

Rich-Client-Entwicklung mit Eclipse 3.3

Errata und Addenda

Stand 30. Juni 2008

Seite 36:

Die Methode `setWorkload()` `setDuration()` der Klasse `Task` beginnt mit:

Seite 38:

```
Iterator<Task> it = result.iterator();
while (it.hasNext())
    if (dependsOn(it.next()))
    if (it.next().dependsOn(this))
        it.remove();
```

Seite 48:

```
changed.add(project);
project.notifyModelListener(ModelListener.MODIFY,
                             changed, null);
return status;
```

Seite 52:

```
private Allocation[] computeResourceAllocation(Resource res) {
    List<Allocation> allo = new ArrayList<Allocation>();
    // Für jede Ressource die Belegungen ermitteln
    if (utiMap == null)
        computeResourceUtilizations(res.getParent());
    // Nach allen Belegungen suchen, die aus einem nicht
    // zu berechnenden Projekt oder von einem schon
    // berechneten stammen
    for (Utilizes uti : utiMap.get(res)) {
```

Seite 66, Fußnote:

1. Seit Eclipse 3.2 unterstützt Eclipse dediziert das Erstellen von Standard-OSGi- und Equinox-Anwendungen, was insbesondere für die Implementierung von eingebetteten Servern interessant ist (siehe auch [Daum20067]).

Seite 190:

Die entsprechende deutsche Übersetzung `about_de_DE.properties` ~~würde wieder im Plugin `com.bdaum.planner.nl` abgelegt (siehe Abschnitt 5.3).~~ **liefert man am besten – wie auch die anderen sprachabhängigen Ressource – in einem separaten Fragment aus (siehe nächster Abschnitt).**

Seite 197:

Schauen wir uns zunächst an, welche `config.ini`-Datei der Produkteditor beim Export **(nach Umstellung auf feature-basierte Konfiguration)** unserer Beispieldanwendung generiert hat:

Seite 212:

Außerdem fügen wir auf der *Runtime*-Seite mit Hilfe der Taste *Add...* das Package `com.bdaum.planner.update` zur Liste der exportierten Packages hinzu. **Anschließend muss im Manifest dieses Features auf der Seite *Dependencies* noch die Taste *Compute* betätigt werden.**

Seite 298:

```
/**
 * Refreshes the master block with new content
 * @param sel - The object to select or null
 */
public void refresh(Object sel) {
    if (sel != null && viewer != null
        && !viewer.getControl().isDisposed()) {
        viewer.setInput(viewer.getInput());
        if (sel != null)
            viewer.setSelection(new StructuredSelection(sel), true);
    }
}
```

Seite 331:

```
control.getDisplay().asyncExec(new Runnable() { // Synchron!
```

Seite 353:

```
URL url = FileLocator.find(bundle, new Path("/gui/"), null);  
url = FileLocator.resolveToFileURL(url);
```

Seite 447:

In der Klasse `PlannerPluginActivator`, die den Lebenszyklus der Anwendung verwaltet, erzeugt man sich dann eine Instanz des Datenbanksystems, wie im nächsten Abschnitt in der Methode `initDb4o()` gezeigt.

```
private void initDb4o(String file) {  
    db=Db4o.openFile(file);  
  
    User u = new User(null, null, new ArrayList<String>());  
    if (!db.get(u).hasNext()) {  
        ArrayList<String> roles = new ArrayList<String>(1);  
        roles.add("Administrator");  
        User user = new User("admin", "password", roles);  
        Arrays.asList(new String[] { "Administrator" });  
        db.set(user);  
        db.commit();  
    }  
}
```

Seite 448:

```
public User loginOnDb4o(String username, String password) {  
    try {  
        User p = new User(username, password, new ArrayList<String>());  
        ObjectSet<Object> result = db.get(p);  
        if (result.hasNext()) {  
            user = (User) result.next();  
            break;  
        }  
        return user;  
    } finally {  
        db.close();  
    }  
}
```

```
}
}
```

Seite 449:

```
public User loginOnDb4o(String username, String password) {
    try {
        Query query=db.query();
        query.constrain(User.class);
        query.descend("name").constrain(username);
        query.descend("password").constrain(password);
        ObjectSet result=query.execute();
        if (result.hasNext()) {
            user = (User) result.next();
            break;
        }
    }
    return user;
} finally {
    db.close();
}
}
```

15.1.2 Abfragen

Wie oben schon erwähnt, kann statt mit Prototypen auch mit Querys gearbeitet werden. Querys sind Java-Objekte, die Schritt für Schritt aufgebaut werden müssen. Das ist zwar etwas mehr Tipparbeit, auf der anderen Seite aber gibt es bei dieser Methode keine Syntaxfehler, die erst zur Laufzeit bemerkt werden. Hier haben wir als Beispiel die Methode ~~checkUserOnDb4o()~~ `loginOnDb4o()` mit Hilfe der *Query*-Technik realisiert:

Seite 498:

Im nächsten Schritt werden die beiden Data Sets miteinander verknüpft. Erzeugen Sie deshalb einen *Joint Data Set*.

~~Dann selektieren Sie den Knoten *Task Data Set* und fügen mit der Kontextfunktion *Insert in Layout* alle Felder in das Design ein, die auf diese Weise als Tabelle angewordnet werden. Unter dem Reiter *Preview Results* können Sie sich nun das Ergebnis einer Abfrage ansehen. Auf der linken Seite wählen Sie aus der Liste den *Project_Task Data Set* und selektieren das Feld `TASKS_ID`.~~

Seite 499:

Den Parameter nennen Sie »~~ProjectReportId~~«

Seite 521:

```
/**  
 * Schließt OpenOffice und beendet das Office-Desktop  
 */  
protectedpublic void terminateOO() {
```

Seite 523:

Unter dem *class*-Attribut spezifizieren Sie `com.bdaum.planner.calc.views.CalcView` als View-Klasse und erzeugen diese mit einem Klick auf *class*.

Anhang E:

Durchgängig muss es **Mylyn** statt ~~Mylin~~ heissen.

Bibliografie Seite 626:

[Horrocks1999] Ian Horrocks; Constructing the User Interface with Statecharts; Addison-Wesley Professional; 1999

[Irvine2003] Veronika Irvine: Drag and Drop – Adding Drag and Drop to an SWT Application, Eclipse Corner Article; www.eclipse.org; 2003.

[Kay2004] Michael Kay; XSLT Programmer's Reference; wrox; 2004

Danksagung

Folgende Leser haben zu diesen Errata beigetragen:

Roman Legat, Tim Krüger, Raphael Charwot, Michael Schaar