

# Overview on the Generated Browser Code Elements

Generating a web based user interface from a UML model implementing the MVC pattern means mapping the pattern to code which the web browser can interpret.

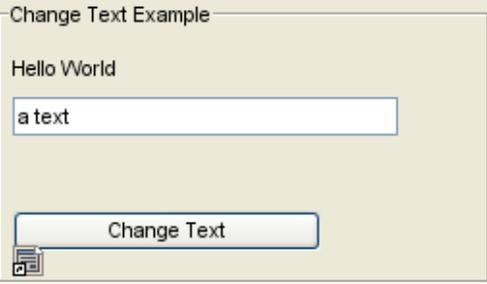
## UML GUI to HTML

To understand the generated HTML code it is essential to know which part of the code resembles which modeled user interface component.

```

26 <body
27 <form id="HelloWorldPanel" class="ui-widget">
28 <fieldset id="GroupBox1" class="ui-widget-content ui-corner-all">
29 <legend>Change Text Example</legend>
30 <div class="e2e-row">
31 <span id="label2">Hello World</span>
32 </div>
33 <div class="e2e-spacer10"></div>
34 <div class="e2e-row">
35 <input name="TextField1" placeholder="a text" type="text" id="TextField1"></input>
36 </div>
37 <div class="e2e-spacer10"></div>
38 <div class="e2e-row">
39 <button id="changeTextButton" class="ui-state-default ui-corner-all">Change Text</button>
40 </div>
41 <div class="e2e-spacer10"></div>
42 </fieldset>
43 </form>
44 <form title="Confirmation" id="textChangeConfirmation" class="ui-dialog">
45 <div class="e2e-row">
46 <span id="label1">Do you want to change the text</span>
47 <div class="e2e-separator"></div>
48 </div>
49 <div class="e2e-spacer10"></div>
50 <div class="e2e-row">
51 <button id="yesButton" class="ui-state-default ui-corner-all">YES</button>
52 <button id="noButton" class="ui-state-default ui-corner-all">NO</button>
53 </div>
54 <div class="e2e-spacer10"></div>
55 </form>
56 </body>
57 </html>
    
```

The code in the above screenshot describes the structure and grouping of the HelloWorld UI example application. It is important to know that depending on the layout type that was chosen, the HTML code will differ. The above example uses the flow layout. In case of the usage of the grid layout, the corresponding HTML elements will be nested into table cells.

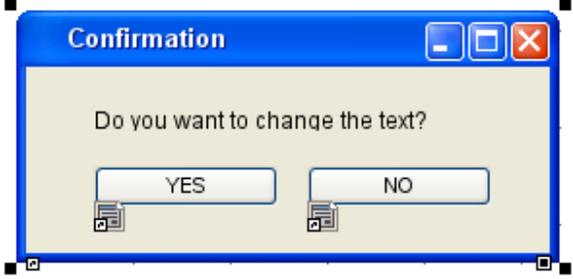
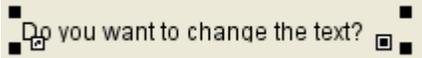
Line	UML Model Element	Description
27		<p>The <code>HelloWorldPanel &lt;&lt;Panel&gt;&gt;</code> element is generated to a <code>&lt;form&gt;</code> HTML tag holding all the UI elements within a group (Line 27-43). The form attribute <code>id</code> holds the name which was given to the UML <code>&lt;&lt;Panel&gt;&gt;</code> object within MagicDraw.</p> <p>In case the <code>id</code> is not a unique name the compiler will generate a warning and will create a new unique <code>id</code> from the name given. If there is no <code>id</code> set, the compiler will also create an <code>id</code>.</p> <p>The form-attribute class holds the name of the CSS class definition which describes the look and feel of the element. Please refer to the UI Widgets chapter for more detailed information. Due to the fact that in most cases a UML container will hold form elements, container elements are translated to HTML forms.</p>

### On this Page:

- [UML GUI to HTML](#)
- [Extensibility of CSS and JavaScript](#)
- [Extensibility of the Controller](#)
  - [Retrieving Data](#)
    - [Using jQuery](#)
    - [Using Bound Controller Variables](#)
  - [JavaScript Attribute Types](#)
  - [Creating custom JavaScript functions](#)
    - [Events](#)

### Related Pages:

- [Overview on the Used Web 2.0 Components](#)
- [Overview on the Used Frameworks](#)
- [Overview on the Generated Browser Code Elements](#)
- [Debugging Custom JavaScript Functions](#)
- [Optimizing Generated JavaScript](#)

28-29		<p>The <code>&lt;&lt;GroupBox&gt;&gt;</code> UML element is generated into a <code>&lt;fieldset&gt;</code> HTML tag. The fieldset attribute name <code>GroupBox1</code> is a generic name due to the fact that the UML element was not named when modeled. In case of customized scripts needing to access the fieldset using a specific name, the UML element needs to be given a name according to the specifications. The class attribute references the CSS class definitions which define the look and feel of the Group Box. The HTML tag <code>&lt;legend&gt;</code> holds the title of the <code>&lt;fieldset&gt;</code>. This title is set within the properties of the UML element within MagicDraw.</p>
31		<p>The Hello World UML <code>&lt;&lt;Label&gt;&gt;</code> is translated into a <code>&lt;span&gt;</code> HTML tag. For the controller JavaScript to be able to access this element and do the text change the <code>&lt;span&gt;</code> tag has an id attribute.</p>
35		<p>The UML element <code>&lt;&lt;TextField&gt;&gt;</code> is generated as a HTML <code>&lt;input&gt;</code> tag. The attributes <code>name</code> and <code>id</code> have a generically generated name. The default text <code>a text</code> is translated as <code>placeholder</code> attribute.</p>
39		<p>The UML element <code>&lt;&lt;Button&gt;&gt;</code> is generated to a HTML <code>&lt;button&gt;</code> tag. The id attribute holds the defined name <code>changeTextButton</code>. The <code>&lt;&lt;Button&gt;&gt;</code> property <code>Text</code> holds the <code>Change Text</code> button label.</p>
44		<p>The <code>&lt;&lt;Frame&gt;&gt;</code> UML container used for the modal dialog is generated to a <code>&lt;form&gt;</code> tag. The behaviour as a modal dialog is steered by the CSS class <code>ui-dialog</code> defined within the <code>&lt;form&gt;</code> tag. The <code>id</code> attribute is given the name specified within the <code>&lt;&lt;Frame&gt;&gt;</code> properties as well as the dialogs title <code>Confirmation</code>.</p>
46		<p>see description of lines 28-29</p>
51-52		<p>see description of line 39</p>

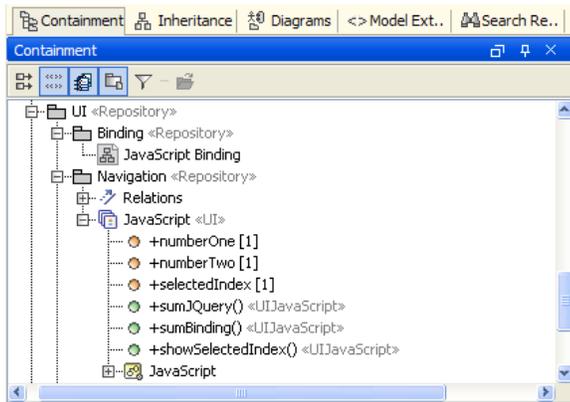
## Extensibility of CSS and JavaScript

The HTML application file hold all references to libraries, frameworks, configuration and style sheets within the **<head>** section. Next to the standard libraries which should not be edited, there are files which can be amended to extend functionality and design of the xUML UI application.

```
1 <head>
2 <meta content="text/javascript" http-equiv="Content-Script-Type"></meta>
3 <title></title>
4 <link href="HelloWorldUI.css" rel="stylesheet" type="text/css"></link>
5 <link href="e2e.css" rel="stylesheet" type="text/css"></link>
6 <link href="jquery.css" rel="stylesheet" type="text/css"></link>
7 <script src="..libs/jquery/jquery-1.4.2.min.js" type="text/javascript"></script>
8 <script src="..libs/jquery/validate/jquery.validate.min.js" type="text/javascript"></script>
9 <script src="e2e.js" type="text/javascript"></script>
10 <script src="configuration.js" type="text/javascript"></script>
11 <script src="..libs/jquery/jquery.livequery.js" type="text/javascript"></script>
12 <script src="..libs/jquery/ui.core.js" type="text/javascript"></script>
13 <script src="..libs/jquery/ui.datepicker.js" type="text/javascript"></script>
14 <script src="..libs/jquery/ui.draggable.js" type="text/javascript"></script>
15 <script src="..libs/jquery/ui.resizable.js" type="text/javascript"></script>
16 <script src="..libs/jquery/ui.dialog.js" type="text/javascript"></script>
17 <script src="..libs/jquery/jquery.history.js" type="text/javascript"></script>
18 <script src="..libs/jquery/dataTables/jquery.dataTables.min.js" type="text/javascript"></script>
19 <script src="HelloWorldUI.js" type="text/javascript"></script>
20 </head>
21 <body>
22 <!--This page requires JavaScript.-->
23 </body>
24 </html>
```

## Extensibility of the Controller

The **<<UI>>** controller can be extended with custom JavaScript functions which enable to enrich the functionality xUML UI application. The custom JavaScript functions have full access to **<<UI>>** controllers context e.g. accessing defined global variables, other JavaScript functions or even accessing an external system using AJAX requests. Due to the fact that xUML UI applications are based on jQuery, all the features of this powerful library can be used.



The JavaScript operation name is defined by assigning the wanted name to a default operation. The normal class operation will be generated as a JavaScript function by assigning the **<<UIJavaScript>>** stereotype.

The JavaScript code itself can be edited either using the specification dialog of the operation element or by using the JavaScript Editor. The editor is accessed via right-clicking on the operation element and choosing the JavaScript Editor (Shift-Enter). The JavaScript Editor does support syntax highlighting but does not support code completion.

```
JavaScript
1 var input1 = parseInt($("#numberOne").val());
2 var input2 = parseInt($("#numberTwo").val());
3
4 var prod = this.multiply(input1, input2);
5
6 $("#productField").val(prod);
7
```

The actual JavaScript is set within the operations **Script** property field. The JavaScript function name can be changed through editing the **Name** properties field.

The generated JavaScript function will look like this:

```

controller.sumjQuery = function() {
    alert('The sum is: ' + (parseInt($("#numberOne").val()) +
        parseInt($("#numberTwo").val())));
}

```

## Retrieving Data

There are different ways to get data from the user interface or the JavaScript application (Controller) itself.

### Using jQuery

This simple snippet will use jQuery to find the element **numberOne** and retrieve its value. This snippet can be used directly within the <<UIJavaScript>> operation without modelling further elements.

```
var val = $("#numberOne").val();
```

jQuery offers a wide range of functions to handle data within E2E UI applications. For a detailed documentation including tutorials and examples on jQuery visit their documentation on [http://docs.jquery.com/Main\\_Page](http://docs.jquery.com/Main_Page).

Another way of course is to access the values using standard JavaScript functionality to access the DOM structure.

```
var val = document.getElementById("numberOne");
```

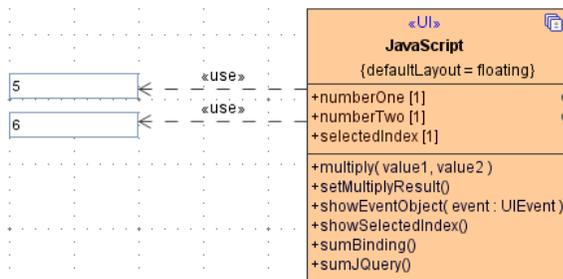
or

```
var val = document.getElementsByName("numberOne");
```

The id and name HTML elements attributes share the same unique identifier. The exception to this are Check Box and Radio Button elements. Their name attribute is used for grouping multiple elements and it is essential to either use jQuery's referencing mechanism `$("#numberOne")` or the `getElementById("numberOne")` JavaScript function.

### Using Bound Controller Variables

Custom global variables defined within the <<UI>> controller can of course be bound to user interface elements. The values are automatically available to the global variables as soon as data is written into the text fields. The binding is done between the <<UI>> class and the user interface elements.



To be able to access these global variables the pointer **this** is used:

```
var sum = parseInt(this.numberOne)+ parseInt(this.numberTwo);
```

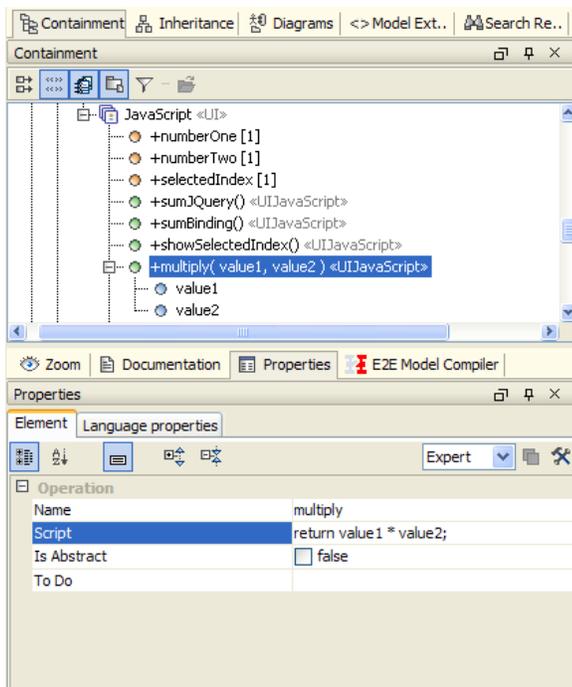
## JavaScript Attribute Types

In this example the JavaScript attributes do not have a specific type defined. In this case, the compiler will choose the type string for them. As a result of this, the variable need to be parsed in JavaScript using parsing functions like `parseInt()`. If the attributes get a type declared, the parsing via JavaScript is not needed anymore because the compiler will do this automatically according to the binding information, e. g. an attribute `x` of base type string, holding a numerical value, is bound to a integer based attribute. In this case the `parseInt()` function will automatically called within the application.

## Creating custom JavaScript functions

It is common to create custom utility functions with input parameters for repeated usage. These functions are not available by default in JavaScript e.g. financial calculations. There are two ways of being able to implement these custom functions. Either code them in a custom JavaScript file and then import it into the model or create them within the `<<UI>>` controller.

Figure: Custom JavaScript Function



The input parameters **value1** and **value2** do not have any special type definition applied. The parameters are accessed from within the JavaScript code by using the names of the parameters. The generated code can be found in the `<controller name>.custom.js` file and looks as follows:

```
controller.multiply = function(value1, value2) {
    return value1 * value2;
}
```

This JavaScript function can be called from within any other custom defined function, but can not be directly applied as a Call function within a UI element.

## Events

In some cases it is essential to work with events. As xUML UI is using jQuery as its main UI library, the jQuery Event is implemented and can be used. Having event data can be a great help when different events use the same event handler and it needs to be distinguished between e.g. the event type. The following event parameters are implemented in the xUML UI:

Property	Description
<b>target</b>	Element identifier for which the event was triggered for.
<b>data</b>	The actual value in case any was passed with the event.

<b>type</b>	The type of event, e.g. click even from a button element. This is more than useful when reusing one event handler for many different events
<b>timestamp</b>	A timestamp.