Welcome to SMARTIT Eclipse Spring

Table of contents

1 Mission....2
2 What's New....2
3 How to Get Started....2
4 Project Status....2
5 Related Work by Others....3

Sponsors Welcome

Do you want to sustain SMARTIT Eclipse Spring development?
- Please contact us, specifying "*SMARTIT Eclipse Spring Sponsorship*" in the subject line of your e-mail.

1. Mission

SMARTIT aims at making Eclipse plug-in programming as easy as J2EE programming is when using the Spring framework.

2. What's New

- 2007-06-20 Installation via Eclipse update manager
- 2007-06-12 New SMARTIT Eclipse Spring version 0.2.1 (minor patch, update usually not required)
- 2007-05-15 New SMARTIT Eclipse Spring version 0.2.0!
- 2007-05-15 Added examples to documentation.

3. How to Get Started

Here is how to get your Eclipse ready for SMARTIT Eclipse Spring development:

1. Download and unpack the distribution file which contains all dependencies.
2. Import all ZIP archives in the `projects` subdirectory of the unpacked distribution archive as projects into your Eclipse workspace.
3. Import all ZIP archives in the `examples` subdirectory of the unpacked distribution archive as projects into your Eclipse workspace. (This step is not required.)
4. Create a new Eclipse plug-in project.
5. Add the following line to the `plugin.xml` of the new plug-in project:

```
<extension point="info.smartit.eclipse.spring.pluginContext"/>
```

6. Create a file named `pluginContext.xml` in your plug-in's root directory.

Add bean definitions to the file `pluginContext.xml` which you created in the last step. This file is processed whenever your plug-in is activated.

Please consult the How-To or the Reference for a description of how to access the beans in your plug-in context.

4. Project Status

Although currently at a "0.x.y" version ID, the core module

`info.smartit.eclipse.spring`) is already at a quite stable state and ready for use. However, we do not yet want to freeze the API and therefore stick to the "0." version ID.

The Eclipse UI bean enhancements which will enable UI component configuration from bean files (`info.smartit.eclipse.spring.ui`) are still in an experimental state and will only be available at a later time.

5. Related Work by Others

The following authors had notable influence on the SMARTIT Eclipse Spring implementation, although they probably don't know. As they inspired part of the SMARTIT Eclipse Spring development, we think they deserve to be mentioned in this context.

Ole Petter Aasen

Back in 2004, O. P. Aasen published a class `AlternateClassPathXmlApplicationContext` which is basically a tweaked `ClassPathXmlApplicationContext` with the ability to resolve classpath resources using the classloader of a configurable class.

The `ConfigurableClassPathXmlApplicationContext` which is used internally by SMARTIT Eclipse Spring uses an arbitrary classloader, which owes much to O. P. Aasen's approach. However, the tweaking is different.

Neil Bartlett

N. Bartlett published an Eclipse-Spring (mind the hyphen!) project on Sourceforge. Although SMARTIT Eclipse Spring "borrowed" many ideas from N. Bartlett's Eclipse-Spring - most notably the proxy idea for referencing beans from `plugin.xml`-, there are major and important differences:

- N. Bartletts Eclipse-Spring uses bean factories rather than application contexts. (The plug-in context in SMARTIT Eclipse Spring is a subclass of an application context.)
- Eclipse-Spring requires a special class in the plug-in for specifying the XML configuration file locations and providing an appropriate classloader. SMARTIT Eclipse Spring does not require such a class as it handles the classloader issue in a different way.
- Eclipse-Spring relies on that the Eclipse UI plug-ins are available, even in applications which do not make use of the Eclipse GUI. SMARTIT Eclipse Spring only requires the core Eclipse plug-ins.

Recently, there started an ongoing Spring OSGi discussion which is not related to SMARTIT Eclipse Spring.

There have been publications of further Spring usages in Eclipse plug-ins. However, the ones

which are known to us lack key features (for example, no dependency injection) and are therefore not mentioned here.