Bootstrapper (alternative download method of the SmartClient)

The bootstrapper (bootstrap.jar in application_server\server\web-app\root) can be used in 2 ways:

1> with a bootstrap.jnlp (and starting it up through webstart)

2> as a standalone launcher, by downloading the jar and double click (java -jar boostrap.jar), supported with Servoy 8.3.2+

in both ways, you can rename the jar (or jnlp file) to what ever you want like "yoursolution.jar/jnlp" and have multiply copies of it for specific clients (like tenants)

With this new release (8.3.2) the bootstrapper has support for native libs or native resources. The correct resources should be downloaded and native libs are added to the system path so that they can be used with System.loadLibrary(name)

The standalone bootstrapper is added because Oracle is dropping support for WebStart (not sure when, or if Java 11 will still have it)

Through webstart:

To speed up (especially under windows) the smart clients startup, you can use a special bootstrap.jar

From Servoy 8.1.0 ang higher you can find the bootstrapper.jar in the SERVOY_INSTALL/application_server/webapps/root directory.

This by passes for the most part webstart, webstart only downloads and verifies the bootstrap.jar and jnlp.

By default the downloads will go into [user-home]\.servoy\libCache\[hostname], but from version that is shipped with Servoy 815 and higher you can also set that home property to any location you want by using a property tag:

<property name="jnlp.bootstrapper.home.dir" value="d:/temp/bootstrap/"/>

inside the <resource> section just below the <j2se> tag.

if 1 file changes on the server it will delete the cache and downloads it again.

The jnlp file can be fully configured what ever you like (or should be filled in correctly)

{{YOUR_FULL_HOSTNAME_AND_CONTEXT_IF_DEPLOYED_THROUGH_WAR}} : That needs to be filled based what the users sees in the browser (so host name with port and with context if the servoy app server is not deployed at the ROOT)

The rest between the <information> tag can be filled in to what ever you want.

between the <resources> tag you can specify any java version (default is only 1.8 added you can add another line that says 1.7)
and this only has a reference to the bootstrap.jar

If you sign your self then you can change anything in that jar (so including the application-name manifest property)
also if you have adjusted the bootstrap.jnlp file (can be renamed also to what ever you like) exactly how you want
Then you can add that jnlp file to the jar file under:

JNLP-INF/APPLICATION_TEMPLATE.JNLP

see: http://docs.oracle.com/javase/7/docs/te ... dJNLP.html

and then sign the jar again, this makes the jnlp file also signed so it doesn't matter at all what kind of things you use/set in it

What should also go between the <resources> tag are all the native resources of your beans/plugin.
So you you have a plugin or bean that has in its jnlp file one or more native resource like the browser suite:
https://www.servoyforge.net/projects/br ... e.jar.jnlp

Then you need to copy those over so the <resources> tags with "os" and/or "arch" needs to be in the bootstrap.jnlp file

The last thing are the arguments:

<argument>{{SOLUTION_NAME}}</argument>

the first argument should always be the solution name that you want to load (so if you want to load directly a specific solution)

You can also not specific it then the select solution dialog is shown.

You can also make profiles (that is what servoy profiles basically is, pushing arguments to the client)
Then you can add multiply arguments to the client by adding more, like adding:

<argument>system.property.SocketFactory.tunnelConnectionMode:http</argument>

as the second argument (the first should then always be a soluton or just "servoy_client")

If you add that argument then the tunnel will be forced in http mode (if the server is configured in "http&Socket)

This way you can push any system.property to servoy that you want.

if you have your own arguments that you now push like: argument=owner:xxxxxxx. then you need to give that as:

<argument>argument:owner:xxxxxxx</argument>

The *bootstrapthreadpoolsize* argument is used to configure the maximum number of concurrent connections used to download the application files.

<argument>bootstrapthreadpoolsize:{{MAX_THREAD_POOL_SIZE}}</argument>

The default value for the maximum concurrent connections is 8. If you want to download the resources one by one (in case of network problems for instance), then you can set the *bootstrapthreadpoolsize to 1.*

bootstrap.jar and bootstrap.jnlp are found in the ROOT of the context
So that is in a default servoy installation: \application_server\server\webapps\ROOT
in a war file it should be in the root of the war file

Then you can access it through:

http://hostname:port/bootstrap.jnlp

(and the above code base should be then http://hostname:port)

Standalone starter: (since 8.3.2, only use able from that Servoy release)

With the stand alone starter the customer doesn't download the bootstrap.jnlp file but the boostrap.jar file itself.

the configuration that you normally would do in the bootstrap.jnlp must then be done by editing the bootstrap.properties file that sits in the root of the bootstrap.jar.

by default some examples are given:

codebase=http://localhost:8080/test/
solution=test
bootstrapthreadpoolsize=8
arg0=system.property.MyKey:MyValue
arg1=argument:owner:1
arg2=owners:2
#jnlp.bootstrapper.home.dir=

codebase: is the same as the code base property in the jnlp file above. This must be the application servers url to the root of the servoy application.

solution: the solution name to load

bootstrapthreadpoolsize: Same as above, the number of threads handling the download

jnlp.bootstrapper.home.dir: The dir where the downloaded files should go into (standard is this {user.home}\.servoy\libCache)

Arguments can be given with sequence numbers argX=

Where system properties can be set like system.property.MyKey:Value

What is currently not supported in this way is system properties or vm arguments that needed to be set before starting java. (like max memory)

Because the starter is started with plain java -jar and doesn't currently relaunch itself.

If you change the properties file then the signing will be broken, the signing must then be removed (META-INF*.RSA and *.SF) or better resigned with your own certificate.

For making a fully standalone installer that also ships java your need to configure the above boostrap.jar correctly

Then download an openjdk jre installation of Java 8 (Java 12 is also fine but that could result in configuring the vm arguments in later on to enable all features) here: https://adoptopenjdk.net/releases.html (this is the other platforms section of the main page)

Now you can click on a for example 64 bit of Jdk8 where you get the option to download just the JRE (as a zip)

create a directory that has in the root the bootstrap.jar

in that directory extract the jre.zip to a jre dir (if the zip already containts a parent dir, rename that directory to jre, so you have a jre\bin\java file.

create a batch/sh file for windows/mac that starts up the command:

jre\bin\javaw.exe -jar boostrap.jar