

CaptainCasa

Enterprise Client

App Start



Table of Content

App Start.....	3
Motivation.....	3
Applet/Web Start require Java Installation.....	3
Stand alone Application.....	4
...App Start!.....	4
Purpose.....	5
Launcher Example - How it looks like.....	6
Launcher - How it works.....	7
Initial Client Installation.....	7
appconfig.xml Configuration File.....	7
Starting the Launcher on Client Side.....	9
Start Scripts on Client Side.....	10
What happens if the Server is not available?.....	10
Building your own App Start Scenario.....	10
The Starter Package.....	10
The appconfig.xml / Server-side definition.....	11
Test...!.....	12
Building an .exe Installer.....	12
Special Issues.....	14
“appstart.finished” - Files in Version Directories.....	14
Native Libraries.....	14
Proxy Configuration.....	14
Open Issues.....	16
Change Log.....	17
Version 20141008.....	17
Version 20150425.....	17

App Start

“App Start” is a small framework for starting JavaFX programs on client side as stand alone application.

App Start is a small program, checking the version of the program to be started against some server definition. If the server indicates that some new version of the program is available then the corresponding software will be downloaded automatically and executed afterwards.

The launcher is a free framework - we are open to share everything if it's of interest for anyone. In case of interest: please contact us via info@CaptainCasa.com.

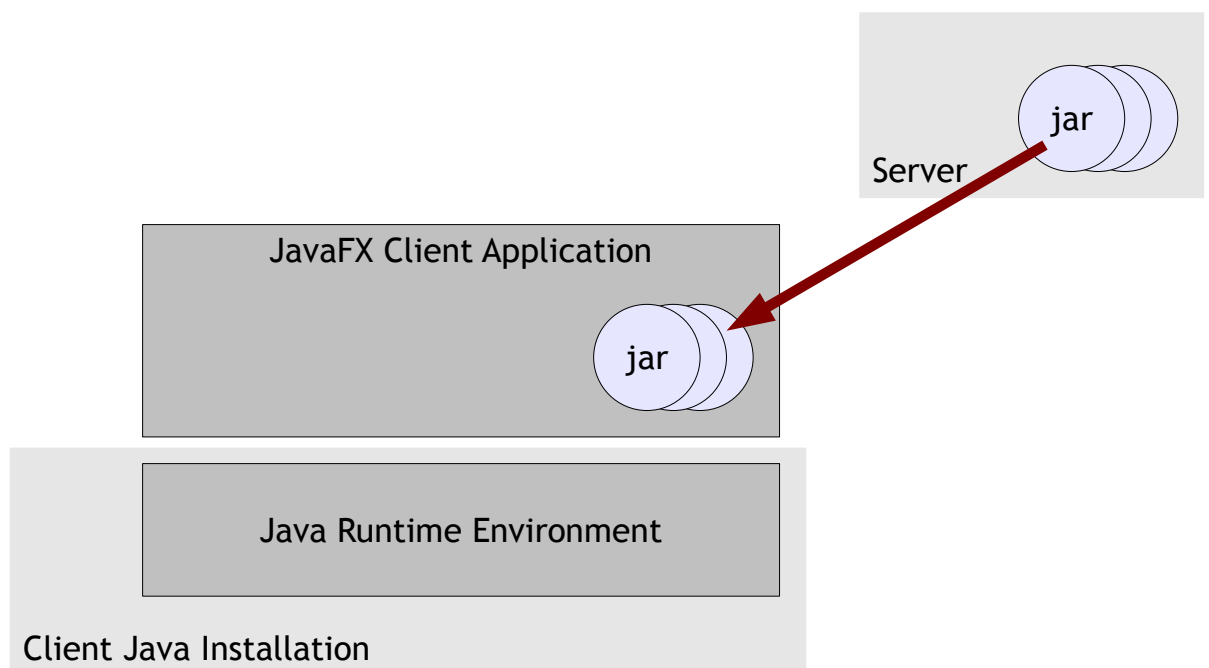
Motivation

The default ways of starting JavaFX based frontends are:

- A. start as Applet
- B. start via Web Start
- C. start as stand alone application

Applet/Web Start require Java Installation

Both the Applet and the Web Start way require a Java installation to be done before being able to start a JavaFX application at the user's client.



As consequence there are some problems:

- Some users/companies follow some general policy to not allow the installation of a Java virtual machine on the end user's client. Typically the background of this policy is the security discussion that came up with running applets within the browser. If this discussion is a valid one or not: the possibility that you convince people to change this

policy is quite low...

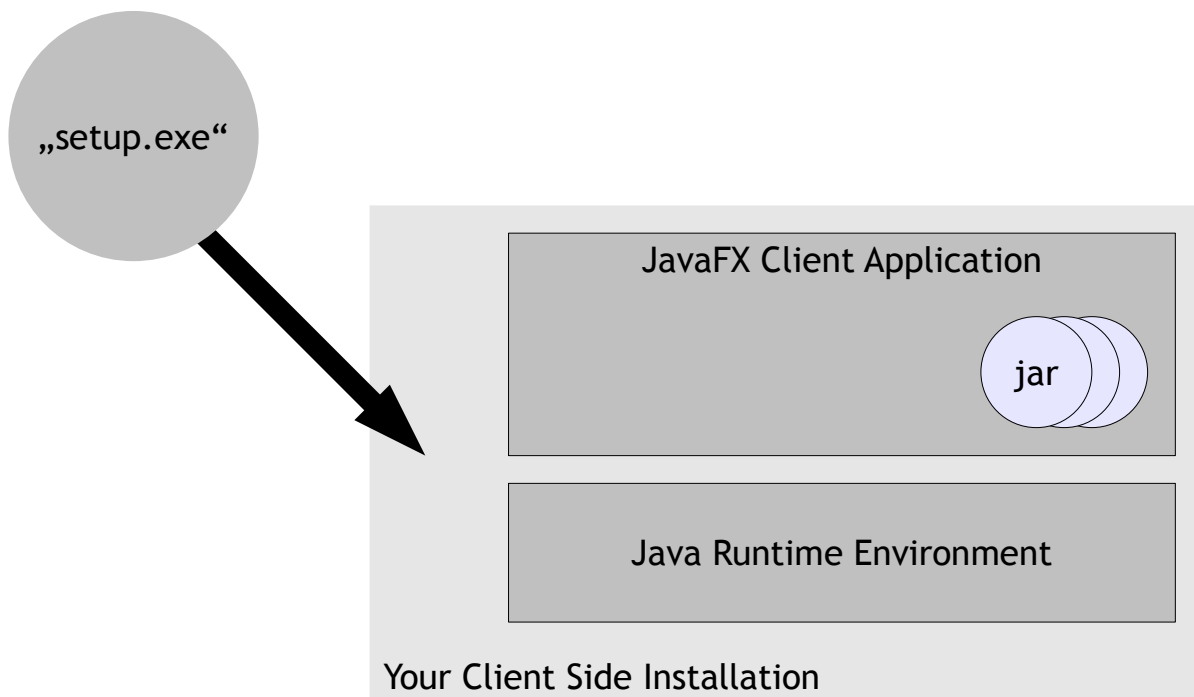
- You have to live with the fact that the version of the runtime environment is not defined by yourself but that is defined by the user's/company's environment. Especially in the JavaFX environment, which is a quite young one, you want to use latest Java FX releases. - JavaFX as “available technology” is part of Java 8 - it will take quite a while until your users will adapt to this release.
- In addition: you want to develop and test your frontend with a certain Java version, and then roll out, based on this version. You do not want to guarantee your software is working with all potential Java releases that will come up...

So, though the Applet/Web Start way is the only one to somehow embed Java processing within the browser environment, it is not the perfect one for rolling out JavaFX frontends today.

Stand alone Application

This is the alternative: you deliver your Java software as a stand alone installation - including the Java Runtime Environment in some sub directory. You do not have to convince your users to install Java on the client! What you deliver is some type of native application that internally uses Java and JavaFX.

This is more and more supported by the Java default tooling - bundling the JavaFX program with the JRE, e.g. to provide a “setup.exe”-like installation.



Problem now:

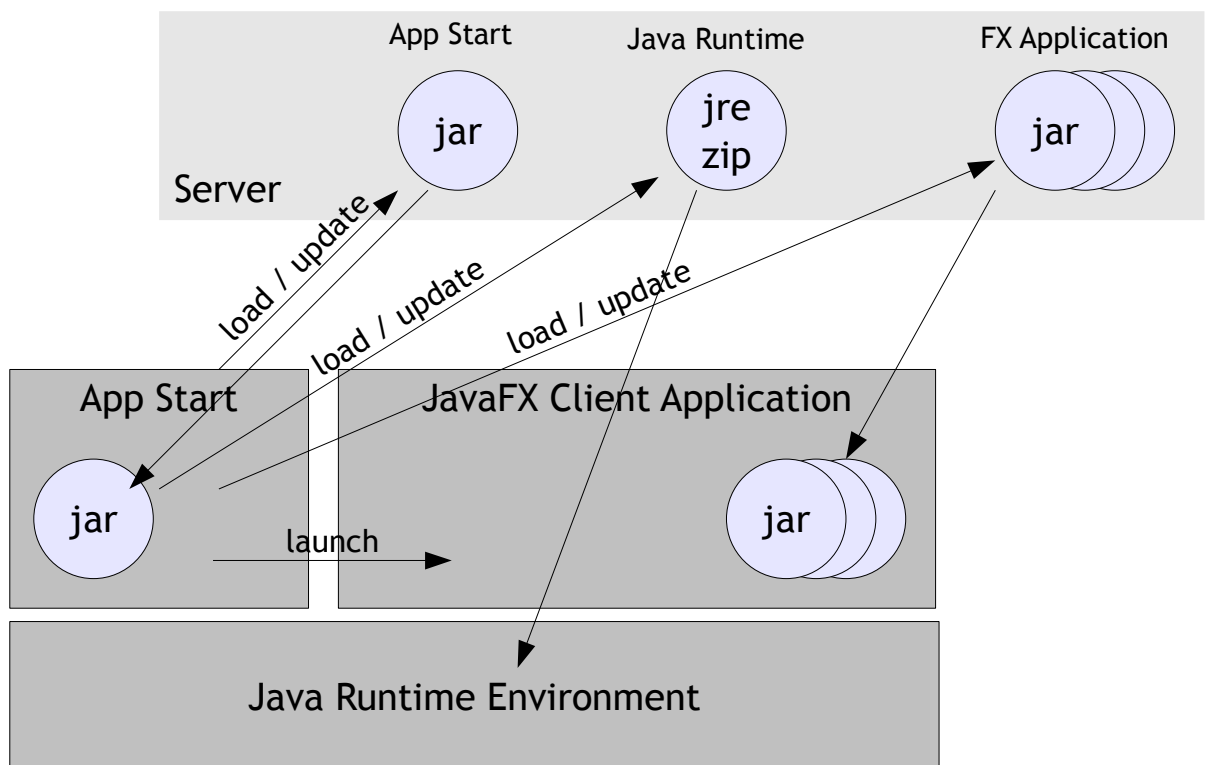
- This is a one time installation only - there is no concept of what happens if the software needs to be updated on client side. - The automated reloading of the Java Application that you know from Applet/Web Start is not available.

...App Start!

This is where our App Start is coming in: a small framework for launching JavaFX based

applications on client side, including the following features:

- During initial client side installation the JavaFX application is installed together with a Java Runtime Environment and with the App Start launcher, so that all together runs as stand alone Java application.
- The JavaFX application is started through the App Start launcher, which itself is a Java program.
- When launching, the current client version of the application is checked against some server version. If the server indicates that some new version is available, then the App Start launcher will load the corresponding software via http(s) and install it within the client system.
- There are three levels of changes that can be applied to the client software:
 - Application level: there is a new version of the JavaFX application that is started.
 - Launcher level: there is a new version of the Launcher itself.
 - Java Runtime level: there is a new Java version to be used.



Of course the “Application Level” is the most typical scenario. The JavaFX application is updated and you want these changes to be rolled out to the users of the application. - But the other scenarios (Launcher level, Java Runtime level) are equally important in the long run: at a certain point of time you want to upgrade the Java virtual machine that your JavaFX application is built on. And a certain point of time, also the launcher will require some update.

Purpose

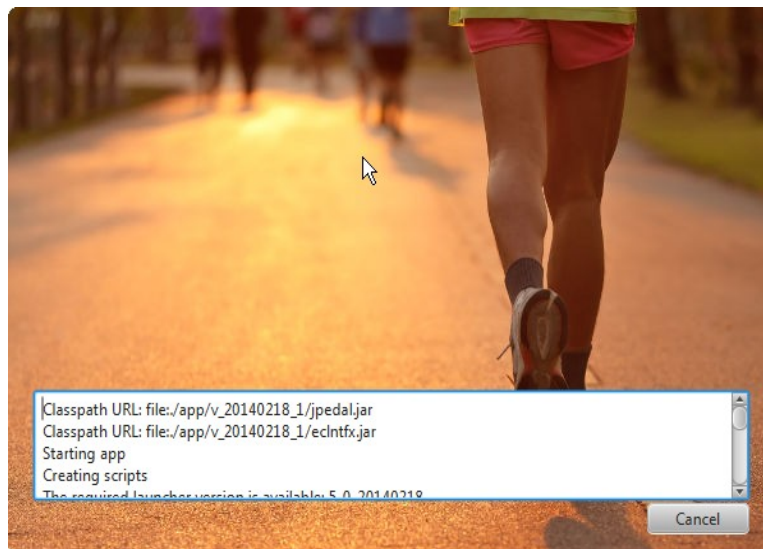
So the App Start launcher's purpose is to automate and simplify the continuous roll out of JavaFX based software started as stand alone application on client side.

Please note: the idea behind the Applet/Web Start processing is to have a general Java

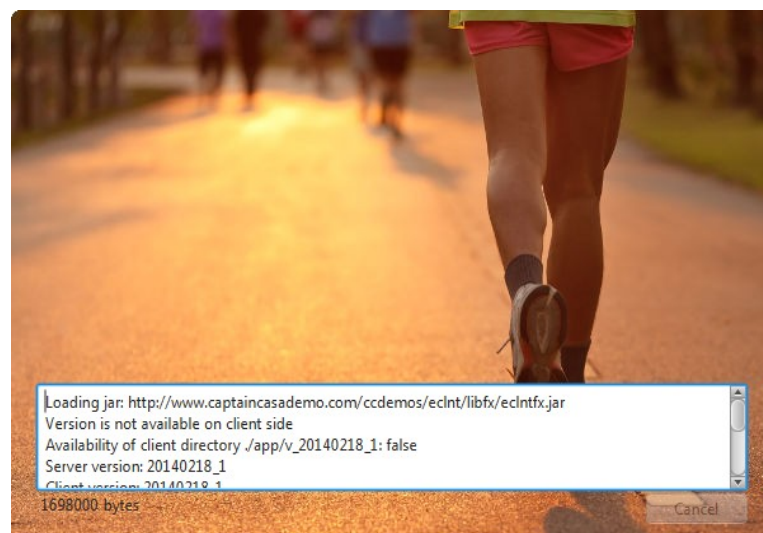
installation on client side, which is used by multiple Java programs. The idea behind App Start processing is to focus on one JavaFX application, which includes a Java Runtime Environment as “invisible, contained part”, that is only used in the context of this one JavaFX application.

Launcher Example - How it looks like

After starting the launcher comes up with a splash screen. In the bottom area of the splash screen there are some status messages telling the user what's going on:



Typically the launcher will only show up for a short duration of time - because it just finds out the buffered client version is in sync with the current server version. If there is some need to upgrade and download corresponding files, then this will be shown to the user:



CaptainCasa Enterprise Client

After the launcher has successfully started the JavaFX application, the splash screen will disappear.

You may test the launcher and see how it's working by starting our JavaFX Enterprise Client though the launcher: <http://www.captaincasa.com/fxclient.html>

Launcher - How it works

The following text demonstrates how this program is embedded into the launcher environment.

Initial Client Installation

The launcher works with a fix directory structure on client side. The structure is created during initial setup of the client application:

```
<anydirectory>
  /bin
    start.bat
  /app
    /v_20140218_1
      xxx1.jar
      xxx2.jar
    /jre
      /v_17_51
        /bin
        /lin
      ...
  /launchfx
    /v_20140214
      launchfx.jar
    /resources
      appconfigurl.xml
      splash.jpg
```

You see that there are three main directories, which contain themselves version-directories:

- The “app” directory holds the jar-file versions of the JavaFX application that is to be launched. This folder typically is empty after installing the launcher - the first version of the application will be loaded when starting the launcher the first time.
- The “jre” directory holds Java runtime versions.
- The “launchfx” directory holds the launcher versions.

After first installation you will find one JRE (v_17_51 in the directory structure above) and one launcher version (v_20140214).

Inside the /launchfx directory there is a /resources-directory. Inside this there are two files:

- A splash screen that is used by the launcher.
- A file “appconfigurl.xml” that contains a link to a server side configuration file. This configuration file contains detailed information about the JavaFX application that is to be launched. The file's content may look like:

```
<appconfigurl url="http://www.captaincasademo.com/ccdemos/appconfig.xml"/>
```

The launcher is started through the bin/start.bat script using the initial Java version. It accesses the URL that is defined in the launchfx/resources/appconfig.xml file, loads its XML content, checks if there is some necessity to download updated versions and finally starts the JavaFX application.

appconfig.xml Configuration File

This is an example of the appconfig.xml file that is loaded from the server:

```
<app
```

```

appname="CaptainCasa Enterprise Client"
appversion="20140218_1"
launcherversion="5_0_20140218"
launcherurl="${appConfigRootUrl}/ecIntlauncher/launcher_5_0_20140218.jar"

javaversion="17_51"
javaurl="${appConfigRootUrl}/ecIntlauncher/jre_17_51.zip">

<!-- Definition of application jars -->
<jar jarurl="${appConfigRootUrl}/ecInt/libfx/ecIntfx.jar"/>
<jar jarurl="${appConfigRootUrl}/ecInt/libfx/jpedal.jar"/>

<!-- Definition of application files (optional) -->
<!--
<file fileurl="${appConfigRootUrl}/abc.def"/>
-->

<!-- Definition of startables -->
<start id="DW"
    name="Demo Workplace"
    classname="org.ecInt.fxclient.elements.PageBrowserStarter">
    <vmparam value="-Xmx256m"/>
    <vmparam value="-Xms128m"/>
    <param value="url=${appConfigRootUrl}/faces/workplace/workplaceFX.jsp"/>
    <param value="loglevel=INFO"/>
    <param value="undecorated=true"/>
</start>

</app>

```

The appconfig.xml contains the following information:

- appname - the name, used for logging purposes in the client
- appversion - this is the current version of the application. The client is keeping all jar-files of the application under a version-directory. When the appconfig.xml contains a new version, then all its jar files will be downloaded to the client into new version-directory.
- launcherversion - this is the version of the launcher to be used
- launcherurl - link to the launchfx.jar file; if the client finds out that it needs to download the launcher version, then this is the link to the corresponding launchfx.jar version
- javaversion - this is the version of the JRE to be used; the version is a name, that does not need to be in sync with some official Java version naming (but of course it makes sense to use names that somehow are related to the Java version)
- javaurl - link to a .zip file containing the JRE. The zip file needs to contain the content of the JRE-directory (typically containing a "bin",
You may explicitly define JREs for each operating system in addition by using the optional attributes:
 - javaurlwindows32 - link to a .zip file containing the Windows32 JRE
 - javaurlwindows64 - link to a .zip file containing the Windows64 JRE
 - javaurllinux32 - link to a .zip file containing the Linux 32 JRE
- <jar .../> - the jar files that form the JavaFX application
 - jarurl - The url to a .jar file that is to be downloaded
 - oscondition - Definition that the file is only used if the launcher is started in a certain operating system environment. The currently supported operating systems are "windows32", "windows64" and "linux32". By defining "oscondition='windows32'" you define that the jar file is only used if the launcher

is started in a 32bit windows environment.

You may specify one or more operating system names by separating them with a semicolon. Example: “oscondition=windows32,windows64” defined that the corresponding jar file is both used in a Windows32 and a Windows 64 environment. This attribute is optional: if not defined then the corresponding jar file is used for all operating systems.

- <file .../> - optional list of additional files that are loaded to the client side
 - fileurl - The url to a file that is to be downloaded
 - oscondition - optional, same as oscondition in <jar .../> section
- <start .../> - the configuration of how to start the client; one application may contain one or several start definitions
 - id - unique id of the start definition
 - classname - start class (extending Application) to launch
 - default - “true” for the default start definition (if using more than one start definition)
 - <param .../> - Application parameters
 - value - the value of the parameter
 - <vmparam .../> - Java runtime parameters
 - value - the value of the parameter
 - <librarypathextension .../> - Optional. Library path extensions. If you pass native libraries (e.g. DLLs) then the default library path which is generated points into the app-version's directory. You may extend the path by adding one or more librarypathextension-definitions.
 - path - the path that you want to append to the library path
 - oscondition - optional, same as oscondition in <jar .../> section.

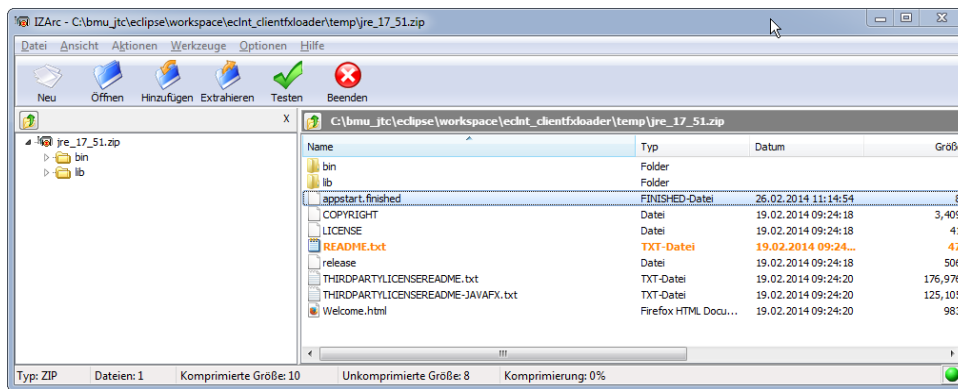
As you can see in the example all URL-definitions can either be defined absolutely (“http://...”) or they can be defined relatively to the path, by which the appconfig.xml file itself is accessed (“\${appConfigRootUrl}/...”).

Starting the Launcher on Client Side

When the launcher starts on client side, it loads the server side appconfig.xml and first checks if the current version situation on client side is up to date:

- 1. The jar files of the JavaFX application to be launched are kept in the directory “app/v_<appversion>/”. The launcher reads the appversion-definition of the appconfig.xml and checks if the corresponding directory is available. If not, it downloads all the jar files (“<jar .../>” definitions) and saves them on client side in a corresponding “app/v_<appversion>” directory. - The internal class loader of the launcher will from now on use these files to start the JavaFX application.
- 2. The Java Runtime Environment that is used for launching the JavaFX application is kept in directory “jre/v_<javaversion>”. If the javaversion-definition of the appconfig.xml is not represented by a corresponding directory, the launcher will download and unzip the java version into the corresponding directory. It expects the zip file to contain the Java runtime files.

Example: the following screen shot shows the JRE 1.7 v51 zip file's structure



The only file which was added within the main directory is the appstart.finished - a simple text file containing the text “Finished”. The role of this file is explained in the chapter “Special Issues”.

- 3. Finally the launcher checks if the launcherversion definition in the appconfig.xml is represented by a corresponding “launchfx/v_<launcherversion>” directory. If not, then it will download the launcher-jar file and save it as “launchfx.jar” in the corresponding directory.

If the launcher detects some version change of type 1. (app version) then there is no need to restart the launcher, the new app version will be loaded and launched immediately.

If the launcher detects some version change of type 2. or 3. (java version, launcher version) then it will not start the JavaFX application, but will request from the user to re-launch the application.

Start Scripts on Client Side

In the /bin directory of the client side launcher, the following scripts are generated:

- start_<start-id>.bat - Per start configuration in appconfig.xml there will be one script that starts the launcher according to the corresponding start-definition.
- start.bat - For the first start-definition in appconfig.xml, which is treated as the default start definition, a start.bat file will be generated.

The script will be adapted whenever there is a change in the Java-version or in the launcher-version.

What happens if the Server is not available?

If the server holding the appconfig.xml definition is not available, then the launcher will use the last valid appconfig.xml definition.

Building your own App Start Scenario

The Starter Package

Somehow the App Start launcher has to come to the user's client. This is the starter package consisting out of...

- The JRE (that is required to start the launcher)
- The launchfx.jar of the App Start launcher
- Two configuration files that YOU have to define...

- the appconfigurl.xml
- the splash screen

The start package is available here: <http://www.CaptainCasa.com/appstart.html>

Download the starter package and unzip its content into any directory. The structure of the directory looks like:

```
<unzipdirectory>
/bin
  start.bat
/jre
  /v_17_15
    /bin
    /lib
  ...
/launchfx
  /resources
    appconfigurl.xml_template    <== to be replaced
    splash.jpg_template          <== to be replaced
  /v_5_0_20140218
    launchfx.jar
```

Now edit the “appconfigurl.xml” so that it points to a server address at which you will provide the “appconfig.xml” file. - And define a “splash.jpg” image of your choice.

That's it! - You now have to check how the starter package is getting on the client of your users. You may zip it and ask the users to unzip and then start the start.bat script. Or you may event package it as an .exe installer. An example for this is given later on.

The appconfig.xml / Server-side definition

Now you need to create the “appconfig.xml” file and store it at the server location, that your “appconfigurl.xml” file points to.

So take this XML as template...

```
<app
  appname="CaptainCasa Enterprise Client"

  appversion="20140218_1"

  launcherurl="${appConfigRootUrl}/ecInt/launchfx/launchfx_5_0_20140218.jar"

  javaversion="17_51"
  javaurl="${appConfigRootUrl}/ecInt/launchfx/jre_17_51.zip">

  <!-- Definition of application jars -->
  <jar jarurl="${appConfigRootUrl}/ecInt/libfx/ecIntfx.jar"/>
  <jar jarurl="${appConfigRootUrl}/ecInt/libfx/jpedal.jar"/>

  <!-- Definition of startables -->
  <start id="Dw"
    name="Demo workplace"
    classname="org.ecInt.fxclient.elements.PageBrowserStarter">
    <vmparam value="-Xmx256m"/>
    <vmparam value="-Xms128m"/>
    <param value="url=${appConfigRootUrl}/faces/workplace/workplaceFX.jsp"/>
    <param value="loglevel=INFO"/>
    <param value="undecorated=true"/>
  </start>
</app>
```

...and update it in the following way:

- Define the appversion of your JavaFX application. Maybe you already have some version numbering that you can apply here. - Please check that the appversion value does only contain simple characters that can be used to form a proper file name. (On client side

a corresponding version directory will be created with prefix “v_”.)

- Define the <jar .../> files that are required by your JavaFX application, and place them at corresponding locations within your server, so that they can be accessed via URL.
 - Please note: the client will download the JAR-files using the corresponding URL-definitions. It may make sense to explicitly name the JAR-files differently with every version that you provide - in this case there will be no caching effect within network (caching proxy etc.).
- Define the <start .../> section. There must be at least one start definition.
- Define the versions of the Java runtime and of the App start launcher and place the corresponding resources at corresponding URL-locations.
 - Create the corresponding zip file for the Java Runtime Environment by simply zipping the content of a JRE installation.
 - The launchfx.jar file is part of the starter package.

Test...!

And now install the update starter package on some client and start the bin/start.bat script. The launcher should show up, downloading your JavaFX application and starting it.

Building an .exe Installer

We use Inno Setup (<http://www.innosetup.com>) for building setup.exe installers. It's much simpler to tell a user “Run this installer!” rather than telling “Unzip and then start the script...”.

The .iss file that we use looks as follows:

```
[Setup]
AppName=CC Enterprise Client FX Launcher
AppVerName=CC Launcher FX @@version@@
AppPublisher=CaptainCasa GmbH
AppPublisherURL=http://www.CaptainCasa.com/
AppSupportURL=http://www.CaptainCasa.com/
AppUpdatesURL=http://www.CaptainCasa.com/
DefaultDirName={localappdata}\CaptainCasa\LauncherFX
DefaultGroupName=CC Enterprise Client FX Launcher
OutputDir=C:\bmu_jtc\eclipse\workspace\ecInt_clientfxloader\temp
OutputBaseFilename=setup_launcherfx_@@version@@
Compression=lzma
SolidCompression=yes
PrivilegesRequired=lowest
SignTool=SIGNTOOL

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Files]
Source: "C:\bmu_jtc\eclipse\workspace\ecInt_clientfxloader\temp\launcherfx\*";
DestDir: "{app}"; Flags: ignoreversion recursesubdirs createallsubdirs

[Icons]
Name: "{group}\Start Enterprise Client"; Filename: "{app}\bin\start.bat";
WorkingDir: "{app}\bin"

[Run]
FileName: "{app}\bin\start.bat"; StatusMsg: "Start Enterprise Client";
Description: "Start Enterprise Client"; Flags: nowait postinstall skipifsilent

[Messages]
WelcomeLabel2=This will install the CaptainCasa FX Client Launcher.
```

Replace all the wording that is CaptainCasa related with your own one.

Under [Files] refer to the directory which holds the updated start package (updated means: with own splash.jpg, with own appconfigurl.xml).

Special Issues

“appstart.finished” - Files in Version Directories

From App Start version 20140226 on, you'll find some strange files “appstart.finished” in the version directories. What's the meaning of these files?

When the App Start launcher downloads e.g. a new version of a JavaFX application, then it loads its libraries (jar-files) according to the definition of appconfig.xml. At the very end, when having downloaded and stored all files, the App Start launcher writes the “appstart.finished” file into the version directory.

A version is only treated as valid, if it contains the corresponding “appstart.finished” file.

This mechanism is used both for the version of the application, of the launcher and of the jre.

Native Libraries

When you want to load native libraries then the following functions are useful:

- In the appconfig.xml you may define files to be downloaded to the client side (“<file .../>” section). These files are directly downloaded into the corresponding “app/v_<version>” directory on client side.
- The batch file that is created for starting the client directs the system library path to the “app/v_<version>” directory on client side.

Result: just add your native libraries as files and directly access them afterwards from your Java code.

In addition you may want to directly extend the system library path by adding “<librarypathextension .../>” definitions to the “<start .../>” definition.

Proxy Configuration

There is a proxy configuration that shows up, if the client fails to connect to the server:



The corresponding data that you user defines is stored in a client file “launchconfig.xml”

that is directly located within the root directory of the launcher. Its content is:

```
<launchconfig>  
  <proxy host="132.132.132.132"  
    port="8080"/>  
</launchconfig>
```

Open Issues

The launcher framework is in development currently. All functions described in the previous chapters are implemented and tested. But development continues...

The nice thing: the launcher may exchange itself, because it is managed in versions as well. So a new version of the launcher can be easily rolled out.

- Signature Management
 - ...would be nice if the launcher could check if all parts that it downloads from the server side are signed with the same certificate
- Visual Appearance
 - ...currently the launcher has a rather technical UI...
- Windows only...
 - ...currently the launcher only supports Windows scripts (.bat).
 - ...currently the URL to Java-versions are Windows-only.

Change Log

Version 20141008

- In addition to JAR-files you can now specify any other file to be downloaded as well via. See appconfig.xml definition, section “<file .../>”
- The batch file now contains the definition of the system library path, pointing into the directory in which the application files are stored on client side
- You now may define operating system dependent download locations for the jre-zip-file (appconfig parameters “javawindows32”, “javawindows64”).

Version 20150425

- There was a bug when reading the URLs for loading the Java runtime - parameters “javawindows32”, “javawindows64”, “javawindows32” were not properly interpreted.
- You may now extend the “start” information by library path information - see chapter “appconfig.xml Configuration File”

Version 20150512

- When updating the JRE and/or the launcher-jar itself then so far the user had to cancel the launcher window and then had to restart the launcher from the start menu (Windows). Now there is a restart button within the launcher, simplifying this procedure dramatically...

CaptainCasa GmbH
Hindemithweg 13
D - 69245 Bammental
06223 484147

<http://www.CaptainCasa.com>
info@CaptainCasa.com

CaptainCasa Enterprise Client