SCC
INFORMATIONSSYSTEME

COMMON CONTROLS

ResourceFactory

Schaltflächen, Stylesheets und Farbtabellen für einen eigenen Painter erzeugen

SCC Informationssysteme GmbH

Version 1.6 · Stand: 14. Januar 2006

Impressum

HERAUSGEBER

SCC Informationssysteme GmbH
Drosselweg 13
64367 Mühltal
+49 6151 13 63 10
www.scc-gmbh.com

Inhalt

1 ResourceFactory

1.1 Worum es geht

Die ResourceFactory ist ein eigenständiges Werkzeug, das HTML-Ressourcen und einige Java-Quelltexte erzeugt. Sie liest XML-Ressourcendefinitionen und Vorlagendateien ein und erstellt daraus die Ressourcen: Schaltflächen, Menüeinträge, Stylesheets und anderes.

Das Werkzeug ist als Apache-Ant-Task umgesetzt und lässt sich damit in einen Ant-Build einbinden.

1.2 Stylesheet-Vorlagen verwenden

Es ist nicht ratsam, die Vorlagendateien aus der Framework-Distribution (Verzeichnis cc-framework/build/resources/templates) zu kopieren oder zu ändern. Sie ändern sich mit der nächsten Fassung des Frameworks höchstwahrscheinlich, weil neue Kontrollelemente neue Stile mitbringen. Wer die Vorlagen ändert, muss seine Änderungen bei jeder neuen Fassung von Hand zusammenführen – das ist meist nicht schwierig, aber vermeidbare Arbeit.

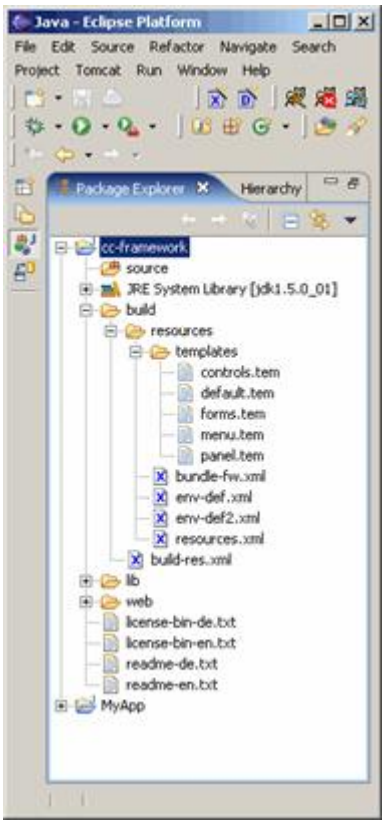
Dieses Dokument zeigt den empfohlenen Weg, das Erscheinungsbild einer Anwendung anzupassen. Wir benutzen dabei die Eclipse-IDE; mit anderen Entwicklungsumgebungen sollte es ebenso funktionieren.

2 Ein Eclipse-Projekt für MyApp einrichten

Der folgende Abschnitt zeigt, wie ein neues Common-Controls-Eclipse-Projekt für die Webanwendung MyApp eingerichtet wird.

2.1 Verzeichnisstruktur

Am besten installiert man die Common-Controls-Distribution parallel zum Verzeichnis der eigenen Anwendung. Dann genügt es, den Inhalt des Verzeichnisses cc-framework auszutauschen, um auf eine neue Fassung des Frameworks zu wechseln.



Die Distribution bringt alle nötigen Stylesheet-Vorlagen im Verzeichnis cc-framework/build mit. Diese Dateien können sich mit künftigen Fassungen des Frameworks ändern!

2.2 Framework-Ressourcen kopieren

Nun sind alle Dateien aus dem Verzeichnis cc-framework/web in das Verzeichnis MyApp/web zu kopieren. Das geht von Hand oder über ein Ant-Skript, das bei jedem Neubau des Projekts aufgerufen wird:

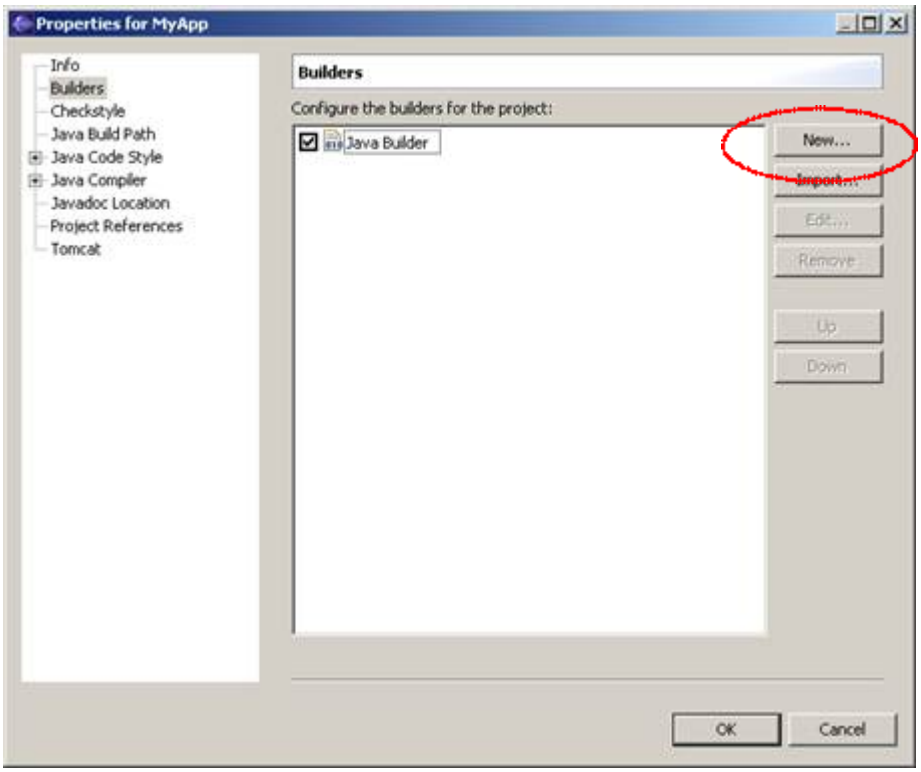
Ant-Skript copy-fw.xml

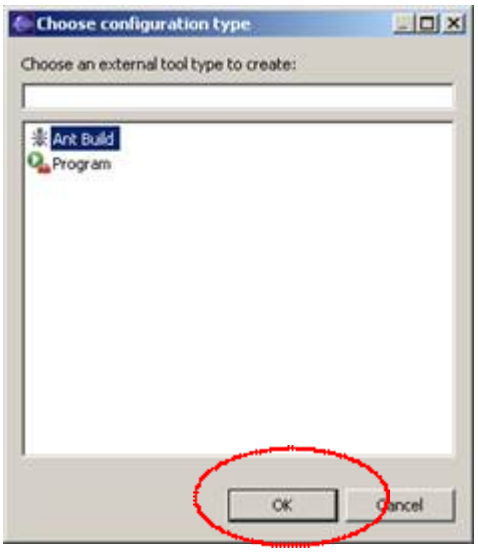
XML

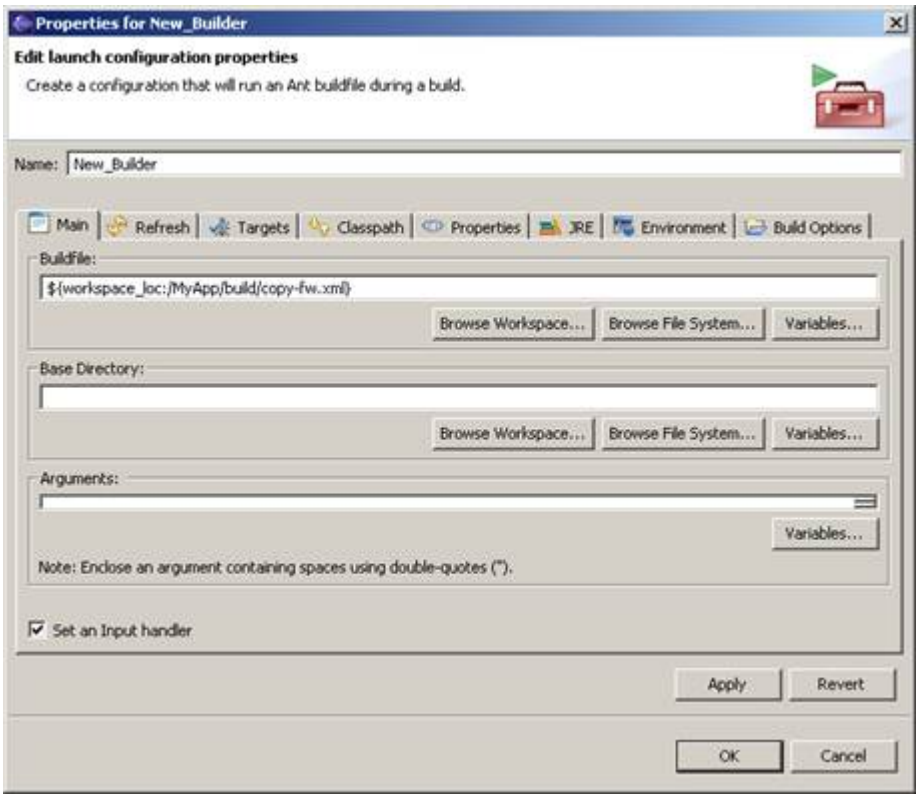
```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- ====================================================================== -->
```

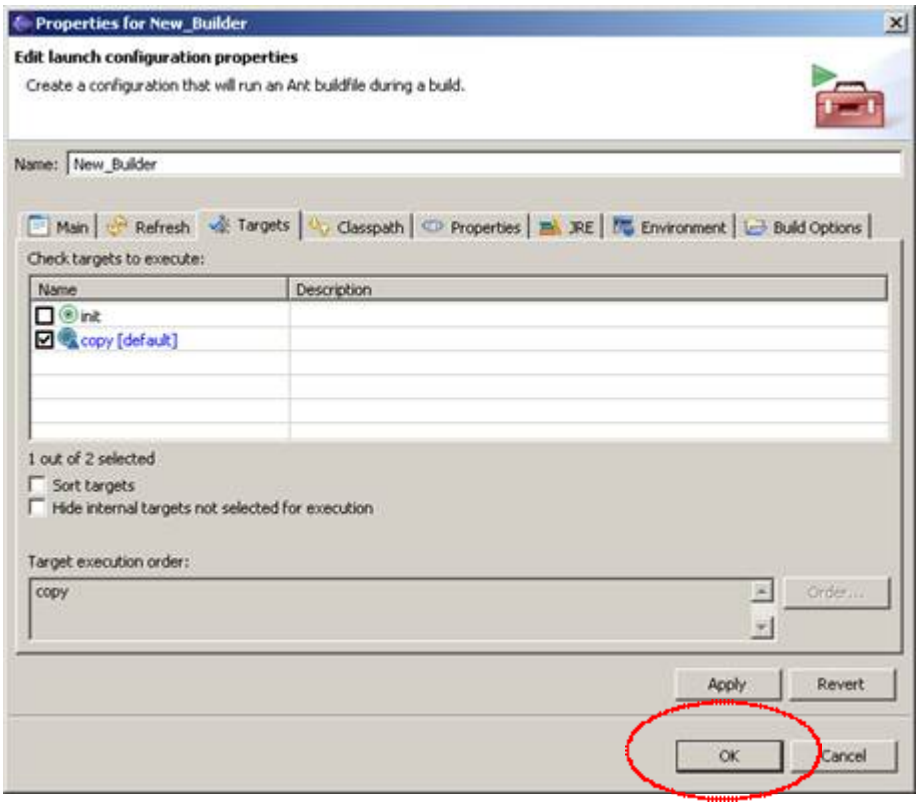
```
<!-- Framework Copy-Script                                                  -->
<!--                                                                        -->
<!-- @author     Harald Schulz                                              -->
<!-- @created    08.01.2005                                                 -->
<!-- @company    SCC Informationssysteme GmbH                               -->
<!--                                                                        -->
<!-- ====================================================================== -->

<project name="copy-fw" default="copy" basedir="..\">

    <description>
        Framework Copy Script
    </description>

    <target name="init">
        <property name="src.webapp"     value="${basedir}/web"/>
        <property name="fw.dir"         value="${basedir}/../cc-framework"/>
    </target>

    <target name="copy" depends="init">

        <!-- Delete old Framework Resources -->
        <delete dir="${src.webapp}/fw"/>

        <!-- Copy new Web Resources -->
        <copy todir="${src.webapp}" filtering="no">
            <fileset dir="${fw.dir}/web">
                <include name="**/*"/>
                <exclude name="fw/def2/**"/>
                <exclude name="META-INF/*.*"/>
            </fileset>
        </copy>
    </target>
</project>

```

Jetzt tragen wir in den Projekteigenschaften einen neuen Ant-Builder ein.


Properties for MyApp
Info
Builders
Checkstyle
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Project References
Tomcat
Builders
Configure the builders for the project:
Java Builder
New...
Import...
Edit...
Remove
Up
Down
OK
Cancel

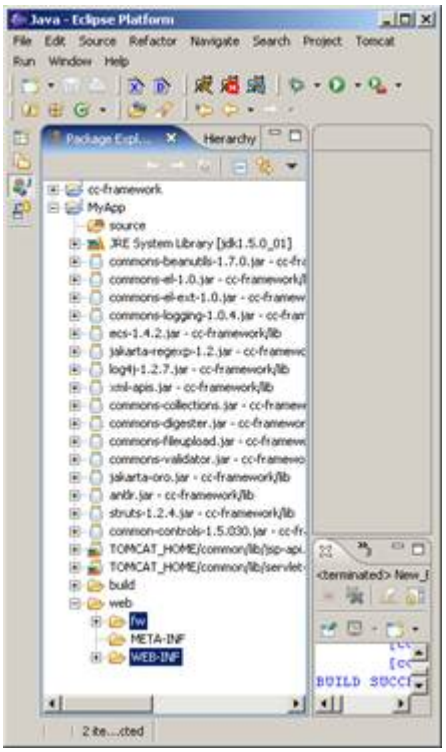

Choose configuration type
Choose an external tool type to create:
Ant Build
Program
OK
Cancel





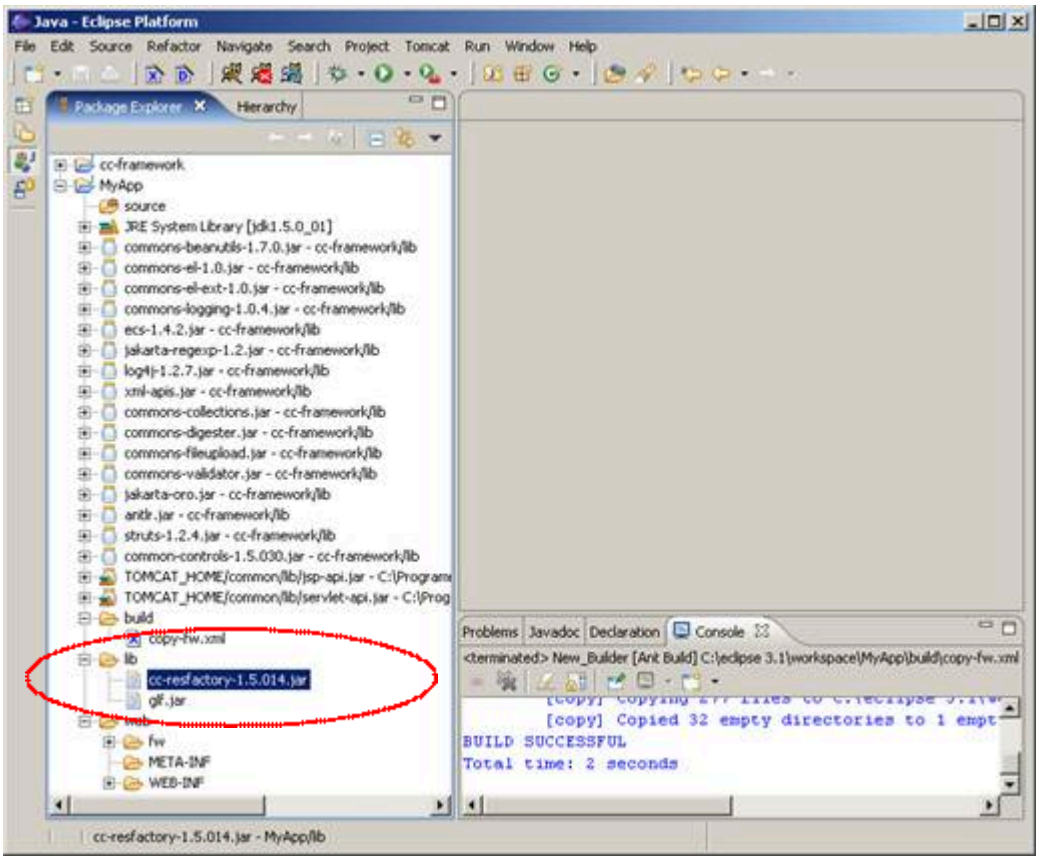
Damit kopiert die Eclipse-IDE bei jedem Neubau des Projekts MyApp die aktuellen Framework-Ressourcen.

Nach einer Projektaktualisierung sind die neuen Verzeichnisse web/fw und web/WEB-INF im Projekt MyApp zu sehen:



2.3 Die ResourceFactory einrichten

Das Ant-Build-Werkzeug braucht cc-resfactory-x.x.xxx.jar und glf.jar im Klassenpfad. In diesem Beispiel sollen die Ressourcen aus der Ant-Ansicht der Eclipse-IDE heraus entstehen; deshalb legen wir die Jar-Dateien in ein eigenes Projekt oder in das Verzeichnis lib des Projekts MyApp.



2.4 Web-Ressourcen erzeugen

Im nächsten Schritt kommt ein neues Ant-Skript in das Verzeichnis build. Es ruft die ResourceFactory auf und verarbeitet alle Ressourcendefinitionen im Verzeichnis build/resources.

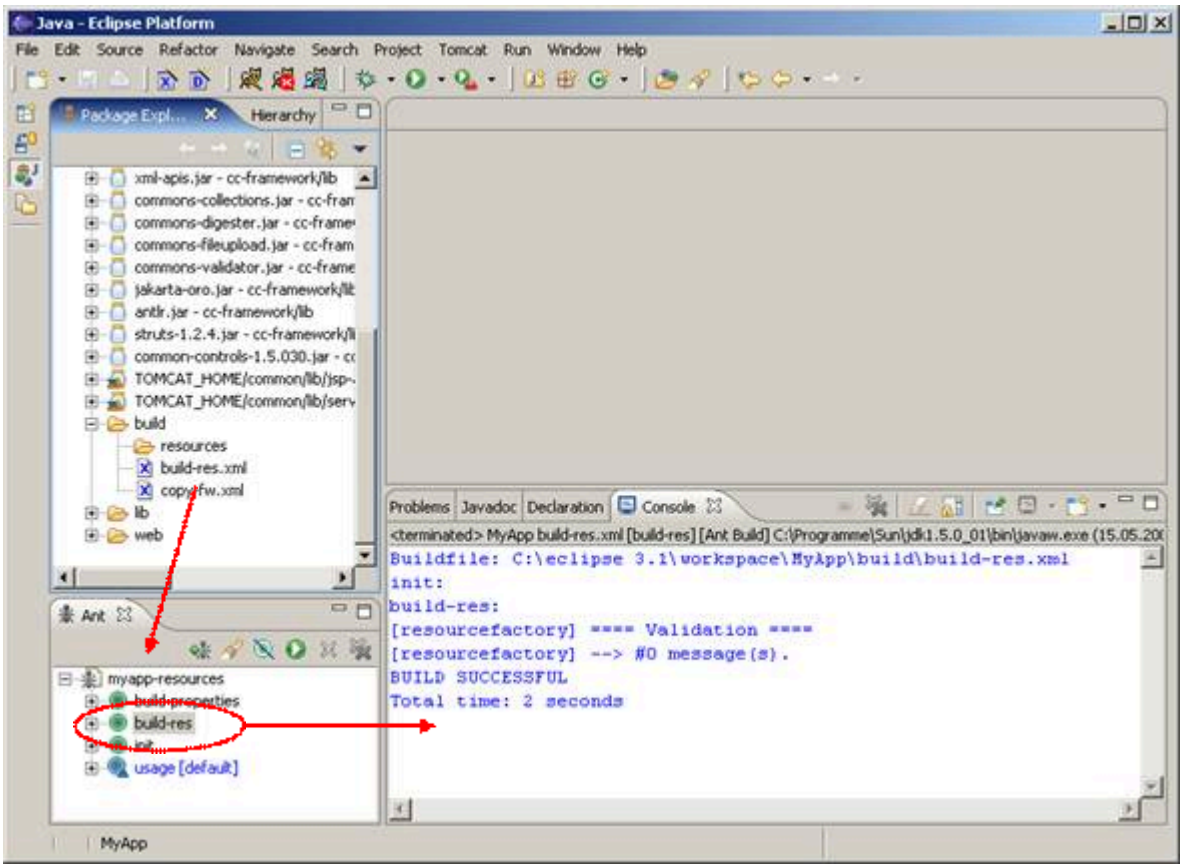
Hier der Quelltext des Ant-Skripts build-res.xml, das die ResourceFactory aufruft:

XML

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<project name="myapp-resources" default="usage" basedir="..\">

    <target    name="usage"
        description="prints a usage message">

        <echo message="###########"/>
        <echo message="# Build script for Imagebuttons"/>
        <echo message="###########"/>
        <echo message=""/>
        <echo message="available commands"/>
        <echo message=""/>
        <echo message="build-res        - creates all resources"/>
        <echo message="build-properties - creates only the property files"/>
        <echo message=""/>
        <echo message="###########"/>
    </target>

    <target name="init"
        description="initialises the build environment">

```

```
23          <!-- Classpath for ResourceFactory -->
24          <path id="resourcefactory.classpath">
25               <fileset dir="lib">
26                    <include name="*.jar"/>
27               </fileset>
28          </path>
29
30          <!-- Task definitions -->
31          <taskdef name="resourcefactory"
32               classname="com.cc.resourcefactory.ant.ResourceFactoryTask">
33               <classpath refid="resourcefactory.classpath"/>
34          </taskdef>
35      </target>
36
37      <!-- ================================================================= -->
38      <!-- Build                                                             -->
39      <!-- ================================================================= -->
40
41      <!-- ================================================================= -->
42      <target    name="build-res"
43              depends="init"
44          description="creates all resources">
45
46          <resourcefactory
47                destdir="${basedir}"
48                painter="def"
49               generate="true">
50
51               <fileset dir="${basedir}/build/resources">
52                    <include name="**/*.xml"/>
53               </fileset>
54          </resourcefactory>
55      </target>
56
57      <!-- ================================================================= -->
58      <target    name="build-properties"
59              depends="init"
60          description="creates only the property resource bundles">
61
62          <resourcefactory
63                destdir="${basedir}"
64                painter="def"
65               generate="true">
66
67               <fileset dir="${basedir}/build/resources/blue">
68                    <include name="**/bundle-app.xml"/>
69               </fileset>
70          </resourcefactory>
71      </target>
72  </project>
73
```

Das neue Ant-Skript ziehen wir einfach in die Ant-Ansicht von Eclipse. Danach genügt ein Klick auf die Aufgabe „build-res“, um die Ressourcen zu erzeugen.

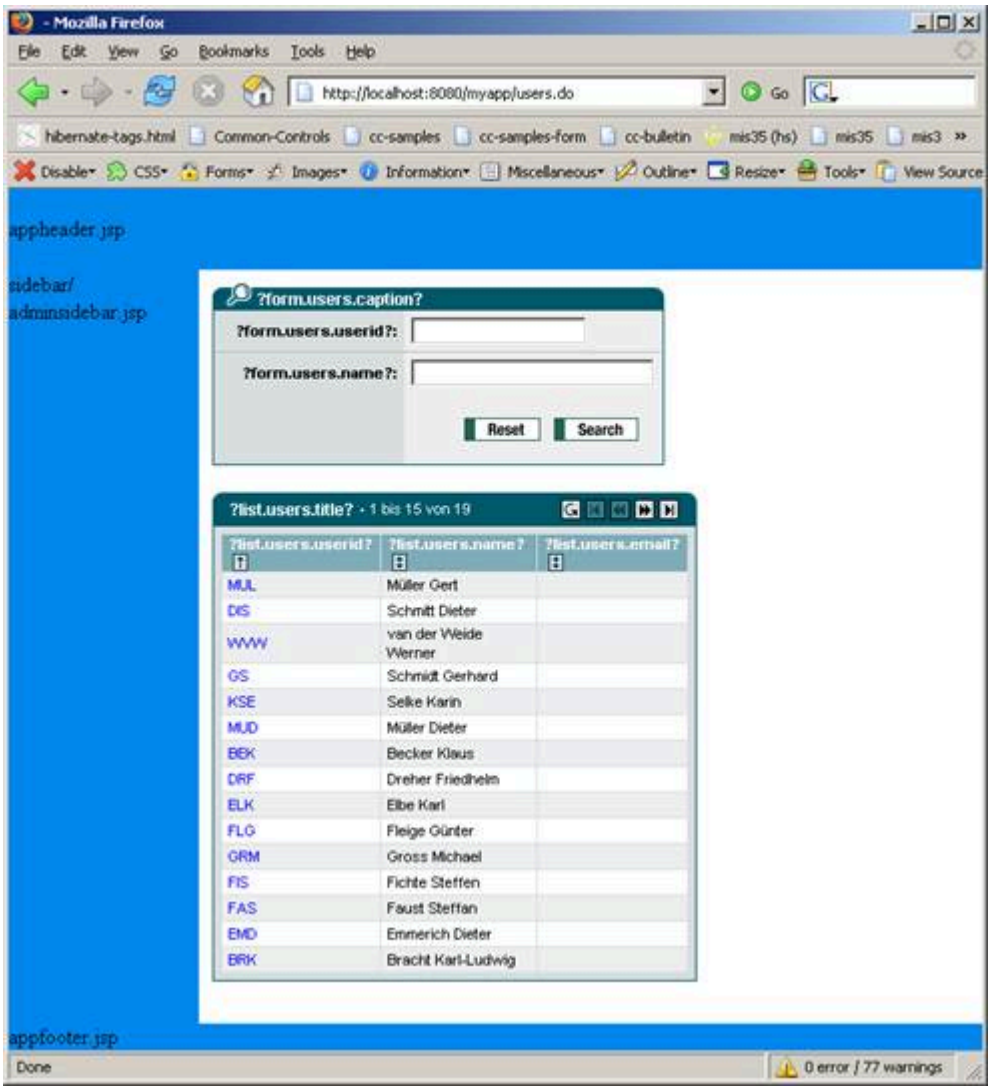


2.5 Die Anwendung vervollständigen

Die Anwendung braucht noch einige Dateien: web.xml, struts-config.xml, die JSP-Seiten und ein paar Struts-Actions. Alle nötigen Ressourcen liegen im ZIP-Archiv.

2.6 Die Anwendung starten

Starten wir die Anwendung und rufen im Browser die Adresse „http://localhost:8080/myapp/users.do“ auf, erscheint die Benutzerliste:


- Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/myapp/users.do
Go
hibernate-tags.html Common-Controls cc-samples cc-samples-form cc-bulletin mis35 (hs) mis35 mis3
Disable CSS Forms Images Information Miscellaneous Outline Resize Tools View Source
appheader.jsp
sidebar/
adminsidebar.jsp
?form.users.caption?
?form.users.userid?:
?form.users.name?:
Reset
Search
?list.users.title? - 1 bis 15 von 19
?list.users.userid?
?list.users.name?
?list.users.email?
MUL Müller Gert
DIS Schmitt Dieter
WWW van der Weide Werner
GS Schmidt Gerhard
KSE Selke Karin
MUD Müller Dieter
BBK Becker Klaus
DRF Dreher Friedhelm
ELK Elbe Karl
FLG Fleige Günter
GRM Gross Michael
FIS Fichte Steffen
FAS Faust Steffan
EMD Emmerich Dieter
BRK Bracht Karl-Ludwig
appfooter.jsp
Done
0 error / 77 warnings

3 Eigene Web-Ressourcen erzeugen

Jetzt sind alle Teile beisammen, um eigene Framework-Ressourcen zu erzeugen.

3.1 Eine eigene PainterFactory umsetzen

Wir beginnen mit einer eigenen Painter-Factory. Dieser Schritt ist nicht nötig, solange nur die Stylesheets ausgetauscht werden sollen – wohl aber, wenn die Bildregistrierungen des Frameworks geändert werden sollen. Genau das ist später vorgesehen.

Unsere MyPainterFactory leiten wir von der DefPainterFactory ab.

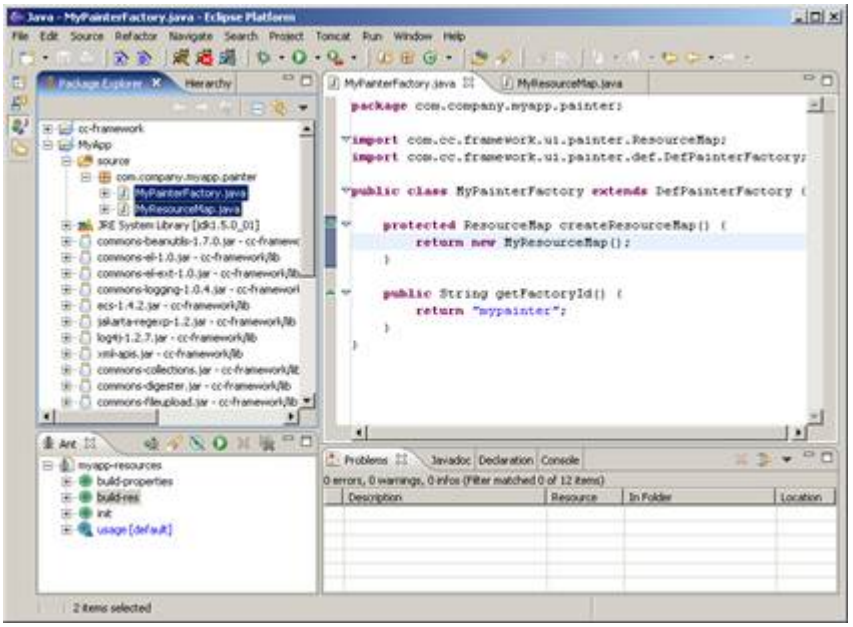
JAVA

```java
package com.company.myapp.painter;

import com.cc.framework.ui.painter.ResourceMap;
import com.cc.framework.ui.painter.def.DefPainterFactory;

public class MyPainterFactory extends DefPainterFactory {

	protected ResourceMap createResourceMap() {
		return new MyResourceMap();
	}

	public String getFactoryId() {
		return "mypainter";
	}
}

```

Die ResourceMap ist die zentrale Stelle, an der das Framework Bilder, Zeichenketten und Farben nachschlägt: Sie ordnet symbolischen Namen Ressourcenobjekte zu. Da später die Bildregistrierungen des Standard-Painters geändert werden sollen, brauchen wir eine eigene MyResourceMap:

JAVA

```java
package com.company.myapp.painter;

import com.cc.framework.ui.painter.def.DefResourceMap;

public class MyResourceMap extends DefResourceMap {

}

```



3.2 Die Rolle der DefPainterFactory

Die DefPainterFactory sollte die Basisklasse jedes eigenen Painters sein: Sie bringt alles mit, was zum Zeichnen der aufwendigen HTML-Kontrollelemente nötig ist – Liste, Baum, Formulare und die übrigen.

Eine PainterFactory lässt sich auch von Grund auf schreiben, empfehlenswert ist das aber nicht.

3.3 MyPainterFactory registrieren

Damit der neue Painter benutzt wird, muss er beim Start der Anwendung registriert werden – oder sobald ein Anwender eine neue Sitzung beginnt. Dafür bietet sich das Frontcontroller-Servlet an:

JAVA

```java
package com.company.myapp.servlet;

import javax.servlet.ServletException;

import org.apache.struts.action.ActionServlet;

import com.cc.framework.Globals;
import com.cc.framework.ui.painter.PainterFactory;
import com.company.myapp.painter.MyPainterFactory;

public class MyActionServlet extends ActionServlet {

    /**
     * @see org.apache.struts.action.ActionServlet#init()
     */
    public void init() throws ServletException {

        super.init();

        // Enable resource key translation for all GUI elements
        getServletContext().setAttribute(Globals.LOCALENAME_KEY, "true");

        // Register painters "global" and "html" for primitive HTML Elements
        PainterFactory.registerApplicationPainters(getServletContext());
```

```
25
26          // Register our custom painter
27          PainterFactory.registerApplicationPainter(getServletContext(),
28                  new MyPainterFactory());
29      }
30  }
31
```

Startet man die Anwendung und sieht sich den erzeugten HTML-Quelltext an, stehen dort die folgenden Framework-Einbindungen:

CODE

```
1  <!-- painter 'global' -->
2  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/common.js'></script>
3  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/environment.js'>
   </script>
4  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/utility.js'></script>
5  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/formatter.js'></script>
6  <!-- end -->
7
8  <!-- painter 'html' -->
9  <!-- end -->
10
11 <!-- painter 'mypainter' -->
12 <link rel='stylesheet' href='fw/def/style/default.css' charset='ISO-8859-1' type='text/css'>
13 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/functions.js'></script>
14 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/resourcemap.js'></script>
15 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/controls.js'></script>
16 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/tabset.js'></script>
17 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/tree.js'></script>
18 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/list.js'></script>
19 <!-- end -->
20
```

3.4 Property-Ressourcen erzeugen

Wir fangen mit einfachen Ressourcen an, den Dateien ApplicationResources.properties. Dazu legen wir die Ressourcendefinition res-bundle-app.xml in das Verzeichnis MyApp/build/resources:

JAVA

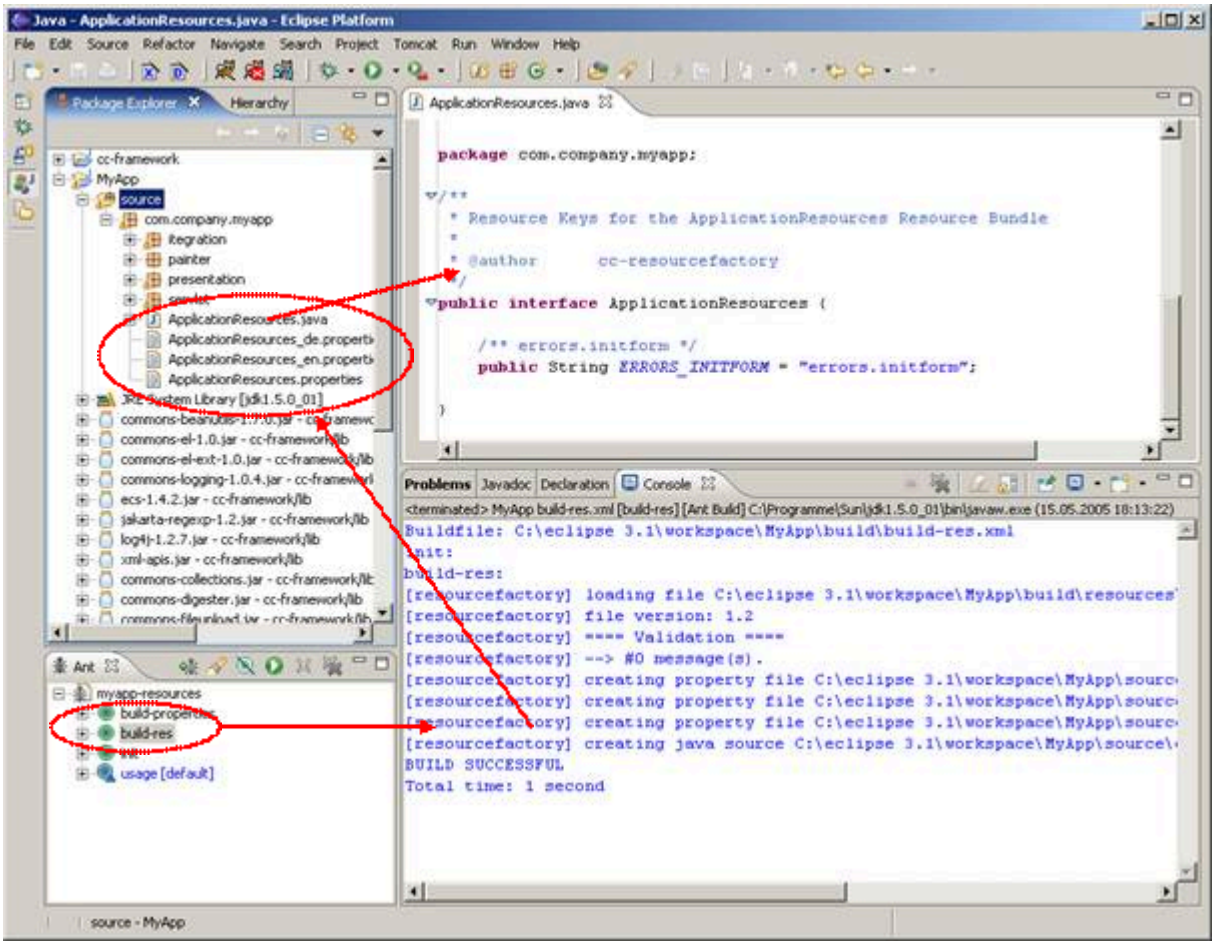
```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2
3  <!DOCTYPE
4  resource-factory
5  PUBLIC "-//common-controls//DTD ResourceFactory 1.2//EN"
6       "http://www.common-controls.com/dtds/resource-factory_1_2.dtd">
7
8  <resource-factory version="1.2">
9
10      <!-- ============================== -->
11      <!-- Resource bundle                -->
12      <!-- ============================== -->
13
14      <resources destdir="source">
15          <bundle
16              package="com.company.myapp"
17              name="ApplicationResources"
```

```
18              defaultlocale="en">
19
20              <!-- ######################### -->
21              <!-- # Directories          # -->
22              <!-- ######################### -->
23
24              <!--
25              We use localized button directories
26              -->
27
28              <resourcekey code="web" public="false">
29                   <resourcekey code="buttons">
30                        <value locale="de">images/buttons/de/</value>
31                        <value locale="en">images/buttons/en/</value>
32                   </resourcekey>
33              </resourcekey>
34
35              <!-- ########################## -->
36              <!-- # errors/messages       # -->
37              <!-- ########################## -->
38
39              <resourcekey code="errors" public="true">
40
41                  <!-- Application Error Messages -->
42                   <resourcekey code="initform">
43                        <value locale="de">Fehler beim initialisieren.</value>
44                        <value locale="en">Error while initializing form.</value>
45                   </resourcekey>
46              </resourcekey>
47
48              <!-- ########################## -->
49              <!-- # Forms                # -->
50              <!-- ########################## -->
51
52              <resourcekey code="form" public="false">
53
54                   <resourcekey code="users">
55                        <resourcekey code="caption">
56                             <value locale="de">Anwender - suchen</value>
57                             <value locale="en">User - search</value>
58                        </resourcekey>
59
60                        <resourcekey code="userid">
61                             <value locale="de">Kennung</value>
62                             <value locale="en">UserId</value>
63                        </resourcekey>
64
65                        <resourcekey code="name">
66                             <value locale="de">Name</value>
67                             <value locale="en">Name</value>
68                        </resourcekey>
69                   </resourcekey>
70              </resourcekey>
71
72              <!-- ########################## -->
73              <!-- # List Controls         # -->
74              <!-- ########################## -->
75
76              <resourcekey code="list" public="false">
77                   <resourcekey code="edit">
78                        <value locale="de">Bearbeiten</value>
79                        <value locale="en">Edit</value>
80                   </resourcekey>
81
```

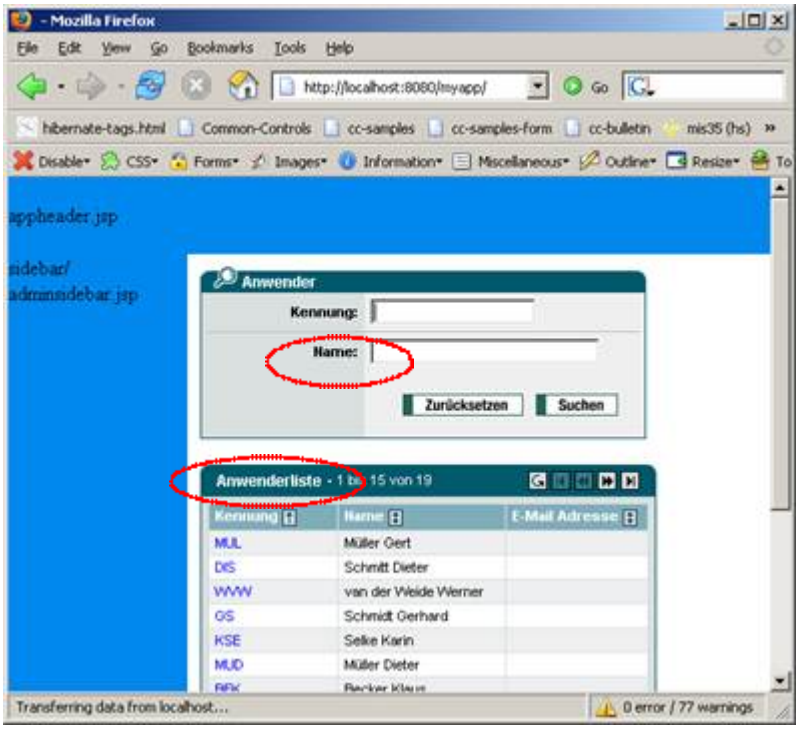
```
                    <resourcekey code="delete">
                         <value locale="de">Löschen</value>
                         <value locale="en">Delete</value>
                    </resourcekey>

                    <resourcekey code="users">
                         <resourcekey code="title">
                               <value locale="de">Anwenderliste</value>
                               <value locale="en">Userlist</value>
                         </resourcekey>

                         <resourcekey code="userid">
                               <value locale="de">Kennung</value>
                               <value locale="en">UserId</value>
                         </resourcekey>

                         <resourcekey code="name">
                               <value locale="de">Name</value>
                               <value locale="en">Name</value>
                         </resourcekey>

                         <resourcekey code="email">
                               <value locale="de">E-Mail Adresse</value>
                               <value locale="en">E-Mail address</value>
                         </resourcekey>
                    </resourcekey>
               </resourcekey>

          </bundle>
     </resources>
</resource-factory>

```

Die Ant-Aufgabe „build-res“ erzeugt daraus die folgenden Dateien:

- eine Schnittstellenklasse com.company.myapp.ApplicationResources mit allen Schlüsseln, die public="true" tragen
- übersetzte Properties-Dateien mit dem Namen ApplicationResources_<<locale>>.properties
- eine Datei ApplicationResources.properties für alle übrigen Sprachen → die Datei der Vorgabesprache



Starten wir die Anwendung jetzt, sind alle Platzhalter der Form ?xxx.yyyy.zzz? durch unsere neuen Werte ersetzt.



3.5 Eigene Farben festlegen

Nun soll eine eigene Farbpalette für die Anwendung entstehen. Sie dient später dazu, angepasste Stylesheets zu erzeugen.

Zuerst kopieren wir die Datei cc-framework/build/resources/env-def.xml nach MyApp/build/resources/env-custom.xml. Sie enthält alle symbolischen Farbnamen.

In dieser Datei ändern wir die Farbwerte so, dass sie unsere eigene Palette wiedergeben.

(→ Die Umgebungsdatei muss zuerst verarbeitet werden, weil sie die Variablen festlegt, die andere Ressourcendefinitionen benutzen. Das erreichen wir, indem sie in der alphabetischen Reihenfolge an erster Stelle steht – unter Microsoft Windows funktioniert das.)

In unserem Beispiel soll die Formularüberschrift orange werden. Dazu setzen wir die folgenden symbolischen Farben auf „orange" (die vollständige Liste steht unter http://www.common-controls.com/en/resources/docs/stylebook_def.html):

- form.color.bg.header
- form.color.border
- form.color.text.section
- list.color.bg.header
- list.color.border.body

Hier ein Ausschnitt aus der geänderten Datei env-custom.xml:

CODE

```
<definitions code="form" name="Forms: Formulare">
   <definitions code="font"  name="Fonts">
       <font code="header" family="${ctrlfont}" weight="bold"   size="8pt"/>
       <font code="body"   family="${ctrlfont}" weight="normal" size="8pt"/>
       <font code="label"  family="${ctrlfont}" weight="bold"   size="9pt"/>
   </definitions>
   <definitions code="color" name="Colortable">
       <color code="bg.field"       name="Background field"    value="${col18}"/>
       <color code="bg.header"      name="Background caption"  value="orange"/>
       <color code="bg.label"       name="Background label"    value="${col15}"/>
       <color code="bg.section"     name="Background section"  value="${col08}"/>
       <color code="bg"             name="Background color"    value="${col15}"/>
       <color code="border.item"    name="Border row"          value="${col10}"/>
       <color code="border.section" name="Border section"      value="orange"/>
       <color code="border"         name="Border"              value="orange"/>
      <color code="text.caption"   name="Text color caption"  value="${col26}"/>
       <color code="text.detail"    name="Text color detail"   value="${col26}"/>
       <color code="text.header"    name="Text color header"   value="${col26}"/>
       <color code="text.section"   name="Text color section"  value="${col02}"/>
   </definitions>
</definitions>
```

Bislang haben wir nur Farbdefinitionen geändert; Ressourcen entstehen dadurch noch keine.

3.6 Die ColorMap erzeugen

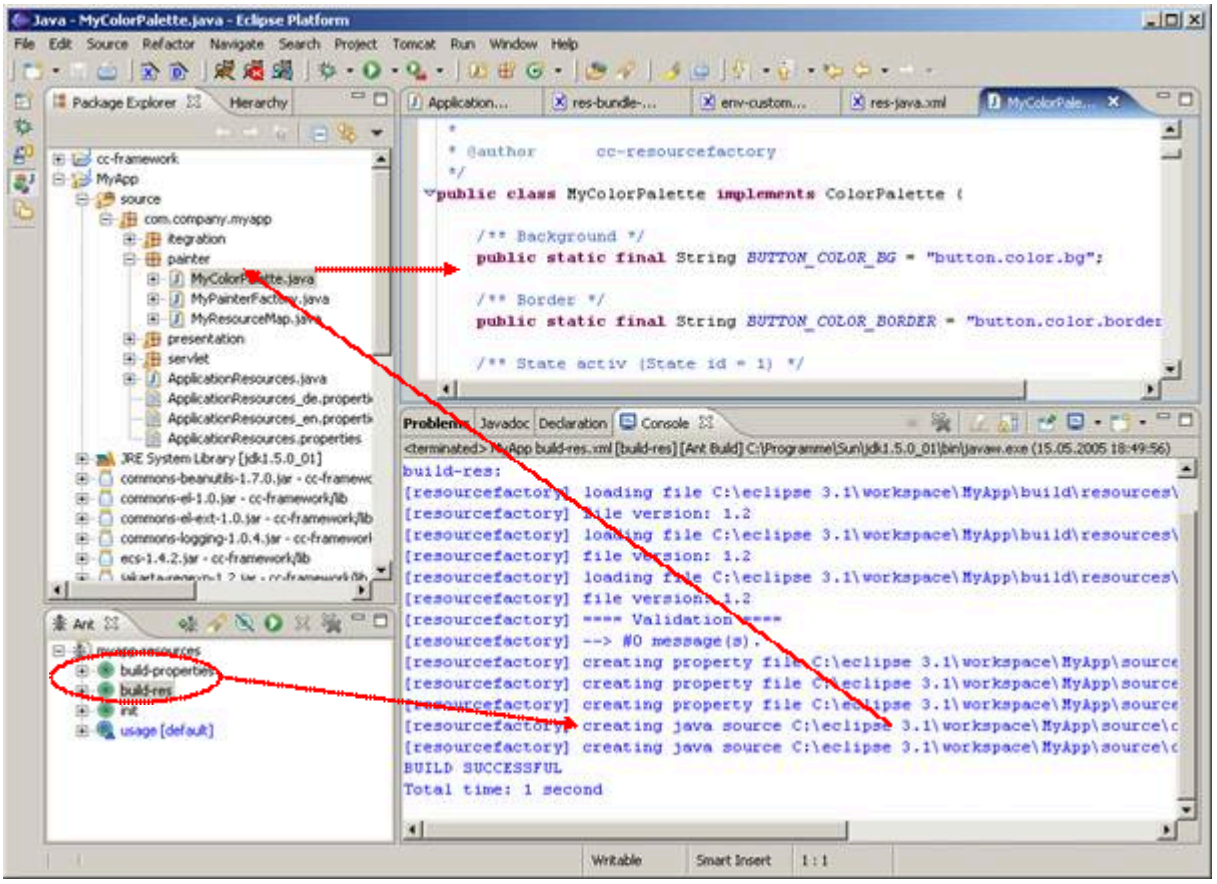
Jetzt erzeugen wir den Java-Quelltext für unsere eigene Farbpalette.

Dazu legen wir die neue Ressourcendefinition res-java.xml in das Verzeichnis MyApp/build/resources:

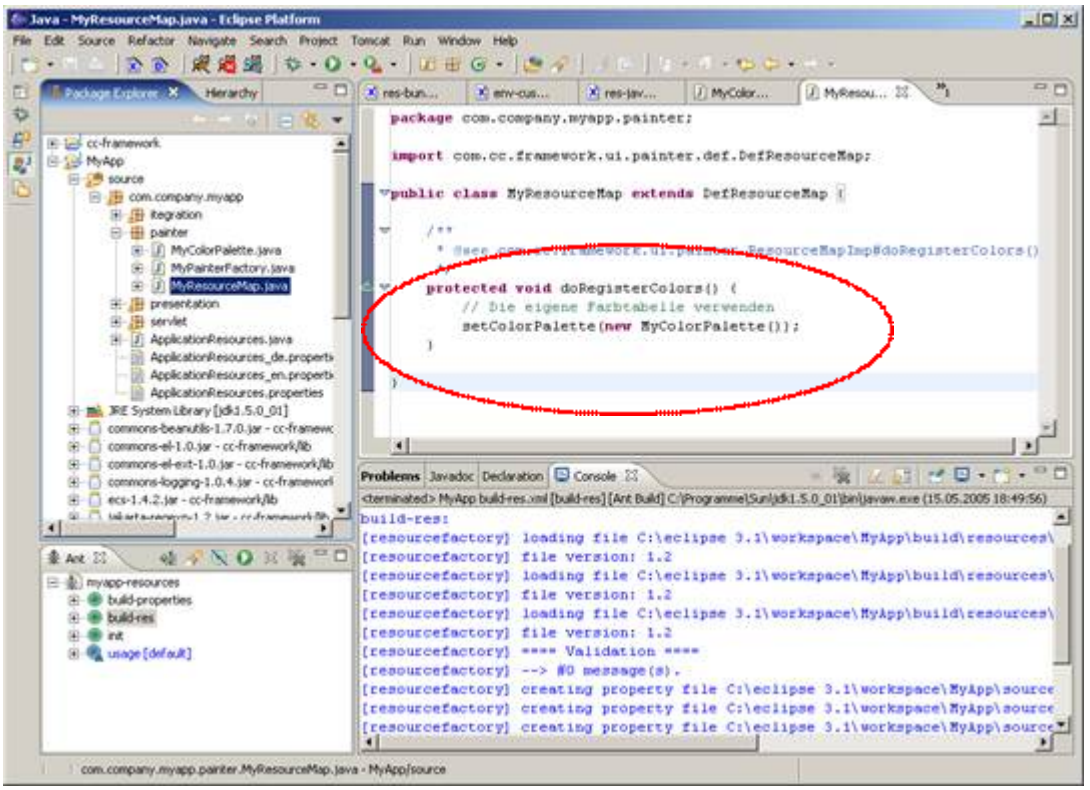
XML

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE resource-factory PUBLIC
    "-//common-controls//DTD ResourceFactory 1.2//EN"
    "http://www.common-controls.com/dtds/resource-factory_1_2.dtd">

<resource-factory version="1.2">

    <!-- ============================== -->
    <!-- Java Painter Classes           -->
    <!-- ============================== -->

    <resources destdir="source">
        <javasource
              type="colorpalette"
              package="com.company.myapp.painter"
              name="MyColorPalette"/>
    </resources>

</resource-factory>

```

Nach dem Lauf der Ant-Aufgabe „build-res“ finden wir die neue Klasse MyColorPalette.



Nun soll MyResourceMap unsere eigene Farbpalette benutzen. Dazu überschreiben wir die Methode doRegisterColors():



3.7 Schaltflächen erzeugen

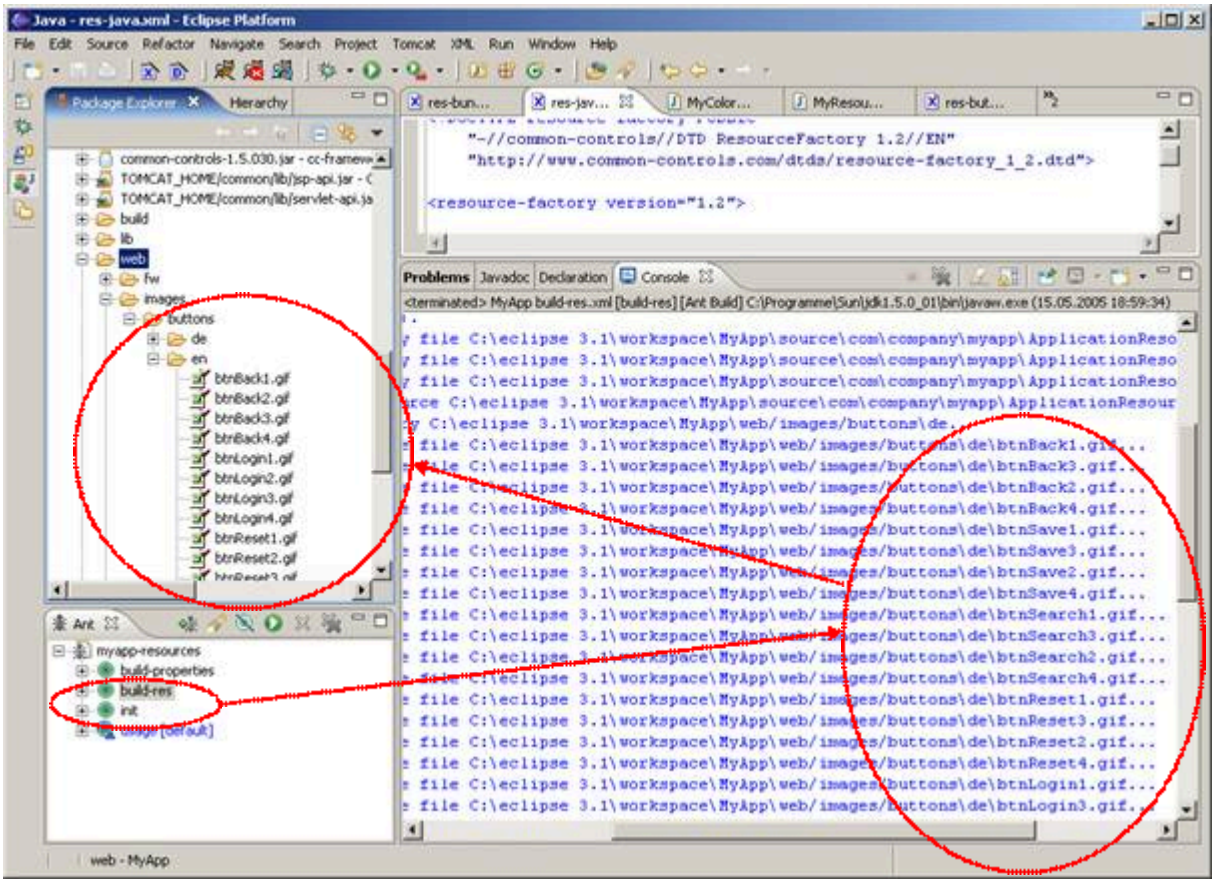
Jetzt sollen übersetzte Schaltflächen entstehen. Dazu legen wir die neue Ressourcendefinition res-buttons.xml in das Verzeichnis MyApp/build/resources.

XML

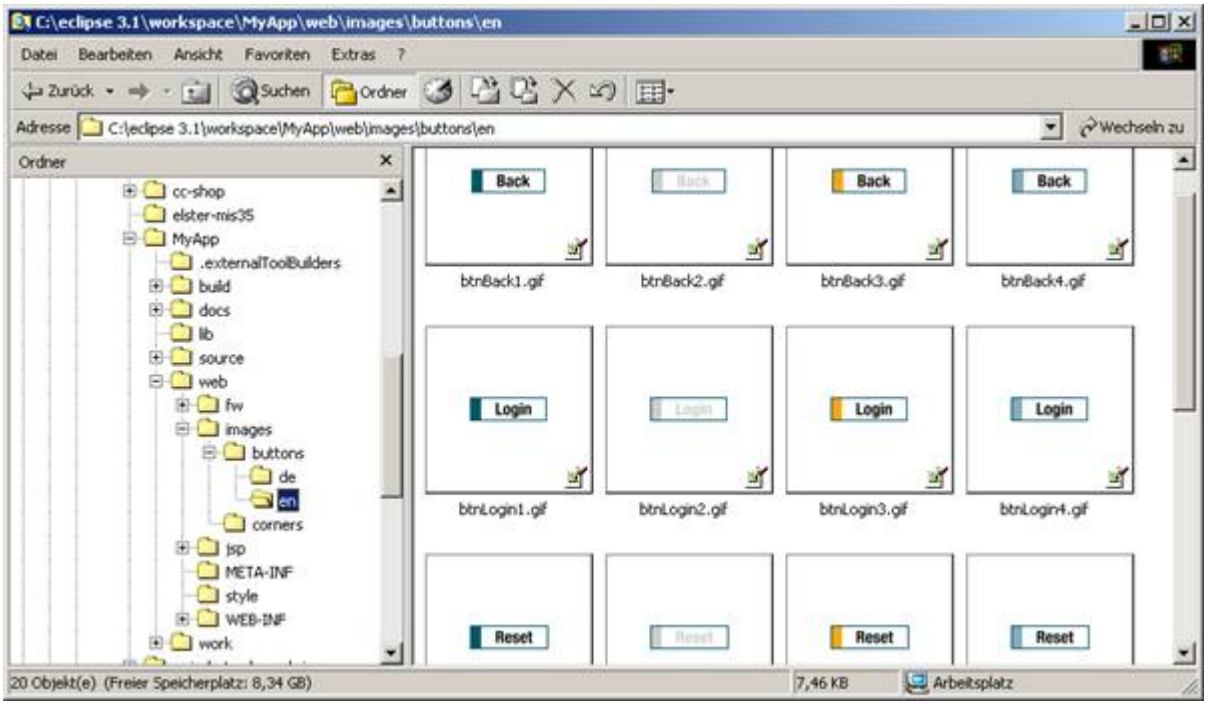
```xml
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE resource-factory PUBLIC
     "-//common-controls//DTD ResourceFactory 1.2//EN"
     "http://www.common-controls.com/dtds/resource-factory_1_2.dtd">

<resource-factory version="1.2">

     <!-- ============================== -->
     <!-- Button Resources               -->
     <!-- ============================== -->

     <resources destdir="web/images/buttons" overwrite="false">

          <!-- ============================== -->
          <!-- German resources               -->
          <!-- ============================== -->
          <resources destdir="de">
               <button name="Back"       label="Zurück"/>
               <button name="Save"       label="Speichern"/>
               <button name="Search"     label="Suchen"/>
               <button name="Reset"      label="Zurücksetzen"/>
               <button name="Login"      label="Anmelden"/>
          </resources>

          <!-- ============================== -->
```

```
27          <!-- English resources              -->
28          <!-- ============================== -->
29          <resources destdir="en">
30               <button name="Back"       label="Back"/>
31               <button name="Save"       label="Save"/>
32               <button name="Search"     label="Search"/>
33               <button name="Reset"      label="Reset"/>
34               <button name="Login"      label="Login"/>
35          </resources>
36      </resources>
37  </resource-factory>
38
```

Das Kennzeichen overwrite="false" weist die ResourceFactory an, eine Ressource nur dann neu zu erzeugen, wenn sie noch nicht vorhanden ist. Daran ist zu denken, wenn sich die Farben der ColorMap ändern – dann muss overwrite auf "true" stehen!



Das Farbschema der Schaltflächen lässt sich über die Farbkonstanten in env-custom.xml verändern; overwrite ist dabei auf "true" zu setzen.



3.8 Eigene Stylesheets erzeugen

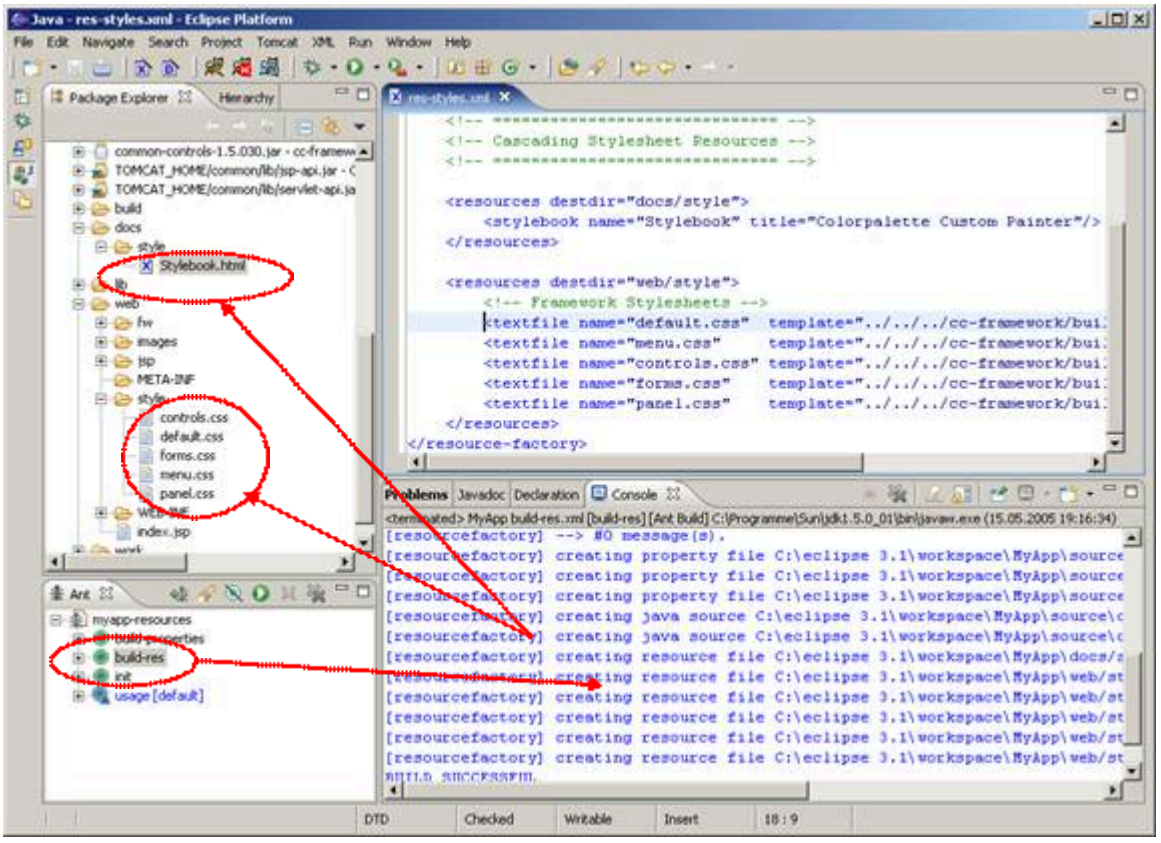
Für Stylesheets, die unser Farbschema benutzen, legen wir die neue Ressourcendefinition res-style.xml in das Verzeichnis MyApp/build/resources.

XML

```xml
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE resource-factory PUBLIC
       "-//common-controls//DTD ResourceFactory 1.2//EN"
       "http://www.common-controls.com/dtds/resource-factory_1_2.dtd">

<resource-factory version="1.2">

       <!-- ============================== -->
       <!-- Cascading Stylesheet Resources -->
       <!-- ============================== -->

       <resources destdir="docs/style">
              <stylebook name="Stylebook" title="Colorpalette Custom Painter"/>
       </resources>

       <resources destdir="web/style">
              <!-- Framework Stylesheets -->
              <textfile name="default.css"  template="../../../cc-framework/build/resources/templates/default.tem"/>
              <textfile name="menu.css"     template="../../../cc-framework/build/resources/templates/menu.tem"/>
              <textfile name="controls.css" template="../../../cc-framework/build/resources/templates/controls.tem"/>
              <textfile name="forms.css"    template="../../../cc-framework/build/resources/templates/forms.tem"/>
              <textfile name="panel.css"    template="../../../cc-framework/build/resources/templates/panel.tem"/>
       </resources>
</resource-factory>
```

```
26
```

Damit werden die Vorlagen aus der Framework-Distribution benutzt und daraus neue Stylesheets mit den Farbdefinitionen aus env-custom.xml erzeugt.



Jetzt muss die ResourceMap nur noch erfahren, dass sie unsere neuen Stylesheets benutzen soll und nicht die unter fw/def/style. Genauer: Sie soll unsere Fassung der Datei default.css benutzen, denn diese Datei bindet alle übrigen Stylesheets desselben Verzeichnisses ein:

Vorlage default.css

JAVA

```
1  /* ========================================
2  ** THIS FILE IS GENERATED WITH
3  ** THE CC-RESOURCEFACTORY TOOL.
4  **
5  ** USER...........: P001002
6  ** PAINTER........: def
7  ** BUILD-FACTORY..: v1.5.014
8  ** BUILD-DATE.....: Sun May 15 19:16:35 CEST 2005
9  ** BUILD-DIRECTORY: C:\eclipse 3.1\workspace\MyApp
10 ** BUILD-FILE.....: C:\eclipse 3.1\workspace\MyApp\build\resources\res-styles.xml
11 **
12 ** DO NOT MODIFY!
13 ** ========================================
14 */
15
16 /* Import the framework default styles */
17 @import "menu.css";
```

```
@import "controls.css";
@import "forms.css";
@import "panel.css";

```

Die DefPainterFactory löst den Namen der Datei default.css über die symbolische Konstante "fw.file.css.default" (= DefResources.FW_FILE_CSS_DEFAULT) auf. Es genügt also, den Wert dieser Konstante in unserer MyResourceMap zu ändern:

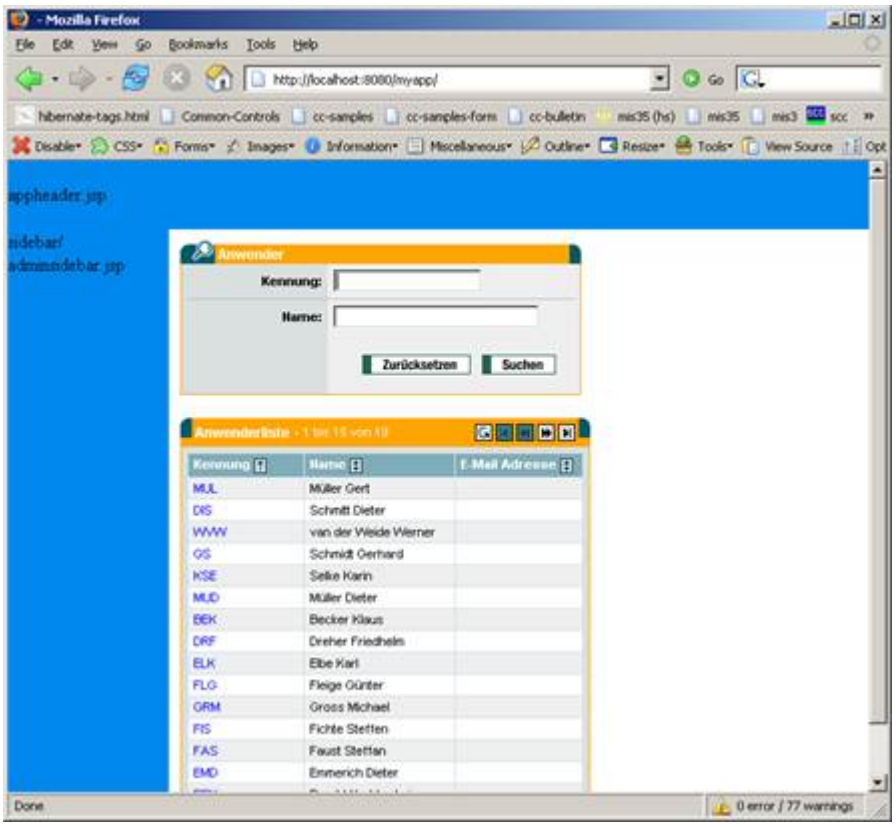
JAVA

```java
package com.company.myapp.painter;

import com.cc.framework.ui.painter.def.DefResourceMap;

public class MyResourceMap extends DefResourceMap {

    protected void doRegisterColors() {
        // Use our onw color map
        setColorPalette(new MyColorPalette());
    }

    protected void doRegisterStrings() {
        super.doRegisterStrings();

        // Register Cascading Stylesheets
        registerString(FW_FILE_CSS_DEFAULT, "style/default.css");
    }
}

```

→ eine Liste aller symbolischen Konstanten steht unter

http://www.common-controls.com/en/resources/docs/card/card_def.html

Starten wir die Anwendung erneut, ist unser neues Farbschema zu sehen.



Im HTML-Quelltext ist die geänderte Adresse zu sehen.

CODE

```
1  <!-- painter 'global' -->
2  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/common.js'></script>
3  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/environment.js'>
   </script>
4  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/utility.js'></script>
5  <script type='text/javascript' language='JavaScript' src='fw/global/jscript/formatter.js'></script>
6  <!-- end -->
7
8  <!-- painter 'html' -->
9  <!-- end -->
10
11 <!-- painter 'mypainter' -->
12 <link rel='stylesheet' href='style/default.css' charset='ISO-8859-1' type='text/css'>
13 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/functions.js'></script>
14 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/resourcemap.js'></script>
15 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/controls.js'></script>
16 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/tabset.js'></script>
17 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/tree.js'></script>
18 <script type='text/javascript' language='JavaScript' src='fw/def/jscript/list.js'></script>
19 <!-- end -->
20
```

3.9 Die Eckbilder austauschen

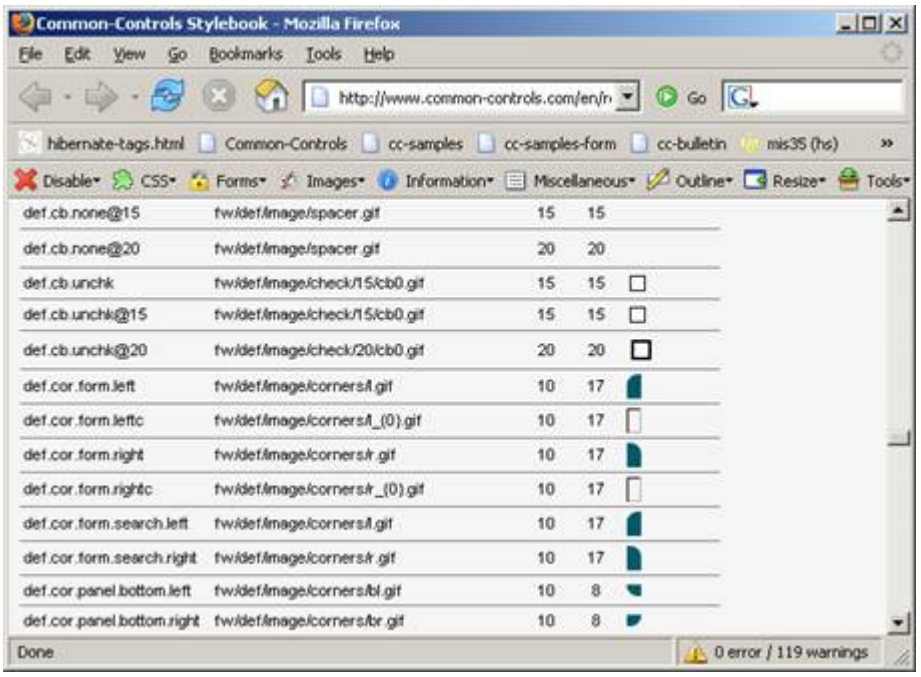
Nun sind die Eckbilder des Rahmens an der Reihe. Ihre symbolischen Namen lauten:

- def.cor.form.search.left

- def.cor.form.search.right
- def.cor.form.left
- def.cor.form.right

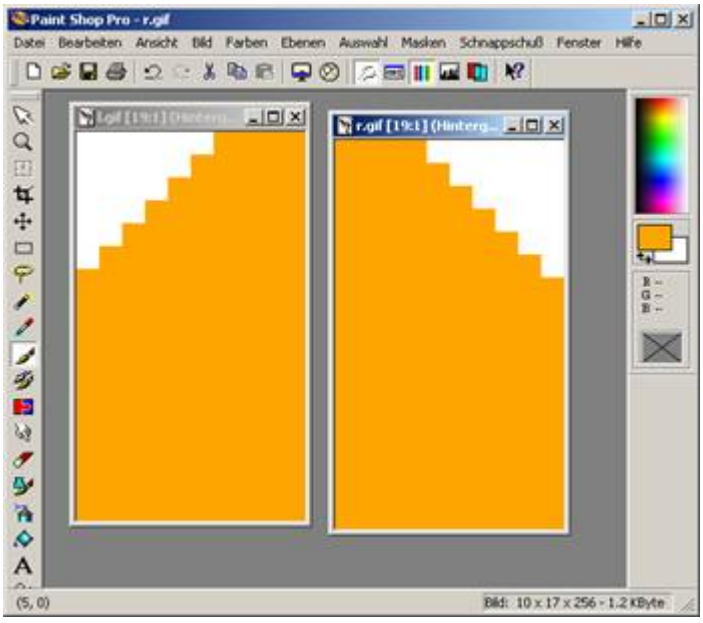
Die Übersicht aller symbolischen Konstanten des DefPainters steht unter:

http://www.common-controls.com/en/resources/docs/card/card_def.html



CODE

```
1  We will use the following simple corner images and copy them to MyApp/web/images/corners
```

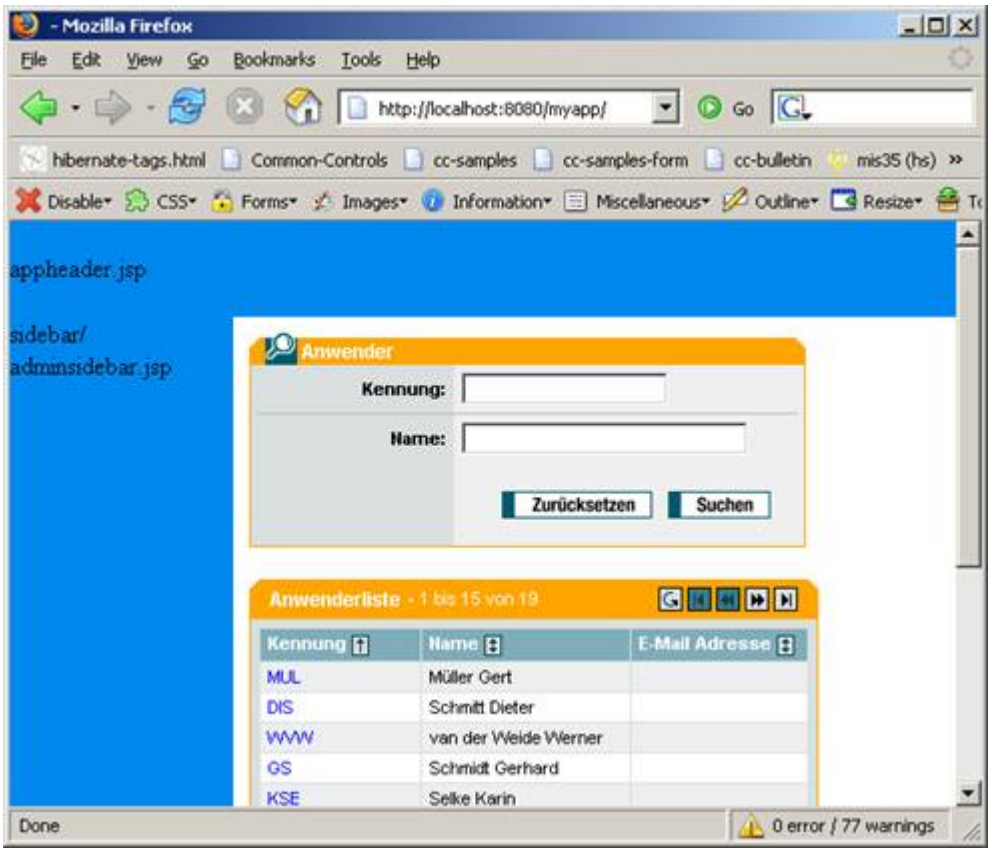


Jetzt registrieren wir unsere neuen Bilder in MyResourceMap:

JAVA

```java
package com.company.myapp.painter;

import com.cc.framework.ui.painter.def.DefResourceMap;

public class MyResourceMap extends DefResourceMap {

    protected void doRegisterColors() {
        // Use our onw color map
        setColorPalette(new MyColorPalette());
    }

    protected void doRegisterStrings() {
        super.doRegisterStrings();

        // Register Cascading Stylesheets
        registerString(FW_FILE_CSS_DEFAULT, "style/default.css");
    }

    protected void doRegisterImages() {
        super.doRegisterImages();

        registerImage(CORNER_FORM_RIGHT, new ImageModelImp("images/corners/r.gif", 10, 17));
        registerImage(CORNER_FORM_LEFT, new ImageModelImp("images/corners/l.gif", 10, 17));
        registerImage(CORNER_FORMSEARCH_RIGHT, new ImageModelImp("images/corners/r.gif", 10, 17));
        registerImage(CORNER_FORMSEARCH_LEFT, new ImageModelImp("images/corners/l.gif", 10, 17));
        registerImage(CORNER_TABLE_RIGHT, new ImageModelImp("images/corners/r.gif", 10, 17));
        registerImage(CORNER_TABLE_LEFT, new ImageModelImp("images/corners/l.gif", 10, 17));
    }
}

```

Nach einem Neustart der Anwendung sieht das Ergebnis so aus:



3.10 Eigene Stylesheets verwenden

TODO

4 Glossar

CC

Common-Controls