

Application

Return Types

[DRAGNDROP](#) [JSDNDEvent](#) [JSEvent](#) [JSRenderEvent](#) [JSWindow](#) [APPLICATION_TYPES](#) [CLIENTDESIGN](#) [ELEMENT_TYPES](#) [LOGGINGLEVEL](#) [UICONSTANTS](#) [WEBCONSTANTS](#) [Renderable](#) [UUID](#)

Method Summary

void	addClientInfo (info) Adds a string of client information which gets stored on the server, and can be viewed on the Clients page of Servoy Server Administration Console.
void	beep () Produces a "beep" sound; commonly used to indicate an error or warning dialog.
Boolean	closeAllWindows () Close all visible windows (except main application window).
void	closeSolution () Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.
void	closeSolution (solutionToLoad) Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.
void	closeSolution (solutionToLoad, methodName) Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.
void	closeSolution (solutionToLoad, methodName, methodArgument) Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.
Boolean	createNewFormInstance (designFormName, newInstanceScriptName) Create a new form instance.
JSWindow	createWindow (windowName, type) Creates a new window that can be used for displaying forms.
JSWindow	createWindow (windowName, type, parentWindow) Creates a new window that can be used for displaying forms.
String	executeProgram (program) Execute a program and returns output.
String	executeProgram (program, params) Execute a program and returns output.
String	executeProgram (program, params, environmentVars) Execute a program and returns output.
String	executeProgram (program, params, environmentVars, startDir) Execute a program and returns output.
void	executeProgramInBackground (program) Execute a program in the background.
void	executeProgramInBackground (program, params) Execute a program in the background.
void	executeProgramInBackground (program, params, environmentVars) Execute a program in the background.
void	executeProgramInBackground (program, params, environmentVars, startDir) Execute a program in the background.
void	exit () Stop and exit application.
Number	getActiveClientCount (currentSolutionOnly) Get the active user count on the server (can be limited to current solution).
JSWindow	getActiveWindow () This gets the currently focused active window; this can be the main application window or a modal dialog.
Number	getApplicationType () Get the application type.
Number	getClientCountForInfo (info) Gets the count for all clients displaying the same additional information in the Clients page of Servoy Server Administration Console.
Object	getClientProperty (name) Sets a UI property.
String	getClipboardString () Gets a string from the clipboard, null if not a string or empty.
String	getCurrentLookAndFeelName () Gets the name of the current Look And Feel specified in Application Preferences.
String	getHostName () Get the name of the localhost.
String	getIPAddress () Get the clients' IP address.
String[]	getLicenseNames () Get the names of the used client licenses (as strings in array).

String	getOSName() Returns the name of the operating system.
String[]	getPrinters() Get all the printer names in an array.
Number	getScreenHeight() Get the screen height in pixels.
Number	getScreenWidth() Get the screen width in pixels.
Date	getServerTimeStamp() Returns a date object initialized on server with current date and time.
String	getServerURL() Gets the HTTP server url.
String	getSolutionName() Returns the name of the current solution.
Number	getSolutionRelease() Get the solution release number.
Date	getTimeStamp() Returns a date object initialized in client with current date and time.
UUID	getUUID() Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object.
UUID	getUUID(byteArray) Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object.
UUID	getUUID(uuidString) Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object.
String	getUserProperty(name) Get a persistent user property.
String[]	getUserPropertyNames() Get all persistent user property names.
Array	getValueListArray(name) Retrieve a valuelist as array, to get real-values for display-values.
Object	getValueListDisplayValue(name, realValue) Retrieve a valuelist display-value for a real-value.
JSDataset	getValueListItems(name) Get all values from a custom or database type value list as dataset (with columns displayValue,realValue).
String[]	getValueListNames() Get all the valuelist names as array.
String	getVersion() Returns the application version.
JSWindow	getWindow() Get the main application window.
JSWindow	getWindow(name) Get a window by window name.
Boolean	isInDeveloper() Returns true if the solution is running in the developer.
Boolean	isLastPrintPreviewPrinted() Check if the last printpreview did print.
void	output(msg) Output something on the out stream.
void	output(msg, level) Output something on the out stream.
void	overrideStyle(originalStyleName, newStyleName) Overrides one style (defined in in a form) with another.
void	playSound(url) Play a sound (AU file, an AIFF file, a WAV file, and a MIDI file).
Boolean	putClientProperty(name, value) Sets a UI property.
void	redo() Redo last action (if possible).
void	removeAllClientInfo() Removes all names given to the client via the admin page.
Boolean	removeClientInfo(info) Removes a string of client information which is stored on the server and previously was added using the application.
void	setClipboardContent(string) Sets a string object in the clipboard.
void	setNumpadEnterAsFocusNextEnabled(enabled) Set if numpad enter should behave like focus next.
void	setStatusText(text) Set the status area value.
void	setStatusText(text, tooltip) Set the status area value.
void	setToolbarVisible(name, visible) Make a toolbar visible or invisible.

void	setUserProperty(name, value) Sets a user property for this client: For heedless clients((including Batch Processors and Authentication clients) the user property is stored in memory and will be lost upon client restart.
void	setValueListItems(name, dataset) Fill a custom type valuelist with values from array(s) or dataset.
void	setValueListItems(name, dataset, autoconvert) Fill a custom type valuelist with values from array(s) or dataset.
void	setValueListItems(name, displayValues) Fill a custom type valuelist with values from array(s) or dataset.
void	setValueListItems(name, displayValues, autoconvert) Fill a custom type valuelist with values from array(s) or dataset.
void	setValueListItems(name, displayValues, realValues) Fill a custom type valuelist with values from array(s) or dataset.
void	setValueListItems(name, displayValues, realValues, autoconvert) Fill a custom type valuelist with values from array(s) or dataset.
Date	showCalendar() Show the calendar, returns selected date or null if canceled.
Date	showCalendar(dateFormat) Show the calendar, returns selected date or null if canceled.
Date	showCalendar(selectedDate) Show the calendar, returns selected date or null if canceled.
Date	showCalendar(selectedDate, dateFormat) Show the calendar, returns selected date or null if canceled.
String	showColorChooser() Show the colorChooser.
String	showColorChooser(colorString) Show the colorChooser.
String	showFontChooser() Show the font chooser dialog.
String	showFontChooser(defaultFont) Show the font chooser dialog.
void	showForm(form) Show the form specified by the parameter, that can be a name (is case sensitive!) or a form object.
String	showI18NDialog() Opens the i18n dialog so users can change translations.
String	showI18NDialog(keyToSelect) Opens the i18n dialog so users can change translations.
String	showI18NDialog(keyToSelect, languageToSelect) Opens the i18n dialog so users can change translations.
Boolean	showURL(url) Shows an URL in a browser.
Boolean	showURL(url, webclientTarget) Shows an URL in a browser.
Boolean	showURL(url, webclientTarget, timeout) Shows an URL in a browser.
Boolean	showURL(url, webclientTarget, webclientTargetOptions) Shows an URL in a browser.
Boolean	showURL(url, webclientTarget, webclientTargetOptions, timeout) Shows an URL in a browser.
void	sleep(ms) Sleep for specified time (in milliseconds).
void	undo() Undo last action (if possible).
void	updateUI()
void	updateUI(milliseconds)

Method Details

addClientInfo

void **addClientInfo** (info)

Adds a string of client information which gets stored on the server, and can be viewed on the Clients page of Servoy Server Administration Console.

The new piece of client information is added on behalf of the running Servoy client.

This function can be called more than once, if you want to add multiple lines of client information.

NOTE:

This function can also be used with the function `<em>getClientCountForInfo</em>` to count the number of clients with matching additional client information.

Parameters

{String} info - A line of text to be added as additional client information on behalf of the running Servoy client.

Returns

void

Sample

```
application.addClientInfo('SaaS company name');
application.addClientInfo('For any issues call +31-SA-AS');
```

beep

void **beep** ()

Produces a "beep" sound; commonly used to indicate an error or warning dialog.

Returns

void

Sample

```
application.beep();
```

closeAllWindows

Boolean **closeAllWindows** ()

Close all visible windows (except main application window). Returns true if operation was successful.

Returns

Boolean - Boolean true if all windows were closed and false otherwise.

Sample

```
var win = application.createWindow("aWindowName", JSWindow.WINDOW, null);
win.setInitialBounds(10, 10, 300, 300);
win.title = "This is a window";
controller.show(win);

var win2 = application.createWindow("anotherWindowName", JSWindow.WINDOW, null);
win2.setInitialBounds(100, 100, 300, 300);
win2.title = "This is another window";
controller.show(win2);

var qdialog = plugins.dialogs.showQuestionDialog("QuestionDialog", "Do you want to close the windows?", "Yes", "No");
if (qdialog == "Yes") {
    application.closeAllWindows();
    controller.show(null);
}
```

closeSolution

void **closeSolution** ()

Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.

If the user has been logged in, this function keeps the user logged in and in the newly open solution, the login is skipped and the solution goes straight to the first form.

If you want to go to a different url, you need to call application.showURL(url) before calling application.closeSolution() (this is only applicable for Web Client).

An alternative option is security.logout() which also does a log out for the user (for solutions that require authentication).

Returns

void

Sample

```
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
application.closeSolution();
//close current solution, open solution 'solution_name', call global method 'global_method_name' with
argument 'my_argument'.
//if the user has been logged in, he will stay logged in
//application.closeSolution('solution_name','global_method_name','my_argument');
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client
only/Web client only) type )
```

closeSolution

void **closeSolution** (solutionToLoad)

Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.

If the user has been logged in, this function keeps the user logged in and in the newly open solution, the login is skipped and the solution goes straight to the first form.

If you want to go to a different url, you need to call application.showURL(url) before calling application.closeSolution() (this is only applicable for Web Client).

An alternative option is security.logout() which also does a log out for the user (for solutions that require authentication).

Parameters

{String} solutionToLoad - Name of the solution to load

Returns

void

Sample

```
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
application.closeSolution();
//close current solution, open solution 'solution_name', call global method 'global_method_name' with
argument 'my_argument'.
//if the user has been logged in, he will stay logged in
//application.closeSolution('solution_name','global_method_name','my_argument');
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client
only/Web client only) type )
```

closeSolution

void **closeSolution** (solutionToLoad, methodName)

Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.

If the user has been logged in, this function keeps the user logged in and in the newly open solution, the login is skipped and the solution goes straight to the first form.

If you want to go to a different url, you need to call application.showURL(url) before calling application.closeSolution() (this is only applicable for Web Client).

An alternative option is security.logout() which also does a log out for the user (for solutions that require authentication).

Parameters

{String} solutionToLoad - Name of the solution to load

{String} methodName - Name of the global method to call

Returns

void

Sample

```
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
application.closeSolution();
//close current solution, open solution 'solution_name', call global method 'global_method_name' with
argument 'my_argument'.
//if the user has been logged in, he will stay logged in
//application.closeSolution('solution_name','global_method_name','my_argument');
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client
only/Web client only) type )
```

closeSolution

void **closeSolution** (solutionToLoad, methodName, methodArgument)

Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.
 If the user has been logged in, this function keeps the user logged in and in the newly open solution, the login is skipped and the solution goes straight to the first form.
 If you want to go to a different url, you need to call `application.showURL(url)` before calling `application.closeSolution()` (this is only applicable for Web Client).
 An alternative option is `security.logout()` which also does a log out for the user (for solutions that require authentication).

Parameters

`{String}` solutionToLoad - Name of the solution to load
`{String}` methodName - Name of the global method to call
`{Object}` methodArgument - Argument passed to the global method

Returns

void

Sample

```
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
application.closeSolution();
//close current solution, open solution 'solution_name', call global method 'global_method_name' with
argument 'my_argument'.
//if the user has been logged in, he will stay logged in
//application.closeSolution('solution_name','global_method_name','my_argument');
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client
only/Web client only) type )
```

createNewFormInstance

Boolean `createNewFormInstance` (designFormName, newInstanceScriptName)

Create a new form instance.

Parameters

`{String}` designFormName - Name of the design form
`{String}` newInstanceScriptName - Name of the new form instance

Returns

Boolean - Boolean (true) if the instance was created succesfully, (false) otherwise

Sample

```
var ok = application.createNewFormInstance('orders','orders_view');
if (ok)
{
    var dialog = application.createWindow("myDialog", JSWindow.DIALOG);
    dialog.show('orders_view')
    //forms['orders_view'].controller.show()
    //forms.xyz.elements.myTabPanel.addTab(forms['orders_view'])
    //forms['orders_view'].elements.mylabel.setLocation(10,20)
}
```

createWindow

JSWindow `createWindow` (windowName, type)

Creates a new window that can be used for displaying forms. Initially the window is not visible.

If there is already a window with the given name, it will be closed and destroyed prior to creating the new window.

Use the form controller `show()` and `showRecords()` methods in order to show a form in this window.

Parameters

`{String}` windowName - the name of the window.
`{Number}` type - the type of the window. Can be one of `JSWindow.DIALOG`, `JSWindow.MODAL_DIALOG`, `JSWindow.WINDOW`.

Returns

JSWindow - the newly created window.

Sample

```
// create and show a window, with specified title, initial location and size
// type of the window can be one of JSWindow.DIALOG, JSWindow.MODAL_DIALOG, JSWindow.WINDOW
// If parentWindow is not specified, the current window will be used as parent; parentWindow parameter is
// only used by dialogs
var win = application.createWindow("windowName", JSWindow.WINDOW);
win.setInitialBounds(10, 10, 300, 300);
win.title = "This is a window";
controller.show(win);
// create and show a non-modal dialog with default initial bounds/title
var nmd = application.createWindow("nonModalDialogName", JSWindow.DIALOG);
controller.showRecords(15, nmd); // 15 is a single-number pk in this case
```

createWindow

JSWindow createWindow (windowName, type, parentWindow)

Creates a new window that can be used for displaying forms. Initially the window is not visible.

If there is already a window with the given name, it will be closed and destroyed prior to creating the new window.

Use the form controller show() and showRecords() methods in order to show a form in this window.

Parameters

{String} windowName - the name of the window.

{Number} type - the type of the window. Can be one of JSWindow.DIALOG, JSWindow.MODAL_DIALOG, JSWindow.WINDOW.

{JSWindow} parentWindow - the parent JSWindow object. If it is not specified, the current window will be used as parent. This parameter is only used by dialogs.

Returns

JSWindow - the newly created window.

Sample

```
// create and show a window, with specified title, initial location and size
var win = application.createWindow("windowName", JSWindow.WINDOW);
win.setInitialBounds(10, 10, 300, 300);
win.title = "This is a window";
controller.show(win);
// create and show a non-modal dialog with default initial bounds/title
var nmd = application.createWindow("nonModalDialogName", JSWindow.DIALOG);
controller.showRecords(15, nmd); // 15 is a single-number pk in this case
```

executeProgram

String executeProgram (program)

Execute a program and returns output. Specify the cmd as you would do in a console.

Parameters

{String} program - (fullpath) of the program to execute

Returns

String - The output generated by the program execution.

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself/");
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], ["MY_ENV_VAR=something"], "/home
/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgram("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgram("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgram("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgram

String **executeProgram** (program, params)

Execute a program and returns output. Specify the cmd as you would do in a console.

Parameters

{**String**} program - (fullpath) of the program to execute
 {**String**[]} params - an array of strings as program arguments

Returns

String - The output generated by the program execution.

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself/");
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], ["MY_ENV_VAR=something"], "/home
/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgram("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgram("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgram("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgram

String **executeProgram** (program, params, environmentVars)

Execute a program and returns output. Specify the cmd as you would do in a console.

Parameters

`{String}` program - (fullpath) of the program to execute
`{String[]}` params - an array of strings as program arguments
`{String[]}` environmentVars - array of strings, each element of which has environment variable settings in the format name=value, or null if the subprocess should inherit the environment of the current process.

Returns

`String` - The output generated by the program execution.

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself/");
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], ["MY_ENV_VAR=something"], "/home
/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgram("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgram("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgram("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgram

`String executeProgram` (program, params, environmentVars, startDir)

Execute a program and returns output. Specify the cmd as you would do in a console.

Parameters

`{String}` program - (fullpath) of the program to execute
`{String[]}` params - an array of strings as program arguments
`{String[]}` environmentVars - array of strings, each element of which has environment variable settings in the format name=value, or null if the subprocess should inherit the environment of the current process.
`{String}` startDir - the working directory of the subprocess, or null if the subprocess should inherit the working directory of the current process.

Returns

`String` - The output generated by the program execution.

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgram("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself/");
application.executeProgram("/home/myself/myapp", ["arg1", "arg2", "arg3"], ["MY_ENV_VAR=something"], "/home
/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgram("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgram("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgram("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgramInBackground

void **executeProgramInBackground** (program)

Execute a program in the background. Specify the cmd as you would do in a console.

Parameters

{String} program - (fullpath) of the program to execute

Returns

void

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself
/");
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "/home/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgramInBackground("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgramInBackground("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgramInBackground("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgramInBackground

void **executeProgramInBackground** (program, params)

Execute a program in the background. Specify the cmd as you would do in a console.

Parameters

{String} program - (fullpath) of the program to execute

{String[]} params - an array of strings as program arguments

Returns

void

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself
/");
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "/home/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgramInBackground("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgramInBackground("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgramInBackground("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgramInBackgroundvoid **executeProgramInBackground** (program, params, environmentVars)

Execute a program in the background. Specify the cmd as you would do in a console.

Parameters

{String} program - (fullpath) of the program to execute

{String[]} params - an array of strings as program arguments

{String[]} environmentVars - array of strings, each element of which has environment variable settings in the format name=value, or null if the subprocess should inherit the environment of the current process.

Returns

void

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself
/");
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "/home/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgramInBackground("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgramInBackground("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgramInBackground("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

executeProgramInBackground

void **executeProgramInBackground** (program, params, environmentVars, startDir)

Execute a program in the background. Specify the cmd as you would do in a console.

Parameters

`{String}` program - (fullpath) of the program to execute
`{String[]}` params - an array of strings as program arguments
`{String[]}` environmentVars - array of strings, each element of which has environment variable settings in the format name=value, or null if the subprocess should inherit the environment of the current process.
`{String}` startDir - the working directory of the subprocess, or null if the subprocess should inherit the working directory of the current process.

Returns

void

Sample

```
// For Windows systems:
// Runs a binary located in the user's home directory. The application will run in the current working
// directory, which in general is the one where Servoy was started from.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"]);
// The same as above, but run the application in the user's home directory.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"], null, "c:
\\Users\\myself\\");
// The same as above, but also set an environment variable for the called program.
application.executeProgramInBackground("c:\\Users\\myself\\myapp.exe", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "c:\\Users\\myself\\");
// For non-Windows systems:
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"]);
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"], null, "/home/myself
/");
application.executeProgramInBackground("/home/myself/myapp", ["arg1", "arg2", "arg3"],
["MY_ENV_VAR=something"], "/home/myself/");
// Open a file with the default application associated with it. (on Windows)
application.executeProgramInBackground("rundll32.exe", ["url.dll,FileProtocolHandler", "filename"]);
// Open a file with the default application associated with it. (on Linux)
application.executeProgramInBackground("xdg-open", ["filename"]);
// Open a file with the default application associated with it. (on MacOS)
application.executeProgramInBackground("open", ["filename"]);
// Open a file with a specific application (on MacOS).
application.executeProgram("open", ["-a", "OpenOffice.org.app", "filename.doc"]);
```

exit

void **exit** ()

Stop and exit application.

Returns

void

Sample

```
// exit application
application.exit();
```

getActiveClientCount

Number **getActiveClientCount** (currentSolutionOnly)

Get the active user count on the server (can be limited to current solution).

Parameters

`{Boolean}` currentSolutionOnly - Boolean (true) to get the active user count on server only to the current solution

Returns

Number - Active user count on the server

Sample

```
var count = application.getActiveClientCount(true);
```

getActiveWindow

JSWindow **getActiveWindow** ()

This gets the currently focused active window; this can be the main application window or a modal dialog. For a webclient getWindow() can return the main window that is not really the main for the current tab in the browser that can return the previous tab that a user could have opened. For this method is better suited because this will give you the actual tab in the browser. Another call would be form.controller.getWindow() of a form that you know in which window it resides.

Returns

[JSWindow](#) - the current active/focussed window.

Sample

```
// get the currently active/focused window or dialog
var activeWindow = application.getActiveWindow();
```

getApplicationType

[Number](#) **getApplicationType** ()

Get the application type.

Returns

[Number](#) - Constant application type

Sample

```
var type = application.getApplicationType();
//see application type constant
```

getClientCountForInfo

[Number](#) **getClientCountForInfo** (info)

Gets the count for all clients displaying the same additional information in the Clients page of Servoy Server Administration Console.

Parameters

[{String}](#) info - The additional client info string to search for.

Returns

[Number](#) - Number of clients

Sample

```
var count = application.getClientCountForInfo('SaaS company name');
application.output('Including yourself, there are ' + count + ' client(s) running on behalf of the company.');
```

getClientProperty

[Object](#) **getClientProperty** (name)

Sets a UI property.

Parameters

[{Object}](#) name - Name of the client property

Returns

[Object](#) - the property value for the given name/key, null if nothing was found

Sample

```
//Only use this function from the solution on open method!
//In smart client, use this to set javax.swing.UIDefaults properties.
application.putClientProperty('ToolTip.hideAccelerator', true)
//To change the comboboxes selection background color, do this:
application.putClientProperty('ComboBox.selectionBackground', new Packages.java.awt.ColorUIResource(
java.awt.Color.RED))

//In web client, use this to change the template directory.
//To change the default dir of templates/default to templates/green_skin, do this:
application.putClientProperty('templates.dir', 'green_skin');
```

getClipboardString

[String](#) **getClipboardString** ()

Gets a string from the clipboard, null if not a string or empty.

Returns

[String](#) - The string from the clipboard

Sample

```
var fromClipboard = application.getClipboardString();
```

getCurrentLookAndFeelName

[String](#) **getCurrentLookAndFeelName ()**

Gets the name of the current Look And Feel specified in Application Preferences.

Returns

[String](#) - Current Look And Feel

Sample

```
var laf = application.getCurrentLookAndFeelName();
```

getHostName

[String](#) **getHostName ()**

Get the name of the localhost.

Returns

[String](#) - Name of the localhost

Sample

```
var hostName = application.getHostName();
```

getIPAddress

[String](#) **getIPAddress ()**

Get the clients' IP address.

Returns

[String](#) - IP address of the client

Sample

```
var ip = application.getIPAddress();
```

getLicenseNames

[String\[\]](#) **getLicenseNames ()**

Get the names of the used client licenses (as strings in array).

Returns

[String\[\]](#) - Client licenses names

Sample

```
var array = application.getLicenseNames();
```

getOSName

[String](#) **getOSName ()**

Returns the name of the operating system.

Returns

[String](#) - Name of the operating system

Sample

```
var osname = application.getOSName();
```

getPrinters

String[] getPrinters ()

Get all the printer names in an array.

Returns

[String\[\]](#) - All printer names

Sample

```
var printersArray = application.getPrinters();
```

getScreenHeight**Number getScreenHeight ()**

Get the screen height in pixels.

Returns

[Number](#) - Screen height

Sample

```
var height = application.getScreenHeight();
```

getScreenWidth**Number getScreenWidth ()**

Get the screen width in pixels.

Returns

[Number](#) - Screen width

Sample

```
var width = application.getScreenWidth();
```

getServerTimeStamp**Date getServerTimeStamp ()**

Returns a date object initialized on server with current date and time.

Returns

[Date](#) - Server time

Sample

```
var servertime = application.getServerTimeStamp();
```

getServerURL**String getServerURL ()**

Gets the HTTP server url.

Returns

[String](#) - HTTP server URL

Sample

```
var url = application.getServerURL();
```

getSolutionName**String getSolutionName ()**

Returns the name of the current solution.

Returns

[String](#) - Current solution name

Sample

```
var solutionName = application.getSolutionName();
```

getSolutionRelease**Number** **getSolutionRelease ()**

Get the solution release number.

Returns**Number** - Current solution release number**Sample**

```
var release = application.getSolutionRelease();
```

getTimeStamp**Date** **getTimeStamp ()**

Returns a date object initialized in client with current date and time.

This should be used instead of new Date() for webclients when the clients are in different times zones then the server.

Then this call will really return a time that is the locals webclients time.

Returns**Date** - Current time at the client**Sample**

```
var clienttime = application.getTimeStamp();
```

getUUID**UUID** **getUUID ()**

Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object. A table column marked as UUID will work with such objects.

Returns**UUID** - The new UUID object**Sample**

```
var new_uuid_object = application.getUUID(); // generate new uuid object
var uuid_object1 = application.getUUID(new_uuid_object.toString()); // convert a string representing an uuid
to an uuid object
var uuid_object2 = application.getUUID(new_uuid_object.toBytes()); // convert a byte array representing an
uuid to an uuid object
```

getUUID**UUID** **getUUID (byteArray)**

Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object. A table column marked as UUID will work with such objects.

Parameters

{byte[]} byteArray - Byte array representing an uuid

Returns**UUID** - The new UUID object**Sample**

```
var new_uuid_object = application.getUUID(); // generate new uuid object
var uuid_object1 = application.getUUID(new_uuid_object.toString()); // convert a string representing an uuid
to an uuid object
var uuid_object2 = application.getUUID(new_uuid_object.toBytes()); // convert a byte array representing an
uuid to an uuid object
```

getUUID**UUID** **getUUID (uuidString)**

Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object. A table column marked as UUID will work with such objects.

Parameters

{String} uuidString - String representing an uuid

Returns

[UUID](#) - The new UUID object

Sample

```
var new_uuid_object = application.getUUID(); // generate new uuid object
var uuid_object1 = application.getUUID(new_uuid_object.toString()); // convert a string representing an uuid
to an uuid object
var uuid_object2 = application.getUUID(new_uuid_object.toBytes()); // convert a byte array representing an
uuid to an uuid object
```

getUserProperty

[String](#) **getUserProperty** (name)

Get a persistent user property.

Parameters

[String](#) name - Name of the property

Returns

[String](#) - Property value

Sample

```
var value = application.getUserProperty('showOrders');
```

getUserPropertyNames

[String\[\]](#) **getUserPropertyNames** ()

Get all persistent user property names.

Returns

[String\[\]](#) - Array of all user property names

Sample

```
// display all user properties
allPropertyNames = application.getUserPropertyNames();
for(var i = 0; i < allPropertyNames.length; i++)
{
    application.output(allPropertyNames[i] + " = " + application.getUserProperty(allPropertyNames[i]));
}
```

getValueListArray

[Array](#) **getValueListArray** (name)

Retrieve a valuelist as array, to get real-values for display-values.

NOTE: this doesn't return a value for a valuelist that depends on a database relation or is a global method valuelist.

Parameters

[String](#) name - The name of the valuelist

Returns

[Array](#) - Named array for the valuelist

Sample

```
var packet_types = application.getValueListArray('packet_types');
if (a_realValue == packet_types['displayValue'])
{
}
```

getValueListDisplayValue

[Object](#) **getValueListDisplayValue** (name, realValue)

Retrieve a valuelist display-value for a real-value.

NOTE: this doesn't return a value for a valuelist that depends on a database relation or is a global method valuelist.

Parameters

[String](#) name - Name of the valuelist

[Object](#) realValue - Real value of the valuelist

Returns

[Object](#) - Display value of the real value from the valuelist

Sample

```
var displayable_status = application.getValueListDisplayValue('case_status', status);
```

getValueListItems

[JSDataSet](#) **getValueListItems** (name)

Get all values from a custom or database type value list as dataset (with columns displayValue, realValue).

NOTE: this doesn't return a value for a valuelist that depends on a database relation or is a global method valuelist.

Parameters

[String](#) name - Name of the valuelist

Returns

[JSDataSet](#) - DataSet with valuelist's display values and real values

Sample

```
//Note:see databaseManager.JSDataSet for full details of dataset
var dataset = application.getValueListItems('my_en_types');
//example to calc a strange total
global_total = 0;
for( var i = 1 ; i <= dataset.getMaxRowIndex() ; i++ )
{
    global_total = global_total + dataset.getValue(i,1);
}
//example to assign to dataprovider
//employee_salary = dataset.getValue(1,1)
```

getValueListNames

[String\[\]](#) **getValueListNames** ()

Get all the valuelist names as array.

Returns

[String\[\]](#) - Array with all valuelist names

Sample

```
var array = application.getValueListNames();
```

getVersion

[String](#) **getVersion** ()

Returns the application version.

Returns

[String](#) - Application version

Sample

```
application.getVersion();
```

getWindow

[JSWindow](#) **getWindow** ()

Get the main application window. This is the window that is created first for this client.

In a smart client this is always just the first started window where the solution is loaded in.

In a webclient when users (so not the developer through a createWindow() call) create there own tabs after

In a webclient the user may open the same solution in a new tab in the same browser. In that case the main solution window will always be the first opened tab, even if that one was already closed.

application.getActiveWindow() will always return the currently active/focused window or dialog.

If you need the window of the current top-level form, controller.getWindow() of that form will always return the correct window.

Returns

[JSWindow](#) - the main application JSWindow.

Sample

```
// close and dispose window resources
var mainWindow = application.getWindow();
```

getWindow

JSWindow **getWindow** (name)

Get a window by window name. When not supplying a name, the main application window is grabbed.

Parameters

{**String**} name - the name of the window. If not specified, the main application JSWindow will be returned.

Returns

JSWindow - the JSWindow with the specified name, or null if no such window exists.

Sample

```
// close and dispose window resources
var win = application.getWindow("someWindowName");
if (win != null) {
    win.destroy();
}
```

isInDeveloper

Boolean **isInDeveloper** ()

Returns true if the solution is running in the developer.

Returns

Boolean - Boolean (true) if the solution is running in the developer, (false) otherwise

Sample

```
var flag = application.isInDeveloper();
```

isLastPrintPreviewPrinted

Boolean **isLastPrintPreviewPrinted** ()

Check if the last printpreview did print.

Returns

Boolean - Boolean (true) is the last print preview did print, (false) otherwise

Sample

```
//attached this method to onShow on the form being shown after printpreview
//set a global called scopes.globals.showPrintPreview to 1 in the onPrintPreview method
if (scopes.globals.showPrintPreview == 1)
{
    scopes.globals.showPrintPreview = 0;//clear for next time
    if (application.isLastPrintPreviewPrinted())
    {
        plugins.dialogs.showInfoDialog('Alert', 'There is printed in printpreview', 'OK')
    }
}
```

output

void **output** (msg)

Output something on the out stream. (if running in debugger view output console tab)

Parameters

{**Object**} msg - Object to send to output stream

Returns

void

Sample

```
// log level is used to determine how/if to log in servoy_log.txt; for smart client java out and err streams
are used
application.output('my very important trace msg');// default log level: info
```

output

void **output** (msg, level)

Output something on the out stream. (if running in debugger view output console tab)

Parameters

{[Object](#)} msg - Object to send to output stream
 {[Number](#)} level - the log level where it should log to.

Returns

void

Sample

```
// log level is used to determine how/if to log in servoy_log.txt; for smart client java out and err streams
are used
application.output('my very important msg',LOGGINGLEVEL.ERROR);// log level: error
```

overrideStyle

void **overrideStyle** (originalStyleName, newStyleName)

Overrides one style (defined in in a form) with another.

Parameters

{[String](#)} originalStyleName - Name of the style to override
 {[String](#)} newStyleName - Name of the new style

Returns

void

Sample

```
//This function will only have effect on forms not yet created, so solution onLoad is the best place to
override'
//For example overriding the use of default/designed style anywhere in the solution from 'mystyle' to
'mystyle_mac'
application.overrideStyle('mystyle','mystyle_mace')//in this case both styles should have about the same
classes
```

playSound

void **playSound** (url)

Play a sound (AU file, an AIFF file, a WAV file, and a MIDI file).

Parameters

{[String](#)} url - URL of the sound file

Returns

void

Sample

```
application.playSound('media:///click.wav');
```

putClientProperty

[Boolean](#) **putClientProperty** (name, value)

Sets a UI property.

Parameters

{[Object](#)} name - Name of the client property
 {[Object](#)} value - New value of the client property

Returns

[Boolean](#) - Boolean (true) if the client property was set with the new value

Sample

```
//Only use this function from the solution on open method!
//In smart client, use this to set javax.swing.UIDefaults properties.
application.putClientProperty('ToolTip.hideAccelerator', true)
//To change the comboboxes selection background color, do this:
application.putClientProperty('ComboBox.selectionBackground', new Packages.java.awt.Color.RED))

//In web client, use this to change the template directory.
//To change the default dir of templates/default to templates/green_skin, do this:
application.putClientProperty('templates.dir', 'green_skin');
```

redovoid **redo** ()

Redo last action (if possible).

Returns

void

Sample

```
application.redo();
```

removeAllClientInfovoid **removeAllClientInfo** ()

Removes all names given to the client via the admin page.

Returns

void

Sample

```
application.removeAllClientInfo();
```

removeClientInfo**Boolean** **removeClientInfo** (info)

Removes a string of client information which is stored on the server and previously was added using the application.addClientInfo('client info')

This function can be called more than once, if you want to delete multiple lines of client information.

Parameters**{String}** info - A line of text to be removed from the client information on behalf of the running Servoy client.**Returns****Boolean** - boolean indicator if info was removed successfully**Sample**

```
var removed = application.removeClientInfo('SaaS company name');
```

setClipboardContentvoid **setClipboardContent** (string)

Sets a string object in the clipboard.

Parameters**{Object}** string - New content of the clipboard**Returns**

void

Sample

```
application.setClipboardContent('test');
```

setNumpadEnterAsFocusNextEnabled

void **setNumpadEnterAsFocusNextEnabled** (enabled)

Set if numpad enter should behave like focus next.

Parameters

{[Boolean](#)} enabled - Boolean (true) if numpad enter should behave like focus next

Returns

void

Sample

```
application.setNumpadEnterAsFocusNextEnabled(true);
```

setStatusText

void **setStatusText** (text)

Set the status area value.

Parameters

{[String](#)} text - New status text

Returns

void

Sample

```
application.setStatusText('Your status text');
```

setStatusText

void **setStatusText** (text, tooltip)

Set the status area value.

Parameters

{[String](#)} text - New status text

{[String](#)} tooltip - Status tooltip text

Returns

void

Sample

```
application.setStatusText('Your status text','Your status tooltip text');
```

setToolbarVisible

void **setToolbarVisible** (name, visible)

Make a toolbar visible or invisible.

Parameters

{[String](#)} name - Name of the toolbar

{[Boolean](#)} visible - Visibility of the toolbar

Returns

void

Sample

```
//example: hide the text toolbar  
application.setToolbarVisible('text',false);
```

setUserProperty

void **setUserProperty** (name, value)

Sets a user property for this client:

 For heedless clients((including Batch Processors and Authentication clients) the user property is stored in memory and will be lost upon client restart.

 For Web client the user property will be stored in a persistent cookie

 For Smart Client it will be stored in client local properties file

Parameters

{String} name - Name of the user property
 {String} value - New value of the user property

Returns

void

Sample

```
application.setUserProperty('showOrders','1');
```

setValueListItems

void **setValueListItems** (name, dataset)

Fill a custom type valuelist with values from array(s) or dataset.

NOTE: if you modify values for checkbox/radio field, note that having one value in valuelist is a special case, so switching between one value and 0 /multiple values after form is created may have side effects

Parameters

{String} name - Name of the valuelist
 {JSDataset} dataset - Dataset with display/real values

Returns

void

Sample

```
//set display values (return values will be same as display values)
application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'));
//set display values and return values (which are stored in dataprovider)
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
(10000,10010,10456));
//set display values and return values converted to numbers
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
('10000','10010', '10456'), true);
//do query and fill valuelist (see databaseManager for full details of queries/dataset)
//var query = 'select display_value,optional_real_value from test_table';
//var dataset = databaseManager.getDataSetByQuery(databaseManager.getDataSourceServerName(controller.
getDataSource()), query, null, 25);

//application.setValueListItems('my_en_types',dataset);
```

setValueListItems

void **setValueListItems** (name, dataset, autoconvert)

Fill a custom type valuelist with values from array(s) or dataset.

NOTE: if you modify values for checkbox/radio field, note that having one value in valuelist is a special case, so switching between one value and 0 /multiple values after form is created may have side effects

Parameters

{String} name - Name of the valuelist
 {JSDataset} dataset - Dataset with display/real values
 {Boolean} autoconvert - Boolean (true) if display values and return values should be converted to numbers

Returns

void

Sample

```
//set display values (return values will be same as display values)
application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'));
//set display values and return values (which are stored in dataprovider)
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
(10000,10010,10456));
//set display values and return values converted to numbers
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
('10000','10010', '10456'), true);
//do query and fill valuelist (see databaseManager for full details of queries/dataset)
//var query = 'select display_value,optional_real_value from test_table';
//var dataset = databaseManager.getDataSetByQuery(databaseManager.getDataSourceServerName(controller.
getDataSource()), query, null, 25);

//application.setValueListItems('my_en_types',dataset);
```

setValueListItems

void **setValueListItems** (name, displayValues)

Fill a custom type valuelist with values from array(s) or dataset.

NOTE: if you modify values for checkbox/radio field, note that having one value in valuelist is a special case, so switching between one value and 0 /multiple values after form is created may have side effects

Parameters

{String} name - Name of the valuelist
{Object[]} displayValues - Display values array

Returns

void

Sample

```
//set display values (return values will be same as display values)
application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'));
//set display values and return values (which are stored in dataprovider)
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
(10000,10010,10456));
//set display values and return values converted to numbers
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
('10000','10010', '10456'), true);
//do query and fill valuelist (see databaseManager for full details of queries/dataset)
//var query = 'select display_value,optional_real_value from test_table';
//var dataset = databaseManager.getDataSetByQuery(databaseManager.getDataSourceServerName(controller.
getDataSource()), query, null, 25);

//application.setValueListItems('my_en_types',dataset);
```

setValueListItems

void **setValueListItems** (name, displayValues, autoconvert)

Fill a custom type valuelist with values from array(s) or dataset.

NOTE: if you modify values for checkbox/radio field, note that having one value in valuelist is a special case, so switching between one value and 0 /multiple values after form is created may have side effects

Parameters

{String} name - Name of the valuelist
{Object[]} displayValues - Display values array
{Boolean} autoconvert - Boolean (true) if display values and return values should be converted to numbers

Returns

void

Sample

```
//set display values (return values will be same as display values)
application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'));
//set display values and return values (which are stored in dataprovider)
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
(10000,10010,10456));
//set display values and return values converted to numbers
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
('10000','10010', '10456'), true);
//do query and fill valuelist (see databaseManager for full details of queries/dataset)
//var query = 'select display_value,optional_real_value from test_table';
//var dataset = databaseManager.getDataSetByQuery(databaseManager.getDataSourceServerName(controller.
getDataSource()), query, null, 25);

//application.setValueListItems('my_en_types',dataset);
```

setValueListItems

void **setValueListItems** (name, displayValues, realValues)

Fill a custom type valuelist with values from array(s) or dataset.

NOTE: if you modify values for checkbox/radio field, note that having one value in valuelist is a special case, so switching between one value and 0 /multiple values after form is created may have side effects

Parameters

{String} name - Name of the valuelist
 {Object[]} displayValues - Display values array
 {Object[]} realValues - Real values array

Returns

void

Sample

```
//set display values (return values will be same as display values)
application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'));
//set display values and return values (which are stored in dataprovider)
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
(10000,10010,10456));
//set display values and return values converted to numbers
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
('10000','10010', '10456'), true);
//do query and fill valuelist (see databaseManager for full details of queries/dataset)
//var query = 'select display_value,optional_real_value from test_table';
//var dataset = databaseManager.getDataSetByQuery(databaseManager.getDataSourceServerName(controller.
getDataSource()), query, null, 25);

//application.setValueListItems('my_en_types',dataset);
```

setValueListItems

void **setValueListItems** (name, displayValues, realValues, autoconvert)

Fill a custom type valuelist with values from array(s) or dataset.

NOTE: if you modify values for checkbox/radio field, note that having one value in valuelist is a special case, so switching between one value and 0 /multiple values after form is created may have side effects

Parameters

{String} name
 {Object[]} displayValues - Display values array
 {Object[]} realValues - Real values array
 {Boolean} autoconvert - Boolean (true) if display values and return values should be converted to numbers

Returns

void

Sample

```
//set display values (return values will be same as display values)
application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'));
//set display values and return values (which are stored in dataprovider)
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
(10000,10010,10456));
//set display values and return values converted to numbers
//application.setValueListItems('my_en_types',new Array('Item 1', 'Item 2', 'Item 3'),new Array
('10000','10010', '10456'), true);
//do query and fill valuelist (see databaseManager for full details of queries/dataset)
//var query = 'select display_value,optional_real_value from test_table';
//var dataset = databaseManager.getDataSetByQuery(databaseManager.getDataSourceServerName(controller.
getDataSource()), query, null, 25);

//application.setValueListItems('my_en_types',dataset);
```

showCalendar**Date** **showCalendar** ()

Show the calendar, returns selected date or null if canceled. Initial value and date format can be also specified.

Returns**Date** - Selected date or null if canceled**Sample**

```
var selectedDate = application.showCalendar();
```

showCalendar**Date** **showCalendar** (dateFormat)

Show the calendar, returns selected date or null if canceled. Initial value and date format can be also specified.

Parameters{**String**} dateFormat - Date format**Returns****Date** - Selected date or null if canceled**Sample**

```
var selectedDate = application.showCalendar();
```

showCalendar**Date** **showCalendar** (selectedDate)

Show the calendar, returns selected date or null if canceled. Initial value and date format can be also specified.

Parameters{**Date**} selectedDate - Default selected date**Returns****Date** - Selected date or null if canceled**Sample**

```
var selectedDate = application.showCalendar();
```

showCalendar**Date** **showCalendar** (selectedDate, dateFormat)

Show the calendar, returns selected date or null if canceled. Initial value and date format can be also specified.

Parameters{**Date**} selectedDate - Default selected date{**String**} dateFormat - Date format**Returns****Date** - Selected date or null if canceled

Sample

```
var selectedDate = application.showCalendar();
```

showColorChooser

String **showColorChooser** ()

Show the colorChooser. Returned value is in format #RRGGBB or null if canceled.

Returns

String - selected color or null if canceled

Sample

```
var selectedColor = application.showColorChooser();
```

showColorChooser

String **showColorChooser** (colorString)

Show the colorChooser. Returned value is in format #RRGGBB or null if canceled.

Parameters

{**String**} colorString - Default color

Returns

String - selected color or null if canceled

Sample

```
var selectedColor = application.showColorChooser();
```

showFontChooser

String **showFontChooser** ()

Show the font chooser dialog. Returns the selected font. Can specify a default font.

Returns

String - selected font

Sample

```
var selectedFont = application.showFontChooser();
elements.myfield.font = selectedFont
```

showFontChooser

String **showFontChooser** (defaultFont)

Show the font chooser dialog. Returns the selected font. Can specify a default font.

Parameters

{**String**} defaultFont - Default font

Returns

String - selected font

Sample

```
var selectedFont = application.showFontChooser();
elements.myfield.font = selectedFont
```

showForm

void **showForm** (form)

Show the form specified by the parameter, that can be a name (is case sensitive!) or a form object.

This will show the form in the active/currently focused window. So when called from a form in a dialog the dialog will show the form.

Parameters

{**Object**} form - Form object or name

Returns

void

Sample

```
application.showForm('MyForm');
```

showI18NDialog**String** **showI18NDialog** ()

Opens the i18n dialog so users can change translations. Returns the key selected by the user (not it's translation) or null if cancel is pressed. Optional parameters specify the initial selections in the dialog.

Returns**String** - selected I18N key or null if cancel is pressed**Sample**

```
application.showI18NDialog("servoy.button.close", "en");
```

showI18NDialog**String** **showI18NDialog** (keyToSelect)

Opens the i18n dialog so users can change translations. Returns the key selected by the user (not it's translation) or null if cancel is pressed. Optional parameters specify the initial selections in the dialog.

Parameters{**String**} keyToSelect - Default selected key**Returns****String** - selected I18N key or null if cancel is pressed**Sample**

```
application.showI18NDialog("servoy.button.close", "en");
```

showI18NDialog**String** **showI18NDialog** (keyToSelect, languageToSelect)

Opens the i18n dialog so users can change translations. Returns the key selected by the user (not it's translation) or null if cancel is pressed. Optional parameters specify the initial selections in the dialog.

Parameters{**String**} keyToSelect - Default selected key{**String**} languageToSelect - Default selected language**Returns****String** - selected I18N key or null if cancel is pressed**Sample**

```
application.showI18NDialog("servoy.button.close", "en");
```

showURL**Boolean** **showURL** (url)

Shows an URL in a browser.

Parameters{**String**} url - URL to show**Returns****Boolean** - Boolean (true) if URL was shown

Sample

```
application.showURL('http://www.example.com');

//webclient specific additional parameters...
//2nd parameter: target frame or named dialog/window, so its possible to control in which (internal) frame
or dialog the url is loaded, '_self' is current window, '_blank' is new dialog, '_top' is main window
//3rd parameter: dialog options used when a dialog is specified, example: 'height=200,width=400,status=yes,
toolbar=no,menubar=no,location=no'
//3rd or 4th parameter: a timeout in seconds when the url should be shown, immediately/0 is default'
```

showURL

Boolean **showURL** (url, webclientTarget)

Shows an URL in a browser.

Parameters

{**String**} url - URL to show
 {**String**} webclientTarget - Target frame or named dialog/window

Returns

Boolean - Boolean (true) if URL was shown

Sample

```
application.showURL('http://www.example.com');

//webclient specific additional parameters...
//2nd parameter: target frame or named dialog/window, so its possible to control in which (internal) frame
or dialog the url is loaded, '_self' is current window, '_blank' is new dialog, '_top' is main window
//3rd parameter: dialog options used when a dialog is specified, example: 'height=200,width=400,status=yes,
toolbar=no,menubar=no,location=no'
//3rd or 4th parameter: a timeout in seconds when the url should be shown, immediately/0 is default'
```

showURL

Boolean **showURL** (url, webclientTarget, timeout)

Shows an URL in a browser.

Parameters

{**String**} url - URL to show
 {**String**} webclientTarget - Target frame or named dialog/window
 {**Number**} timeout - A timeout in seconds when the url should be shown

Returns

Boolean - Boolean (true) if URL was shown

Sample

```
application.showURL('http://www.example.com');

//webclient specific additional parameters...
//2nd parameter: target frame or named dialog/window, so its possible to control in which (internal) frame
or dialog the url is loaded, '_self' is current window, '_blank' is new dialog, '_top' is main window
//3rd parameter: dialog options used when a dialog is specified, example: 'height=200,width=400,status=yes,
toolbar=no,menubar=no,location=no'
//3rd or 4th parameter: a timeout in seconds when the url should be shown, immediately/0 is default'
```

showURL

Boolean **showURL** (url, webclientTarget, webclientTargetOptions)

Shows an URL in a browser.

Parameters

{**String**} url - URL to show
 {**String**} webclientTarget - Target frame or named dialog/window
 {**String**} webclientTargetOptions - Dialog options used when a dialog is specified / a timeout in seconds when the url should be shown

Returns

Boolean - Boolean (true) if URL was shown

Sample

```
application.showURL('http://www.example.com');

//webclient specific additional parameters...
//2nd parameter: target frame or named dialog/window, so its possible to control in which (internal) frame
or dialog the url is loaded, '_self' is current window, '_blank' is new dialog, '_top' is main window
//3rd parameter: dialog options used when a dialog is specified, example: 'height=200,width=400,status=yes,
toolbar=no,menubar=no,location=no'
//3rd or 4th parameter: a timeout in seconds when the url should be shown, immediately/0 is default'
```

showURL

Boolean **showURL** (url, webclientTarget, webclientTargetOptions, timeout)

Shows an URL in a browser.

Parameters

{**String**} url - URL to show
 {**String**} webclientTarget - Target frame or named dialog/window
 {**String**} webclientTargetOptions - Dialog options used when a dialog is specified / a timeout in seconds when the url should be shown
 {**Number**} timeout - A timeout in seconds when the url should be shown

Returns

Boolean - Boolean (true) if URL was shown

Sample

```
application.showURL('http://www.example.com');

//webclient specific additional parameters...
//2nd parameter: target frame or named dialog/window, so its possible to control in which (internal) frame
or dialog the url is loaded, '_self' is current window, '_blank' is new dialog, '_top' is main window
//3rd parameter: dialog options used when a dialog is specified, example: 'height=200,width=400,status=yes,
toolbar=no,menubar=no,location=no'
//3rd or 4th parameter: a timeout in seconds when the url should be shown, immediately/0 is default'
```

sleep

void **sleep** (ms)

Sleep for specified time (in milliseconds).

Parameters

{**Number**} ms - Sleep time in milliseconds

Returns

void

Sample

```
//Sleep for 3 seconds
application.sleep(3000);
```

undo

void **undo** ()

Undo last action (if possible).

Returns

void

Sample

```
application.undo();
```

updateUI

void **updateUI** ()

Returns

void

updateUI

void **updateUI** (milliseconds)

Parameters

{[Number](#)} milliseconds - How long the update should take in milliseconds

Returns

void