Styling Solutions

Servoy offers a layered mechanism of styling the UI of solutions, with the options slightly differing for Smart Clients and Web Clients.

Smart Client: Java Swing Look And Feel > Look And Feel Theme (optional, depending on the used Look And Feel) > Styling within the Solution

Web Client: Default StyleSheet > Web Client Template modification > Styling within the Solution

This chapter descibes the various level of Styling options for a solution in Servoy. First the smart and Web client specific Styling options will be discussed, after which Styling options within the Solution are discussed, that are applicable to both Clients.

In This Chapter

- Smart Client specific styling options
  - About Java Swing Look and Feels
    - Themes
    - Adding additional Look and Feels
    - Creating custom Look and Feels
    - Specifying the default Look And Feel for Smart Clients
    - Manually changing the Look And Feel in a Smart Client
- Web Client specific styling options
  - Web Client Default StyleSheet
  - Web Client Template modification
- Solution Styling options
  - Designtime styling options
    - StyleSheets
      - Switching StyleSheets
    - Designtime properties
  - Runtime dynamic styling options
    - Updating UI component properties through their scripting API
    - Conditional styling using the onRender event

Smart Client specific styling options

Servoy Smart Clients are Java Swing applications and Java Swing has the concepts of Look And Feels (LAFs) that can be applied on all Swing components.

Servoy supports using Java Swing Look and Feels out of the box.

About Java Swing Look and Feels

Java Swing Look and Feels come in two variaties:

- Cross platform Look And Feels
- Platform specific Look And Feels

Cross platform Look And Feels aim to provide the exact same UI regardless the platform, for example Windows, OSX or Linux, on which the LAF is used. Platform specific Look And Feels will only work on the platform they are designed for.

Java on each platform ships with at least a platform specific LAF, that is the default LAF used by the Servoy Smart Client. Java also ships 2 cross platform LAFS, Metal and Nimbus, of which Nimbus is the more modern LAF.

Themes

Some LAFs support multiple themes.

Adding additional Look and Feels

Besides the LAFs that are shipped with Java itself, additional third party LAFs can be added to Servoy, by placing the LAF library in the *{servoyInstall} /application_server/lafs* directory.

⚠ **A word of caution on third party Look And Feels**

There are many third party Look And Feels around for Java, both free/open source LAFs and commercial LAFs. The quality of these LAFs differs, so when opting to use a third party LAF, make sure to properly test the LAF in combination with the developed Solution in Servoy. A public listing of the majority of the available third party LAFs can be found on http://www.javootoo.com/

Creating custom Look and Feels

Creating custom Look and Feels requires detailed Java knowledge and is in general a lot of work.

Servoy partners with Centigrade, a company that provides User Interface Design services and products, one of the products being a customizable Look and Feel that is fully compatible with Servoy. For more information, see the Centigrade website

Specifying the default Look And Feel for Smart Clients

The Smart Client Settings on the Servoy Application Server allow the configuration of the default LAF for all Smart Clients, through the 'selectedlnf' property. This property needs to be set to the className of the Look And Feel class. When the classNAme is unknown, it can be retrieved by calling the following function in a Smart Client, after manually selecting the LAF in the Smart Client Preferences:

```
var laf = Packages.javax.swing.UIManager.getLookAndFeel()
application.output(laf.getClass().getName())
```

Not every LAF is supported on every platform. Notoriously the OSX platform is limited in the supported LAFs. It is therefor possible to exclude the Smart clients started on OSX from getting the supplied LAF, using the 'pushLnfToMac' setting.

If the LAF supports themes, the default theme to use can be specified through the 'lnf.theme' setting.

Manually changing the Look And Feel in a Smart Client

Through the Preference panel in the Smart Client, users can select the Look and Feel of their liking. The Preference panel is accessible through Edit > Preferences > Look and Feel.

Access to the Preference panel by the user can be removed using the window plugin , that allows to remove the Preference menuitem in the Edit menu

Web Client specific styling options

Web Client Default StyleSheet

The default styling of Forms and Element in the Web Client is determined by a default stylesheets that can be customized. See Customizing the Web Client for more info.

Web Client Template modification

Each Form created in Servoy Developer results in a customizable HTML and CSS template which are at runtime utilized to create the HTML markup of the Web Client. See Customizing the Web Client for more info on template customization.

Solution Styling options

Within a Solution, regardless of whether the Solution will run in a Smart or Web Client, Servoy provides 2 layers of designtime styling options and 2 layers of runtime dynamic styling options

Designtime styling options

The two levels of designtime styling options are StyleSheets and designtime properties of all the UI components

StyleSheets

Servoy provides the ability to separate the Styling from the Solution's code through CSS StyleSheets. These StyleSheets can be created and managed inside the Servoy Developer IDE (Solution Explorer > Resources > Styles).

A StyleSheet is linked to a Form using the Form's 'styleName' property.

The StyleSheet can define default styling for each UI objects within Servoy (Forms and Elements) and provide any number of additional StyleClasses for each UI object. These additional StyleClasses can be set on individual UI objects through their 'styleClass' property.

In addition to providing styling for specific UI objects, the StyleSheet can also provide odd/even/selected for the rows in the grids of forms in TableView and Portals.

When creating a new StyleSheet in Servoy Developer, the "New StyleSheet wizard" provides a option to insert a StyleSheet sample in the newly created StyleSheet. This sample provides a good overview of the available options.

The CSS Editor in Servoy Developer provides full code completion for the supported CSS properties and their values.

For an overview of the supported CSS properties, see Supported CSS style properties

Switching StyleSheets

At designtime, Forms can be designed to use a specific StyleSheet. In order to facilitate skinning the Solution, for example for different customers, Servoy provides the option to dynamically switch the StyleSheet use at runtime using the following code:

```
application.overrideStyle('baseStyle', 'customerXStyle');
```

When this code gets executed, any Form that was designed to use the StyleSheet 'baseStyle' will instead start using the 'customerXStyle'. Note that this will only take effect on Forms that are loaded after performing the switch.

Designtime properties

While the StyleSheet support provides generic styling options, with the option to differentiate on individual UI objects through additional StyleClasses, all the UI objects also support several styling related properties, like borderType, background anf fontType for example.

As long as these properties are set to default the styling of the element will be according to the applicable StyleSheet and StyleClasses. When specifying a a custom value for the property, it overrides the Styling inherited through the StyleSheet.

Runtime dynamic styling options

At runtime there are 2 levels of dynamically changing styling:

- Updating UI component properties through their scripting API
- Conditional styling using the onRender event

Updating UI component properties through their scripting API

Each UI component provides a scripting API and that API provides methods to alter the appearance of the UI component dynamically at runtime.

For a complete overview of the runtime API of all UI components see RuntimeForm for Form instances and the childnodes of elements for all the different Element types

Conditional styling using the onRender event

The onRender event on Forms and Elements provides a way to conditionally change the appearance of the UI component.

The onRender event is available on Forms, Portals and Elements. This event allows changing display properties of supporting components just before they are shown or updated. As such it can be used for conditional formatting for example.

On Form and Portals the event is fired not only for the Form/Portal itself, but also for all the standard Servoy Elements on the Form/Portal, if the element does not have its own onRender event handler. The Form/Portal level onRender event will not fire for beans.

The onRender event handler is called with a parameter of type JSRenderEvent, that provides the following functions:

- getRecord(): the record of the rendered object
- getRecordIndex(): the index of the rendered object
- getRenderable(): the rendered object
  The returned object is of type Renderable. A Renderable object can be an instance of a RuntimeForm, RuntimePortal or any of the other RuntimeXxxx elements.
  The Renderable class exposes all the properties that can be set in the onRender event and also utility functions to get the rendering element type and its dataprovider.
  The Renderable class is a generic class and some of the properties and methods are not applicable on the actual object being rendered. For example, if the object being rendered is an instance of RuntimeForm, the property toolTipText or the method getDataProviderID are irrelevant. When these are set/called anyway, they will fail silently
- hasFocus(): whether or not the rendered object has the focus
- isRecordSelected(): whether or not the record of the rendering object is selected

Any updates made in the onRender event to the rendering object are persistent within the Client session until changed through the runtime API of the element or in a consecutive onRender event. This means that in the onRender logic, both states of a property need to be handled. This means that if the onRender is used to set the fgcolor of a field depending if the dataprovider's value is negative or not, the fgcolor needs to be explicitly set for both negative and positive numbers. When the same foreground property is also set in scripting and should overrule the onRender, the developer needs to take care of this inside the onRender logic

Example use: Making negative values in a column red and zero values orange

```
/*
 * Called before the form component is rendered.
 *
 * @param {JSRenderEvent} event the render event
 */
function onRender(event) {
/** @type {JSRecord<db:/udm/orders>}*/
var rec = event.getRecord()
if (rec.amt_tax == 0) {
    event.getRenderable().fgcolor = '#ff6600';
 } else if (rec.amt_tax < 0) {
    event.getRenderable().fgcolor = '#ff0000';
 } else {
    event.getRenderable().fgcolor = '#000000';
 }
}
```

⚠️ About performance: The onRender event will be fired often. It's therefor advised to keep the logic inside the onRender event handler method as optimized as possible, for quick execution. It's advised to refrain from calling any code outside the API of the JSRenderEvent or the Renderable class.