

# Application



Apr 03, 2024 20:27

## Return Types

APPLICATION\_TYPES CLIENTDESIGN DRAGNDROP ELEMENT\_TYPES CSSPosition Renderable JSDNDEvent JSEvent JSRenderEvent JSupload JSWindow LOGGINGLEVEL UICONSTANTS UUID WEBCONSTANTS NGCONSTANTS

## Supported Clients

SmartClient WebClient NGClient MobileClient

## Methods Summary

|          |                                                                                        |                                                                                                                                                                                                                                                                                   |
|----------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| void     | <a href="#">addClientInfo(info)</a>                                                    | 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     | <a href="#">beep()</a>                                                                 | Produces a "beep" sound; commonly used to indicate an error or warning dialog.                                                                                                                                                                                                    |
| Boolean  | <a href="#">closeAllWindows()</a>                                                      | Closes all visible windows (except main application window).                                                                                                                                                                                                                      |
| void     | <a href="#">closeSolution()</a>                                                        | Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.                                                                                                                                         |
| void     | <a href="#">closeSolution(solutionToLoad)</a>                                          | Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.                                                                                                                                         |
| void     | <a href="#">closeSolution(solutionToLoad, methodArgument)</a>                          | Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.                                                                                                                                         |
| void     | <a href="#">closeSolution(solutionToLoad, methodName)</a>                              | Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.                                                                                                                                         |
| void     | <a href="#">closeSolution(solutionToLoad, methodName, methodArgument)</a>              | Closes the currently open solution and optionally opens another solution, calling a specified global method with the specified arguments.                                                                                                                                         |
| Boolean  | <a href="#">createNewFormInstance(designFormName, newInstanceScriptName)</a>           | Create a new form instance.                                                                                                                                                                                                                                                       |
| JSWindow | <a href="#">createWindow(windowName, type)</a>                                         | Creates a new window that can be used for displaying forms.                                                                                                                                                                                                                       |
| JSWindow | <a href="#">createWindow(windowName, type, parentWindow)</a>                           | Creates a new window that can be used for displaying forms.                                                                                                                                                                                                                       |
| void     | <a href="#">executeLater(function, delay)</a>                                          | Runs at method at the given delay in milliseconds.                                                                                                                                                                                                                                |
| void     | <a href="#">executeLater(function, delay, arguments)</a>                               | Runs at method at the given delay in milliseconds with the arguments given to the method.                                                                                                                                                                                         |
| String   | <a href="#">executeProgram(program)</a>                                                | Execute a program and returns output.                                                                                                                                                                                                                                             |
| String   | <a href="#">executeProgram(program, params)</a>                                        | Execute a program and returns output.                                                                                                                                                                                                                                             |
| String   | <a href="#">executeProgram(program, params, environmentVars)</a>                       | Execute a program and returns output.                                                                                                                                                                                                                                             |
| String   | <a href="#">executeProgram(program, params, environmentVars, startDir)</a>             | Execute a program and returns output.                                                                                                                                                                                                                                             |
| void     | <a href="#">executeProgramInBackground(program)</a>                                    | Execute a program in the background.                                                                                                                                                                                                                                              |
| void     | <a href="#">executeProgramInBackground(program, params)</a>                            | Execute a program in the background.                                                                                                                                                                                                                                              |
| void     | <a href="#">executeProgramInBackground(program, params, environmentVars)</a>           | Execute a program in the background.                                                                                                                                                                                                                                              |
| void     | <a href="#">executeProgramInBackground(program, params, environmentVars, startDir)</a> | Execute a program in the background.                                                                                                                                                                                                                                              |
| void     | <a href="#">exit()</a>                                                                 | Stop and exit application.                                                                                                                                                                                                                                                        |
| Object   | <a href="#">generateBrowserFunction(functionString)</a>                                | This generates a browser function for the given function string that can be executed in the browser. The resulting object can be assigned into a config/property object that is then assigned to a component. The component will receive this function as a real function object. |
| Number   | <a href="#">getActiveClientCount(currentSolutionOnly)</a>                              | Get the active user count on the server (can be limited to current solution).                                                                                                                                                                                                     |
| JSWindow | <a href="#">getActiveWindow()</a>                                                      | This gets the currently focused active window; this can be the main application window or a modal dialog.                                                                                                                                                                         |
| Number   | <a href="#">getApplicationType()</a>                                                   | Get the application type.                                                                                                                                                                                                                                                         |
| Number   | <a href="#">getClientCountForInfo(info)</a>                                            | Gets the count for all clients displaying the same additional information in the Clients page of Servoy Server Administration Console.                                                                                                                                            |
| Object   | <a href="#">getClientProperty(name)</a>                                                | Sets a UI property.                                                                                                                                                                                                                                                               |
| String   | <a href="#">getClipboardString()</a>                                                   | Gets a string from the clipboard, null if not a string or empty.                                                                                                                                                                                                                  |
| String   | <a href="#">getCurrentLookAndFeelName()</a>                                            | Gets the name of the current Look And Feel specified in Application Preferences.                                                                                                                                                                                                  |
| String   | <a href="#">getHostName()</a>                                                          | Get the name of the localhost.                                                                                                                                                                                                                                                    |
| String   | <a href="#">getIPAddress()</a>                                                         | Get the clients' IP address.                                                                                                                                                                                                                                                      |
| Array    | <a href="#">getLicenseNames()</a>                                                      | Get the names of the used client licenses (as strings in array).                                                                                                                                                                                                                  |
| JSLogger | <a href="#">getLogger()</a>                                                            | Get a JSLogger instance which offers an API for logging with arguments.                                                                                                                                                                                                           |
| JSLogger | <a href="#">getLogger(loggerName)</a>                                                  | Get a JSLogger instance which offers an API for logging with arguments.                                                                                                                                                                                                           |
| String   | <a href="#">getMediaURL(mediaName)</a>                                                 | Get the media url that can be used to server a media in NGClient.                                                                                                                                                                                                                 |
| String   | <a href="#">getOSName()</a>                                                            | Returns the name of the operating system of the client.                                                                                                                                                                                                                           |
| Array    | <a href="#">getPrinters()</a>                                                          | 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    | getServerUUID()                                                 | Get the uuid from this server instance (the same value that is shown on the admin page)                                                                                                      |
| String    | getSolutionName()                                               | Returns the name of the current solution.                                                                                                                                                    |
| Number    | getSolutionRelease()                                            | Get the solution release number.                                                                                                                                                             |
| Array     | getStartupArguments()                                           | Get the parameters which are provided by startup.                                                                                                                                            |
| 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.                                                                                                                                                              |
| Array     | 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).                                                                                   |
| Array     | getValueListNames()                                             | Get all the valuelist names as array.                                                                                                                                                        |
| String    | getVersion()                                                    | Returns the application version.                                                                                                                                                             |
| Object    | getVersionInfo()                                                | Get the full version information of this solution and all its modules.                                                                                                                       |
| 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 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).                                                                                                                                                              |
| Boolean   | refreshGlobalMethodValueList(element)                           | Refresh a global method valuelist by forcing it to call the global method.                                                                                                                   |
| Boolean   | refreshGlobalMethodValueList(element, propertyName)             | Refresh a global method valuelist by forcing it to call the global method.                                                                                                                   |
| 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: <br> In NGClient this is stored in the locale storage of the browser, so it will be persisted over restarts as long as the user didn't clear the data. |
| 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  | <code>show18NDialog(keyToSelect, languageToSelect)</code>               | Opens the i18n dialog so users can change translations. |
| Boolean | <code>showURL(url)</code>                                               | Shows an URL in a browser.                              |
| Boolean | <code>showURL(url, browserTarget)</code>                                | Shows an URL in a browser.                              |
| Boolean | <code>showURL(url, browserTarget, timeout)</code>                       | Shows an URL in a browser.                              |
| Boolean | <code>showURL(url, browserTarget, browserTargetOptions)</code>          | Shows an URL in a browser.                              |
| Boolean | <code>showURL(url, browserTarget, browserTargetOptions, timeout)</code> | Shows an URL in a browser.                              |
| void    | <code>sleep(ms)</code>                                                  | Sleep for specified time (in milliseconds).             |
| void    | <code>undo()</code>                                                     | Undo last action (if possible).                         |
| void    | <code>updateUI()</code>                                                 | Updates the UI (painting).                              |
| void    | <code>updateUI(milliseconds)</code>                                     | Updates the UI (painting).                              |

## Methods Details

### 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.

#### Supported Clients

SmartClient, WebClient, NGClient

#### Sample

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

### beep()

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

#### Supported Clients

SmartClient

#### Sample

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

### 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.

#### Supported Clients

SmartClient, WebClient, NGClient

**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()**

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).

**Supported Clients**

SmartClient,WebClient,NGClient

**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');
//application.closeSolution('solution_name', {a: 'my_string_argument', p1: 'param1', p2: 'param2'});//close current solution, open solution 'solution_name', call solution's onOpen with argument 'my_argument' and queryParams p1,p2
//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(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

**Supported Clients**

SmartClient,WebClient,NGClient

**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');
//application.closeSolution('solution_name', {a: 'my_string_argument', p1: 'param1', p2: 'param2'});//close
current solution, open solution 'solution_name', call solution's onOpen with argument 'my_argument' and
queryParams p1,p2
//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(solutionToLoad, 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

**Object** methodArgument Argument passed to the solution onOpen

**Supported Clients**

SmartClient,WebClient,NGClient

**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');
//application.closeSolution('solution_name', {a: 'my_string_argument', p1: 'param1', p2: 'param2'});//close
current solution, open solution 'solution_name', call solution's onOpen with argument 'my_argument' and
queryParams p1,p2
//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(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

**Supported Clients**

SmartClient,WebClient,NGClient

**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');
//application.closeSolution('solution_name', {a: 'my_string_argument', p1: 'param1', p2: 'param2'});//close
current solution, open solution 'solution_name', call solution's onOpen with argument 'my_argument' and
queryParams p1,p2
//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(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

**Supported Clients**

SmartClient,WebClient,NGClient

**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');
//application.closeSolution('solution_name', {a: 'my_string_argument', p1: 'param1', p2: 'param2'});//close
current solution, open solution 'solution_name', call solution's onOpen with argument 'my_argument' and
queryParams p1,p2
//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(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

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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**

|               |            |                                                                                                                                    |
|---------------|------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>String</b> | windowName | the name of the window. Should not be null.                                                                                        |
| <b>Number</b> | type       | the type of the window. Can be one of JSWindow.DIALOG, JSWindow.MODAL_DIALOG, JSWindow.WINDOW. (WINDOW does not work for NGClient) |

**Returns**

**JSWindow** the newly created window.

**Supported Clients**

SmartClient,WebClient,NGClient

**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 (WINDOW does not
work for NGClient)
// 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(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**

|                 |              |                                                                                                                                        |
|-----------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>String</b>   | windowName   | the name of the window. Should not be null.                                                                                            |
| <b>Number</b>   | type         | the type of the window. Can be one of JSWindow.DIALOG, JSWindow.MODAL_DIALOG, JSWindow.WINDOW (WINDOW doesn't work in NGClient).       |
| <b>JSWindow</b> | 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.

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
// create and show a window, with specified title, initial location and size (WINDOW does not work for NGClient)
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
```

**executeLater(function, delay)**

Runs at method at the given delay in milliseconds.

This is like a simple scheduler to quickly run something after a bit of delay

**Parameters**

**Function** function The function to call

**Number** delay The millis that has to elapse before the function is called.

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample****executeLater(function, delay, arguments)**

Runs at method at the given delay in milliseconds with the arguments given to the method.

This is like a simple scheduler to quickly run something after a bit of delay

**Parameters**

**Function** function The function to call

**Number** delay The millis that has to elapse before the function is called.

**Array** arguments The arguments that are given to the function when called.

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample****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.

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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  
**Array** params an array of strings as program arguments

**Returns**

**String** The output generated by the program execution.

**Supported Clients**

SmartClient, WebClient, NGClient

**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(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

**Array** params an array of strings as program arguments

**Array** environment 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.

**Vars**

**Returns**

**String** The output generated by the program execution.

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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

**Array** params an array of strings as program arguments

**Array** environment 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.

**Vars**

**String** startDir the working directory of the subprocess, or null if the subprocess should inherit the working directory of the current process.

**ng**

**Returns**

**String** The output generated by the program execution.

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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

[Array](#) params an array of strings as program arguments

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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                                                                                                                                                 |
| Array  | params          | an array of strings as program arguments                                                                                                                                             |
| Array  | 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. |

**Supported Clients**

SmartClient, WebClient, NGClient

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

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

**Parameters**

|                               |                                                                                                                                                                                      |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>String</b> program         | (fullpath) of the program to execute                                                                                                                                                 |
| <b>Array</b> params           | an array of strings as program arguments                                                                                                                                             |
| <b>Array</b> environment Vars | 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. |
| <b>String</b> startDir        | the working directory of the subprocess, or null if the subprocess should inherit the working directory of the current process.                                                      |

**Supported Clients**

SmartClient,WebClient,NGClient

**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()**

Stop and exit application.

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**generateBrowserFunction(functionString)**

This generates a browser function for the given function string that can be executed in the browser. The resulting object can be assigned into a config/property object that is then assigned to a component. The component will receive this function as a real function object.

This is a more dynamic variant of the spec property "clientfunction" <https://wiki.servoy.com/display/DOCS/Property+Types>

**Parameters**

**String** functionString The function string of a js function that should be running in the clients browser.

**Returns**

**Object** An object that can be assigned to a javascript/json object that is send to the client

**Supported Clients**

NGClient

**Sample**

```
var options = { myfunction: application.generateBrowserFunction("function(param) { return param + 1 }") };
elements.component.setOptions(options);
```

**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

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**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.

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getApplicationType()**

Get the application type.

**Returns**

[Number](#) Constant application type

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

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

**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

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**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

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
//Only use this function from the solution on open method!
//In smart client, use this to set javax.swing.UIManager defaults 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');
```

**getClipboardString()**

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

**Returns**

[String](#) The string from the clipboard

**Supported Clients**

SmartClient, NGClient

**Sample**

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

**getCurrentLookAndFeelName()**

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

**Returns**

[String](#) Current Look And Feel

**Supported Clients**

SmartClient

**Sample**

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

**getHostName()**

Get the name of the localhost.

**Returns**

[String](#) Name of the localhost

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

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

**getIPAddress()**

Get the clients' IP address.

**Returns**

[String](#) IP address of the client

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

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

**getLicenseNames()**

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

**Returns**

[Array](#) Client licenses names

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

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

**getLogger()**

Get a JSLogger instance which offers an API for logging with arguments.

Available logging levels are (in order): fatal, error, warn, info, debug and trace.

If no loggerName is given to this method, it returns the default logger (LoggerFactory.getLogger(Debug.class))

NOTE: the default logging level of the the default logger is 'warn', so info, debug and trace events are not logged.

**Returns**

[JSLogger](#) a new JSLogger instance

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
var log = application.getLogger(); // returns the default logger.
application.output("is logging level 'warn' enabled? " + log.isWarnEnabled); // if false, next line won't log
log.warn.log("this logger logs {} {} {}", "all", "my", "arguments");
```

**getLogger(loggerName)**

Get a JSLogger instance which offers an API for logging with arguments.

Available logging levels are (in order): fatal, error, warn, info, debug and trace.

The argument should be the name of a logger that is configured in myServoyInstallationDir/application\_server/log4j.xml.

A new logger can be configured in log4j.xml by adding the following line:

```
<Logger name="myLogger" level="INFO"/>
```

**Parameters**

[String](#) loggerName the name of the logger, as configured in log4j.xml

**Returns**

[JSLogger](#) a new JSLogger instance

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
var log = application.getLogger("myLogger");
application.output("is logging level 'warn' enabled? " + log.isWarnEnabled); // if false, next line won't log
log.warn.log("this logger logs {} {} {}", "all", "my", "arguments");
```

**getMediaURL(mediaName)**

Get the media url that can be used to server a media in NGClient.

**Parameters**

[String](#) mediaName Name of the media

**Returns**

[String](#)

**Supported Clients**

NGClient

---

**Sample**

```
application.getMediaURL('solution.css');
```

**getOSName()**

Returns the name of the operating system of the client.  
In Smart Client this will return os.name system property. In Web/NG Client will return "OSFamily majorVersion.minorVersion".

**Returns**

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

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getPrinters()**

Get all the printer names in an array.

**Returns**

[Array](#) All printer names

**Supported Clients**

SmartClient,WebClient

**Sample**

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

**getScreenHeight()**

Get the screen height in pixels.

**Returns**

[Number](#) Screen height

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getScreenWidth()**

Get the screen width in pixels.

**Returns**

[Number](#) Screen width

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getServerTimeStamp()**

Returns a date object initialized on server with current date and time.  
For NG and web clients this is the same as new Date() in scripting.

**Returns**

[Date](#) Server time

**Supported Clients**

SmartClient,WebClient,NGClient

---

**Sample**

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

**getServerURL()**

Gets the HTTP server url.

This url will end with a / so don't append to this server url something that starts with a / again because RFC 3986 says that the path of a url (the part after the domain[:poort]) can not start with 2 slashes.

**Returns**

[String](#) HTTP server URL

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

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

**getServerUUID()**

Get the uuid from this server instance (the same value that is shown on the admin page)

**Returns**

[String](#)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var uuid = application.getServerUUID();
```

**getSolutionName()**

Returns the name of the current solution.

**Returns**

[String](#) Current solution name

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getSolutionRelease()**

Get the solution release number.

**Returns**

[Number](#) Current solution release number

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getStartupArguments()**

Get the parameters which are provided by startup.

It returns an array with 2 elements, a string that is the startup argument and a map containing all named startup arguments, or null if there is no argument passed

**Returns**

[Array](#) Array with 2 elements, a string that is the startup argument and a map containing all named startup arguments, or null if there is no argument passed

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var args_array = application.getStartupArguments();
// the first element in the array is the 'argument' value from the startup
var argument = args_array[0];
// the second element is a map containing all the named startup arguments
var startupArgumentObj = args_array[1];
var arg1 = startupArgumentObj['arg1_name'];
var arg2 = startupArgumentObj['arg2_name'];
```

**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.  
 For NG clients this is only useful when displaying on the client using format property (Use local time), and then this is equivalent to new Date() on the client side, so basically this can be used to pre-fill with 'now' such a display.

**Returns**[Date](#) Current time at the client**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**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**Supported Clients**

SmartClient,WebClient,NGClient

**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(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**[Array](#) byteArray Byte array representing an uuid**Returns**[UUID](#) The new UUID object**Supported Clients**

SmartClient,WebClient,NGClient

**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(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

**Supported Clients**

SmartClient,WebClient,NGClient

**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(name)**

Get a persistent user property.

In NGClient this is stored in the locale storage of the browser, so it will be persisted over restarts as long as the user didn't clear the data.

**Parameters**

[String](#) name Name of the property

**Returns**

[String](#) Property value

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

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

**getUserPropertyNames()**

Get all persistent user property names.

**Returns**

[Array](#) Array of all user property names

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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

#### Supported Clients

SmartClient,WebClient,NGClient

#### Sample

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

### 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

#### Supported Clients

SmartClient,WebClient,NGClient

#### Sample

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

### 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

#### Supported Clients

SmartClient,WebClient,NGClient

#### 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()

Get all the valuelist names as array.

#### Returns

[Array](#) Array with all valuelist names

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getVersion()**

Returns the application version.

**Returns**[String](#) Application version**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**getVersionInfo()**

Get the full version information of this solution and all its modules.

This will return an object that is a map of Name(String)-&gt;Version(String) of the solution and all its modules.

**Returns**[Object](#) Name->Version map object.**Supported Clients**

SmartClient,WebClient,NGClient

**Sample****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 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.**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**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.**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**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

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

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

**isLastPrintPreviewPrinted()**

Check if the last printpreview did print.

**Returns**

[Boolean](#) Boolean (true) is the last print preview did print, (false) otherwise

**Supported Clients**

SmartClient,WebClient

**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(msg)**

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

**Parameters**

[Object](#) msg Object to send to output stream

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**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(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.

**Supported Clients**

SmartClient,WebClient,NGClient

**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(originalStyleName, newStyleName)**

Overrides one style with another. In NGClient, it overrides the original stylesheet media defined on a solution with another media.

**Parameters**

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

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
// Smart Client/Web Client usage
//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

//NGClient usage
application.overrideStyle('oldstylesheet.css','mystylesheets/newstylesheet.css');
//Also less is supported also with compiling it at runtime
applicaiton.overrideStyle('solution.less', 'tenant.less'); // tenant.less can be a solution model changed or generated file, then it will be recompiled at runtime.
```

**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

**Supported Clients**

SmartClient,WebClient

**Sample**

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

**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

**Supported Clients**

SmartClient,WebClient,NGClient

**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.swing.plaf.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');
```

**redo()**

Redo last action (if possible).

**Supported Clients**

SmartClient

**Sample**

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

**refreshGlobalMethodValueList(element)**

Refresh a global method valuelist by forcing it to call the global method. The element which has the valuelist must be provided.

If there is no propertyName specified, the element must have only one valuelist property.

**Parameters**

[Object](#) element form element

**Returns**

[Boolean](#) boolean indicating if valuelist was refreshed

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
application.refreshGlobalMethodValueList(elements.mytypeahead);
```

**refreshGlobalMethodValueList(element, propertyName)**

Refresh a global method valuelist by forcing it to call the global method. The element which has the valuelist must be provided.

The valuelist is searched under provided property from the spec - for usage in NGClient custom components.

**Parameters**

[Object](#) element form element

[String](#) propertyName name of property from the spec

**Returns**

[Boolean](#) boolean indicating if valuelist was refreshed

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
application.refreshGlobalMethodValueList(elements.mycustomcomponent, 'myvaluelistProperty');
```

**removeAllClientInfo()**

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

**Supported Clients**

SmartClient,WebClient,NGClient

---

**Sample**

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

**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

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

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

**setClipboardContent(string)**

Sets a string object in the clipboard.

**Parameters**

[Object](#) string New content of the clipboard

**Supported Clients**

SmartClient, NGClient

**Sample**

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

**setNumpadEnterAsFocusNextEnabled(enabled)**

Set if numpad enter should behave like focus next.

**Parameters**

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

**Supported Clients**

SmartClient

**Sample**

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

**setStatusText(text)**

Set the status area value.

**Parameters**

[String](#) text New status text

**Supported Clients**

SmartClient, WebClient

**Sample**

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

**setStatusText(text, tooltip)**

---

Set the status area value.

NOTE: Most modern browsers do not support status bar and status text anymore. This method sets status property of the window using javascript.

#### Parameters

`String` text New status text  
`String` tooltip Status tooltip text

#### Supported Clients

SmartClient, WebClient

#### Sample

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

### setToolbarVisible(name, visible)

Make a toolbar visible or invisible.

#### Parameters

`String` name Name of the toolbar  
`Boolean` visible Visibility of the toolbar

#### Supported Clients

SmartClient

#### Sample

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

### setUserProperty(name, value)

Sets a user property for this client:

In NGClient this is stored in the locale storage of the browser, so it will be persisted over restarts as long as the user didn't clear the data.

For headless 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 a properties file on the client machine.

#### Parameters

`String` name Name of the user property  
`String` value New value of the user property

#### Supported Clients

SmartClient, WebClient, NGClient, MobileClient

#### Sample

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

### 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 may have side effects

NOTE: This is expensive operation, which triggers refresh of all visible forms. Over usage of this method may inflict performance issues.

#### Parameters

`String` name Name of the valuelist  
`JSDataSet` dataset Dataset with display/real values

#### Supported Clients

SmartClient, WebClient, NGClient

**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(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 may have side effects

NOTE: This is expensive operation, which triggers refresh of all visible forms. Over usage of this method may inflict performance issues.

**Parameters**

|                  |             |                                                                                   |
|------------------|-------------|-----------------------------------------------------------------------------------|
| <b>String</b>    | name        | Name of the valuelist                                                             |
| <b>JSDataSet</b> | dataset     | Dataset with display/real values                                                  |
| <b>Boolean</b>   | autoconvert | Boolean (true) if display values and return values should be converted to numbers |

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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 may have side effects

NOTE: This is expensive operation, which triggers refresh of all visible forms. Over usage of this method may inflict performance issues.

**Parameters**

|               |               |                       |
|---------------|---------------|-----------------------|
| <b>String</b> | name          | Name of the valuelist |
| <b>Array</b>  | displayValues | Display values array  |

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**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(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 may have side effects

NOTE: This is expensive operation, which triggers refresh of all visible forms. Over usage of this method may inflict performance issues.

**Parameters**

**String** name            Name of the valuelist  
**Array** displayValues   Display values array  
**Boolean** autoconvert   Boolean (true) if display values and return values should be converted to numbers

**Supported Clients**

SmartClient,WebClient,NGClient

**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(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 may have side effects

NOTE: This is expensive operation, which triggers refresh of all visible forms. Over usage of this method may inflict performance issues.

**Parameters**

**String** name            Name of the valuelist  
**Array** displayValues   Display values array  
**Array** realValues       Real values array

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**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(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 may have side effects

NOTE: This is expensive operation, which triggers refresh of all visible forms. Over usage of this method may inflict performance issues.

**Parameters**

**String** name ;  
**Array** displayValues Display values array  
**Array** realValues Real values array  
**Boolean** autoconvert Boolean (true) if display values and return values should be converted to numbers

**Supported Clients**

SmartClient,WebClient,NGClient

**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()**

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

**Supported Clients**

SmartClient

**Sample**

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

**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

**Supported Clients**

SmartClient

**Sample**

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

**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

**Supported Clients**

SmartClient

**Sample**

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

**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

**Supported Clients**

SmartClient

**Sample**

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

**showColorChooser()**

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

**Returns**

[String](#) selected color or null if canceled

**Supported Clients**

SmartClient

**Sample**

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

**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

**Supported Clients**

SmartClient

**Sample**

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

**showFontChooser()**

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

**Returns**

[String](#) selected font

**Supported Clients**

SmartClient

**Sample**

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

**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

**Supported Clients**

SmartClient

**Sample**

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

**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

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

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

**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

**Supported Clients**

SmartClient

**Sample**

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

**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

#### Supported Clients

SmartClient

#### Sample

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

### 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

#### Supported Clients

SmartClient

#### Sample

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

### showURL(url)

Shows an URL in a browser.

#### Parameters

[String](#) url URL to show

#### Returns

[Boolean](#) Boolean (true) if URL was shown

#### Supported Clients

SmartClient,WebClient,NGClient

#### Sample

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

//NGClient and 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; default is
'_blank'
//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(url, browserTarget)

Shows an URL in a browser.

#### Parameters

[String](#) url URL to show

[String](#) browserTarget Target frame or named dialog/window

#### Returns

[Boolean](#) Boolean (true) if URL was shown

#### Supported Clients

SmartClient,WebClient,NGClient

**Sample**

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

//NGClient and 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; default is
'_blank'
//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(url, browserTarget, timeout)**

Shows an URL in a browser.

**Parameters**

**String** url URL to show  
**String** browserTarget 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

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

//NGClient and 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; default is
'_blank'
//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(url, browserTarget, browserTargetOptions)**

Shows an URL in a browser.

**Parameters**

**String** url URL to show  
**String** browserTarget Target frame or named dialog/window  
**String** browserTargetOptions 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

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

//NGClient and 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; default is
'_blank'
//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(url, browserTarget, browserTargetOptions, timeout)**

Shows an URL in a browser.

**Parameters**

|                     |                                   |                                                                                                    |
|---------------------|-----------------------------------|----------------------------------------------------------------------------------------------------|
| <code>String</code> | <code>url</code>                  | URL to show                                                                                        |
| <code>String</code> | <code>browserTarget</code>        | Target frame or named dialog/window                                                                |
| <code>String</code> | <code>browserTargetOptions</code> | Dialog options used when a dialog is specified / a timeout in seconds when the url should be shown |
| <code>Number</code> | <code>timeout</code>              | A timeout in seconds when the url should be shown                                                  |

**Returns**

`Boolean` Boolean (true) if URL was shown

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

//NGClient and 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; default is
'_blank'
//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(ms)**

Sleep for specified time (in milliseconds).

**Parameters**

`Number` ms Sleep time in milliseconds

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

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

**undo()**

Undo last action (if possible).

**Supported Clients**

SmartClient

**Sample**

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

**updateUI()**

Updates the UI (painting). If in a script an element changed and the script continues doing things, you can give an number in ms how long this can take.  
Warning: this gives the UI time to paint, but this also means that it will give the ui time to respond to all other events, so if a user keeps clicking on other stuff this will also be handled right away inside this call.

NOTE:In NGClient, this method will send to browser all outstanding changes. If called too often (with many changes), can cause performance issues.

**Supported Clients**

SmartClient,NGClient

**Sample**

```
application.updateUI(500);
//continue doing things
```

**updateUI(milliseconds)**

---

Updates the UI (painting). If in a script an element changed and the script continues doing things, you can give an number in ms how long this can take.  
Warning: this gives the UI time to paint, but this also means that it will give the ui time to respond to all other events,  
so if a user keeps clicking on other stuff this will also be handled right away inside this call.

NOTE:In NGClient, this method will send to browser all outstanding changes. If called too often (with many changes), can cause performance issues.

**Parameters**

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

**Supported Clients**

SmartClient,NGClient

**Sample**

```
application.updateUI(500);  
//continue doing things
```