Eclipse Scout

Migration Guide

Scout Team

Version 6.1

Table of Contents

WebContent Resources

The *.html files (index.html, login.html, logout.html etc.) have been changed. The simplest way to migrate these files is to create them anew using the Scout SDK or maven archetype and compare them with your files. For details see chapter Content Security Policy (CSP).

Text Provider Service

The method `AbstractDynamicNlsTextProviderService#getDynamicNlsBaseName` has been made public. Adjust the method in your text provider service accordingly.

Content Security Policy (CSP)

TODO IMO: Describe CSP, its implications on Scout applications and how to configure it.

By default, inline `<script>` tags in HTML files are prohibited by CSP rules. Bootstrapping JavaScript code was therefore moved to dedicated **.js* files in the `WebContent/res` folder. Existing projects using CSP have to manually perform the following steps:

1. Open each **.html* file in `your.project.ui.html/src/main/resources/WebContent` folder and check if there are any inline script parts. Only `<script>` tags with embedded JavaScript code are considered "inline". Tags with a `src` attribute don't need to be changed.
2. Transfer the content of each script part to a **.js* file in the `res` subdirectory (e.g. *index.html* ⇒ *res/index.js*) and delete the now empty <script> part.
3. Add a reference to the **.js* file in the `<head>` section using the `<scout:script>` tag, e.g.: `<scout:script src="res/index.js" />`
4. If the extracted **.js* file contains `<scout:message>` tags, they have to be moved back to the `<body>` of the corresponding **.html* file (because the NLS translation can only process HTML files). The attribute `style` has to be changed from `javascript` to `tag`.

Example:

Listing 1. login.html before migration (Scout 6.0)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Contacts Application</title>
    <scout:include template="head.html" />
    <scout:stylesheet src="res/scout-login-module.css" />
    <scout:script src="res/jquery-all-macro.js" />
    <scout:script src="res/scout-login-module.js" />
    <script> ①
      $(document).ready(function() {
        scout.login.init({texts: <scout:message style="javascript" key="ui.Login" key
="ui.LoginFailed" key="ui.User" key="ui.Password" /> });
      });
    </script>
  </head>
  <body>
    <scout:include template="no-script.html" />
  </body>
</html>
```

① Prohibited inline script.

Listing 2. login.html after migration (Scout 6.1)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Contacts Application</title>
    <scout:include template="head.html" />
    <scout:stylesheet src="res/scout-login-module.css" />
    <scout:script src="res/jquery-all-macro.js" />
    <scout:script src="res/scout-login-module.js" />
    <scout:script src="res/login.js" /> ①
  </head>
  <body>
    <scout:include template="no-script.html" />
    <scout:message style="tag" key="ui.Login" key="ui.LoginFailed" key="ui.User" key=
"ui.Password" /> ②
  </body>
</html>
```

① External script reference allowed by CSP.

② Moved from JavaScript call to `<body>`, changed style to `tag`.

Listing 3. res/login.js after migration (Scout 6.1)

```
$(document).ready(function() {
  scout.login.init(); ①
});
```

① Translated texts are extracted automatically from DOM.

Customizing CSP directives

The method `org.eclipse.scout.rt.server.commons.servlet.HttpServletControl.getCspDirectives()` is no longer available. CSP directives are now configured by the the bean `org.eclipse.scout.rt.server.commons.servlet.ContentSecurityPolicy`. To customize the rules, replace this bean with your own implementation and override the method `initDirectives()`. The bean provides fluent-style with_XXX_() methods.

By default, the `report-uri` for CSP violations is now called **/csp-report** (instead of /csp.cgi).

scout.graphics.prefSize() [JS]

The signature of JavaScript method `scout.graphics.prefSize()` has changed:

- **Old**: scout.graphics.prefSize($elem, includeMargin, options)
- **New**: scout.graphics.prefSize($elem, options)

The argument *includeMargin* was moved to the options object. See code documentation for a description of all options.

scout.ModelAdapter [JS]

If you have not created any custom widgets, you can skip this. If you only used `BeanFields` for customizing you can skip it as well.

Previously every widget with a corresponding part on the server extended `scout.ModelAdapter`. A model adapter is the connector with the server, it takes the events sent from the server and calls the corresponding methods on the widget. It also sends events to the server whenever an action happens on the widget. To make the widgets usable without a server, they don't extend from `scout.ModelAdapter` anymore but directly from `scout.Widget`. That means every widget with a server counter part have been separated into widget and model adapter, similarly to the server side where a `IJsonAdapter` exists for every model object. The model adapter creates the widget and attaches itself to it meaning it listens for events triggered by the widget and sends elected ones to the server. It also takes the events from the server and calls the corresponding methods of the widget.

So if you created custom widgets you have to separate them as well. Create for each widget a separate file called the same way as the Widget + 'Adapter'. That adapter extends either directly from `scout.ModelAdapter` or from the corresponding adapter of the parent widget.

Example: You have created a `XyField.js` which extends from `FormField.js`. Now create a file called `XyFieldAdapter.js` and extend it from `FormFieldAdapter.js`.

You now have to move the server event handling methods to the adapter, if there are any at all. If your widget does not contain a method called `onModelAction`, you are fine. Beside these action events the server may send property change events as well. For every property change event the adapter will automatically call the corresponding setter method. If there is none it will call the generic method `Widget.setProperty` which eventually calls the `_sync` and `_render` methods of the property. So if your widget contains `_sync` methods they will still be called on a server property change like before. But now you should create a JS property event to inform other widgets by using `Widget._setProperty` (note the _). This was previously done automatically for every property which is still done if there is no `_sync` method. If there is one you have to take care of it by yourself.

For the opposite direction meaning events from UI to server you have to more or less replace the calls of `_send()` with `trigger()`. In the adapter you have to handle these widget events and call the `_send()` method accordingly. If it is a property change event it is even simpler. Just call `_addRemoteProperties` in the constructor of the model adapter for every property which should be sent to the server.

_send()

The signature of JavaScript method `scout.ModelAdapter._send()` has changed:

- **Old**: scout.ModelAdapter._send(type, data, delay, coalesceFunc, noBusyIndicator)
- **New**: scout.ModelAdapter._send(type, data, options)

Instead of passing individual arguments, pass all but the first two arguments in an options object: * `delay` * `coalesce` * `showBusyIndicator`

Old:

```
this._send('selected', eventData, null, function() { ... });
```

New:

```
this._send('selected', eventData, {
  coalesce: function() { ... }
});
```

scout.Widget [JS]

If you have not created any custom widgets, you can skip this.

destroy()

With the separation of widget and model adapter the destroy handling has been refactored. This means every widget may now be destroyed. Previously only the widgets which extended from `scout.ModelAdapter` could be destroyed. The big advantage is that every widget now behaves the same and that there finally is a counter part for the `_init()` called `_destroy()` which makes it possible to do cleanup like removing listeners.

For you it means you have to decide whether you want to destroy or only remove your widgets. A widget knows the following states:

1. initialized
2. rendered
3. removed
4. destroyed

You can remove and render the same widget as many times you want, but if you destroy it you may not use it again and you would have to create a new one. It eventually has to be destroyed though for a proper cleanup. Normally this is done by the parent widget, but in some rare cases you have to take care of it by your own.

So check all the occurrences of `YourWidget.remove()` and maybe replace them with destroy.

EventSupport

Every widget now installs the event support by default. Previously `_addEventSupport` had to be called in the constructor of the widget. This may now be removed.

KeyStrokeContext

The method `_addKeyStrokeContextSupport` has been removed. If your widget needs keystroke support override `_createKeyStrokeContext` and provide one. You can probably use the default `scout.KeyStrokeContext`. The parameter of `_initKeyStrokeContext` has been removed as well. Just use `this.keyStrokeContext` instead.

Mnemonics

Mnemonics are not supported anymore. Remove all mnemonics (&) from your text files as they will not be considered anymore! && was used escape the mnemonic behaviour and display a single '&' in a text. Replace && with &!

The following methods were or will be removed:

- `StringUtility.removeMnemonic`
- `StringUtility.getMnemonic`
- `IAction.PROP_TEXT_WITH_MNEMONIC`

- `IAction.PROP_MNEMONIC`
- `IAction.getTextWithMnemonic`
- `IAction.getMnemonic`
- `strings.removeMnemonic`
- `strings.removeAmpersand`

Changes in objectType syntax and scout.create() [JS]

The "objectType" is a string describing which JavaScipt "class" to use when creating an object instance using `scout.create()` (roughly similar to a Java class name). To make the object factory more robust, the separator between the type and the model variant was change from `.` to `:`. The namespace separator remains `.`. This allows the following forms of object types:.

- `"StringField"`: name without namespace, i.e. a type in the default namespace (resolves to `scout.StringField`)
- `"myproject.StringField"`: namespace qualified name
- `"StringField:MyVariant"`: type with variant (resolves to `scout.MyVariantStringField`), can also be combined with a namespace

Migration: Check your *objectFactories.js* and *defaultValues.json* files (if you have any in your project) for types with variant and convert the separator from `.` to `:`.

getFocusOwner

Method getFocusOwner() was removed from IDesktop, IForm and DesktopEvent. Since replacing the old rich client ui technologies (swing, swt) with the modern html ui, this method didn’t work correctly anymore.

There are no plans to implement correctly because of multiple reasons. It would increase network traffic between browser and ui server and also would be quite unreliable. The old behaviour was a synchronious result from the ui (swing, swt), which was feasible in rich client environments. But with a distant browser, a realtime result is hard to achieve and might already by outdated by its arrival at the ui server.

If such functionality is needed, it has to be programmed with java script within the browser.

FinalValue

setIfAbsent has been renamed to setIfAbsentAndGet. setIfAbsent now returns a boolean denoting, if a value was set or not.

@PostConstruct

A method annotated with @PostConstruct in a Bean is now guaranteed to run exactly once. The constructor may still run more than once.

Tree

The method `AbstractTree#execAutoCheckChildNodes` got two new parameters and the default implementation now considers `enabledNodesOnly` and does not always ignore disabled nodes.

Changed behavior of scout.HtmlComponent() constructor function [JS]

The constructor function scout.HtmlComponent() no longer links the $comp to the new instance. Instead, the static function scout.HtmlComponent.install() should be used to create a new HtmlComponent and link it to $comp. The constructor function should never be used anymore in custom code. (If you do, you will get errors.)

The new static method makes it clearer that it will alter the state of $comp. For a normal constructor, such behavior is unexpected and thus discouraged.

Migration: Check all **.js* files in your project for occurences of `new scout.HtlmComponent` and replace them with `scout.HtmlComponent.install`.

```
// Old, do not use anymore!
this.$container = $parent.appendDiv('my-widget');
new scout.HtmlComponent(this.$container, this.session);

// New, change your code to this (no change in first line):
this.$container = $parent.appendDiv('my-widget');
scout.HtmlComponent.install(this.$container, this.session);
```

Do you want to improve this document? Please edit this page on GitHub.