

# Errata und Addendum "Java Entwicklung mit Eclipse 2"

Stand: 6. Oktober 2003

Inzwischen ist die Rohübersetzung ins Englische fertig (wird bei John Wiley & Sons erscheinen).

Während der Übersetzung des Buches fand ich noch einige Verbesserungsmöglichkeiten. Diese Fehlerkorrekturen und Ergänzungen möchte ich den Lesern der deutschsprachigen Ausgabe nicht vorenthalten. In der folgenden Liste sind zu löschende Textpassage **durchgestrichen und gelb markiert**. Neue Textpassagen sind in **roter** Farbe gehalten.

In der inzwischen erschienen Neuauflage sind die Errata und Addenda, die hier mit einer **blauen** Überschrift versehen sind, bereits berücksichtigt.

## **Seite 21, Abschnitt 1.3.3:**

Die Seiten *Classfiles* und *JDK Compliance* der Compiler-Einstellungen befinden sich inzwischen auf einer gemeinsamen Seite *Compliance & Classfiles*.

## **Seite 29, Abschnitt 1.5, zweiter Absatz:**

Erzeugt wird eine *Scrapbook*-Seite durch Anklicken des *Create a Scrapbook Page* Buttons in der Eclipse-Toolbar, direkt rechts neben dem **Button für ein neues Interface** *New Java Package* **Button**.

## **Seite 30, Abschnitt 1.5, erster Absatz, letzter Satz:**

**Selbstverständlich kann das Scrapbook auch zusammen mit dem Debugger benutzt werden (siehe Abschnitt 4.1).**

## **Seite 32, Abschnitt 1.6.2, vierter Absatz, letzter Satz:**

Mit dem **Advanced Search** *Search Scope* kann man die Suche auf bestimmte Kapitel beschränken.

## **Seite 32, Abschnitt 1.6.2, fünfter Absatz, dritter Satz:**

In diesem Zusammenhang ist auch der **Synchronize Navigation** *Show in Table of Contents*-Button (links neben dem Druckersymbol) nützlich:

## **Seite 38, Abschnitt 1.6.3, am Ende des Abschnitts hinzufügen:**

- *Generate Delegate Methods ...* . Diese Funktion kann auf nicht-primitive Felder angewandt werden und repliziert die Methoden des Feldtypes in die Typdeklaration (Klasse oder Interface), die das entsprechende Feld enthält.

## **Seite 39, Abschnitt 1.6.4, Bildunterschrift 1-21:**

Abb. 1-21 Der Korrektur-Assistent in Aktion. Der **unbenutzte Parameter *args* und der fehlerhafte Methodenname *println()* sind** rot unterstrichen. Links unter dem **Wanddreieck** *Task-Symbol* erkennt man die gelbe Glühbirne. Ein Klick darauf bietet als Korrektur die Methodennamen *print()* und *println()* an. Rechts davon wird angezeigt, wie die Methode nach Auswahl von *println()* aussehen würde.

### Seite 50, Abschnitt 2.2.2, zweiter Absatz, dritter Satz:

So ist beispielsweise die Ressource `BaseSynthesizerAnimationEvent.java` im Pfad `\eclipse\workspace\DukeSpeaks\com\sun\speech\engine\synthesisfreetts\relp\BaseSynthesizerAnimationEvent.java` gespeichert.

### Seite 51, Abschnitt 2.2.4, Aufzählung:

*Go to > Resource*. Bringt eine alphabetische Auswahlliste von Ressourcen und springt dann zur ausgewählten Ressource.

- Mit der Funktion *Select Working Set ...* kann eine benannter Working Set ausgewählt werden, um die im Navigator angezeigten Ressourcen auf die Ressourcen dieses Working Sets zu beschränken. Die Funktion gestattet auch die Neuanlage von Working Sets.
- Die Funktion *Deselect Working Set* entfernt diese Einschränkungen wieder.
- Mit der Funktion *Edit Active Working Set ...* kann der aktive Working Set modifiziert werden.

### Seite 52, Abschnitt 2.3, erster Absatz, zweiter Satz:

In Abbildung 2-1 sehen wir textbasierte Dateien wie die `.java`- und `.html`-Dateien und die Datei `Makefile`, aber auch binäre Dateien wie die `.class`-Dateien.

### Seite 52, Abschnitt 2.3, dritter Absatz, dritter Satz:

Hierzu ruft man die Funktion *Window > Preferences > Workbench > File Associations* auf.

### Seite 53, Abschnitt 2.4.1, zweiter Absatz:

Die Java-Spezifikation verlangt allerdings, dass sich die Package-Struktur isomorph auf eine Verzeichnisstruktur abbilden lässt. So müssen – wie in Abbildung 2-3 gezeigt – alle Ressourcen des Package `com.sun.speech.engine.textfreetts.rel` im relativen Pfad `com/sun/speech/engine/textfreetts/rel` gespeichert sein. In Eclipse ist diese Pfadangabe immer relativ zum Quelltext-Wurzelverzeichnis für das Projekt. Der relative Pfad `com/sun/speech/engine/textfreetts/rel` entspricht daher dem physischen Pfad `\eclipse\workspace\DukeSpeaks\com\sun\speech\engine\textfreetts\rel`.

### Seite 54, Abschnitt 2.4.2, erster Absatz, letzter Satz:

Zusätzlich lässt sich die Ansicht auf ein *Working Set* (eine vom Benutzer definierte Auswahl von Ressourcen) beschränken.

Unter dem Drop-down Menü der Werkzeugleiste finden wir auch die gleichen Funktionen zur Verwaltung von Working Sets.

### Seite 58, Abschnitt 2.5.2, Aufzählung:

- **Add JavaDoc Comment.** Fügt eine JavaDoc-Kommentar zum selektierten Element hinzu (siehe Abschnitt 1.6.3).
- **Generate Getter and Setter.** Diese Funktion erscheint nur bei Typen mit Feldern und erlaubt die Erzeugung von Zugriffsmethoden (siehe Abschnitt 1.6.3).
- **Source>....** Verschiedene Funktionen zum automatischen Vervollständigung von Code (siehe Abschnitt 1.6.3).

### Seite 62, Abschnitt 2.7, dritter Aufzählungspunkt, anfügen:

Das gleiche erreicht man, indem man mit der rechten Maustaste auf die Titelzeile eines Views klickt und dann die Kontextfunktion *Fast View* auswählt.

### Seite 69, Abschnitt 2.11, erster Absatz, erster Satz:

Die *Java Browsing*-Perspektive (Abbildung 2-15) erlaubt eine etwas andere Sicht auf die Struktur eines Java-Projekts und erinnert an *VisualAge*-Zeiten.

### Seite 72, Abschnitt 3.1, letzter Absatz, erster Satz:

Damit uns in diesem Kapitel das in Kapitel 1 erstellte **ReadmeHelloWorld**-Beispiel nicht stört, legen wir uns noch ein *Working Set* an.

### Seite 78, Abschnitt 3.3.2, Codebeispiel, Methode *fireAnimationEvent()*:

```
AnimationEvent e =
    new AnimationEvent(currentTime, end, phone);
```

### Seite 96, Abschnitt 4.1.3, Aufzählung:

- *Step Into* (F5). Die schrittweise Ausführung wird ggf. auch in eine aufzurufende Methode hinein fortgesetzt. **Befinden sich allerdings mehrere Methodenaufrufe auf einer Zeile, so werden alle diese Methoden schrittweise ausgeführt. In einem solchen Fall ist es besser, den in Frage kommende Methodenaufruf zu selektieren und dann die Kontextfunktion *Step into Selection* aufzurufen.**

### Seite 99, Abschnitt 4.1.5, erster Absatz, letzter Satz:

Mit dem Code-Assistenten (siehe Abschnitt 1.6.3) reicht die Eingabe von **std::sysout** bzw. **std::syserr** um eine solche Testausgabe in den Code einzufügen.

### Seite 101, Abschnitt 4.2, erster Absatz, letzter Satz:

Detaillierte Information über JUnit erhält man unter [www.junit.org](http://www.junit.org) oder in [Westphal2003] bzw. [Link2002].

### Seite 116, Abschnitt 5.1.4, Bildunterschrift 5-3:

**Abb. 5-3** Der Synchronize-View zeigt die Unterschiede zwischen lokalem Workspace und zentralem Repository. Hier hatten wir an der Datei `PlayerTextTest.java` eine Änderung angebracht und dann das Synchronize... Kommando ausgeführt. Die Datei wird nun mit einem Pfeil nach rechts für eine herausgehende Änderung angezeigt.

### **Seite 121, Teil 2, letzter Absatz, letzter Satz:**

Auf Grund dieser Tatsache haben auch alle `SWTJFace`-Komponenten den nativen Look&Feel, ohne jedoch eine direkte Entsprechung im nativen Windowing-System zu besitzen.

### **Seite 125, Abschnitt 6.3, erster Absatz, erster Satz:**

Im Package `org.eclipse.swt` sind lediglich drei Klassen definiert:

### **Seite 149, Abschnitt 6.5.9, erste Tabelle, zweites Reihe:**

Vor jeder Tabellenzeile jedem Baumknoten wird eine Checkbox abgebildet.

### **Seite 150, Abschnitt 6.5.9, Codebeispiel:**

```
// SelectionListener hinzufügen
```

### **Seite 152, Abschnitt 6.5.10, letzter Satz:**

Zusätzlich muss sie für jede Positionierung auch in der Ereignisverarbeitung Größenveränderung der Gruppe aufgerufen werden.

### **Seite 155, Abschnitt 6.5.12, Menüs:**

Menüs können mit Hilfe der Klasse `Menu` aufgebaut werden. Mit den folgenden Stilkonstanten kann das Erscheinungsbild von Werkzeugleisten Menüs bestimmt werden:

### **Seite 156, Abschnitt 6.5.12, dritter Absatz:**

Das folgende Beispiel zeigt den Aufbau eines einfachen Menüs mit zwei Menütiteln einem einzigen Menütitel:

### **Seite 162, Abschnitt 6.6.3, Tabelle, letzte Reihe:**

Dieses Feld kontrolliert den Minimalabstand zwischen Feldern GUI-Elementen in Pixeln.

### **Seite 171, Abschnitt 6.7.4, letzter Absatz, vorletzter Satz:**

Mindestens die folgenden Dateiformate werden gelesen: BMP, GIF, JPG, PNG und ICO.

### **Seite 172, Abschnitt 6.7.4, am Ende des Codebeispiels:**

```
gc.drawImage(buffer, 0, 0);  
// Den Grafikkontext des Puffers entsorgen  
bufferGC.dispose();  
// Den Puffer entsorgen  
buffer.dispose();
```

### **Seite 183, Abschnitt 7.2, fünfter Absatz:**

Mit der Methode `getReturnCode()` oder als Ergebnis der `open()` Methode erhält man die betätigte Taste: `Window`. `OK` für den `OK`-Button, `Window.CANCEL` für den `Cancel`-Button.

### **Seite 193, Abschnitt 7.3.3, Überschrift:**

7.3.3 Zellenfeldeditoren

### **Seite 196, Abschnitt 7.4.1, zweiter Satz unter Tabelle:**

Dabei werden `TextSelection`-Instanzen ausgetauscht.

### **Seite 199, Abschnitt 7.4.2, Anmerkungen, erster Satz:**

Anmerkungen zu einem Dokument werden abseits der `IDocument`-Instanz verwaltet. Das Package `org.eclipse.jface.text.source` stellt dazu das Interface `IAnnotationModel` mit der Standardimplementierung `AnnotationModel` bereit. Mit deren `connect()`-Methode kann das Modell mit dem Dokument verbunden werden. Dem `SourceViewer` wird das jeweilige Annotierungsmodell mit der `setDocument()`-Methode zusammen mit der `IDocument`-Instanz mitgeteilt.

### **Seite 200, Abschnitt 7.4.2, letzter Satz:**

Ein einfaches Beispiel für einen Inhaltsassistenten finden Sie in Kapitel 8.7 beim HTML-Editor.

### **Seite 201, Abschnitt 7.4.2, letzter Satz:**

Ein einfaches Beispiel für die regelbasierte Textrepräsentation finden Sie in Kapitel 8.7 beim HTML-Editor.

### **Seite 203, Abschnitt 7.5.2, letzter Absatz, erster Satz:**

Der `StatusLineManager` realisiert sich selbst mit Hilfe erzeugt beim Aufruf der Methode `createControl()` als ein `StatusLine`-Objekt.

### **Seite 213, Abschnitt 8.1, Aufzählung:**

- Es sollte möglich sein, den Text der Beschreibung mittels HTML-Markup zu dekorieren. Beim Erstellen von Beschreibungen sollte die Applikation HTML-unkundige Benutzer adäquat unterstützen, z.B. beim Einfügen von Schlüsselwörtern in den Text.

### **Seite 215, Abschnitt 8.2, zweiter Absatz:**

Außerdem werden noch die JARs `boot.jar` und `runtime.jar` erforderlich, deren Funktionalität von dem hier verwendeten `TableViewer` genutzt wird. Diese Abhängigkeit des `JFace` von der Eclipse-Plattform soll erst mit Version 2.23.0 fallen.

### **Seite 218, Abschnitt 8.3, Codebeispiel:**

```
/**
 * Player-Modul. Dieses Modul demonstriert verschiedene Techniken
 * des SWT, insbesondere das Zusammenspiel des SWT-Threads mit anderen
 * Threads (inklusive AWT-Threads).
 */
```

### Seite 220, Abschnitt 8.3, erster Absatz nach Code:

Die Methode `run()` entspricht in etwa den SWT-Programmen aus Kapitel 76.

### Seite 233, Abschnitt 8.3, zweites Codebeispiel:

```
private void showPlaylist() {
    if (playlistWindow == null) {
        // Neues Playlist-Fenster erzeugen
        playlistWindow = new PlaylistWindow(toplevelShell, this.playlistModel);
    }
}
```

### Seite 243, Abschnitt 8.6, letzter Absatz vor letztem Codebeispiel:

In der Methode `close()` entfernen wir wieder unseren Eintrag als `SelectionListener` aus dem `Playlist-ViewerModell`.

### Seite 247, Abschnitt 8.6, zweiter Absatz nach Codebeispiel, dritter Satz:

Für die Titelspalte erzeugen wir einen `TextCellEditor`, für die Spalten mit der Sound-Datei und der Bilddatei erzeugen wir `FileCellEditoren` und für die ZeileSpalte mit der Beschreibung einen `DescriptionCellEditor`.

### Seite 247, Abschnitt 8.6, zweites Codebeispiel:

```
/**
 * Konstruktor für PlaylistViewer.
 * @param parent - übergeordnetes Composite
 * @param parentstyle - Stilangaben
 * @param model - das Playlist-Modell
 */
```

### Seite 251, Abschnitt 8.6, erstes Codebeispiel:

```
// Für die Statuszeile setzen wir die Schriftfarbe auf rot.
statusLine.setForeground(
    Display.getCurrent().parent.getDisplay().getSystemColor(SWT.COLOR_RED));
```

### Seite 257, Abschnitt 8.6, dritter Absatz nach Codebeispiel:

Beim Laden des GIF-Bildes benötigen wir eine `Display`-Instanz, um aus dem Bild eine `Image`-Instanz zu erzeugen.

### Seite 258, Abschnitt 8.6, erstes Codebeispiel:

```
/**
 * @see org.eclipse.jface.viewers.IContentProvider#dispose()
 */
public void dispose() {
    // Zum Schluss geben wir das Warn-Ikon wieder frei
    if (alertImage != null) {
        alertImage.dispose();
        alertImage = null;
    }
}
```

### Seite 264, Abschnitt 8.7, zweiter Satz:

Die Klasse baut auf einem `RuleBasedScanner` auf (siehe Abschnitt 7.4.2).

### Seite 267, Abschnitt 8.7, Codebeispiel:

```
private List computeProposals(String qualifier) {
```

```

List propList = new ArrayList();
for (int i = 0; i < TAGS.length; i++) {
    String insert = "$" + TAGS[i] + " ";
    // Cursor zwischen beide Tags stellen
    int cursor = insert.length();
    // Vorschlag erzeugen
    if (insert.startsWith(qualifier))
        // wir lassen nur die Vorschläge zu,
        // die dem angefangenen Schlüsselwort entsprechen
        propList.add(insert);
}
return propList;
}

```

### **Seite 273, Abschnitt 8.9, erster Absatz:**

Wie schon erwartet, sieht die Applikation (bis auf das Beschreibungsfenster) wie eine native Applikation des jeweiligen Betriebssystems aus. Die Fenster und Dialoge verhalten sich unter Windows wie echte Windows-Fenster und -Dialoge, und sind in der Tat nichts anderes.

### **Seite 283, Abschnitt 9.2.2, am Ende als neuen Absatz anfügen:**

Ebenfalls wichtig ist die Methode `getDialogSettings()`, mit der man die Einstellungen aller persistenter Dialoge im Plugin erhält (siehe auch Abschnitt 7.2.3). Für jeden einzelnen Dialog legt man am Besten in diesen Einstellungen eine eigene Sektion an. Das Laden und Abspeichern der Einstellungen übernimmt das Plugin.

### **Seite 287, Abschnitt 9.3.1, "Das Wurzelverzeichnis", letzter Satz:**

Das Wurzelverzeichnis kann man von der jeweiligen Plugin-Instanz mit `myPlugin.getWorkspace().getRoot()` erhalten.

### **Seite 309, Abschnitt 9.4.4, letzter Absatz vor Bild, dritter Satz:**

Durch Angabe von `minOccurs="0"` können optionale Knoten definiert werden.

### **Seite 309, Abschnitt 9.4.4, Bildunterschrift 9-12, erster Satz:**

Der Schema-Editor mit der geöffneten Schemadatei »Project Manager Diagrams vFilter.exsd«.

### **Seite 318, Abschnitt 9.5.3, erster Absatz, zweiter Satz:**

Die Standardimplementierung `EditorActionBarContributor` besitzt eine Methode `getActionBars()`, mit der man eine `IActionBars`-Instanz holen kann.

### **Seite 320, Abschnitt 9.5.3, Tabelle, dritte Zeile:**

Diese Methode muss überschrieben **erweitert** werden, wenn man die erweiternde Unterklasse Ressourcen (Farben, Fonts, Drucker, etc.) bei Beendigung freigeben muss.

### **Seite 324, Abschnitt 9.5.4, letzter Absatz, vierter Satz:**

Die Methode liefert eine `IActionBars`-Instanz, von der man mittels der Methode `getMenuManager()` bzw. `getToolManager()` die Manager für das Menü (`IMenuManager`) bzw. die Werkzeugleiste (`IToolManager`) erhält.

### **Seite 332, Abschnitt 9.5.5, Tabelle, zweite Reihe, zweite Spalte, vorletzter Satz:**

Wird dieses Attribut nicht angegeben, so ist die Aktion unabhängig von der Zahl der selektierten Elemente aktiv (wie »\*«) hängt die Aktivierung der Aktion nicht von der Zahl der selektierten Elemente ab, sondern kann durch Programmlogik bestimmt werden.

### **Seite 345, Abschnitt 9.5.10, "Eine Hilfe-Kontextzuordnung erstellen", Ende des Abschnitts:**

Hier spezifiziert man im Attribut file den Namen des Inhaltsverzeichnisses der Hilfe-Kontextzuordnung (z.B. contexts.xml).

In den Abschnitten 11.3 und 11.11.32 sehen wir, wie solche Manifest-Deklarationen aussehen können.

### **Seite 354, Abschnitt 10.3.1, als zweiten Satz einfügen:**

...oder ob die Versionsnummer des Features von den Plugins übernommen werden soll. Wichtig ist nach der Änderung von Versionsnummern in Plug-ins diese Taste erneut zu betätigen, um das Feature-Manifest auf den neuesten Stand zu bringen!

### **Seite 356, Abschnitt 10.3.2, erster Absatz, vierter Satz:**

Über die Modifikation dieser Variablen lässt sich so der BuildExport-Prozess steuern.

### **Seite 356, Abschnitt 10.3.2, erster Absatz, vierter Satz:**

**Achtung:** Sie sollten beide Variablen nicht gleichzeitig verwenden, da dies zu fehlerhaftem Verhalten führen kann (Eclipse 2.4 RC2). Falls Sie beide Variablen in der selben Properties-Datei verwenden, sollten Sie die generierten Archive immer noch einmal kontrollieren. Manchmal sind in einem solchen Falle die Ergebnisse etwas überraschend.

### **Seite 356, Abschnitt 10.3.2, letzter Satz vor der unteren Tabelle:**

Dabei kann man die üblichen Ant-Muster für Dateipfade verwenden:

### **Seite 357, Abschnitt 10.3.3, erster Absatz, erster Satz:**

Jedes primäre Feature (siehe Abschnitt 10.3.1) kann eine Willkommenseite welcome.xml definieren, die den Benutzer mit dem Feature vertraut macht.

### **Seite 359, Abschnitt 10.4.1, vorletzter Absatz, letzter Satz:**

Existiert eine solche Datei noch nicht, kann man sie mit einem Druck auf die Package Taste im Feature-Editor des zugehörigen Feature-Projekts erzeugen (siehe Abschnitt 10.3.2). durch Anwendung der Kontextfunktion *Create Ant Build File* auf die Manifestdatei `plugin.xml` erzeugen.

### **Seite 361, Abschnitt 10.4.1, 6. Zeile von unten:**

```
// Dateinamen extrahieren
String urls = (url.getPath() + EXAMPLE_FILE).substring(5);
```

### **Seite 365, Abschnitt 10.4.2, erster Absatz, letzter Satz:**

Dabei werden die zuvor mit dem Feature-Editor erstellten ANT-Skripte verwendet, die auf Basis der früher im Feature Editor gemachten Angaben erstellt werden (siehe Abschnitt 10.3.2).

### Seite 365, Abschnitt 10.4.3, zweiter Absatz:

Solche Archive erstellt man nicht manuell, sondern greift auf das Target `build.source` des ANT-Skripts `build.xml` zurück. Dazu selektiert man dieses Skript und ruft die Kontextfunktion `Run ANT` auf. (Falls dieses Skript nicht existiert, erzeugen wir es, indem wir die Kontextfunktion `Create Ant Build File` auf die Manifestdateien `plugin.xml` oder `feature.xml` ausführen.) Im folgenden Dialog entfernt man auf der Seite `Targets` das Häkchen vom `von Run-Default Target` und setzt stattdessen ein Häkchen auf das Target `build.source`. Dann betätigt man die Taste `Run`.

### Seite 366, Abschnitt 10.5, dritter Absatz, zweiter Satz:

Dort sind unter `Current Configuration Eclipse Platform` die Details der aktuellen Konfiguration aufgelistet.

### Seite 368, Abschnitt 10.6.1, Aufzählung, erster Punkt, vierter Satz:

Diese Programmzeile wird zusätzlich mit einem Kommentar wie `// $NON-NLS-1$` versehen, der anzeigt, dass bei der nächsten Ausführung der Funktion `Externalize Strings` diese Zeile die entsprechende Stringkonstante (nun der Schlüssel) nicht untersucht werden soll.

### Seite 371, Abschnitt 10.6.3, an den Abschnitt anfügen:

Allerdings hat dieses Vorgehen einen gravierenden Nachteil: bei einem fehlenden Sprachpaket wird einfach nichts angezeigt, anstatt auf die englische Version zurückzugreifen. Glücklicherweise gibt es jedoch eine Alternative, und die kommt ohne Substitutionsvariable aus. Sie beruht ausschließlich auf einer Namenskonvention für Verzeichnisstrukturen. So können z.B. deutschsprachige Hilfe- und Willkommenstexte in einem Verzeichnis `n1/de` oder auch `n1/de/DE` abgelegt werden, je nachdem, ob man sich auf das gesamte deutschsprachige Gebiet oder nur auf Deutschland beziehen will. Zum Ablaufzeitpunkt wertet Eclipse die `Locale`-Information der JVM aus und sucht dann nach einem entsprechenden Ordner innerhalb des `n1/` Verzeichnisses. Wird ein solcher Ordner nicht gefunden, werden die Standardhilfe- und -willkommenseiten verwendet. Diese Standardseiten, die normalerweise in englischer Sprache abgefasst sind, werden nicht innerhalb des `n1/` Verzeichnisses abgelegt.

### Seite 371, Abschnitt 10.6.4, erster Abschnitt, letzter Satz:

Auch den übersetzten `$nl`-Ordner importiert man in das Fragment und gibt ihn in der `build.properties`-Datei in der Variablen `bin.includes` an (siehe Abschnitt 10.3.2). Eine Ausnahme bilden die `plugin_locale.properties` Dateien. Sie verbleiben nicht im Projektverzeichnis, sondern müssen in den `src`-Ordner gestellt werden.

### Seite 371, Abschnitt 10.6.4, dritter Abschnitt:

Im Ordner `de-DE/n1/de` befinden sich die übersetzte Willkommenseite, das übersetzte Inhaltsverzeichnis, die übersetzte Kontextzuordnung für die Hilfe und ein Ordner `html`, der alle übersetzten HTML-Seiten enthält. Diesen Ordner `de-DE/n1/de` fügen wir der Variablen `bin.includes` in der Datei `build.properties` zu.

### Seite 380, Abschnitt 11.3.1, Aufzählung, zweiter Punkt, letzter Satz:

Die hier gemachten Definitionen dienen als Bindeglied zu den Deklarationen für Tastenkürzel. **Außerdem geben wir eine Beschreibung an. Die ist zwar nicht gefordert, verhindert aber, dass Eclipse beim Aufruf der Funktion *Window>Preferences>Workbench>Keys* auf einen Fehler läuft!**

### Seite 381, Abschnitt 11.3.1, Mitte des Code-Beispiels:

```
<action
  toolbarPath="com.bdaum.SpellChecker.spell_checker"
  id="com.bdaum.SpellChecker.action1"
  class=
    "com.bdaum.SpellChecker.actions.CheckSpellingActionDelegate"
  enablesFor="**"
  icon="icons/basic/correction_view.gif"
  helpContextId="com.bdaum.SpellChecker.action_context"
  label="Check spelling"
  menubarPath="edit/spelling"
  definitionId="com.bdaum.SpellChecker.check_spelling"
  tooltip="Checks the spelling of any text"/>
</actionSet>
```

### Seite 381, Abschnitt 11.3.1, Ende des Code-Beispiels:

```
<actionDefinition
  name="Check Spelling"
  description="Starts Spell Checking"
  id="com.bdaum.SpellChecker.check_spelling"/>
```

### Seite 390, Abschnitt 11.5:

Den Aufruf

```
updateActionEnablement(action);
```

aus der Methode `setCurrentFocusControl()` entfernen und statt dessen in die Methode

```
public void focusGained(FocusEvent e) {
}
```

einsetzen.

### Seite 390, Abschnitt 11.5, Ende des Code-Beispiels:

```
/* FocusListener-Methoden */
```

### Seite 392, Abschnitt 11.5, Anfang des Code-Beispiels:

```
public void run(IAction action) {
  // Textbehälter holen
  final Object target = getSpellCheckingTarget();
  if (container.target != null) {
```

### Seite 393, Abschnitt 11.5, Anfang des Code-Beispiels:

```
private Object getSpellCheckingTarget() {
  // Workbench-Komponente mit Hilfe des Verweises holen
```

### Seite 400, Abschnitt 11.6, Mitte des Code-Beispiels:

```
// Menüs bzw. Werkzeugleiste füllen
private void fillContribution(IMenuManager menuManager, ContributionManager manager) {
```

### Seite 403, Abschnitt 11.6.1, Anfang des Code-Beispiels:

```
// TableView aktivieren/aktualisieren  
viewer.setInput(currentEvent);
```

### Seite 405, Abschnitt 11.6.3, dritter Absatz, zweiter Satz:

Jedes Aktions-Icon kann drei Zustände einnehmen:

### Seite 419, Abschnitt 11.9.1, Codebeispiel, unten:

```
/**  
 * Method initializePublicPreferences.  
 * @param store - der PreferenceStore  
 * @param prefix - der aktuelle Präfix für die Schlüssel  
 */
```

### Seite 420, Abschnitt 11.9.1, Codebeispiel, oben:

```
/**  
 * Method initializeHiddenPreferences.  
 * nicht-öffentliche Einstellungen für den SpellCheck-Algorithmus  
 * @param store - der PreferenceStore  
 * @param prefix - der aktuelle Präfix für die Schlüssel  
 */
```

### Seite 427, Abschnitt 11.10.3, Codebeispiel, Mitte:

```
// Die aktive Hilfe läuft nicht im SWT-Thread ab.  
// Deshalb müssen wir GUI-Zugriffe mit syncExec() kapseln.  
// Das Display müssen wir vom aktuellen Thread CheckSpellingActionDelegate holen,  
// da wir keinen Zugriff auf ein Widget haben, das uns  
// eine Display-Instanz liefern könnte.
```

### Seite 427, Abschnitt 11.10.3, Codebeispiel, unten:

```
// aktives Workbench-Fenster holen  
IWorkbenchWindow window =  
    PlatformUI.getWorkbench().getActiveWorkbenchWindow();  
// wenn keines aktiv ist, nehmen wir das erste Beste.  
if (window == null)  
    window = PlatformUI.getWorkbench().getWorkbenchWindows()[0];
```

### Seite 433, Abschnitt 11.11.4, Codebeispiel, Mitte:

```
/**  
 * Method initializePublicPreferences.  
 * Java-spezifische Vorgabewerte für die öffentlichen Optionen.  
 * @param store - der PreferenceStore  
 * @param prefix - der aktuelle Präfix für die Schlüssel  
 */
```

### Seite 437, Abschnitt 11.12.2, zweiter Absatz, vierter Satz:

Auf dem nächsten Screen ergänzen wir die vorgeschlagene Feature ID noch mit einem Präfix, um sie eindeutig zu machen, z.B. »com.bdaum.SpellCheckerforEclipse« ersetzen wir die vorgeschlagene FeatureID durch »com.bdaum.SpellChecker«. Diese ID stimmt mit der ID unseres SpellChecker-Plugins überein.

### Seite 437, Abschnitt 11.12.2, vierter Absatz, dritter Satz:

Im Abschnitt *License Agreement* geben wir dann im Feld *Optional URL* den Wert `license.html` an.

### Seite 438, Abschnitt 11.12.2, Codebeispiel, dritte Zeile:

```
id="com.bdaum.SpellCheckerforEclipse"
```

### Seite 440, Abschnitt 11.12.3, letzter Absatz, letzter Satz:

Diese enthalten die Installationsdateien, die mit den in Abschnitt 11.12.2 erzeugten konfigurierten Ant-Skripten `build.xml` erzeugt wurden.

### Seite 440, Abschnitt 11.12.3, Codebeispiel, siebte Zeile:

```
<feature url="features/com.bdaum.SpellCheckerforEclipse_1.0.0.jar"
id="com.bdaum.SpellCheckerforEclipse" version="1.0.0"/>
```

### Seite 444, Anhang A, Tabelle GUI-Design, Zeile anfügen:

Name	Erläuterung	Homepage
W4Eclipse	Visueller Web-GUI-Designer der Firma INNOOPRACT ( <a href="http://www.innoopract.de">www.innoopract.de</a> ). Kommerzielles Produkt, jedoch kostenlos für die ersten 5000 Objekte.	<a href="http://w4toolkit.com">http://w4toolkit.com</a>
SWT-Designer	Visueller GUI-Designer für SWT und JFace. Kommerzielles Produkt, die Community-Version ist jedoch frei erhältlich.	<a href="http://www.swt-designer.com">http://www.swt-designer.com</a>

### Seite 445, Anhang A, Tabelle Webprojekte, Zeile anfügen:

Name	Erläuterung	Homepage
MyEclipse	Verschiedene Werkzeuge für die J2EE -Entwicklung, enthält insbesondere einen JSP-Editor und -Debugger. MyEclipse ist das Produkt eines Joint Ventures zwischen Genuitec ( <a href="http://www.genuitec.com">www.genuitec.com</a> ) und dem sächsischen Start-up Bebbosoft ( <a href="http://www.bebbosoft.de">www.bebbosoft.de</a> ). Kommerzielles Produkt.	<a href="http://www.myeclipse.org">http://www.myeclipse.org</a>

### Seite 448, Anhang B, erster Absatz:

In beiden Fällen löscht man am besten die bisherigen Verweise auf externe JARs und fügt sie anschließend mit der Funktion *Add external JARs* neu hinzu.

Für Plugin-Projekte geht das freilich einfacher mit der Kontextfunktion *Update Classpath ....*

### Seite 449, Anhang C, Projekt 3:

Die Engine für die Rechtschreibprüfung (Version 0.34) findet man unter <http://sourceforge.net/projects/jazzy>.

**Seite 451, Bibliographie, einfügen:**

[Link2002] Johannes Link, Peter Fröhlich; Unit Tests mit Java; dpunkt.verlag, Heidelberg, 2003