MATHEMA campus 2011

Rappen für Kenner
Webanwendungen mit Eclipse RAP

Manfred Borzechowski

mail@borzechowski.de
www.borzechowski.de

Themen

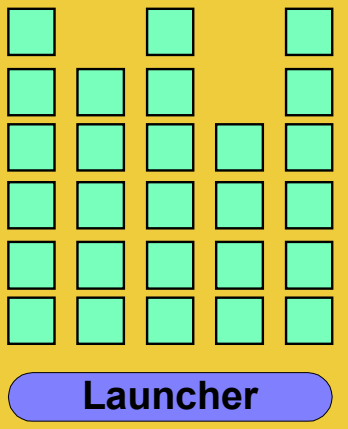
- Einstieg RAP
- Unterschiede RCP-RAP
- Gestaltung der RAP-UI
- Deployment

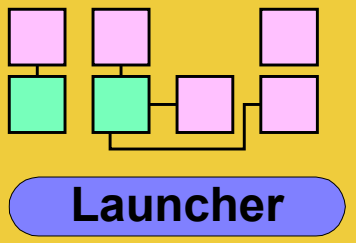
Einstieg RAP

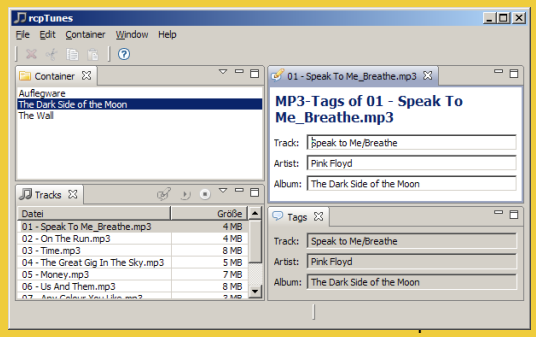
MATHEMA campus 2011

Eclipse RCP in a Nutshell

- Eclipse IDE:
  - Modular durch Plug-ins / OSGi Bundles
  - Basic **Core** Plug-ins: OSGi, Preferences, Jobs
  - Basic **UI** Plug-ins: Workbench, SWT, JFace
  - Und: JDT, Mylin, Debugging, PHP, JEE, Reporting.
- Eclipse RCP:
  - Modulare Fat Clients
  - Basierend auf Eclipse Basic **Core** und **UI** Plug-ins
  - Zusätzliche Plug-ins für Domain-Funktionalität
  - Verhalten, L&F, Modularität analog Eclipse







Projektanforderungen

- Gegeben: 2 Clients für einen Datenbestand
  - Fat Client: Sachbearbeiter
  - Web Client: Online-User
- Anforderung 3. Client:
  - Rich Web Client für Online-Sachbearbeiter-Light

Randbedingungen:
Hohes Java/RCP-Know How
Geringes Seam/Ajax/CSS-Know How

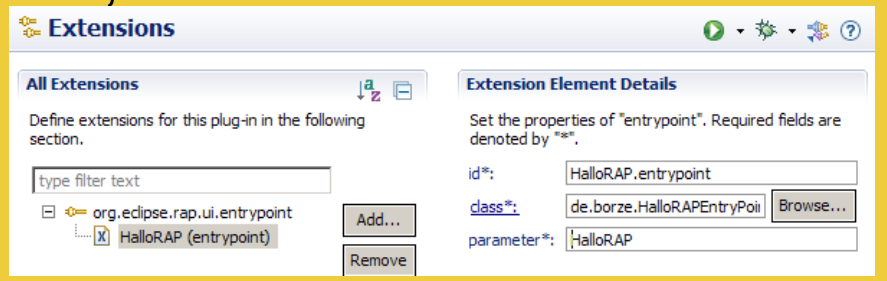
	Fat Client (Eclipse RCP)	Rich Web Client (?)	Web Client (Ajax, Seam,...)
Komplexe Funktionen	+	o	-
Hoher Datendurchsatz	+	o	-
Intuitiv bedienbar	-	o	+
Design anpassbar	-	o	+
Ohne Installation lauffähig	-	+	+
Einfache Programmierung	+		-

Die Idee von RAP

- Eclipse RAP:
  - Komplette Eclipse-Workbench läuft als (JEE) Webanwendung
- Auf dem Server:
  - OSGi-Container, **RWT**, JFace, Workbench
  - Programmierung zu ca. 90-95% wie bei „normalem“ RCP
  - Gewisse Unterschiede im API (hauptsächlich SWT)
- Im Browser:
  - HTML-Widgets, die (mit Ajax) mit Server kommunizieren
  - (wovon der Entwickler nichts wissen muss)
  - Für's „Styling“ gewisse CSS-Kenntnisse nötig

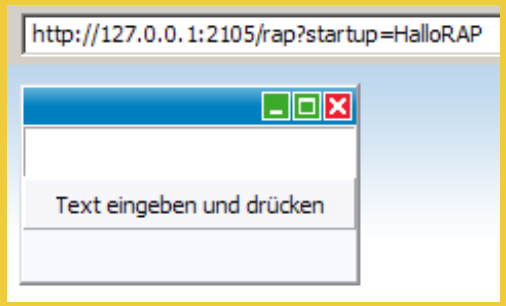
Der Einstieg – Hallo RAP

- Rezept
  - Man nehme ein Eclipse für RCP/RAP Developer,
  - konfiguriere eine RAP Target-Platform,
  - erstelle ein leeres Plug-in Projekt,
  - füge eine Dependency zu **org.eclipse.rap.ui** hinzu,



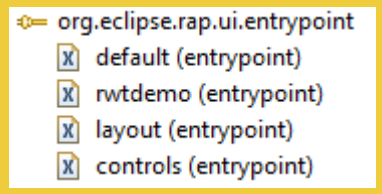
  - erstelle eine Extension zu **entrypoint**,
  - und programmiere in der dazugehörigen Klasse analog SWT.

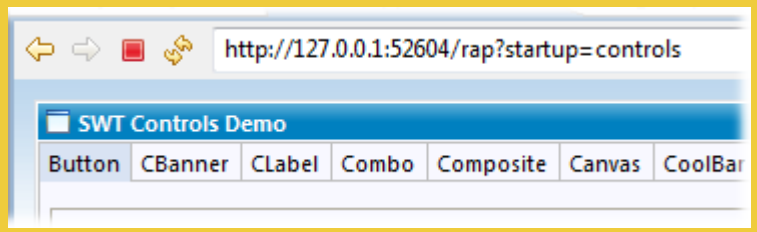
```
public class HalloRAPEntryPoint implements IEntryPoint {


    @Override
    public int createUI() {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setLayout(new FillLayout(SWT.VERTICAL));
```



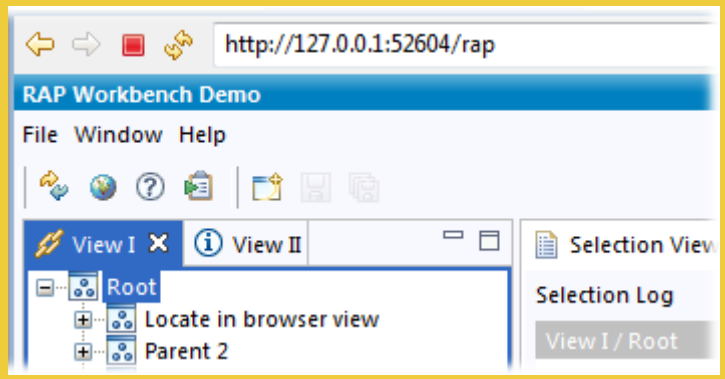
Weitere Beispielanwendungen

- Im Plug-in **org.eclipse.rap.ui.demo**
  - Importieren
  - Weitere EntryPoints finden
  - Mit Startup-Parameter auswählen

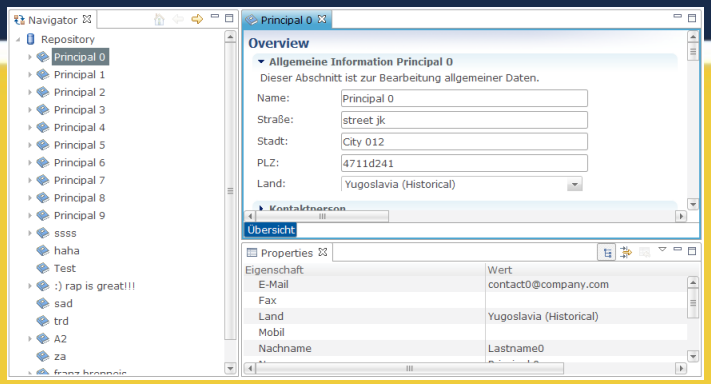




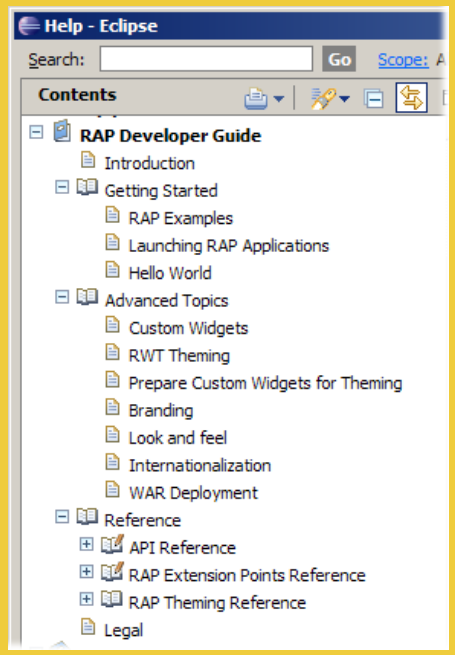
  - Ohne Parameter:
    - **default**
    - RAP-Anwendung mit serverseitiger **Workbench**



Dokumentation und weitere Informationen



- Demo-Projekte
  - http://rap.eclipsesource.com/rapdemo/rms
- Projekt HomePage
  - http://eclipse.org/rap/
- Developer Guide
  - Help Contents
- FAQ
  - http://wiki.eclipse.org/RAP/FAQ



Unterschiede RCP-RAP

RWT - Unterschiede im API

- Fast vollständig vorhanden:
  - Layout
  - Widgets
    - Missing: Native Dialog, TrayItem
  - DnD
  - Events
    - Nicht alle Listener werden unterstützt
- Ca. zur Hälfte vorhanden
  - Custom
    - Missing: StyledText & Co

MISSING

Konzeptionelle Unterschiede

- Für den Programmierer
  - Entwicklung fast wie „normales“ RCP
    - Request-Response-Modell versteckt
    - Spezielle Beachtung der Paradigm Gaps
- Für den Anwender
  - Ansicht fast wie normaler Fat Client
    - Keine „Seiten“, kein „Weiter“
    - Spezielle Beachtung der Benutzerführung

Du bist nicht allein

- **Eine** Workbench – **mehrere** (gleichzeitige) Benutzer
  - *Keine* Singletons für benutzerspezifische Daten!
  - *Session*-Mechanismus notwendig
    - Manuell:
      - **`RWT.getSessionStore()`**
    - Gemanaged:
      - **`SessionSingletonBase.getInstance()`**

```
public class UserData extends SessionSingletonBase {
```

```
UserData userData = (UserData)
    SessionSingletonBase.getInstance(UserData.class);
```

You are not alone

- **Mehrere** Benutzer – **mehrere** Sprachen
  - Formats, Collatoren etc spezifisch für **`RWT.getLocale()`**
  - Für Text-RessourceBundles besser: **`RWT.NLS`**

```
public class Messages {
    private static final String BUNDLE_NAME = "de.borze.comps.messages";

        public static Messages get() {
            return (Messages) RWT.NLS.getISO8859_1Encoded(BUNDLE_NAME, Messages.class);
        }

        // message keys
        public String DatumUndUhrzeit;
        public String Aktualisieren;
```

```
button.setText(Messages.get().Aktualisieren);
```

Dateien für den Benutzer

- **Kein** FileDialog, FileOutputStream **nicht** möglich
- Alternativ: Browser-Widget
  - Download-Link mit setText()
  - Anzeige oder sofort-Download mit setUrl()
- Aber: **Wohin** zeigt die URL?
  - In das Internet (trivial)
  - In lokale statische Ressource (konfigurieren)
  - Auf ServiceHandler (programmieren)

```
<extension
    point="org.eclipse.equinox.http.registry.resources">
    <resource
        alias="/files"
        base-name="files">
    </resource>
</extension>
```

```
IServiceManager manager = RWT.getServiceManager();
IServiceHandler handler = new ReceiptHandler();
manager.registerServiceHandler( "receipt", handler );
```

```
public class ReceiptHandler implements IServiceHandler {

    @Override
    public void service() throws IOException, ServletException {
        HttpServletResponse response = RWT.getResponse();
        response.setContentType("application/rtf");
```

Dateien vom Benutzer

- **Kein** FileDialog, direkter FileInputStream **nicht** möglich
- Alternativ:
  - **Upload**-Widget (Plug-in org.eclipse.rwt.widgets.upload)

```
upload = new Upload(this, SWT.NONE, Upload.SHOW_UPLOAD_BUTTON | Upload.SHOW_PROGRESS);
upload.setBrowseButtonText("Datei auswählen...");
upload.setUploadButtonText("Datei hochladen");
upload.addUploadListener(this);
```

  - Daten landen in Session-spezifischem FileStore
  - Listener werden bei Fortschritt, Fehler und Ende benachrichtigt

```
@Override
public void uploadFinished(UploadEvent uploadEvent) {

    UploadItem uploadItem = upload.getUploadItem();
    String fileName = uploadItem.getFileName();
    long fileSize = uploadItem.getFileSize();
    String contentType = uploadItem.getContentType();
    InputStream inputStream = uploadItem.getFileInputStream();
```

Rückruf

- Manipulation von RWT-Widgets nur im UI-Thread
- Änderungen aus nicht-UI-Thread
  - in Runnable verpacken und mit Display.(a)syncExec() ausführen
  - Zusätzlich **UICallBack**-Mechanismus aktivieren und deaktivieren

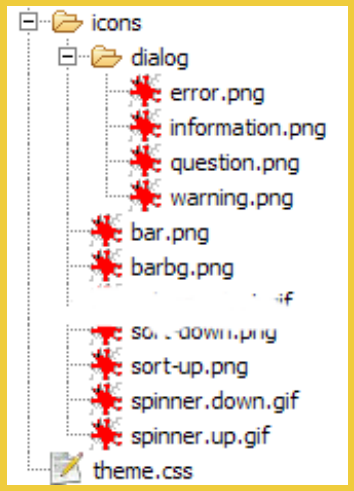
```
UICallBack.activate(id);
Thread bgThread = new Thread(runnable);
bgThread.setDaemon(true);
bgThread.start();
```

```
Runnable runnable = new Runnable() {
    public void run() {
        // Langdauernde Funktion...
        display.syncExec(new Runnable() {
            public void run() {
                MessageDialog.openInformation(shell, "Info", "Fertig");
                UICallBack.deactivate(id);
            }
        });
```

Gestaltung der RAP-UI

Themes – Bilder und Styles

- Das RAP-Demo-Plug-in enthält ein alternatives **Theme**
  - Muss als Extension konfiguriert werden
- Icons für Dialoge
- Grafische Widget-Elemente
  - **Nicht** für Scrollbar
- CSS-StyleSheet
  - Styles für SWT-Widgets
  - Qualifiziert nach verwendetem SWT-Style




```
<extension
    id="org.eclipse.rap.demo.themes"
    point="org.eclipse.rap.ui.themes">
  <theme
      id="org.eclipse.rap.demo.alttheme"
      name="Alternative Demo Theme"
      file="theme1/theme.css"/>
</extension>
```

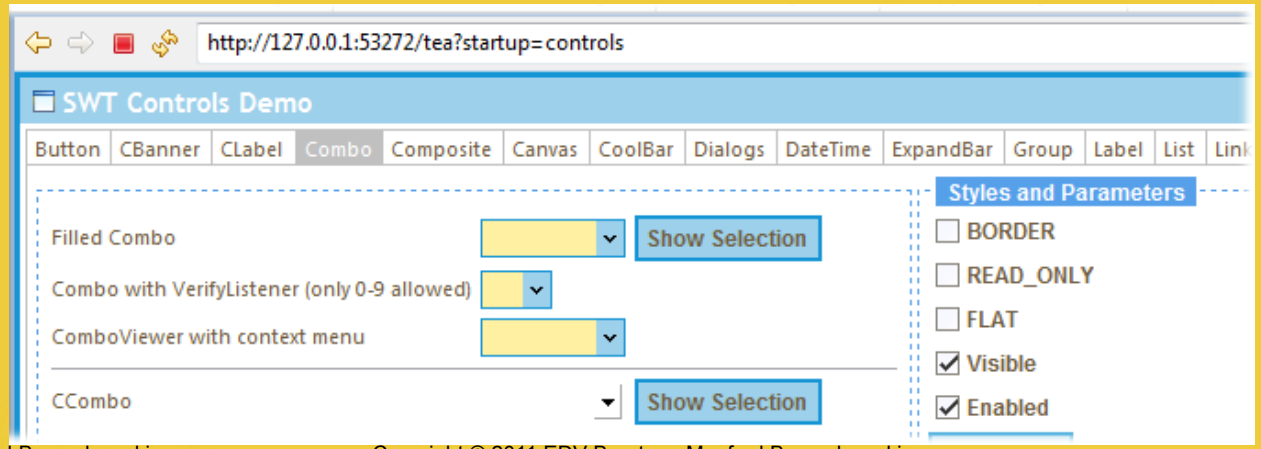
```
Button {
  color: #705e42;
  padding: 3px 6px 3px 4px;
  font: bold 12px Arial, Helvetica, sans-serif;
}

Button[PUSH], Button[TOGGLE] {
  background-color: #9dd0ea;
  border: 2px #1695d4;
}

Button[TOGGLE][FLAT]:pressed {
  background-color: rgb( 227, 221, 158 );
}
```

Branding – It's tea-time

- Ein **Branding** kombiniert
  - Theme
  - favicon.ico
  - Servlet-Name
  - Hintergrund-HTML

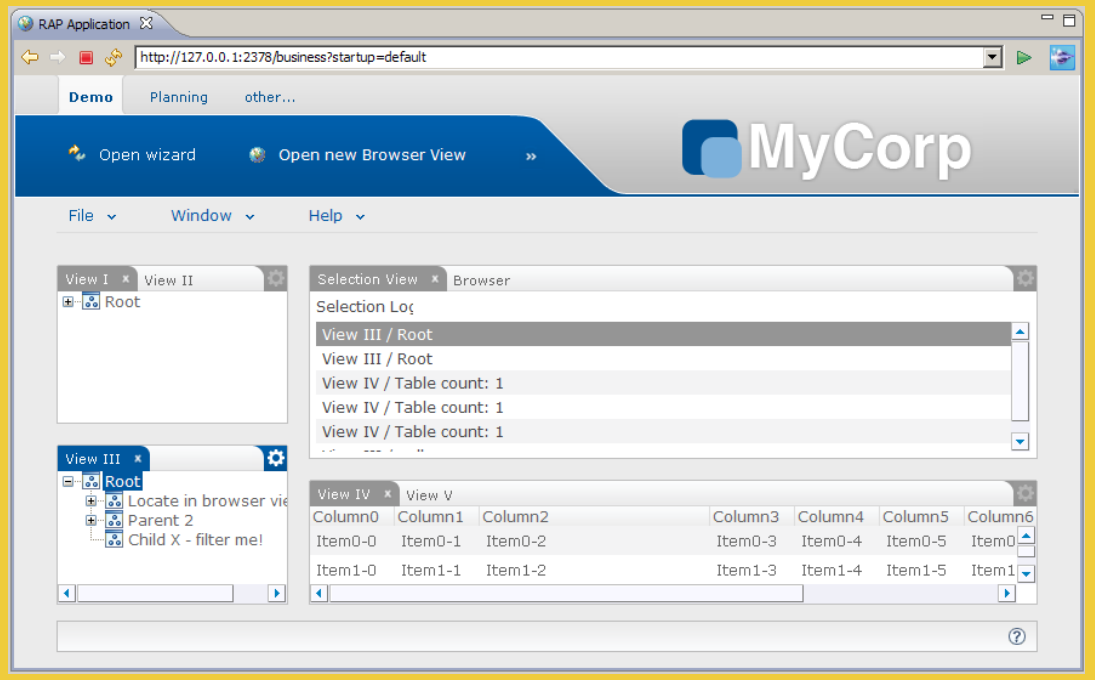
```
<extension
      point="org.eclipse.rap.ui.branding">
   <branding
         body="body.html"
         defaultEntrypointId="org.eclipse.rap.demo.entrypoint1"
         exitConfirmationClass="org.eclipse.rap.demo.DemoExitConfirmation"
         favicon="icons/favicon2.ico"
         id="org.eclipse.rap.demo.branding1"
         servletName="tea"
         themeId="org.eclipse.rap.demo.alttheme"
         title="It&apos;s tea-time">
   </branding>
</extension>
```

- Aufruf mit Servlet-Name (anstelle rap) zeigt Theme



Presentation – Let's do Business

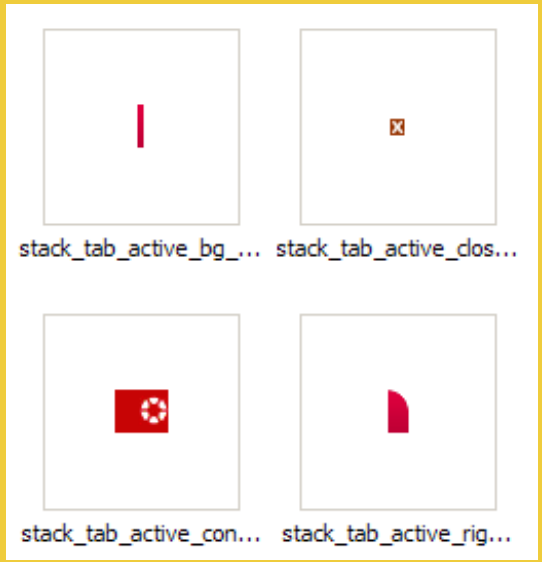
- Ein Branding kann **zusätzlich** eine PresentationFactory referenzieren
  - Icons für grafische Workbench-Elemente
  - Klassen zum Layouten von
    - MenuBar
    - CoolBar
    - ViewParts
    - ...

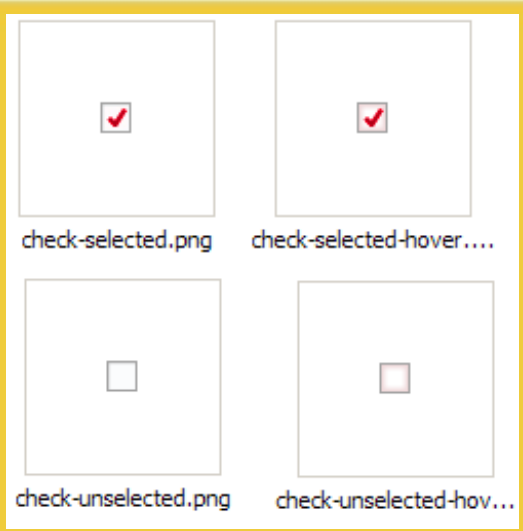


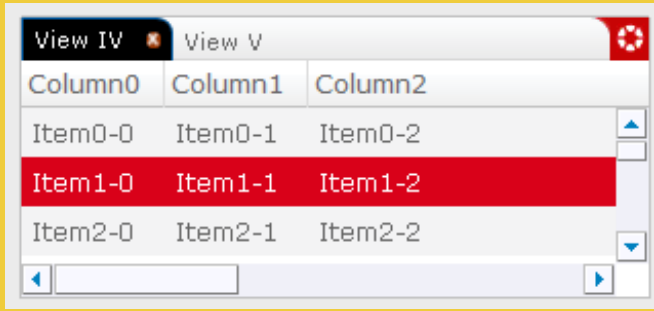
- 2 Beispiele im Plug-in *org.eclipse.rap.design.example*
  - "fancy" und "business"

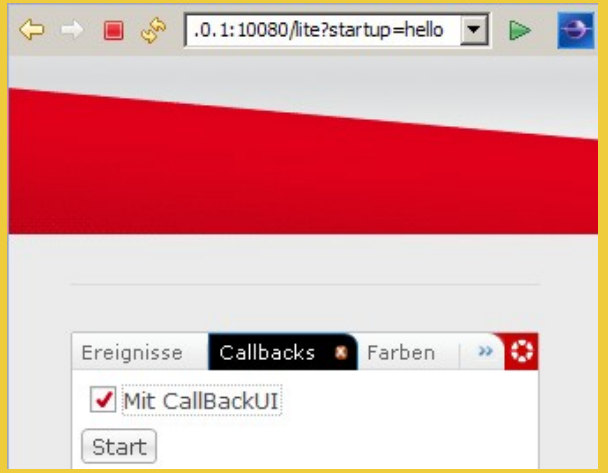
Ein eigenes Business gestalten

- Theme-Icons
- Workbench-Icons
- CSS
- Layout-Klassen



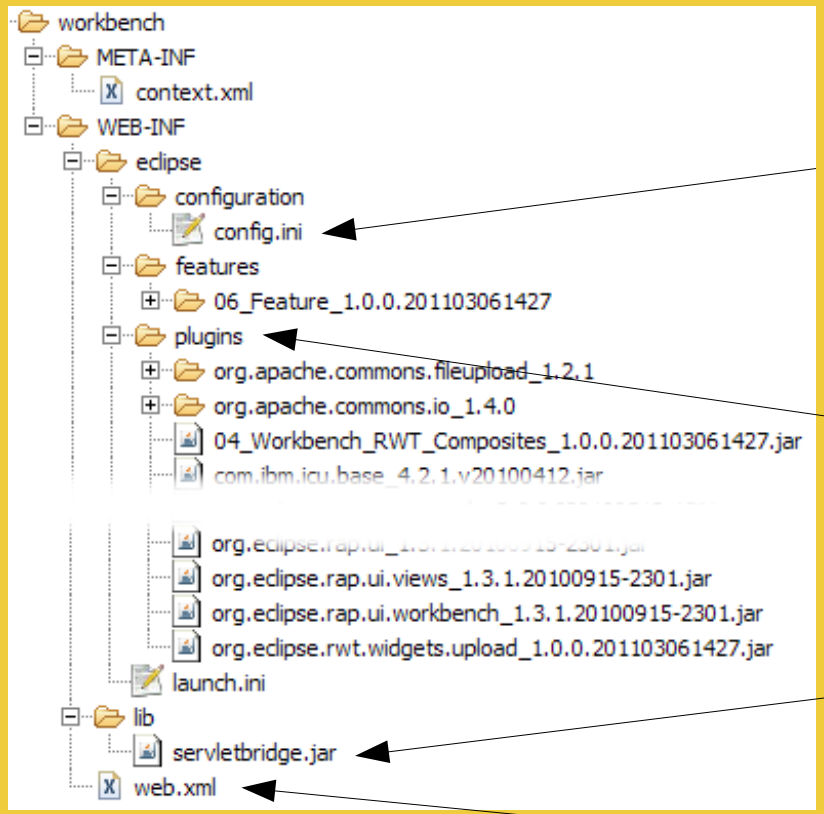






Deployment

Layout einer RAP Webanwendung



definiert zu startende OSGi-Bundles

die OSGi-Bundles

steuert OSGi-Framework und delegiert HTTP-Requests

konfiguriert Servlet-Bridge

Vorgehensweise

- Feature-Projekt und ServletBridge auschecken
  - psf-Datei aus Online-Help benutzen
- Feature-Umfang definieren
  - Inklusive Theme oder Design (wg Servlet-Name)
- Ggf. web.xml anpassen
  - Z.B. zusätzliche jsp-Dateien, consoleLog, console
- Intern Feature exportieren
  - PDE Export, JRE der Workbench verwenden
- Config-File erstellen
  - Mit ConfigIniCreator aus Feature-Projekt
- War-Datei bauen

Fragen?

Vielen Dank!

mail@borzechowski.de