Eclipse Scout
Migration Guide

Scout Team

Version 7.0

Table of Contents

Migration Guide from Scout 6.1 to Scout 7.0

If you upgrade from version 6.0, also see the migration guide for the 6.1 release.
https://eclipsescout.github.io/6.1/migration-guide.html

API Changes (Java)

New scout modules for Jackson-based REST services

The following new Scout modules were added to support Jackson-based REST services:

- `org.eclipse.scout.rt.rest`
- `org.eclipse.scout.rt.rest.test`
- `org.eclipse.scout.rt.jackson`
- `org.eclipse.scout.rt.jackson.test`

In order to use REST services based on the JAX-RS Jersey implementation, the following section must be added to the project `web.xml`:

```
<!-- JAX-RS Jersey Servlet -->
<servlet>
  <servlet-name>api</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>javax.ws.rs.Application</param-name>
    <param-value>org.eclipse.scout.rt.rest.RestApplication</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

The `RestApplication` class searches for all implemenations of `IRestResource` and exposes them as REST services.

Removed properties "enabledProcessing" on IButton and IAction

When a user performs two consecutive clicks on a button or a menu within a very short period of time, the action is executed twice. In some cases, this is an unwanted behavior. A new property "preventDoubleClick" was added to IButton and IMenu to disable it (see release notes).

In older Scout versions, two flags (`m_enabledProcessing` and `m_enabledProcessingAction`, respectively) were use to accomplish a similar objective. Whenever the click/action handler was started, the flag was set to `false`, and set back to `true` at the end of the execution. A second call to the method had no effect. However, this logic no longer works since the introduction of the HTML UI. The new UI strictly allows *only one* UI event to work on the client model simultaneously (subsequent click events are hold back by the *UiSession* lock). Consequently, the "enabledProcessing" flag is never `false`, and no clicks are prevented.

Because this construct no longer works, the corresponding properties and methods were removed, namely:

- ~~AbstractButton: m_enabledProcessing~~
- ~~IButton/AbstractButton: isEnabledProcessing()~~
- ~~IButton/AbstractButton: setEnabledProcessing()~~
- ~~IButton/AbstractButton: disarm()~~
- ~~ButtonEvent: TYPE_DISARM~~
- ~~AbstractAction: m_enabledProcessingAction~~
- ~~IAction/AbstractAction: isEnabledProcessingAction~~

Migration: Delete any code that references any of the removed methods. There is not replacement, as the whole concept became disfunctional with the HTML UI. To prevent accidental double clicks, use the new `getConfiguredPreventDefault()` property. If for very rare and special cases such a flag is required, it has to be implemented manually by overriding `AbstractButton.doClick()` and `AbstractAction.doAction()`.

Added HTTP abstraction layer Google HTTP Client for service tunnel.

If your application previously used a custom (not provided by Scout) implementation of the `org.eclipse.scout.rt.shared.servicetunnel.http.HttpServiceTunnel` class a migration might be necessary. For the service tunnel a new HTTP abstraction layer (Google HTTP Client 1.22) was introduced to support different low-level HTTP libraries (previously just the `java.net.HttpURLConnection` was used).

For the `HttpServiceTunnel`class the method `URLConnection createURLConnection(ServiceTunnelRequest call, byte[] callData)` has been replaced by `HttpResponse executeRequest(ServiceTunnelRequest call, byte[] callData)`, also several methods (`addCustomHeaders`, `addSignatureHeader`, `addCorrelationId`, `createAuthToken` and `interceptHttpResponse`) signatures have changed their `URLConnection` parameter to either a `HttpRequest` or `HttpResponse` parameter.

API Changes (JavaScript)

render()

The parameter `$parent` has been removed from the `_render` method because `this.$parent` is available for every widget. There is no need to have a parameter `$parent` which points to the same variable. Use `this.$parent` instead.

Also `$parent` is now optional when calling widget.render(). The `$parent` may be resolved using `this.parent`. No need to always write `widget.render(this.$container)` anymore, instead just write `widget.render()` if the `$container` of the `parent` should be used as `$parent`.

Promises

With jQuery 3 the promise API is now Promises/A+ compliant. This means you may need to adjust your code if you use promises.

We noticed the following effects:

- If a rejection is catched using a fail handler, the fail handler has to return a rejected promise as well, otherwise the next success handler would be called instead of the next fail handler.
- Every callback is now executed asynchronously. This is especially relevant for the tests.
- Catch has been added → replace fail(null, func) for better readability.

See also https://jquery.com/upgrade-guide/3.0/ for details.

Removed addClassSVG, removeClassSVG, attrSVG, removeAttrSVG

These functions are now supported by jQuery directly. Just use `addClass`, `removeClass`, `attr` and `removeAttr`.

Property Change Event

The property change event has been simplified.

The event had 3 properties:

- ~~`newProperties`~~
- ~~`oldProperties`~~
- ~~`changedProperties`~~

This was added to be able to react to multiple property change events at once. Since 6.1, bulk property changes don't exist anymore, so there is no need for these properties anymore.

Now, with 7.0, the property change event has the following properties:

- `name`
- `oldValue`
- `newValue`

This makes handling the event easier. Check your propertyChange event handlers and adjust them accordingly.

Other Changes

Maven provided dependencies

In Maven dependencies with the scope provided are not transitive. This makes sense if a dependency is set to provided depending on the environment. Any artifacts that are not intended to be used in a certain environment should not have the scope provided and are therefore now transitive. We removed any current dependency `javax.servlet:javax.servlet-api` except for the one in the artifact `org.eclipse.scout.rt.server.commons`.

To migrate your project, remove any dependency to `javax.servlet:javax.servlet-api`, `javax.xml.ws:jaxws-api` or `javax.ws.rs:javax.ws.rs-api`. Then add to all artifacts with packaging type war the dependency to `javax.servlet:javax.servlet-api` with scope provided. Depending on

the container, you may want also to add the depdendency `javax.xml.ws:jaxws-api` with scope provided to the war artifact.

```
<project>

  <artifactId>myproject.server.war</artifactId>
  <packaging>war</packaging>

  <dependencies>
    <dependency>
      <groupId>myproject</groupId>
      <artifactId>myproject.server</artifactId>
    </dependency>

    <!-- provided by container -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

Migration Guide from Scout 6.0 to Scout 6.1

API Changes (Java)

Text Provider Service

The method `AbstractDynamicNlsTextProviderService#getDynamicNlsBaseName` has been made public. Adjust the method in your text provider service accordingly.

Mnemonics

Mnemonics are not supported anymore. All affected texts were either edited or removed because they are not used anymore.

Migration: Remove all mnemonics (`&`) from your text files as they will not be considered anymore. Replace `&&` with `&`. (`&&` was used to escape the mnemonic behaviour and display a single `&` in a text.)

The following methods were or will be removed:

- `StringUtility.removeMnemonic`
- `StringUtility.getMnemonic`
- `IAction.PROP_TEXT_WITH_MNEMONIC`
- `IAction.PROP_MNEMONIC`

- `IAction.getTextWithMnemonic`
- `IAction.getMnemonic`
- `strings.removeMnemonic`
- `strings.removeAmpersand`

getFocusOwner

Method getFocusOwner() was removed from IDesktop, IForm and DesktopEvent. Since replacing the old rich client ui technologies (swing, swt) with the modern html ui, this method didn't work correctly anymore.

There are no plans to implement correctly because of multiple reasons. It would increase network traffic between browser and ui server and also would be quite unreliable. The old behaviour was a synchronious result from the ui (swing, swt), which was feasible in rich client environments. But with a distant browser, a realtime result is hard to achieve and might already by outdated by its arrival at the ui server.

If such functionality is needed, it has to be programmed with java script within the browser.

FinalValue

`setIfAbsent` has been renamed to `setIfAbsentAndGet`. `setIfAbsent` now returns a boolean denoting, if a value was set or not.

@PostConstruct

A method annotated with `@PostConstruct` in a Bean is now guaranteed to run exactly once. The constructor may still run more than once.

Tree

The method `AbstractTree#execAutoCheckChildNodes` got two new parameters and the default implementation now considers `enabledNodesOnly` and does not always ignore disabled nodes.

Table

`Cell.setText(null)` has no effect anymore. The JavaScript table has been improved so that every column is now able to compute the text based on the value of a cell. This does only happen if no explicit text is provided which means `cell.setText(null)` would trigger that behavior. If you really want to set a value but no text, you can use `cell.textText("")` instead.

CookieUtility

`CookieUtility` was moved from *org.eclipse.scout.rt.ui.html* to *org.eclipse.scout.rt.server.commons*. Migrate by updating your imports. The Method `addCookie()` is now called `addPersistentCookie()`. Additional methods are available (to add a session cookie, or delete an existing cookie).

Pair

The `Pair` class was made abstract and two default implementations for a mutable pair (class `MutablePair`) and immutable pair (`ImmutablePair`) were added. Since the former `Pair` class was immutable, all occurences were changed to use the new `ImmutablePair` class. Migrate by update all occurences to use the new `ImmutablePair`.

Customizing CSP directives

The method `org.eclipse.scout.rt.server.commons.servlet.HttpServletControl.getCspDirectives()` is no longer available. CSP directives are now configured by the the bean `org.eclipse.scout.rt.server.commons.servlet.ContentSecurityPolicy`. To customize the rules, replace this bean with your own implementation and override the method `initDirectives()`. The bean provides fluent-style *withFooBar()* methods.

StringUtility.contains() deprecated

The method `StringUtility.contains()` was marked as deprecated and will be removed in the P-release. The method was often used incorrectly due to poor documentation and unconventional implementation. The utility provides multiple new methods that can be used as a replacement:

containsString()

null-safe variant of String.contains()

containsStringIgnoreCase()

like containsString(), but ignores capitaliziation. Make sure to read the JavaDoc!

containsRegEx()

checks if the given regular expression matches part of the given string (essentially, this method automatically adds `.*` on both sides of the regular expression)

matches()

null-safe variant of String.matches(), also allows to set the pattern flags

BrowserInfo

The class `BrowserInfo` was renamed to a more generic `HttpClientInfo` name, since the HTTP client can either be a browser, but may also be another server using the built-in HTTP client of the VM.

Futher the `HttpClientInfo` instance for each request is cached on the current HTTP session, if a session is available. Use the new `HttpClientInfo get(HttpServletRequest request)` method to get the cached HTTP client info.

Virtual Tree Node

The Virtual tree node has been deleted. The main reason for this was because of table pages: If an `AbstractPageWithTable` contains a lot of rows, for each of them a child page is created. To have these

child pages as lightweight as possible the virtual node was introduced. This node was created for each row and only after activating a row (click by the user) the real child page has been created.

Now instead of creating a virtual node first an probably the real page afterwards the real page is created directly. Therefore the instance creation of pages below table pages should be very fast and not perform any backend calls. To assure this it is recommended to move any expensive operation currently implemented in the `execInit()` method to `execCreateChildPages()` or `execPageActivated()`. Permission checks or similar operations, which use the `setVisibleGranted(boolean)` method, should be moved to the newly created `execCalculateVisible()` method. The default behavior is that the `execCalculateVisible` method is executed on instance creation. Subclasses of `AbstractPageWithTable` potentially have a large number of child pages. To avoid performance issues due to expensive permission checks, the `execCalculateVisible` for these children is only executed before loading the page data.

Furthermore to save memory it is recommended to create the tables below pages lazily. The table is created upon the first access to `IPage.getTable()`. Therefore try not to use `getTable()` in the page init phase. Instead a new callback `execInitTable` is available to initialize the table at the moment it is created. There is also an overload `getTable(boolean)` that can be used to access the table without automatically creating it.

Enabled Property of Form Fields

The inheritance of the enabled property of form fields has been changed so that changing this property on a composite field does no longer automatically propagate the value to the children. Instead a form field is only considered to be enabled if all parent fields are enabled too.

To have the same behaviour you can use the method `formField.setEnabled(yourValue, true /* update parents */, true /* update children */)` which also propagates the value to parent and child fields. The same method also exists for the enabled-granted property: `formField.setEnabledGranted(yourValue, true, true)`. However often it may no longer be required to actively propagate the new value to children. Therefore it is recommended to check the business logic manually where possible.

Furthermore the meaning of property change listeners changed. Check all the listeners using the `org.eclipse.scout.rt.client.ui.form.fields.IFormField.PROP_ENABLED` property. This property is now only fired if the state of the form field itself has changed. If the enabled state of a parent field is modified, this property change event will no longer be fired. The actual enabled state of the field could have changed even though because the parents have an influence now. If the listener should also be notified about changes of the parents use the new property `org.eclipse.scout.rt.client.ui.form.fields.IFormField.PROP_ENABLED_COMPUTED`.

Icons in Tree

When the new Html UI was introduced the support for icons on tree nodes was dropped. But some projects really missed that feature so it was introduced again with this release. This means when your tree node provides an `iconId`, the UI will display the icon referenced by the ID. The tree supports bitmap and font-icons. Since there are Scout projects migrating from an older Scout version (before Html UI was introduced) to a Scout version with Html UI (but before 6.1) they may

still have iconIds configured, but since these icons were never displayed in their application, they probably want to stick with that behavior without changing their getConfiguredIconId() methods. For that purpose the Session init property `showTreeIcons` was introduced. By default the property is false, which means you won't see icons in the Tree, even if your model has an iconId configured. Set the property to true, to enable the support for icons (this will be default starting from release 6.2.x). Example for `index.js`:

```
$(document).ready(function() {
  var app = new scout.RemoteApp();
  app.init({
    session: {
      showTreeIcons: true
    }
  });
});
```

ITableHolder

The class `ITableHolder` was part of the old array based table loaders which has been replaced with a bean based approach in the last release. Therefore the class `ITableHolder` has been removed. The constants that were present on that interface can be accessed using the interface `org.eclipse.scout.rt.client.ui.basic.table.ITableRow` from the client or `org.eclipse.scout.rt.platform.holders.ITableBeanRowHolder` from outside the client.

NumberUtility.nvl(), DateUtility.nvl(), StringUtility.nvl()

The `nvl()` methods on NumberUtility, DateUtility and StringUtiltiy were moved to a generic `ObjectUtility.nvl()`. The existing methods were deprecated and will be removed with next Scout release. Additionally the existing methods were restricted to use Number respectively Date only.

StringUtility.substituteWhenEmpty()

The existing methods was deprecated and will be removed with next Scout release. Use StringUtility.hasText() and StringUtility.emptyIfNull() or StringUtility.nullIfEmpty() instead.

CompareUtility

The various null-safe compare methods on CompareUtility were moved to the new generic ObjectUtility. The existing methods were deprecated and will be removed with next Scout release.

ThreadInterrupted-, TimedOut- and FutureCancelledExceptions ("extends java.lang.RuntimeException") are now PlatformErrors ("extends java.lang.Error")

There were circumstances where the cancellation of long-running actions did not work or lead to unpleasant behaviors (for example multiple `ExceptionForm`, that is displayed after a cancellation by the user). Most of time caught exceptions where the reason for such behaviors.

In order to get rid of those problems, we have decided that the former `RuntimeExceptions` will become `Errors` and therefore should no longer be swallowed by `catch (RuntimeException e)`. See **Eclipse Scout: Technical Guide** for more information about the new `Throwable` hierarchy.

Type of "labelPosition" property changed to "byte" (IFormField)

The type of the labelPosition property was changed from `int` to `byte`. This affects the setters, getters and `getConfiguredLabelWidth` methods. The position constants in `IFormField` were adjusted.

Occurrences where such methods were overridden need to be adjusted. Otherwise no changes should be necessary.

IDeviceTransformer

Some methods on `IDeviceTransformer` where changed. Projects with own contributions to the device transformation process must apply these changes accordingly.

Old method	New method	Description
—	`transformPageTable(table, page)`	New callback that can be used to transform the page's table. Unlike `transformPage` this method is not called during the *execInitPage* phase, but during the *execInitTable* phase.
`transformPageDetailForm(form)`	`notifyPageDetailFormChanged(form)`	The existing method was renamed to avoid confusion with `transformPageTable` and to clearify that this method is called every time, the desktop's detail form changes (not only when the detail form is first initialized).

Old method	New method	Description
`transformPageDetailTable(table)`	`notifyPageDetailTableChanged(table)`	The existing method was renamed to match the new method `notifyPageDetailFormChanged` and to clearify that this method is called every time, the desktop's detail table changes (not only when the detail table is first initialized).

AbstractTree.getConfiguredMultiSelect() deprecated

The method `AbstractTree.getConfiguredMultiSelect()` was marked as deprecated. Multiselection on trees was never supported by the UI even though the model suggested so. The method will be declared final in the next Scout release, with its default implementation returning false, in case multiselection support is added in a future release.

API Changes (JS)

scout.graphics.prefSize()

The signature of JavaScript method `scout.graphics.prefSize()` has changed:

- Old: `scout.graphics.prefSize($elem, ~~includeMargin~~, options)`
- New: `scout.graphics.prefSize($elem, options)`

The argument *includeMargin* was moved to the options object. See code documentation for a description of all options.

scout.ModelAdapter

If you have not created any custom widgets, you can skip this. If you only used `BeanFields` for customizing you can skip it as well.

Previously every widget with a corresponding part on the server extended `scout.ModelAdapter`. A model adapter 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. To make the widgets usable without a server, they don't extend from `scout.ModelAdapter` anymore but directly from `scout.Widget`. That means every widget with a server counter part have been separated into widget and model adapter, similarly to the server side where a `IJsonAdapter` exists for every model object. The model adapter creates the widget and attaches itself to it meaning it listens for events triggered by the widget and sends elected ones to the server. It also takes the events from the server and calls the corresponding methods of the widget.

So if you created custom widgets you have to separate them as well. Create for each widget a separate file called the same way as the Widget + 'Adapter'. That adapter extends either directly from `scout.ModelAdapter` or from the corresponding adapter of the parent widget.

Example: You have created a `XyField.js` which extends from `FormField.js`. Now create a file called `XyFieldAdapter.js` and extend it from `FormFieldAdapter.js`.

You now have to move the server event handling methods to the adapter, if there are any at all. If your widget does not contain a method called `onModelAction`, you are fine. Beside these action events the server may send property change events as well. For every property change event the adapter will automatically call the corresponding setter method. If there is none it will call the generic method `Widget.setProperty` which eventually calls the `_sync` and `_render` methods of the property. So if your widget contains `_sync` methods they will still be called on a server property change like before. But now you should create a JS property event to inform other widgets by using `Widget._setProperty` (note the _). This was previously done automatically for every property which is still done if there is no `_sync` method. If there is one you have to take care of it by yourself.

For the opposite direction meaning events from UI to server you have to more or less replace the calls of `_send()` with `trigger()`. In the adapter you have to handle these widget events and call the `_send()` method accordingly. If it is a property change event it is even simpler. Just call `_addRemoteProperties` in the constructor of the model adapter for every property which should be sent to the server.

scout.ModelAdapter._send()

The signature of JavaScript method `scout.ModelAdapter._send()` has changed:

- Old: `scout.ModelAdapter._send(type, data, ~~delay, coalesceFunc, noBusyIndicator~~)`
- New: `scout.ModelAdapter._send(type, data, options)`

Instead of passing individual arguments, pass all but the first two arguments in an options object: * `delay` * `coalesce` * `showBusyIndicator`

Old:

```
this._send('selected', eventData, null, function() { ... });
```

New:

```
this._send('selected', eventData, {
  coalesce: function() { ... }
});
```

scout.Widget

If you have not created any custom widgets, you can skip this.

destroy()

With the separation of widget and model adapter the destroy handling has been refactored. This

means every widget may now be destroyed. Previously only the widgets which extended from `scout.ModelAdapter` could be destroyed. The big advantage is that every widget now behaves the same and that there finally is a counter part for the `_init()` called `_destroy()` which makes it possible to do cleanup like removing listeners.

For you it means you have to decide whether you want to destroy or only remove your widgets. A widget knows the following states:

1. initialized
2. rendered
3. removed
4. destroyed

You can remove and render the same widget as many times you want, but if you destroy it you may not use it again and you would have to create a new one. It eventually has to be destroyed though for a proper cleanup. Normally this is done by the parent widget, but in some rare cases you have to take care of it by your own.

So check all the occurrences of `YourWidget.remove()` and maybe replace them with destroy.

EventSupport

Every widget now installs the event support by default. Previously `_addEventSupport` had to be called in the constructor of the widget. This may now be removed.

KeyStrokeContext

The method `_addKeyStrokeContextSupport` has been removed. If your widget needs keystroke support override `_createKeyStrokeContext` and provide one. You can probably use the default `scout.KeyStrokeContext`. The parameter of `_initKeyStrokeContext` has been removed as well. Just use `this.keyStrokeContext` instead.

Changes in "objectType" syntax and scout.create()

The "objectType" is a string describing which JavaScipt "class" to use when creating an object instance using `scout.create()` (roughly similar to a Java class name). To make the object factory more robust, the separator between the type and the model variant (e.g. defined by `@ModelVariant` annotation in Java) was change from `.` to `:`. The namespace separator remains `.`. This allows the following forms of object types:.

- `"StringField"`: name without namespace, i.e. a type in the default namespace (resolves to `scout.StringField`)
- `"myproject.StringField"`: namespace qualified name
- `"StringField:MyVariant"`: type with variant (resolves to `scout.MyVariantStringField`), can also be combined with a namespace

Migration: Check your *objectFactories.js* and *defaultValues.json* files (if you have any in your

project) for types with variant and convert the separator from `.` to `:`.

Changed behavior of scout.HtmlComponent() constructor function

The constructor function scout.HtmlComponent() no longer links the $comp to the new instance. Instead, the static function scout.HtmlComponent.install() should be used to create a new HtmlComponent and link it to $comp. The constructor function should never be used anymore in custom code. (If you do, you will get errors.)

The new static method makes it clearer that it will alter the state of $comp. For a normal constructor, such behavior is unexpected and thus discouraged.

Migration: Check all **.js* files in your project for occurences of `new scout.HtmlComponent` and replace them with `scout.HtmlComponent.install`.

```
// Old, do not use anymore!
this.$container = $parent.appendDiv('my-widget');
new scout.HtmlComponent(this.$container, this.session);

// New, change your code to this (no change in first line):
this.$container = $parent.appendDiv('my-widget');
scout.HtmlComponent.install(this.$container, this.session);
```

Popup: Renamed option "installFocusContext" to "withFocusContext"

The initialization option `installFocusContext` for *Popup.js* instances was renamed to `withFocusContext` to match the corresponding property name.

Migration: Check if your project explicitly sets `installFocusContext = false` in popup widget instance (created via `scout.create('scout.Popup', { ⋯ })`) or in subclasses of `scout.Popup`. If it does, rename the option name to `withFocusContext`.

Other Changes

CSP report URL

By default, the `report-uri` for CSP violations is now called **/csp-report** (instead of /csp.cgi).

Reorganized *.html files due to strict CSP rules

The *.html files (index.html, login.html, logout.html etc.) have been changed to comply with the default Content Security Policy (CSP) rules.

The simplest way to migrate these files is to create them anew using the Scout SDK or maven archetype and compare them with your files. Otherwise, following this guide:

By default, inline `<script>` tags in HTML files are prohibited by CSP rules. Bootstrapping JavaScript code was therefore moved to dedicated **.js* files in the `WebContent/res` folder. Existing projects using CSP have to manually perform the following steps:

1. Open each **.html* file in `your.project.ui.html/src/main/resources/WebContent` folder and check if there are any inline script parts. Only `<script>` tags with embedded JavaScript code are considered "inline". Tags with a `src` attribute don't need to be changed.
2. Transfer the content of each script part to a **.js* file in the `res` subdirectory (e.g. *index.html* ⇒ *res/index.js*) and delete the now empty <script> part. Note that the content has changed as well, to initialize the application the new app object has to be used (scout.init → new scout.RemoteApp().init, scout.login.init → new scout.LoginApp().init, scout.logout.init → new scout.LogoutApp().init)).
3. Add a reference to the **.js* file in the `<head>` section using the `<scout:script>` tag, e.g.: `<scout:script src="res/index.js" />`
4. If the extracted **.js* file contains `<scout:message>` tags, they have to be moved back to the `<body>` of the corresponding **.html* file (because the NLS translation can only process HTML files). The attribute `style` has to be changed from `javascript` to `tag`.
5. Check the web.xml files of your *ui.html.app.** projects. If you use the scout login form and if you have listed the files to be excluded explicitly (instead of using */res/**), then you need to add the new **.js* files to the `filter-exclude` section as well.

Example:

Listing 1. login.html before migration (Scout 6.0)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Contacts Application</title>
    <scout:include template="head.html" />
    <scout:stylesheet src="res/scout-login-module.css" />
    <scout:script src="res/jquery-all-macro.js" />
    <scout:script src="res/scout-login-module.js" />
    <script> ①
      $(document).ready(function() {
        scout.login.init({texts: <scout:message style="javascript" key="ui.Login" key
="ui.LoginFailed" key="ui.User" key="ui.Password" /> });
      });
    </script>
  </head>
  <body>
    <scout:include template="no-script.html" />
  </body>
</html>
```

① Prohibited inline script.

Listing 2. login.html after migration (Scout 6.1)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Contacts Application</title>
    <scout:include template="head.html" />
    <scout:stylesheet src="res/scout-login-module.less" />
    <scout:script src="res/jquery-all-macro.js" />
    <scout:script src="res/scout-login-module.js" />
    <scout:script src="res/login.js" /> ①
  </head>
  <body>
    <scout:include template="no-script.html" />
    <scout:message style="tag" key="ui.Login" key="ui.LoginFailed" key="ui.User" key=
"ui.Password" /> ②
  </body>
</html>
```

① External script reference allowed by CSP.

② Moved from JavaScript call to `<body>`, changed style to `tag`.

Listing 3. res/login.js after migration (Scout 6.1)

```
$(document).ready(function() {
  new scout.LoginApp().init(); ①
});
```

① Translated texts are extracted automatically from DOM.

Renamed *.css files to *.less

Because the former **.css* files actually were LESS files, we've changed the wrong file extension from `.css` to `.less`. This allows editors with LESS support/validation to properly work with the LESS syntax and simplifies the usage of the LESS *@import* clause, since the `(less)` hint is not required anymore.

Steps required to migrate from an older Scout version to version 6.1:

- Rename all **.css* files in directory `/WebContent/res` to **.less*
- Chane the include syntax in **-macro.less* and **-module.less*:
  - Old: `//@include("scout-module.css")`
  - New: `@import "scout-module.less";`
- In each **.html* file in directory `/WebContent`, use **.less* extension in stylesheet tag:

- Old: `<scout:stylesheet src="res/myapp-all-macro.css" />`
- New: `<scout:stylesheet src="res/myapp-all-macro.less" />`

Importing regular `.css` files in module files (**-module.less*) is still supported, and required in some cases. Just make sure that all stylesheets using LESS do have a **.less* file extension. Macros and modules must always be LESS files.

UiHttpSessionListener replaced by HttpSessionMutex

The HttpSessionListener class `org.eclipse.scout.rt.ui.html.UiHttpSessionListener` has been replaced by the listener class `org.eclipse.scout.rt.server.commons.HttpSessionMutex`. Therefore if the class `UiHttpSessionListener` is registered in the `web.xml` file replace it with `org.eclipse.scout.rt.server.commons.HttpSessionMutex`.

Version check on startup

After a release upgrade, the cached resources (e.g. index.html, *.js, *.css) have most likely changed and must be re-downloaded from the server. Usually, this happens automatically, because the *index.html*'s ETag has changed and the server does not respond with `HTTP 304 Not Modified`. However, we have found that there are rare cases where browsers start the JS app without checking if *index.html* has to be updated (e.g. in Firefox when restoring tabs from a previous session or in Chrome when the "auto discard tab" feature has discarded the application's tab). This results in a mismatch between the UI and the UI server.

To fix potential problems caused by old resources, a version check is performed during application startup. To enable this version check in existing applications, include the tag `<scout:version>` in `index.html`. New Scout projects created using the *helloworld* archetype already include the tag.

The current version is determined by the value of the configuration property `scout.application.version`.

Example:

```
...
<body>
  ...
  <scout:version /> ①
</body>
...
```

① will be replaced by the application's version

Migration Guide to Scout 6.0

This document describes what needs to be done when migrating from Eclipse Scout 5.0 to Eclipse Scout 6.0. If you are updating to a 6.0 service release, see the chapter Service Release Migration.

Service Release Migration

Neon.3 (6.0.300)

- JAX-WS Pooled Port Provider (since 6.0.300)

Neon.2 (6.0.200)

None.

Neon.1 (6.0.100)

- Changed behavior for tables with autoResizeColumns = true (since 6.0.100)

Project Structure

With the upgrade to pure maven without OSGi the project structure should be changed to the maven default [1: https://maven.apache.org/guides/getting-started/]:

Listing 4. Eclipse Plugin Structure (Scout 5.0, old)

```
org.eclipsescout.helloworld.client
    pom.xml
    plugin.xml
    src
        org
            eclipsescout
                helloworld
                    ClientSession.java
                    Activator.java

org.eclipsescout.helloworld.test
    pom.xml
    plugin.xml
    src
        org
            eclipsescout
                helloworld
                    HelloworldTest.java
```

Listing 5. Maven Project Structure (Scout 6.0, new)

```
org.eclipsescout.helloworld.client
    pom.xml
    src
        main
            java
                org
                    eclipsescout
                        helloworld
                            ClientSession.java
        test
            java
                org
                    eclipsescout
                        helloworld
                            HelloworldTest.java
```

Manifest.MF

Manifest.MF is no longer used. Migrate dependencies to `pom.xml`!

AccessControlService

The IAccessControlService has been improved to allow for other keys than the userId. AbstractAccessControlService is now generic with key as a type parameter.

If you want to use access control based on the userid as before, extend UserIdAccessControlService and change the API of execLoadPermissions to

```
protected abstract PermissionCollection execLoadPermissions(String userId)
```

IShellService

The ShellService can no longer be used because there is no access to the client side shell. Instead you can use the following code to send a document to clients:

```
BinaryResource binaryResource = new BinaryResource(templateFile.getName(), myRawData);
ClientSessionProvider.currentSession().getDesktop()
  .openUri(binaryResource, OpenUriAction.DOWNLOAD);
```

Desktop

- Renamed `IDesktop.openUrlInBrowser` to `openUri`, since passed String is not always an URL but sometimes an URI like tel:123 or mailto:foo@bar.com, etc.
- Renamed `IDesktop.openDownloadInBrowser` to `downloadResource`, added overridden methods with `BinaryResource` parameter, so it's not required to create a `IDownloadHandler` instance to use the download methods.
- Renamed `IUrlTarget` to `ITargetWindows`, `UrlTarget` to `TargetWindow`
- Renamed `DesktopEvent.TYPE_OPEN_URL_IN_BROWSER` to `TYPE_OPEN_URI`
- Renamed `DesktopEvent.TYPE_OPEN_DOWNLOAD_IN_BROWSER` to `TYPE_DOWNLOAD_RESOURCE`
- F5 Keystroke on the desktop to reload the current page is no longer necessary because the table itself now provides the f5 keystroke → Remove the keystroke from your desktop to prevent refreshing the table twice.

Offline

Offline functionality in the scout client was removed (not needed anymore). Delete - OfflineState - IClientSession.getOfflineSubject - IDesktop.changeVisibilityAfterOfflineSwitch

Mobile

org.eclipse.scout.rt.client.mobile has been merged with org.eclipse.scout.rt.client. A lot of the mobile code in that plugin has been removed because the ui is now smarter and reacts bettter to smaller screens than the previous uis.

If your project contains a mobile plugin, it is suggested to merge it with your client plugin as well.

The new mobile approach is slightly different than the one before Neon:

- There is still a device transformer which transforms the client model into a mobile optimized model. But the transformation is a lot simpler than before and is more or less limited to the adjustment of some properties.
- The transformation mainly affects mobile phones (resp. devices with a small screen size). On tablets the application will look nearly the same as on a desktop device. There are just a few optimizations made regarding touch input (e.g. the smartfield looks different).
- Previously a form based approach had been used. Every outline page was wrapped in a page form, the navigation happened by showing or hiding the corerct form. Now, the outline tree has been enhanced with mobile specific functionality. This means the navigation happens completely in the outline tree, no forms are used. There is also no need for a home form anymore which displays the available outlines. The outlines may be switched in the same way as with the desktop style. The advantage is that the mobile style (or rather the style for small devices) looks and behaves similar to the regular desktop style which should make it easier for the user.

ToolButton

- `IToolButton` has been removed, it is not necessary anymore because the desktop may now display any kind of menus.
- `IFormToolButton` has been renamed to `IFormMenu`. These menus, which display a form when selected, may now be used on any menu capable component and not only on the desktop. Therefore it has been moved from the package `org.eclipse.scout.rt.client.ui.desktop.outline` to `org.eclipse.scout.rt.client.ui.form`.

Menu

All owners of an `IContextMenu` now share a common interface: `IContextMenuOwner`. This interface provides a method `getMenuByClass(T)`, analogous to `getFieldByClass(T)`, `getColumnByClass(T)` etc.

`ITree` and `ITable` provided a similar method `getMenu(T)`. This method was deprecated in favor of `getMenuByClass(T)`.

Usually, the migration is completed by simply renaming all calls to the old method. However, it should be noted that the old behavior is not exactly reproduced in a special case: When more than one implementation of the given class T was found, the old method just returned the first instance found. The new method throws an exception in this case, because the order of the instances is not really defined. If you really want to find *any* instance of the given class, retrieve the list of all instances using getMenus() and apply the filtering by yourself.

The constructors of OutlineMenuWrapper changed. For details consult the javadoc. This was needed to ensure the correct menuTypes throughout the wrapped menu's sub-hierarchy.

The `CopyColumnsWidthsMenu` has been deleted and was replaced with a new button in `OrganizeColumnsForm`.

Message Box

- Removed title. No replacement, title is not supported anymore.
- renamed intro text to header & info/actionText to body.
- using method chaining to construct message box
  - getHiddenText() → getHiddenText()
  - setHiddenText(hiddenText) → withHiddenText(hiddenText) and returning instance of IMessageBox
- Renamed startMessageBox to show
- Removed MessageBox(String title, String introText, String okButtonText) → MessageBoxes.create().withHeader(introText).withYesButtonText(okButtonText)
- Removed MessageBox(String title, String introText, String actionText, String yesButtonText, String noButtonText, String cancelButtonText) → MessageBoxes.create().withHeader(introText).withBody(actionText)

.withYesButtonText(yesButtonText).withNoButtonText(noButtonText).cancelButtonText(cancelButtonText);

- Removed MessageBox(String title, String introText, String actionText, String yesButtonText, String noButtonText, String cancelButtonText, String hiddenText, String iconId) → MessageBoxes.create().withHeader(introText) .withBody(actionText).withYesButtonText(yesButtonText).withNoButtonText(noButtonText).withCancelButtonText(cancelButtonText).withHiddenText(hiddenText).withIconId(iconId);
- Moved MessageBox.showDeleteConfirmationMessage methods to MessagesBoxes class
- If html needs to be displayed, use the new html(IHtmlContent) method. Header / body methods do not support html.

Table

Table API Changes

- Renamed `ITable.resetDisplayableColumns()` to `resetColumns()`
- Removed `ITable.resetColumns(boolean, boolean, boolean, boolean)` from interface (is now protected in `AbstractTable`)
- `AbstractTable.execResetTable(…)`: changed signature
  - old: `protected void execResetColumns(boolean visibility, boolean order, boolean sorting, boolean widths)`
  - new: `protected void execResetColumns(Set<String> options)`
- Changed signature of `ClientUIPreferences.getTableCustomizerData`
  - From `ClientUIPreferences.getTableCustomizerData(String customizerKey)` to `ClientUIPreferences.getTableCustomizerData(ITableCustomizer customizer, String configName)`
  - From `ClientUIPreferences.setTableCustomizerData(String customizerKey, Object customizerData)` to `ClientUIPreferences.setTableCustomizerData(ITableCustomizer customizer, String configName)`
- Replaced `ITableColumnFilterManager` by `TableUserFilterManager`

  Reason for the rename is because more filter types were added. There are currently 2 filter types: Column filter and text filter, there will be a chart filter in the future. Additionally, the filtering now happens in the UI. The ui sends the filtered rows to the ui server to update its table state so that getFilteredRows return the currently visible rows on the ui. This rowsFiltered event leads to a creation of `UserTableRowFilter` which contains the filtered rows. This is the only active filter on a table. The filters managed by `TableUserFilterManager` are actually only filter states and are not added to the table.

- `AbstractColumn.execPrepareEdit(ITableRow)` must not return `null` anymore use `Cell.setEditable(boolean)` instead.
- Added `ITable#rowIconVisible` to control whether the row icon is visible or not. If set to `true` the gui creates a column which contains the row icons. The column has a fixed width, is not

moveable and always the first column (resp. the second if the table is checkable). The column is not available in the model.

If you need other settings or if you need the icon at another column position, you cannot use the row icons. Instead you have to create a column and use `Cell#setIconId(String)` to set the icons on it's cells.

If you used `ITableRow#setIconId` or `AbstractTable#getConfiguredDefaultIconId` and still want the icons to be visible, you have to set `getConfiguredRowIconVisible` to `true`.

- Refactored editable behaviour of cells.
  - `Table.isCellEditable` only returns `cell.editable` and does not consider table or row enabled and visible states. Conforms to the behaviour of the other cell properties (text, cssStyle, etc).
  - `execIsEditable` has been removed. Use `cell.setEditable` (e.g. in `execDecorateCell`) if you want a cell to behave differently than the column.
  - `decorateCellInternal` does not write properties to the cell anymore, this is now done initially or if the column property changes. Advantage: It's now possible to modify the cell properties outside of `execDecorateCell`. Furthermore, there is no need to execute this code so many times.
  - Removed `ICell.setEnabled`. Did not have any effect, use `row.setEnabled` instead. Or `ICell.setEditable` if you would like to control editability of a cell.
- `InternalTableRow` / `AbstractTable`: checked state of a row is moved to the table. The `TYPE_ROWS_UPDATED` is no longer used to notify about rows checked. Instead there is an event `TYPE_ROWS_CHECKED` which is fired when rows are checked or unchecked. Also there is a new Method on the model which is executed when rows are checked (`execRowsChecked`). This method is also available in extensions.

  A row should be set to checked from the model even if the row is disabled. For this, the method `setRowsChecked` is extended with a new parameter to identify if only enabled rows should be checked or not. The ui should only check enabled rows, so the ui-facade calls the method with true.

- Removed `ITable#rowHeightHint` / `getConfiguredRowHeightHint`

  This property was added because with rap and swt it was not possible to have variable table rows as height as their content. The only possibility to get multiline rows was to set a fixed height. This limitation is now gone. If you still want every row to have a fixed height on multiline tables, you can use css to achieve it.

Custom Table Sorting

Added `IColumn.uiSortPossible`

Sorting of table data is done by the ui whenever possible. This has the advantage, that it is faster, that less data is transferred and that it works in offline mode. The drawback is that it is not possible in every case.

Example: If an invisible column has `alwaysSortAtBegin` set to `true`, the sorting is delegated to the model. Furthermore smart columns can not be sorted by the ui because the value is not known.

If you implemented custom sorting (e.g. by overriding `AbstractColumn.compareTableRows`), you have to set `getConfiguredUiSortPossible` to `false`.

Table Field & Page

- Removed "populate status" and "selection status" methods from `IPage` and `ITableField`. The only status is on the table itself and is called "table status". `IPage` and `ITableField` have new convenience methods for getting/setting the table status (without requiring null checks on getTable()).

  Migration:

  - Replace `IPage.setPagePopulateStatus()` by `IPage.setTableStatus()`.
  - Replace `IPage.getPagePopulateStatus()` by `IPage.getTableStatus()`.
  - Properties `PROP_TABLE_SELECTION_STATUS`, `PROP_TABLE_POPULATE_STATUS`, `PROP_TABLE_STATUS_VISIBLE` no longer exist on `ITableField`. If you need to listen to them, change your listener target the the field's `ITable`.
  - Method `ITableField.createDefaultTableStatus()` was dropped without replacement. "Selection status" is not supported by Html UI at the moment (selection is visualized on the UI only, not in the model).
  - `ITableField.get/setTableStatus()` convenience methods with Strings were dropped without replacement. Use `ITableField.getTableStatus().get/setMessage()` instead.
  - `ITableField.get/setTableSelectionStatus()` were dropped without replacement. "Selection status" is not supported by Html UI at the moment (selection is visualized on the UI only, not in the model).
  - Change `ITableField.get/setTablePopulateStatus()` to `ITableField.get/setTableStatus()`
  - `ITableField.updateTableStatus()` was dropped without replacement. Simply set the table status with `ITableField.setTableStatus()`.
  - `getConfiguredTableStatusVisible()` was dropped without replacement. Instead, the initial "table status visible" property should be set on the table. (In most cases, you can simply move the `getConfiguredTableStatusVisible()` method from the table field to the table.
- Removed `AbstractPageWithTable.getConfiguredShowTableRowMenus`. Replacement: none (no functionality was provided).
- Removed `AbstractPageWithTable.getConfiguredShowEmptySpaceMenus` Replacement: if return value was false, override `computeTableEmptySpaceMenus` and return an empty list instead.
- API of `IPageWithTable` and `IPageWithNodes` merged and moved duplicate methods to `IPage`
  - `IPage` now has a `T getTable()` method, also changed abstract classes implementing these interfaces. `IPage` now expects a type parameter for the table.
  - API `IPage`:
    - added `T getTable()`

        - added `boolean isDetailFormVisible()`
        - added `void setDetailFormVisible(boolean visible)`
        - added `ITreeNode getTreeNodeFor(ITableRow tableRow)`
        - added `IPage getPageFor(ITableRow tableRow)`
        - added `ITableRow getTableRowFor(ITreeNode treeNode)`
        - added `List<ITableRow> getTableRowsFor(Collection<? extends ITreeNode> treeNodes)`
    - API `IPageWithNodes`:
        - `getInternalTable()` replaced by `getTable`
        - moved to `IPage`: `ITreeNode getTreeNodeFor(ITableRow tableRow)`
        - moved to `IPage`: `ITableRow getTableRowFor(ITreeNode childPageNode)`
    - API `IPageWithTable`:
        - moved to `IPage`: `T getTable()`
        - moved to `IPage`: `ITreeNode getTreeNodeFor(ITableRow tableRow)`
        - moved to `IPage`: `ITableRow getTableRowFor(ITreeNode childPageNode)`
        - moved to `IPage`: `List<ITableRow> getTableRowsFor(Collection<? extends ITreeNode> childPageNodes)`
- Improved page detail form handling: The detail form is now created and started when the page gets activated and closed when the page gets disposed, similar to the search form. API added `getConfiguredDetailForm`, `execInitDetailForm`, `createDetailForm`, `startDetailForm`.

  Remove the detail form handling code from `execPageActivated` / `execPageDeactivated` / `execPageDisposed` and use either `getConfiguredDetailForm` / `execInitDetailForm` or `createDetailForm`.

Changed behavior for tables with autoResizeColumns = true (since 6.0.100)

Before 6.0.1, the column width was used as weight for the calculation of the real column width. To make sure the columns don't get too small on small screens, this width is now also used as minimum / preferred width. It is not a hard minimum, the user can still make the column smaller.

So if you have tables with autoResizeColumns set to true, check the widths of the columns and adjust them if needed. The easiest way to do this is to make the screen smaller until a horizontal scrollbar appears. Then adjust the values if the column is too small and make sure the content is readable most of the time. But don't make the columns too big because you want to avoid horizontal scrollbar on large screens.

Outline

- Removed `IOutlineTableForm`, `IOutlineTreeForm` and all sub-classes. They're not supported by the new Html UI anymore.

Default Page selection of Outlines

For an Outline having a selected page is not mandatory anymore. An outline overview or the default detail form will be displayed if no page is selected. Therefore activating an outline does not automatically select the first page anymore.

If the previous behavior is still wanted, one can implemented `IDesktop.execOutlineChanged` and call `activateFirstPage` if active page is `null`.

Wizard

- Argument `containerForm` was removed. Use `getContainerForm()` instead.
- Method `decorateWizardContainerForm` was renamed to `execDecorateContainerForm` (same as `execCreateContainerForm`).

  Old code (MyWizard extends AbstractWizard):

```
@Override
protected IWizardContainerForm execCreateContainerForm() {
  MyWizardContainerForm containerForm = new MyWizardContainerForm(this);
  decorateWizardContainerForm(containerForm);
  // more custom modifications
  return containerForm;
}
```

New code:

```
@Override
protected IWizardContainerForm execCreateContainerForm() {
  return new MyWizardContainerForm(this);
}

@Override
protected void execDecorateContainerForm() {
  getContainerForm().setXyz(...);
  // more custom modifications
}
```

- Some properties were removed from IWizard:
  - `displayHint`, `displayViewId`, `modal` -→ no replacement. Set them on the wizard container form. If the wizard container form does not provide the correct value, the wizard may change them in `execDecorateContainerForm()`.
  - `iconId`, `tooltipText`, `wizardNo` -→ no replacement (legacy properties, never used).
  - `titleHtml`: use `subTitle` instead.

- `getWizard[…]Button()` methods in `IWizardContainerForm` no longer return `IButton`, but `IWizardAction`. This change allows returning menus instead of buttons. `IWizardAction` serves as a common interface for `IButton` and `IAction` and provides some methods that are commonly used for the wizard buttons (e.g. `setVisible`, `setEnabled`). Because `IAction` calls its label "text", those menus have to override getLabel/setLabel and delegate the calls to the corresponding "text" methods. Alternatively, the class `AbstractWizardMenu` may be used instead of `AbstractMenu`.
  - For own implementations of `IWizardContainerForm`, replace the return value `IButton` by `IWizardAction`.
  - For code that previously used the `setView(boolean, boolean, boolean)` method on wizard buttons, a new `setView(boolean, boolean)` method was introduced on `IButton` and `IAction` (because it does not make sense to make a button "mandatory"). This can be migrated by just deleting the third argument.

Form

- `get/setBasicTitle` removed
- `get/setSubTitle` added
- `PROP_SUB_TITLE` added
- `composeTitle` removed.
- Added default behaviour to `AbstractForm.execCreateFormData` The method now creates a new instance of the form data based on the form data annotation. Also added createFormData to the IForm interface. If `execCreateFormData` was implemented and just used the default constructor of the corresponding form data class, the method may be removed.
- Removed display-hint `IForm.DISPLAY_HINT_POPUP_DIALOG`. Not supported anymore. Use dialog or popup-window instead.

Form Fields

- Deleted `AbstractCheckBox`, `AbstractCheckboxExtension`, `ICheckBoxExtension`, `ICheckBox`.

  Use `AbstractBooleanField`, `AbstractBooleanExtension`, `IBooleanExtension`, `IBooleanField` instead.
- Renamed package `imagebox` to `imagefield` due to consistency reason.
- Deprecated `getConfiguredAutoDisplayText` in `AbstractValueField`. The display text is always updated automatically.
- Removed `AbstractDoubleField` and `AbstractDoubleColumn`. Use `AbstractBigDecimalField` and `AbstractBigDecimalColumn` instead. See Bug 464770.
- Renamed package `org.eclipse.scout.rt.client.ui.form.fields.colorpicker` to `.colorfield`.
- Removed `ContributedKeyStroke` method from all FormField classes because these are only the menus which are added on the field. Use `getMenus()` instead.
- All `AbstractExtensible*` Scout elements have been deleted. Use the normal element instead (e.g. use `AbstractStringField` instead of `AbstractExtensibleStringField`). For extension support use

the corresponding extension object (e.g. `AbstractStringFieldExtension`).

- Deleted `IMailField`, `AbstractMailField` and all associated classes and files.

  An application that requires a facility to compose an e-mail should create a form with the fields required for that application (Multiline-StringField for plain-text E-Mails, RichTextField for HTML e-mails, FileChooserField for file uploads, etc.)

- Deleted ICustomField/AbstractCustomField and all associated classes and files.
- Deleted IDocumentField/AbstractDocumentField and all associated classes and files.
- `getConfiguredTreat0AsNull` in Smartfield has been deleted. (see also Bugzilla 469902).
- Changed return value of `IGroupBox.getConfiguredScrollable` to `TriState`. Mainbox is now scrollable by default.

  Migration:

  - You can remove getConfiguredScrollable() from your mainboxes
  - If you want another groupbox to be scrollable, you have to set the groupbox to scrollable while setting the mainbox to scrollable = `false`.

- Removed `IToolButton` from forms. Therefore `IToolButtons` can not be added anymore as an extension to forms. Instead `IToolButton` can be defined inside the MainBox of a form. (`IToolButton` now is an `IMenu` and adding menus to GroupBoxes as extension is also supported.)
- Simplified form tool buttons: Refactored API to be consistent with detail and search form handling of a page. Remove the form handling code from `execStartForm` and use either `getConfiguredForm` / `execInitForm` or `createForm`.
- The search table control now gets selected if the search is required. If you had a `SearchFormToolButton`, remove the code in `Desktop.execPageSearchFormChanged`.
- When setting an inner form into an `WrappedFormField` using `setInnerForm(IForm)` the given form life cycle is handled by the wrapped form field. This means it is automatically started, disposed etc.

Validate on any Key

Replace ValidateOnAnyKey mechanism (`getConfiguredValidateOnAnyKey`) (Bug 459893):

- removed:
  - `IBasicField.setValidateOnAnyKey(boolean)`
  - `IBasicField.isValidateOnAnyKey()`
  - `IBasicField.PROP_VALIDATE_ON_ANY_KEY`
- use new updateDisplayTextOnModify-mechanism instead:
  - `IBasicField.setUpdateDisplayTextOnModify(boolean)`
  - `IBasicField.isUpdateDisplayTextOnModify(boolean)`
  - `AbstractBasicField.execChangedDisplayText()`
  - `IBasicField.PROP_UPDATE_DISPLAY_TEXT_ON_MODIFY`

- `IBasicFieldUIFacade` renamed and changed method:
  - from: `boolean setTextFromUI(String newText, boolean whileTyping)`
  - to: `void setValueFromUI(String value)`
- removed `IColorFieldUiFacade`

String Field

- Deleted `AbstractTextField`, `AbstractTextFieldExtension`, `ITextFieldExtension`, ITextField`.

  Use `AbstractStringField`, `AbstractStringExtension`, `IStringExtension`, `IStringField` instead.
- Method renaming: `getConfiguredDecorationLink()` → `getConfiguredHasAction()`.
- Method renaming: `isDecorationLink()` → `isHasAction()`.
- Method renaming: `setDecorationLink(boolean)` → `setHasAction(boolean)`.
- Method renaming/signature change: `execLinkAction(java.net.URL)` → `execAction()`. `execAction()` can access value using `getValue()`, it could create the old URL using `org.eclipse.scout.commons.IOUtility.urlTextToUrl(getValue())`.
- Removed `IStringField.isSpellCheckAsYouTypeEnabled()`, `IStringField.setSpellCheckAsYouTypeEnabled(boolean)` added `IStringField.setSpellCheckEnabled(boolean)` new ui delegates spell checker to browser, new property can be used to enable/disable spell checker for certain fields (by default it is enabled for multi-line fields, see `AbstractStringField.computeSpellCheckEnabled()`).
- Removed `IStringField.isSelectAllOnFocus()`, `IStringField.setSelectAllOnFocus(boolean)`, `IStringColumn.isSelectAllOnEdit()`, `IStringColumn.setSelectAllOnEdit(boolean)`.

Button

- The default of `getConfiguredGridUseUiWidth` was changed from true to false. This was done so that buttons are aligned with other fields by default. This only affects the grid cell, the button itself is still as width as it used to be because of `fillHorizontal = false`.

Browser Field

- `IBrowserField` is no longer a value field. The `RemoteFile` value was changed to a property of type `BinaryResource`.
  - Instead of `setValue()/getValue()` use `setBinaryResource()/getBinaryResource()`.
  - Instead of `execChangedValue()` use a `BrowserFieldListener`.
  - If you relied on the browser field to be "save needed" when setting the value (RemoteFile), you have to call `touch()` manually, because the browser field will never report "save needed" by itself (because it has no value).
- removed `AbstractBrowserField.execAcceptLocationChange`, `AbstractBrowserField.execLocationChanged`, `AbstractBrowserField.doLocationChange`. Use `AbstractBrowserField.execPostMessage` as replacement.

- Refactored `execHyperlinkAction`. With the new html ui real hyperlinks are handled by the browser. Other links (formerly local links) are now called app links. The new method `execAppLinkAction` is only called for app links, hence the parameters url and local are not necessary anymore.
  - Removed parameter url and local and renamed path to ref.
  - Renamed to `execAppLinkAction`

Date Field

- Removed the members `m_autoDate` and `m_autoTimeMillis` from `AbstractDateField`. They were replaced by a single property `PROP_AUTO_DATE` of type `java.util.Date`.

  Replace `getConfiguredAutoTimeMillis()/setAutoTimeMillis()/getAutoTimeMillis()` by `getConfiguredAutoDate()/setAutoDate()/getAutoDate()`. If both a date and time part should be set, combine them in the same `java.util.Date` argument. The methods `DateUtility.createDateTime()` and `DateUtility.convertDoubleTimeToDate()` may be useful.

- The UI facade of `AbstractDateField` was changed. To support offline, more responsive date/time validation on the new Html UI, formatting and parsing has to be performed on the UI layer, not on the model layer (otherwise, the UI would have to wait for the model on every key press).

  The date field UI facade was changed in the following way: Instead of sending a text to the model and validating/parsing it there, a already valid (from the parsing perspective) date is sent to the model. The model may then still validate it (e.g. check ranges), but the parsing is done entirely on the UI. As a consequence, not all date format patterns defined in SimpleDateFormat are supported anymore, only the most commonly used. By default, the date field uses locale-dependent patterns that are supported by the UI, see `getDefaultDateFormatPatternByLocale()` and `getDefaultTimeFormatPatternByLocale()`. Both the date and the time part of a date field have a separate pattern, because they are rendered in two separate fields on the UI.

  The method `AbstractDateField.execParseValue()` is no longer supported. It cannot be removed entirely, because it is defined on `AbstractValueField`, but is marked as deprecated and final to make it clear that it is never called. If any subclass had overridden this method, it should be deleted. The code cannot be migrated, because it is now performed in the UI only.

- Removed methods not used by the HTML UI: + remove unused execShiftDate, execShiftTime from AbstractDateField + remove unused adjustDate, adjustTime from IDateField + removed fireDateShiftActionFromUI, fireTimeShiftActionFromUI from IDateFieldUIFacade

HTML Field

- For `IHtmlField` attachments `RemoteFile` has been replaced by `BinaryResource`, therefore method signatures of `getAttachments()` and `setAttachments` have changed.
  - Replace `RemoteFile` with `BinaryResource`.
  - Attachments must be used within the `IHtmlField's value as `src="binaryResource:test.png"` (instead of `src="test.png"`). Append `binaryResource:` prefix where attachments are used.

    - New feature: Icons can be used without adding them as attachment using `src="iconid:ApplicationLogo"`.
    - New property for selection tracking, changeable with methods `isSelectionTrackingEnabled()` and `setSelectionTrackingEnabled(boolean)`. Selection tracking with `getSelectionStart()` and `getSelectionEnd()` is only possible when selection tracking is enabled.
- Removed html editor support on html field. If you used a html editor you can create a custom field an include an existing html editor.

Tree, TreeField & TreeBox

- If all child nodes of a node in a tree are deleted, a `TreeEvent` with the new type `ALL_CHILD_NODES_DELETED` is fired (instead of `NODES_DELETED`). This is useful for optimization.

  If you previously added a listener for the type `NODES_DELETED`, you have to check if your implementation needs to listen to the new `ALL_CHILD_NODES_DELETED` as well.

- `AbstractTreeNode` / `AbstractTree` / `AbstractTreeBox`: checked state of a row is moved to the tree. The `TYPE_NODE_UPDATED` is no longer used to notify about node checked. Instead there is an event `TYPE_NODES_CHECKED` which is fired when nodes are checked or unchecked. Also there is a new Method on the model which is executed when nodes are checked (`execNodessChecked`). This method is also available in extensions.

  Also the implementation to check child nodes of a tree when a parent is checked is moved from the `AbstractTreeBox` to the tree. But the configuration can be done on the `AbstractTreeBox`. A node should be set to checked from the model even if the node is disabled. For this, the method `setNodesChecked` is extended with a new param to identify if only enabled nodes should be checked or not. The ui should only check enabled nodes, so the ui-facade calls the method with true.

Calendar, CalendarField, Planner

- Moved display-mode constants from `ICalender` and `IPlanner` to separate interface classes and let `IPlannerDisplayMode` extend `ICalenderDisplayMode` because they share some constants.
- Removed `get/setColor()` from `ICalenderItem`, replaced with `get/setCssClass()`.
- Removed `decorateCell/-Internal` method from `AbstractCalendarItemProvider`
- Moved `get/setExternalKey()` from `ICalendarAppointment` to base class `ICalenderItem`.
- Removed cell instance from `CalendarComponent`

Utilities

- Removed methods `UserAgentUtility.isRichClient()` and `.isWebClient()`.
- `HTMLUtility` has been deprecated. There is no replacement.
- `NumberUtility.sum(double…)` -→ use `sum(Number…)`

- `NumberUtility.sum(long…)` -→ use `sum(Number…)`
- Removed `NumberUtility.avg(double…)`
- Removed `NumberUtility.divide(double, double)`
  - `NlsUtility.getDefaultLocale()` has been removed -→ use `NlsLocale.CURRENT`
- The following Classes have been moved. Organize imports to fix errors:
  - `IDNDSupport`
  - `TransferObject`
  - `TextTransferObject`
  - `ResourceListTransferObject`
  - `JavaTransferObject`
  - `ImageTransferObject`
  - All Classes that once existed in `org.eclipse.scout.commons.*`. Most of them have been moved to `org.eclipse.scout.rt.platform.*`.
- Renamed `FileListTransferObject` to `ResourceListTransferObject`
- Removed `isText()`, `isFileList()`, `isImage()`, `isLocalObject()` from `TransferObject`. Replacement: `instanceof` check for the appropriate subclasses of `TransferObject`.
- Removed `TextTransferObject(String plainText, String htmlText)` and `TextTransferObject.getHtmlText()`. See Bug 465797.
- Moved `MultiClientSessionCookieStore` to `org.eclipse.scout.rt.servicetunnel` and renamed it to `MultiSessionCookieStore`. It can now be used in client and server environments.

  To make the service tunnel work with multiple sessions over HTTP, the `MultiSessionCookieStore` has to be installed. This is not done automatically, because the cookie manager is global for the entire JVM. Overriding this global variable may break things in a JEE environment with multiple applications or a pre-installed custom cookie manager. There are two options to install Scout's `MultiSessionCookieStore`:

  - Set the default cookie manager programmatically somewhere in your code. This is the way provided by the JVM, see http://docs.oracle.com/javase/tutorial/networking/cookies/cookiemanager.html for details.
  - Use Scout's auto-install mechanism by setting the property `org.eclipse.scout.rt.servicetunnel.multiSessionCookieStoreEnabled` in your `config.properties` to `true`. This is the recommended way.

Cryptography

`EncryptionUtility`, `PublicKeyUtility`, `TripleDES` have been deprecated because these classes use insecure cryptography. Use the new `SecurityUtility` or the Java Cryptography Architecture instead [2: http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html].

When changing the cryptography algorithms in you application please keep in mind that all existing encrypted, hashed or signed data becomes invalid! Consider migrating these data first.

Various API Changes

- Changed `ILookupRow` to fluent API: use with... instead of set...
- `IClientSession.stopSession()` was renamed to `stop()` to match `IServerSession.stop()`.
- Deleted validation rule infrastructure: Deleted package org.eclipse.scout.rt.shared.validate with all subpackages and the containing classes. Furthermore the class org.eclipse.scout.rt.shared.data.form.ValidationRule has been deleted.
- `UiLayer`: Removed values `JSP`, `JSF`, `RAP`, `SWING` and added value `HTML`.
- `UserAgentUtility`: API removed `isRapUi()`, `isSwingUi()`
- The unused, obsolete classes `org.eclipse.scout.rt.client.ui.form.fields.ValueFieldEvent` and `org.eclipse.scout.rt.client.ui.form.fields.ValueFieldListener` were removed.

Logging API

Scout switched from a custom, typically java.util.logging-based logger implementation to SLF4j. The log format does not support indexed placeholders anymore.

The regular expression pattern `\{\d+\}` finds potential occurrences. Replace those within log formats with `{}`. See SLF4j MessageFormatter.

Listing 6. Placeholders in log format

```
LOG.info("message {}", obj); // this worked before and still works. No action required

LOG.info("message {0}", obj); // the index is not supported anymore. You have to
remove it (see previous statement)
```

Indexed placeholders are actually deprecated since Scout's open-source debut. The values were filled in from left to right, independent of the possibly declared index.

Logging configuration

migrate logging.properties to logback.xml

1) in logging.properties apply the following regex replacements:

```
search: ^(\w.*)\.level\s*=\s*(ALL|OFF|SEVERE|WARNING|INFO|FINE|FINEST)\s*$
replace: <logger name="$1" level="$2"/>

search: ^(\w.*)\.useParentHandlers\s*=\s*(false)\s*$
replace: <logger name="$1"><appender-ref ref="CONSOLE"/></logger>

search: ^#+\s*(.*)$
replace: <!-- $1 -->

search: (FINEST|finest)
replace: TRACE

search: (FINE|fine)
replace: DEBUG

search: (WARNING|warning)
replace: WARN

search: (SEVERE|severe)
replace: ERROR
```

2) create a new logback.xml as

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>

  <appender name="CONSOLE"
    class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg %n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="CONSOLE" />
  </root>
  <!-- (3) -->

</configuration>
```

3) include the converted content of logging.properties at 1.

4) adjust the format pattern if needed

available variables are

```
%d{HH:mm:ss.SSS}
%thread
%-5level
%logger{36}
%msg
%n
%X{scout.ui.session.id}
%X{scout.session.id}
%X{http.request.method}
%X{http.request.uri}
%X{http.session.id}
%X{scout.user.name}
%X{subject.principal.name}
```

Default ui.html pattern

```
<pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg [%X{subject.principal.name} @
%X{http.request.method} %X{http.request.uri} %X{scout.ui.session.id}]%n</pattern>
```

Default server pattern

```
<pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg [%X{subject.principal.name} @
%X{http.session.id} in %thread ]%n</pattern>
```

Text cleanup

All unused texts in ScoutTextProviderService were removed. If you were using one of the deleted ones, you find them in:

- ScoutTexts_bg.properties
- ScoutTexts_cs.properties
- ScoutTexts_da.properties
- ScoutTexts_de_DE.properties
- ScoutTexts_de.properties
- ScoutTexts_el.properties
- ScoutTexts_es.properties
- ScoutTexts_fi.properties
- ScoutTexts_fr_BE.properties
- ScoutTexts_fr.properties
- ScoutTexts_hr.properties
- ScoutTexts_hu.properties

- ScoutTexts_it.properties
- ScoutTexts_ja.properties
- ScoutTexts_nl_BE.properties
- ScoutTexts_nl.properties
- ScoutTexts_no.properties
- ScoutTexts_pl.properties
- ScoutTexts_pt_br.properties
- ScoutTexts_ru.properties
- ScoutTexts_se.properties
- ScoutTexts_sk.properties
- ScoutTexts_sl.properties
- ScoutTexts_sr.properties
- ScoutTexts_tr.properties
- ScoutTexts_zh_TW.properties
- ScoutTexts_zh.properties
- ScoutTexts.properties

Migrate to the new Job API

Eclipse jobs are replaced by Scout Job Manager API.

In a nutshell

Scout provides a job manager based on Java Executors framework to run tasks in parallel, and on Quartz Trigger API to support for schedule plans. A task (aka job) can be scheduled to commence execution either immediately upon being scheduled, or delayed some time in the future. A job can be single executing, or recurring based on some schedule plan.

A job is defined as some work to be executed asynchronously and is associated with a `JobInput` to describe how to run that work. The work is given to the job manager in the form of a `Runnable` or `Callable`. The only difference is, that a `Runnable` represents a 'fire-and-forget' action, meaning that the submitter of the job does not expect the job to return a result. On the other hand, a `Callable` returns the computation's result, which the submitter can await for. Of course, a runnable's completion can also be waited for.

See Scout architecture documentation for more information.

Static accessors

- ServerJob.getCurrentSession() → ServerSessionProvider.currentSession()
- ClientJob.getCurrentSession() → ClientSessionProvider.currentSession()

- ServerJob.isCurrentJobCancelled() → RunMonitor.CURRENT.get().isCancelled()

Raw Eclipse Job

Listing 7. before Scout 'N' release (<=5.0.x)

```
new Job("job-name") {

  @Override
  protected IStatus run(IProgressMonitor monitor) {
    // do something
  }
}.schedule();
```

Listing 8. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withName("job-name"));
```

ServerJob

Listing 9. before Scout 'N' release (<=5.0.x)

```
new ServerJob("job-name", ServerJob.getCurrentSession()) {

@Override
protected IStatus runTransaction(IProgressMonitor monitor) throws Exception {
  // do something
  return Status.OK_STATUS;
}
}.schedule();
```

Listing 10. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withRunContext(ServerRunContexts.copyCurrent())
    .withName("job-name"));
```

ServerJob.runNow(...)

Listing 11. before Scout 'N' release (<=5.0.x)

```
new ServerJob("job-name", ServerJob.getCurrentSession()) {

@Override
protected IStatus runTransaction(IProgressMonitor monitor) throws Exception {
  // do something
  return Status.OK_STATUS;
}
}.runNow(new NullProgressMonitor());
```

Listing 12. since Scout 'N' release (>=5.1.x)

```
ServerRunContexts.copyCurrent().run(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
});
```

ServerJob with other Subject

Listing 13. before Scout 'N' release (<=5.0.x)

```
new ServerJob("job-name", ServerJob.getCurrentSession(), subject) {

@Override
protected IStatus runTransaction(IProgressMonitor monitor) throws Exception {
  // do something
  return Status.OK_STATUS;
}
}.schedule();
```

Listing 14. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withName("job-name")
    .withRunContext(ServerRunContexts.copyCurrent()
        .withSubject(subject)));
```

ClientSyncJob

Listing 15. before Scout 'N' release (<=5.0.x)

```
new ClientSyncJob("job-name", ClientSessionProvider.currentSession()) {

  @Override
  protected void runVoid(IProgressMonitor monitor) throws Throwable {
    // do something
  }
}.schedule();
```

Listing 16. since Scout 'N' release (>=5.1.x)

```
ModelJobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, ModelJobs
    .newInput(ClientRunContexts.copyCurrent())
    .withName("job-name"));
```

ClientAsyncJob

Listing 17. before Scout 'N' release (<=5.0.x)

```
new ClientAsyncJob("job-name", ClientSessionProvider.currentSession()) {

  @Override
  protected void runVoid(IProgressMonitor monitor) throws Throwable {
    // do something
  }
}.schedule();
```

Listing 18. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withRunContext(ClientRunContexts.copyCurrent())
    .withName("job-name"));
```

Delayed execution

Listing 19. before Scout 'N' release (<=5.0.x)

```
new Job("job-name") {

@Override
protected IStatus run(IProgressMonitor monitor) {
  // do something
}
}.schedule(5_000);
```

Listing 20. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withName("job-name")
    .withExecutionTrigger(Jobs.newExecutionTrigger()
        .withStartIn(5, TimeUnit.SECONDS)));
```

Repeatedly execution with a fixed delay

Listing 21. before Scout 'N' release (<=5.0.x)

```
new Job("job-name") {

@Override
protected IStatus run(IProgressMonitor monitor) {
  // do something
  schedule(5_000);
}
}.schedule(5_000);
```

Listing 22. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withName("job-name")
    .withExecutionTrigger(Jobs.newExecutionTrigger()
        .withSchedule(FixedDelayScheduleBuilder.repeatForever(5, TimeUnit.SECONDS))
        .withStartIn(5, TimeUnit.SECONDS)));
```

Check for cancellation

Listing 23. before Scout 'N' release (<=5.0.x)

```
new Job("job-name") {

@Override
protected IStatus run(IProgressMonitor monitor) {
  // do first chunk of work
  if (monitor.isCanceled()) {
      return Status.CANCEL_STATUS;
  }
  // do second chunk of work
  if (monitor.isCanceled()) {
    return Status.CANCEL_STATUS;
  }
  // do third chunk of work
  return Status.OK_STATUS;
}
}.schedule();
```

Listing 24. since Scout 'N' release (>=5.1.x)

```
Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do first chunk of work
    if (RunMonitor.CURRENT.get().isCancelled()) {
      return;
    }
    // do second chunk of work
    if (RunMonitor.CURRENT.get().isCancelled()) {
      return;
    }
    // do third chunk of work
  }
}, Jobs.newInput()
    .withName("job-name"));
```

Join job

Listing 25. before Scout 'N' release (<=5.0.x)

```
Job job = new Job("job-name") {

  @Override
  protected IStatus run(IProgressMonitor monitor) {
    // do something
    return Status.OK_STATUS;
  }
};
job.schedule();
job.join();
```

Listing 26. since Scout 'N' release (>=5.1.x)

```
IFuture<Void> future = Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withName("job-name"));

future.awaitDone();
```

Join job with a maximal wait time

Listing 27. before Scout 'N' release (<=5.0.x)

```
Job job = new Job("job-name") {

  @Override
  protected IStatus run(IProgressMonitor monitor) {
    // do something
    return Status.OK_STATUS;
  }
};
job.schedule();
job.join(5_000, new NullProgressMonitor());
```

Listing 28. since Scout 'N' release (>=5.1.x)

```
IFuture<Void> future = Jobs.schedule(new IRunnable() {

  @Override
  public void run() throws Exception {
    // do something
  }
}, Jobs.newInput()
    .withName("job-name"));

future.awaitDone(5, TimeUnit.SECONDS);
```

Join job and get the job's computation result

Listing 29. before Scout 'N' release (<=5.0.x)

```
final AtomicReference<String> result = new AtomicReference<>();

Job job = new Job("job-name") {

  @Override
  protected IStatus run(IProgressMonitor monitor) {
    // do something
    result.set("abc");
    return Status.OK_STATUS;
  }
};
job.schedule();
job.join();
System.out.println(result);
```

Listing 30. since Scout 'N' release (>=5.1.x)

```
IFuture<String> future = Jobs.schedule(new Callable<String>() {

  @Override
  public String call() throws Exception {
    // do something
    return "result";
  }
}, Jobs.newInput()
    .withName("job-name"));

String result = future.awaitDoneAndGet();
System.out.println(result);
```

Session Cookie Configuration

The Scout HTML UI Session cookie requires some security flags. Please refer to the Scout documentation chapter "Session Cookie (JSESSIONID Cookie) Configuration" to learn how to configure your JSESSIONID cookie.

Client Notifications

Check out the docs for a description about client notifications.

Changes in a nutshell

- There is only one poller per client (instead of per session): `ClientNotificationPoller`
- Long polling is used instead of polling in regular intervals
- Client notifications are plain serializable objects and do not need to implement the interface `IClientNotification` anymore
- `ClientNotificationRegistry` is used to register client notifications instead of `IClientNotificationService`
- If a notification needs to be handled temporarily, `AbstractObservableNotificationHandler` can be used to register a listener
- If a notification needs to be handled always, a handler can be created as subtypes of `INotificationHandler<T extends Serializable>` to always handle messages of type `T` `instead of creating a `IClientNotificationConsumerListener`
- The method `coalesce` on the client notification is replaced with a class of type `ICoalescer<T>`
- ServiceTunnel is now a bean instead of a member of the client session

Publishing Notifications

Listing 31. before Scout 'N' release (<=5.0.x)

```
SERVICES.getService(IClientNotificationService.class)
    .putNotification(new UserChangedClientNotification(userId), new
UserKeyClientNotificationFilter(userId, 60000L));
```

Listing 32. since Scout 'N' release (>=5.1.x)

```
String userId = "testUser";
BEANS.get(ClientNotificationRegistry.class)
    .putForUser(userId, new UserChangedClientNotification(userId));
```

Handling Notifications

Listing 33. before Scout 'N' release (<=5.0.x)

```
IClientNotificationConsumerListener m_userChangedNotificationListener = new
IClientNotificationConsumerListener() {
@Override
public void handleEvent(ClientNotificationConsumerEvent event, boolean sync) {
  if (event.getClientNotification() instanceof UserChangedClientNotification) {
    //handle ...
  }
}
};
SERVICES.getService(IClientNotificationConsumerService.class)
    .addClientNotificationConsumerListener(AbstractCoreClientSession.get(),
m_userChangedNotificationListener);
```

Listing 34. since Scout 'N' release (>=5.1.x)

```
class UserChangedClientNotificationHandler implements INotificationHandler
<UserChangedClientNotification> {

  @Override
  public void handleNotification(UserChangedClientNotification notification) {
    //handle ...
  }
}
```

JAX-WS Pooled Port Provider (since 6.0.300)

Creating web service and port instances are expensive operations (at least if the reference implementation Metro or the one bundled with the JRE is used). Especially parsing the WSDL and XSD files as well as building JAXB contexts and it is even worse if they are performed in parallel (due to synchronization).

The `PooledPortProvider` is the new default strategy for creating ports. Actually the pooled provider uses two pools, one for service instances and another for port instances (which are created by a service instance). A Scout transaction member keeps track of leased ports and puts them back into the pool when the Scout transaction releases its resources. Further, the transaction member ensures that the same port is used within a transaction, once it has been leased.

Port instances are reset when they are put back into the pool. Some JAX-WS implementations provide a suitable operation for resetting the port (i.e. Metro as well as the RI bundled with Java 8). Otherwise the request context is cleansed as good as possible. The corresponding `JaxWsImplementorSpecifics.resetRequestContext(Object)` can be extended to customize the cleansing.

The `AbstractWebServiceClient` does not distinguish between `PortProducer` and `PortCache` anymore. Both are `IPortProvider` strategies and the new `PooledPortPorvider` is just another one, frankly the new default. Setting the configuration property `jaxws.consumer.portPool.enabled` to `false` disables the pool and enables the previous behavior (bare `PortProducer`, wrapped by a `PortCache` instance that stashes ports).

The internal state of the pools is reported on the diagnostics servlet.

Migration

The following method has been renamed and the return type has been changed to `IPortProvider`. More important it returns a pooled, cached, bare provider or uses any other strategy. In other words, the `AbstractWebServiceClient` does not wrap the producer into a `PortCache` anymore.

```
class: org.eclipse.scout.rt.server.jaxws.consumer.AbstractWebServiceClient
old:   getConfiguredPortProducer(Class<SERVICE>, Class<PORT>, URL, String, String,
IPortInitializer)
new:   getConfiguredPortProvider(Class<SERVICE>, Class<PORT>, URL, String, String,
IPortInitializer)
                       ^^^^^^^^
```

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