Eclipse Scout Migration Guide

Version 23.1

Table of Contents

Looking for something else? Visit https://eclipsescout.github.io for all Scout related documentation.

About This Document

This document describes all relevant changes **from Eclipse Scout 22.0 to Eclipse Scout 23.1**. If existing code has to be migrated, instructions are provided here.

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

Obtaining the Latest Version

Scout Runtime for Java

Scout Runtime artifacts for Java are distributed using Maven Central:

- 23.1.0 on *Maven Central*
- 23.1.0 on *mvnrepository.com*

Usage example in the parent POM of your Scout application:

```
<dependency>
    <groupId>org.eclipse.scout.rt</groupId>
    <artifactId>org.eclipse.scout.rt</artifactId>
    <version>23.1.0</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

Scout Runtime for JavaScript

Scout Runtime artifacts for JavaScript are distributed using npm:

- Scout Core Runtime
- All official Scout JavaScript packages

Usage example in your package.json:

```
{
  "name": "my-module",
  "version": "1.0.0",
  "devDependencies": {
    "@eclipse-scout/cli": "23.1.0",
    "@eclipse-scout/releng": "^22.0.0"
  },
  "dependencies": {
    "@eclipse-scout/core": "23.1.0",
    "jquery": "3.6.0"
  }
}
```

The pre-built Scout JavaScript assets are also available using a CDN (e.g. to be directly included in a html document): https://www.jsdelivr.com/package/npm/@eclipse-scout/core?path=dist

IDE Tooling (Scout SDK)

Scout officially supports IntelliJ IDEA and Eclipse for Scout Developers.

IntelliJ IDEA

You can download the Scout plugin for IntelliJ IDEA from the JetBrains Plugin Repository or you can use the plugins client built into IntelliJ IDEA. Please refer to the IntelliJ Help on how to install and manage plugins.

Eclipse

You can download the complete Eclipse IDE with Scout SDK included here:
Eclipse for Scout Developers

To install the Scout SDK into your existing Eclipse IDE, use this P2 update site:
https://download.eclipse.org/scout/releases/12.0/

New 3rd Party requirements

Scout 23.1 requires at least Node 18.12.1. Older versions will not work. A new version can be obtained from the Node download page.

Scout now requires at least pnpm 7.16.0. You can install it as described on the pnpm installation page.

To update your application the following steps might be required:

- Update the version of the `maven_rt_plugin_config-master` in your `pom.xml` files to the newest `23.1.x` release. See Maven central for a list of versions available.
- Update the Scout versions (`package.json` and `pom.xml`) as shown in *Obtaining the Latest Version*.
- If you are using Eclipse and web-service providers, update the `.factorypath` files as shown in the `JAX-WS Appendix` of the Scout documentation.
- The support for the Maven build plugin `org.kuali.maven.plugins:properties-maven-plugin` has been replaced with `org.codehaus.mojo:properties-maven-plugin`. Update the groupId accordingly. Please consult the plugin documentation for the old and the new plugin for instruction on how to migrate the plugin configuration.
- The dependency to `org.jboss:jandex` has been renamed to `io.smallrye:jandex` and the corresponding build plugin from `org.jboss.jandex:jandex-maven-plugin` to `io.smallrye:jandex-maven-plugin`. In case one of these artifacts is used in your code, update the groupId accordingly.

Update of other 3rd Party JavaScript libraries

Perform the following migration in all your `package.json` files:

- If you specify the minimum engines versions, please update them as follows:

```
{
  "engines": {
    "node": ">=18.12.1",
    "npm": ">=9.1.1",
    "pnpm": ">=7.16.0"
  }
}
```

- Update the following dependencies (if existing in your application):

```
"jasmine-core": "4.5.0"
"karma": "6.4.1"
"eslint": "8.27.0"
```

- Remove the following dependencies:

```
"@babel/core"
"@babel/eslint-parser"
"@babel/eslint-plugin"
```

Replace the content of your .eslintrc.js with the following:

```
module.exports = {
  extends: '@eclipse-scout'
};
```

RegisterNamespace

In all your `index.js` files, replace the last line as follows:

Listing 1. From

```
window.yourNamespace = Object.assign(window.yourNamespace || {}, self);
```

Listing 2. To

```
ObjectFactory.get().registerNamespace('yourNamespace', self);
```

See the JsDoc of the `registerNamespace` method if you are curious what it does.

ObjectType as Class Reference (Scout JS)

As described in the {release-notes-link}, it is now possible to use class references for the `objectType`. Even though the string based style still works and a migration is not necessary, the class reference style is the preferred way for the future.

We highly recommend you migrate your code, so you can benefit of the improved type safety and code completion.

To do so, you can use the Scout migration tool, which migrates the strings and also tries to add the required imports automatically. Alternatively, you can do it by yourself using your IDE and Find/Replace.

To use the migration tool, follow these steps:

1. Open the `package.json` of your ui module (e.g. `your.project.ui.html`).
2. Add a devDependency to `@eclipse-scout/migrate` with the version `~23.1.0`.
3. Add a script as described in Listing 3.
4. Adjust the `moduleMap` argument by replacing `yourNamespace` with the namespace in your `index.js` file and `your.project.ui.html` with the name of your module.
   The moduleMap entry is used to add imports to your index.js automatically.
5. If you have dependencies to other Scout based modules, you can add them to the `moduleMap` to have the imports to these modules created automatically. Just add the namespace of the other module and the name of the npm package as follows: `--moduleMap.otherNamespace @other/npm-module`
6. Add a devDependency to `@eclipse-scout/tsconfig`.
7. Create a `tsconfig.json` beside your `package.json` according to the `Readme.md` of the `@eclipse-scout/tsconfig` module.
8. Run the `migrate` script by pressing the play button next to it in IntelliJ or using the command line: `npm run migrate`
9. Review the changes and add the missing imports, if there are any.
10. Remove the script and added dependencies from you `package.json`. Also remove the `tsconfig.json` unless you want to use TypeScript anyway.

Listing 3. Object Type Migration Script

```
{
  "scripts": {
    "migrate": "migrate --migrate objectType --sources src/main/js/**/*.js
--moduleMap.yourNamespace path:your.project.ui.html/src/main/js/index.js"
  },
  "devDependencies": {
    "@eclipse-scout/migrate": "~23.1.0"
  }
}
```

Alternative Way to Migrate ObjectType

If the above script somehow does not work for you, you can do the migration by yourself as follows:

1. Find and replace all occurrences of `scout.create()`, `objectType:`, `lookupCall:` and `logicalGrid:` (use Ctrl-Shift-R in IntelliJ and enable Regex).

   ```
   Find: scout.create\('(.*\.)?(.*)'
   Replace: scout.create\($2
   ```

   ```
   Find: objectType: '(.*\.)?(.*)'
   Replace: objectType: $2
   ```

   ```
   Find: lookupCall: '(.*\.)?(.*)'
   Replace: lookupCall: $2
   ```

   ```
   Find: logicalGrid: '(.*\.)?(.*)'
   Replace: logicalGrid: $2
   ```

   The regex considers objectTypes with and without namespace.

2. Add imports
   Now we need to add the required imports. You can either manually add them, but with a lot of files it is a tedious task. Unfortunately, IntelliJ does not provide a possibility to automatically add all missing imports. But there is a trick: resolve multiple inspections at once using code analysis.

   a. Start the action `Run inspection by name` and select the inspection `Unresolved JavaScript variable`.

   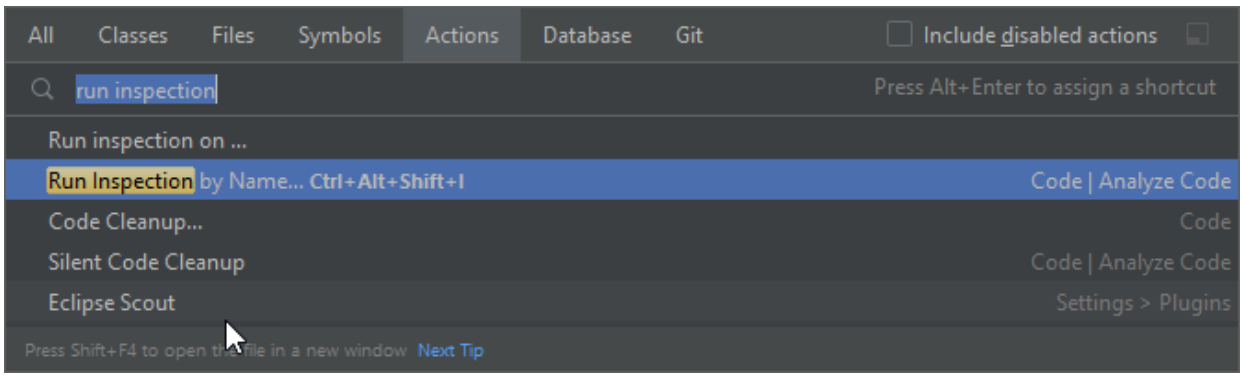

   Figure 1. Start Action

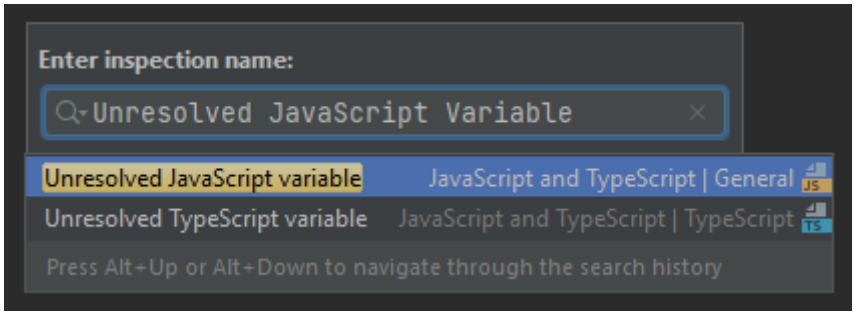

Figure 2. Select inspection

b. Select `Uncommitted files` and disable all inspection options, because the inspection is not limited to missing imports. Click OK to run the analysis.

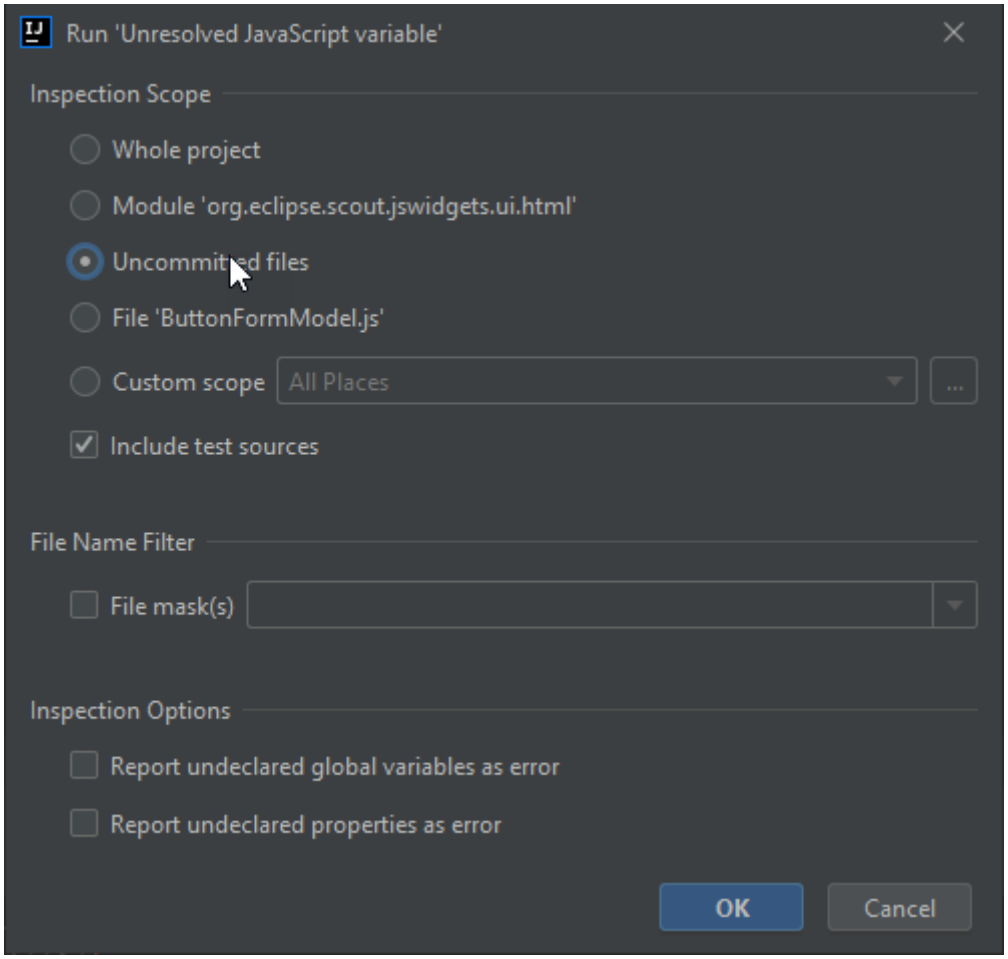

Figure 3. Run Action

c. In the result view, select one or more problems and click the button `Missing import statement`.

   Note: Sometimes `Missing import statement` is located in a drop down called `Apply quick fixes to all the problems`.

   Note: Even though we disabled all inspection options, there may still be problems other than `Missing import statement`. If that is the case, resolving multiple problems is not possible unless you unselect these problems including theirs parent node.

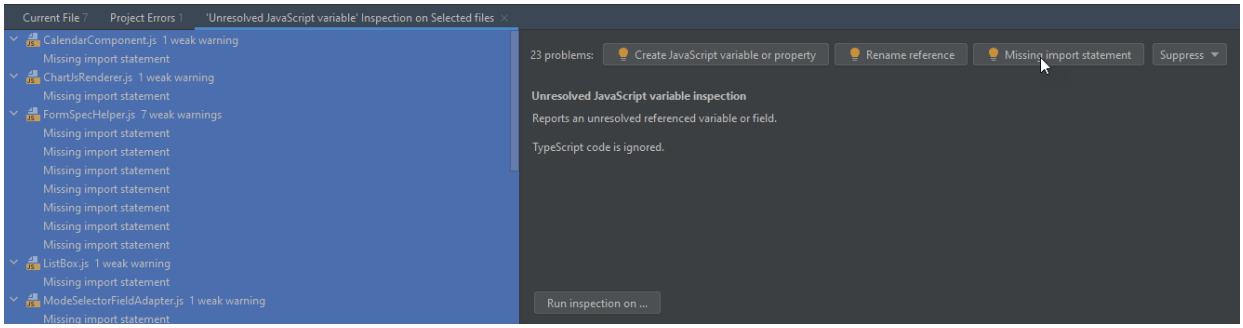

Figure 4. Resolve problems

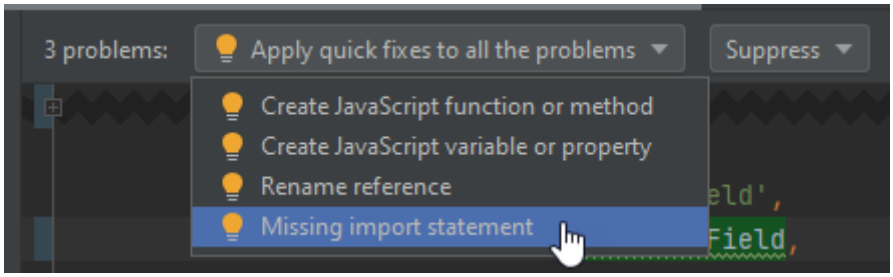

Figure 5. Button in drop down

d. Finally, you need to verify, whether the imports are correct. Please make sure all imports are present for every object type and the classes are imported from a module and not a single file.

Listing 4. Good import

```
import {Button, GroupBox} from '@eclipse-scout/core';
```

Listing 5. Bad import

```
import Button from '@eclipse-scout/core/src/form/fields/button/Button.js';
```

options-parameter for filter-methods in menus.js (Scout JS)

The methods `filter` and `filterAccordingToSelection` in `menus.js` now have an options-parameter. This parameter combines the former parameters `onlyVisible`, `enableDisableKeyStrokes` and `notAllowedTypes`. The following are two migration examples:

Listing 6. Old

```
let allowedTypes = ['Example.MenuType'],
  onlyVisible = true,
  enableDisableKeyStrokes = false;

// example 1
menus.filter(this.menus, allowedTypes, true, false);

// example 2
menus.filter(this.menus, allowedTypes, onlyVisible, enableDisableKeyStrokes);
```

Listing 7. New

```
let allowedTypes = ['Example.MenuType'],
  onlyVisible = true,
  enableDisableKeyStrokes = false;

// example 1
menus.filter(this.menus, allowedTypes, {
  onlyVisible: true,
  enableDisableKeyStrokes: false
});

// example 2
menus.filter(this.menus, allowedTypes, {onlyVisible, enableDisableKeyStrokes});
```

Rename WidgetTooltip.widget (Scout JS)

The `widget` property of `WidgetTooltip` has been renamed to `content`. Together with the property itself the corresponding setter has been renamed from `setWidget` to `setContent`.

Rename WidgetPopup.widget (Scout JS & Scout Classic)

The `widget` property on `WidgetPopup` classes (and all its subclasses) has been renamed to `content`:

1. Scout JS
   a. Update the `widget` property in all JS models creating WidgetPopups from `widget` to `content`.
   b. Rename the `_renderWidget` method on subclasses of `WidgetPopup` to `_renderContent`.
   c. `setWidget` on WidgetPopups has been renamed to `setContent`.
2. Scout Classic
   a. `IWidgetPopup#PROP_WIDGET` has been renamed to `IWidgetPopup#PROP_CONTENT`.
   b. `IWidgetPopup#getWidget` has been renamed to `IWidgetPopup#getContent`.
   c. `AbstractWidgetPopup#getConfiguredWidget` as been renamed to `AbstractWidgetPopup#getConfiguredContent`.
   d. `AbstractWidgetPopup#createWidget` as been renamed to `AbstractWidgetPopup#createContent`.
   e. `AbstractWidgetPopup#setWidget` as been renamed to `AbstractWidgetPopup#setContent`.

Rename ProposalChooser.model (Scout JS)

The `model` property on `ProposalChooser` classes (and all its subclasses) has been renamed to `content`:

1. Scout JS
   a. Rename the `_createModel` method on subclasses of `ProposalChooser` to `_createContent`.
   b. Rename the `_renderModel` method on subclasses of `ProposalChooser` to `_renderContent`.

JQuery (Scout JS)

The following functions have been removed because they are not in use anymore.

- removeThis
- suppressEventIfDisabled
- colorOpacity
- copyCssIfGreater
- backupSelection
- restoreSelection
- onSingleOrDoubleClick

The following functions have been renamed

1. widthToContent → cssWidthToContentAnimated

Table (Scout JS)

TableFilter.js has been removed because it has no benefits.

With TypeScript, the interface `Filter<TableRow>` should be used instead. With JavaScript, `extends from TableFilter` can be removed.

Testing / karma-jasmine-scout (Scout JS)

The following functions have been removed because they are not in use anymore.

- createAdapterModel
- stripCommentsFromJson
- definedProperty
- sameProperty
- JQuery.triggerMouseMove
- JQuery.triggerWithPosition

The following functions have been renamed.

- widgetCloneProperty → toHaveClonedWidgetProperty

The jquery-plugin jqueryExtensions has been refactored to a ES6 module to avoid pollution (of code completion) of the JQuery object outside test environment. In the rare case you used these functions for testing, you need to import them now, the JQuery-object does not contain them anymore.

Various Widget Adjustments (Scout JS)

The following model properties have been removed because they are not in use.

- WidgetModel.loadJsonModel
- TooltipSupportOptions.$parent

JS Build Improvements

Imports

Scout JS files cannot be imported directly anymore and need to be imported from the scout module. You are probably using the correct import style already, since file based import was bad practise anyway. In case there are file based imports, just adjust them as follows:

Listing 8. Old

```
import Table from '@eclipse-scout/core/src/table/Table'
```

Listing 9. New

```
import {Table} from '@eclipse-scout/core'
```

The following modules are affected:

- @eclipse-scout/core/src → @eclipse-scout/core
- @eclipse-scout/core/src/testing → @eclipse-scout/core/testing

This applies only to imports of `js` files. Imports to `less` files should stay untouched (for now).

IId types, IdFactory and IdExternalFormatter

Up to release 2022 the support for typed identifiers (IId) was limited to root ids wrapping one specific Java identifier. The support was extended to include composite ids wrapping multiple java identifiers. As part of this new feature a refactoring of the existing id classes was done which may require some migration steps:

- `IId` was typed with a generic parameter `<WRAPPED_TYPE>` which was removed and moved to `AbstractRootId`. If you need this generic parameter, change your code to use `AbstractRootId` instead of `IId` or consider removing the generic parameter at interface-level in your code as well.
- `IId` up to release 2022 were limited to wrap a single value. If your code rely on a single wrapped value, replace `IId` by `IRootId` and `AbstractId` by `AbstractRootId`. If you don't change your code type and use `IId` you implicitly add support for composite types in your APIs.
- Some factory methods for building ids and serialization where moved between classes and/or renamed (the serialized string representation is identical):
  - `IdExternalFormatter.getTypeName()` → `IdInventory.getTypeName()`
  - `IdExternalFormatter.getIdClass()` → `IdInventory.getIdClass()`
  - `IdExternalFormatter.toExternalForm()` → `IdCodec.toQualified()`
  - `IdExternalFormatter.fromExternalForm()` → `IdCodec.fromQualified()`
  - `IdExternalFormatter.fromExternalFormLenient()` → `IdCodec.fromQualifiedLenient()`
  - `IId.unwrapAsString()` → `IdCodec.toUnqualified()` (Note: Unwrap as string is still available for `IRootId` but according to the javadoc should only be used for logging and debugging purpose)
  - `IdFactory.createFromString()` → `IdCodec.fromUnqualified()`
- Completeness test for ids was improved. Use `AbstractIdStructureTest` as base for id completeness tests in own maven modules.
  - Implementations of `AbstractStringId` are required to create a null-id instance if invoked with an empty string (e.g. use `if (StringUtility.isNullOrEmpty(id)) { return null; }` in your static `of()` method, see `FixtureStringId` as example)
  - The former base classes `AbstractUuIdStructureTest` and `AbstractStringIdStructureTest` were integrated and removed

Renaming of DoStructureMigration to DataObjectMigration

Because the `DoStructureMigrator` executes value migrations too (`IDoValueMigrationHandler`), several renamings were applied.

Classes:

- DoStructureMigrationContext → DataObjectMigrationContext
- DoStructureMigrationCountingPassThroughLogger → DataObjectMigrationCountingPassThroughLogger
- DoStructureMigrationInventory → DataObjectMigrationInventory
- DoStructureMigrationPassThroughLogger → DataObjectMigrationPassThroughLogger
- DoStructureMigrationStatsContextData → DataObjectMigrationStatsContextData
- DoStructureMigrator → DataObjectMigrator
- IDoStructureMigrationGlobalContextData → IDataObjectMigrationGlobalContextData
- IDoStructureMigrationLocalContextData → IDataObjectMigrationLocalContextData
- IDoStructureMigrationLogger → IDataObjectMigrationLogger

Test classes:

- TestDoStructureMigrationInventory → TestDataObjectMigrationInventory
- TestDoStructureMigrator → TestDataObjectMigrator

Methods:

- DataObjectMigrationInventory#getMigrationHandlers → #getStructureMigrationHandlers
- DataObjectMigrationInventory#getDoMigrationContextValues → #getDoStructureMigrationTargetContextDatas

Removals (see deprecation warning in 22.0 for further information):

- DoStructureMigrator#migrateDataObject(DataObjectMigrationContext, IDataObject)
- DoStructureMigrator#migrateDataObject(DataObjectMigrationContext, IDataObject, IDataObjectMigrationLocalContextData…)
- DoStructureMigrator#migrateDataObject(DataObjectMigrationContext, IDataObject, NamespaceVersion, IDataObjectMigrationLocalContextData…)
- TestDoStructureMigrationInventory#TestDoStructureMigrationInventory(List<INamespace>, Collection<ITypeVersion>, Collection<Class<? extends IDoStructureMigrationTargetContextData>>, IDoStructureMigrationHandler…)

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