Eclipse Scout

Migration Guide

Scout Team

Version 10.0

Table of Contents

About This Document

This document describes all relevant changes **from Eclipse Scout 9.0 to Eclipse Scout 10.0**. If existing code has to be migrated, instructions are provided here.

Service Releases

Scout 10.0 will continue to be maintained for a while and a new build may be released from time to time. Beside bugfixes, these service releases may even contain some minor features.

The following changes were made after the initial 10.0 release.

10.0.10

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

- AjaxCall (since 10.0.10)

New Web Tool Stack

Scout updated the web tool stack from a custom implementation to a standard Node.js, Webpack and Babel based setup.

Thanks to the power of these tools it is now possible to write modern JavaScript code which will be transpiled at build time so that old browsers can execute the code.

To run the JavaScript build a JavaScript runtime is required. This can be downloaded from Node.js. It is recommended to use a Node.js version 12 because this is a LTS release. After downloading run the installer and add the installation directory to the PATH environment variable. This allows to use commands like `node` or `npm` from the command line. These commands will be required later.

Automatic Migration

In order to reduce the time needed for migrating your code to Scout 10 there is a migration application available which will migrate most of the code automatically.

This application migrates a single Maven module. For a standard Scout application it has to be executed for the .ui module. If your project contains more modules containing JavaScript, CSS or HTML code, the migration application has to be executed for each of these (e.g. .app module). To launch the migration application follow these steps:

1. Clone the following git repository and import it into an empty eclipse workspace: https://github.com/BSI-Business-Systems-Integration-AG/scout-10-migration-template. The workspace and git repository are only necessary temporarily and can be deleted again after the migration.

2. The migration app requires at least Java 8. Also make sure the migration project uses the same encoding as your source code files (a regular scout project uses UTF-8 by default). If you use a different encoding, adjust the property in the pom.xml and press Alt+F5 to reimport the changes:

```
<properties>
  <maven.compiler.target>1.8</maven.compiler.target>
  <maven.compiler.source>1.8</maven.compiler.source>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

3. The migration requires api files to properly generate the import statements. The api files for Scout are already placed in the folder `api` of the project. If your JavaScript code depends on other libraries built with Scout and not only org.eclipse.scout.rt.ui.html, then you need to include the api files from these libraries as well.
4. Adjust the methods in the `Config` class to configure the migration application to your local setup. Make sure to read the `JavaDoc` to understand what exactly you need to configure.
5. The migration application has some prerequisites that the code can be migrated. Ensure that these are met before launching:
   a. The source must be properly formatted.
   b. One file must only contain one JavaScript class.
   c. All files should be committed to VCS (not really a precondition but makes it easier to restart again if the migration fails).
6. If the above conditions are fulfilled the migration can be started using the launch config `migrate.launch`.
7. The migration is executed in two steps (which will result in two separate VCS commits):
   a. First: all existing files are overwritten with the migrated content.
   b. Second: the files are moved to theirs new locations.
8. Having two steps is necessary to preserve the VCS history. When the first step is finished, the migration app asks you to commit the changes and confirm. Do NOT commit and confirm yet.
9. Have a look at the output. If there is an error you should revise your configuration (maybe you didn't copy all api files?) and your code (did you extract all JS classes into a separate file?, etc.). To fix the errors abort the migration, reset all files (e.g. use git reset --hard), fix the configuration and run the migration again.
10. If there are no errors anymore have a look at the the migrated files. You will notice that the files are not properly formatted yet. You can do it now, if you like to have a clean git history, or do it later after the migration.
11. If you are pleased with the files commit them. Make sure all files are committed.
12. Now it is time to proceed with the second step. Confirm the message.
13. The migrated files are now moved to theirs new location. When it's done, commit all the files.

After the migration execution there are some manual post processing tasks that must be executed,

see next chapter.

Build Stack Migration

Dependencies to other JS modules are defined in a package.json file and can be installed using NPM. You will notice that sometimes the tool PNPM will be mentioned. This is a different package manager also used by Scout itself to build the Scout modules. This results in slightly different tasks. If you don't know what to use yet we suggest starting with npm, you can always upgrade to pnpm (or maybe even Yarn) later.

Depending on the type of application, you either have a *.ui.html module or a *.ui and an *.app module. The ui module typically contains the JS code and the app module the static resources (WebContent folder). If you only have one module it contains both. So if the guide tells you to do something in the .app module and you don't have one, do it in the .ui.html module instead.

1. Please do the Automatic Migration first even if you don't have any JS or CSS files in your project because it will also migrate your HTML files (e.g. index.html etc.).
2. In the pom.xml of your parent project change the version of the parent maven_rt_plugin_config-master to 3.8.0. Also update your Scout dependencies to the most recent Scout 10 version.
3. If you use GIT, create a file (or open the existing one) named `.gitignore` in the .ui module and the .app module (if available) and add the following content:

```
node_modules
dist
test-results
```

4. If you only have a .ui module without an .app module (typical Scout Classic app), follow these steps. Otherwise skip this section and continue with the next step.
   a. Copy the following file to the root of your .ui.html module (replace the placeholder ${simpleArtifactName} with your application name and ${version} with your application version): package.json
   b. Copy the following file to the root of the .ui.hml module: webpack.config.js
5. If you have a separate .app module you should have skipped the previous step. Instead do the following steps.
   a. Copy the following file to the root of the .ui module (replace the placeholders ${simpleArtifactName} with your application name and ${version} with your application version): package.json
   b. Copy the following file to the root of the .ui module: webpack.config.js
   c. Copy the following file to the root of the .app module (replace the placeholders ${simpleArtifactName} with your application name, ${version} with your application version and ${rootArtifactId} with the root module name): package.json

If you use pnpm, referencing other npm modules in the same workspace is easier. In that case you should replace "file:../${rootArtifactId}.ui/" with the version of your .ui module (e.g. 1.0.0-snapshot).

d. Copy the following file to the root of the .app module (replace the placeholder ${simpleArtifactName} with your application name): webpack.config.js. If you have JavaScript tests, copy the following file to the root of the .ui module: karma.conf.js

6. Copy the following file to the directory src/test/js of the .ui module: test-index.js
7. The migrator should have renamed `yourapp-module.js` to `index.js`, `yourapp-module.less` to `index.less` and migrated theirs contents. Check these files for TODOS and resolve them, maybe you need to adjust an import. Also, if you have multiple themes, there should be an additional index file for each theme (e.g. index-dark.less). Check these files too.
8. The migrator should also have renamed `yourapp-macro.js` to `yourapp.js`, `yourapp-macro.less` to `yourapp-theme.less` and moved them to the src folder. The former index.js is now included into yourapp.js. There should be a theme file for each theme (e.g. yourapp-theme-dark.less). If not, create them manually. Make sure that the theme files properly import the according index files (e.g. yourapp-theme-dark.less should include index-dark.less files if available).
9. Open your app file (the one that contains something like `new App().init`, e.g. src/main/js/yourapp.js) and remove the modelsUrl, if present.
10. Check your html files. The migrator should have removed obsolete style and script tags and added references to `yourapp.js` and `yourapp-theme.js`. If you had multiple custom `*-macro.js` or `*-macro.less files` there may still be references to an old `*-macro.less` file → Use the new matching `*-theme.css` file instead or maybe merge it with the regular `yourapp-theme.css` file.
11. If you have other html files than the migrated index.html, login.html etc. migrate them manually in the same way index.html has been migrated (adjust script, style and include tags).
12. Move all files and folders in src/main/resources/WebContent/res one folder up directly into WebContent and delete the empty res folder.
13. Search for all occurrences of `/res` within the .ui and .app modules and remove the res folder.
14. In the file src/main/webapp/WEB-INF/web.xml files of the .ui.html.app.dev and .ui.html.app.war modules change the `filter-exclude` list of the `AuthFilter` declaration to the following and replace ${simpleArtifactName} with your application name (if your application uses the Scout LoginApp). The `AuthFilter` may also be called `UiServletFilter` or `YourAppNameUiServletFilter`.

```
/favicon/*
/fonts/*
/logo.png
/jquery*.js
/login*.js
/logout*.js
/${simpleArtifactName}-theme*.css
/eclipse-scout*.js
```

15. If you have a Repository.js change the global object holding the repositories from `${yourAppNamespace}.repositories = {};` to `static repositories = {};` and change all references

in this file from `${yourAppNamespace}.repositories` to `Repository.repositories`.

16. If you have Jasmine specs, follow these steps. Otherwise you can skip it and continue with the next step.
    a. Remove any Jasmine server test launch configurations (**jasmine**.launch files).
    b. Remove any Spec runner HTML files (Spec*Runner*.html files).
    c. Remove all entries of the Maven plugins `jasmine-maven-plugin` and `phantomjs-maven-plugin` from the pom.xml files.
    d. In all pom.xml files remove the entries of the Maven plugin `maven-dependency-plugin` that runs in phase `generate-test-sources` and unpacks files from `org.eclipse.scout.rt.ui.html` or `org.eclipse.scout.rt.ui.html.test`.
    e. Move all Specs from src/test/js/${yourAppNamespace} to src/test/js (one folder up)
17. If your .ui and .app modules are separated, in the pom.xml of the .ui module add the .following properties:
    a. `master_skip_npm_build_dev=true`
    b. `master_skip_npm_build_prod=true`
    c. `master_skip_copy_webpack_build_output=true`
18. Remove the following properties from all your config.properties files: `scout.ui.prebuild`, `scout.ui.prebuild.files`, `scout.dev.scriptfile.rebuild`, `scout.dev.scriptfile.persist.key`.
19. If you use pnpm, create a file called `pnpm-workspace.yaml` in the parent folder of your modules (which is most likely the root of your git repository) and include your npm modules (.ui, .app).
20. Open a terminal in the folder of the .ui module and run the command `npm install` (or `pnpm install`, if you use pnpm). This installs all dependencies that are required by the .ui module.
21. Open a terminal in the folder of the .app module and run the command `npm install` (or `pnpm install`, if you use pnpm). This installs all dependencies that are required by the .app module (including the .ui module of your project).
22. In the terminal of the .app module run the following command: `npm run build:dev`. This triggers the transpiler that creates the JavaScript build output in the dist folder of the .app module. Only after this command has been executed the server can find the web resources to deliver them to the browser.
23. If there are any build errors, fix them manually. The migration application might not fix every possible code correctly.
24. In the terminal of the .ui module run the following command: `npm run testserver:start`. This executes the Jasmine Specs in a Chrome browser (Chrome must be installed locally, the same applies to ChromeHeadless if running the build in a CI environment, see Build / Continuous Integration).
25. Now it is time to start your app!
    Since you already ran `npm run build:dev` all the required JS and CSS files should be created and you can start the UI server using your existing launch file. If you now use IntelliJ you need to create a run configuration that uses `JettyServer` as main class (have a look at your Eclipse launch file for details).
26. If you like, you can create a launch group that start the JS build and the UI server together. Have

a look at the Scout archetype for an example (the new Scout Project wizard in Eclipse uses the archetype).

27. Test all your html files (index.html, login.html, logout.html etc.) and all your themes (dark, custom). If everything looks fine, you are done. If not, have a look at the Troubleshooting section.

Instead of `npm run build:dev` you can also use `npm run build:dev:watch` which will watch your JS and CSS files. This means you do not need to restart the task when you change JS or CSS files. This does not include the webpack.config.js, changing that file requires running the npm task again.

Troubleshooting

1. After starting the server the page in the browser stays blank.
   This most likely happens when your created bundles don't match the bundles required by the HTML files. Check the Network tab in your browser's DevTools. Which files are being loaded? Are there 404 requests? Check your dist folder. What files are there? To fix it, adjust your HTML files or the entrypoints and chunks in `webpack.config.js`.
2. I am not sure if I missed something.
   You can always create a new Scout app based on the arche type and compare it with your code. The easiest way is to use Eclipse and create a new Scout Project. Make sure you choose the correct application type (Java for Scout Classic apps, JavaScript for Scout JS apps).
3. I don't understand why I had to do this and what it is for.
   Have a look at the Build Stack chapter in the Technical Guide.
4. I did everything you said but it still does not work.
   Ask at Stack Overflow or the Forum, we are happy to help you out.

Build / Continuous Integration

With the new web tool stack the build of the JS and CSS code does not happen at runtime anymore but during build time. In order to make your life easier most of the npm tasks are automatically started by maven when running mvn install. But there are still a few adjustments you need to make on your CI jobs in order to build your application.

1. Node is installed automatically during maven build when the module contains a package.json. So there is no need to install node on the build server. Nothing to do here for you.
2. To run JavaScript specs Scout now uses ChromeHeadless instead of PhantomJS. If you have JS specs you need to make sure there is a Chrome installed on your build server. The following installation guide worked for our linux servers: https://gist.github.com/ipepe/94389528e2263486e53645fa0e65578b#gistcomment-2379515.
3. In order to display the test results you need to add the new test-results dir in your job configuration (e.g. `*/test-results/*/test-*.xml`)
4. You only have to do this step if you want to share your npm packages between different applications.

If you want to deploy npm artifacts to a custom npm repository (e.g. Artifactory), you need to add `.npmrc` file to the home directory of your build user on the build server (similar to the `.settings.xml` of maven). In order to deploy the artifacts you can use the official npm cli interface (npm publish). If you want to publish snapshots you can use the following command.

```
cd your.app.ui
./target/node/node ./target/node/node_modules/npm/bin/npm-cli.js publish
--tag=snapshot
```

API Changes (Java)

StrictSimpleDateFormat

`org.eclipse.scout.rt.jackson.dataobject.StrictSimpleDateFormat` was removed. Use `org.eclipse.scout.rt.platform.util.date.StrictSimpleDateFormat` instead.

ObjectUtility

`nvlOptional()` was renamed to `nvlOpt()`.

Data Objects

The Scout data object support was moved from the Scout platform to the module `org.eclipse.scout.rt.dataobject`. The package imports of all data object related classes therefore changed: From `org.eclipse.scout.rt.platform.dataobject` to `org.eclipse.scout.rt.dataobject`

IDoEntity raw list access

The following methods to access a raw list attribute where changed semantically. The getter now always return a non-null value. If the attribute is not available, an empty list is added as attribute value into the entity and the new created empty list is returned.

- `List<Object> getList(String attributeName)`
- `<T> List<T> getList(String attributeName, Class<T> type)`
- `<T> List<T> getList(String attributeName, Function<Object, T> mapper)`

Additionally two new methods were added, which allows to access an optionally available list attribute, without adding a new attribute value to the entity, if the attribute is not available:

- `Optional<List<Object>> optList(String attributeName)`
- `<T> Optional<List<T>> optList(String attributeName, Class<T> type)`

Renamings

`org.eclipse.scout.rt.client.ui.desktop.datachange.DoChangeEvent` → `org.eclipse.scout.rt.client.ui.desktop.datachange.ItemDataChangeEvent`

Dependencies

All modules which use data objects were extended with a dependency to `org.eclipse.scout.rt.dataobject`

- `org.eclipse.scout.rt.rest`
- `org.eclipse.scout.rt.mom.api`

Renamings in ErrorDo

- `org.eclipse.scout.rt.rest.error.ErrorDo#status` → `org.eclipse.scout.rt.rest.error.ErrorDo#httpStatus`
- `org.eclipse.scout.rt.rest.error.ErrorDo#code` → `org.eclipse.scout.rt.rest.error.ErrorDo#errorCode`

CacheBuilder

The following methods on `CacheBuilder` where removed, since they were unused and covered unused, old functionality:

- Method `org.eclipse.scout.rt.shared.cache.CacheBuilder.addCacheInstance(ICache<K, V>)`
- Method `org.eclipse.scout.rt.shared.cache.CacheBuilder.getCacheInstances()`

Move ICache and transactional Map

`AbstractTransactionalMap` and its concrete implementations `ConcurrentTransactionalMap` and `CopyOnWriteTransactionalMap` have been moved to `org.eclipse.scout.rt.platform.util.collection`.

`ICache`, its implementations and cache wrappers have been moved to `org.eclipse.scout.rt.platform.cache`.

Remove SessionStore Properties

The following properties are no longer used and can be deleted without replacement: * `scout.ui.sessionStore.maxWaitForAllShutdown` * `scout.ui.sessionStore.valueUnboundMaxWaitForWriteLock` * `scout.ui.sessionstore.housekeepingMaxWaitForShutdown`

Authorization API

The authorization API of scout was extended and moved from `org.eclipse.scout.rt.shared` into its own module. You may check the technical guide for further details.

- Introduced `IPermissionCollection` and `IPermission` interfaces
- Let all current scout permission (e.g. `CopyToClipboardPermission`) implement `IPermission`
- All scout permission names are now prefixed with `scout.`
- `RemoteServiceAccessPermission#getName` returns a stable name instead of the service operation

pattern

- Deleted `BasicHierarchyPermission`. If required, you may copy from an older version of scout.
- `org.eclipse.scout.rt.shared.services.common.security.IAccessControlService` moved to `org.eclipse.scout.rt.security`
- `IAccessControlService#getPermissionLevel` removed; use instead `ACCESS#getGrantedPermissionLevel`
- `IAccessControlService#checkPermission` removed; use instead `ACCESS#check`
- `IAccessControlService#getPermissions` must now **never** return `null`. Instead `NonePermissionCollection` or `AllPermissionCollection` may be returned.
- `org.eclipse.scout.rt.shared.services.common.security.ACCESS` moved to `org.eclipse.scout.rt.security.ACCESS`
- `ACCESS#check` now fails if argument is `null` (before succeeds).
- `org.eclipse.scout.rt.shared.services.common.security.AbstractAccessControlService` moved to `org.eclipse.scout.rt.security`
- `AbstractAccessControlService#getUserIdOfCurrentUser` moved to `Sessions#getCurrentUserId()`

Load Permissions

With the new `IPermissionCollection`, loading of permissions in `AbstractAccessControlService#execLoadPermissions` has changed.

- Create a new instance by calling `BEANS.get(DefaultPermissionCollection.class)` instead of `new java.security.Permissions()`.
- Add permissions with a permission level: `permissions.add(new ReadUsersPermission(), PermissionLevel.ALL);`
- Do not forget to set permission collection as read only: `permissions.setReadOnly();`

There is also a `AllPermissionCollection` which may be used instead of `DefaultPermissionCollection`.

TestingUtility → BeanTestingHelper

The following methods are deprecated. Use the corresponding methods on `BeanTestingHelper` via `BeanTestingHelper.get()` instead:

- `registerBeans`
- `registerBean`
- `unregisterBean`
- `unregisterBeans`
- `mockConfigProperty`

The following replacement regex can be applied on all Java files:
`\bTestingUtility\.(registerBeans|registerBean|unregisterBean|unregisterBeans|mockConfigProperty)` to `BeanTestingHelper.get().$1`

The following methods are deprecated and will be removed in a future release without a replacement:

- `registerWithReplace`
- `registerWithTestingOrder`
- `clearHttpAuthenticationCache`

MailHelper.getCharacterEncodingOfPart(Part)

`MailHelper.getCharacterEncodingOfPart(Part)` is deprecated, use `ObjectUtility.nvl(BEANS.get(MailHelper.class).getPartCharset(part), StandardCharsets.UTF_8).name()` instead if same behavior is required.

API Changes (JavaScript)

WidgetTile

The Widget's main property `tileWidget` is now a fully fledged property. Up until now the property could only be set on initialization and couldn't be changed during runtime. Now the `tileWidget` property can be set dynamically during runtime.

This required a change within the div-structure with which the widget is rendered. The tile and the wrapped widget used to share the same container div. Now they both have their own container div as shown in the example below:

Listing 1. Scout 9.0

```
<div class="form-field tile">
    <div class="field"></div>
</div>
```

Listing 2. Scout 10.0

```
<div class="tile">
    <div class="form-field">
        <div class="field"></div>
    </div>
</div>
```

AjaxCall (since 10.0.10)

Instances of `AjaxCall` now return a single object of type `AjaxError` when rejecting the promise.

For historical reasons, jQuery's AJAX function rejects the promise with three arguments: `jqXHR`, `textStatus` and `errorThrown`. The Promise specification only allows for one "reason" argument. To bring *AjaxCall* in line with this definition and to make rethrowing the error easier, it now wraps the three arguments into one object of type `AjaxError`. It provides the following properties:

`jqXHR`

HTTP request object (jQuery argument #1)

`textStatus`
: A string describing the type of error that occurred (jQuery argument #2)

`errorThrown`
: The textual portion of the HTTP status (jQuery argument #3)

`requestOptions`
: The options object that was passed to the `$.ajax` call. Can be used to retrieve the original URL for debugging or logging purposes.

Migration: Check your code for any uses of `AjaxCall` and change the signature of any *fail()* or *catch()* function.

```
// Old:
var ajaxCall = scout.create('AjaxCall', ajaxOptions);
ajaxCall.call()
  .then(function(data) {
    processData(data);
  })
  .catch(function(jqXHR, textStatus, errorThrown) {
    handleError(jqXHR, textStatus, errorThrown);
  });

// New:
var ajaxCall = scout.create('AjaxCall', ajaxOptions);
ajaxCall.call()
  .then(function(data) {
    processData(data);
  })
  .catch(function(ajaxError) {
    handleError(ajaxError.jqXHR, ajaxError.textStatus, ajaxError.errorThrown);
  });
```

REST Service Changes

Any changes which may change how REST consumer or provider behave.

Renamings in ErrorDo

`org.eclipse.scout.rt.rest.error.ErrorDo` used by `org.eclipse.scout.rt.rest.client.proxy.ErrorDoRestClientExceptionTransformer` and some `org.eclipse.scout.rt.rest.exception.AbstractExceptionMapper<E>` was slightly changed:

- `ErrorDo#status` → `ErrorDo#httpStatus`
- `ErrorDo#code` → `ErrorDo#errorCode`

Different HTTP status codes

A REST service client using `ErrorDoRestClientExceptionTransformer` will now transform

- any client request error (HTTP `4xx` status codes) into a `VetoException`
- `403 - Forbidden` into a `org.eclipse.scout.rt.dataobject.exception.AccessForbiddenException`
- `404 - Not Found` into a `org.eclipse.scout.rt.dataobject.exception.ResourceNotFoundException`

The `org.eclipse.scout.rt.rest.exception.VetoExceptionMapper` used by a REST service provide will now create an error response with status `400 - Bad Request` (this was formerly a `403`).

Cookies disabled by default

By default, a REST service client will no longer use cookies.
If required, cookies can be enabled by using

```
clientBuilder.property(RestClientProperties.ENABLE_COOKIES, true);
```

in `configureClientBuilder` of `AbstractRestClientHelper`.

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