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.

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.

JavaScript Migration

If your application contains JavaScript code it needs to be migrated to EcmaScript 6. If your application does not contain any JavaScript or CSS code you can skip this chapter and continue with Build Stack Migration.

In order to reduce the time needed for migrating all the JavaScript code there is a migration application available which will migrate the code automatically.

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

1. Create an empty Eclipse workspace. The workspace is only necessary temporarily and can be deleted again after the migration.

2. Create a new simple Maven project in your new empty workspace (File → New → Project → Maven → Maven Project → Next, choose `Create a simple project`). Click Next.

3. Choose a Group- and Artifact Id and click Finish. Since the project is only necessary for the migration and can be deleted afterwards, the naming does not matter.

4. Open the created pom.xml and add a dependency to `org.eclipse.scout.rt:org.eclipse.scout.migration.ecma6:10.0.0-SNAPSHOT` and update the project using Alt+F5.

5. The migration app requires Java 8. Add the following properties to the pom.xml and press Alt+F5 again:

```
<properties>
  <maven.compiler.target>1.8</maven.compiler.target>
  <maven.compiler.source>1.8</maven.compiler.source>
</properties>
```

6. Create a new class `Config` in src/main/java extending the class `org.eclipse.scout.migration.ecma6.configuration.Configuration`.

7. Add an `@Replace` annotation to the class

8. In the src/main/resources folder create a folder `META-INF` and within an empty file called `scout.xml`.

9. Open a Browser and download the API definition files from the Scout Git Repository. To do so right click on the small `plain` link for each file and save the target in a local directory (e.g. in a folder called api in your newly created migration app).
   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.

10. Overwrite the following methods in the created `Config` class to configure the migration application to your local setup:

    a. `getConfiguredApiBase`: Point to the local directory where the API files from the previous step are stored.

    b. `getPersistLibraryFileName`: The application will create an API file for your project too (like the ones downloaded at the beginning). This is the place to specify the file name. Choose a name that matches the module name under migration.

    c. `getPersistLibraryName`: Choose the simple name of the module under migration.

    d. `getSourceModuleDirectory`: Absolute path to the Maven module that should be migrated. If you have a Scout Classic application this is the module that ends with .ui.html. If you have a Scout JS application it is typically the module that ends with .ui.

    e. `getConfiguredTargetBase`: Where the migrated files should be stored. If you want to perform an in-place migration to commit the changes to VCS return `getSourceModuleDirectory()`. To test and tweak the migration you can choose an temporary output directory.

    f. `getJsFolderName`: The subfolder within src/main/js in which your JavaScript code is stored.

    g. `getNamespace`: The JavaScript application namespace. This is the prefix all your JavaScript code lives in. If you have code like `helloworld.MyClass.prototype`, then the namespace is `helloworld`.

11. Create a new Java Application launch configuration that uses your newly created project and launches the `org.eclipse.scout.migration.ecma6.Migration` class.

12. 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.

13. If the above conditions are fulfilled the migration can be started using the created launch config.

14. 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.

15. 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.

16. 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.

17. 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.

18. If you are pleased with the files commit them. Make sure all files are committed.

19. Now it is time to proceed with the second step. Confirm the message.

20. 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 if you have JS or CSS files in your project, see JavaScript Migration. If you ran the automatic migration for the module that contains the entry HTML files (e.g. index.html), and you should have done that, then some of the following tasks may have already been done, either completely or partially. However, please read carefully and follow step by step.

2. In the pom.xml of your parent project change the version of the parent maven_rt_plugin_config-

master to 3.7.0.

3. Create a file named `.gitignore` in the .ui module and the .app module (if available) with 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

6. If you have JavaScript tests, copy the following file to the root of the .ui module: karma.conf.js

7. Copy the following file to the directory src/test/js of the .ui module: test-index.js

8. Open your existing index.js file in the .app module. It either exists in the WebContent folder (or the res sub folder) or in the src folder of the module. Replace the content with the following (replace the placeholder ${simpleArtifactName} with your application name):

Listing 1. For Scout Classic

```
import {RemoteApp} from '@eclipse-scout/core';
new RemoteApp().init();
```

Listing 2. For Scout JS

```
import {App} from '@${simpleArtifactName}/ui';
new App().init({
  bootstrap: {
    textsUrl: 'res/texts.json',
    localesUrl: 'res/locales.json'
  }
});
```

> ℹ️ The new files don't require $(document).ready and modelsUrl anymore. Other options are still valid so make sure you don't accidentally remove them.

9. Move this index.js to the src/main/js directory in the same module. If a file with that name already exists, append the content to the existing file and remove the former one.

10. Move the file src/main/resources/WebContent/res/login.js (if it exists) to src/main/js in the same module and replace it with the following content:

```
import {LoginApp} from '@eclipse-scout/core';
new LoginApp().init();
```

11. Move the file src/main/resources/WebContent/res/logout.js (if it exists) to src/main/js in the same module and replace it with the following content:

```
import {LogoutApp} from '@eclipse-scout/core';
new LogoutApp().init();
```

12. Move the file *-macro.less to src/main/js in same module and rename it to theme.less. Make sure the imports point to the new index.less files of the corresponding npm modules (the migrator renamed the former *-module.less to index.less).

```
@import "~@eclipse-scout/core/src/index";
@import "index";
```

13. Create the file src/main/js/theme-dark.less in the .ui module and link all the index files with the suffix -dark. If you have custom dark files you also need to create an index-dark.less linking these files:

```
@import "theme";
@import "~@eclipse-scout/core/src/index-dark";
@import "index-dark"; // Only necessary if you have custom dark files
```

14. If you have more themes, do the same for each theme.

15. Delete the file src/main/resources/WebContent/res/*macro.js in the .app module.

16. Adjust the content of the file src/main/resources/WebContent/index.html in the .app module according to index.html (replace ${displayName} with the title of your application and ${simpleArtifactName} with your application name).

> **ℹ** Differences: the name of the script files are defined in webpack.config.js. The scripts are moved to the end of the <body> tag. The include tags now require the includes folder.

17. Replace the content of file src/main/resources/WebContent/login.html (if available) in the .app module with login.html (replace ${displayName} with the title of your application and ${simpleArtifactName} with your application name)).

18. Replace the content of file src/main/resources/WebContent/logout.html (if available) in the .app module with logout.html (replace ${displayName} with the title of your application and ${simpleArtifactName} with your application name)).

19. In the file src/main/resources/WebContent/popup-window.html in the .app module add `includes/` in front of the template attribute of the <scout:include> tag.

20. If you have other html files, to the same for those as you have done for index.html or login.html (adjust script, style and include tags).

21. Move all files and folders in src/main/resources/WebContent/res one folder up directly into WebContent and delete the empty res folder.

22. Search for all occurrences of `/res` within the .ui and .app modules and remove the res folder.

23. 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
```

24. 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`.

25. 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)

26. If your .ui and .app modules are separated, in the pom.xml of the .ui module add the .following properties:

    a. `master_skip_pnpm_install_dev=true`

    b. `master_skip_pnpm_install_prod=true`

    c. `master_skip_copy_webpack_build_output=true`

27. 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`.

28. 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).

29. 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.

30. 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).

31. 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.

32. If there are any build errors, fix them manually. The migration application might not fix every possible code correctly.

33. 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).

34. 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).

35. 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.conf.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 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`

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 3. Scout 9.0

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

Listing 4. Scout 10.0

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

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.