Eclipse Scout
Migration Guide

Scout Team

Version 9.0

Table of Contents

About This Document . . . 1
API Changes (Java) . . . 1
- Disabling Close- & Cancel-Buttons . . . 1
- Add Custom Header for REST Services . . . 1
- Replacements for ServerRunContextFilter . . . 2
- Property Lookup Order Changed . . . 2
- Menu Wrapping . . . 3
- CollatorProvider Behavior Changed . . . 3
- MOM: Changed Logger Name for Incoming/Outgoing JMS Messages . . . 4

API Changes (JavaScript) . . . 4
- Rename of LESS Variables . . . 4
- Default Value for Scout JSON Model Attribute "type" . . . 4
- Page Can Be Declared in the Static Outline JSON Model . . . 5
- Form in FormTableControl is Adapted Automatically . . . 5
- Table Inside PageWithTable Uses Default Reload Handler . . . 5
- Possibility to Prepare a LookupCall in ListBox & RadioButtonGroup . . . 5
- Toggle Action and Button Now Trigger Event . . . 5

Other Changes . . . 6
- Java 11 Support . . . 6

About This Document

This document describes all relevant changes **from Eclipse Scout 8.0 to Eclipse Scout 9.0**. If existing code has to be migrated, instructions are provided here.

Attention: The here described functionality has not yet been released and is part of an upcoming release.

API Changes (Java)

Disabling Close- & Cancel-Buttons

Until Scout 8.0 a Close- or Cancel-Button ignored the enabled-granted property to ensure the form can be closed even if the full form has been disabled using `setEnabledGranted(false)`. This was confusing because the same convenience was not available for all other enabled dimensions.

Since Scout 9.0 Close- and Cancel-Buttons can be disabled like any other form field. But one special handling is still present: The method `isEnabledIncludingParents` ignores the enabled state of the parents and always returns the state of the button only.

So if a Form or GroupBox is disabled using `setEnabled(false)` or `setEnabledGranted(false)` or any other dimension, the full form gets disabled except the Close- and Cancel-Buttons. As soon as the button is disabled explicitly (e.g. by calling `setEnabled(false)` on the button itself or by propagating to the button using `setEnabled(false, false, true)` on a parent composite) it will be disabled and the form cannot be closed anymore.

Migration:

If you disable the root group box of a form (with child propagation) and want the close- or cancel button to stay enabled anyway, the button must be re-enabled again afterwards.

If you have Close- or Cancel-Buttons that need to be disabled, it must be disabled explicitly on the button itself or by propagating the change on a parent form field.

Add Custom Header for REST Services

By default REST services exposed by Scout using the `org.eclipse.scout.rt.rest.RestApplication` are protected against CSRF now. This requires a custom header to be present on all non GET or HEAD requests. Please do also ensure that GET or HEAD requests do follow the HTTP specification and therefore do not perform any data or state changing operations!

As long as the ScoutJS App or the `org.eclipse.scout.rt.rest.client.AbstractRestClientHelper` is used to call REST services this header is automatically added. For other 3rd party clients the custom header `X-Requested-With` must be added. There is no value for the header required. Typically `XMLHttpRequest` is passed but the value can be chosen freely.

If there are non CSRF prone clients (like backend systems) such clients may also be excluded from

the CSRF header check by replacing the Scout Bean `AntiCsrfHelper` and ignoring such requests.

See the Scout Bean `org.eclipse.scout.rt.rest.csrf.AntiCsrfHelper` for more details.

Replacements for ServerRunContextFilter

Before Scout 9 the `ServerRunContextFilter` was used to create Scout server contexts for REST APIs. This filter used a user based TTL cache that was not bound to the HTTP session.

Starting with Scout 9 there are two new filters available:

- `HttpRunContextFilter`: Creates a Scout context without HTTP- and server sessions for stateless REST backends. It supports subject, correlationId, locale, transaction, etc.
- `HttpServerRunContextFilter`: Creates a Scout server context that additionally has a user-agent and an optional Scout server session.

Migration:

If a Scout server session is required in the application use

- `ServerRunContextFilter` for clients that have no cookie support
- `HttpServerRunContextFilter` for clients with cookie support

If the application does not have a Scout server session use

- `HttpServerRunContextFilter` with session support disabled if a server dependency is available in the application.
- `HttpRunContextFilter` if only a server-commons dependency is available in the application.

Property Lookup Order Changed

The Scout properties are now resolved in a slightly different order. The environment variables are now resolved *before* the `config.properties` file.

1. System properties
2. Environment variables
3. Config properties file
4. Default value of property

Environment variable typically have stricter naming conventions for their names (POSIX). To allow overriding most Scout properties with environment variables, these lookup rules are applied in-order to find a matching environment variable:

1. An exact match of your property key (`my.property`)
2. A match where periods are replaced by underscores (`my_property`)
3. An uppercase match of your property key (`MY.PROPERTY`)

4. An uppercase match where periods are replaced by underscores (`MY_PROPERTY`)

If there are no environment variables which collide with Scout properties, then no migration is required.

In a deployment where a colliding environment variable exists, the following migration options exist:

- If it is an intentional override: No migration required
- If the override is an unintended collision of the environment
  - Remove the colliding environment variable from the environment of the process your application runs in, or
  - Change the property key, or
  - Force the intended property to the desired value using a Java system property (`-D` JVM parameter).

Menu Wrapping

The `OutlineMenuWrapper` has been adapted so that no manual instance can be created anymore (the constructors are no longer public). Instead the new factory bean `org.eclipse.scout.rt.client.ui.desktop.outline.MenuWrapper` should be used. This helps to ensure that the correct wrapper is created for each menu type and because it is a bean it may be extended for custom menu types. The same is for the methods `unwrapMenu` and `containsWrappedMenu` which have been moved to the new `MenuWrapper` class as well.

CollatorProvider Behavior Changed

The `CollatorProvider` bean now uses the *NaturalCollatorProvider* by default (see release notes). This may result in different results when sorting text.

Migration:

If your application already used a custom `CollatorProvider` bean, this may not be necessary anymore. Check the implementation.

If your application whishes to restore the previous behavior, the `CollatorProvider` has to be replaced like this:

```
@Replace
public class MyCollatorProvider extends CollatorProvider {

  @Override
  public Collator getInstance(Locale locale) {
    return Collator.getInstance(locale); // use JVM default
  }
}
```

MOM: Changed Logger Name for Incoming/Outgoing JMS Messages

The code for logging incoming/outgoing JMS messages has been extracted into a dedicated class. This results in a changed logger name (the log level is still `DEBUG`).

Migration:

If your application configures a custom log level for `org.eclipse.scout.rt.mom.jms.JmsMomImplementor`, change it to `org.eclipse.scout.rt.mom.jms.LogJmsMessageHandler`.

API Changes (JavaScript)

Rename of LESS Variables

If you created a custom theme, you might have to adjust some LESS variables.

- Renamed `@scrollbar-thumb-color` to `@scrollbar-thumb-main-color`
- Renamed `@scrollbar-thumb-inverted-color` to `@scrollbar-thumb-inverted-main-color`
- Renamed `@calendar-mode-active-text-color` to `@calendar-mode-selected-color`
- Renamed `@planner-mode-active-text-color` to `@planner-mode-selected-color`
- Renamed `@popup-border-color` to `@control-popup-border-color`
- Renamed `@view-tab-active-color` to `@simple-tab-active-color`
- Renamed `@view-tab-active-background-color` to `@simple-tab-active-background-color`
- Renamed `@view-tab-background-color` to `@simple-tab-background-color`
- Renamed `@table-header-menu-cmd-active-background-color` to `@table-header-menu-cmd-selected-background-color`
- Renamed `@table-header-menu-cmd-active-border-color` to `@table-header-menu-cmd-selected-border-color`
- Renamed `@table-header-menu-cmd-active-color` to `@table-header-menu-cmd-selected-color`
- Renamed `@table-control-active-color` to `@table-control-selected-color`
- Renamed `@table-control-active-background-color` to `@table-control-selected-background-color`
- Renamed `@scrollbar-side-margin` to `@scrollbar-side-padding`

Default Value for Scout JSON Model Attribute "type"

In the Scout JSON model the `"type": "model"` must no longer be specified as it is the default value now. Therefore all occurrences can be deleted. This makes the static JSON model more compact, easier to write and better readable.

Page Can Be Declared in the Static Outline JSON Model

Before 9.0 it was necessary to add pages (`PageWithTable`, `PageWithNodes`) programmatically to outlines as pages could not be declared in the JSON model. This is now possible. The Outline JSON model may now contain an attribute `nodes` which may contain custom pages. Refer to the helloworld application (created by Eclipse) or the JS Widgets application for an example.

Form in FormTableControl is Adapted Automatically

A form in a `FormTableControl` is automatically adapted to match the semantics in which the form exists:

- It is automatically set to non-modal
- It does not ask if a save is needed as it will never be saved anyway.
- It is configured to DisplayHint.VIEW

Until now if the `FormTableControl` was used in a project, this adaption of the form must have been done manually. This code can now be removed as it will be done by Scout now.

Table Inside PageWithTable Uses Default Reload Handler

A table within a `PageWithTable` has a default reload handler installed now. It calls `loadTableData` on the page which discards all rows and loads them again by calling `_loadTableData`. If a custom reload handler was installed, it may be removed now.

Furthermore the `_loadTableData` method now also gets an optional argument holding the exported data of the first form that is attached to the table using a `FormTableControl` (typically the SearchForm).

Possibility to Prepare a LookupCall in ListBox & RadioButtonGroup

Until now it was necessary to set the lookup call programmatically to a ListBox or LookupCall if a `prepareLookupCall` listener was registered. Otherwise the call has been executed before the listener could have been attached. This is no longer the case: The LookupCall is executed on first use only and not during the creation of the widget which allows to attach `prepareLookupCall` listeners. The LookupCall may now also be declared in the static JSON model of the widget even though a `prepareLookupCall` listener is registered.

Toggle Action and Button Now Trigger Event

The action resp. click event is currently not fired if the action/button is a toggle action or the button has menus. This means, if you want to be informed when a user clicks a regular button, you would listen for the action event. If you want to be informed when a user clicks a toggle button, you would

have to listen for a property change event.

To make it easier, the action event is now always fired (in addition to the property change event). This helps, if you just want to know whether the button was clicked and aren't interested in the selected state.

If you accidentally registered an action resp. click listener for toggle actions or buttons, or menu with child menus or buttons with menus, the listener will now be informed. So make sure to check your toggle actions and buttons so that the action is not executed twice.

Note: Due to compatibility reasons the behavior for Scout Classic has not been changed.

Other Changes

Java 11 Support

Scout 9 officially supports OpenJDK 11. Please note that Java 9 and 10 are not supported and that Oracle only provides free Java 8 updates for commercial use until end of January 2019. Therefore it is recommended to use OpenJDK 11.0.1 or newer. The following chapters describe the actions to migrate your code to be Java 11 capable.

General Migration

- Scout increased the minimum Maven version from 3.2.1 to 3.5.3. You might need to update your tool-chain accordingly.
- The JAX-RS API version has been updated from 2.0.1 to 2.1.1. This may be relevant if you use a container that already includes a JAX RS runtime instead of bundling your own runtime.
- Update the `maven_rt_plugin_config-master` in all poms to version 3.1.0
- If you use the Scout JAX-RS support using Jersey you have to add the dependency `org.glassfish.jersey.inject:jersey-hk2` to all poms where the artifact `jersey-container-servlet-core` is referenced as dependency.
- Batik has been updated from 1.7 to 1.10. If you are using Batik in your project you might need to update some imports. E.g.:
  - `org.apache.batik.dom.svg.SAXSVGDocumentFactory` to `org.apache.batik.anim.dom.SAXSVGDocumentFactory`
  - `org.apache.batik.dom.svg.SVGDOMImplementation` to `org.apache.batik.anim.dom.SVGDOMImplementation`
- Because of duplicate classes in `javax.activation:javax.activation-api` and the corresponding implementation `com.sun.activation:javax.activation` it might be necessary to exclude the former dependency where both are present on the classpath. Such cases will be reported by the `duplicate-finder-maven-plugin` during maven build.
- If you are using the `maven-assembly-plugin`: The `descriptor` tag pointing to the xml file has been replaced with a `descriptors` list element. Furthermore the `classifier` configuration element does no longer exist and must be deleted if present. Example: `<descriptor>assembly.xml</descriptor>` can be replaced with

```
    <descriptors>
       <descriptor>assembly.xml</descriptor>
    </descriptors>
```

JAX WS Migration

- The JAX WS API version has been updated from 2.2.10 to 2.3.1. This may be relevant if you use a container that already includes a JAX WS runtime instead of bundling your own runtime.
- The JAX WS RI (reference implementation) has been removed from the JRE. Therefore `org.eclipse.scout.rt.server.jaxws.implementor.JaxWsRISpecifics` should not longer by used unless the application is still running with Java 8. Otherwise migrate to `org.eclipse.scout.rt.server.jaxws.implementor.JaxWsMetroSpecifics` (which is the new default value).
- Because the RI is no longer part of the JRE, the Metro implementation must be added instead. To do so please add dependency `com.sun.xml.ws:jaxws-rt` to the poms of your server war and dev projects.
- Because the JAX WS classes are no longer part of the JRE, the corresponding libraries must be added to all `.factorypath` files in your workspace. Please add the following elements:

```
    <factorypathentry kind="VARJAR"
       id="M2_REPO/javax/jws/javax.jws-api/1.1/javax.jws-api-1.1.jar"
       enabled="true" runInBatchMode="false"/>
    <factorypathentry kind="VARJAR"
       id="M2_REPO/javax/annotation/javax.annotation-api/1.3.2/javax.annotation-api-
1.3.2.jar"
       enabled="true" runInBatchMode="false"/>
    <factorypathentry kind="VARJAR"
       id="M2_REPO/javax/xml/ws/jaxws-api/2.3.1/jaxws-api-2.3.1.jar"
       enabled="true" runInBatchMode="false"/>
```

- The `jaxws-maven-plugin` from `org.codehaus.mojo` is not Java 11 capable. Therefore replace the groupId with `com.helger.maven` in all your poms to use a Java 11 capable maven plugin.

Changed Computation of NodeIdentifier

If multiple Scout applications are connected together to form a cluster, the application-scoped bean `org.eclipse.scout.rt.platform.context.NodeIdentifier` provides an ID for each node. This string can be used to identify cluster nodes in log messages, cluster messages etc.

It is recommended to assign a unique and stable name to each node by setting the property *scout.nodeId* when launching the application. The default implementation of `NodeIdentifier` also checks for properties of some well-known application servers (e.g. *jboss.node.name*). If no explicitly assigned identifier is found, a random UUID is generated.

Note that a random ID will not be stable, i.e. it will change each time the application is restarted. This is fine for single-node and most multi-node setups. However, there are cases where stable IDs are required, for example when using durable topics or cleaning up node-specific data.

In previous releases, `NodeIdentifier` sometimes returned the local hostname and a port number instead of a random UUID. This behavior was removed from Scout 9, because it is unreliable and the used port number was not always correct. If your application relies on the hostname, consider explicitly setting the *scout.nodeId* property or customizing the bean `NodeIdentifier`.

Do you want to improve this document? Have a look at the sources on GitHub.