDestination_localFile: string)
```

Downloads source in URL to the local file.

3.1.28 Function DisableScreenUpdate

Name	
VB	**`Sub disableScreenUpdate(disableScrUpdate as Boolean)`**
JS	**`Void disableScreenUpdate (disableScrUpdate: bool)`**

Disable the update of the main screen with **`disableScrUpdate 'true' and enable`** the update of the main screen with **`disableScrUpdate 'false'`**.

3.1.29 Function GetCommandLineContent

Name	
VB	**`Sub GetCommandLineContent() as string`**
JS	**`string getCommandLineContent ()`**

Get the current content of the command line.

3.1.30 Function CascadeWindows

Name	
VB	**`Sub CasecadeWindows()`**
JS	**`Void casecadeWindows ()`**

Cascade windows.

3.1.31 Function TileWindowsVertical

Name	
VB	**`Sub TileWindowsVertical()`**
JS	**`Void tileWindowsVertical ()`**

Tile windows vertical.

3.1.32 Function TileWindowsHorizontal

Name	
VB	**`Sub TileWindowsHorizontal ()`**
JS	**`Void tileWindowsHorizontal ()`**

Tile windows horizontal.

3.2 The `WindowCollection` object

The WindowCollection object is returned from the `windows` property of the Application object.

3.2.1 Attribute Count

Name		type	access
VB	**Count**	**Long**	**[get]**
JS	**count**	**int**	**[get]**

Specifies the number of items contained in this collection.

3.2.2 Function Item

Name		type	access
VB	**Item(index as long)**	**WindowObject**	**[get]**
JS	**IWindow item(index: int)**	**IWindow**	***function***

Returns the WindowObject at index *index* contained in this collection.

3.2.3 Function GetWindowSnapshot

Name	
VB	**Sub GetWindowSnapshot() as WindowSnapshot**
JS	**WindowSnapshot getWindowSnapshot()**

Gets a snapshot of all currently open windows. You can restore this state by a call to *restoreWindowSnapshot*.

3.2.4 Function RestoreWindowSnapshot

Name	
VB	**Sub RestoreWindowSnapshot(snapShot as WindowSnapshot)**
JS	**void restoreWindowSnapshot(snapShot: WindowSnapshot)**

Restores the snapshot of open windows taken by a call to *getWindowSnapshot*.

3.3 The Window object

The `window` object is returned by the `Item` property of the `WindowsCollection` object.

3.3.1 Attribute Text

Name		type	access
VB	**Text**	**String**	**[get/set]**
JS	**text**	**string**	**[get/set]**

The Window text of this window object. The meaning of this property is dependent on the type of the window object.

3.3.2 Attribute Visible

Name		type	access
VB	**Visible**	**Boolean**	**[get/set]**
JS	**visible**	**bool**	**[get/set]**

Specifies if the WindowObject is visible. You can hide a window object by setting this property to False.

3.3.3 Attribute WindowID

Name		type	access
VB	`WindowID`	`Long`	`[get]`
JS	`windowID`	`int`	`[get]`

Specifies the ID used to access this WindowObject.

3.3.4 Function Activate

	Name
VB	`Sub Activate`
JS	`void activate()`

Brings the window to the front and gives it keyboard focus.

3.3.5 Function Close

	Name
VB	`Sub Close`
JS	`void close()`

Closes this window object.

3.3.6 Function Minimize

	Name
VB	`Sub Minimize`
JS	`void minimize()`

Minimizes this window object.

3.3.7 Function Maximize

	Name
VB	`Sub Maximize`
JS	`void maximize()`

Maximizes this window object.

3.3.8 Function Restore

	Name
VB	`Sub Restore`
JS	`void restore()`

Restores the size of the window object.

3.4 The `ActiveWindow` object

The `ActiveWindow` object is returned from the `activeWindow` property of the `application` object.

3.4.1 Attribute Caption

Name		type	access
VB	Caption	String	[get/set]
JS	caption	string	[get/set]

Specifies the caption (title) of the active window. You can get and set this property. By assigning a value to this property, the title of the active window can be changed.

3.4.2 Attribute ClipBoard

Name		type	access
VB	ClipBoard	String	[get/set]
JS	clipBoard	string	[get/set]

Specifies the content of the ClipBoard. You can get and set this property. By assigning a value to this property, the assigned value is placed on the ClipBoard. Although this is a property of the active window, the ClipBoard is a Windows global resource.

3.4.3 Attribute CommandLine

Name		type	access
VB	CommandLine	String	[set]
JS	commandLine	string	[set]

By assigning a value to this property or calling this function, the supplied value is pasted into the command line.

3.4.4 Attribute MaterialCode

Name		type	access
VB	MaterialCode	String	[get]
JS	materialCode	string	[get]

Returns the material code of the current title (content of 002@) if known, an empty string otherwise.

3.4.5 Attribute Messages

Name		type	access
VB	Messages	MessageCollection	[get]
JS	messages	MessageCollection	[get]

Contains the collection of messages as received from the server or added by scripts. Refer to sections 3.6 and 3.7 for more details about the MessageCollection and Message objects.

3.4.6 Attribute CodeData

Name		type	access
VB	CodedData	Boolean	[get/set]
JS	codedData	bool	[get/set]

Specifies if CodedData is switched on or not for the active window. You can toggle CodedData by assigning True or False to this property.

3.4.7 Attribute NoviceMode

Name		type	access
VB	NoviceMode	Boolean	[get/set]
JS	noviceMode	bool	[get/set]

Specifies if NoviceMode is switched on or not for the active window. You can toggle NoviceMode by assigning True or False to this property.

3.4.8 Attribute Status

Name		type	access
VB	Status	String	[get]
JS	status	string	[get]

Contains the content of the standard Pica3 variable "/V".
Note: In contrast to some other properties of ActiveWindow, the status is stored as a real property of the window, i.e. each window will keep its status property even if there are multiple connections.

3.4.9 Attribute Title

Name		type	access
VB	Title	TitleObject	[get]
JS	title	TitleObject	[get]

Returns the TitleObject of the active window if present, or nothing if not. Refer to TitleObject for a description of the methods and properties of this object.

3.4.10 Attribute TitleCopyfile

Name		type	access
VB	TitleCopyfile	string	[get/set]
JS	titleCopyfile	string	[get/set]

Specifies the path to the currently active TitleCopy file, relative to that of the WinIBW executable.. Although this is a property of the active Window, this setting is WinIBW global. You can set the TitleCopy file by assigning the relative path of a file to this property.

3.4.11 Attribute WindowID

Name		Type	access
VB	WindowID	Long	[get]
JS	windowID	int	[get]

Returns the windowID of the active window.

3.4.12 Attribute Variable

Name	type	access
VB Variable(name as string)	**string**	**[get/set]**
JS variable(name: string)	**string**	**[get/set]**

NOTE: When using this property in standard scripts, you have to use a different syntax!

```
string getVariable(name: string);
void   setVariable(name: string, value: string);
```

Via this attribute, users can access to all of the variables described in the tables of the following sub-section, with the `name` of P3GXX, here XX is Var-ID's in these tables.

Besides P3GXX, there are also special variables such as "scr", "buf", which can be accessed by users.

e.g. application.activeWindow.variable("P3GPP") will get the current PPN,

application.activeWindow.variable("P3GSE") will get the current title set number,

and

application.activeWindow.variable("scr") will get the screen code of the current screen.

3.4.12.1 Accessible Variables

This chapter lists all variables sent between WinIBW3 and CBS, and accessible from both standard and user scripts via the attribute 'variable' describe in the section 3.4.12.

CBS <-> WinIBW3 Variables

The following variables are the context variables that may be set/changed by both WinIBW3 and CBS. Each of them is listening to changes sent by the other component.

Var-ID	Variable Function
A*	Current ILL request number
PR	Current title presentation format
SD	Current SDI request number
SE	Current title set number
TA	Current language code
TI	Current title number

WinIBW3 -> CBS Variables

The following variables are initiated and changed by WinIBW3. CBS will accept and react to them as needed.

Var-ID	Variable Function

!K	Copy-signal OBSOLETE; variable was sent whenever a record was copied by WinIBW3 itself.
!S	Version number IBW
!T	Terminaltype
!U	Unicode support flag
!C	Character set

CBS -> WinIBW3 Variables
The following variables may be set/changed only by CBS. WinIBW3 just receives and reacts on them.

Var-ID	Variable Function
#S	Host identification
/V	Standardized Pica3 Status
!A	Cursor address
!P	Cursor position
!R	Cursor line
!V	Cursor field
!X	Purge Cache
#C	Character set

CBS -> WinIBW3 Variables
WinIBW3 uses the following variables sent by CBS to compose a configurable menu caption:

Var-ID	Variable Function
CN	CBS host name
SY	System name
BE	Database name
E07	Screen name (“login”, “database selection”, etc) This variable is part of a screen definition and therefore not a ‘real’ variable. Note: The E07 can be accessed via ‘scr’ instead of ‘P3GE07’, when using the attribute ‘variable’ describe in the section 3.4.12.
PP	PPN
UK	User ID (‘key’)

UM	User name
UL	User library ID
UB	User library name

3.4.13 Function CloseWindow

Name	
VB	**`Sub CloseWindow`**
JS	**`void closeWindow()`**

Closes the active window.

3.4.14 Function AppendMessage

Name	
VB	**`Sub AppendMessage(message as string, style as long)`**
JS	**`void appendMessage(message: string, style: int)`**

Appends *message* to the message bar with the given style. Possible values are:
1: Shows an error message
2: Shows a warning
3: Show a notification

3.4.15 Function ShowMessage

Name	
VB	**`Sub ShowMessage(message as string, style as long)`**
JS	**`void showMessage(message: string, style: int)`**

Shows *message* in the message bar with *style* as style. Possible values are:
1: Shows an error message
2: Shows a warning
3: Show a notification

Existing messages are overwritten.

3.4.16 Function Command

Name	
VB	**`Sub Command(command as string [,inNewWindow as Boolean = False])`**
JS	**`void command(command: string, inNewWindow: bool)`**

Executes the command as specified in *command* in the active window. If *inNewWindow* is set to True, the command will be executed in a newly open window. In user scripts of both VBScript and JavaScript, *inNewWindow* is optional and defaults to False.

3.4.17 Function GetLastCommand

Name	
VB	**`Sub getLastCommand() as string`**
JS	**`string getLastCommand()`**

Return the last command from the command history.

3.4.18 Function CopyTitle

Name	
VB	**`Sub CopyTitle`**
JS	**`void copyTitle()`**

Executes the title copy function in the same way as “TitleCopy” on the Edit menu. If there is no title available in the active window, the function does nothing

3.4.19 Function PressButton

Name	
VB	**`Sub PressButton(button as VARIANT)`**
JS	**`void pressButton(button: string)`**

Takes either a string or a number as parameter. Invoking with a number will invoke the button on the button bar starting with 1, i.e. `PressButton(1)` will simulate a click on the first button in the button bar. Invoking with a string will invoke the button with the label specified by string, i.e. `PressButton("Invoer")` will invoke the button with the label "Invoer".

3.4.20 Function SimulateIBWKey

Name	
VB	**`Sub SimulateIBWKey(key as string)`**
JS	**`void simulateIBWKey(key: string)`**

Simulates pressing a WinIBW key, where *strKey* is the internal presentation of the key. Commonly used keys are:

"FE"	Escape
"FR"	Enter
"F1" to "F12"	Usually related to the buttons on the Pica button bar.

These keys are actually defined in the screen definition and are the same as used by DosIBW.

3.4.21 Function ProcessURL

Name	
VB	**`Sub ProcessURL (url as string)`**
JS	**`void processURL(url: string)`**

Opens a URL specified as *url.*

3.4.22 Function FindString

Name	
VB	**`Sub FindString (what as string) as Boolean`**
JS	**`bool findString(what: string)`**

The function is only used for the long presentation mode.
Search for *what* in the current long presentation title. If *what* occurs in the current long presentation title, the return value is true. Otherwise, the return value is false. In case that the active window is not in the long presentation mode, the return value is false.

3.4.23 Function FindTagContent

Name	
VB	**`Sub FindTagContent (tag as string [, occurrence as long = 0] [, includeTag as Boolean = true]) as string`**
JS	**`string findTagContent(tag: string, occurrence: int, includeTag: bool)`**

The function is only used for the long presentation mode.
Returns the content of tag *tag* with occurrence *occurrence*. The first occurrence has number 0, the second has number 1, and so on. If *includeTag* is set to True, the tag is included in the returned string. In user scripts of both VBScript and JavaScript, the following parameters are optional:

- *occurrence:* default value is 0,

- *includeTag:* default value is True.

Although the data type of the *tag* is 'string', it must be a number string indicating a tag. If *tag* is specified with a text string other than a number string, it will not be recognized and an empty string will be returned as a result. E.g. the following example will return an empty string:

```
FindTagContent("country")
```

The following example will return the content of the first occurrence of the tag "5500", including the tag:

```
FindTagContent("5500")
```

The following example will return the content of the first occurrence of the tag "5500", excluding the tag:

```
FindTagContent("5500", false)
```

The following example will return the content of the second occurrence of the tag "5500", including the tag:

```
FindTagContent("5500", 1)
```

3.5 The `Title` object

The `title` object is returned from the `title` property of the `activeWindow` object.

3.5.1 Attribute CanCopySelection

Name		type	access
VB	CanCopySelection	Boolean	[get]
JS	canCopySelection	bool	[get]

Returns true if text is selected and can be copied.

3.5.2 Attribute CanCutSelection

Name		type	access
VB	CanCutSelection	Boolean	[get]
JS	canCutSelection	bool	[get]

Returns true if text is selected and can be cut.

3.5.3 Attribute CanPaste

Name		type	access
VB	CanPaste	Boolean	[get]
JS	canPaste	bool	[get]

Returns true if text can be pasted.

3.5.4 Attribute CanUndo

Name		type	access
VB	CanUndo	Boolean	[get]
JS	canUndo	bool	[get]

Returns true if the last operation can be undone.

3.5.5 Attribute CanRedo

Name		type	access
VB	CanRedo	Boolean	[get]
JS	canRedo	bool	[get]

Returns true if the last undone operation can be redone.

3.5.6 Attribute SelEnd

Name		type	access
VB	SelEnd	Long	[get/set]
JS	selEnd	int	[get/set]

Cursor position after the last character of the selected text.

3.5.7 Attribute SelStart

Name		type	access
VB	SelStart	Long	[get/set]
JS	selStart	int	[get/set]

Cursor position before the first character of the selected text.

3.5.8 Attribute CurrentField

Name		type	access
VB	CurrentField	string	[get]
JS	currentField	string	[get]

Returns the current field, i.e. the current tag plus contents at the current cursor position.

3.5.9 Attribute CurrentLineNumber

Name		type	access
VB	**`CurrentLineNumber`**	**`Long`**	**`[get]`**
JS	**`currentLineNumber`**	**`int`**	**`[get]`**

Returns the index of the current line.

3.5.10 Attribute Tag

Name		type	access
VB	**`Tag`**	**`string`**	**`[get]`**
JS	**`tag`**	**`string`**	**`[get]`**

The tag of the line, where the cursor is positioned.

3.5.11 Attribute Selection

Name		type	access
VB	**`Selection`**	**`string`**	**`[get]`**
JS	**`selection`**	**`string`**	**`[get]`**

The currently selected text.

3.5.12 Attribute TagAndSelection

Name		type	access
VB	**`TagAndSelection`**	**`string`**	**`[get]`**
JS	**`tagAndSelection`**	**`string`**	**`[get]`**

The selected text prefixed by the tag of where the selection occurs.

3.5.13 Function CharLeft

Name	
VB	**`Sub CharLeft([count as long = 1 [, select as Boolean = False]])`**
JS	**`void charLeft(count: int, select: bool)`**

Moves the cursor *count* characters to the left. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.14 Function CharRight

Name	
VB	**`Sub CharRight([count as long = 1 [, select as Boolean = False]])`**
JS	**`void charRight(count: int, select: bool)`**

Moves the cursor *count* characters to the right. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.15 Function CharLeft

Name	
VB	**`Sub Cut`**
JS	**`void cut()`**

Cuts the selected text and places it on the clipboard.

3.5.16 Function Copy

Name	
VB	`Sub Copy`
JS	`void copy()`

Copies the selected text to the clipboard.

3.5.17 Function CopyToFile

Name	
VB	`Sub CopyToFile(fileName as string) as Boolean`
JS	`bool copyToFile(fileName: string)`

Equivalent to CopyToFile on the Edit menu.

Note that if the parameter *fileName* contains backslashes, these must be escaped with a second backslash in the Javascript variant:

copyToFile("C:\\temp\\output.txt");

This does not apply to Visual Basic scripting.

3.5.18 Function DeleteLine

Name	
VB	`Sub DeleteLine([count as long = 1])`
JS	`void deleteLine(count: int)`

Deletes *count* lines in the current title at the current cursor position. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1.

3.5.19 Function DeleteSelection

Name	
VB	`Sub DeleteSelection()`
JS	`void deleteSelection()`

Deletes the currently selected text from this title.

3.5.20 Function DeleteToEndOfLine

Name	
VB	`Sub DeleteToEndOfLine`
JS	`void deleteToEndOfLine()`

Deletes the text from the current cursor position until the end of the line.

3.5.21 Function DeleteWord

Name	
VB	`Sub DeleteWord([count as long = 1])`
JS	`void deleteWord(count)`

Deletes *count* words at the cursor position. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1.

3.5.22 Function EndOfBuffer

Name

```
VB  Sub EndOfBuffer([select as Boolean = False])
JS  void endOfBuffer(select: bool)
```

Moves the cursor to the end of the buffer. If *select* is True, the text is selected. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.23 Function EndOfField

Name

```
VB  Sub EndOfField([select as Boolean = False])
JS  void endOfField(select: bool)
```

Moves the cursor to the end of the current field. If *select* is True, the text is selected. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.24 Function Find

Name

```
VB  Sub Find(what as string [, caseSensitive as Boolean = False
         [, lineOnly as Boolean = False
         [, wholeWord as Boolean = False]]]) as Boolean
JS  bool find(what: string, caseSensitive: bool, lineOnly: bool,
         wholeWord: bool)
```

Searches for *what* in the current title and marks the string if found. If *caseSensitive* is set to True, the search is case sensitive. If *lineOnly* is set to True, the search is performed in the current line only. If wholeWord is set to True, the search matches whole words only. In user scripts of both VBScript and JavaScript, *caseSensitive*, *lineOnly* and *wholeWord* are optional and default to False.

3.5.25 Function FindTag

Name

```
VB  Sub FindTag(tag as string [, occurrence as Number = 0
         [, includeTag as Boolean = True
         [, moveToPosition as Boolean = False
         [, removeIgnoredText as Boolean = False]]]) as string
JS  string findTag(tag: string, occurrence: short, includeTag: bool,
         moveToPosition: bool, removeIgnoredText: bool)
```

Returns the content of tag *tag* with occurrence *occurrence*. The first occurrence has number 0, the second has number 1, and so on. If *includeTag* is set to True, the tag is included in the returned string. If *moveToPosition* is set to True, the found text is selected until the end of the line. The tag is included in the selection depending on the value of *includeTag*. If *removeIgnoredText* is set to True, text marked as 'ignored' is removed from the result. *tag* may be specified truncated at the end.
In user scripts of both VBScript and JavaScript, the followings are optional:

- *Occurrence:* default value is 0,
- *includeTag:* default value is True,
- *moveToPosition:* default value is False,
- *removeIgnoredText:* default value is False.

The following example will return the content of the first tag starting with 47, including the tag. The found text including the tag is selected until the end of the line:
FindTag("47", 0, True, True)

3.5.26 Function FindTag2

Name

```
VB  Sub FindTag2(tag as string [, occurrence as Number = 0
            [, includeTag as Boolean = True
            [, moveToPosition as Boolean = False
            [, removeIgnoredText as Boolean = False]]]) as string
JS  string findTag2(tag: string, occurrence: short,
            includeTag: bool, moveToPosition: bool,
            removeIgnoredText: bool)
```

Returns the content of tag *tag* with occurrence *occurrence*. The first occurrence has number 0, the second has number 1, and so on. If *includeTag* is set to True, the tag is included in the returned string. If *moveToPosition* is set to True, the cursor will be moved to the beginning of the found tag if *includeTag* is set to True or the beginning of the content of the found tag if *includeTag* is set to False. If *removeIgnoredText* is set to True, text marked as 'ignored' is removed from the result. *tag* may be specified truncated at the end.
In user scripts of both VBScript and JavaScript, the followings are optional:

- *Occurrence:* default value is 0,
- *includeTag:* default value is True,
- *moveToPosition:* default value is False,
- *removeIgnoredText:* default value is False.

The following example will return the content of the first tag starting with 47, including the tag. The cursor will be moved to the beginning of the found tag:
FindTag2("47", 0, True, True)

3.5.27 Function InsertText

Name

```
VB  Sub InsertText(text as string)
JS  void insertText(text: string)
```

Inserts *text* at the cursor position or replace the currently selected texts with *text*.

3.5.28 Function InsertText2

Name	
VB	**`Sub InsertText2(text as string)`**
JS	**`void insertText(text: string)`**

Inserts *text* at the cursor position or in the front of the currently selected texts.

3.5.29 Function JoinLines

Name	
VB	**`Sub JoinLines()`**
JS	**`void joinLines()`**

Joins the current line with the following line.

3.5.30 Function LineDown

Name	
VB	**`Sub LineDown([count as long = 1 [, select as Boolean = False]])`**
JS	**`void lineDown(count: int, select: bool)`**

Moves the cursor *count* lines down. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.31 Function LineUp

Name	
VB	**`Sub LineUp([count as long = 1 [, select as Boolean = False]])`**
JS	**`void lineup(count: int, select: bool)`**

Moves the cursor *count* lines up. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.32 Function PageDown

Name	
VB	**`Sub PageDown([count as long = 1 [, select as Boolean = False]])`**
JS	**`void pageDown(count: int, select: bool)`**

Moves the cursor *count* pages down. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.33 Function PageUp

Name	
VB	**`Sub PageUp([count as long = 1 [, select as Boolean = False]])`**
JS	**`void pageDown(count: int, select: bool)`**

Moves the cursor *count* pages up. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.34 Function Paste

Name

```
VB  Sub Paste
JS  void paste()
```

Pastes the content of the clipboard at the current cursor position or selection.

3.5.35 Function PasteTitle

Name

```
VB  Sub PasteTitle
JS  void pasteTitle()
```

Executes the paste title function in the same way as “Paste Title” on the Edit menu. If there is no title available in the active window, the function does nothing

3.5.36 Function Redo

Name

```
VB  Sub Redo
JS  void redo()
```

Redo the last undone operation.

3.5.37 Function Replace

Name

```
VB  Sub Replace(replacingText as string)
JS  void replace(replacingText: string)
```

Replaces the currently selected text with *replacingText*. If no text is selected, the *replacingText* will be inserted at the cursor position.

3.5.38 Function Replace2

Name

```
VB  Sub Replace2(replacingText as string)
JS  void replace2(replacingText: string)
```

Replaces the currently selected text with *replacingText*. If no text is selected, no replacing takes place.

3.5.39 Function ReplaceAll

Name

```
VB  Sub ReplaceAll(search as string, replacingText  as string
          [, caseSensitive as Boolean = False
          [, wholeWord as Boolean = False]])
JS  void replaceAll(search: string, replacingText : string,
         caseSensitive: bool, wholeWord: bool)
```

Replaces all occurrences of *search* by *replacingText*. If *caseSensitive* is set to True, the search is performed case sensitive. If *wholeWord* is set to True, the search matches whole words only. In user scripts of both VBScript and JavaScript, *caseSensitive* and *wholeWord* are optional and default to False.

3.5.40 Function SelectAll

Name

```
VB  Sub SelectAll
JS  void selectAll()
```

Select the entire title.

3.5.41 Function SelectNone

Name

```
VB  Sub SelectNone
JS  void selectNone()
```

Select nothing.

3.5.42 Function SetSelection

Name

```
VB  Sub SetSelection(start as long, end as long
         [, scrollToPosition as Boolean = False])
JS  void setSelection(start: int, end: int, scrollToPosition: bool)
```

Sets the current selection to begin at the character position specified by *start* and to end at the character position specified by *end*. If *scrollToPosition* is set to True, the selection is scrolled into view. In user scripts of both VBScript and JavaScript, *scrollToPosition* is optional and defaults to False.

3.5.43 Function StartOfBuffer

Name

```
VB  Sub StartOfBuffer([select as Boolean = False])
JS  void startOfBuffer(select: bool)
```

Moves the cursor to the start of the buffer. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.44 Function StartOfField

Name

```
VB  Sub StartOfField([select as Boolean = False])
JS  void startOfField(select: bool)
```

Moves the cursor to the start of the current field, i.e. the current tag. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.45 Function Undo

Name

```
VB  Sub Undo
JS  void undo()
```

Undo the last operation.

3.5.46 Function WordLeft

Name	
VB	**Sub WordLeft([count as long = 1 [, select as Boolean = False]])**
JS	**void wordLeft(count: int, select: bool)**

Moves the cursor *count* words to the left. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.5.47 Function WordRight

Name	
VB	**Sub WordRight([count as long = 1 [, select as Boolean = False]])**
JS	**void wordRight(count: int, select: bool)**

Moves the cursor *count* words to the right. If *select* is set to True, the text is selected. In user scripts of both VBScript and JavaScript, *count* is optional and defaults to 1. In user scripts of both VBScript and JavaScript, *select* is optional and defaults to False.

3.6 The `MessageCollection` object

The `MessageCollection` object is returned from the `messages` property of the active window.

3.6.1 Attribute Count

Name		type	access
VB	**Count**	**Long**	**[get]**
JS	**count**	**int**	**[get]**

Specifies the number of items contained in this collection.

3.6.2 Attribute Item

Name		type	access
VB	**Item(index as long)**	**Message object**	**[get]**
JS	**IMessage item(index: int)**	**IMessage**	***function***

Returns the Message object at index *index* contained in this collection. Refer to The Message object for a description of the properties of the Message object. The items in the collections are numbered starting from 0.

The following VBScript example shows how to use both properties of the Messages collection of the Application.ActiveWindow:

```
Sub ShowFirstMessage()
   If Application.ActiveWindow.Messages.Count > 0 Then
      MsgBox Application.ActiveWindow.Messages.Item(0).Text
   End If
End Sub
```

3.7 The `Message` object

The `Message` object is contained in the `MessageCollection` returned from the `message` property of the active window.

3.7.1 Attribute Text

Name		type	access
VB	**Text**	**String**	**[get]**
JS	**text**	**string**	**[get]**

Returns the text of the message.

3.7.2 Attribte Type

Name		type	access
VB	**Type**	**Integer**	**[get]**
JS	**Type**	**int**	**[get]**

Returns the type of the message.

The following VBScript example shows how to access all messages in the Application.ActiveWindow.Messages collection:

```
Sub ShowMessages()
   Dim msg
   For Each msg In Application.ActiveWindow.Messages
      MsgBox msg.Text
   Next
End Sub
```

The following JavaScript example shows how to access all messages in the collection:

```
function ShowMessages() {
   count = application.activeWindow.messages.count;
   for (i = 0; i < count; i++) {
      msg = application.activeWindow.messages.item(i);
      application.messageBox("message", msg.text, "");
   }
}
```

3.8 The `WindowSnapshot` object

The `WindowSnapshot` object is returned from the `getWindowSnapshot` function of the WindowCollection object. This object has no properties or methods. Its sole purpose is to store a snapshot of windows.

4 Utility objects

For file input and output, and prompting and alerting the user during execution of scripts, three utility objects are available. These objects have to be declared before they can be used. This declaration looks like this:

```
const utility = {
   newFileInput: function() {
      return Components.classes["@oclcpica.nl/scriptinputfile;1"]
        .createInstance(Components.interfaces.IInputTextFile);
   },

   newFileOutput: function() {
      return Components.classes["@oclcpica.nl/scriptoutputfile;1"]
        .createInstance(Components.interfaces.IOutputTextFile);
   },

   newPrompter: function() {
      return Components.classes["@oclcpica.nl/scriptpromptutility;1"]
        .createInstance(Components.interfaces.IPromptUtilities);
   }
};
```

After this declaration, the prompter object can be used as follows:

```
var thePrompter = utility.newPrompter();
thePrompter.alert("test", "hallo");
thePrompter.alertCheck("test", "hallo", "Check me", false);
```

```
var thePrompter = utility.newPrompter();
var theAnswer = thePrompter.select("Select", "Please make a selection",
        "One\nTwo\nThree\nFour");

if (! theAnswer) {
   // user canceled the dialog
   ...
   return;
}

if (theAnswer == "One") {
   // user chose one
   ...
} else if (theAnswer == "Two") {
   // user chose two
   ...
}
```

An alternative approach to declare only a FileInput object, could be:

```
var fileInput = Components.classes["@oclcpica.nl/scriptinputfile;1"]
                       .createInstance(Components.interfaces.IInputTextFile);
```

The following sections describe all functions for the three utility objects.

4.1 Prompter object

After declaring a prompter object as described above, the following methods are available:

Name

```
JS  void alert(dialogTitle: string, text: string)
```

Puts up an alert dialog box with an OK button.

Name

```
JS  bool alertCheck(dialogTitle: string, text: string,
           checkMsg: string, defaultCheck: bool)
```

Puts up an alert dialog with an OK button and a message with a checkbox. Returns if the checkbox was checked.

Name

```
JS  bool confirm(dialogTitle: string, text: string)
```

Puts up a dialog with OK and Cancel buttons. Returns true for OK, false for Cancel.

Name

```
JS  bool confirmCheck(dialogTitle: string, text: string,
           checkMsg: string, defaultCheck: bool)
```

Puts up a dialog with OK and Cancel buttons, and a message with a single checkbox. Returns true for OK, false for Cancel. The value of the checkbox must be retrieved via the getCheckValue() method of this object.

Name

```
JS  int confirmEx(dialogTitle: string, text: string,
           button0Title: string, button1Title: string,
           button2Title: string, checkMsg: string, defaultCheck: bool)
```

Puts up a dialog with up to 3 buttons. Button 0 will be the default button. If you want to ommit a button or the checkbox, set it to null or an empty string. The value of the checkbox must be retrieved via the getCheckValue() method of this object.
Returns the index of the button pressed (starting with 0).
When the buttons texts are one of “OK”, “CANCEL”, “YES”, “NO”, “SAVE”, “DONTSAVE” or “REVERT”, the texts displayed on the dialog are the localized texts.

Name

```
JS  int prompt(dialogTitle: string, text: string, value: string,
           checkMsg: string, defaultCheck: bool)
```

Puts up a dialog with an edit field and an optional checkbox.
The edit field is pre-filled with *value*.
Returns true for OK, false for Cancel.
The value of the checkbox must be retrieved via the getCheckValue() method of this object. The value of the editfield must be retrieved via the getEditValue() method of this object.

Name

```
JS  int promptPassword(dialogTitle: string, text: string,
           password: string, checkMsg: string, defaultCheck: bool)
```

Puts up a dialog with a password field and an optional checkbox.
The edit field is pre-filled with *value*.
Returns true for OK, false for Cancel.
The value of the checkbox must be retrieved via the getCheckValue() method of this object. The value of the password must be retrieved via the getEditValue() method of this object.

Name

```
JS  int select(dialogTitle: string, text: string,
            selectList: string)
```

Puts up a dialog box which has a list box of strings.
Returns null, if the dialog was canceled, the selected item otherwise.
The strings must be separated by '\n'.

Name

```
JS  int getCheckValue()
```

Returns the value of the checkBox presented in some dialogs (set on successfull return of the dialog).

Name

```
JS  int getEditValue()
```

Returns the value of the editbox presented in some dialogs (set on successfull return of the dialog).

Name

```
JS  void setDebug(debug: bool)
```

This is for debugging purposes only. If set to true, the script will show error-messages on exceptions.

4.2 FileInput object

Name

```
JS  bool open(fileName: string)
```

Opens the file with absolute path *fileName*.

Name

```
JS  bool openSpecial(theDirName: string, theRelativePath: string)
```

Constructs a fileName with a special WinIBW directory and attempts to open it.
theDirName must be one of the following predefined strings:

"dwlfile"	the users download file; *theRelativePath* will be ignored.
"prnfile"	the users print file; *theRelativePath* will be ignored.
"BinDir"	the main WinIBW directory; *theRelativePath* must point to a valid file in a valid subdirectory.
"ProfD"	the user's profile directory.

Name

```
JS  bool openViaGUI(dialogTitle: string, initialPath: string,
            defaultName: string, aFilter: string,
            theFilterName: string)
```

Shows a FileOpenDialog and initialize the TextFileInput with the chosen file. Returns false if the dialog was canceled.
defaultName is the default file name. *aFilter* specifies the types of files to display (e.g. "*.txt; *.doc").
theFilterName is the displayed name of the filter (e.g. "My Input Files").

Name

JS bool remove()
Try to delete this file.

Name

JS void close()
Closes the file.

Name

JS string readLine()
Reads a line of text; returns null for EOF or error.
Note that readLine() should be used (instead of read()), if the file contains utf-8 data.

Name

JS string read(nrOfBytes: int)
Reads atmost the maximum number of bytes; NOTE: these are bytes and the read string will contain the bytes in "raw form".
Note that readLine() should be used (instead of read()), if the file contains utf-8 data.

Name

JS int getAvailable()
Returns the number of bytes available in the stream.

Name

JS bool isEOF()
Return if the TextFileInput is at EOF.

Name

JS string getPath()
Returns the full path of the file; returns null on error.

Name

**JS string getSpecialPath(theDirName: string,
theRelativePath: string)**
Returns a special WinIBW Path; maybe useful for openViaGUI. This function is static, so you do not need a TextFileInput object to use it.
theDirName must be one of the following predefined strings:

"dwlfile"	the users download file; *theRelativePath* will be ignored.
"prnfile"	the users print file; *theRelativePath* will be ignored.
"BinDir"	the main WinIBW directory; *theRelativePath* must point to a valid file in a valid subdirectory.
"ProfD"	the user's profile directory.

When *theRelativePath* is supplied it will be appended to theDirName taking care of using '\' as required.

Name

JS void setDebug(debug: bool)
This is for debugging purposes only. If set to true, the script will show error-messages on exceptions.

4.3 FileOutput object

Name

`JS bool create(fileName: string)`

Creates the file with absolute path *fileName*.

Name

`JS bool createSpecial(theDirName: string, theRelativePath: string)`

Constructs a fileName with a special WinIBW directory and attempts to create it.
theDirName must be one of the following predefined strings:

"dwlfile"	the users download file; *theRelativePath* will be ignored.
"prnfile"	the users print file; *theRelativePath* will be ignored.
"BinDir"	the main WinIBW directory; *theRelativePath* must point to a valid file in a valid subdirectory.
"ProfD"	the user's profile directory.

Name

```
JS  bool createViaGUI(dialogTitle: string, initialPath: string,
         defaultName: string, aFilter: string,
         theFilterName: string)
```

Shows a FileOpenDialog and initialize the TextFileInput with the chosen file. Returns false if the dialog was canceled.
defaultName is the default file name. *aFilter* specifies the types of files to display (e.g. "*.txt; *.doc").
theFilterName is the displayed name of the filter (e.g. "My Input Files").

Name

`JS bool remove()`

Try to delete this file.

Name

`JS void close()`

Closes the file.

Name

`JS bool writeLine(theLine: string)`

Writes a line of text (a \n is appended automatically); returns false on error.

Name

`JS bool write(theText: string)`

Writes *theText* to the file; returns false on error.

Name

`JS void setTruncate(truncate: bool)`

By default files are created in append-mode, i.e. text written to the file will be appended to any already existing content. If you set *truncate* to true, the file content will be erased prior to writing. You may set *truncate* to true before or after calling any of the create-functions, but be aware that any content present in the file will be erased.

Name

`JS string getPath()`

Returns the full path of the file; returns null on error.

Name

JS string getSpecialPath(theDirName: string, theRelativePath: string)

Returns a special WinIBW Path; maybe useful for openViaGUI. This function is static, so you do not need a TextFileInput object to use it.
theDirName must be one of the following predefined strings:

"dwlfile"	the users download file; *theRelativePath* will be ignored.
"prnfile"	the users print file; *theRelativePath* will be ignored.
"BinDir"	the main WinIBW directory; *theRelativePath* must point to a valid file in a valid subdirectory.
"ProfD"	the user's profile directory.

When *theRelativePath* is supplied it will be appended to theDirName taking care of using '\' as required.

Name

JS void setDebug(debug: bool)

This is for debugging purposes only. If set to true, the script will show error-messages on exceptions.

5 XUL programming

5.1 Introduction

Standard scripts can be extended with dialog boxes. These dialog boxes are created using XUL, the Mozilla XML User Interface Language. This document does not provide an extensive course into XUL programming; for that purpose we refer to books such as:

- Creating Applications with Mozilla, Boswell e.a. (O'Reilly)
- Rapid Application Development with Mozilla, Nigel McFarlane (Prentice Hall)

5.2 Example dialog box

The following example code may serve as a starting point to create custom dialog boxes with. This dialog box contains a collection of the most commonly used controls, such as an input field, checkboxes, radio buttons and a pull down list.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<?xml-stylesheet href="chrome://global/skin/global.css" type="text/css"?>
<!-- Include your own stylesheet here. -->

<!-- Define the dialog: -->
<dialog
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
   title="Sample dialog box presenting different elements"
   onload="onLoad();"
   buttons="accept,cancel"
   ondialogaccept="return onAccept();"
   ondialogcancel="return onCancel();"
   onunload="application.activate();"
   style="min-width: 33em"
   id="SampleDialog"
>

<script>
<![CDATA[
var application = Components.classes["@oclcpica.nl/kitabapplication;1"]
                        .getService(Components.interfaces.IApplication);

function onLoad()
{
   // Whatever needs to be done when the SampleDialog opens.
   return true;
}

function onAccept()
{
   // The Accept button is pressed..
   var theTextBox = document.getElementById("idSampleTextbox").value;
   application.messageBox("SampleDlg", "You entered: '" + theTextBox + "'",  "");
   return true;
}

function onCancel()
{
   // The Cancel button is pressed..
   alert("You pressed Cancel. Nothing will be done with the data.");
   return true;
}

]]>
```

```
</script>

   <!-- Layout the dialog controls: -->
   <vbox>

      <separator/>
      <hbox align="center" flex="1">
         <label align="left,bottom" value="Sample textbox to enter text:" />
         <textbox align="right" id="idSampleTextbox" value="Initial value" />
      </hbox>

      <separator/>
      <groupbox>
         <caption label="Sample checkboxes" />
         <checkbox id="idSampleCheckbox1"
                         label="First checkbox option" checked="true" />
         <checkbox id="idSampleCheckbox2" label="Second checkbox option" />
         <checkbox id="idSampleCheckbox3" label="Third checkbox option" />
      </groupbox>

      <groupbox>
         <caption label="Sample radio group" />
         <radiogroup id="idSampleRadiogroup" orient="horizontal">
            <radio id="idRadio1" group="idSampleRadiogroup"
                   label="First radio button" checked="true" />
            <radio id="idRadio2" group="idSampleRadiogroup"
                   label="Second radio button" />
         </radiogroup>
      </groupbox>

      <groupbox>
         <caption label="Sample menu list" />
         <menulist id="idSampleMenulist" >
            <menupopup>
               <menuitem value="mi-one"   label="Menu Item One"/>
               <menuitem value="mi-two"   label="Menu Item Two"/>
               <menuitem value="mi-three" label="Menu Item Three"/>
               </menupopup>
         </menulist>
      </groupbox>

   </vbox>
   <separator/>

</dialog>
```

5.3 Calling a XUL dialog box in WinIBW

The XUL and Javascript code that constitute a XUL dialog box usually reside in one or two files in the **\chrome\ibw\content\xul** directory under the WinIBW root directory. Some extra code is needed to activate the dialog box, in the form of a standard script function. This function (included in one of the existing standard script files, or in a separate standard script file) must be included in the WinIBW setup as well.

Assuming that the XUL dialogbox is defined in a file called ExampleDialogBox.xul, the code to activate this dialog box looks like:

```
function open_xul_dialog(theUrl, theFeatures, theArguments)
{
   // try to get the window-watcher
   var ww     = Components.classes["@mozilla.org/embedcomp/window-watcher;1"]

       .getService(Components.interfaces.nsIWindowWatcher);
   if (!ww) {
      // no chance, give up
      return false;
   }

   // let's try to get a valid parent
   var theParent = ww.activeWindow;

   var features = null;
   if (theFeatures != null) {
      features = theFeatures;
   } else {
      // you may choose to remove some of the features
      // you may also want to specify width=xxx and/or height=xxx
      features = "centerscreen,chrome,close,titlebar,resizable,modal,dialog=yes";
   }

   // it doesn't matter, if we don't have a parent
   // we just use the active window, whether its null or not
   ww.openWindow(theParent, theUrl, "", features, theArguments);
}

function OpenExampleDialogBox()
{
   open_xul_dialog("chrome://ibw/content/xul/ExampleDialogBox.xul", null);
}
```

In the WinIBW user interface, the function OpenExampleDialogBox() can then be attached to a shortcut key or a menu item or toolbar button.

5.4 Access to the WinIBW object model

The Javascript environment inside a XUL dialog box differs slightly from the environment for the standard Javascript functions. The standard functions have direct access to the WinIBW object model, via the **application** object. In a XUL dialog box, this objects needs to be declared first, before it can be used:

```
var application = Components.classes["@oclcpica.nl/kitabapplication;1"]
                        .getService(Components.interfaces.IApplication);
```

After that, the complete WinIBW object model is at the programmer's disposal.

5.5 Access to the utility objects

For the same reason as in the previous section, the utility objects for file input and output and prompts need to be declared before they can be used:

```
const utility = {
   newFileInput: function() {
      return Components.classes["@oclcpica.nl/scriptinputfile;1"]
        .createInstance(Components.interfaces.IInputTextFile);
   },

   newFileOutput: function() {
      return Components.classes["@oclcpica.nl/scriptoutputfile;1"]
        .createInstance(Components.interfaces.IOutputTextFile);
   },

   newPrompter: function() {
      return Components.classes["@oclcpica.nl/scriptpromptutility;1"]
        .createInstance(Components.interfaces.IPromptUtilities);
   }
};
```

5.6 Preferences and environment variables

WinIBW3 no longer uses the registry to store user preferences and other settings. Instead, the Mozilla preferences-service is used.

There are two ways to read and write preferences. The first is with a set of functions in the WinIBW object model, all part of the **application** object (see section 3.1):

- getProfileInt
- writeProfileInt
- getProfileString
- writeProfileString

These functions are modeled after the old registry functions in WinIBW2, and can be used to migrate existing code to WinIBW3. There is no variant for boolean values, but this can be overcome by using integer values 0 an 1, or even string values “true” and “false”.

5.6.1 Mozilla preferences service

It is also possible to use the Mozilla preferences-service directly, for instance when the WinIBW object model is not used within XUL javascript functions. The following example demonstrates how to do this:

```
// Declare a (global) variable to access the preferences service:
const thePrefs =
   Components.classes["@mozilla.org/preferences-service;1"]
   .getService(Components.interfaces.nsIPrefBranch);

// Write values to the user preferences file:
thePrefs.setBoolPref("winibw.doit", true);
thePrefs.setCharPref("winibw.dowhat", "phoneHome");
thePrefs.setIntPref("winibw.howlong", 10);

// Read values from the user preferences file:
var bDoIt = thePrefs.getBoolPref("winibw.doit“);
var sDoWhat = thePrefs.getCharPref("winibw.dowhat“);
var iHowLong = thePrefs.getIntPref("winibw.howlong“);
```

Values are always written to the file **user_prefs.js** in the users profile directory. When reading preferences, WinIBW first tries user-prefs.js, and if it can’t find the specified preference there, it tries setup.js.

5.6.2 Mozilla environment interface

With the Mozilla environment interface it is possible to read and write environment variables. The following code illustrates this:

```
// Declare a variable to access the environment:
const theEnv = Components.classes["@mozilla.org/process/environment;1"]
               .getService(Components.interfaces.nsIEnvironment);

// Read an environment variable after checking that it exists:
if (theEnv.exists("ENV_VAR_NAME")) {
   someVariable = theEnv.get("ENV_VAR_NAME");
}
```

This is a function that can be used to expand strings that contain environment variables enclosed in % - characters (e.g. %PATH%).

```
// Simple helper function to expand environment variables
// variables not found will be replaced by an empty string
function expandEnvironmentVariables(theURL)
{
   if (theURL == null) return null;
   // our URL is url-encoded; a % will be represented as %25
   var theMatches = theURL.split(/(%25.*?)%25/g);

   if (theMatches == null) {
      // nothing to do
      return theURL;
   }

   for (var i = 0; i < theMatches.length; i++) {
      if (theMatches[i].search(/^%25/) == 0) {
       // looks like an environment variable
       theMatches[i] = theEnv.get(theMatches[i].substr(3));
      }
   }

   return theMatches.join("");
}
```

5.7 Tips and tricks

5.7.1 Reloading XUL dialogs without restarting WinIBW

WinIBW3 contains the function **ClearScreenCache**, located in the category Special. This function is by default not accessible from the user interface, but is very handy when developping XUL dialogs. When this function is attached to, for example, the shortcut key combination CTRL+ALT+ENTER, changes to the XUL dialog (and associated Javascript code) can be reloaded without having to restart WinIBW3, by simply pressing this key combination.
Note that for reloading the standard script functions there is another function available, also located in the category Special: **ReloadStandardScripts**.

5.7.2 Hide functions from the user interface

If you have functions in the standard script files that you don't want to show up in the Customization dialog box, there are two methods to accomplish this.
The first method is to add a dummy parameter to the function. WinIBW3 shows only functions with an empty parameter list in the UI, and skips all functions with one or more parameters. You don't need to reference the parameter in the function itself.

```
function privateHelper(dummyparameter)
{
   // Do things...
}
```

Another method is to prepend the function with one or more underscores:

```
function _privateHelper()
{
   // Do things...
}
```

5.7.3 Conditionally enable standard script functions

It is possible to disable and enable standard script functions depending on certain conditions. A common example is functions that can only be used during cataloging. Such functions can be disabled when there is no title editor on the screen.
To accomplish this, create a second function with the same name as the original one, and prepend it with two underscores. This new function should return false when the associated user interface element must be disabled, and true when it must be enabled.

```
function __catalogingFunction() // note the two underscores!
{
   if (!applicaton.activeWindow.title) return false;
   return true;
}

function catalogingFunction ()
{
   // Do things...
}
```

WinIBW3 does not show the function with the underscores, so only the function without underscores can be attached to the user interface. When displaying the menu, WinIBW3 executes the second function (if it can find one), and enables or disables the UI element according to the return value. If there is no such function, the UI element is always enabled.

A disadvantage of this method is that the user does not get information why a certain function cannot be used. If the function is always enabled, it can start with a similar check, and display a message informing the user why the function cannot be used at that moment.

5.7.4 Keep WinIBW on the foreground when closing a XUL dialog

Use `application.activate()` to avoid an inconveniency in WinIBW when using message boxes, after which WinIBW will not be in the foreground when the XUL dialog box is closed. Simply add an **unload** event handler to the <dialog> tag in XUL:

```
<dialog
   xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
   title="ExampleDialogBox"
   onload="onLoad();"
   buttons="accept, cancel"
   ondialogaccept="return onAccept();"
   ondialogcancel="return onCancel();"
   onunload="application.activate();"
   ...
>
```

5.7.5 Error tracking

try - catch

In non-trivial functions it is a good practice to trap potential errors with the try {} catch {} construction.
Example:

```
function complexStuff ()
{
   try {
      do complex stuff...
   }
   catch (e) { alert(e); }
}
```

Trace window
When the Javascript code contains syntactical errors, such as a missing end parenthesis, this can be very difficult to locate, because the function is simply not loaded when the XUL dialog is opened. In such cases the Trace dialog can be of help, because certain errors in the XUL or Javascript code are reported here.

6 VBScript versus JavaScript

Although WinIBW3 uses JavaScript for its standard scripts, this does not mean that browser specific Document Object Models are supported. E.g. you can not execute code like `window.location = 'http://www.oclcpica.nl';` or `document.write('hello world!');`. For the objects and their methods and properties available in standard scripts please refer to The WinIBW3 Application Object Model.

6.1 Case sensitivity

One big difference between JavaScript and VBScript is that JavaScript is case-sensitive, whereas VBScript is not. This means, that you have to type variables, functions, keywords etc. exactly as they are defined. In JavaScript `myVariable` and `MyVariable` are treated as two different variables, whereas in VBScript they refer to the same.
By convention JavaScript uses mixedCase. This means that variables, properties and functions usually start with a lowercase letter and use capital letters only at word boundaries inside the name, e.g. `addSyntaxColor` or `beginOfPage`.

6.2 Comments

VBScript only knows single-line comments. In VBScript comments may be started with either a ' (i.e. a single quote) or the string `rem`. They start with the just mentioned characters and end at the end of the line.
JavaScript knows single-line comments as well. They are denoted by `//` (i.e. two slashes) and also end at the end of the line.
Besides from single-line comments JavaScript also knows multi-line comments. They are started with `/*` (i.e. a slash followed by an asterisk) and end with a `*/`. It is not allowed to nest comments.

6.3 Statements

In VBScript statements always end at the end of a line. If you want to continue a statement on a second line, you have to terminate the line with an underscore.
In JavaScript statements are terminated by a semicolon.

6.4 Functions

VBScript knows functions and subs. Functions return a value, subs don't. Functions are called with parentheses, subs without. In VBScript subs and functions have the general syntax of:

```
Sub SubName([Parameters])
        ' some code
End Sub

Function FunctionName([Parameters])
        ' some code
        FunctionName = <the return value>
End Function
```

JavaScript only knows functions. They may or may not return a value. They are always called with parentheses. In JavaScript functions have the general syntax of:

```
function functionName([parameters]) {
        // some code
        [return <the return value>;]
    }
```

As you can see, JavaScript uses braces instead of the `End` keyword. To have a function return a value, you simple use `return`.

WinIBW supports optional parameters in its Application Object Model for VBScript. If you do not supply an optional parameter in a function call, WinIBW automatically supplies the default value. E.g. the following two VBScript calls are equivalent:

```
    Application.ActiveWindow.Command "\log user password", False
    Application.ActiveWindow.Command "\log user password"
```

The second parameter of the `Command` function (specifying if the command is to be executed in a second window) is optional. If you omit it, WinIBW will use the default, which is `False` in this case.
In JavaScript, WinIBW does not support optional parameters, i.e. you must specify all of them. If you do not supply all parameters, the script engine will report a runtime error. For example the call:

```
    application.activeWindow.command("\\log user password", false);
```

is ok, whereas:

```
    application.activeWindow.command("\\log user password");
```

will generate a runtime error.

6.4.1 A simple example

Here is a simple sample of a script function, which performs a login to the system.

VBScript:

```
Sub LoginTest()
    ' send a log command to the server
    Application.ActiveWindow.Command "\log user password", False
End Sub
```

JavaScript:

```
function LoginTest() {
    // send a log command to the server
    application.activeWindow.command("\\log user password", false);
}
```

Please note, that the "\\" is not a typo; refer to Strings for more information.

6.5 Strings

6.5.1 VBScript

Scripts in VBScript are encoded in ISO-Latin-1 and VBScript treats all string literals as ISO-Latin-1 encoded.

In VBScript scripts, a Unicode character can be specified in a string with the following 3 ways:

- By `ChrW(ddddd)`, here `ddddd` is the decimal digits for a specific Unicode character. E.g. ChrW(20320) for the Chinese word “你”.

  Note: `ChrW(ddddd)` can be used for both VBScript and the WinIBW3 Application Object Model.
- By escape sequences in string literals like “`\uhhhh`”, here `hhhh` is four hexadecimal digits for a specific Unicode character. E.g “`\u4F60`” for the Chinese word “你”.

  Note: The escape sequences in string literals like “`\uhhhh`” can be only used for the WinIBW3 Application Object Model.
- From WinIBW3 version 3.4 above, simply just by “你” via Copy-Paste action in the WinIBW3 embedded user script editor.

Note: In VBScript scripts, when you pass a string of the form “`\uhhhh`” to a standard function provided by VBScript, you may get unexpected results, e.g. asking the length of the string “`\u4F60`” via `len("\u4F60")` will get `6`, since VBScript treats this kind of string simply as the sequence of the characters it contains.

In VBScript scripts, Strings have to be enclosed in double quotes.

In VBScript scripts, strings are concatenated with the `&` operator.

In VBScript scripts, strings can be compared with each other by means of the following operators:

- ‘=’, equal to
- ‘<=’, less than or equal to
- ‘>=’ , greater than or equal to
- ‘<>’ , not equal to

For examples, the followings are valid values for the strings in VBScript scripts:

`ChrW(20320) & ChrW(22909) & " !"`

`"\u4F60\u597D !"` (In case of being used for the WinIBW3 Application Object Model)

`"你好 !"` (From WinIBW3 version 3.4 above)

6.5.2 JavaScript

In JavaScript scripts, ISO-Latin-1encoded strings can be used, and Unicode characters can be specified via escape sequences in string literals. They all will be converted on the fly at the run-time. *Table 1* lists the available escape sequences in JavaScript:

Character sequence	Meaning
`\b`	Backspace
`\f`	Form-Feed
`\n`	Newline
`\r`	Carriage Return
`\t`	Tabulator
`\'`	Single quote, which does not terminate the string
`\"`	Double quote, which does not terminate the string
`\\`	A single backslash
`\xhh`	An ISO-Latin-1 character, specified by the two hexadecimal digits `hh`
`\uhhhh`	A Unicode character, specified by the four hexadecimal digits `hhhh.` E.g "`\u4F60`" for the Chinese word "你".

Table 1

Besides, a Unicode character can also be specified in a string of JavaScript **user** scripts with the following way:

- From WinIBW3 version 3.4 above, simply just by "你" via Copy-Paste action in the WinIBW3 embedded user script editor.

In JavaScript scripts, the length of the Unicode string "`\u4F60`" (ARABIC LETTER HAMZA ISOLATED FORM) will be indicated as `1`, because the string contains exactly one Unicode character.

In JavaScript scripts, strings may be enclosed in double or in single quotes.

In JavaScript scripts, strings are concatenated with the + operator.

In JavaScript scripts, strings can be compared with each other by means of the following operators:

- '`==`' , equal to
- '`<=`', less than or equal to
- '`>=`' , greater than or equal to
- '`!=`' , not equal to

(refer to Operators for more information about operators).

For examples, the followings are valid values for the strings in JavaScript scripts:

```
"\u4F60\u597D !"
```

```
'\u4F60' + '\u597D !'
```

`"你好 !"` (in case in JavaScript **user** scripts, and from WinIBW3 version 3.4 above)

6.6 Messages

6.6.1 VBScript

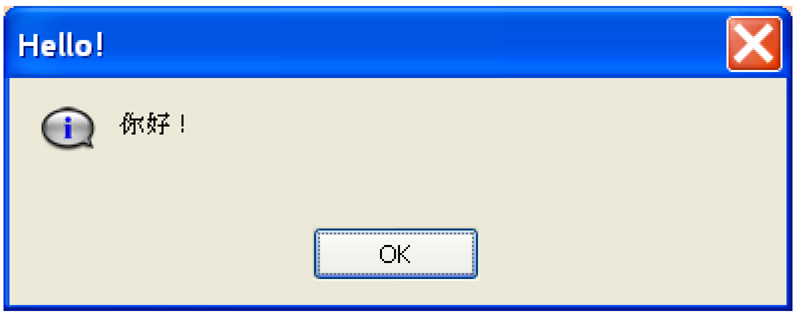
If you want to present a message containing Unicode characters to the user in a user script of VBScript, there are a number of ways to do so. For examples, a call to one of the following 3 statements:

```
application.messageBox "Hello!", "\u4F60\u597D !", ""

application.messageBox "Hello!", ChrW(20320) & ChrW(22909) & " !", ""

application.messageBox "Hello!", "你好 !", ""
```

will display the following message box to the user:

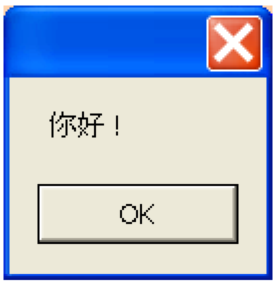


and a call to one of the following 2 statements:

```
msgBox ChrW(20320) & ChrW(22909) & " !"

msgBox "你好 !"
```

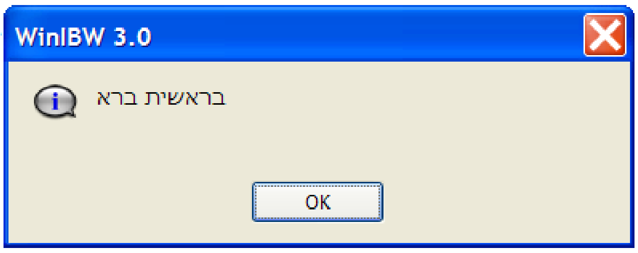
will display the following message box to the user:



A call to

```
Application.MessageBox "WinIBW 3.0",
"\u05D1\u05E8\u05D0\u05e9\u05d9\u05EA \u05D1\u05e8\u05d0",
"message-icon"
```

will display:


WinIBW 3.0

בראשית ברא

As you (perhaps) can see, the Hebrew text also flows correctly from the right to the left.

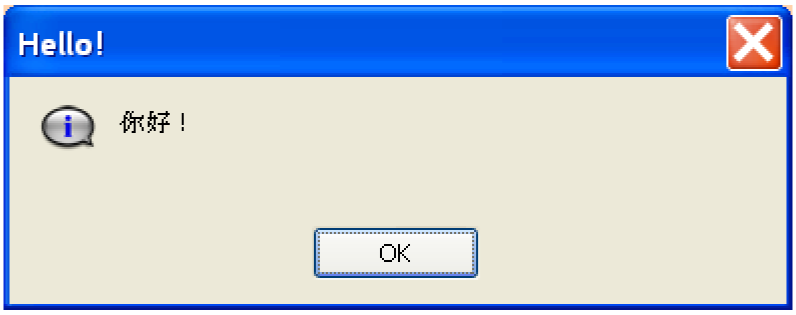
6.6.2 JavaScript

If you want to present a message containing Unicode characters to the user in JavaScript scripts, there are 2 ways. A call to one of the following 2 statements:

```
application.messageBox ("Hello!", "\u4F60\u597D !", "");
```

```
application.messageBox ("Hello!", "你好 !", "");
```
(in case in JavaScript **user** scripts)

will display the following message box to the user:


Hello!

你好！

Note: many JavaScript references list the `alert` function as global function to show message boxes. Actually the `alert` function is not a global function, but a member of the `window` object. You can thus not use it in a standard script.

6.7 Variables

In VBScript variables are declared with `dim`.
In JavaScript you use `var` instead. JavaScript also allows you to optionally initialize variables in the declaration.
Example:

```
var i;                    // just declaration
var x = 0;                // declaration and initialization
var msg = “Hello world!”; // declaration and initialization
```

6.8 Operators

Here is an overview of the most important operators in JavaScript and the differences to VBScript. For a complete description of all JavaScript operators, please refer to the JavaScript documentation. These are some of the many interesting online resources about JavaScript:

- http://www.webreference.com/javascript/reference/core_ref/
- http://developer.mozilla.org/en/docs/JavaScript

- http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/
- http://javascript.internet.com/tutorials/

	Meaning	Description	Example

`=`	Assignment	Assigns a value to a variable.	`i = 0;`
			`msg = "Hello";`

`==`	Comparison	Compares if the two operands are equal. Please note, that this is different from VBScript, where the comparison operator is `=` (i.e. the same as the assignment operator).
		If you accidentally use the assignment operator in a condition, the condition will always evaluate to true.
		I.e. the condition `if (i = 0)` will always be true!
			`if (i == 0) {`
			`  i = 34;`
			`}`

`&&`	Logical and	Checks if both operands are true. Please note, that this is different from VBScript, where you use `AND`.
		Please note also the remarks about Testing conditions below.
			`if (a && b){`
			`  doIt();`
			`}`

`||`	Logical or	Checks if at least one of the two operands is true. Please note, that this is different from VBScript, where you use `OR`.
		Please note also the remarks about Testing conditions below.
			`if ( a || b){`
			`  doIt();`
			`}`

`+`	Addition	Performs an addition of the two operands. This operator is also used for string concatenation, as opposite to VBScript where `&` is used for that case.	`var i = 3 + 4;`
			`msg = "hello " + "world";`

`++`	Post- or Pre-Increment	Increments the operand with one.
		The pre-increment operator is placed before the operand, the post-increment operator is placed after the operand.
		The difference between post-increment and pre-increment is, that pre-increment will first increment the value and return the incremented value, whereas the post-increment operator will return the unmodified value and increment it afterwards.
		This operator does not exist in VBScript.
			`var i = 0;`
			`var x = 0;`
			
			`// do a post-increment`
			`x = i++; // i is now 1`
			`         // x is still 0`
			
			`x = 0;`
			`i = 0;`
			
			`// do a pre-increment`
			`x = ++i; // i is now 1`
			`         // x is now 1`

`--`	Post- or Pre-	Decrements the operand with one.	`var i = 10;`

	decrement	The pre-decrement operator is placed before the operand, the post-decrement operator is placed after the operand.
		The difference between post-decrement and pre-decrement is, that pre-decrement will first decrement the value and return the decremented value, whereas the post-decrement operator will return the unmodified value and decrement it afterwards. This operator does not exist in VBScript.
			`var x = 10;`
			
			`// do a post-decrement`
			`x = i--; // i is now 9`
			`         // x is still 10`
			
			`x = 10;`
			`i = 10;`
			
			`// do a pre-decrement`
			`x = --i; // i is now 9`
			`         // x is now 9`

+=
-=
*=
etc.
	Assignment with operation	Performs the operation corresponding to the first character of the operator (i.e. +, -, *) on the left operand and right operand and assigns the result to the left operand. This operator does not exist in VBScript.	`var i = 10;`
			
			`i += 5; // i is now 15`
			`// this is the same as`
			`// i = i + 5;`
			
			`i -= 3; // i is now 12`
			`// this is the same as`
			`// i = i - 3;`
			
			`// you can also use the`
			`// += operator on strings`
			`var msg = "hello ";`
			`msg += "world";`
			
			`// msg now contains`
			`// "hello world"`

6.9 Testing conditions

JavaScript support the following `if/else` constructs. These are very similar to the corresponding VBScript constructs:

```
if (expression) {
        // statements
}

if (expression) {
        // statements
} else {
        // statements
}

if (expression) {
        // statements
} else if (expression) {
      // statements
} else {
      // statements
}
```

There is however one big difference in evaluation of the boolean expressions. VBScript does not use short circuit evaluation; JavaScript does.
Short circuit evaluation means, that the evaluation of a boolean expression stops as soon, as it is known, that further evaluations can not change the result of the already performed evaluations.

This gets most clear in an example:

```
Function returnTrue()
      MsgBox "returnTrue() called"
      returnTrue = True
End Function

Sub Test
      If returnTrue() OR returnTrue() Then
            MsgBox "Expression is true"
      End If
End Sub
```

If you call the `Test` sub, you will see two times the message "returnTrue() called". When you have a closer look at the IF-Statement in the `Test` sub, it will be clear, that after the first call to `returnTrue`, the IF-Statement can not evaluate to False anymore. (Even if the second call to `returnTrue()` would return False, the IF-Statement still would evaluate to True.)
Even if it is no longer necessary to evaluate the second `returnTrue()` to get the result of the IF-Statement, VBScript will call it. This is known as non-short-circuit evaluation.

If we code the same functions in JavaScript, it may look like this:

```
function returnTrue() {
      application.messageBox("WinIBW 3.0",
            "returnTrue() called", "message-icon");
      return true;
}

function test() {
      if (returnTrue() || returnTrue()) {
            application.messageBox("WinIBW",
                  "Expression is true", "message-icon");
      }
}
```

If you call the `test()` function, you will only see one time the message "returnTrue() called". As you already know, the second call to `returnTrue()` is not necessary to evaluate the if-statement, so JavaScript will not perform it. This is known as short-circuit evaluation.

Short-circuit evaluation is very useful. On one hand it is faster (because it does not perform superfluous evaluation) on the other hand it is practical. Have a look at the following code snippet:

```
        if (application.activeWindow && application.activeWindow.title
                && application.activeWindow.title.canPaste()) {
                application.activeWindow.title.paste();
        }
```

This code works perfectly well in JavaScript. If you don't have an `activeWindow`, the evaluation stops. If you have an `activeWindow`, but no title, the evaluation stops; in this case it is not tried to access the `canPaste()` function of the title object.
If you try to do the same in VBScript and you do not have an `activeWindow` or a `title` object, you will receive a runtime error, because the VBScript engine will try to access objects, which are null.

On the other hand you have to watch out in JavaScript, not to use code like:

```
        if (initialize() && performAction() && deinitialize()) {
                // wow, everything worked fine
        }
```

You probably don't want `performAction()` to be called, when the `initialize()` function returned false; so far it's ok. However, if the `performAction()` function returns false, your `deinitialize()` function will not be called and that's probably not what you want.
Please have also a look at Error handling to see, how you can handle this more elegantly.

6.10 Error handling

VBScript is not very smart as far as error handling is concerned. All you can do is use an `on error resume next` statement, to prevent runtime errors.
JavaScript has built-in support for exceptions.

The general syntax is:

```
        try {
                // statements
        }
        catch (argument) {
                // statements
        }
        finally {
                // statements
        }
```

The `catch` and `finally` blocks are optional, but either `catch` or `finally` must be present.

JavaScript first executes the statements in the `try` block. If any of these statements throws an exception, the code in the `catch` block is executed (if present) immediately, i.e. the following statements in the `try` block are not executed. The `catch` block receives the thrown exception as argument. After that, the `finally` block is executed (if present). The `finally` block will always be executed, i.e. both when an exception is thrown and when no exception is thrown.

To go back to our example in Testing conditions we could do the following:

```
function initialize() {
	// do the required initialization
	if (something_went_wrong) {
		throw "failed to start the flux compensator";
	}
}

function performAction() {
	// perform the actions
	if (something_went_wrong) {
		throw "failed to perform actions";
	}
}
```

You could now do easily, something like:

```
try {
	initialize();
	performAction();
}
catch (exception) {
	application.messageBox("WinIBW",
			"something went wrong: " + exception,
			"alert-icon");
}
finally {
	deinitialize();
}
```

The script will first invoke your `initialize()` function. If the `initialize()` function does not throw an exception, the `performAction()` function will be called. If the `initialize()` function throws an exception, the code in the `catch` block will be executed (and the `performAction()` function will be omitted). If the `performAction()` function throws an exception, the code in the `catch` block will be executed. If neither `initialize()` nor `performAction()` throw an exception, the `catch` block will not be executed at all. Finally, the code in the `finally` block will be executed. This happens in any case, i.e. when an exception was thrown and when no exception was thrown.
For more information on exceptions please have a look at the JavaScript reference.

6.11 Loops

JavaScript mainly support three loop constructs: `for, while, do/while`. For the other constructs please refer to the JavaScript reference.

The while loop is the simplest loop:

```
while (expression) {
        // statements
}
```

The statements will be executed, as long as expression is true.

The `do/while` loop is similar, but it first executes the statements and checks an expression afterwards to determine if the loop should go on, i.e. the statements will always be executed at least once.

```
do {
        // statements
} while (expression);
```

The most powerful is the `for` loop:

```
for (initializing; test; update) {
        // statements
}
```

The `for` loop first performs the statement indicated by `initializing`. This is usually initializing of a variable. It then evaluates the expression indicated by `test`. If `test` evaluates to true, the statements in the loop are executed. After that the statement indicated by `update` are performed and the loop continues evaluating `test` and so on.

Here are some examples:

```
// while loop
var i = 2;
while (i < 12) {
        application.messageBox("Alert", "We did not fix " + i
                        + " bugs", "message-icon");
        i++;
}
application.messageBox("Really", "No, we fixed " + i
                        + " bugs today!", "message-icon");


// do/while loop
var pwd = "";
do {
        pwd = ask_for_password();
} while (pwd != "the password");

// for loop
var i;

for (i = 0; i < 35; i++) {
        do_something(i);
}
```

```
// another for loop
var doryphore;

for (doryphore = 0; find_doryphore(); doryphore++) {
      kill_doryphore();
}
application.messageBox(“Yippieh!”,
                  “removed “ + doryphore + “ doryphores!”,
                  “message-icon”);
```

7 Changes in WinIBW3

7.1 An Overview of Changes in the Application Object Model

The following table lists the changes in the Application Object Model in WinIBW3 comparing with WinIBW2.

Object	Member	Change	Description
`application`	`newWindow()`	new	This function has been moved from `activeWindow`. Otherwise it would be impossible to open a new window from script, if all windows are closed.
`application`	`messageBox()`	new	Shows a messagebox that can display Unicode strings.
`application`	`NsizeX`	renamed	Renamed to Width.
`application`	`Height`	new	The width of the application window.
`application`	`onTimer()`	removed	This event is no longer fired.
`application`	`Timer`	removed	This event no longer exists.
`application`	`HTMLToPica()`	removed	WinIBW does no longer perform this kind of character set conversion.
`application`	`PicaToHTML()`	removed	WinIBW no longer performs this kind of character set conversion.
`application`	`HtmlToLatin()`	removed	WinIBW no longer performs this kind of character set conversion.
`application`	`LatinToHtml()`	removed	WinIBW no longer performs this kind of character set conversion.
`application`	`LatinToPica()`	removed	WinIBW no longer performs this kind of character set conversion.
`application`	`PicaToLatin()`	removed	WinIBW no longer performs this kind of character set conversion.
`application`	`TableToPica()`	removed	WinIBW no longer performs this kind of character set conversion.
`application`	`CreateAXDoc()`	removed	No longer supported
`application`	`FindOrCreateAXDoc()`	removed	No longer supported
`application`	`DisableScreenUpdate()`	removed	No longer supported

`application`	`DownloadToFile()`	added	No longer supported for user scripts for security reasons. In standard scripts you may use the built-in functionality to perform this action. From WinIBW 3.3.3 or later, the function is supported.
`application`	`GetMachineProfileString()`	removed	No longer supported due to security considerations.
`application`	`GetMachineProfileInt()`	removed	No longer supported due to security considerations.
`application`	`WriteMachineProfileString()`	removed	No longer supported due to security considerations.
`application`	`WriteMachineProfileInt()`	removed	No longer supported due to security considerations.
`application`	`CreateWAOToolbar()`	removed	No longer supported.
`application`	`DestroyWAOToolbar()`	removed	No longer supported.
`application`	`AddWAOToolbar()`	removed	No longer supported.
`application`	`RemoveWAOToolbar()`	removed	No longer supported.
`application`	`LoadWAO()`	removed	No longer supported.
`application`	`LCMField()`	removed	No longer supported.
`application`	`LoadWAO()`	removed	No longer supported.
`application`	`Pause()`	renamed	Renamed to PauseScript
`application`	`dict`	removed	No longer supported due to operating system dependencies.
`application`	`windows`	semantic	The windows collection is now always live, i.e. it reflects the current state of all windows. It may no longer be used to make a snapshot of the window state. Please refer to the `getWindowSnapshot` and `restoreWindowSnapshot` members of the `windowCollection` object.
`application`	`closeWindowsExcept();`	removed	Please refer to the `getWindowSnapshot` and `restoreWindowSnapshot` members of the `windowCollection` object instead.
`application`	`bOverWrite`	renamed	renamed to `OverwriteMode`
`activeWindow`	`PasteTitle()`	moved	moved to `title` object
`activeWindow`	`PageDown()`	removed	no longer supported
`activeWindow`	`PageUp()`	removed	no longer supported

activeWindow	LineDown()	removed	no longer supported
activeWindow	LineUp()	removed	no longer supported
activeWindow	BeginOfPage()	removed	no longer supported
activeWindow	EndOfPage()	removed	no longer supported
activeWindow	NewWindow();	removed	This function has been moved to the application object.
activeWindow	DocType	renamed	This member has been renamed to materialCode. The name DocType was not clear at all.
activeWindow	materialCode	renamed	This is the new name for the previous member DocType.
activeWindow	Messages	semantic	The messages collection is shared between all windows, i.e. compared to previous versions of WinIBW, each window refers to the same message collection.
activeWindow	SimulateIBWKey	changed	This function no longer returns a value.
activeWindow	CopyTitle	changed	This function no longer returns a value.
title	LSelStart	renamed	Renamed to SelStart
title	LSelEnd	renamed	Renamed to SelEnd
title	Cut()	new	Please refer to the documentation of the title object
title	Copy()	new	Please refer to the documentation of the title object
title	Paste()	new	Please refer to the documentation of the title object
title	PasteTitle()	new	Same as PasteTitle on the application object. Please refer to the documentation of the title object.
title	Undo()	new	Please refer to the documentation of the title object
title	Redo()	new	Please refer to the documentation of the title object
title	CanCutSelection	new	Please refer to the documentation of the title object

`title`	`CanCopySelection`	new	Please refer to the documentation of the `title` object
`title`	`CanPaste`	new	Please refer to the documentation of the `title` object
`title`	`CanUndo`	new	Please refer to the documentation of the `title` object
`title`	`CanRedo`	new	Please refer to the documentation of the `title` object
`title`	`SelectAll()`	new	Please refer to the documentation of the `title` object
`title`	`SelectNone()`	new	Please refer to the documentation of the `title` object
`title`	`GetCurrentField()` `CurrentField`	change	changed function `GetCurrentField()` to property `CurrentField`
`title`	`GetTagAndSelection()` `TagAndSelection`	change	changed function `GetTagAndSelection()` to property `TagAndSelection`
`title`	`GetSelection()` `selection`	change	changed function `GetSelection()` to property `Selection`
`title`	`GetTag()` `Tag`	change	changed function `GetTag()` to property `Tag`
`title`	`GetCurrentLine()` `CurrentLineNumber`	change	changed function `GetCurrentLine()` to property `CurrentLineNumber`
`Window`	`Activate()`	new	Brings the window to the front and gives it keyboard focus.
`windows`	`getWindowSnapshot()`	new	Please refer to the documentation of the `windowsCollection` object.
`windows`	`restoreWindowSnapshot()`	new	Please refer to the documentation of the `windowsCollection` object.
`message`	`text`	change	property changed to read-only
`message`	`type`	change	property changed to read-only

7.2 Scripting Interface Changes/Renames from WinIBW2 to WinIBW3

7.2.1 Scripting Interface Renames in the Application Object

WinIBW2	WinIBW3
Attribute NsizeX	Attribute Width
Attribute bOverWrite	Attribute OverwriteMode
function Pause ()	function PauseScript ()

7.2.2 Scripting Interface Renames in the ActiveWindow Object

WinIBW2	WinIBW3
Attribute DocType	Attribute MaterialCode

7.2.3 Scripting Interface Changes/Renames in the Title Object

WinIBW2	WinIBW3
function GetCurrentField ()	Attribute CurrentField
function GetCurrentLine ()	Attribute CurrentLineNumber
function GetTagAndSelection ()	Attribute TagAndSelection
function GetSelection ()	Attribute Selection
function GetTag ()	Attribute Tag
Attribute lSelStart	Attribute SelStart
Attribute lSelEnd	Attribute SelEnd

7.3 Conversion from WinIBW2 User Scripts to WinIBW3 User Scripts

In http://cbs.pica.nl:8080/winibw/wi2wi3.html, a web-based WinIBW2 Script to WinIBW3 script Conversion Utility created by OCLC can be found. This conversion utility can help you convert a WinIBW2 user script of VBScript to a WinIBW3 user script of VBScript easily. After pressing button 'Show Help' and following the instructions, you will get to know how to use this conversion utility.