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, following the Simultaneous Release Cycle. Beside bugfixes, these service releases may even contain some minor features.

The following changes were made after the initial 10.0 release.

Simrel 2020-03 — TBD

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

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 https://nodejs.org/. 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.

Migration for Scout Classic applications

This migration section applies to Scout Classic applications which do not have custom JavaScript code. For all other applications follow the description on how to migrate Scout JS applications in the next section.

1. In the pom.xml of your parent project change the version of the parent maven_rt_plugin_config-master to 3.3.0-SNAPSHOT.
2. Create a file named `.gitignore` in the .ui.html module with the following content:

   ```
   node_modules
   dist
   test-results
   ```

3. Copy the following file to the root of your .ui.html module (replace the placeholder

${rootArtifactId} with your application name and ${version} with your application version): package.json

4. Copy the following file to the root of the .ui.hml module: webpack.config.js
5. Overwrite the file src/main/resources/WebContent/res/index.js with the following content:

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

6. Move this file index.js to src/main/js
7. Overwrite the file src/main/resources/WebContent/res/login.js with the following content:

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

8. Move the login.js to src/main/js
9. Overwrite the file src/main/resources/WebContent/res/logout.js with the following content:

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

10. Move the logout.js to src/main/js
11. Delete the files src/main/resources/WebContent/res/*macro.js
12. Delete the files src/main/resources/WebContent/res/*macro.less
13. Rename the file src/main/js/*-module.less in theme.less and add the following line at the top

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

14. Replace the content of file src/main/resources/WebContent/index.html in the .ui.html module with this (replace ${displayName} with the title of your application): index.html
15. Replace the content of file src/main/resources/WebContent/login.html in the .ui.html module with this (replace ${displayName} with the title of your application): login.html
16. Replace the content of file src/main/resources/WebContent/logout.html in the .ui.html module with this (replace ${displayName} with the title of your application): logout.html
17. Remove property `scout.ui.prebuild` and `scout.ui.prebuild.files` from all config.properties files
18. Open a terminal and navigate to the .ui.html module
19. Run command `npm install`. This will download all JavaScript dependencies
20. When finished run command `npm run build:dev`. This starts the transpile process and creates the output in the dist subfolder. The UI server will then pickup the files from there.

Migration for Scout JS applications

In Scout JS applications the project specific JavaScript code must be migrated to EcmaScript 6. For this a migration application is available. This application migrates a single Maven module. For a standard Scout JS 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. To launch the migration application follow these steps:

1. 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.
2. Create a new simple Maven project in Eclipse (File → New → Project → Maven → Maven Project → Next, choose `Create a simple project`). Click Next.
3. Choose a Group- and Artifact Id and click Finish.
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. Create a new class `Config` in src/main/js extending the class `org.eclipse.scout.migration.ecma6.Configuration`.
6. Add an `@Replace` annotation to the class
7. In the src/main/resources folder create a folder `META-INF` and within a file `scout.xml` (can be empty).
8. Overwrite the following methods in the created `Config` class to configure the migration application to your local setup:
    a. `getLibraryApiDirectory`: Point to the local directory where the API files from the first step are stored.
    b. `getPersistLibraryFileName`: The application will create an API file for your project too. This is the place to specify the file name.
    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. `getTargetModuleDirectory`: 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`.
9. Create a new Java Application launch configuration that uses your newly created project and launches the `org.eclipse.scout.migration.ecma6.Migration` class.
10. 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.

11. If the above conditions are fulfilled the migration can be started using the created launch config.
12. The migration is executed in two steps:

    a. First all existing files are overwritten with the migrated content.

    b. Second the files are moved to their new locations.

13. When the first step finished, the migration app requests confirmation to continue with the second step. This gives you the possibility to commit the first step to VCS which preserves the history.

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

1. In the pom.xml of your parent project change the version of the parent maven_rt_plugin_config-master to 3.3.0-SNAPSHOT.
2. Create a file named `.gitignore` in the .ui.html module and the .app module with the following content:

```
node_modules
dist
test-results
```

3. 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
4. Copy the following file to the root of the .ui module: webpack.config.js
5. 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. 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
8. Copy the following file to the root of the .app module (replace the placeholder ${simpleArtifactName} with your application name): webpack.config.js
9. Create the file src/main/js/index.js in the .app module with the following content (replace the placeholder ${simpleArtifactName} with your application name):

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

10. Create the file src/main/js/theme.less in the .ui module with the following content:

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

11. Create the file src/main/js/theme-dark.less in the .ui module with the following content:

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

12. Delete the files src/main/resources/WebContent/res/**macro.** in the .app module
13. Delete the file src/main/resources/WebContent/res/index.js in the .app module
14. Replace the content of file src/main/resources/WebContent/index.html in the .app module with this (replace ${displayName} with the title of your application and ${simpleArtifactName} with your application name): index.html
15. In the file src/main/resources/WebContent/popup-window.html in the .app module add `includes/` in front of the template attribute of the <scout:include> tag.
16. Move all files and folders in src/main/resources/WebContent/res one folder up directly into WebContent and delete the empty res folder.
17. Search for all occurrences of `res` within the `WebContent` folder and remove the res folder.
18. If you have a Repository.js change the global object holding the repositories from `${yourAppNamespace}.repositories = {};` to `static repositories = {};` and change all references in this file from `${yourAppNamespace}.repositories` to `Repository.repositories`.
19. Remove any Jasmine server test launch configurations (**jasmine**.launch files).
20. Remove any Spec runner HTML files (Spec*Runner*.html files).
21. Remove all entries of the Maven plugins `jasmine-maven-plugin` and `phantomjs-maven-plugin` from the pom.xml files.
22. 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`.
23. 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`

24. Move all Specs from src/test/js/${yourAppNamespace} to src/test/js (one folder up).
25. Open a terminal in the folder of the .ui module and run the command `npm install`. This installs all dependencies that are required by the .ui module.
26. Open a terminal in the folder of the .app module and run the command `npm install`. This installs all dependencies that are required by the .app module (including the .ui module of your project).
27. 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.
28. 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).
29. If there are any build errors, fix them manually. The migration application might not fix any possible code correctly.

The steps above used NPM to install dependencies (`npm install`). Depending on your needs there might be other frameworks that better suit your setup. We recommend having a look at the following alternatives:

1. PNPM
2. Yarn

Please note that Scout uses PNPM internally during the Maven build.

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`.

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)

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`).

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