Eclipse Scout Migration Guide

Version 22.0

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 11.0 to Eclipse Scout 22.0**. If existing code has to be migrated, instructions are provided here.

Obtaining the Latest Version

Scout Runtime for Java

Scout Runtime artifacts for Java are distributed using Maven Central:

- 22.0-SNAPSHOT on *Maven Central*
- 22.0-SNAPSHOT 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>22.0-SNAPSHOT</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",
  "dependencies": {
    "@eclipse-scout/core": "22.0.0-snapshot",
    "jquery": "3.5.1"
  }
}
```

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)

Starting with Scout 11 the IDE Tooling requires at least Java 11 to run. With the help of the SDK plugins you can still develop applications running with Java 8, but the IDE itself requires Java 11 or newer.

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:
http://download.eclipse.org/scout/releases/11.0/

TypeVersion Annotation Type Change

The type version of a data object is used to identify a certain structure version of the stored data object. A data object may be stored in a database or be available as a container to export certain data for import in a different compatible system. Such a data object may evolve over time and undergo structural changes. Some structural changes make it necessary to apply migrations to existing serialized data objects to comply with the new structure.

In order to prepare for migration support, the type version annotation so far containing a String value changes to a value of Class<? extends ITypeVersion>. A ITypeVersion represents a namespace/version and it's dependencies.

Migration

For each different String value used in type version annotation, create an implementation of `ITypeVersion` as described in *Data Objects/Namespace and ITypeVersion* in the technical documentation.

`DataObjectInventory#getTypeVersion` returns `NamespaceVersion` instead of `String`. Use `NamespaceVersion::unwrap` to access text representation.

Annotation EnumVersion removed

`EnumVersion` was designed for migration support similar as the TypeVersion but was never part of any serialization output of a data object, therefore couldn't be used as indicator for migrations. Support for `EnumVersion` is removed.

Migration

Remove `EnumVersion` annotation on `IEnum` implementors.

Native Notification Support

The new notifications displayed by the browser use the application logo configured in `AbstractDesktop#getConfiguredLogoId()` by default.

If you use native notifications, you should provide a logo with a resolution of at least 150x150 px. If your application logo already has such a resolution, it should be fine. If your application logo has a lower resolution or is an SVG, you should use a different image for the notifications (SVGs are not supported by Chrome notifications). To do so, just configure the native notification defaults on your desktop.

```
@Override
protected NativeNotificationDefaults getConfiguredNativeNotificationDefaults() {
  return super.getConfiguredNativeNotificationDefaults()
    .withIconId("notification_logo.png");
}
```

Application Logo / Info Form

The image `application_logo_large` and the constant `AbstractIcons.ApplicationLogo` have been removed. The name was confusing and it was only used for the `ScoutInfoForm`. The info form now uses the logo of the desktop (`IDesktop#getLogoId()`) by default. So if you prefer to use a different logo for the info form, just extend the info form, override the method `getProductLogo()` and return the name of your preferred image.

In case you don't use SVG logos yet, you should consider doing so to prevent blurry logos.

LESS Variables

- @navigation-background-color → @desktop-navigation-background-color
- @navigation-color → @desktop-navigation-color
- @outline-title-margin-left/right → @outline-title-padding-left/right

Notes: @desktop-navigation-background-color now points to @desktop-header-background-color; Instead of customizing the navigation background color it is suggested to now customize the header background color.

Browser Field

The type of the `data` argument for the callback `execPostMessage` was changed from *String* to *Object*. This allows for the widest variety of data that can be sent from an embedded page to your application:

- String
- Number
- Boolean
- IDataObject (objects or arrays)

In previous Scout versions, all messages were always converted to text. Now, the data type is preserved as accurately as possible. JSON objects or arrays are converted to IDoEntity or DoList with the help of the IObjectMapper bean. To use this feature, an implementation of IDataObjectMapper needs to be present at runtime. If no implementation is available, the data will be converted to text automatically.

Migration:

Adjust the signature of all implementations of `AbstractBrowserField#postMessage` in your code.

- Change `execPostMessage(String data, String origin)` to `execPostMessage(Object data, String origin)`.
- If the message sent by the embedded web page is not a text, the appropriate data type is now passed. Check the expected type using `instanceof` or convert it to String manually.
- If you want objects and arrays to be converted to IDataObjects automatically, make sure there is an implementation of IObjectMapper present at runtime, e.g. by adding a dependency to the module *org.eclipse.scout.rt.jackson* in pom.xml.

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