SCC
INFORMATIONSSYSTEME

COMMON CONTROLS

ResourceFactory

Generating buttons, style sheets and colour maps for your own painter

SCC Informationssysteme GmbH

Version 1.6 · As of: 14. Januar 2006

Impressum

PUBLISHED BY

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

Contents

1 ResourceFactory

1.1 About

The ResourceFactory is a stand-alone tool that creates HTML Resources and some Java Sources. It reads in some XML resource definition and template files and creates the resources (Buttons, Menu items, Cascading StyleSheets etc.).

The Tool is implemented as an Apache Ant Task. So it is possible to integrate it in an Ant build process.

1.2 Using CSS Templates

It its not advisable to copy or change the template files that come with the framework distribution (directory cc-framework/build/resources/templates). The template files will most likely change with the next framework release because there will be new styles for new control elements. If the templates are changed it is necessary to merge the changes with the next framework release manually (this should not be too difficult but it is still some avoidable work).

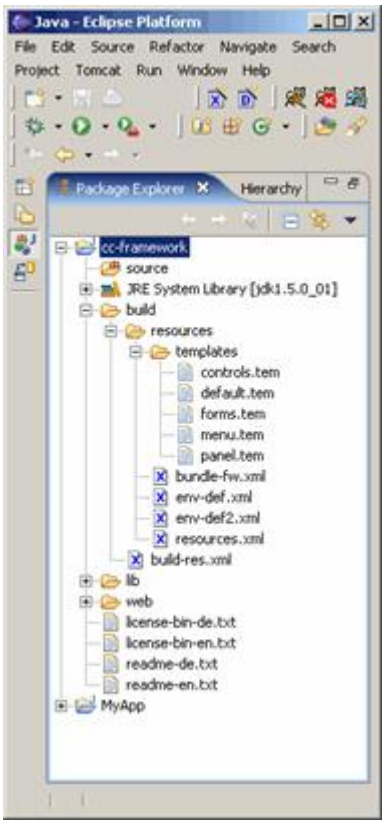
The following document shows the best practice to customize an Applications GUI Design. We use the Eclipse IDE in this project but the concepts should work with other IDE's, too.

2 Setting up an Eclipse Project for MyApp

The following section shows how to setup a new Common-Controls Eclipse Project for the Web Application MyApp

2.1 Directory Structure

It is best to install the common-controls distribution parallel to our application directory. This allows us to easily update to a new version of the Common-Controls framework by just exchanging the content of the cc-framework directory.



The distribution comes with all the necessary Cascading StyleSheet Template files in the cc-framework/build directory. These files may change with future framework version!

2.2 Copy Framework Resources

Now we have to copy all the files from the cc-framework/web directory into our MyApp/web directory. We can do this manually or we can use an Ant script that gets called whenever a project rebuild is done:

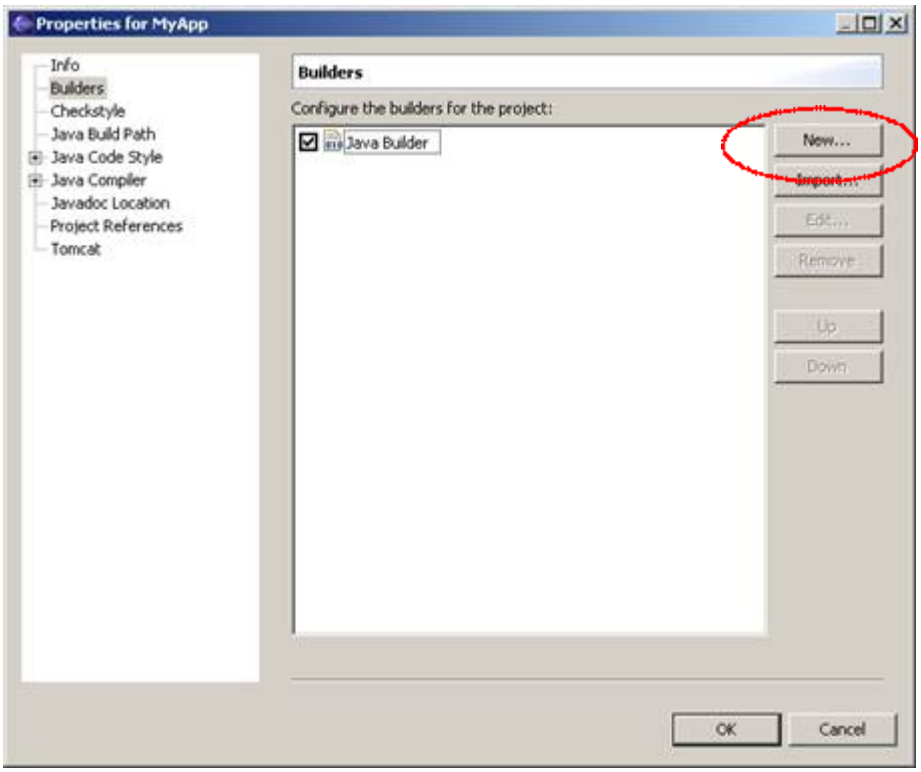
Ant Script copy-fw.xml

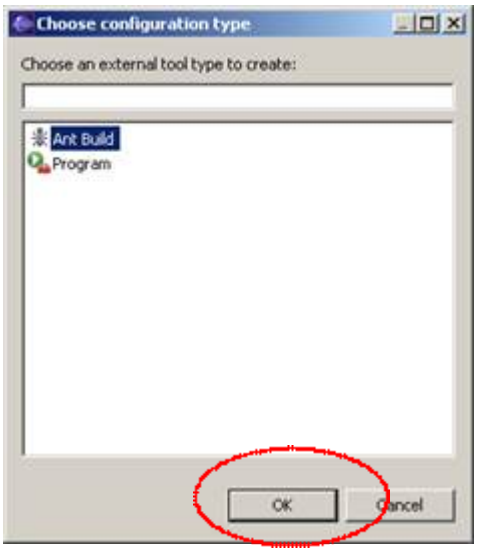
XML

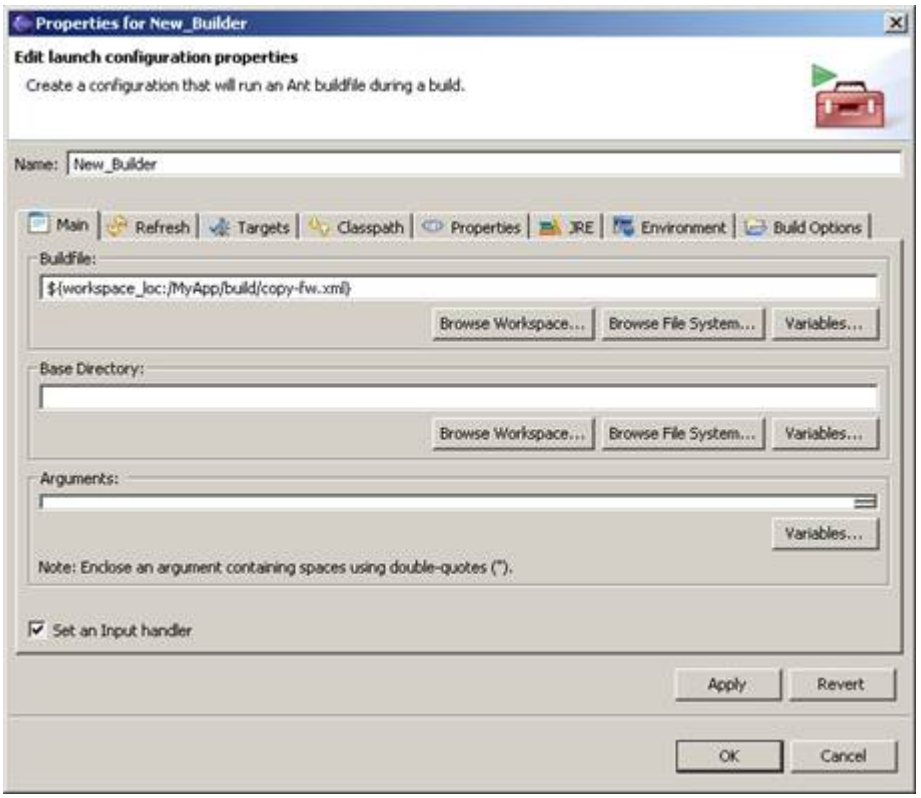
```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

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

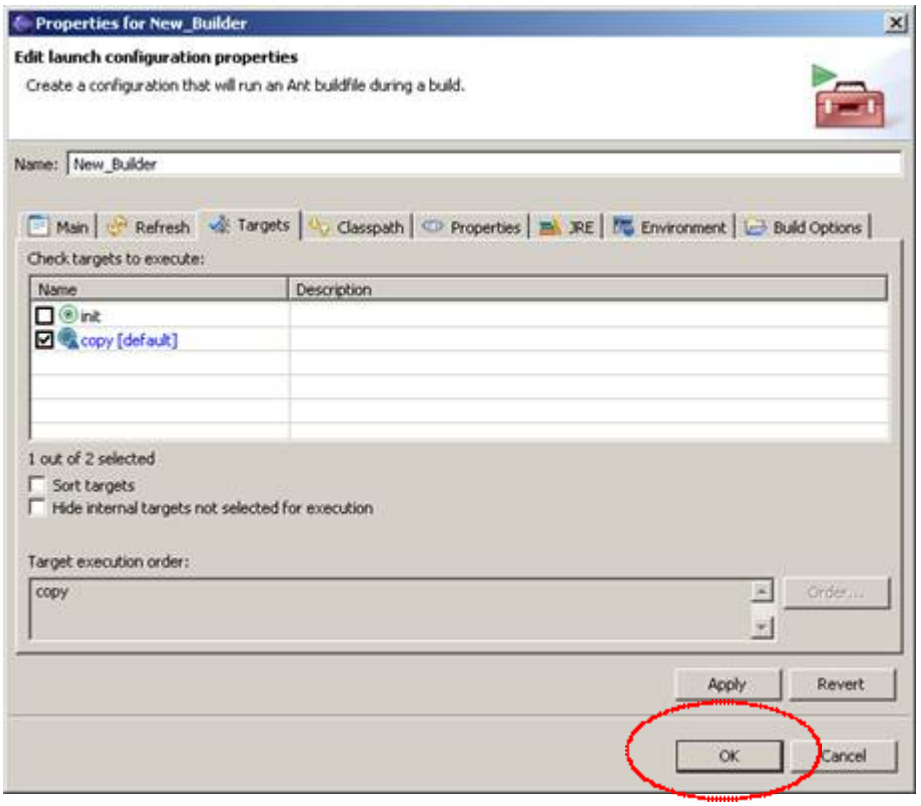
```
<!-- Framework Copy-Script                                                  -->
<!--                                                                        -->
<!-- @author     Harald Schulz                                              -->
<!-- @created    08.01.2005                                                 -->
<!-- @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>

```

Now we have to add a new Ant project builder to our project properties


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

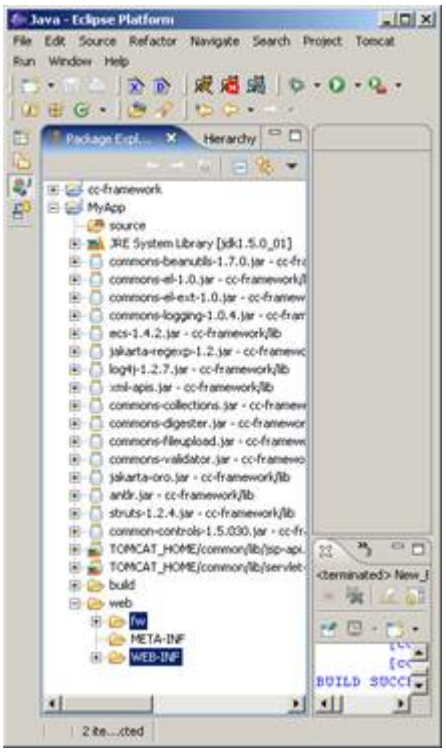

Choose configuration type
Choose an external tool type to create:
Ant Build
Program
OK
Cancel





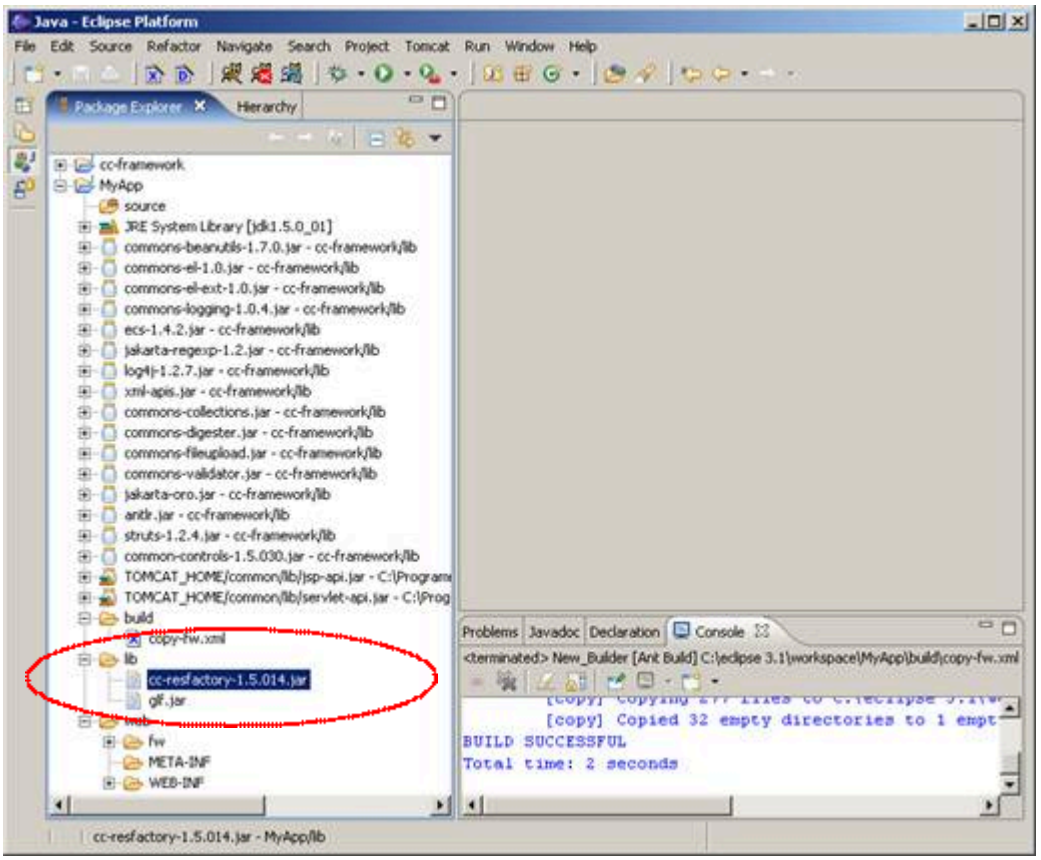
Now the Eclipse IDE will copy the latest framework resources whenever we do a rebuild of our MyApp project.

After a project refresh the new directories web/fw and web/WEB-INF should be visible in the MyApp project:



2.3 Setup ResourceFactory Tool

The Ant Build Tool needs the cc-resfactory-x.x.xxx.jar and glf.jar in the classpath. In this example we want to create our resources from within the Eclipse IDE's Ant view. So we copy the Jars in a separate Project or in the lib directory of our MyApp project.



2.4 Web Resource Generation

In the next Step we will add a new Ant Script to our build directory. This Ant Script will call the resource-factory tool and process all the resource definition files in the build/resources directory.

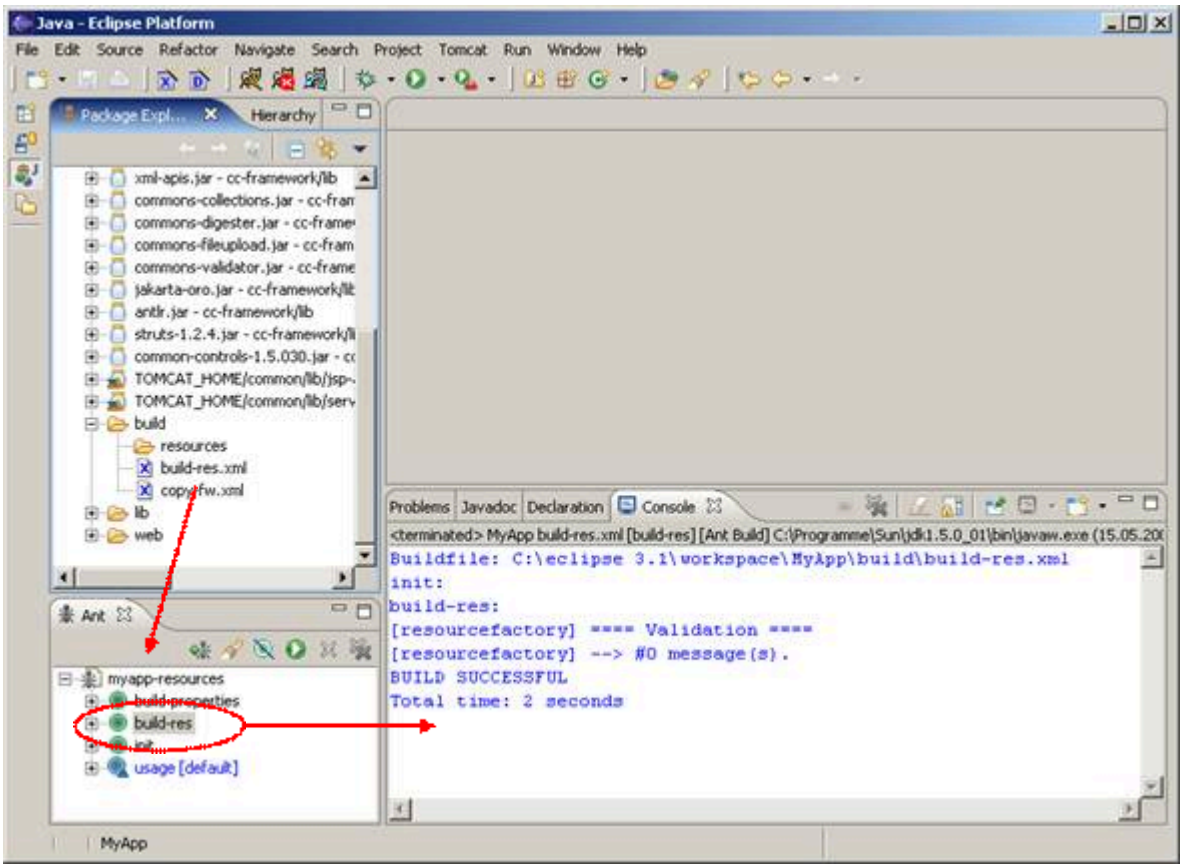
Here is the source for the build-res.xml ant script that calls resource-factory

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
```

We simply Drag&Drop this new Ant Script in the Eclipse Ant View. This allows us to start the Resource generation by simply clicking on the “build-res” Task in the Eclipse Ant View.

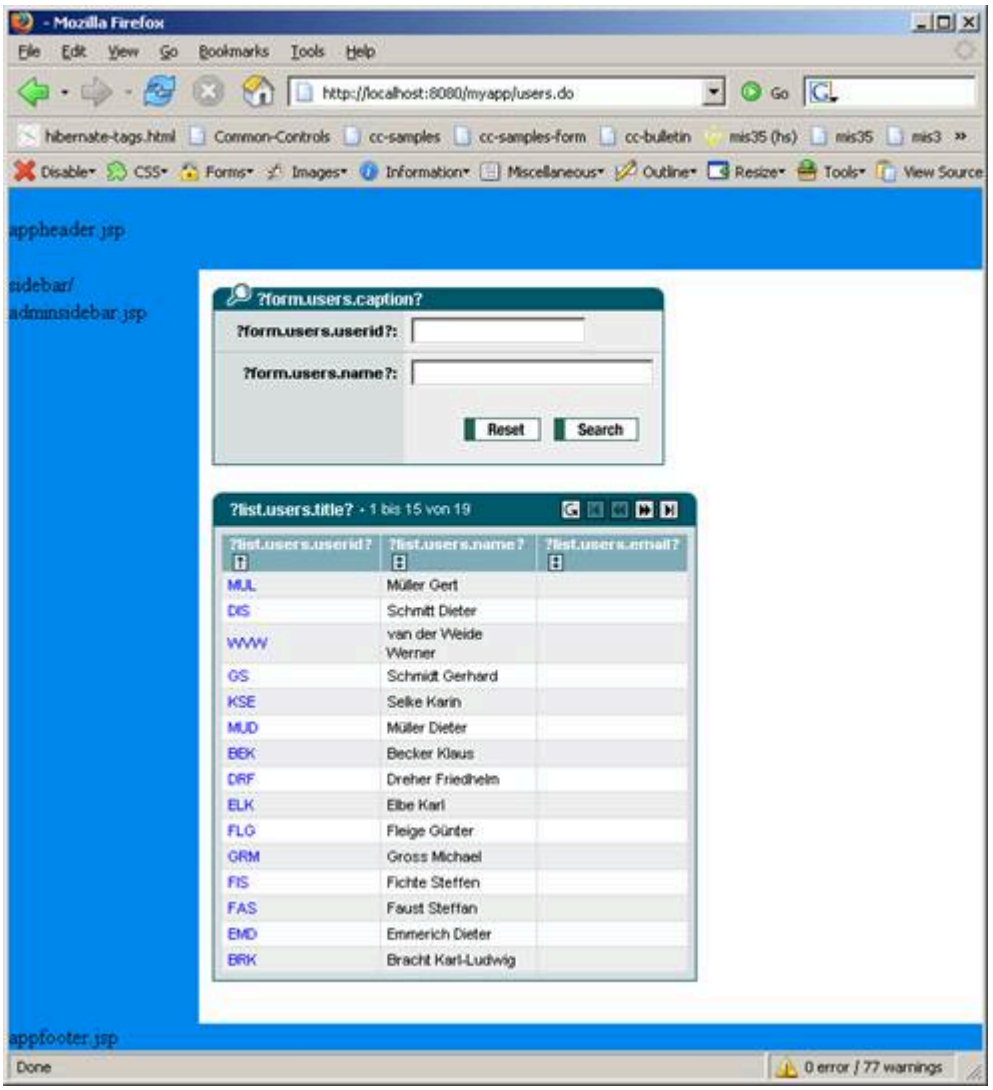


2.5 Completing the Application

The application needs some additional files like web.xml, struts-config.xml or JSP pages and Some Struts Actions. You can find all the necessary resources in the zip archive.

2.6 Starting the Application

When we start the application and type the URL “http://localhost:8080/myapp/users.do” in the browser window we should see the User Browse List:


- 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
WVW 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 Custom Web Resource Generation

Now we have all parts together to create our own custom framework resources

3.1 Implementing a Custom PainterFactory

We start by implementing our own painter factory. This step is not required when you just want to exchange the Cascading StyleSheet files but it is necessary when you want to change the framework Image registrations. That's what we want to do in a later step.

We derive our MyPainterFactory from the DefPainterFactory.

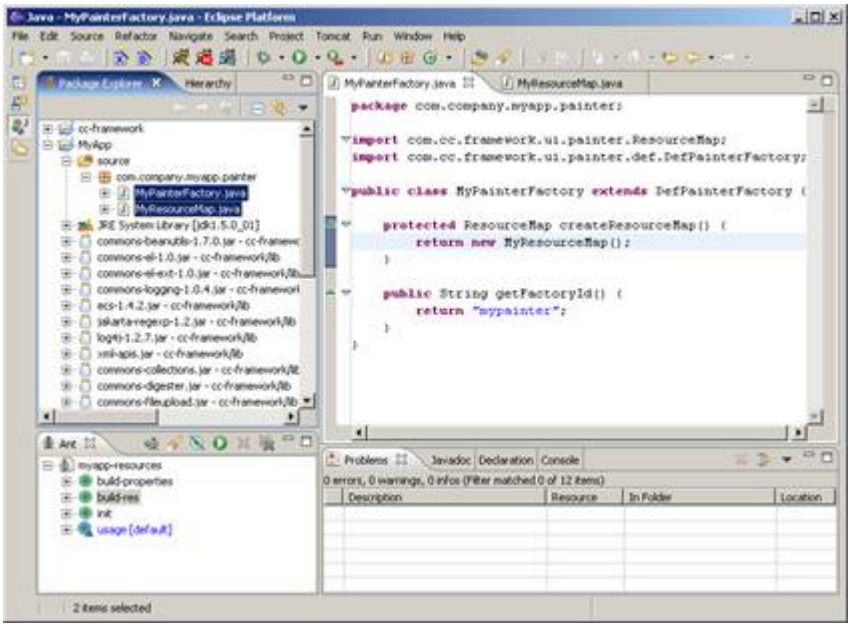
JAVA

```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";
	}
}

```

The ResourceMap is the central place for Image, String and Color lookups in the framework. It maps symbolic names to Resource Objects. We will change the image registrations of the default painter later. So we need our own MyResourceMap:

JAVA

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

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

public class MyResourceMap extends DefResourceMap {

}

```



3.2 Role of the DefPainterFactory

The DefPainterFactory should be the base class for every custom painter because it has all the functionality to render the complex HTML Controls (List, Tree, Forms etc.).

It is possible to build a PainterFactory from scratch but this is not recommended.

3.3 Register MyPainterFactory

If we want to use our new painter we have to register it at application start-up (or when a user creates a new session). This can be done in the Frontcontroller Servlet:

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
```

When you start the application and look at the generated HTML code you will see the following framework includes:

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 Creating Property Resources

We start with some simple resources. So lets begin with the ApplicationResources.properties files. We simply add the res-bundle-app.xml Resource definition file to our MyApp/build/resources directory:

JAVA

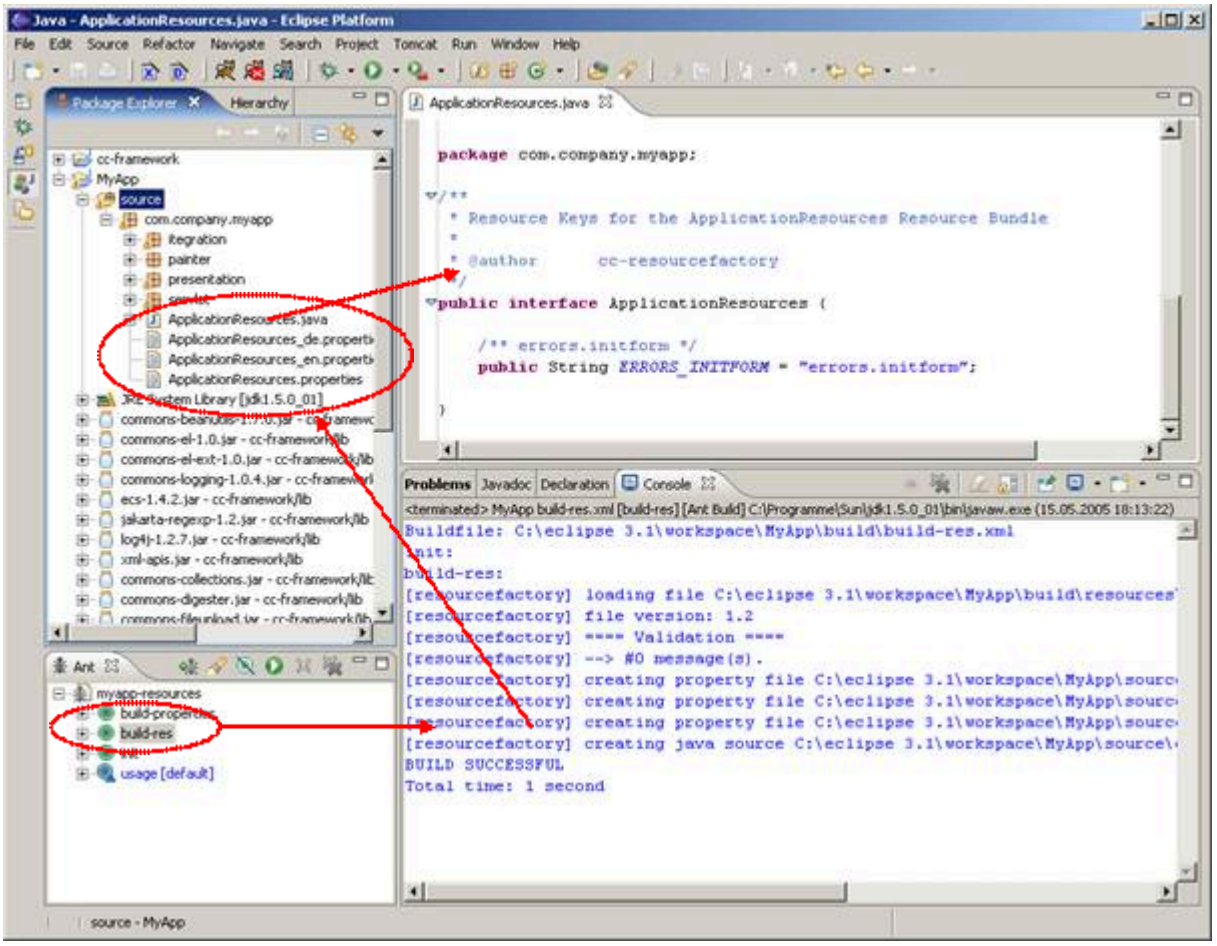
```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2
3  <!DOCTYPE
4  resource-factory
5  PUBLIC "-//common-controls//DTD ResourceFactory 1.2//EN"
6       "http://www.common-controls.com/dtds/resource-factory_1_2.dtd">
7
8  <resource-factory version="1.2">
9
10      <!-- ============================== -->
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
```

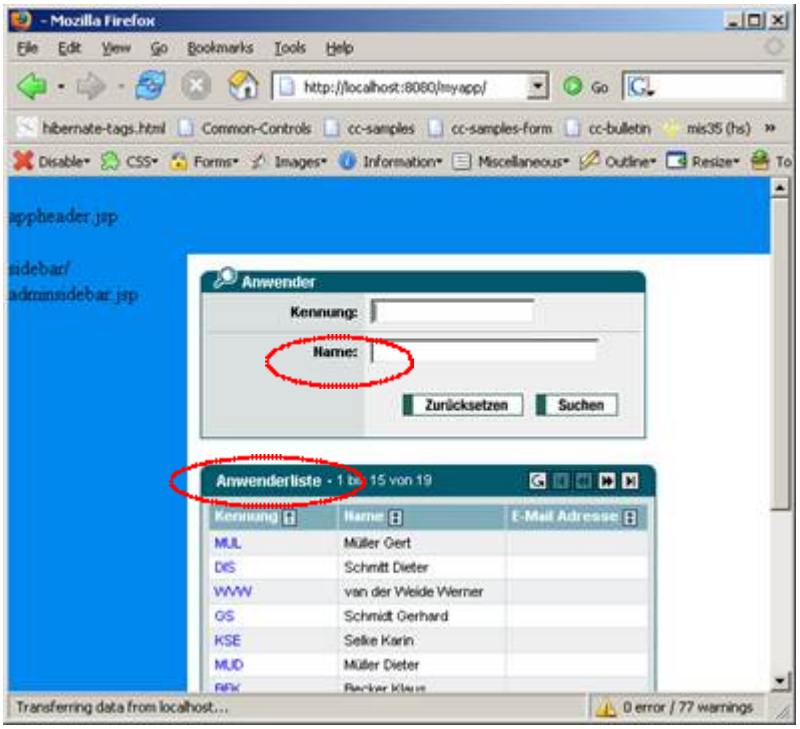
```
                    <resourcekey code="delete">
                         <value locale="de">Löschen</value>
                         <value locale="en">Delete</value>
                    </resourcekey>

                    <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>

```

When we now start the “build-res” Ant Task it will create the following files:

- an interface class com.company.myapp.ApplicationResources that contains all property keys with public=”true”
- localized properties files with the name ApplicationResources_<<locale>>.properties
- a default AplicationResources.properties file for all other languages à the default language file



When we now start our application we can see that all the ?xxx.yyyy.zzz? mark-ups have been replaced by our new property values



3.5 Declaring Custom Colors

Now we want to define our own application color palette. We will use this palette later to create our customized Cascading StyleSheets.

We will first copy the cc-framework/build/resources/env-def.xml file and rename it to MyApp/build/resources/env-custom.xml. This file contains all symbolic color names.

We will edit this file and change the color values to reflect our own custom palette.

(à The environment xml file must be parsed first because it defines the variables that are used by some other resource definition files. We archive this by making it the first file in the alphabetical order. This works on Microsoft Windows systems)

In our Example we want an orange frame title. So we change the following symbolic colors to “orange” (for a complete list see 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

Here is an excerpt of the changed env-custom.xml file

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>
```

So far we have changed only the color definitions. This will generate no resources.

3.6 Creating the ColorMap

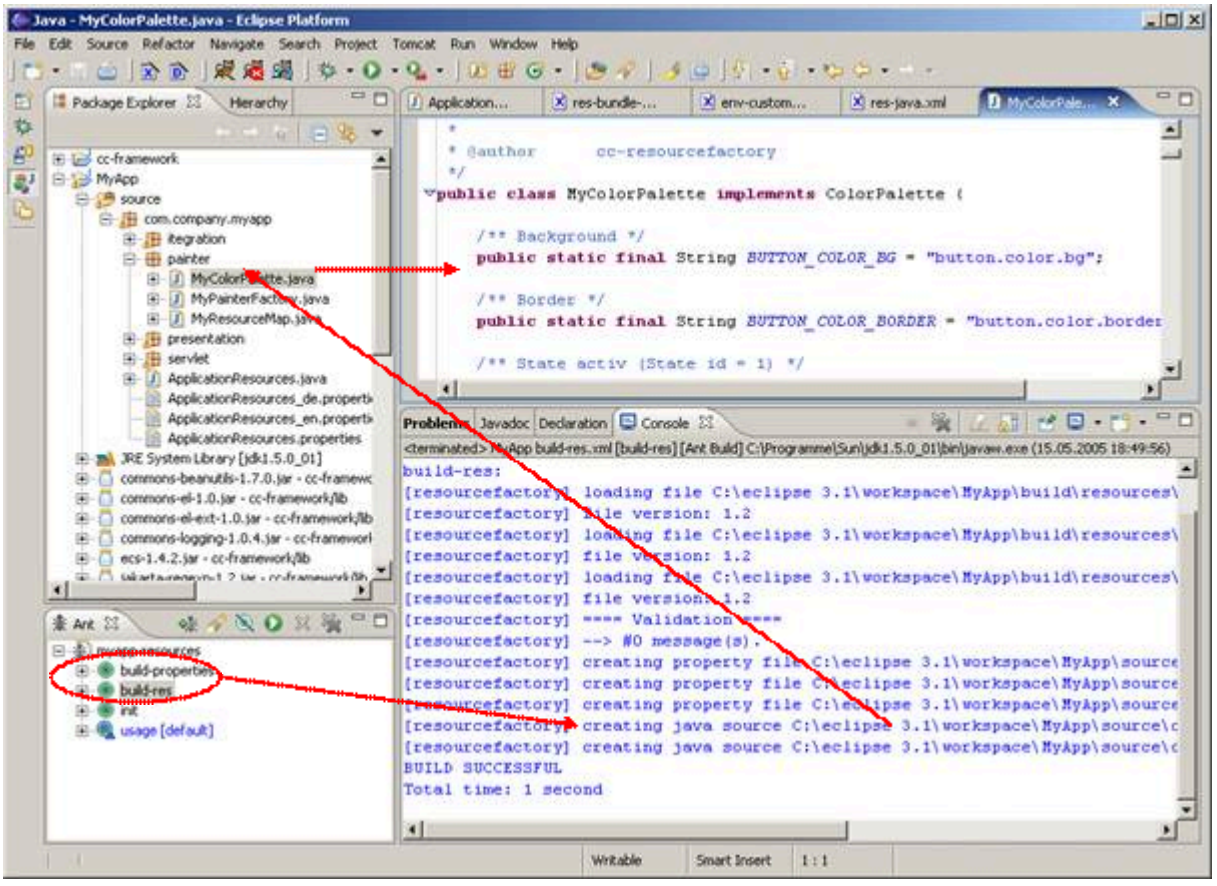
No we create the java source code for our custom color palette.

We add the new resource definition file res-java.xml to our MyApp/build/resources directory:

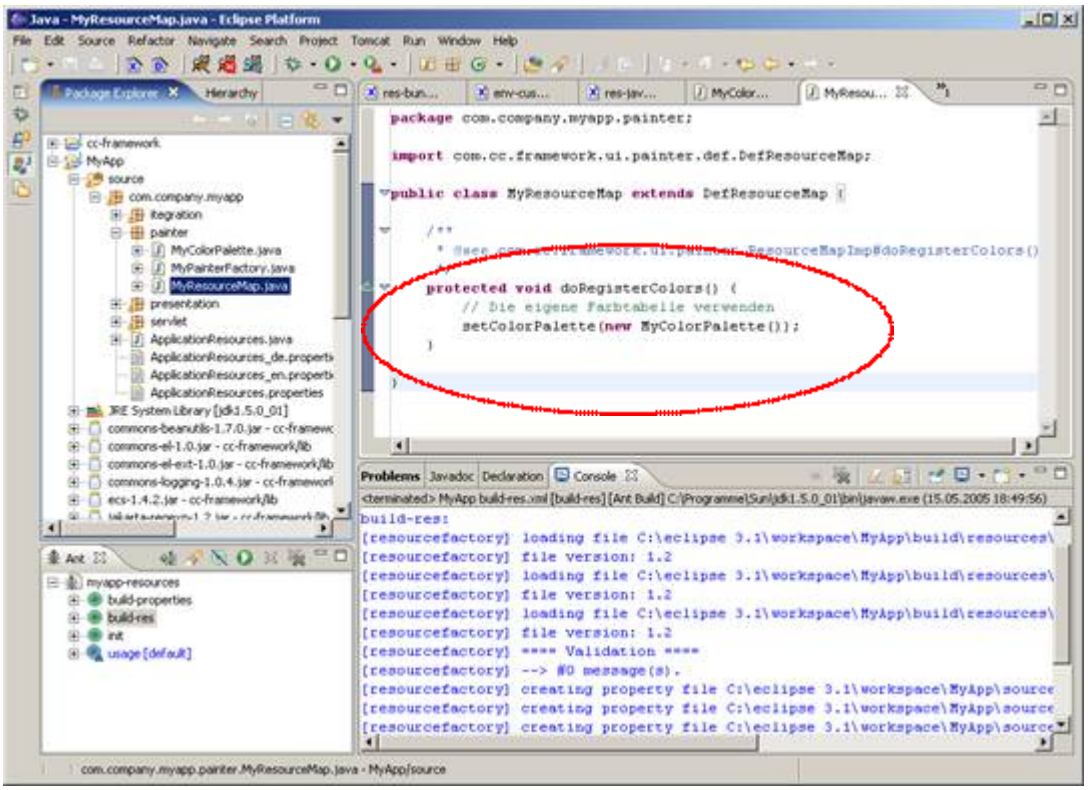
XML

```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>

```

After running the “build-res” Ant Task we find the new Class MyColorPalette



Now lets change MyResourceMap to use our own color palette. We do this by overwriting the doRegisterColors() method:



3.7 Creating Image Buttons

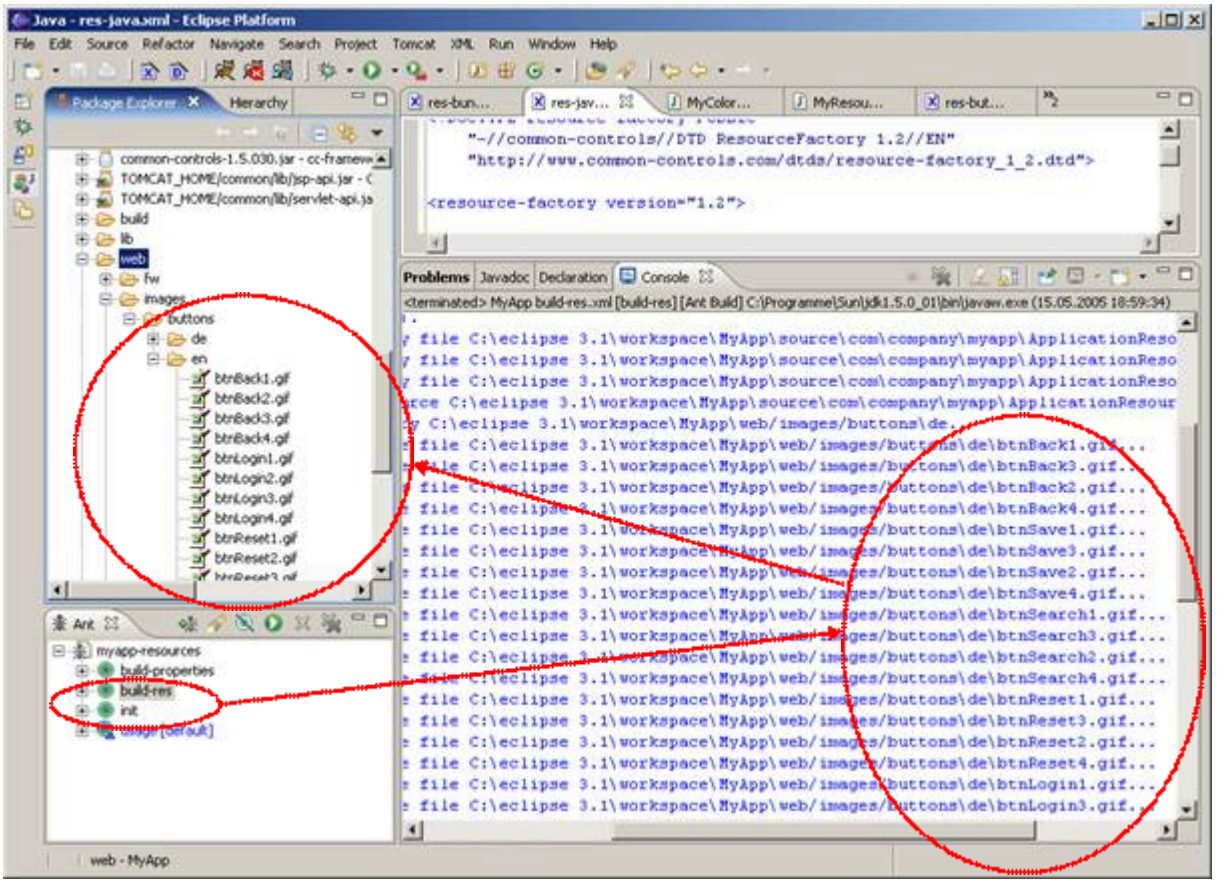
Now we want to create some localized image buttons. So we add the new resource definition file res-buttons.xml to our MyApp/build/resources directory.

XML

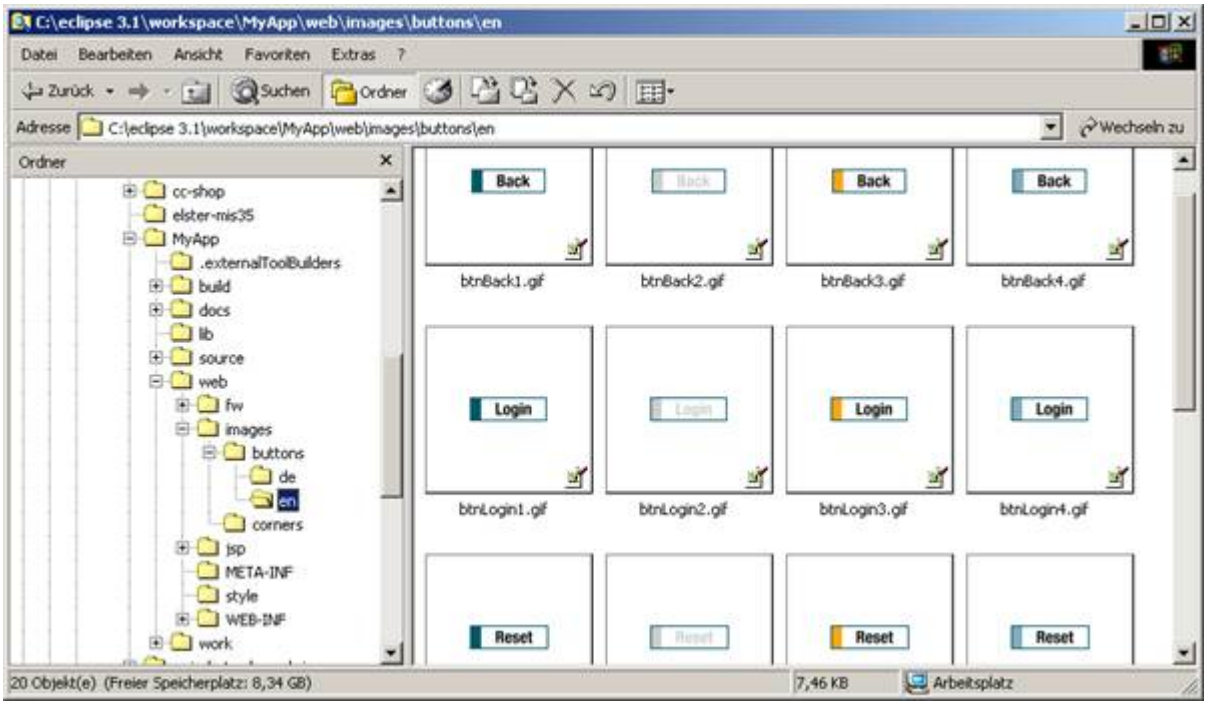
```xml
<?xml version="1.0" encoding="iso-8859-1"?>

<!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
```

The overwrite=”false” flag directs the Resource Factory tool to create a new resource only when it is not already existing. Keep this in mind when you change the colors in your color map. You will have to set overwrite to “true” in this case!



You can change the color schema of your buttons by playing around with the color constants in env-custom.xml (remember to set overwrite to “true”).



3.8 Creating Custom StyleSheets

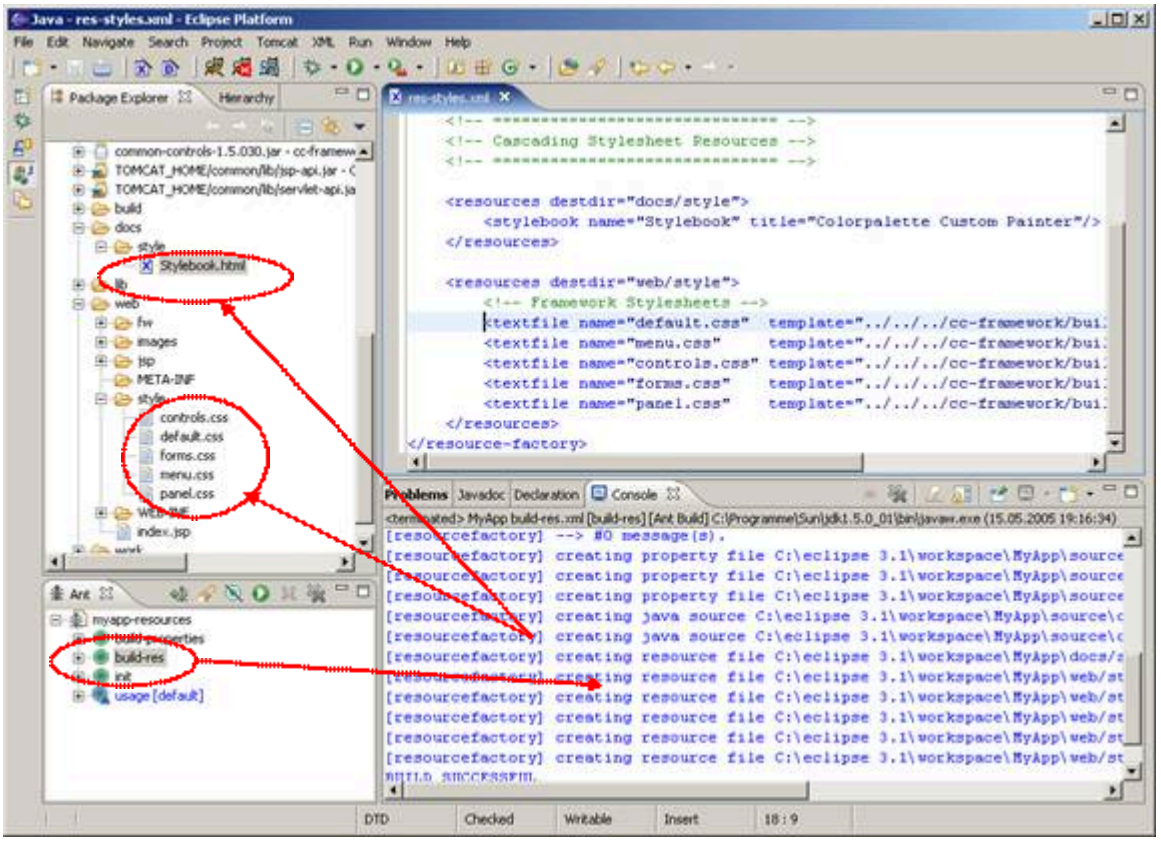
To create the custom StyleSheets that use our color schema we have to add a new resource definition file res-style.xml to our MyApp/build/resources directory.

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
```

This will use the templates defined in the framework distribution and create new StyleSheets with the color definitions in the env-custom.xml file.



Now all we have to do is to tell our ResourceMap to use our new Cascading StyleSheets and not the StyleSheets in fw/def/style. To be more specific we have to use our version of the default.css file because this file includes all the other StyleSheets from the same directory:

Template 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";

```

The DefPainterFactory uses the symbolic constant "fw.file.css.default" (= DefResources.
FW_FILE_CSS_DEFAULT) to resolve the name of the default.css file. So all we have to do is to change the value of this constant in our MyResourceMap:

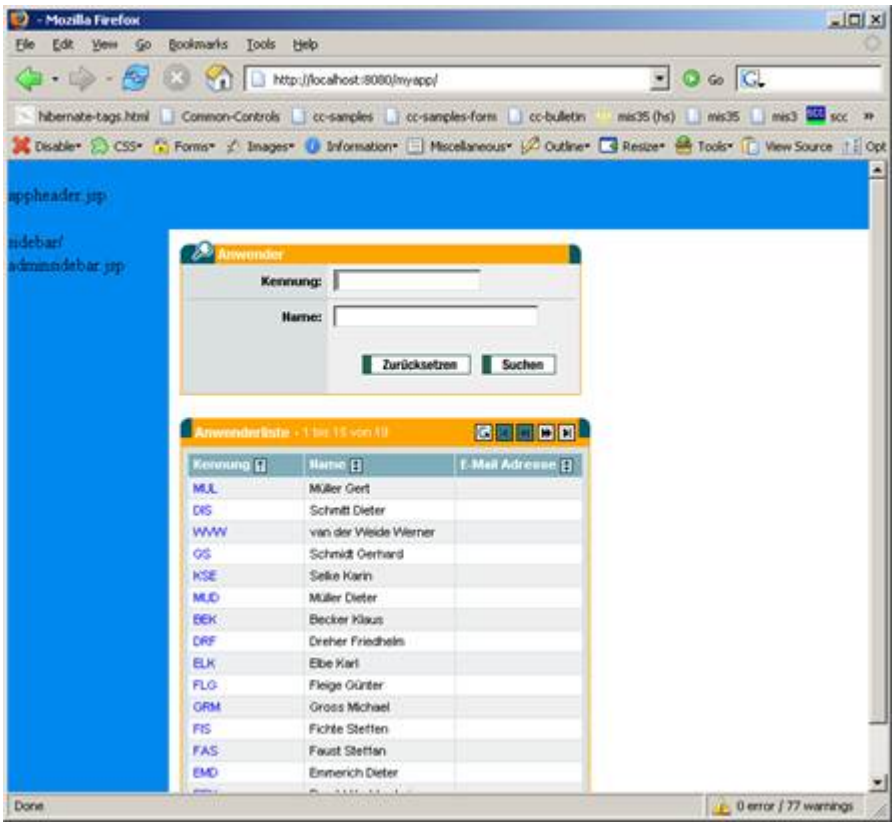
JAVA

```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");
    }
}

```

à a list of all symbolic constants can be found at

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

When we now start the application again we can see our new custom color schema



The HTML code shows the changed URL

CODE

```
<!-- painter 'global' -->
<script type='text/javascript' language='JavaScript' src='fw/global/jscript/common.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/global/jscript/environment.js'>
</script>
<script type='text/javascript' language='JavaScript' src='fw/global/jscript/utility.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/global/jscript/formatter.js'></script>
<!-- end -->

<!-- painter 'html' -->
<!-- end -->

<!-- painter 'mypainter' -->
<link rel='stylesheet' href='style/default.css' charset='ISO-8859-1' type='text/css'>
<script type='text/javascript' language='JavaScript' src='fw/def/jscript/functions.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/def/jscript/resourcemap.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/def/jscript/controls.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/def/jscript/tabset.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/def/jscript/tree.js'></script>
<script type='text/javascript' language='JavaScript' src='fw/def/jscript/list.js'></script>
<!-- end -->

```

3.9 Changing the Corner Images

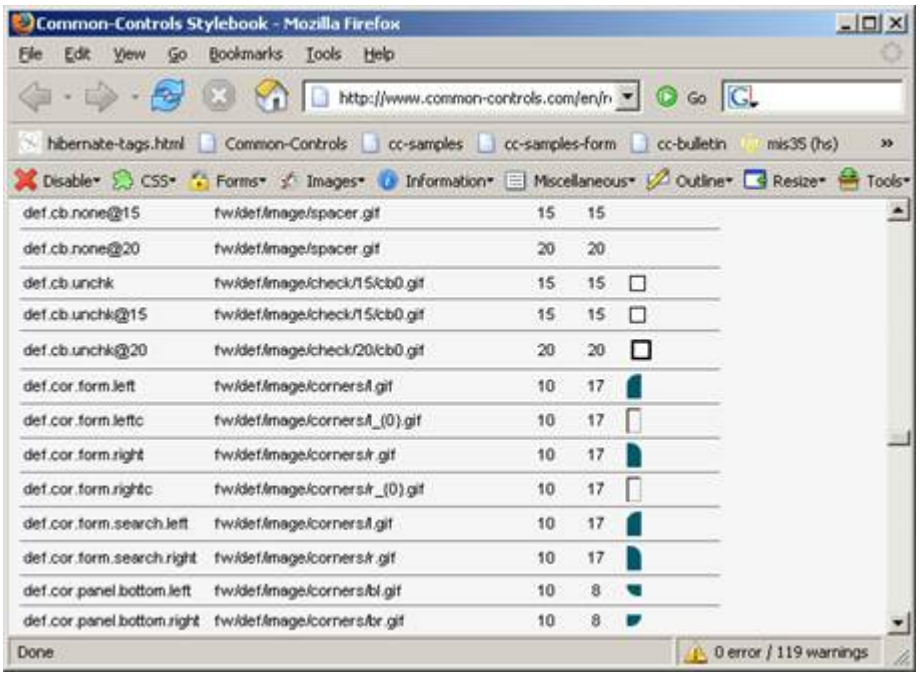
Now we have to change the frames corner images. The symbolic names of the images are

- def.cor.form.search.left

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

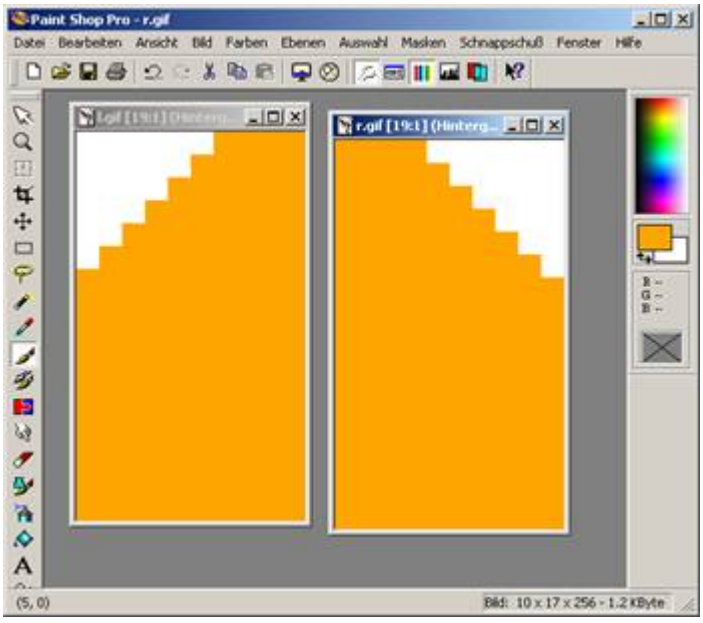
The overview of all symbolic constants of the DefPaintercan be found at:

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
```

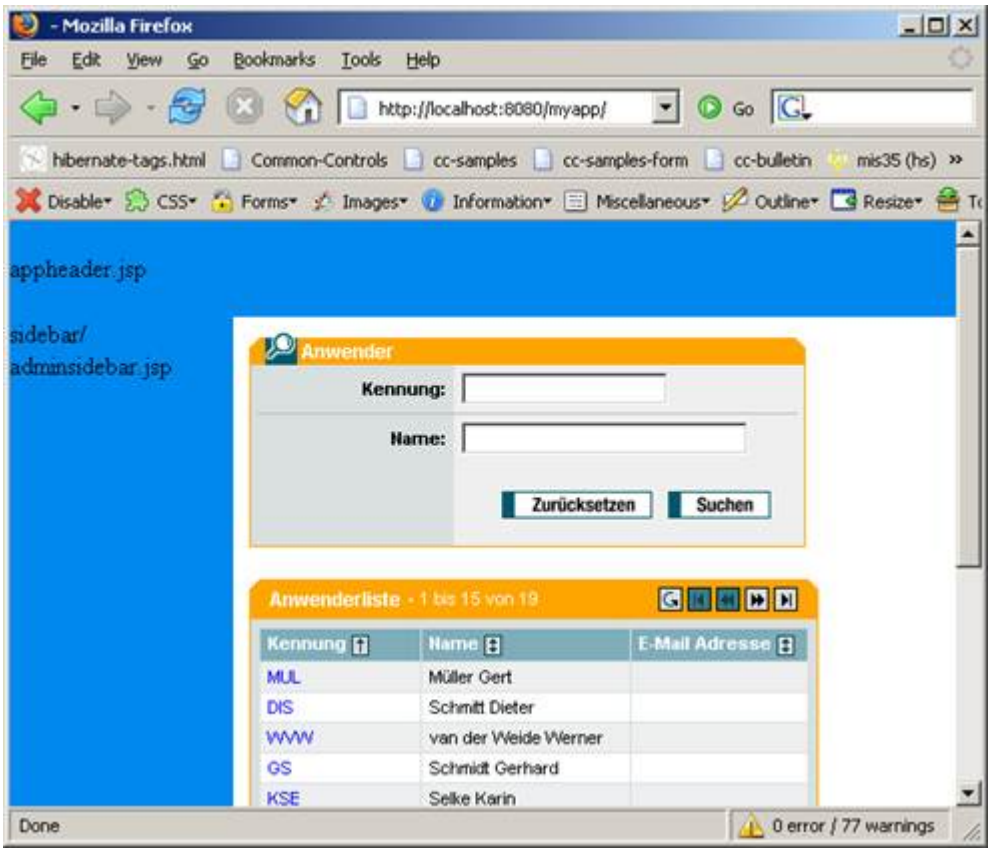


Now we have to register our new images in our MyResourceMap:

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));
    }
}

```

When we restart the application we will get the following result:



3.10 Using Custom StyleSheets

TODO

4 Glossary

CC

Common-Controls