

Customizing the Web Client

The Servoy Web Client has a pure HTML & CSS based UI that runs in the Browser, while all the business logic is executed on the Application Server.

With the UI running in the browser and being pure HTML and CSS, it lends itself for customization.

Customization of the Web Client is possible in a few different ways:

- Modifying the Form HTML & CSS templates
- Changing the `servoy_web_client_default.css` template
- Interacting with the browser environment through non-editable HTMLAreas

This chapter explains the above mentioned techniques.

In This Chapter

- [Customizing the Form HTML & CSS Templates](#)
 - [Viewing the Templates](#)
 - [Modifying the Templates](#)
 - [Storing Modified Templates](#)
 - [Creating Multiple Sets of Customizations](#)
 - [The Pitfall of Modifying HTML Templates](#)
- [Changing the servoy_web_client_default.css Template](#)
- [Interacting with the Browser Environment through Non-editable HTMLAreas](#)
 - [Example 1](#)
 - [Example 2](#)
 - [Example 3](#)

Customizing the Form HTML & CSS Templates

For each form that is part of the designtime design of a solution, Servoy generates both an HTML and CSS template dynamically. These templates are filled with data at runtime.

The generated templates can be customized and stored in a specific location on the Servoy Application Server, using a name identical to the dynamic generated template. If a template is available on disk, that template will be used at runtime, instead of the dynamically generated template.

Viewing the Templates

The dynamically generated templates are available through the browser at `{serverURL}/servoy-webclient/templates/default/{solutionName}/{formName}.[html/css]` and viewing the source of the page. The templates can also be accessed via WEBDAV.

Modifying the Templates

The source of the template can be modified with any preferred tool. When modifying the templates, care must be taken not to alter any of the identifiers used by Servoy to fill the template at runtime with data.

In case of a CSS template, these will be all the ID's used on the StyleClasses, for example the ID 'form_frm_company' in the sample below:

```
#form_frm_company {
    border-style: none;
    min-width: 800px;
    min-height: 600px;
    position: absolute;
    top: 0px;
    left: 0px;
    right: 0px;
    bottom: 0px
}
```

In case of editing HTML templates, these are the 'id' and 'servoy:id' attributes on the nodes in the sample below:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<!-- Servoy webclient page Copyright 2011 Servoy -->
<html xmlns:servoy>
  <head>
    <title>dialog - Servoy</title>
    <servoy:head>
    </servoy:head>
  </head>
  <body id='servoy_page'>
    <form id='servoy_dataform'>
      <servoy:panel>
        <div servoy:id='servoywebform' id='form_dialog'>
          <div id='sfw_form_dialog' style='position: absolute; height: 0px; right: 0px; left: 0px;'/>
          <div id='sfh_form_dialog' style='position: absolute; bottom: 0px; top: 0px; width: 0px;'/>
          <div servoy:id='View'>
            <div servoy:id='sv_5636ffed_4465_43f0_97bb_23a7d800942f' id='sv_5636ffed_4465_43f0_97bb_23a7d800942f'
class='formpart'>
              <div servoy:id='sv_da70a6fe_39c8_445b_88bc_7aca5c1bb0b6' class='tabpanel' >
                <div servoy:id='webform' style='overflow: auto;position: relative' class='webform' ></div>
              </div>
              <div style="white-space: nowrap;" servoy:id='sv_1e460f73_5cac_4396_815a_c09af002918e' class='label' ><
            /div>
            <div style="white-space: nowrap;" servoy:id='sv_5eae2200_ae8d_4100_844b_4d825775f1d3' class='label' ><
          /div>
          <div servoy:id='sv_1399876f_ed4f_410e_a7e7_91839bd7390c_wrapper'
id='sv_1399876f_ed4f_410e_a7e7_91839bd7390c_wrapper' >
            <button type='submit' servoy:id='sv_1399876f_ed4f_410e_a7e7_91839bd7390c' class='button' ></button>
          </div>
          <div servoy:id='sv_8fb398c4_064f_4222_82b9_74f4fdc8410e_wrapper'
id='sv_8fb398c4_064f_4222_82b9_74f4fdc8410e_wrapper' >
            <button type='submit' servoy:id='sv_8fb398c4_064f_4222_82b9_74f4fdc8410e' class='button' ></button>
          </div>
          <div servoy:id='sv_5bdbe19a_0e8b_407d_855e_28c97dcade9e_wrapper'
id='sv_5bdbe19a_0e8b_407d_855e_28c97dcade9e_wrapper' >
            <button type='submit' servoy:id='sv_5bdbe19a_0e8b_407d_855e_28c97dcade9e' class='button' ></button>
          </div>
        </div>
      </div>
    </servoy:panel>
  </form>
</body>
</html>

```

Any additions that are to end up in the Header section of the generated HTML Markup of the Web Client can be placed within the `<servoy:head></servoy:head>` tags.

Inside the HTML Templates Servoy uses long UUID's as ID's, for example 'sv_5636ffed_4465_43f0_97bb_23a7d800942f', sometimes postfixed with 'wrapper': these ID's match the UUID of the objects as available through the SolutionModel's API. At runtime however, these ID's are replaced with dynamically generated short ID's.

 The ID of the HTML node representing a form or element in the Web Client at runtime can be retrieved at runtime through the WebClientUtils plugin.

Storing Modified Templates

When modified, the templates need to be stored in the following location to be picked up by Servoy at runtime:

```
{servoyInstall}/application_server/server/webapps/ROOT/servoy-webclient/templates/{subdir}/{solutionName}/
{formName}. [html/css].
```

Creating Multiple Sets of Customizations

The `subdir` part of the location is the 'default' directory by default, but it is possible to create another subdirectory in the `{servoyInstall}/application_server/server/webapps/ROOT/servoy-webclient/templates/` directory for storing the modified templates. At runtime, in the solution, tell Servoy to use that directory instead of the default directory using the following code:

```
application.setUIProperty(APP_WEB_PROPERTY.WEBCCLIENT_TEMPLATES_DIR, 'customDirectoryName');
```

This mechanism allows for creating multiple variations of the templates, for example one variation per customer in a SaaS environment.

The Pitfall of Modifying HTML Templates

Modified templates become out of sync with the design of the form as soon as the form is altered in Servoy Developer. When this happens, the modified template needs to be removed and the then dynamically generated template needs to be modified and stored in the correct location again.

Changing the `servoy_web_client_default.css` Template

Besides the form HTML and CSS, there is also the `servoy_web_client_default.css` template. This template contains the default CSS used for styling the forms and elements in a Web Client. Modifying this template will result in changing the overall look of all solution in the Web Client.

This CSS can be found at `{serverURL}/servoy-webclient/templates/default/` and be changed via webdav.

Example

As an example to center all webclient solutions add:

```
#forms
{
  left: 50%;
  width: 940px;
  margin-left: -480px;
  top: 0px;
  bottom: 0px;
  position: absolute;
}
```

For an extensive example customization of the `servoy_web_client_default.css` template see the [Alternative CSS for Web Client](#) project on [ServoyForge](#).

 For branding some of the generic web pages that are part of the Servoy environment and are utilized by the Web Client, see [Branding](#).

Interacting with the Browser Environment through Non-editable HTMLAreas

One of the standard UI widgets in Servoy is an HTMLArea. This widget provides one of the ways to interact with the browser environment, as it has some special behavior when set to not editable and used in the Web Client.

Foremost, the non-editable HTMLArea is meant to render the HTML string contained in the dataprovider attached to the HTMLArea.

The special behavior comes in when the HTML string contained in the dataprovider contains specific values for certain attributes on specific HTML tags:

- Servoy media URL's (`media:///...`) inside Style tags or attributes are rewritten so they can be resolved in the browser
- Servoy media URL's used in the 'src' attribute of Style, Script and IMG tags are rewritten so they can be resolved in the browser
- Script, Style and Link tags automatically end up in the head section of the DOM in the browser
- Event handlers that in the HTML start with 'javascript:', followed by a Servoy method identifier are rewritten to perform a callback to the server
- An onload event handler on the body of the HTML is executed in the browser when the DOM is ready

 The HTML string in the dataprovider attached to the non-editable HTMLArea needs to be valid XHTML, as it needs to be parsed and processed in order to perform the above mentioned rewrites.

Example 1

The following example renders the latest 100 tweets from the servoy twitter account in the non-editable HTMLArea.

The code snippet shows a variable called `html` being filled with a custom HTML string. The technique used here to create the HTML string is using an XML object, on which `.toXMLString()` is called to convert it into a String.

As the code of the `initTweets` function contains invalid characters for XML, the code is wrapped in a CDATA tag. The CDATA opening and closing tags are to be removed from the actual HTML string that is put in the `html` variable for Servoy to be able to correctly parse and inject the HTML string into the Web Client's HTML markup.

The tweet stream is initialized through the calling of the `initTweets()` function through the onload event on the body.

The required libraries for this example are stored in the Media Library of Servoy.

⚠ While storing the libraries in the Media Library is convenient for deployment of the libraries with a solution, the contents of the Media Library is part of the solution design and thus will be loaded into memory in each client. In case of the Smart Client this means that the libraries will also be downloaded, while they have no use in the Smart Client.

```
var html = (<html>
<head>
<script src="media:///jquery-1.6.2.min.js" type="text/javascript"></script>
<!-- JQuery Tweet plugin: http://tweet.seaofclouds.com/-->
<script src="media:///jquery.tweet.js" type="text/javascript"></script>
<style src="media:///jquery.tweet.css" media="all" rel="stylesheet" type="text/css"/>
<script type='text/javascript'>
<![CDATA[
function initTweets() {
    $(".tweet").tweet({
        username: "servoy",
        avatar_size: 32,
        count: 100,
        loading_text: "Getting you the latests tweets right now...",
        refresh_interval: 60,
        template: '{avatar}{text}<span>{time} - {retweet_action} - {reply_action} - {favorite_action}</span>'
    }).bind("empty", function() { $(this).append("No matching tweets found"); });
}
}]>
</script>
</head>
<body onload="initTweets()">
<div class="tweet" style="width: 100%; height: 100%"></div>
</body>
</html>).toXMLString().replace('<![CDATA[', '');
```

⚠ While this example will work on a simple form, there is a risk of it breaking. This example includes JQuery. Servoy itself also utilizes JQuery in certain scenario's. The logic of Servoy is as such that it will only conditionally include JQuery conditionally when required. When that happens when this html is also being shown, there can be a conflict between the 2 JQuery inclusions. This issue can be easily solved by excluding the JQuery library in the html of this example and forcing Servoy to include the JQuery library it ships using the WebClientUtils plugin.

Example 2

This example shown the creation of a callback to the Web Client business logic that runs on the Server.

Inside the body a button is placed with a onclick event handler that is prefixed with 'javascript:', followed by a global method called myCoordinatesCallbackMethod that should be defined in the solution.

The call to globals.myCoordinatesCallbackMethod() in the onclick handler is setup to send 3 arguments into the callback: the first two are local, browser-side variables x and y, while the third is a hardcoded false value.

The 'browser:' prefix on the argument indicates that the value of a browser-side variable should be included in the callback.

The doCallback() function is a little helper function that can be called by browser-side logic to programmatically perform the click on the button.

Through the style attribute on the button, the button is made invisible to the user, while remaining programmatically clickable.

```
var html = (<html>
<head>
<script>
<![CDATA[
var x = 10;
var y = 10;
function doCallback() {
    document.getElementById("myCoordinatesCallbackHandler").click()
}
}]>
</script>
</head>
<body>
<button id="myCoordinatesCallbackHandler" onclick="javascript:globals.myCoordinatesCallbackMethod(browser:x, browser:y, false)" style="visibility: hidden">&nbsp;</button>
</body>
</html>).toXMLString().replace('<![CDATA[', '');
```

⚠ When using the 'browser:' syntax to send back String values, the parameter needs to be double-quoted. Make sure to use single quotes to surround the overall event handler code:

```
onclick='javascript:globals.myZipCodeCallbackMethod(\"browser:zipcode\");'
```

Example 3

Servoy hosts a Google Maps integration sample solution. For more information on this example see [Google Maps](#).