

Eclipse Scout

Release Notes

Scout Team

Version 6.1

Table of Contents

Overview	1
Service Release Change Log	1
What's New	1
Strong content security policy CSP	1
Enforcement of Model Thread	1
Properties support for Lists, Maps and imports.....	2
Binary resources support for HtmlField and BeanField	2
TriState capability for check boxes	3
New UriOpenAction: SAME_WINDOW	3
New methods in StringUtility	3
New ObjectUtility.....	3
New form field property "preventInitialFocus"	4
Multiple Dimensions Support	4
Form Field Enabled Inheritance	4
Table: new property "groupingStyle"	5
UnloadRequestHandler for <code>navigator.sendBeacon()</code>	5
Preparations for Scout JS	5
Button: new property 'defaultButton'	6
Icons for tree nodes.....	6
Improved iOS Homescreen Mode (since 6.0.300).....	6

Overview

Eclipse Scout 6.1 is a preview of the Eclipse Oxygen release. The latest stable version is *6.1.0.B006*. The official release for Eclipse Oxygen is Eclipse Scout 7.0 and will be released in June 2017.

- Download SDK: [Eclipse for Scout Developers](#)
- Runtime on Maven Central: [6.1.0.B006](#)

Service Release Change Log

Neon.3 (6.0.300)

- [Improved iOS Homescreen Mode \(since 6.0.300\)](#)
- Detailed change log: <https://github.com/eclipse/scout.rt/compare/6.0.200.0...6.0.300.RC2>.

Neon.2 (6.0.200)

- Detailed change log: <https://github.com/eclipse/scout.rt/compare/6.0.100.RC4.2...6.0.200.0>.

Neon.1 (6.0.100)

- [\[Responsive Table \(since 6.0.100\)\]](#)
- Detailed change log: <https://github.com/eclipse/scout.rt/compare/6.0.0.RC4...6.0.100.RC4.2>.

What's New

This document shows some of the new features delivered with the release 6.1. The release contains a lot of bugfixes and even some features not mentioned here. If you are interested in the detailed change log see <https://github.com/eclipse/scout.rt/compare/releases/6.0.x...releases/6.1.x>.

Strong content security policy CSP

The stronger CSP disables inline javascript in html. Therefore the 'New Scout Project' wizard now creates a js file per html file and includes it using the script element. To migrate existing projects, see the [Scout Migration Guide](#).

Enforcement of Model Thread

Every operation which results in a modification of the model and eventually of the ui has to be performed by the model thread. This has been true for a long time and still is. If the wrong thread was used, unexpected behavior was the result, like a delayed update of the ui or concurrency exceptions. To prevent such behavior in the future, an exception will be thrown if an operation is executed in the wrong thread.

If you get such an exception, you'll need to wrap your operation in a model job and schedule it using `ModelJobs.schedule()`, see the chapter ModelJobs in the [Scout Technical Documentation](#) for details.

Properties support for Lists, Maps and imports

Config properties support list- and map-data-structures. Furthermore other properties files can be imported. See the tech documentation section "Configuration Management" for more details.

Binary resources support for HtmlField and BeanField

Binary resources such as images or videos can now be used in the following widgets:

- HtmlField
- BeanField

Examples

BeanField

This is a BeanField

A BeanField is useful to display data in a custom way. The value of the field is a bean containing all the data relevant for presentation. Thus, compared to the HtmlField, the HTML itself is not delivered by the server but created on the browser using JavaScript and the raw data of the bean.

The header and description are part of the bean sent by the server. This is not necessary, you can access texts directly using javascript, like it is done for this text.

AppLinks are supported as well [Click me!](#)



Figure 1. Binary resource on a model field.

- Html enabled StringColumn
- BeanColumn



Image (Html enabled StringColumn)	Image (BeanColumn)	Name	Size	Type	Date Modified
		bird.jpg	206 KB	JPG Image	01.01.70 00:59

Figure 2. Binary resource on a column

TriState capability for check boxes

Added support for tri-state value (`true`, `false` and `null` instead of just `true` and `false`) to boolean field and boolean column.

The new property `triStateEnabled` controls whether the boolean field/column behaves as a normal checkbox (false) or a tri-state checkbox (true).

A normal checkbox has values true/false. A tri-state checkbox has values true/false/null. The null value is interpreted as "undefined" and rendered as a filled square.

New UriOpenAction: SAME_WINDOW

The enum `UriOpenAction` provides a new value to open a URI in the current window: `SAME_WINDOW`

New methods in StringUtility

`StringUtility` provides the following new methods:

- `containsString()`
- `containsStringIgnoreCase()`
- `containsRegEx()`
- `matches()`
- `endsWith()`
- `startsWith()`
- `length()`
- `indexOf()`
- `lastIndexOf()`
- `split()` (with *limit* argument)

All methods are null-safe, unit tested and documented with JavaDoc.

New ObjectUtility

`ObjectUtility` was added as new utility for generic object methods and provides null-safe implementations of various `Object` methods. Various methods from former `CompareUtility`:

- `equals()`
- `notEquals()`
- `nvl()`
- `isOneOf()`
- `compareTo()`

And a new method `ObjectUtility.toString(Object)` providing a null-safe implementation of `Object.toString()` returning `null` if specified object is `null`.

New form field property

"preventInitialFocus"

By default, the first enabled field on a form gets the focus when the form is opened. This may not be desired in some cases (e.g. if the first field is a HTML field that contains app links). The new property `PROP_PREVENT_INITIAL_FOCUS` can be used to prevent the initial focus to be set to this field. The default value is `false`. For `AbstractHtmlField` and `AbstractBeanField`, the default is set to `true`.

Multiple Dimensions Support

Some components now support more dimensions for various attributes. E.g. until now there have been two dimensions for Form Field visibility: visible and visible-granted. Now there are also custom dimensions available. See the chapter 'Multiple Dimensions Support' in the [Scout Technical Documentation](#) for details and examples.

Currently the following attributes support multiple dimensions:

- Actions: visible, enabled
- Columns: visible
- Tree Nodes: visible, enabled
- Outlines: visible
- Form Fields: visible, enabled, label-visible
- Data Model Attributes: visible
- Data Model Entity: visible
- Wizard Steps: visible, enabled
- Trees: enabled
- Tables: enabled

Form Field Enabled Inheritance

The inheritance of the enabled property for Form Fields has been changed. Now the enabled properties are no longer propagated to children if it is changed on a composite field. Instead a field is only considered to be enabled if itself and all of its parents are enabled. This allows to toggle an entire box to disabled and back to enabled without touching the child fields. This has the advantage that the original state is restored when the box is set back to enabled.

With this change the `getConfiguredEnabled` on composite fields now also automatically affects children. There is no need to overwrite `execInit()` and call `setEnabled(false)` anymore.

Table: new property "groupingStyle"

The new property `groupingStyle` can be set to `bottom` (default) or `top`. Depending on the value aggregate rows are rendered on the bottom of grouped rows or on the top of grouped rows. The new top style can be set to have an aggregate row as a title for a group of table rows, this is useful for separating a table into multiple categories.

UnloadRequestHandler for `navigator.sendBeacon()`

When a client leaves the application (e.g. puts `about:blank` in the address bar) one last "unload" request to the UI server is sent in order to properly clean up the session on the server.

If the browser supports the `Beacon API` `navigator.sendBeacon()` is used for this request. Unfortunately `application/json` is not a CORS-safelisted request-header which implies that we can't use the `JsonMessageRequestHandler` for the unload handling. Therefore a separate `UnloadRequestHandler` was introduced which handles all requests to `/unload/[UiSessionId]`. (For more information, see <https://git.eclipse.org/r/#/c/89422/>)

To cut a long story short, new traffic to `/unload` will be sent by the clients. Please check your container and firewall configuration.

Preparations for Scout JS

A classic Scout application has a client model written in Java, and a UI which is rendered using JavaScript. With this approach you can write your client code using a mature and type safe language. Unfortunately you cannot develop applications which have to run offline because the UI and the Java model need to be synchronized.

With Scout JS this will change. You will be able to create applications running without a UI server because there won't be a Java model anymore. The client code will be written using JavaScript (or TypeScript) and executed directly in the browser.

This release (6.1) is the first step in this direction. Several actions have been performed:

1. Created `scout.App`

The new `App` object represents the *Single Page Application*. It will be initialized when the page loads and prepares all the necessary things the application needs to run, like texts, codes, fonts, logger and the session. These things may be different in case of a classic remote application and a Scout JS application. That is why there is another app called `scout.RemoteApp` which extends the `scout.App`. For you it basically means: if you create a Scout Classic App, use `scout.RemoteApp`, otherwise use `scout.App`.

2. Separated Widget and Model Adapter

A `ModelAdapter` is the connector with the server, it takes the events sent from the server and calls the corresponding methods on the widget. It also sends events to the server whenever an action happens on the widget. So if there is no server, there is no need for such adapters. This means

in a Scout JS app you will only work with widgets, adapters are only required for remote apps.

3. Enhanced Widgets

With a Scout Classic app a lot happens on the UI server, like validating a form when the ok button is pressed. We started to enhance the JavaScript widgets with similar functionality and added API to use them. One example: The `ValueField` on Java side has a value and a display text. If a text is entered it will be parsed to get the value, or if a value is set the format function is called to get the text. This has not existed on JavaScript side, because the server only sent the text. This has been changed, parse and format functions now exist on the JS `ValueField` as well.

The preparations done in this release are just the first step. You could create a Scout JS app with this release, but a lot of the widgets are not ready to use yet. See also the [Scout Migration Guide](#) to migrate your existing JavaScript code.

Button: new property 'defaultButton'

A button may now be marked as default button which gives him a dedicated look to attract users attention. It will just change the look, the behavior stays the same.

Note: The first button or menu which has an `Enter` keystroke will automatically get that look too. This is existing behavior and hasn't changed. The new property has been added to give you more control, but actually you should always prefer the *enter keystroke approach* to provide a consistent behavior.

Icons for tree nodes

As in earlier Scout releases with Swing, SWT and RAP UI, the Outline and all Trees in Scout now support an icon per tree node. Simply set the `iconId` property on a `TreeNode` and reference either a character from an icon-font in your Scout project or a bitmap icon which is defined in your Scout project. See the migration guide for more details and the global property `showTreeIcons` which can turn on/off icons for all Tree instances. You should take care that all icons you use in a single tree have the same size. Here's an example for an outline with icons:

Improved iOS Homescreen Mode (since 6.0.300)

If the app is running in iOS home screen mode the HTTP session will be lost whenever the user leaves the app (e.g. switches to another app or just downloads a file). This means he has to login again and navigate to the previous location, again. To avoid this a persistent mode has been introduced. This means if the app is running in the home screen mode, a persistent session cookie is created so that the same http session may be used the next time the app is activated. Also, the client session id will be put in the local storage instead of the session storage. This makes sure the same client session as before is used. Deep link handling is deactivated in that case otherwise it would always navigate to the url which was active when "add to home screen" was pressed.

This persistent mode is only active when the app is running on iOS with the home screen mode.

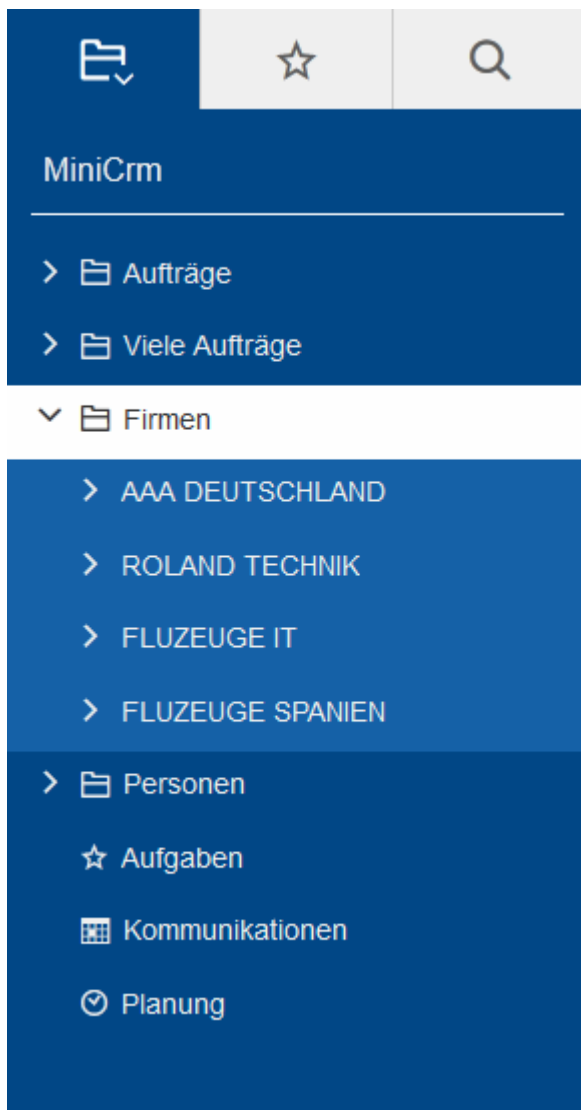


Figure 3. Outline with icons



Do you want to improve this document? Please [edit this page](#) on GitHub.