

Working with design agency

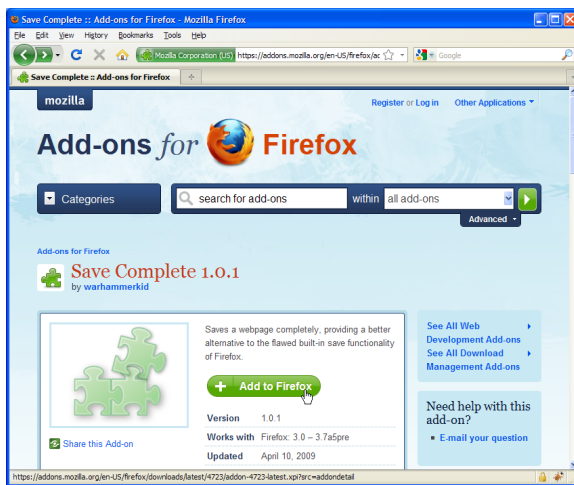
In most web application projects there is a design agency involved mostly designing the user interfaces within the corporate design of the company. xUML UI applications come with predefined themes which in some cases might not fit into the companies design concept e.g. integrate the application into the existing corporate portal site.

Due to the fact that the user interfaces are modeled within MagicDraw and not part of any design agencies delivery package, the agency can fully concentrate on the styling and not on the content of the screens. The content of the screens is modeled according to the business needs.

To be able for the design agency to develop their styles, they need a rough idea and a working file to design on.

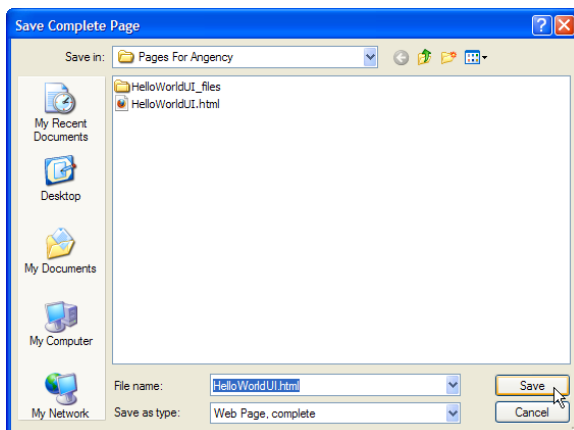
Preparing the files for the Agency to work on

To be able to save a working version of an xUML UI application a Firefox Add-On is needed. This is due to Firefox not being able to export CSS content which are referenced or imported. To install the Add-On follow this link <https://addons.mozilla.org/en-US/firefox/addon/4723/>. The Add-On **Save Complete** (Version 1.0.1) installs automatically when clicked on the green Add to Firefox button.



After the installation Firefox will restart and will offer a new File menu entry **Save Complete Page As....** Load the page that the agency needs to work with into Firefox and use the new menu entry to save the complete page including all referenced files.

Figure: Save Complete Page As...



The result is the main HTML file and a folder containing all the CSS and JavaScript files. Both, the HTML file and the folder can now be sent to a design agency and they will be able to work on the look and feel.

Working with a design tool

On this Page:

- [Preparing the files for the Agency to work on](#)
- [Working with a design tool](#)
- [Replacing/Re-importing changed files](#)
 - [Overwriting / Extending CSS](#)
 - [Assigning CSS Classes to UI Elements](#)
- [Importing Custom Resources](#)
 - [Importing a Custom Theme Roller Package](#)
 - [Importing a custom JavaScript or CSS](#)
 - [Deploy UI Resources](#)
- [CSS and JavaScript as Global Resources](#)
 - [Deploying Global Resources](#)
- [Debugging CSS](#)

Most design tools offer an import mechanism for existing pages or whole sites. This can also be done with the package that is saved via Firefox for external design implementation. Important though is, that the CSS files are amended **without** changing the class names.

Replacing/Re-importing changed files

To close the cycle, the changed design, coming from an external party, needs to be re-imported into the Builder Project. To be able to understand what happens to the design that gets re-imported it is essential to know about how styles are handled during runtime within the browser.

Overwriting / Extending CSS

Overwriting or extending a CSS class is a mechanism which is used to lay over the amended design over the predefined one. When the xUML UI application is loaded, the external references to style sheets and JavaScript resources are loaded. The default style sheet initially defined is loaded first due to its style sheet reference within the HTML file is in top order. The imported style sheets follow in lower order and are loaded after the default is loaded. Now, if the class name `.menu` of the customized style is loaded second, it will overwrite or extend the CSS class attributes.

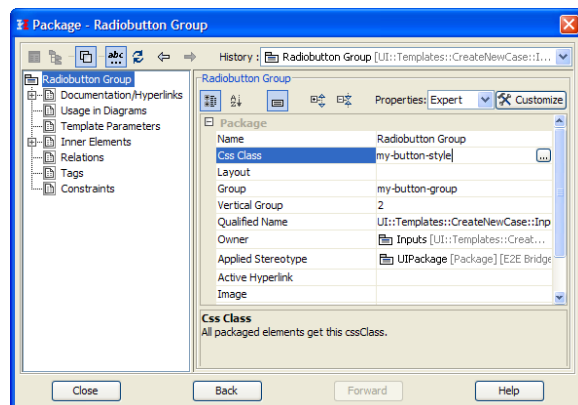
Loading Order	Style Sheet	CSS class definitions
1	default.css	<pre> .menu { font-family:Arial; font-size:12px; color: blue; background: white; margin-top: 5px; margin-bottom:12px; } .menuitem { font-family:Arial; font-size:10px; color: blue; background: white; margin-top: 5px; margin-bottom:12px; } </pre>
2	customized.css	<pre> .menu { font-family: Helvetica; font-size:11px; color: orange; background: #222222; margin-top: 6px; margin-bottom:13px; } .menuitem { font-family: Arial; font-size: 10px; color: blue; background: white; margin-top: 5px; margin-bottom: 12px; text-indent: 20px; } </pre>

The above table shows the loading order and the effect of the overwriting (`.menu` class) and extending (`.menuitem` class) classes. The class attributes in bold are changed by the **customized.css** class definitions.

Assigning CSS Classes to UI Elements

To have more control on UI elements, it is possible to assign a custom **CSS Class** on the `<<UIPackage>`, `<<UIContainer>` or `<<HTMLElement>`. This is helpful if there for example needs to be special design implemented. This css class definition is implemented in a custom css file which needs to be imported into the project to take effect.

Figure: CSS Class Assignment



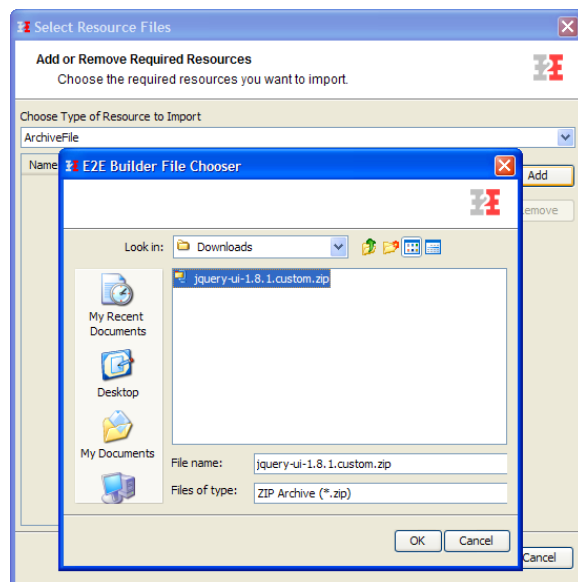
Importing Custom Resources

Despite if it is a single new css file or a customized theme package the customized styles are imported into the project using the Resource Importer.

Importing a Custom Theme Roller Package

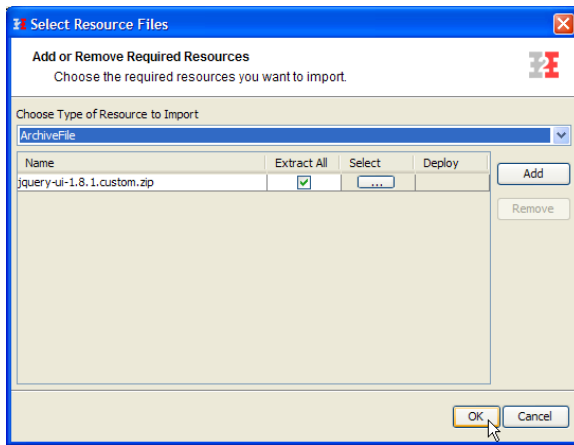
As described previously the changed CSS attributes will overwrite and/or extend the built in jQuery Theme CSS definitions. When using the online Theme Roller, the CSS class names will match the names of the jQuery default definitions.

Figure: Importing Custom Theme Package



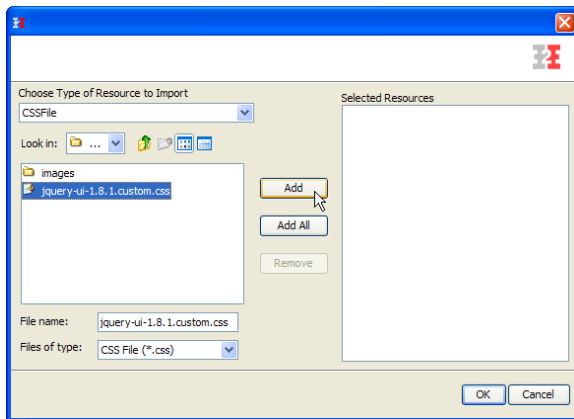
A custom theme package is an archive containing not only CSS definitions but also image and script resources which are needed for e.g. displaying the different widgets. All these files need to be imported with the setting **Extract All** set to true. When deployed, the archive will be extracted and available to the user interface application.

Figure: Theme Roller Resource Import Settings



When importing ThemeRoller themes, it is essential that the main CSS theme file is added as a single resource. Only this way the resource can be identified and added to the Component Diagram. Selecting a main CSS resource is not only important when importing ThemeRoller archives, it is important to do so with all design archives which have one main style sheet referencing other style sheets or images.

Figure: Selecting Theme Roller CSS File

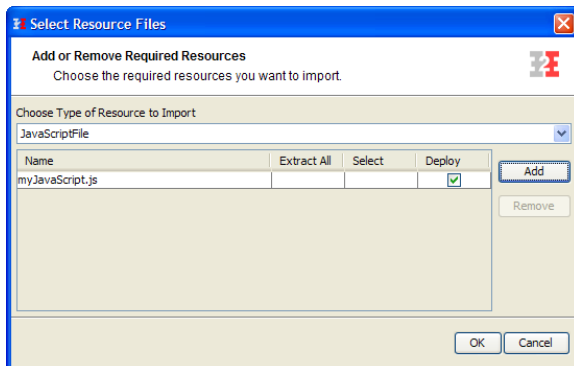


It is essential when choosing the jQuery UI main CSS library to have the resource type set to **CSSFile** in the **Choose Type of Resource to import** drop down list. This rule is valid for any CSS file which should be imported and used within the application.

Importing a custom JavaScript or CSS

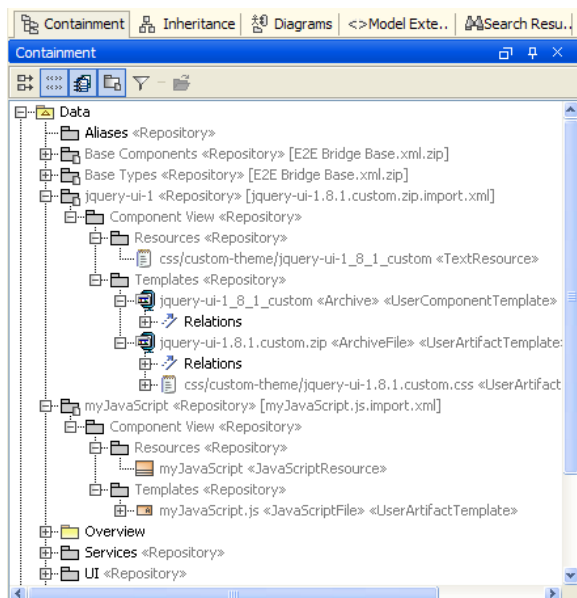
Importing custom JavaScript files as well as custom CSS files follows the same process. Important is, that the Type of resource is set in the import wizard. Only this way it will be handled properly.

Figure: Custom JavaScript Import Settings



As with all imported resources, they are made available in the Magic Draw Containment tree. The resources themselves are not needed in any model or diagram. They are added to the **UIRepository** in the Component Diagram.

Figure: Imported Resources



Deploy UI Resources

The imported resources which are UI relevant (CSS, HTML, JavaScript) are added to the **UIRepositoryArtifact**. Although they are available to be added to the xUML service components, they belong to the **UIRepository**.

Figure: Adding UI Repository

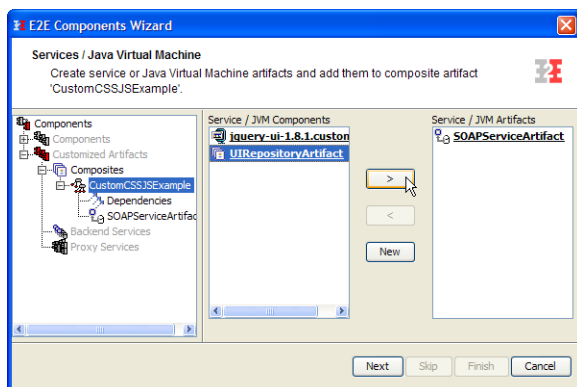
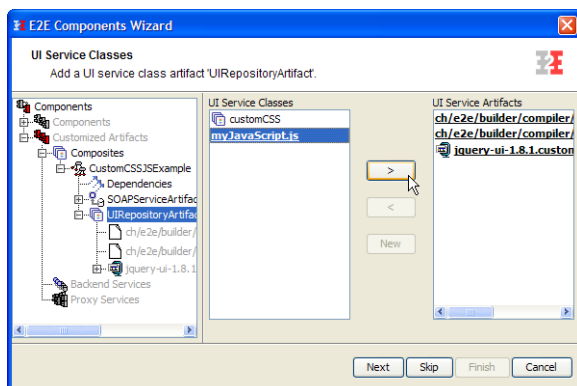
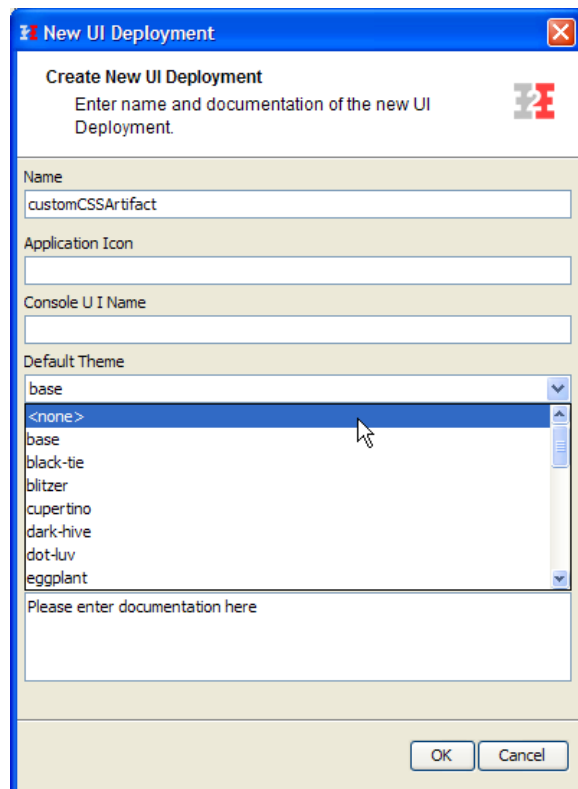


Figure: Adding Custom JavaScript and Theme to UI Repository



When importing a jQuery UI Theme, despite of if it was created using the online ThemeRoller tool or if it was developed manually, it is important to set the **Default Theme** to **<none>**. In general the imported theme's CSS classes will overwrite the built-in ones but it would take longer for the Browser to download the CSS files.

Figure: UI Deployment Settings



After deploying the xUML service and opening the user interface, the theme will be changed from the built in one to the custom designed one.

Figure: xUML UI without custom CSS

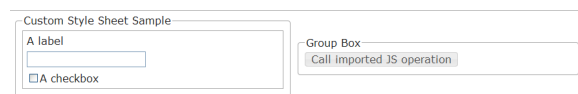


Figure: xUML UI with custom CSS

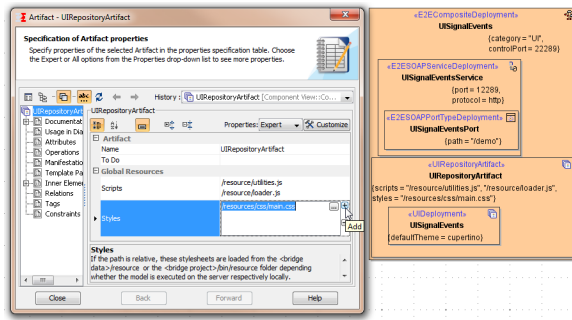


CSS and JavaScript as Global Resources

This feature needs a Bridge installation to work or manually copying of the resource files into the **bin/resource** directory of the Builder Project.

Next to bundling CSS style sheets and JavaScript files with the xUML service it is also possible to separate them from the service and make them be part of the xUML Runtime's global resources. This allows even more freedom for designers and developers to work independently on a project. Further a big advantage to keeping resources globally accessible is the fact that all xUML services can share them which of course keeps maintenance efforts low and changes to the design extremely fast (e.g. change or corporate design).

To make this possible, the relative references to the CSS and JavaScript files need to be set within the component diagrams <<UIRepositoryArtefact>>. Within the Global Resources section of the objects specifications there are two tagged values, Styles and Scripts. Both of these tagged values hold a list of paths to the resources relative location.



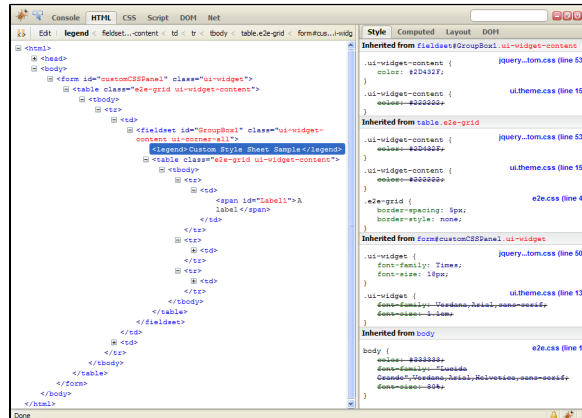
How to upload global resources is described in the E2E Bridge User Guide. Next to upload single files, whole zip and jar archives can be uploaded and at the same time be extracted.

Deploying Global Resources

The global resources are deployed either as single files or as a directory structure by upload and extracting a zip or jar archive. To deploy the resources, access the E2E Bridge via it's administration interface and upload the resources as described in [Deploying and Managing Resources](#).

Debugging CSS

As with JavaScript it is possible to debug CSS styles. The tool recommended is Firebug which is also used to debug xUML user interfaces. Especially when overwriting style sheet classes it is essential to be able to follow the loading sequence and which class is overwritten and which CSS class attributes are involved.



The CSS debugger is opened by clicking on the HTML tab in the debug menu. The debugger is a combination of the page structure with all its HTML elements, which is shown on the left, and on the right side the CSS classes which are defined on the corresponding element. The CSS classes are organized in the loading sequence starting from the bottom. Each sequence step belongs to a CSS definition which resides in a style sheet file e.g. [e2e.css](#).

Inherited from <code>form#customCSSPanel.ui-widget</code>	
<pre> .ui-widget { font-family: Times; font-size: 18px; } .ui-widget { font-family: Verdana,Arial,sans-serif; font-size: 1.1em; } </pre>	<pre> jquery...tom.css (line 50) .ui-widget { font-family: Times; font-size: 18px; } ui.theme.css (line 13) .ui-widget { font-family: Verdana,Arial,sans-serif; font-size: 1.1em; } </pre>

In this case the CSS class `.ui-widget` from the original built-in style sheet theme definition will be overwritten by the imported jQuery UI theme. In detail this would be the `font-family` and the `font-size` class attributes. The attributes which will be overwritten are stroked through.