## Developer Guideline

### Web Dynpro
**HTMLIsland & HTMLContainer**

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
<th>Reviewed/Approved at / by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Final</td>
<td>09.12.2012</td>
<td></td>
</tr>
</tbody>
</table>
## Contents

1. Introduction ................................................................................................................. 3  
2. Architecture ................................................................................................................. 4  
   2.1 Overview .................................................................................................................. 4  
   2.2 HTMLIsland and related Web Dynpro UI Elements ................................................... 5  
3. Development ................................................................................................................. 7  
   3.1 Creating an HTMLIsland ............................................................................................ 7  
   3.2 Security Notes .......................................................................................................... 22  
   3.3 Naming conventions and namespaces ...................................................................... 24  
   3.4 Restrictions .............................................................................................................. 25
1 Introduction

Web Dynpro provides UI elements that allow integration of custom HTML, custom JavaScript, and custom CSS into a Web Dynpro application. The relevant UI elements are HTMLIsland, HTMLContainer and HTMLFragment. Section 2.2 “HTMLIsland and related Web Dynpro UI Elements” describes these UI elements related to HTML integration in detail.

Section 3.1 “Creating an HTMLIsland” includes a step-by-step description on how to integrate an HTMLIsland into Web Dynpro ABAP. Following the steps described in this section, you can build an application containing an HTMLIsland from scratch.

There are certain restrictions when working with HTMLIslands and HTMLContainers. Please carefully read section 3 “Restrictions” to ensure you meet all requirements when implementing HTMLIslands or HTMLContainers.

To ensure that SAP product standards are fulfilled in Web Dynpro applications containing HTMLIslands and/or HTMLContainers, there are certain things you need to be aware of. If you plan to develop such applications, please read SAP note 1742528.
2 Architecture
This section describes the architecture and technical aspects of the Web Dynpro UI elements used to integrate custom HTML into Web Dynpro.

2.1 Overview
Figure 1 shows the general aspects of HTML integration:

![Diagram of HTML integration](image)

**Figure 1: Overview of UI elements for HTML integration**

Both HTMLIsland and HTMLContainer are of type AbstractHTMLElement. They hold references to JavaScript and CSS sources. These sources can be modelled in the back end as aggregations to the respective Web Dynpro UI element. HTMLIslands contain static HTML, whereas HTMLContainers have an aggregation to content, which can be either HTML, Web Dynpro UI elements or a combination of both. The differences between HTMLIsland and HTMLContainer are described in further detail in section 2.2 “HTMLIsland and related Web Dynpro UI Elements”.

Communication between the back end system and the HTMLIsland or HTMLContainer instance on the client (browser) is done via JavaScript calls. JavaScript calls can contain data as payload and they are compiled in the back end via an ABAP API. Communication from client to back end is realized by a JavaScript callback API.
2.2 HTMLIsland and related Web Dynpro UI Elements

Three UI elements can be used to integrate custom HTML in Web Dynpro ABAP: HTMLIsland, HTMLContainer and HTMLFragment. HTMLFragments are usually used as part of HTMLContainers and contain a fragment of HTML. Table 1 below shows the major differences between HTMLIsland and HTMLContainer.

<table>
<thead>
<tr>
<th></th>
<th>HTMLIsland</th>
<th>HTMLContainer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application case</td>
<td>You can embed interactive, stateful content (for example, charts) into your Web Dynpro applications.</td>
<td>You can integrate HTML or JavaScript-based UI controls into your Web Dynpro application (for example, an enhanced UI control for processing formatted text). You can enhance Web Dynpro UI elements with HTML (for example, add a background image to a Table).</td>
</tr>
<tr>
<td>Creating HTML</td>
<td>HTML is created on the client side. Only a static “anchor” HTML should be placed in the HTMLIsland property staticHtml. The HTMLIsland content remains stable on the client.</td>
<td>HTML is created on the server side. As a result, the content of the HTMLContainer is rebuilt with each server roundtrip. To create HTML on server side, the API IF_WD_HTML_WRITER should be used</td>
</tr>
</tbody>
</table>

Table 1: Comparison of HTMLIsland and HTMLContainer

As you can see, you can decide which UI element you want to use depending on the place where the HTML should be build. If you want to build it on the server side, you should use HTMLContainer. If you want to create the HTML elements on client side, you should use HTMLIsland.

The subsequent sections describe the three UI elements for HTML integration in more detail.

2.2.1 HTMLIsland

HTMLIslands can define JavaScript and style sheet sources, and they contain a property staticHtml which can contain HTML code. Once the HTMLIsland UI element gets displayed, HTML code is sent to the browser and JavaScript and style sheet sources are loaded. As the name staticHtml already indicates, the HTML needs to be static, meaning that changes to it on server side are not possible. However, you are free to build arbitrary client-side HTML. The staticHtml usually serves as an anchor for JavaScript libraries to build upon it.

Example

The property staticHtml contains the following value:

```html
<div id="myHtmlIsland"></div>
```

The JavaScript source can then get a reference to the HTMLIsland. For example:

```javascript
var myIsland = document.getElementById('myHtmlIsland')
```

Doing it this way, the <div>-tag defined in the staticHtml property functions as the root element for the HTMLIsland.

The HTMLIsland uses a so-called placeholder mechanism, which keeps the state of the HTMLIsland on client side. Thus, if you dynamically build HTML code on client side, it will still be available after a server roundtrip.

2.2.2 HTMLContainer and HTMLFragment

Similar to HTMLIslands, HTMLContainers are used to load JavaScript and style sheet sources on the client. In contrast to HTMLIslands, however, they do not have a property staticHtml. They can aggregate other Web Dynpro UI elements (for example InputFields or TextViews) and, which is
often important in this context, HTMLFragments. Therefore, they allow you to place custom HTML around Web Dynpro UI elements.

Example

You like to have a custom background behind a Web Dynpro Table or C-Table. To achieve this, you can build a UI tree with the HTMLContainer in the following way:

```
<html>
  <body>
    <div id="HTML_FRAGMENT_BEFORE_WD">
      <div id="MyTablebackground">
        <div id="HTML_FRAGMENT_AFTER_WD">
          <div id="MyTablebackground">
            <div id="HTML_FRAGMENT_AFTER_WD">
              <div id="MyTablebackground">
                <div id="HTML_FRAGMENT_AFTER_WD">
                  <div id="MyTablebackground">
                    <div id="HTML_FRAGMENT_AFTER_WD">
                      <div id="MyTablebackground">
                        <div id="HTML_FRAGMENT_AFTER_WD">
                          <div id="MyTablebackground">
                            <div id="HTML_FRAGMENT AFTER_WD">
                              <div id="MyTablebackground">
                                <div id="HTML_FRAGMENT AFTER_WD">
                                  <div id="MyTablebackground">
                                    <div id="HTML_FRAGMENT AFTER_WD">
                                      <div id="MyTablebackground">
                                        <div id="HTML_FRAGMENT AFTER_WD">
                                          <div id="MyTablebackground">
                                            <div id="HTML_FRAGMENT AFTER_WD">
                                              <div id="MyTablebackground">
                                                <div id="HTML_FRAGMENT AFTER_WD">
                                                  <div id="MyTablebackground">
                                                    <div id="HTML_FRAGMENT AFTER_WD">
                                                      <div id="MyTablebackground">
                                                    </div>
                                                  </div>
                                                </div>
                                              </div>
                                            </div>
                                          </div>
                                        </div>
                                      </div>
                                    </div>
                                  </div>
                                </div>
                              </div>
                            </div>
                          </div>
                        </div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Now you set the html property of the HTMLFragment with the ID "HTML_FRAGMENT_BEFORE_WD" to the following value:

```html
<div id="MyTablebackground">
  <div id="MyTablebackground">
    <div id="MyTablebackground">
      <div id="MyTablebackground">
        <div id="MyTablebackground">
          <div id="MyTablebackground">
            <div id="MyTablebackground">
              <div id="MyTablebackground">
                <div id="MyTablebackground">
                  <div id="MyTablebackground">
                    <div id="MyTablebackground">
                      <div id="MyTablebackground">
                        <div id="MyTablebackground">
                          <div id="MyTablebackground">
                            <div id="MyTablebackground">
                              <div id="MyTablebackground">
                                <div id="MyTablebackground">
                                  <div id="MyTablebackground">
                                    <div id="MyTablebackground">
                                      <div id="MyTablebackground">
                                        <div id="MyTablebackground">
                                          <div id="MyTablebackground">
                                            <div id="MyTablebackground">
                                              <div id="MyTablebackground">
                                            </div>
                                          </div>
                                        </div>
                                      </div>
                                    </div>
                                  </div>
                                </div>
                              </div>
                            </div>
                          </div>
                        </div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

In the HTMLFragment with the ID "HTML_FRAGMENT_AFTER_WD", you set the html property to the following value:

```html
</div>
```

This allows you to define a style sheet rule for the <div>-container placed around the table, for instance to set a background image. The result could look like the following:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>D017</td>
<td>19.09.2012</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>17.10.2012</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>14.11.2012</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>12.12.2012</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>02.01.2013</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>06.02.2013</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>06.03.2013</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D017</td>
<td>06.04.2013</td>
<td>422.94</td>
<td>USD</td>
<td>747-400</td>
</tr>
<tr>
<td>AA</td>
<td>D054</td>
<td>21.09.2012</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
<tr>
<td>AA</td>
<td>D054</td>
<td>19.10.2012</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
<tr>
<td>AA</td>
<td>D064</td>
<td>16.11.2012</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
<tr>
<td>AA</td>
<td>D064</td>
<td>14.12.2012</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
<tr>
<td>AA</td>
<td>D064</td>
<td>11.01.2013</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
<tr>
<td>AA</td>
<td>D064</td>
<td>08.02.2013</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
<tr>
<td>AA</td>
<td>D064</td>
<td>08.03.2013</td>
<td>422.94</td>
<td>USD</td>
<td>A310-300</td>
</tr>
</tbody>
</table>

As you can see, HTMLContainer allows placing HTML around Web Dynpro UI elements.

In contrast to HTMLIsland, the HTML does not need to be static, as it is re-rendered each time the HTMLContainer is rendered. On the other hand, the HTML state on client side is lost after each server roundtrip that changes the view in which the HTMLContainer resides.
3 Development

3.1 Creating an HTMLIsland

The following sections will guide you through all the steps of implementing an HTMLIsland into a Web Dynpro application. You will build an entire application containing an HTMLIsland from scratch. The application is kept as simple as possible and will focus on several aspects of integrating an HTMLIsland into Web Dynpro ABAP, like adding JavaScript and CSS sources and enabling communication. To complete this tutorial, you should have basic knowledge of Web Dynpro.

You can find the complete Web Dynpro component and application in the SAP system under WDR_HTML_ISLAND_TUTORIAL.

3.1.1 Creating the Web Dynpro Component

Log into the back end system and start the development workbench (transaction SE80). Choose Web Dynpro Comp. / Intf. in the object list of the Repository Browser and create a new Web Dynpro component, e.g. Z_HTML_ISLAND_TUTORIAL. Leave the default settings and confirm the dialog box. Double-click on the MAIN view in the view node of the object list. You should now have an empty component and see something like this:

![Web Dynpro Explorer: Display View for Z_HTML_ISLAND_TUTORIAL](image)

3.1.2 Adding an HTMLIsland

Make sure you are in edit mode. In the Layout tab, right click on ROOTUIELEMENTCONTAINER and select Insert Element. As Type select HtmlIsland and confirm. A dialog box appears that notifies you of...
certain things you should consider. There is also a reference to SAP note 1742528. Before you develop elements that integrate HTML into Web Dynpro, make sure you read this note carefully.

In the property staticHtml of your HTMLIsland, enter the following value:

```html
<div id="myHtmlIsland"><p>Hello World!</p></div>
```

Save your changes and activate the component. Right-click on your component and select Create → Web Dynpro Application. Leave the values and confirm.

To make sure you run your application in STANDARDS mode, switch to the Parameters tab and add parameter WDPREFERREDRENDERING with the value STANDARDS:

![Parameters Tab](image)

Note that browser STANDARDS mode is supported only by Internet Explorer version 9 and higher versions, as well as by Firefox, Safari, and Chrome.

Click on save again to create the application. After these steps the Web Dynpro component looks as follows:

![Web Dynpro Component](image)

### 3.1.3 Testing the “Hello World!” HTMLIsland

You can now right-click on your application “Z_HTML_ISLAND_TUTORIAL” and select Test. A browser opens showing the Web Dynpro ABAP application with the HTMLIsland:
3.1.4 Adding a JavaScript source

To add a JavaScript source to your component, open a text editor of your choice and create a JavaScript file with the following content:

```javascript
var MyHTMLIsland = MyHTMLIsland || {
    saySomething: function (message) {
        alert(message);
    }
};
```

When running this code, the JavaScript object “MyHTMLIsland” with the function “saySomething” is created. Save the file and name it “z_html_island_tutorial.js”.

Go back to your component in the SAP system and upload the file to the MIME repository. To do this, right-click on component “Z_HTML_ISLAND_TUTORIAL” and choose Create → Mime Object → Import in the context menu. Select the file “z_html_island_tutorial.js” and click Open. Confirm the dialog boxes that appear.

As you will change this JavaScript file several times in the course of this tutorial, it is more convenient to change the cache time to expire. Thus, you do not need to clear the browser and back end cache each time you replace this file. To change the expiry time, right-click the MIME file “z_html_island_tutorial.js” and choose Expiration Time Client Cache from the context menu. Change to edit mode, select the Individual Expiry Date checkbox, set the time, for example to 1 second, and save your settings:
If you develop a productive application, make sure you choose a reasonable value for the cache expiration. In a productive system, you should use a higher value as frequent changes are not likely.

Open the Layout tab in the MAIN view, right-click on the HTML_ISLAND and select Insert Script. Fill the property source with the value “z_html_island_tutorial.js”:

![Property table showing source set to z_html_island_tutorial.js]

### 3.1.5 Adding a JavaScript call

In the next step, you create and execute a JavaScript call that contains the following parameter:

```javascript
MyHTMLIsland.saySomething("Hello!");
```

You first need to add a button that triggers this. In the Layout tab of the MAIN view, right-click on “ROOTUIELEMENTCONTAINER”, choose Insert Element and select type “Button”. Confirm the dialog box to create the button. To have the button displayed on top of the page, right-click it and select To First Position. Fill the property text with an appropriate value, for instance “Say Something”. Create a new event for “onAction” and name it “SAY_SOMETHING”. Now the layout should look as follows:
After the button is created, you need to implement the event handler. However, before you can access the HTMLIsland, you need to create and fill a reference to the HTMLIsland UI element. Go to the Attributes tab in the MAIN view and add an attribute “M_HTML_ISLAND” of type ref to “CL_WD_HTML_ISLAND”: 
Now switch to the Methods tab and double-click on “WDDOMODIFYVIEW”. Fill the reference to the HTMLIsland UI element in the initial load of the application using the following code:

```
method WDDOMODIFYVIEW.
  if first_time = abap_true.
    wd_this->m_html_island ?= view->get_element( `HTML_ISLAND` ).
  endif.
endmethod.
```

Now that you have the reference to the HTMLIsland UI element you can add the script call in the “onAction” event handler of the button. Go to the Actions tab and double click on “SAY_SOMETHING”. Add the following code to perform a JavaScript call on the HTMLIsland UI element:

```
method ONACTIONSAY_SOMETHING.
  data l_call type ref to if_wd_html_script_call.

  l_call = cl_wd_html_script_call=>new_call( )  " Create a new script call
  l_call->variable( `MyHTMLIsland`  )."  MyHTMLIsland.
  l_call->function( `saySomething` )  " MyHTMLIsland.saySomething( l_call->add_string( `Hello!` )  " MyHTMLIsland.saySomething(“Hello!”);
  wd_this->m_html_island->add_script_call( l_call ).
endmethod.
```

If you now start the application again, you can execute the JavaScript call by pressing the Say Something button:

![JavaScript Call Example](https://example.com/JavaScriptCallExample.png)

As done in the previous example, any kind of JavaScript call can be created. As parameter a string value, a Boolean, null, the callback API or values from the context can be passed. Using a callback API you can fire Web Dynpro actions from HTMLIsland on client side. The procedure will be described in section 3.1.8 “Calling a Web Dynpro event from the browser”. The following section describes how data can be transmitted from context to HTMLIsland as JavaScript call parameter.
3.1.6 Sending data from context to HTML Island

HTMLIsland and HTMLContainer provide the possibility to define a data model on UI element level and bind it to the Web Dynpro context. These bound context data can be used as parameter for JavaScript calls. In the following steps, you create the context and the data model, fill and bind it, enhance the JavaScript source and transmit the data from Web Dynpro to HTMLIsland. You create a single context attribute of type date (D) as well as a multiple context node.

Open the context tab of the MAIN view of your component “Z_HTML_ISLAND_TUTORIAL”. In the context menu of the root node CONTEXT, select Create → Attribute. Enter DATE as the name of the attribute and “D” as its type.

Right-click the root node CONTEXT again and choose Create → Node. Name the node as “LIST” and set the “Cardinality” to “0..n”. Right-click the “LIST” node you just created and select Create → Attribute. Enter “TEXT” as the name of the attribute and “STRING” as its “Type”. The context should now look like this:

Now you need to fill some sample data in the new context. First you fill the multiple context node. Select the context node “LIST” again and enter the name “SUPPLY_LIST” for the property Supply Function. Double click on “SUPPLY_LIST” to get to the implementation of the supply method. To fill the context with some arbitrary data in the supply function, enter following code:

```
method supply_list .

  data l_list type wd_this->elements_list.
  data l_list_item like line of l_list.

  do 5 times.
    l_list_item-text = 'Item ' && sy-index.
    append l_list_item to l_list.
  enddo.
  node->bind_table( l_list ).
endmethod.
```

To fill a value for context attribute “DATE”, go back to the list of all methods using the Method list button and double click on “WDDOINIT”. Fill the method with the following code:

```
method WDDOINIT .

  wd_context->set_attribute( name = 'DATE' 
                             value = sy-datum ).
endmethod.
```

At this point, you have a context filled with data. In the next step, you create the model on UI element side. Switch to the Layout tab, right-click on UI element “HTML_ISLAND”, and select Insert Data Element. As “ID” enter “DATE_PARAMETER”, as “Type” choose JsonParameter. Enter “date” for the property name and bind the property value to the context attribute “DATE”:
Right-click on the “HTML_ISLAND” UI element and select Insert Data Element. Enter “LIST” as “ID” and “JsonDataSource” as “Type”. Enter the value “list” for property name and bind property dataSource to the context node “LIST”:

Right-click on the previously created data element “LIST” and choose Insert Data Element. Enter “TEXT_PARAMETER” for the “ID” and “JsonParameter” for the “Type”. For property name enter “text”. Bind property value to the context attribute “TEXT”: 
To trigger the data transport from context to HTMLIsland, add another Button to the **MAIN** view. Right-click on “ROOTUIELEMENTCONTAINER” and select **Insert Element** from the context menu. Enter “CONTEXT_BUTTON” as “ID” and “Button” as “Type”. Move the button that was just created below the other button “BUTTON” and set property **text** to “Send Data from Context”. In the “Events” group, create a new “onAction” handler and name it “SEND_CONTEXT”. Double-click on “SEND_CONTEXT” to get to the source code view of the event handler.

In the first step, you create and add a JavaScript call, which contains the context attribute “DATE” as parameter. In the subsequent steps, you add the list, which requires that you enhance the JavaScript file. To create a JavaScript call that reads and adds the context attribute “DATE”, enter the following code in the event handler of “CONTEXT_BUTTON”:

```java
method ONACTIONSEND_CONTEXT ,

    wd_this->m_html_island->add_script_call(
        cl_wd_html_script_call=>new_call(
            )->variable( 'MyHTMLIsland'
            )->function( 'saySomething'
            )->add_ui_elem_parameter( 'date' )
        ),

endmethod .
```

Please note that in contrast to the way you created the JavaScript call in section 3.1.5 “Adding a JavaScript call”, you now use chained method calls and returning parameters. This is just another way of creating the call without using a reference variable to IF_WD_HTML_SCRIPT_CALL. The important difference is that you use the method “add_ui_elem_parameter( )” to add the value of the context attribute as parameter to the JavaScript call instead of passing the value directly as parameter. The advantage of doing it via the context is that you can choose the representation as json, internal, or external.

Save and activate the view. Check the result by executing the application:
As you can see, the date is formatted as a typical JSON notation. This is because as you modelled the JsonParameter “DATE_PARAMETER” in the UI tree, you left the default value “json” of the property representation.

If you switch property representation to “internal”, the internal ABAP representation will be transmitted:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties (JsonParameter)</td>
<td>DATE_PARAMETER</td>
</tr>
<tr>
<td>name</td>
<td>date</td>
</tr>
<tr>
<td>readOnly</td>
<td></td>
</tr>
<tr>
<td>representation</td>
<td>internal</td>
</tr>
<tr>
<td>value</td>
<td>MAIN.DATE</td>
</tr>
</tbody>
</table>

Results in the following display:

If you switch property representation to “external”, the date will be formatted according to the user settings in the back end system:
In the next step, you enhance the JavaScript source to handle the list data source. Open the JavaScript file `z_wdr_html_island_tutorial.js` in an editor of your choice and add the code for the function "printList":

```javascript
var MyHTMLIsland = MyHTMLIsland || {  
    saySomething: function (message) {  
        alert(message);  
    },  
    printList: function (array) {  
        var div = document.getElementById('myHtmlIsland'),  
            p,  
            i,  
            max = array.length;  
        for(i = 0; i < max; i += 1) {  
            p = document.createElement('p');  
            p.appendChild(document.createTextNode(array[i].text));  
            div.appendChild(p);  
        }  
    }  
};
```

Save the file and replace the existing file in the MIME repository with the new one. In the back end system, right-click on MIME file "z_html_island_tutorial.js" and select Upload/Download → Upload and Replace. Choose the file from your local file system and confirm.

Now you need to trigger this new JavaScript method from Web Dynpro side. Add the JavaScript call to the event handler of the button "BUTTON_CONTEXT", which is the Web Dynpro action "SEND_CONTEXT". In the Actions tab of the MAIN view, double click on "SEND_CONTEXT". Below the existing code, add another JavaScript call:

```javascript
method ONACTIONSEND_CONTEXT .
    wd_this->m_html_island->add_script_call(  
        cl_wd_html_script_call=>new_call(  
            )->variable('MyHTMLIsland'  
            )->function('saySomething'  
        )
    );
```
3.1.7 Adding a CSS source

In this section, you create and add a simple style sheet for the HTMLIsland. Create a new text file “z_html_island_tutorial.css” and open it in an editor of your choice. Add the following CSS statements to this text file:

```
#myHtmlIsland {
    color: red;
}
```
Save the file and upload it to the MIME repository of your component Z_HTML_ISLAND_TUTORIAL. In the Layout tab of the MAIN view, right-click on the UI element "HTML_ISLAND" and choose Insert Style from the context menu. Fill the property `source` with the value "z_html_island_tutorial.css".

Save and activate the component and run the application. As you can see, the style sheet information (which is the text color red) is applied to the HTMLIsland:
3.1.8 Calling a Web Dynpro event from the browser

The previous steps described how communication from the back end to the instance of the HTMLIsland on the client is realized. In the last step, you want to implement communication in the other direction, i.e., triggering a Web Dynpro event from client to back end. In the present example, a text should be displayed in the message area once the user has pressed a button on the HTMLIsland.

The HTMLIsland UI element can have an arbitrary number of events that trigger a Web Dynpro action. You need to define these events as aggregations to the HTMLIsland. You can then fire them from the browser using a JavaScript callback API, which is passed to the client. In the following steps, you will enhance the JavaScript source, pass a reference to the JavaScript callback API in method WDDOMODIFYVIEW and, finally, add the aggregation EVENTS to your HTML_ISLAND along with an onAction event handler.

First, you enhance the JavaScript source. Open the JavaScript file z_wdr_html_island_tutorial.js in an editor of your choice and add the code for the function addCallback. The JavaScript source now has three functions (i.e. saySomething, printList and addCallback):

```javascript
var MyHTMLIsland = MyHTMLIsland || {
  saySomething: function (message) {
    alert(message);
  },
  printList: function (array) {
    var div = document.getElementById('myHtmlIsland'),
    p,
    i,
    max  = array.length;
    for(i = 0; i < max; i += 1) {
      p = document.createElement('p');
      p.appendChild(document.createTextNode(array[i].text));
      div.appendChild(p);
    }
  },
  addCallback: function (callback) {
    var div   = document.getElementById('myHtmlIsland'),
    input  = document.createElement('input');
    input.type = 'button'
    input.value = 'Fire Event';
    input.onclick = function () {
      callback.fireEvent('Event','Hello Web Dynpro');
    };
    div.appendChild(input);
  }
};
```

`callback.fireEvent` has two parameters. The first parameter refers to the name of the HTMLEvent and the second parameter specifies the value of the DATA parameter that is used in the onAction event handler.

Save the JavaScript file and replace the existing file in the MIME repository with the new one. In the back end system, right-click on the MIME file “z_html_island_tutorial.js” and select Upload/Download → Upload and Replace. Choose the file from your local file system and confirm.

Now you need to trigger the new JavaScript method from Web Dynpro side. The button to trigger the Web Dynpro event on the HTMLIsland should be displayed when the application is first loaded. Therefore, you need to create the JavaScript call that sets the reference to the callback API in method WDDOMODIFYVIEW. The callback reference itself is added as parameter to a JavaScript call via the
**method** addCallback. Go to the **Methods** tab, double-click on WDDOMODIFYVIEW and add the JavaScript call. Afterwards the code of the method looks as follows:

```
method WDDOMODIFYVIEW .
  if first_time = abap_true.
    wd_this->m_html_island != view->get_element( 'HTML_ISLAND' ).
    wd_this->m_html_island->add_script_call(
      cl_wd_html_script_call=>new_call( )->variable( 'MyHTMLIsland' )->function( 'addCallback' )->add_callback_api( )).
  endif.
endmethod.
```

In the next step, you need to create an aggregation EVENT to your HTML_ISLAND element. Go to the **Layout** tab of the **MAIN** view. Right-click on HTML_ISLAND and choose **Insert Event**. Enter “Event” as its name.

Create a new “onAction” event and name it “HTML_ISLAND_EVENT”. When creating the action, make sure to set the checkbox **Transfer UI Event Parameters**. This will add the event parameter DATA to the event handler. Double-click on it to get to the source code view of the event handler and add the following code:

```
method ONACTIONHTML_ISLAND_EVENT .
  wd_comp_controller->wd_get_api( )->get_message_manager( )->report_success( data ).
endmethod.
```
The value of parameter DATA was specified in the JavaScript code (in this case, the parameter value is “Hello Web Dynpro”).

Save, activate and test your component. If you now press the Fire Event button, the text “Hello Web Dynpro” is displayed as a success message in the message area:

3.2 Security Notes

3.2.1 Secure encoding of HTML

Since HTML has special handling for characters like < and >, you need to ensure that they are correctly displayed when you run your application. You can create secure encoding of HTML either in the backend or on the client.

3.2.1.1 Secure encoding of HTML in the backend

In the backend, the IF_WD_HTML_WRITER API is available, enabling HTML code to be written and checked for correct escaping. With this API you can also check the nesting of HTML elements.

You can create an instance of this interface with static method NEW_WRITER of class CL_WD_HTML_WRITER. With parameter DO_NESTING_CHECK you can specify whether nesting should be checked.

Example

You want to have the following text displayed in your HTML island application:
I am inside a <p> paragraph

Use the following code to write securely encoded HTML code:

```java
data html_writer type ref to if_wd_html_writer.
html_writer = cl_wd_html_writer=>new_writer( ).
html_writer->start_element( `p` ).
html_writer->add_text( `I am inside a &lt;p&gt; paragraph` ).
html_writer->end_element( `p` ).
```

The following code returns the HTML as a string:

```java
write html_writer->get_html( ).
"&lt;p&gt;I am inside a &lt;p&gt; paragraph&lt;/p&gt;"
```

In addition, you can use the following method calls to create HTML:

"CSS classes
html_writer->add_class( `myCssClass` ).

"CSS styles
html_writer->add_style( name = `background-color` ,
                       value = `red` ).

"Attributes
html_writer->add_attribute( name = `src` ,
                           value = `anyImage` ).

"An empty element
html_writer->empty_element( `br` ).
```

### 3.2.1.2 Secure encoding of HTML on the client

You also have the possibility to encode HTML on the client. To do this, you can use the following JavaScript code:

```javascript
function encode(api, string) {
  return api.encodeHTML(string);
}
```

This JavaScript code contains a reference to the callback API which you can pass from the backend via a JavaScript parameter. To do this, you call method ADD_CALLBACK_API() of interface IF_WD_HTML_SCRIPT_CALL in your Web Dynpro code.

In addition to HTML, you can create secure encoding of XML, JavaScript, URL or CSS using the following functions in JavaScript:

- `encodeXML()`
- `encodeJS()`
- `encodeURL()`
- `encodeCSS()`
3.3    Naming conventions and namespaces
When you implement HTMLIslands or HTMLContainers there are certain naming conventions you need to consider. SAP note 1742528 includes a list of reserved prefixes and names of identifiers.

3.3.1    Naming conventions for custom JavaScript
For custom JavaScript you should reduce the number of global variables to a minimum. The best practice is to use a separate namespace, thus only one global variable. Such a namespace can, for instance, be defined as follows:

    var MyApp = MyApp || {};

After this definition, you can add variables and functions to the namespace MyApp. For example:

    MyApp.myVariable = 5;
    MyApp.myObject = {
        my_member: "I’m a member variable",
        myObjectFunction: function () {
            ...
        }
    };
    MyApp.myFunction = function () {
        ...
    };

This naming convention keeps the global namespace clean and prevents collision with other JavaScript libraries or code.

3.3.2    Naming conventions for SAP internal development
Global JavaScript variables should be built as follows:

    sap.<componentName>.<libraryName>.<variableName>

For instance, you can use the following naming for the component HCM and the library BusinessCards:

    sap.hcm.businessCards.createBusinessCard

Note that the creation of variables inside the sap namespace needs to be tolerant. For instance:

    var sap = sap || {};
    sap.hcm = sap.hcm || {};

Never use the following namings as existing objects would be destroyed:

    sap = {};
    sap.hcm = {};

CSS class names should be built as follows:

    sap<CmpShortCut><LibraryShortCut><ClassName>

The shortcuts, however, need to be aligned with and approved by the chief architect of the component. For instance, you can use the following naming for the component HCM and the library BusinessCards:
3.4 Restrictions
You must run Web Dynpro applications that use elements to integrate custom HTML in the STANDARDS mode of the browser you are using. This can be achieved by setting application parameter WDPREFERREDRENDERING to the value STANDARDS. Note that STANDARDS mode is supported only by Internet Explorer version 9 and higher versions, as well as by Firefox, Safari, and Chrome. Older Internet Explorer versions are rendered always in QUIRKS mode. For further information refer to SAP note 1753544.

A further restriction applies to HTMLIslands only: they must not be used in containers with scrollingMode not equal to none.