CaptainCasa Enterprise Client

Embedded Usage Mode

Table of Contents

Embedded Usage Mode

Goal

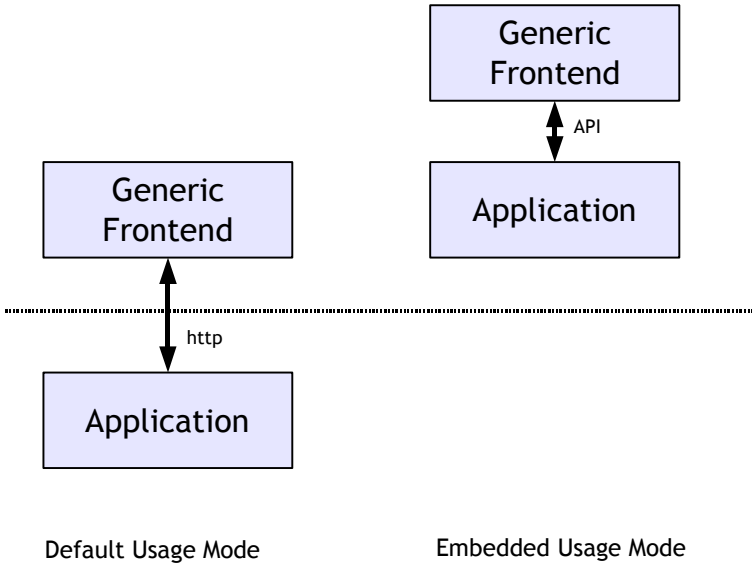
Scenarios

Typical scenarios are:

- Offline Capability: you may want to provide applications that do not (always) depend on the availability of network resources.
- Locally installed application: you may want to provide your application as single-user, locally installed application. Maybe this is for demo-purposes or for low-level/ entrance-level use.
- Power user support: dedicated users may run their application within the client – and do not share application server resources with other users. The application is run as “fat client”, i.e. the application logic is executed on frontend side.

One and the same Application – Two Usage Modes

The nice news: there is no change to your application required for running it in embedded usage mode. It is just the technology environment that changes:



It's your choice how much logic the application running on client side should own. You may include an embedded database on client side (e.g. something like hypersonic HSQL), you may connect the client side application to a central database by using JDBC, etc.

Technology behind

Normal Usage Mode

The CaptainCasa Enterprise Client consists of two framework parts:

- Frontend: the generic Java Client
- Backend: the server side component library + integration into JSF

In the typical installation the frontend part is running in a browser environment, staterd as applet or webstart application. The backend part is running within a servlet container – Tomcat being the one used by default. Frontend and backend are connected via HTTP over TCP/IP.

Embedded Usage Mode

In the embedded usage mode the application processing is directly integrated into the frontend program. Communication between frontend and application processing is NOT done by HTTP over TCP/IP – but is done by direct API integration between frontend and backend.

You see: the application that is normally running as backend application, on server side, now runs as embedded part of the client program.

While normally the frontend is opening a socket communication in order to talk to the backend processing, now the frontend directly, by Java call, exchanges information with the backend part.

Technology Environment

Still, the frontend is decoupled from the application processing by an explicit XML page definition – there is no change to the general architecture of having a decoupled, generic client concept.

The “only” difference: now the server is running in the process of the client. This is made possible by using the Tomcat servlet engine in a so called “embedded mode”, and by using the “MemoryProtocolHandler” for connecting the client part with the server part.

Technology Advantages

The fact, that the embedding of the backend processing is done by eliminating any socket communication has several advantages:

- Performance – no dealing with sockets
- Stability – no timeouts, no socket overloads
- Installation – no personal firewalls blocking the communication
- Security – no risk of opening a mini-web server with application functions behind on client side which could be accessed from other clients.

Running your Application as Embedded Application

Embedded Usage Environment

The environment for using the embedded mode is installed with the normal setup-program that is available for Windows installations. It is part of the .zip file download as well – please check the directories below “resources”. This documentation does focus on the installation scenario under Windows – there are no differences to other operating systems than the “.bat” file needs to be translated to a “.sh” file.

After installation you will recognize a new directory being created within the “resources” directory of your installation:

```
<installdir>
    /documentation
    /resources
        /eclipseplugin
        /embeddedusage
            /client
            /embeddedserver
                /bin
                /conf
                /lib
                /webapps
                    /demos
            /templates
                build_standalone_project.xml_template
                ccbuild_standalone.xml
                start.bat_template
    /server
```

Directory “client”

This is the directory in which all “.jar” files are stored which belong to the client part. They are normally located within your web application's “/eclnt/lib” directory.

Directory “embeddedserver”

This is the directory in which the server part is kept. Indeed it is an extract of a normal Tomcat installation:

- In the “bin” directory and the “lib” directory all the jar files are kept that normally are shipped with Tomcat.
- In the “conf” dierctory the normal Tomcat configuration files are kept.
- In the “webapps” directory the web applications are kept, one per sub-directory.

Directory “templates”

This directory contains some script files that you require for quickly building your application so that it runs in embedded mode.

This is the file to start the application. Have a look into its content – the only difference to the normal client start file is:

- Not only the client libraries are included into the class path definition, but also the ones required for starting Tomcat in embedded mode
- The address of the page to be opened now starts with “embedded://embedded” instead of “http://host:port”

```
set JAVA_HOME=C:\bmu_jtc\jdk\jre
set CALTAINA_HOME=.\tomcat

set TCP=.\lib\eclnt.jar
set TCP=%TCP%;.\client\swt.jar
set TCP=%TCP%;.\client\jasperreports-3.0.0.jar
set TCP=%TCP%;.\client\commons-beanutils-1.7.jar
set TCP=%TCP%;.\client\commons-collections-2.1.jar
set TCP=%TCP%;.\client\commons-digester-1.7.jar
set TCP=%TCP%;.\client\commons-javaflow-20060411.jar
set TCP=%TCP%;.\client\commons-logging-1.0.2.jar
set TCP=%TCP%;.\client\commons-logging-api-1.0.2.jar
set TCP=%TCP%;.\client\PDFRenderer.jar

set TCP=%TCP%;.\embeddedserver\bin\bootstrap.jar
set TCP=%TCP%;.\embeddedserver\bin\tomcat-juli.jar
set TCP=%TCP%;.\embeddedserver\bin\log4j-1.2.15.jar

set TCP=%TCP%;.\embeddedserver\lib\annotations-api.jar
set TCP=%TCP%;.\embeddedserver\lib\catalina.jar
set TCP=%TCP%;.\embeddedserver\lib\catalina-ant.jar
set TCP=%TCP%;.\embeddedserver\lib\el-api.jar
set TCP=%TCP%;.\embeddedserver\lib\jasper.jar
set TCP=%TCP%;.\embeddedserver\lib\jasper-el.jar
set TCP=%TCP%;.\embeddedserver\lib\jasper-jdt.jar
set TCP=%TCP%;.\embeddedserver\lib\jsp-api.jar
set TCP=%TCP%;.\embeddedserver\lib\servlet-api.jar
set TCP=%TCP%;.\embeddedserver\lib\tomcat-coyote.jar
rem set TCP=%TCP%;.\embeddedserver\lib\tomcat-i18n-fr.jar
rem set TCP=%TCP%;.\embeddedserver\lib\tomcat-i18n-ja.jar

start %JAVA_HOME%\bin\javaw.exe -cp %TCP%
org.eclnt.client.page.PageBrowser embedded://embedded
/demos/faces/workplace/workplace.jsp loclevel=SEVERE
```

All the start parameters that you usually can pass to the client (“headerline”, “footerline”, “country”, ...) are applicable as well when starting the embedded mode.

Deploying your Application - “Quick shot”

This chapter describes how to set up a first environment to run your application in embedded mode. For continuous development we strongly recommend to use ANT scripts that automate the execution of tasks – see the next chapter for explanations how to automate the deployment using ANT.

Deploying

Deploying your Application is simple:

- Create a new directory at any location of your choice:

```
YOURDIRECTORY
```

- Copy the content of the embedded usage environment (“everything inside /resources/embeddedusage”) into this directory:

```
YOURDIRECTORY
    client
    embeddedserver
    templates
```

- Create a directory below “embeddedserver/webapps” that represents your web application:

```
YOURDIRECTORY
    client
    embeddedserver
        yourwebapp
    templates
```

- Copy the webcontent part of your application below this directory, so that the application is placed within the webapps directory just like with normal Tomcat installations.

```
YOURDIRECTORY
    client
    embeddedserver
        yourwebapp
            eclnt
            eclntjsfserver
            META/INF
            WEB/INF
            ...
    templates
```

- Copy the „start.bat_template“ from the templates directory into your directory and rename it to „start.bat“:

```
YOURDIRECTORY
    start.bat
    client
    embeddedserver
        yourwebapp
            eclnt
            eclntjsfserver
            META/INF
            WEB/INF
            ...
    templates
```

- Update the start.bat files content according to your environment.

Updating start.bat

Have a look into the template file:

```
set TCP=.\client\eclnt.jar
set TCP=%TCP%;.\client\swt.jar
set TCP=%TCP%;.\client\jasperreports-3.0.0.jar
set TCP=%TCP%;.\client\commons-beanutils-1.7.jar
set TCP=%TCP%;.\client\commons-collections-2.1.jar
set TCP=%TCP%;.\client\commons-digester-1.7.jar
set TCP=%TCP%;.\client\commons-javaflow-20060411.jar
set TCP=%TCP%;.\client\commons-logging-1.0.2.jar
set TCP=%TCP%;.\client\commons-logging-api-1.0.2.jar
set TCP=%TCP%;.\client\PDFRenderer.jar

set TCP=%TCP%;.\embeddedserver\bin\bootstrap.jar
set TCP=%TCP%;.\embeddedserver\bin\tomcat-juli.jar
set TCP=%TCP%;.\embeddedserver\bin\log4j-1.2.15.jar

set TCP=%TCP%;.\embeddedserver\lib\annotations-api.jar
set TCP=%TCP%;.\embeddedserver\lib\catalina.jar
set TCP=%TCP%;.\embeddedserver\lib\catalina-ant.jar
set TCP=%TCP%;.\embeddedserver\lib\el-api.jar
set TCP=%TCP%;.\embeddedserver\lib\jasper.jar
set TCP=%TCP%;.\embeddedserver\lib\jasper-el.jar
set TCP=%TCP%;.\embeddedserver\lib\jasper-jdt.jar
set TCP=%TCP%;.\embeddedserver\lib\jsp-api.jar
set TCP=%TCP%;.\embeddedserver\lib\servlet-api.jar
set TCP=%TCP%;.\embeddedserver\lib\tomcat-coyote.jar
```

```
start @JAVAW@ -cp %TCP% org.eclnt.client.page.PageBrowser
embedded://embedded /@WEBAPP@/faces/@PAGE@ loglevel=SEVERE @CLIENTPARAMS@
```

The start.bat content looks very similar to starting the CaptainCasa Enterprise Client via batch file. But: now all libraries of the embedded Tomcat are part of the classpath definition as well.

The URL that is called in order to open the page is very similar to the normal starting of CaptainCasa Enterprise Client as well. The only difference is:

- „host:port“ need to be set to „embedded://embedded“

Replace the „@JAVAW@“, „@CONTEXTROOT@“ and „@PAGE@“ so that the parameters fit to your local environment.

That's it – now you can start your application in embedded mode!

Fine Tuning

- You may wipe out the eclnt directory in your application – the client is not loaded from the web application anymore...
- You may pass a very long duration of http sessions within your web.xml definition. There typically is no timeout necessity for fat client applications. - The definition of session timeouts is done within the web.xml deployment descriptor of your application:

```
    <session-config>
      <session-timeout>-1</session-timeout>
    </session-config>
```

When setting the session timeout to “-1” then the session will never be timed out.

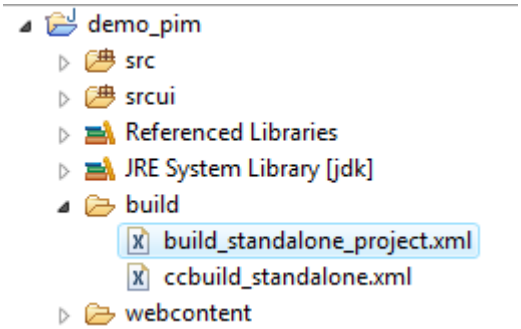
Deploying your Application – Automation by using ANT

We strongly recommend to use an ANT task in order to deploy you application into an embedded usage environment.

There are two corresponding template files available in the resources/embeddesusage/templates directory of your installation.

- ccbuild_standalone.xml – This file is a ready to use ANT script that can be called by another ANT script. Certain parameters need to be defined by the calling ANT script.
- build_stanalone_project.xml_template – This is a template for a calling ANT script.

Copy both files into your project, into a “build” subdirectory and rename the „xml_template“-file to „xml“. - Your project should look like:



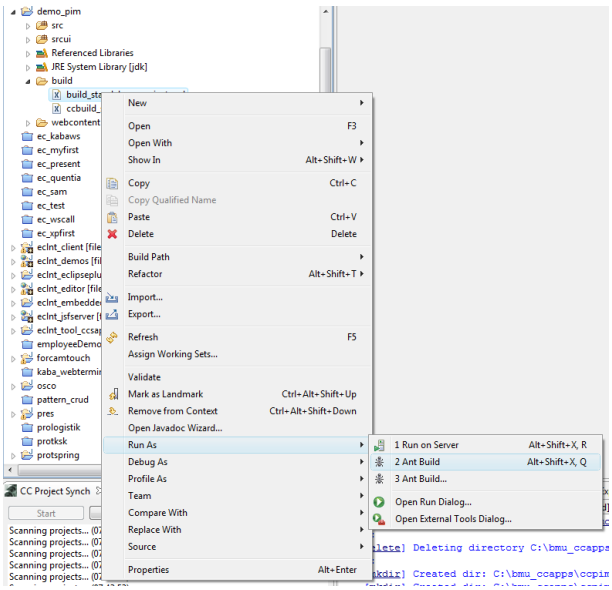
The „build_standalone_project.xml“ script does nothing else than defining properties and calling script „ccbuild_standalone.xml“:

```
<!--
    This build files is designed to be placed into the /build directory of your
    project directory.
-->

<project name="BUILDEMBEDDED" default="BUILD">

       <description>Build embedded version of web application</description>

       <!--
        property definitions

        ADJUST THE FOLLOWING PROPERTIES SO THAT THEY CORRESPOND TO
        YOUR ENVIRONMENT.

        (.) captincasa.dir - installation directory of CaptainCasa
        (.) standalone.dir - directory in which the stand alone application will be created
        (.) standalone.webapp and .startpage - JSP file that is opened by start.bat,
            typically use the same context root that you use for normal web processing
        (.) standalone.javaw - javaw-call that is placed into the start.bat file.
        (.) standalone.clientparams - client parameters that are appended for starting
            the Enteprise Client

        PAY ATTENTION: the ANT script that is called (ccbuild_standalone.xml) is
        intended to be run within the /build directory of your project!
       -->

       <target name="BUILD" description="BUILD">
               <ant antfile="ccbuild_standalone.xml">
                       <property name="captaincasa.dir" value="C:/EnterpriseClient"/>
                       <property name="standalone.dir" value="C:/test/myapp"/>
                       <property name="standalone.webapp" value="mywebapp"/>
                       <property name="standalone.startpage" value="mypage.jsp"/>
                       <property name="standalone.clientparams" value="startwidth=768
startheight=1024"/>
                       <property name="standalone.javaw" value="C:\java\bin\javaw.exe"/>
               </ant>
       </target>

</project>
```

A brief explanation for each parameter is available within the XML file.

Please have a look into the ANT script that is called (ccbuild_standalone.xml) – you will see that “nothing great” is done rather than copying certain directories and creating/updating a batch file to start.

After having updated the properties of „build_standalone_project.xml“ you now can execute the script. When using Eclipse then you can execute by right mouse click onto the script file and selecting „Run As > ANT Build“.



Restrictions

There are only a few restrictions when running your application in embedded mode.

Tomcat only...

The first restriction is obvious: only Tomcat based applications can be run in embedded mode because the embedded modes is based on the fact that Tomcat can be started as embedded servlet engine.

If your application is a “tough J2EE application” utilizing Enterprise Java Features that are outside the scope of Tomcat then, well, it's just not possible to distribute in embedded mode. Of course you could think about only distributing only a certain, “more lightweight” part of your application, but that's a different story.

Browser Component

When using the BROWSER component of CaptainCasa then you normally pass a URL which is passed to the browser. - Now, in embedded mode, there is no http server which can respond to a URL, consequently you need to pass the URL as file-URL (e.g. “file:///C:/temp/...”).

In you code, when creating the URL you need to check if you are running in embedded mode prior to creating the URL. There is a corresponding function in “HttpSessionAccess” that tells in which mode you are running with your server side application:

```
...
if (HttpSessionAccess.checkIfRunningInEmbeddedMode() == false)
{
    // “normal processing”
    ...
    ...
    String brwoserURL = “abcd.xyz”; // relative download URL, which
                                    // is internally resolved as
                                    // http://...../abcd.xyz
}
else
{
    // “embedded processing”
    ...
    ...
    String browserURL = “file:////..../abcd.xyz”;
}
```

In case you “require” a certain directory to place your file: there is the function “HttpSessionAcces.getServletTempDirectory”. This function passes back the directory that is managed by the servlet container per web application for placing temporary files. In case of Tomcat this is the “/work” directory, parallel to “/webapps”.

When using the CaptainCasa BufferedContentMgr then you should used the function “IBufferedContent.getURLForBrowserUsage()” instead of “IbufferedConten.getURL()”.

One additional remark: an other area where URLs are used, is the area of file download components (e.g. FILEDOWNLOADLINK). These components are not “outside components” as the BROWSER but are using the normal CaptainCasa client communication for retrieving the download data from the server. In other words: they are aware of the client running on top of an embedded Tomcat, so you do not have to worry about at all. - Of course you need to pass back URLs that point back into your web application and not URLs that start with “http://....”.