Application

Return Types

DRAGNDROP JSEvent APPLICATION_TYPES ELEMENT_TYPES LOGGINGLEVEL UICONSTANTS WEBCONSTANTS UUID

Method Summery

void	#addClientInfo(info) Adds a string of client information which gets stored on the server, and can be viewed on the Clients page of Servoy Server Administration Console.
void	#beep() Produces a "beep" sound; commonly used to indicate an error or warning dialog.
Boolean	#closeForm([windowOrDialogName/closeAll]) Close the dialog/window with the given name (call this method to hide the form shown with 'showFormInDialog' or 'showFormInWindow').
void	#closeSolution([solutionToLoad], [method], [argument]) Close the current open solution and optionally open a new one.
Boolean	#createNewFormInstance(designFormName, newInstanceScriptName) Create a new form instance.
String	#executeProgram(programName(fullpath), [arg1], [arg2], [argN], [#], environmentvar1, [environmentvarN], [startdirectory]) Execute a program and returns output.
void	#executeProgramInBackground(programName(fullpath), [arg1], [arg2], [argN], [#], [environmentvar1], [environmentvarN], [startdirectory]) Execute a program in the background.
void	#exit() Stop and exit application.
Number	#getActiveClientCount(currentSolutionOnly) Get the active user count on the server (can be limited to current solution).
Number	#getApplicationType() Get the application type.
Number	#getClientCountForInfo(info) Gets the count for all clients displaying the same additional information in the Clients page of Servoy Server Administration Console.
String	#getClipboardString() Gets a string from the clipboard, null if not a string or empty.
String	#getCurrentLookAndFeelName() Gets the name of the current Look And Feel specified in Application Preferences.
String	#getHostName() Get the name of the localhost.
String	#getIPAddress() Get the clients' IP address.
String[]	#getLicenseNames() Get the names of the used client licenses (as strings in array).
String	#getOSName() Returns the name of the operating system.
String[]	#getPrinters() Get all the printer names in an array.
Number	#getScreenHeight() Get the screen height in pixels.
Number	#getScreenWidth() Get the screen width in pixels.
Date	#getServerTimeStamp() Returns a date object initialized on server with current date and time.
String	#getServerURL() Gets the HTTP server url.
String	#getSolutionName() Returns the name of the current solution.
Number	#getSolutionRelease() Get the solution release number.
Object[]	#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([uuidStringOrByteArray]) Get a new UUID object (also known as GUID) or convert the parameter (that can be string or byte array) to an UUID object.
String	#getUserProperty(name) Get a persistent user property.
String[]	#getUserPropertyNames() Get all persistent user property names.

Array	#getValueListArray(name) Retrieve a valuelist as array, to get real-values for display-values.
Object	#getValueListDisplayValue(name, realValue) Retrieve a valuelist display-value for a real-value.
JSDataSet	#getValueListItems(name) Get all values from a custom or database type value list as dataset (with columns displayValue,realValue).
String[]	#getValueListNames() Get all the valuelist names as array.
String	#getVersion() Returns the application version.
Number	#getWindowHeight([windowName]) Get the window height in pixels.
Number	#getWindowWidth([windowName]) Get the window width in pixels.
Number	#getWindowX([windowName]) Get the window X location in pixels.
Number	#getWindowY([windowName]) Get the window Y location in pixels.
Boolean	#isInDeveloper() Returns true if the solution is running in the developer.
Boolean	#isLastPrintPreviewPrinted() Check if the last printpreview did print.
void	#output(msg)
void	#output(msg, [level]) Output something on the out stream.
void	#overrideStyle(originalStyleName, newStyleName) Overrides one style (defined in in a form) with another.
void	#playSound(url) Play a sound (AU file, an AIFF file, a WAV file, and a MIDI file).
void	#redo() Redo last action (if possible).
void	#removeAllClientInfo() Removes all names given to the client via the admin page.
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, [tip]) Set the status area value.
void	#setToolbarVisible(name, visible) Make a toolbar visible or invisible.
Boolean	#setUIProperty(name, value) Sets a UI property.
void	#setUserProperty(name, value) Set a persistent user property.
void	#setValueListItems(name, displayValArray/dataset, [realValuesArray], [autoconvert(false)]) Fill a custom type valuelist with values from array(s) or dataset.
void	#setWindowLocation(x, y, [windowName]) Set the window location.
void	#setWindowSize(width, height, [windowName]) Set the window size.
Date	#showCalendar([selecteddate], [dateformat]) Show the calendar, returns selected date or null if canceled.
String	#showColorChooser([colorString]) Show the colorChooser.
String	#showFontChooser([fontString]) 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.
void	#showFormInDialog(form, [x], [y], [width], [height], [dialogTitle], [resizable], [showTextToolbar], [windowName], [modal]) Show the specified form in a dialog.
void	#showFormInWindow(form, [x], [y], [width], [height], [dialogTitle], [resizable], [showTextToolbar], [windowName]) Show the specified form in a window.
String	#showI18NDialog([keyToSelect], [languageToSelect]) Opens the i18n dialog so users can change translations.
Boolean	#showURL(url, [webclientTarget], [webclientTargetOptions/timeout], [timeout]) Shows an URL in a browser.
void	#sleep(ms) Sleep for specified time (in milliseconds).
void	#undo() Undo last action (if possible).
void	#updateUI([milliseconds]) Updates the UI (painting).

Method Details

addClientInfo

void **addClientInfo**(info)

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

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

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

NOTE:
This function can also be used with the function getClientCountForInfo
to count the number of clients with matching addditional client information.

Parameters

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

Returns

void

Sample

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

beep

void **beep**()

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

Returns

void

Sample

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

closeForm

Boolean **closeForm**([windowOrDialogName/closeAll])

Close the dialog/window with the given name (call this method to hide the form shown with 'showFormInDialog' or 'showFormInWindow'). If (true) is passed, then all the windows/dialogs will be closed. If the name is missing or null, the default dialog/window will be closed.

Parameters

[windowOrDialogName/closeAll] – Name of the dialog/window to close, or (true) to close all open dialogs/windows.

Returns

Boolean – Boolean (true) if the dialog(s)/window(s) were closed, (false) otherwise

Sample

```
application.closeForm(); // closes the current dialog/window
//application.closeForm('windowOrDialogName'); //closes the dialog/window with this specific name
```

closeSolution

void **closeSolution**([solutionToLoad], [method], [argument])

Close the current open solution and optionally open a new one.

Parameters

[solutionToLoad] – Name of the solution to load
[method] – Name of the global method to call
[argument] – Argument passed to the global method

Returns

void

Sample

```
application.closeSolution();
//application.closeSolution('solution_name','global_method_name','my_argument');//log out, open solution
'solution_name', call global method 'global_method_name' with argument 'my_argument'
//note: specifying a solution will not work in developer due to debugger dependencies
```

createNewFormInstance

Boolean **createNewFormInstance**(designFormName, newInstanceScriptName)

Create a new form instance.

Parameters

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

Returns

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

Sample

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

executeProgram

String **executeProgram**(programName(fullpath), [arg1], [arg2], [argN], [#], environmentvar1, [environmentvarN], [startdirectory])

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

Parameters

programName(fullpath) – Name of the program to execute
[arg1] – Argument
[arg2] – Argument
[argN] – Argument
[#] – Divider between program args, environment vars and startdir
environmentvar1 – Environment variable
[environmentvarN] – Environment variable
[startdirectory] – Program start directory

Returns

String – The output generated by the program execution.

Sample

```
//'#' is divider between program args, environment vars and startdir
var program_output = application.executeProgram('c:/temp/program.ext','arg0','arg1','argN','#','path=c:
/temp','#','c:/temp');
```

executeProgramInBackground

void **executeProgramInBackground**(programName(fullpath), [arg1], [arg2], [argN], [#], [environmentvar1], [environmentvarN], [startdirectory])

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

Parameters

programName(fullpath) – Name of the program to execute in background
[arg1] – Argument
[arg2] – Argument
[argN] – Argument
[#] – Divider between program args, environment vars and startdir
[environmentvar1] – Environment variable
[environmentvarN] – Environment variable
[startdirectory] – Environment variable

Returns

void

Sample

```
//'#' is divider between program args, environment vars and startdir
application.executeProgramInBackground('c:/temp/program.ext','arg0','arg1','argN');
```

exit

void **exit**()

Stop and exit application.

Returns

void

Sample

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

getActiveClientCount

Number **getActiveClientCount**(currentSolutionOnly)

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

Parameters

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

Returns

Number – Active user count on the server

Sample

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

getApplicationType

Number **getApplicationType**()

Get the application type.

Returns

Number – Constant application type

Sample

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

getClientCountForInfo

Number **getClientCountForInfo**(info)

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

Parameters

{String} info – The additional client info string to search for.

Returns

Number – Number of clients

Sample

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

getClipboardString

String **getClipboardString**()

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

Returns

String – The string from the clipboard

Sample

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

getCurrentLookAndFeelName

String **getCurrentLookAndFeelName**()

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

Returns

String – Current Look And Feel

Sample

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

getHostName

String **getHostName**()

Get the name of the localhost.

Returns

String – Name of the localhost

Sample

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

getIPAddress

String **getIPAddress**()

Get the clients' IP address.

Returns

String – IP address of the client

Sample

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

getLicenseNames

String[] **getLicenseNames**()

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

Returns

String[] – Client licenses names

Sample

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

getOSName

String **getOSName**()

Returns the name of the operating system.

Returns

String – Name of the operating system

Sample

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

getPrinters

String[] **getPrinters**()

Get all the printer names in an array.

Returns

String[] – All printer names

Sample

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

getScreenHeight

Number **getScreenHeight**()

Get the screen height in pixels.

Returns

Number – Screen height

Sample

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

getScreenWidth

Number **getScreenWidth**()

Get the screen width in pixels.

Returns

Number – Screen width

Sample

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

getServerTimeStamp

Date **getServerTimeStamp**()

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

Returns

Date – Server time

Sample

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

getServerURL

String **getServerURL**()

Gets the HTTP server url.

Returns

String – HTTP server URL

Sample

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

getSolutionName

String **getSolutionName**()

Returns the name of the current solution.

Returns

String – Current solution name

Sample

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

getSolutionRelease

Number **getSolutionRelease**()

Get the solution release number.

Returns

Number – Current solution release number

Sample

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

getStartupArguments

Object[] **getStartupArguments**()

Get the parameters which are provided by startup.

Returns

Object[] – Array with 2 elements, the startup argument and an object containing all startup arguments, or null if there is no argument passed

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 an object containing all the startup arguments
var startupArgumentObj = args_array[1];
var arg1 = startupArgumentObj.arg1_name;
var arg2 = startupArgumentObj.arg2_name;
```

getTimeStamp

Date **getTimeStamp**()

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

Returns

Date – Current time at the client

Sample

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

getUUID

UUID **getUUID**([uuidStringOrByteArray])

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

[uuidStringOrByteArray] – String or byte array representing an uuid

Returns

UUID – The new UUID object

Sample

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

getUserProperty

String **getUserProperty**(name)

Get a persistent user property.

Parameters

{String} name – Name of the property

Returns

String – Property value

Sample

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

getUserPropertyNames

String[] **getUserPropertyNames**()

Get all persistent user property names.

Returns

String[] – Array of all user property names

Sample

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

getValueListArray

Array **getValueListArray**(name)

Retrieve a valuelist as array, to get real-values for display-values.
NOTE: this doesn't return a value for a valuelist that depends on a database relation or is a global method valuelist.

Parameters
{String} name – The name of the valuelist

Returns
Array – Named array for the valuelist

Sample

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

getValueListDisplayValue

Object **getValueListDisplayValue**(name, realValue)

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

Parameters
{String} name – Name of the valuelist
{Object} realValue – Real value of the valuelist

Returns
Object – Display value of the real value from the valuelist

Sample

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

getValueListItems

JSDataSet **getValueListItems**(name)

Get all values from a custom or database type value list as dataset (with columns displayValue,realValue).
NOTE: this doesn't return a value for a valuelist that depends on a database relation or is a global method valuelist.

Parameters
{String} name – Name of the valuelist

Returns
JSDataSet – DataSet with valuelist's display values and real values

Sample

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

getValueListNames

String[] **getValueListNames**()

Get all the valuelist names as array.

Returns
String[] – Array with all valuelist names

Sample

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

getVersion

String **getVersion**()

Returns the application version.

Returns
String – Application version

Sample

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

getWindowHeight

Number **getWindowHeight**([windowName])

Get the window height in pixels. If windowName is not specified or null, it will use either the default dialog (if it is shown) or the main application window.

Parameters

[windowName] – Name of the window

Returns

Number – Window height

Sample

```
var height = application.getWindowHeight('customerDialog');
```

getWindowWidth

Number **getWindowWidth**([windowName])

Get the window width in pixels. If windowName is not specified or null, it will use either the default dialog (if it is shown) or the main application window.

Parameters

[windowName] – Name of the window

Returns

Number – Window width

Sample

```
var width = application.getWindowWidth('customerDialog');
```

getWindowX

Number **getWindowX**([windowName])

Get the window X location in pixels. If windowName is not specified or null, it will use either the default dialog (if it is shown) or the main application window.

Parameters

[windowName] – Window name

Returns

Number – Window X location

Sample

```
var x = application.getWindowX('customerDialog');
```

getWindowY

Number **getWindowY**([windowName])

Get the window Y location in pixels. If windowName is not specified or null, it will use either the default dialog (if it is shown) or the main application window.

Parameters

[windowName] – Name of the window

Returns

Number – Window Y location

Sample

```
var y = application.getWindowY('customerDialog');
```

isInDeveloper

Boolean **isInDeveloper**()

Returns true if the solution is running in the developer.

Returns

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

Sample

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

isLastPrintPreviewPrinted

Boolean **isLastPrintPreviewPrinted**()
Check if the last printpreview did print.
Returns
Boolean – Boolean (true) is the last print preview did print, (false) otherwise
Sample

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

output
void **output**(msg, [level])
Output something on the out stream. (if running in debugger view output console tab)
Parameters
{Object} msg – Object to send to output stream
{Number} [level] – the log level where it should log to.
Returns
void
Sample

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

overrideStyle
void **overrideStyle**(originalStyleName, newStyleName)
Overrides one style (defined in in a form) with another.
Parameters
{String} originalStyleName – Name of the style to override
{String} newStyleName – Name of the new style
Returns
void
Sample

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

playSound
void **playSound**(url)
Play a sound (AU file, an AIFF file, a WAV file, and a MIDI file).
Parameters
{String} url – URL of the sound file
Returns
void
Sample

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

redo
void **redo**()
Redo last action (if possible).

Returns
void
Sample

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

removeAllClientInfo

void **removeAllClientInfo**()
Removes all names given to the client via the admin page.
Returns
void
Sample

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

setClipboardContent

void **setClipboardContent**(string)
Sets a string object in the clipboard.
Parameters
{Object} string – New content of the clipboard
Returns
void
Sample

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

setNumpadEnterAsFocusNextEnabled

void **setNumpadEnterAsFocusNextEnabled**(enabled)
Set if numpad enter should behave like focus next.
Parameters
{Boolean} enabled – Boolean (true) if numpad enter should behave like focus next
Returns
void
Sample

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

setStatusText

void **setStatusText**(text, [tip])
Set the status area value.
Parameters
text – New status text
[tip] – Status tooltip text
Returns
void
Sample

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

setToolbarVisible

void **setToolbarVisible**(name, visible)
Make a toolbar visible or invisible.
Parameters
{String} name – Name of the toolbar
{Boolean} visible – Visibility of the toolbar
Returns
void

Sample

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

setUIProperty

Boolean **setUIProperty**(name, value)

Sets a UI property.

Parameters

{Object} name – Name of the UI property

{Object} value – New value of the UI property

Returns

Boolean – Boolean (true) if the UI property was set with the new value

Sample

```
//Only use this function from the solution on open method!
//In smart client, use this to set javax.swing.UIDefaults properties.
application.setUIProperty('ToolTip.hideAccelerator', true)

//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.setUIProperty('templates.dir','green_skin');
```

setUserProperty

void **setUserProperty**(name, value)

Set a persistent user property.

Parameters

{String} name – Name of the user property

{String} value – New value of the user property

Returns

void

Sample

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

setValueListItems

void **setValueListItems**(name, displayValArray/dataset, [realValuesArray], [autoconvert(false)])

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

Parameters

name – Name of the valuelist

displayValArray/dataset – Display values array or DataSet

[realValuesArray] – Real values array

[autoconvert(false)] – Boolean (true) if display values and return values should be converted to numbers

Returns

void

Sample

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

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

setWindowLocation

void **setWindowLocation**(x, y, [windowName])

Set the window location. If windowName is not specified or null, it will use either the default dialog (if it is shown) or the main application window.

Parameters
x – Window new X location
y – Window new Y location
[windowName] – Name of the window

Returns
void

Sample

```
application.setWindowLocation(10,10,'customerDialog');
```

setWindowSize

void **setWindowSize**(width, height, [windowName])

Set the window size. If windowName is not specified or null, it will resize either the default dialog (if it is shown) or the main application window.

Parameters
width – Window new width
height – Window new height
[windowName] – Name of the window

Returns
void

Sample

```
application.setWindowSize(400,400,'customerDialog');
```

showCalendar

Date **showCalendar**([selecteddate], [dateformat])

Show the calendar, returns selected date or null if canceled.

Parameters
[selecteddate] – Default selected date
[dateformat] – Date format

Returns
Date – Selected date or null if canceled

Sample

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

showColorChooser

String **showColorChooser**([colorString])

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

Parameters
[colorString] – Default color

Returns
String – selected color or null if canceled

Sample

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

showFontChooser

String **showFontChooser**([fontString])

Show the font chooser dialog. Returns the selected font.

Parameters
[fontString] – Default font

Returns
String – selected font

Sample

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

showForm

void **showForm**(form)

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

Parameters
{Object} form – Form object or name
Returns
void
Sample

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

showFormInDialog

void **showFormInDialog**(form, [x], [y], [width], [height], [dialogTitle], [resizable], [showTextToolbar], [windowName], [modal])
Show the specified form in a dialog. (NOTE: x, y, width, height are initial bounds - applied only the fist time a dialog is shown)

NOTE:
In the Smart Client, no code is executed after the function showFormInDialog if the dialog is modal.

NOTE:
x, y, width and height coordinates are only applied the first time the specified dialog is shown.
Use APP_UI_PROPERTY.FULL_SCREEN for these values when the dialog should be full-screen.
Parameters
form – The form to be shown in the dialog.
[x] – The "x" coordinate of the dialog.
[y] – The "y" coordinate of the dialog.
[width] – The width of the dialog.
[height] – The height of the dialog.
[dialogTitle] – The title of the dialog.
[resizable] – true if the dialog size should be modifiable; false if not.
[showTextToolbar] – true to add a text toolbar; false to not add a text toolbar.
[windowName] – The name of the window; defaults to "dialog" if nothing is specified. Window and dialog names share the same namespace.
[modal] – true if the dialog should be modal; false if not. Defaults to true.
Returns
void
Sample

```
//Show the specified form in a modal dialog, on default initial location and size (x,y,w,h)
//application.showFormInDialog(forms.contacts);
//Note: No code is executed after the showFormInDialog until the dialog is closed if it is created as a modal
dialog.
//Show the specified form in a non-modal dialog with a specified name, on default initial location and size (x,
y,w,h)
//application.showFormInDialog(forms.contacts,'contactsdialog',false);
//Show the specified form in a modal dialog, at a specified initial location and size with custom title, not
resizable but with text toolbar
application.showFormInDialog(forms.contacts,100,80,500,300,'my own dialog title',false,true,'mydialog',true);
```

showFormInWindow

void **showFormInWindow**(form, [x], [y], [width], [height], [dialogTitle], [resizable], [showTextToolbar], [windowName])
Show the specified form in a window. (NOTE: x, y, width, height are initial bounds - applied only the fist time a window is shown)

NOTE:
Forms in windows cannot be modal. They are more independent then dialogs, even non-modal ones. For example in SC, a non-modal dialog will always be shown on top of the parent window and it will not have a separate entry in the OS window manager (for example Windows taskbar).

NOTE:
x, y, width and height coordinates are only applied the first time the specified window is shown.
Use APP_UI_PROPERTY.FULL_SCREEN for these values when the window should be full-screen.
Parameters
form – The form to be shown in the dialog.
[x] – The "x" coordinate of the dialog.
[y] – The "y" coordinate of the dialog.
[width] – The width of the dialog.
[height] – The height of the dialog.
[dialogTitle] – The title of the dialog.
[resizable] – true if the dialog size should be modifiable; false if not.
[showTextToolbar] – true to add a text toolbar; false to not add a text toolbar.
[windowName] – The name of the window; defaults to "dialog" if nothing is specified. Window and dialog names share the same namespace.
Returns
void

Sample

```
//Show the specified form in a window, on default initial location and size
//application.showFormInWindow(forms.contacts);
//Show the specified form in a window with a specified name, on default initial location and size
//application.showFormInWindow(forms.contacts,'contactsWindow');
//Show the specified form in a window, at a specified initial location and size with custom title, not
resizable but with text toolbar
application.showFormInWindow(forms.contacts,100,80,500,300,'my own window title',false,true,'mywindow');
```

showI18NDialog

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

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

Parameters

[keyToSelect] – Default selected key
[languageToSelect] – Default selected language

Returns

String – selected I18N key or null if cancel is pressed

Sample

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

showURL

Boolean **showURL**(url, [webclientTarget], [webclientTargetOptions/timeout], [timeout])

Shows an URL in a browser.

Parameters

url – URL to show
[webclientTarget] – Target frame or named dialog/window
[webclientTargetOptions/timeout] – Dialog options used when a dialog is specified / a timeout in seconds when the url should be shown
[timeout] – A timeout in seconds when the url should be shown

Returns

Boolean – Boolean (true) if URL was shown

Sample

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

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

sleep

void **sleep**(ms)

Sleep for specified time (in milliseconds).

Parameters

{Number} ms – Sleep time in milliseconds

Returns

void

Sample

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

undo

void **undo**()

Undo last action (if possible).

Returns

void

Sample

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

updateUI

void **updateUI**([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.

Parameters

[milliseconds] – How long the update should take in milliseconds

Returns

void

Sample

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