

# window

## Return Types

[CheckBox](#) [Menu](#) [MenuBar](#) [MenuItem](#) [Popup](#) [RadioButton](#) [ToolBar](#)

## Method Summary

[ToolBar](#) [#addToolBar\(name\)](#)  
Add a toolbar.

[ToolBar](#) [#addToolBar\(name, row\)](#)  
Add a toolbar.

[ToolBar](#) [#addToolBar\(name, displayname\)](#)  
Add a toolbar.

[ToolBar](#) [#addToolBar\(name, displayname, row\)](#)  
Add a toolbar.

[void](#) [#cancelFormPopup\(\)](#)  
Close the current form popup panel without assigning a value to the configured data provider.

[void](#) [#closeFormPopup\(retval\)](#)  
Close the current form popup panel and assign the value to the configured data provider.

[Popup](#) [#createPopupMenu\(\)](#)  
Creates a new popup menu that can be populated with items and displayed.

[Boolean](#) [#createShortcut\(shortcut, methodName\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, methodName, arguments\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, methodName, contextFilter\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, methodName, contextFilter, arguments\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, method\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, method, arguments\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, method, contextFilter\)](#)  
Create a shortcut.

[Boolean](#) [#createShortcut\(shortcut, method, contextFilter, arguments\)](#)  
Create a shortcut.

[MenuBar](#) [#getMenuBar\(\)](#)  
Get the menubar of the main window, or of a named window.

[MenuBar](#) [#getMenuBar\(windowName\)](#)  
Get the menubar of the main window, or of a named window.

[ToolBar](#) [#getToolBar\(name\)](#)  
Get the toolbar from the toolbar panel by name.

[String\[\]](#) [#getToolBarNames\(\)](#)  
Get all toolbar names from the toolbar panel.

[void](#) [#maximize\(\)](#)  
Maximize the current window or the window with the specified name (Smart client only).

[void](#) [#maximize\(windowName\)](#)  
Maximize the current window or the window with the specified name (Smart client only).

[Boolean](#) [#removeShortcut\(shortcut\)](#)  
Remove a shortcut.

[Boolean](#) [#removeShortcut\(shortcut, contextFilter\)](#)  
Remove a shortcut.

[void](#) [#removeToolBar\(name\)](#)  
Remove the toolbar from the toolbar panel.

[void](#) [#setFullScreen\(full\)](#)  
Bring the window into/out of fullscreen mode.

[void](#) [#setStatusBarVisible\(visible\)](#)  
Show or hide the statusbar.

[void](#) [#setToolBarAreaVisible\(visible\)](#)  
Show or hide the toolbar area.

[void](#) [#showFormPopup\(elementToShowRelatedTo, form, scope, dataproviderID\)](#)  
Show a form as popup panel, where the closeFormPopup can pass return a value to a dataprovider in the specified scope.

## Method Details

addToolBar

**ToolBar** addToolBar(name)

Add a toolbar.

**Parameters**

{String} name

**Returns**

ToolBar

**Sample**

```
// Note: method addToolBar only works in the smart client.

// add a toolbar with only a name
var toolbar0 = plugins.window.addToolBar("toolbar_0");
toolbar0.addButton("click me 0", feedback_button);

// add a toolbar with a name and the row you want it to show at
// row number starts at 0
var toolbar1 = plugins.window.addToolBar("toolbar_1", 2);
toolbar1.addButton("click me 1", feedback_button);

// add a toolbar with a name and display name
var toolbar2 = plugins.window.addToolBar("toolbar_2", "toolbar_2_internal_name");
toolbar2.addButton("click me 2", feedback_button);

// add a toolbar with a name, display name and the row you want the
// toolbar to show at. row number starts at 0
var toolbar3 = plugins.window.addToolBar("toolbar_3", "toolbar_3_internal_name", 3);
toolbar3.addButton("click me 3", feedback_button);
```

addToolBar

**ToolBar** addToolBar(name, row)

Add a toolbar.

**Parameters**

{String} name

{Number} row

**Returns**

ToolBar

**Sample**

```
// Note: method addToolBar only works in the smart client.

// add a toolbar with only a name
var toolbar0 = plugins.window.addToolBar("toolbar_0");
toolbar0.addButton("click me 0", feedback_button);

// add a toolbar with a name and the row you want it to show at
// row number starts at 0
var toolbar1 = plugins.window.addToolBar("toolbar_1", 2);
toolbar1.addButton("click me 1", feedback_button);

// add a toolbar with a name and display name
var toolbar2 = plugins.window.addToolBar("toolbar_2", "toolbar_2_internal_name");
toolbar2.addButton("click me 2", feedback_button);

// add a toolbar with a name, display name and the row you want the
// toolbar to show at. row number starts at 0
var toolbar3 = plugins.window.addToolBar("toolbar_3", "toolbar_3_internal_name", 3);
toolbar3.addButton("click me 3", feedback_button);
```

addToolBar

**ToolBar** addToolBar(name, displayname)

Add a toolbar.

**Parameters**

{String} name

{String} displayname

**Returns**

ToolBar

## Sample

```
// Note: method addToolBar only works in the smart client.

// add a toolbar with only a name
var toolbar0 = plugins.window.addToolBar("toolbar_0");
toolbar0.addButton("click me 0", feedback_button);

// add a toolbar with a name and the row you want it to show at
// row number starts at 0
var toolbar1 = plugins.window.addToolBar("toolbar_1", 2);
toolbar1.addButton("click me 1", feedback_button);

// add a toolbar with a name and display name
var toolbar2 = plugins.window.addToolBar("toolbar_2", "toolbar_2_internal_name");
toolbar2.addButton("click me 2", feedback_button);

// add a toolbar with a name, display name and the row you want the
// toolbar to show at. row number starts at 0
var toolbar3 = plugins.window.addToolBar("toolbar_3", "toolbar_3_internal_name", 3);
toolbar3.addButton("click me 3", feedback_button);
```

addToolBar

**ToolBar** **addToolBar**(name, displayname, row)

Add a toolbar.

### Parameters

{String} name

{String} displayname

{Number} row

### Returns

ToolBar

## Sample

```
// Note: method addToolBar only works in the smart client.

// add a toolbar with only a name
var toolbar0 = plugins.window.addToolBar("toolbar_0");
toolbar0.addButton("click me 0", feedback_button);

// add a toolbar with a name and the row you want it to show at
// row number starts at 0
var toolbar1 = plugins.window.addToolBar("toolbar_1", 2);
toolbar1.addButton("click me 1", feedback_button);

// add a toolbar with a name and display name
var toolbar2 = plugins.window.addToolBar("toolbar_2", "toolbar_2_internal_name");
toolbar2.addButton("click me 2", feedback_button);

// add a toolbar with a name, display name and the row you want the
// toolbar to show at. row number starts at 0
var toolbar3 = plugins.window.addToolBar("toolbar_3", "toolbar_3_internal_name", 3);
toolbar3.addButton("click me 3", feedback_button);
```

cancelFormPopup

void **cancelFormPopup**()

Close the current form popup panel without assigning a value to the configured data provider.

### Returns

void

## Sample

```
//Show a form as popup panel, where the closeFormPopup can pass return a value to a dataprovider in the
specified scope.
plugins.window.showFormPopup(null,forms.orderPicker,foundset.getSelectedRecord(),"order_id");
//do call closeFormPopup(ordervalue) from the orderPicker form
```

closeFormPopup

void **closeFormPopup**(retval)

Close the current form popup panel and assign the value to the configured data provider.

**Parameters**

{Object} retval – return value for data provider

**Returns**

void

**Sample**

```
//Show a form as popup panel, where the closeFormPopup can pass return a value to a dataprovider in the
specified scope.
plugins.window.showFormPopup(null,forms.orderPicker,foundset.getSelectedRecord(),"order_id");
//do call closeFormPopup(ordervalue) from the orderPicker form
```

createPopupMenu

Popup **createPopupMenu**()

Creates a new popup menu that can be populated with items and displayed.

**Returns**

Popup

**Sample**

```
// create a popup menu
var menu = plugins.window.createPopupMenu();
// add a menu item
menu.addMenuItem("an entry", feedback);

if (event.getSource()) {
    // display the popup over the component which is the source of the event
    menu.show(event.getSource());
    // display the popup over the components, at specified coordinates relative to the component
    //menu.show(event.getSource(), 10, 10);
    // display the popup at specified coordinates relative to the main window
    //menu.show(100, 100);
}
```

createShortcut

Boolean **createShortcut**(shortcut, methodName)

Create a shortcut.

**Parameters**

{String} shortcut

{String} methodName – scopes.scopename.methodname or formname.methodname String to target the method to execute

**Returns**

Boolean

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, methodName, arguments)

Create a shortcut.

### Parameters

{String} shortcut

{String} methodName – scopes.scopename.methodname or formname.methodname String to target the method to execute

{Object[]} arguments

### Returns

**Boolean**

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * Handle keyboard shortcut.
 *
 * @param {JSEvent} event the event that triggered the action
 */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, methodName, contextFilter)

Create a shortcut.

### Parameters

{String} shortcut

{String} methodName – scopes.scopename.methodname or formname.methodname String to target the method to execute

{String} contextFilter – only triggers the shortcut when on this form

### Returns

**Boolean**

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, methodName, contextFilter, arguments)

Create a shortcut.

### Parameters

{String} shortcut

{String} methodName – scopes.scopename.methodname or formname.methodname String to target the method to execute

{String} contextFilter – only triggers the shortcut when on this form

{Object[]} arguments

### Returns

**Boolean**

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, method)

Create a shortcut.

### Parameters

{String} shortcut

{Function} method – the method/function that needs to be called when the shortcut is hit

### Returns

**Boolean**



## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, method, arguments)

Create a shortcut.

### Parameters

{String} shortcut

{Function} method – the method/function that needs to be called when the shortcut is hit

{Object[]} arguments

### Returns

**Boolean**

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, method, contextFilter)

Create a shortcut.

### Parameters

{String} shortcut

{Function} method – the method/function that needs to be called when the shortcut is hit

{String} contextFilter – only triggers the shortcut when on this form

### Returns

**Boolean**

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

createShortcut

**Boolean** createShortcut(shortcut, method, contextFilter, arguments)

Create a shortcut.

### Parameters

{String} shortcut

{Function} method – the method/function that needs to be called when the shortcut is hit

{String} contextFilter – only triggers the shortcut when on this form

{Object[]} arguments

### Returns

**Boolean**

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * Handle keyboard shortcut.
 *
 * @param {JSEvent} event the event that triggered the action
 */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

getMenuBar

**MenuBar** getMenuBar()

Get the menubar of the main window, or of a named window.

### Returns

**MenuBar**

## Sample

```
// create a new window
var win = application.createWindow("windowName", JSWindow.WINDOW);
// show a form in the new window
forms.my_form.controller.show(win);
// retrieve the menubar of the new window
var menubar = plugins.window.getMenuBar("windowName");
// add a new menu to the menubar, with an item in it
var menu = menubar.addMenu();
menu.text = "New Menu";
menu.addMenuItem("an entry", feedback);
// retrieve the menubar of the main window
var mainMenubar = plugins.window.getMenuBar();
// add a new menu to the menubar of the main window
var menuMain = mainMenubar.addMenu();
menuMain.text = "New Menu in Main Menubar";
menuMain.addMenuItem("another entry", feedback);
```

getMenuBar

**MenuBar** getMenuBar(windowName)

Get the menubar of the main window, or of a named window.

### Parameters

[{String}](#) windowName – the name of the window

### Returns

[MenuBar](#)

### Sample

```
// create a new window
var win = application.createWindow("windowName", JSWindow.WINDOW);
// show a form in the new window
forms.my_form.controller.show(win);
// retrieve the menubar of the new window
var menubar = plugins.window.getMenuBar("windowName");
// add a new menu to the menubar, with an item in it
var menu = menubar.addMenu();
menu.text = "New Menu";
menu.addMenuItem("an entry", feedback);
// retrieve the menubar of the main window
var mainMenubar = plugins.window.getMenuBar();
// add a new menu to the menubar of the main window
var menuMain = mainMenubar.addMenu();
menuMain.text = "New Menu in Main Menubar";
menuMain.addMenuItem("another entry", feedback);
```

### getToolBar

[ToolBar](#) **getToolBar**(name)

Get the toolbar from the toolbar panel by name.

### Parameters

[{String}](#) name

### Returns

[ToolBar](#)

### Sample

```
// Note: method getToolBar only works in the smart client.

// the toolbar must first be create with a call to addToolBar
plugins.window.addToolBar("toolbar_0");

// get the toolbar at the panel by name
var toolbar = plugins.window.getToolBar("toolbar_0");
// add a button to the toolbar
toolbar.addButton("button", feedback_button);
```

### getToolBarNames

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

Get all toolbar names from the toolbar panel.

### Returns

[String\[\]](#)

### Sample

```
// Note: method getToolBarNames only works in the smart client.

// create an array of toolbar names
var names = plugins.window.getToolBarNames();

// create an empty message variable
var message = "";

// loop through the array
for (var i = 0 ; i < names.length ; i++) {
    //add the name(s) to the message
    message += names[i] + "\n";
}

// show the message
plugins.dialogs.showInfoDialog("toolbar names", message);
```

maximize

void **maximize()**

Maximize the current window or the window with the specified name (Smart client only).

**Returns**

void

### Sample

```
// maximize the main window:
plugins.window.maximize();

// create a new window
var win = application.createWindow("windowName", JSWindow.WINDOW);
// show a form in the new window
forms.my_form.controller.show(win);
// maximize the window
plugins.window.maximize("windowName");
```

maximize

void **maximize(windowName)**

Maximize the current window or the window with the specified name (Smart client only).

**Parameters**

{String} windowName

**Returns**

void

### Sample

```
// maximize the main window:
plugins.window.maximize();

// create a new window
var win = application.createWindow("windowName", JSWindow.WINDOW);
// show a form in the new window
forms.my_form.controller.show(win);
// maximize the window
plugins.window.maximize("windowName");
```

removeShortcut

Boolean **removeShortcut(shortcut)**

Remove a shortcut.

**Parameters**

{String} shortcut

**Returns**

Boolean

## Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * * Handle keyboard shortcut.
 * *
 * * @param {JSEvent} event the event that triggered the action
 * */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

removeShortcut

**Boolean** removeShortcut(shortcut, contextFilter)

Remove a shortcut.

### Parameters

{String} shortcut

{String} contextFilter – only triggers the shortcut when on this form

### Returns

**Boolean**

### Sample

```
// this plugin uses the java keystroke parser
// see http://java.sun.com/j2se/1.5.0/docs/api/javawx/swing/KeyStroke.html#getKeyStroke(java.lang.String)
// global handler
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut);
// global handler with a form context filter
plugins.window.createShortcut('control shift I', scopes.globals.handleOrdersShortcut, 'frm_contacts');
// form method called when shortcut is used
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut);
// form method called when shortcut is used and arguments are passed to the method
plugins.window.createShortcut('control RIGHT', forms.frm_contacts.handleMyShortcut, new Array(argument1,
argument2));
// Passing the method argument as a string prevents unnecessary form loading
//plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', new Array(argument1,
argument2));
// Passing the method as a name and the contextFilter set so that this shortcut only trigger on the form
'frm_contacts'.
plugins.window.createShortcut('control RIGHT', 'frm_contacts.handleMyShortcut', 'frm_contacts', new Array
(argument1, argument2));
// remove global shortcut and form-level shortcut
plugins.window.removeShortcut('menu 1');
plugins.window.removeShortcut('control RIGHT', 'frm_contacts');
// shortcut handlers are called with an JSEvent argument
/**
 * Handle keyboard shortcut.
 *
 * @param {JSEvent} event the event that triggered the action
 */
//function handleShortcut(event)
//{
//  application.output(event.getType()) // returns 'menu 1'
//  application.output(event.getFormName()) // returns 'frm_contacts'
//  application.output(event.getElementName()) // returns 'contact_name_field' or null when no element is
selected
//}
// NOTE: shortcuts will not override existing operating system or browser shortcuts,
// choose your shortcuts careful to make sure they work in all clients.
```

### removeToolBar

void **removeToolBar**(name)

Remove the toolbar from the toolbar panel.

#### Parameters

{String} name

#### Returns

void

### Sample

```
// Note: method removeToolBar only works in the smart client.

// the toolbar must first be create with a call to addToolBar
var toolbar = plugins.window.addToolBar("toolbar_0");

// add a button to the toolbar
toolbar.addButton("button", feedback_button);

// removing a toolbar from the toolbar panel is done by name
// the plugin checks the existence of the toolbar
// when the toolbar does not exist it will not throw an error though.
plugins.window.removeToolBar("toolbar_0");
```

### setFullScreen

void **setFullScreen**(full)

Bring the window into/out of fullscreen mode.

#### Parameters

{Boolean} full



## Returns

void

## Sample

```
// active fullscreen mode
plugins.window.setFullScreen(true);
```

## setStatusBarVisible

void **setStatusBarVisible**(visible)

Show or hide the statusbar.

## Parameters

{[Boolean](#)} visible

## Returns

void

## Sample

```
// hide the statusbar
plugins.window.setStatusBarVisible(false);
```

## setToolBarAreaVisible

void **setToolBarAreaVisible**(visible)

Show or hide the toolbar area.

## Parameters

{[Boolean](#)} visible

## Returns

void

## Sample

```
// hide the toolbar area
plugins.window.setToolBarAreaVisible(false);
```

## showFormPopup

void **showFormPopup**(elementToShowRelatedTo, form, scope, dataproviderID)

Show a form as popup panel, where the closeFormPopup can pass return a value to a dataprovider in the specified scope.

## Parameters

{[RuntimeComponent](#)} elementToShowRelatedTo – element to show related to or null to center in screen

{[controller](#)} form – the form to show

{[Object](#)} scope – the scope to put retval into

{[String](#)} dataproviderID – the dataprovider of scope to fill

## Returns

void

## Sample

```
//Show a form as popup panel, where the closeFormPopup can pass return a value to a dataprovider in the
specified scope.
plugins.window.showFormPopup(null,forms.orderPicker,foundset.getSelectedRecord(),"order_id");
//do call closeFormPopup(ordervalue) from the orderPicker form
```