Profound UI – Client-side Scripting

This section introduces the Profound UI API available for custom scripting along with the API documentation.

get(id)

This function is a shortcut for getElementValue(), trimmed. It is used to retrieve an element's value by its id and trim the leading and trailing spaces from that value.

Parameters:

- id – element identifier – String; the element can be an output field or an input control such as a textbox or a dropdown.

Example:

```
var value = get("TextBox1");
```

getElementValue(id)

This function retrieves an element value by id. The value can then be passed to a chart, an AJAX request, or some other process. Leading and trailing spaces are not be trimmed with this function.

Parameters:

- id – element identifier – String; the element can be an output field or an input control such as a textbox or a dropdown.

Example:

```
var value = getElementValue("TextBox1");
```

getObj(id)

This function returns a reference to the object with the specified id on the screen.
This function call is equivalent to the native JavaScript syntax:

```
document.getElementById(id);
```

Parameters:

- id – element identifier – String;

Example:

A reference to a textbox object on the screen can be returned by writing this line of code:

```
var textbox = getObj('I_5_10');
```

After a reference is obtained, the object's methods and property can be accessed through native JavaScript. For example:

```
textbox.style.backgroundColor = 'yellow';          // highlight textbox
textbox.value = '';          // clear textbox
textbox.focus();             // place cursor within textbox
```

hideElement(id)

This function hides a specific display or input element on the screen.

Parameters:

- id – element identifier or an object reference to the element, which is to be hidden

Example:

```
hideElement("OutputField2");

hideElement(outputFieldObj);
```

hideElements(id1, [id2, id3, etc...])

This function hides many display or input elements on the screen with one call.

Parameters:

- id1 – element identifier to hide or an object reference
- [id2, id3, etc…] – identifer2, identifer3, etc…

Example:

```
hideElements("TextBox1", "OutputField1", "OutputField2");
```

removeElement(id)

This function removes a specific element from the screen.

Parameters:

- id – element identifier or an object reference to the element, which is to be removed

Example:

```
removeElement("TextBox1");
```

removeElements(id1, [id2, id3, etc…])

This function removes one or more elements from the screen.

Parameters:

- id1– element identifier to remove or an object reference
- id2, id3, etc… - identifer2, identifer3, etc… (Optional)

Example:

```
removeElements("TextBox1", "OutputField1", "OutputField2");
```

changeElementValue(id, value)

This function assigns a new value to a field. It can be used with both output fields and input elements to automate and simulate user input. For example, clicking a button on the screen may trigger an option number to be entered into an input field and the enter key to be pressed automatically.

Parameters:

- id – element identifier
- value – new value to assign

Example:

```
changeElementValue("TextBox1", "IT Manager"); // places IT Manager in textbox with id: TextBox1


changeElementValue("TextBox1", "");           // clear textbox
```

changeElementClass(id, customClass)

This function changes the CSS class of an element, allowing it to assume new cascading style sheet attributes such as borders and colors.

Parameters:

- id – element identifier
- customClass – CSS class name

Example:

```
changeElementClass('OutputField1', 'BigText');
```

The function call listed above is equivalent to the native JavaScript syntax:

```
document.getElementById("OutputField1").className = 'BigText';
```

setTab(tabPanelid, tab)

This function sets the active tab on a Tab Panel. This function can be executed before the Tab Panel is created by the designer in which case the setTab action is automatically delayed until the Tab Panel is rendered.

Parameters:

- tabPanelId – the Tab Panel identifier
- tab – the tab number to activate. The first tab is tab 0, the second tab is tab 1, etc.

Example:

In this example, we are activating the first tab in the tab panel.

```
setTab("TabPanel", 0);
```

ltrim(string)

This function trims leading spaces from a string.

Parameters:

- string – the string from which the leading spaces will be removed.

Example:

```
var myName = ltrim(" Michael"); // places "Michael" into myName
```

rtrim(string)

This function trims trailing spaces from a string.

Parameters:

- string – the string from which the trailing spaces will be removed.

Example:

```
var myName = rtrim("Michael "); // places "Michael" into myName
```

trim(string)

This function trims all leading and trailing spaces from a string.

Parameters:

- string – the string from which all leading and trailing spaces will be removed.

Example:

```
var myName = trim(" Michael "); // places "Michael" into myName
```

preventEvent(event)

This function cancels an event and prevents it from bubbling up.

Parameters:

- event – event name to cancel

Example:

In this example, we prevent the onclick event if other fields on the screen are not validated properly.

Note: We assume that validated() is a custom-built function that is available for use. We also assume that validated() returns true or false to indicate the validity of certain fields on the screen.

```
var submitButton = getObj("my_button");
submitButton.onclick = function(event)
{
    if(!validated())        //if fields on the screen are not valid
    {
        preventEvent(event);
    }
}
```

attachCalendar(id, format)

This function attaches a pop-up calendar to any input field.

Parameters:

- id – input field id
- format – date format (Optional).

Example:

```
attachCalendar("TextBox1");
```

postToNewWindow(url, parm1, value1 [, parm2, value2, etc…])

This function posts data to a new browser window.

Parameters:

- url – the url to which the data will be posted
- parm1 – the first parameter's name
- value1 – the first parameter's value
- parm2 – the second parameter's name (Optional)
- value2 – the second parameter's value (Optional)

Example:

In this example, we have a tracking number for shipments in an order maintenance application. We are posting this tracking number to the UPS tracking website once the tracking button on the screen is clicked.

```
// get the button
var trackButton = getObj("TrackingButton");
// assign the onclick event to the button so that it posts data to UPS
trackButton.onclick = function(event)
{
    // set the url
    var url = "http://www.ups.com/tracking/tracking.html";

    // set the parameter and its value
    var param1 = "trackNums";
    var trackingNum = get("I_6_20");

    // post
    postToNewWindow(url, param1, trackingNum);
}
```

createNamedElement(type, name)

This function provides a cross-browser compatible method of creating named HTML element that can be submitted through an HTML form.

Parameters:

- type – the element type or HTML tag name (input, select, textarea, etc…)
- name – the name attribute of the created HTML element

Example:

```
var inputField = createNamedElement("input", "company");
```

setDOMAttribute(element, attribute, value)

This function sets a DOM/HTML property or attribute on an element. The DOM is the browser's Document Object Model that represents all of the HTML on a page.

Parameters:

- element – the reference to a DOM element
- attribute – the HTML attribute to set
- value – the value to assign to the attribute specified

Examples:

In this example, we will set the value attribute of a textbox to "Please Enter your name".

```
setDOMAttribute("TextBox1", "value", "Please Enter your name");
```

The line of code above is equivalent to:

```
changeElementValue("TextBox1", "Please Enter your name");
```

In the next example, we will disable a textbox so users will not be able to type any information in that textbox. In this case, we assign an onclick event handler function to a button that changes the "disabled" attribute on an input field to true.

```
// get the button object
var button = getObj("my_button");
// assign an onclick handler function to the button
button.onclick = function()
{
    setDOMAttribute("I_6_20", "disabled", true);
}
```

Other DOM properties such as size, maxlength, align, etc. can be changed as well depending on the type of DOM element.

addEvent(obj, eventName, func)

This function attaches an event to an object. For example, you can attach an “click” event to a button. The same can be accomplished by assigning the onlclick property on a button; however, with the addEvent() function, you can attach multiple unrelated “click” events to the same object.

Parameters:

- obj – the object or element to which the event will be applied
- eventName – the event name. Example: “click”, “mouseover”, etc.
- func – the event handler function

Examples:

The following code assigns an event to a button.

```
// get the button object
var myButton = getObj("my_button");

// assign the greeting() function as an event handler
addEvent(myButton, "click", greeting);
function greeting()
{
     alert("Hello There!");
}
```

Alternatively, an anonymous function can be used when assigning the event.

```
// get the button object
var myButton = getObj("my_button");
// assign anonymous function as an event handler
addEvent(myButton, "click", function ()
{
    alert("Hello There!");
});
```

removeEvent(obj, eventName, func)

This function removes an event from an object.

Parameters:

- obj – the object or element from which the event will be removed.
- eventName – eventName to remove from the object.
- func – the event handler function.

Example:

The example below assumes that the greeting() function has previously been added as an event to the myButton object.

```
removeEvent(myButton, "click", greeting);
```

getMouseX(event)

This function returns the X coordinate position of the mouse based on a mouse event object.

Parameters:

- event – the event object captured by an event such as onmousemove, onclick, or onmouseover

Example:

The code below assigns an event that reports mouse coordinates as the user hovers the mouse cursor over an image element. It assumes that an image element with the id of “myImage” and an output field with the id of “MouseCoordinates” have been created in the Profound UI designer.

```
addEvent(getObj("myImage"), "mousemove", function(event) {
    getObj("MouseCoordinates").innerHTML = getMouseX(event) + "," + getMouseY(event);
});
```

getMouseY(event)

This function returns the Y coordinate position of the mouse based on a mouse event object.

Parameters:

- event – the event object captured by an event such as onmousemove, onclick, or onmouseover

Example:

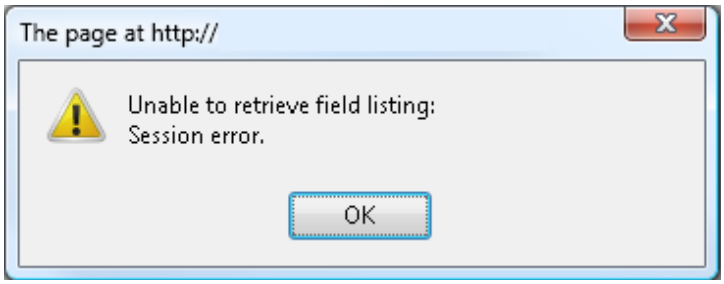
The code below assigns an event that reports mouse coordinates as the user hovers the mouse cursor over an image element. It assumes that an image element with the id of “myImage” and an output field with the id of “MouseCoordinates” have been created in the Profound UI designer.

```
addEvent(getObj("myImage"), "mousemove", function(event) {
    getObj("MouseCoordinates").innerHTML = getMouseX(event) + "," + getMouseY(event);
});
```

showErrors()

This function displays errors that occur as a result of server requests. If no errors have occurred, a “No errors have been reported” message will be displayed as shown below.

This function does not accept any parameters. The function is typically used for debugging Profound UI components that make requests to the server for data. For example, if no data appears on your database-driven chart, you may type “javascript:showErrors()” in the browser’s address bar, and you will then see all errors that may have occurred relating to authority, file access, etc.



getInnerText(object)

This function retrieves the text contained in an output element. This value returned by this function is similar to the content returned by the innerHTML property of an object; however, all html formatting, such as tags and non-breaking space characters, is ignored.

Parameters:

- object – reference to an output element from which to retrieve the text

Example:

```
var text = getInnerText(getObj("D_5_5"));
```

applyProperty(obj, propName, propValue)

This function is used to apply a designer property in customization scripts.

Parameters:

- obj – the object or element to which design property will be applied. This could also be an element ID (string)
- propName – name of the designer property to apply to the object.
- propValue – value of the designer property to apply to the object.

Example:

The code below changes a textbox to a date field by changing the design field type designer property into a date field rather than textbox:

```
var myTextBox = getObj("TextBox1"); //get the textbox object reference
applyDesignProperty(myTextBox, "field type", "date field");
```

pui.click([id])

This function submits a response to the server. This API can accept an optional parameter containing an ID or an object reference to a button, or a similar element, that has a bound response property to be sent to the server.

Parameters:

- id – ID or an object reference to a button, or a similar element, that has a bound response property to be sent to the server.

Example:

```
pui.click();                    //send response to server

pui.click("Exit");              //send response to server by triggering the Exit button
```