

## Errata and Addenda

for

### Professional Eclipse 3 for Java Developers

Last modifications: January 4, 2006

#### Page 7:

On the toolbar click the ~~Create a~~New Java Project icon, as shown in Figure 1.6.

#### Page 7:

(~~Create a~~New Java Class)

#### Seite 15:

The key shortcut Ctrl+Shift+F works even faster (~~Esc Strg~~+F under Linux).

#### Page 24:

Eclipse ~~shows the result in a pop-up window. You may insert it into the current scrapbook content by pressing Ctrl+Shift+D.~~ inserts the result or an error message directly behind the expression entered.

#### Page 34:

For example, if you position the cursor on the parameter of a method declaration and press Ctrl+1 (~~or click the green lieghtbulb that mysteriously appeared on the left margin of the editor~~), various functions ...

#### Page 45:

In the meantime version 1.0 of the VE was released. The operation of this tool resembles very closely that of VE version M1 (including the creation of SWT GUIs). New visual Java classes can be created via File > New > Visual Class. The kind of class can be determined in the following wizard page in the Style field. There you can choose between an SWT application (see chapter 8) and various Swing or AWT containers. SWT and Swing components cannot be integrated with the VE, in this case you have to resort to manual programming (see the *Widgets that Swing* section in

chapter 8). I recommend not to add the SWT-JAR manually to a project but to let the VE perform this task.

**Page 48:**

Detailed information about the implementation of Java Beans is found in the book *Java Beans 101* by Stearns.

**Page 83:**

The open the `JavaClipAudioPlayer` class in the `FreeTTS` package project,...

**Page 150:**

Under later Eclipse version you need to modify these paths accordingly.

A simpler way to add SWT functionality and optionally JFace functionality, too, is to use the Add Library button instead of the Add External JARs button. From the then appearing list, simple select Standard Widget Toolkit(SWT). Eclipse then adds all required JAR files to the build path.

**Page 191:**

In addition, the `SWT_AWT` class provides the `new_Shell()` method. This method creates an **equally sized** new SWT shell for a given AWT canvas, **so this canvas is presented in its own window but within the SWT application**. This shell may host other SWT GUI elements which then appear on top of the AWT canvas within an AWT context.

**Page 192:**

```
import java.awt.RenderingHints;
import java.util.ArrayList;
import java.util.Iterator;
```

**Page 219:**

This is donw with the help of `IPositionUpdater` instances. ... , all registered `IPositionUpdater` instances are invoiked ...

**Page 226:**

Input fields that you want to utilize this functionality must implement the interface `IContentAssistSubjectControl`.

**Page 318:**

In particular, you can obtain the location of the `Eclipse working workspace root` directory via the method `getLocation()`.

**Page 318:**

For example, the methods `savePluginPreferences()` and `getPluginPreferences()` store and retrieve the current preferences. With `find()` you can obtain the URL of a specified `workspace plug-in` resource, and with `openStream()` you can open an input stream on `such a specified workspace` resource.

**Page 335:**

This extension point allows defining specific user `activitiescapabilities` such as Java Development or Plug-in Development. Each `activitycapability` can correlate with a set of workbench plug-ins. Only if the end user activates an `activitycapability` the correlated plug-ins will become visible in the workbench. `ActivitiesCapabilities` may rely on other `activitiescapabilities`, so that if one `activitycapability` is activated, the other required `activitiescapabilities` are activated as well.

**Page 354:**

How can a component register with the selection service to notify it about selection events? To do this, the component `needs only to implement the interface registers an instance of type ISelectionProvider with implementing the methods addSelectionListener(), removeSelectionListener(), getSelection(), and setSelection() with the associated IWorkbenchSite`. When a component is activated, the workbench always checks automatically to see if `the component implements this interface a ISelectionProvider is registered with the site`.

**Page 410:**

First, you want to make sure that the functions for plug-in development are enabled in your Eclipse platform. [Go to Window > Configure Activities... and mark Plug-in Development.](#) [Invoke Window > Preferences > Workbench > Capabilities and mark Development.](#)

**Page 419:**

```
<attribute name="preferences" type="string">
  <annotation>
    <appInfo>
      <meta.attribute kind="java" basedOn=
        "com.bdaum.SpellChecker.preferences.SpellCheckerPreferences"/>
    </appInfo>
  </annotation>
</attribute>
```

**Page 504:**

<code>postRestore</code>	<a href="#">This method is invoked after a window has been restroed. It is called only for windows whose state is stored permanently.</a>
--------------------------	---

**Page 514:**

```
org.eclipse.help
org.eclipse.help.base
org.eclipse.help.ui
org.eclipse.help.webapp
org.eclipse.tomcat
```

**Page 516:**

... so I will skip this step in this chapter. However, the Rich Client Platform requires an explicit specification of the help system. Therefore you need to specify the following extension points: `org.eclipse.help.base.webapp` and `org.eclipse.ui.helpSupport`. To the later add the child element *config* specifying class `org.eclipse.help.ui.internal.DefaultHelpUI`. This

class provides the standard help services for the Eclipse workbench. Alternatively, you may discard the `DefaultHelpUI` and `Tomcat` and implement your own help GUI on the basis of the `AbstractHelpUI` class.

**Page 516:**

```
<requires>
<import plugin="org.eclipse.core.runtime"/>
<import plugin="org.eclipse.help"/>
<import plugin="org.eclipse.help.base"/>
<import plugin="org.eclipse.help.ui"/>
<import plugin="org.eclipse.help.webapp"/>
<import plugin="org.eclipse.tomcat"/>
<import plugin="org.eclipse.ui"/>
<import plugin="org.eclipse.ui.forms"/>
</requires>
```

**Page 517:**

```
<extension point="org.eclipse.help.base.webapp"/>
<extension point="org.eclipse.ui.helpSupport">
    <config class="org.eclipse.help.ui.internal.DefaultHelpUI"/>
</extension>
<extension id="product"
    point="org.eclipse.core.runtime.products">
    <product name="Hex Game Machine"
        application="com.bdaum.Hex.RcpApplication"
        description="The game of Hex">
        <property name="appName" value="Hex"/>
        <property name="windowImage" value="hexWindow.gif"/>
    </product>
</extension>
```

**Page 535:**

Wouldn't that be a nice exercise?

Alternatively, you can remove the tab on top of the Hex View by using the `addStandaloneView()` method instead of the `addView()` method in class `RcpPerspective`.

**Page 558:**

Steams, Beth: “*Java Beans 101*”; Sun Microsystems,  
<http://java.sun.com/developer/onlineTraining/Beans/bean01/index.html>; 2000.