JSmooth 0.9.9-7 User Manual

Build 20070519-254

Rodrigo Reyes

Revision History

Revision 1.0	10 Sep 2003 First draft	RR
Revision 2.0	19 May 2007 Update for release 0.9.9-7	RR

Table of Contents

1. Introduction 1
- 1.1. What is JSmooth? 1
- 1.2. Web Support 2
- 1.3. Features 2

2. JSmoothGen: Windows Project Editor 3
- 2.1. Running the JSmoothGen project editor 3
- 2.2. Creating your first exe 4

3. JSmoothGen structure 4
- 3.1. Skeleton Selection 4
- 3.2. The Windows Executable 5
- 3.3. Defining the Java Application parameters 7
- 3.4. Selecting the most suitable JVM 9
- 3.5. Configuring the JVM 11

4. Command Line 13
- 4.1. Running jsmoothcmd.exe 13

5. Using JSmooth as an Ant Task 13
- 5.1. Defining the ant task 13
- 5.2. Using the `jsmoothgen` task 14

6. Using the JSmooth JNI functions 14
- 6.1. Examples 14

7. Frequently Asked Question 16
- 7.1. Wrappers Behaviour 16
- 7.2. Trouble & issues with wrappers 18

8. License 18

9. Third-Party libraries used 19

1. Introduction

1.1. What is JSmooth?

JSmooth is a Java Executable Wrapper generator with advanced JRE detection features. It builds standard Windows executable binaries (.exe) that contain all the information needed to launch your java application, i.e. the classpath, the jvm version required, the java properties, and so on.

It also offers full control on deployment issues such as missing, old, or broken JRE installed on the end-user computer. The JSmooth wrappers provide smooth solutions, such as automatic JRE download and installation, redirection to a web site, or simply a meaningful message to the users on what's going wrong.

Because JSmooth is 100% java, it can build windows executables from any environement running Java, including linux, solaris, etc.

1.2. Web Support

- The JSmooth web page is located at http://jsmooth.sourceforge.net/. Please visit it for the latest news.
- The JSmooth Project web page is hoster at http://sourceforge.net/projects/jsmooth.
- Should you find any bug, please submit it to the jsmooth tracker [http://sourceforge.net/tracker/?group_id=86730]. For any question relative to JSmooth, please use the forum [http://sourceforge.net/forum/?group_id=86730] (hosted at sourceforge's).

1.3. Features

1.3.1. Smooth User Experience

- Your users see and use a standard windows Exe binary. No more technical explanation on .jar, on javaw association, etc.
- The JSmooth EXE are smart: They know how to swiftly search for all the JVM installed on a computer, and determine which best suits the requirements of your application. If no VM is found, your end-users are in good hands with jsmooth: the wrappers can redirect them to a web page, or even better, they can propose the user to automatically download and install a Java Environment.

1.3.2. Flexible automatic Java VM detection

- Detects the location of any Sun JVM installed. The wrappers use several strategies to detect all the JVM available on a computer, using windows registry, environment variables, or windows path (basically, whatever gives hints is used).
- Detects and uses Microsoft's JView (for 1.0 and 1.1 Java applications only), if available.
- The JVM search sequence is fully customizable using the GUI. You can force the executable to search in the path first, and in the registry last, or in JAVA_HOME first. We have all the flavours!
- Sometimes it's more convenient to bundle your own JRE with your application. JSmooth can deal with that too: you just need to define in which folder the JRE is packaged. It falls back nicely to a standard JVM search if the JRE is not where it should be.
- Specify which versions of the JVM are compatible with your software! You can set a minimum version, but also a maximum JVM version.

1.3.3. Graphical User Interface

- A graphical project editor makes configuring and customizing your exe wrapper a breeze. The inline help balloons provide non-intrusive yet extensive information on each parameter available.
- Associate an icon to your executable, using a .ICO, .PNG, or .GIF file (automatic color reduction is done if necessary).

1.3.4. Application configuration

- Easy java properties configuration: specify the tag=value pairs you want the wrappers to pass to your java application.

- Pass also environment variables from the system: just define the java properties to pass to your application, and use the standard Windows %VARIABLE% syntax.
- Some special variables that are normally not available to Java program, such as the executable path and name, and even the Windows computer name, can be passed as well!
- Specify all the classpath configuration, adding .jar, .zip, or directories, with an intuitive GUI.
- Need to modify the current directory used by your java application ? The wrappers will handle this for you.

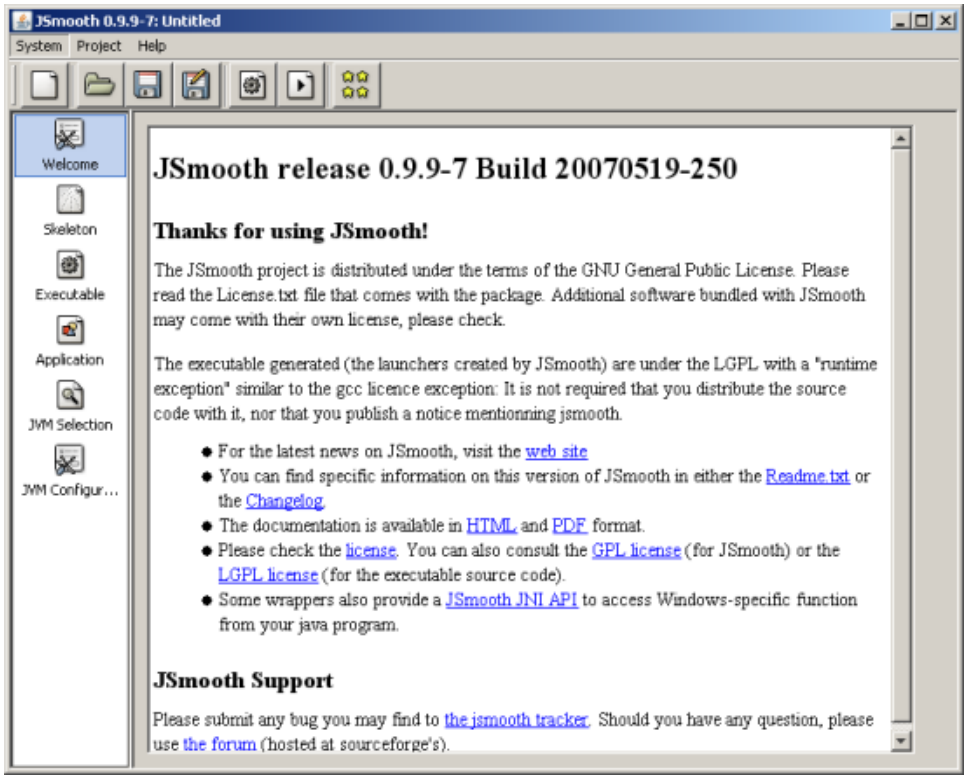
1.3.5. Wrappers

- Provided with several exe wrappers skeletons, for either GUI, console-based application, or windows services !
- If none of the provided wrappers pleases you, just create your own ! It's open-source and easy to plug in the jsmooth framework!

2. JSmoothGen: Windows Project Editor

2.1. Running the JSmoothGen project editor

The JSmoothGen application is the project editor. It provides a (hopefully nice) graphical user interface for the configuration of all the parameters available to the wrappers.



The JSmoothGen Application

2.2. Creating your first exe

Building your own EXE wrapper for your java application just needs a few simple steps: specify the name of the executable, define the main java class, optionally a classpath if it needs more than its own jar, and that's all.

2.2.1. Quick Start

The minimum data you need to start the creation of the binary executable for your application is to specify a classpath and the fully-qualified name of your main class. You can configure both in the `Application` panel (add new jars or directories using the icon with a yellow + sign, and type the class name in the Main Class field.

Once done, go to the `Windows Executable` panel, and type the name of the executable in the Executable Name field. Do not forget to add the .exe suffix, for your executable name. Something like `my-app.exe` should be fine.

Of course, you can specify many more options for the java wrapper, but this is enough for a basic java application. Now, click on the save button (or in the File+Save menu) and select a filename for the project. This is an important step, because all the path stored in the file, and displayed in the GUI are relative to this project file. Selecting a location under the root directory of your project is a good (and safe) choice.

Once your project is saved, it is just a matter of clicking on the Project+Create Exe. And that's it. You can use the Project+Run Exe to run the program, or use the windows explorer to run the executable.

Warning

You can modify the location of the every file in the project, but you must always keep this simple rule in mind:

- All the files saved and displayed in the Graphical User Interface are relative to the project file.

3. JSmoothGen structure

3.1. Skeleton Selection

The JSmooth application is based on a template system called "skeleton". Choosing a skeleton is the first step in the creation of a jsmooth executable.

A skeleton is a template with a specific behaviour. For instance, the following skeleton are bundled with JSmooth as of version 0.9.3:

- The Console Wrapper is a skeleton designed for console applications (ie. which are run in the windows console). This skeleton takes care of passing the command line arguments to java application, it ensures the i/o are routed to the current console, and so on.

  If no JVM is found, the application displays a text message to the standard output.

- The Autodownload wrapper is designed for standard Windows applications, which do not use console.

  If no JVM is found, the wrapper displays a customized message, and offers the end-user to automatically download and install a JRE. The version of the JRE to download is a configurable parameter of the wrapper, and uses the Sun's autodownload feature initially provided for Web Start.

- The Custom Web Download wrapper is a wrapper for GUI-only applications (no console)

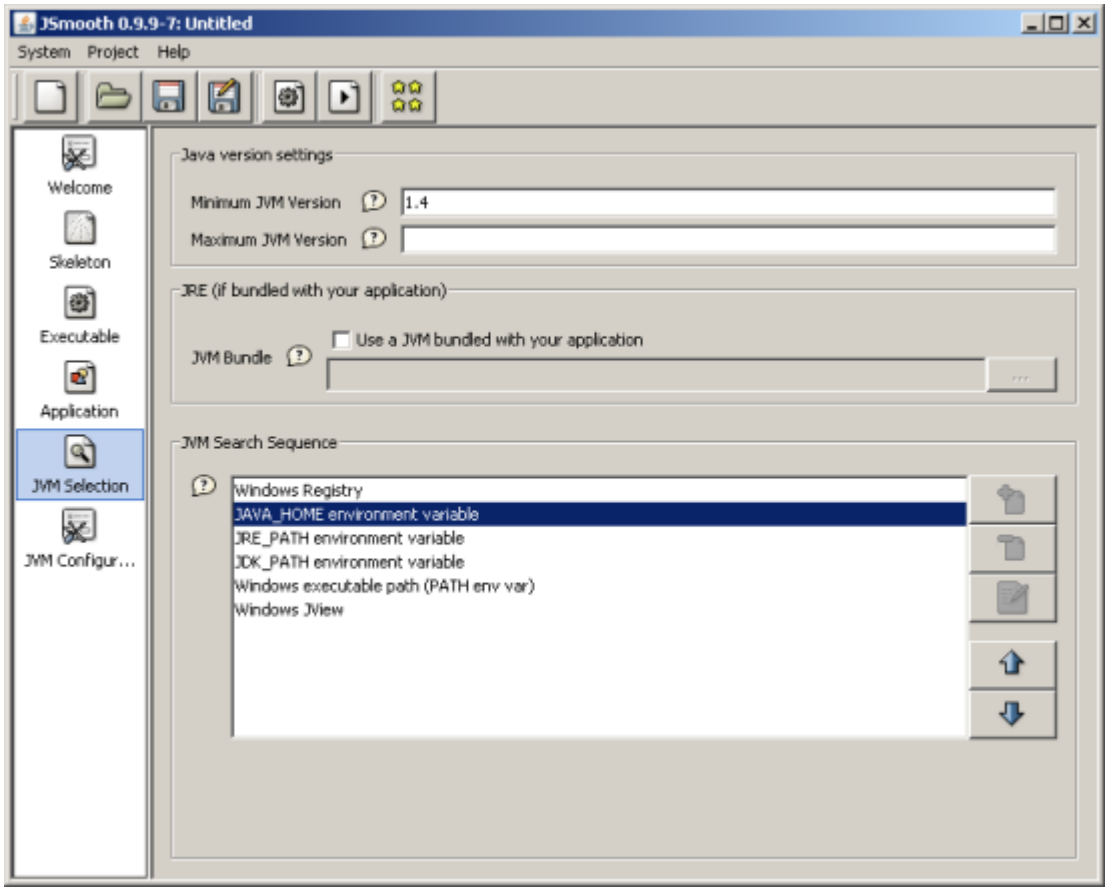
  If no JVM is found, the wrapper displays a customized message, and offers the end-user to download a customized file from the internet. The URL is a configurable parameter of the wrapper, and accepts http, https, and ftp. The file download is automatically executed or displayed (according to the nature of the file).

- The Windowed Wrapper is designed for standard Windows applications, which do not use console.

  If no JVM is found, the skeleton displays a Windows OK/Cancel alert. If the user selects the OK button, the default web browser is launched on a web page. Both the message of the alert and the URL of the web page are configurable.

- The Windows Service Wrapper provides a wrapper for application designed to work as services.

  The wrapper can be installed, uninstalled, started, or stopped just like any other windows service. If no JVM is found, the service does not run, and a specific message is logged.



The Skeleton selection panel

Each skeleton has its own set of option, most of them are optional. Be sure to check them all, as they can modify drastically the behaviour of your wrapper. Please refer to the inline help for the details.

3.2. The Windows Executable

The windows executable created by the JSmooth wrapping system can be configured in many ways.

- The executable name is the most obvious parameter. This is simply the name of the win32 file created by JSmooth.

- The current directory can be modified to suit your needs. For instance, if you want your executable to be in a `bin\` sub-directory, but still want it to access the resources as if it were in the base directory, it may be convenient to modifiy the current directory to be `..`

  When this parameter is specified, the wrapper simply changes the current directory with the value "as is". For instance, to change the current directory to a `res` subdirectory, write `res` in the Current Directory field.

  ## Warning

  When run as a Windows Application, the default current directory is always the directory of the executable binary. However, for a console wrapper, the default current directory is the path where the command line is currently tied.

  Changing the current directory for a console application is probably not what you want. A common issue is for a console application to find resources which are located in the file system (i.e not in the jar of the application). Lots of application work this way, but Java applications, unfortunately, do not have access to the information they need: the location of the binary executable they are launched from. A work-around is to use an environment variable specific to your application. This is was ANT does, for instance.

  You can do something equivalent with JSmooth without using an environment variable (which may be complicated to set under windows). Instead, define a Java Property, and set `${EXECUTABLEPATH}` as its value. The effect of this is to pass to your application a java property that contains the directory where the executable is located. Just use `System.getProperty()` call to retrieve the value.

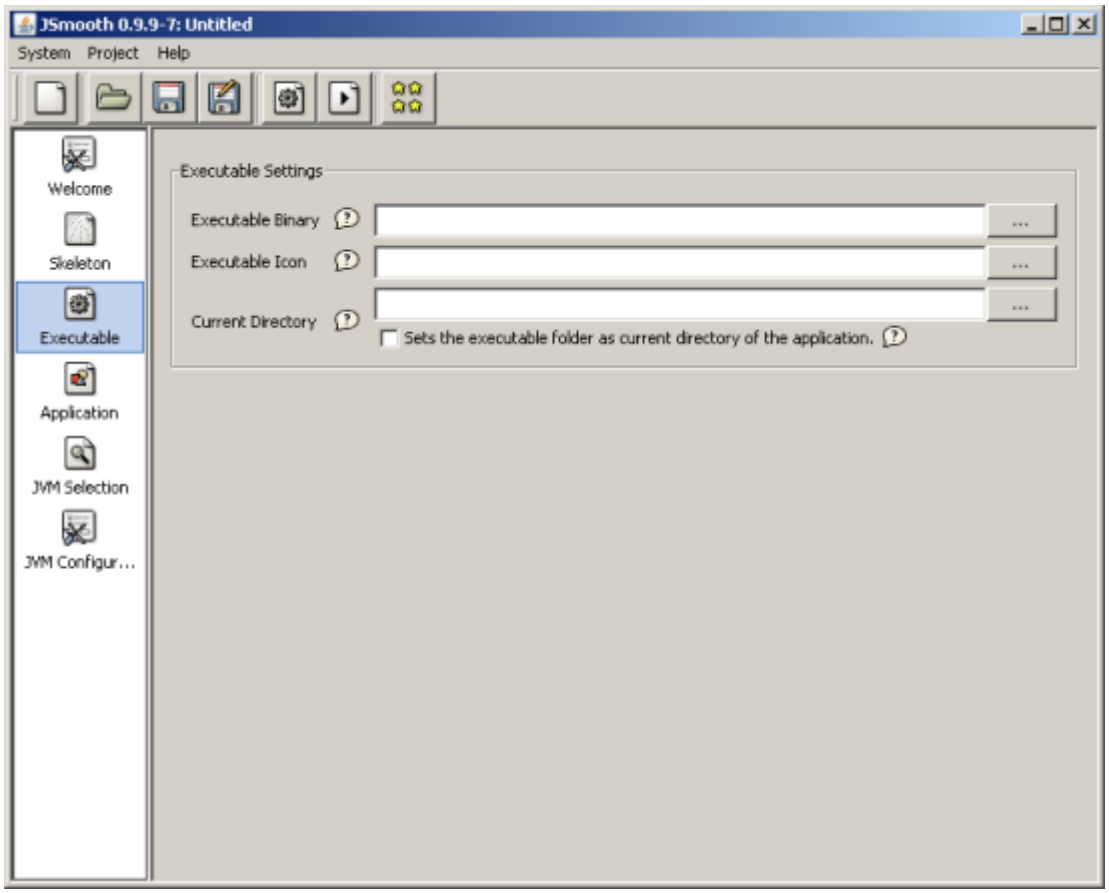
  For GUI application, it's easier: just enable the option "Sets the executable folder as current directory of the application" and define a standard file association in Windows. Alternatively, you can also use the JNI features of jsmooth to retrieve the executable path.

- The executable icon field specify the icon image that is associated to the executable, under Windows. The default configuration of JSmoothGen support Windows Icon files (.ICO) as well as the standard types supported by the JVM (GIF, PNG, and JPG files).

  The current implementation only sets up a single icon image in the executable. Here is a brief description on how JSmoothGen creates the executable icon:

  - If the user selects a standard Windows icon (.ICO), it chooses [1] first 32x32 icons, then 64x64, then 16x16.

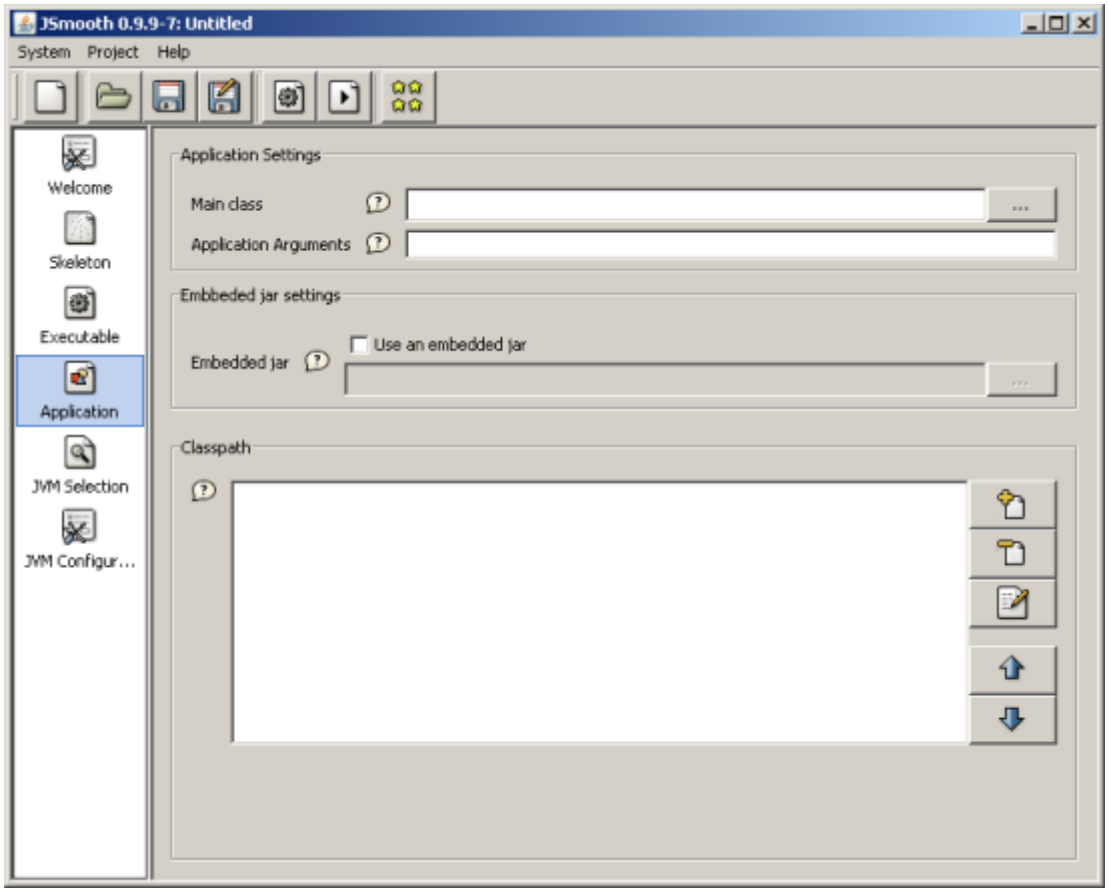
  - If the user selects any other recognized image file, the image is scaled to be 32x32, and quantized to fit in 256 colors. The color reduction is rough and it is always better to use an image processing software for the job.



The Executable panel

3.3. Defining the Java Application parameters

This is the panel where you can specify all the parameters related to the java application itself.



The application-specific configuration panel

- Edit the Main Class field to specify the class used as entry point of the program. This field is mandatory.
- The Arguments field allows you to specify default parameters for the application. Those arguments are passed as if they were specified on the command line, and are made available in the String array of the main method by Java.

Warning

The Arguments may be overriden by some wrappers. For instance, the Console Wrapper always uses the arguments from the command line instead.

- Using the Embedded Jar field, you can optionally specify a jar to be embedded (as a resource) in the EXE wrapper. This jar is extracted each time the application is launched, therefore you should keep it as small as possible.

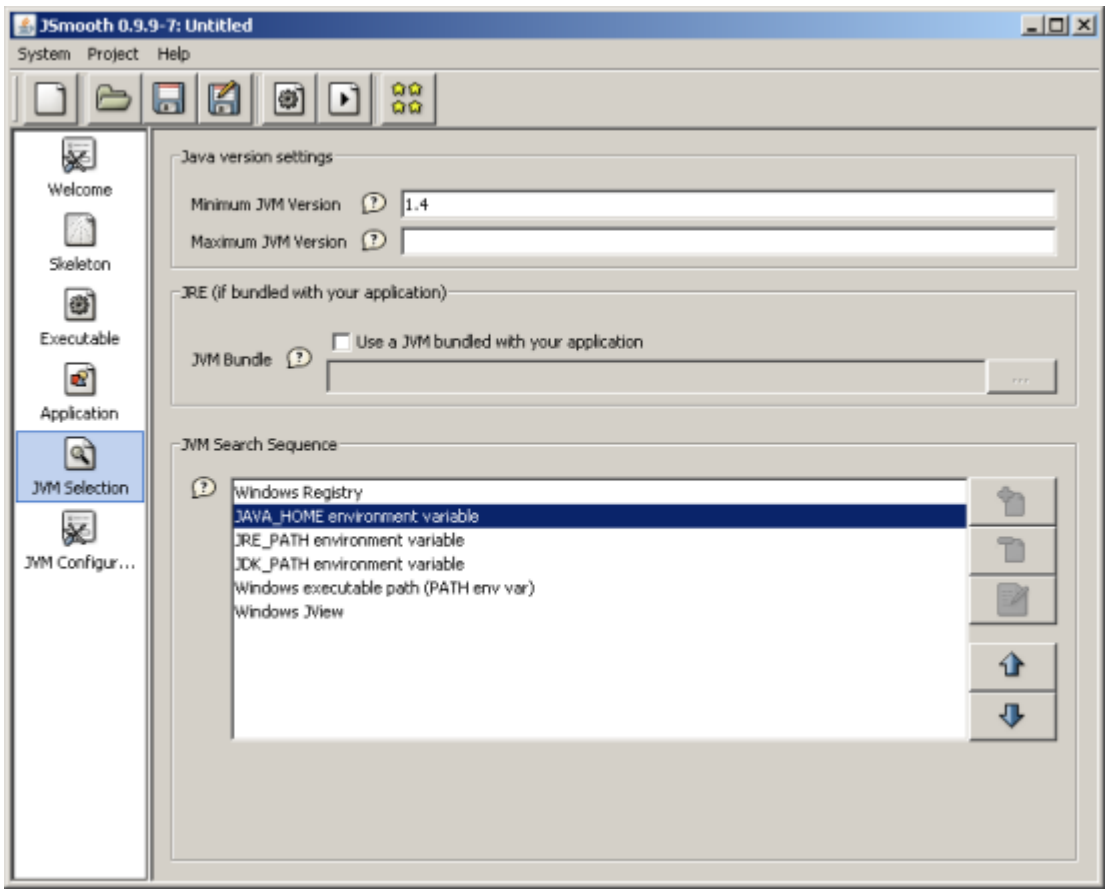
  This is specially useful for application packaged as a single EXE file.

- The ClassPath list provides a mean to edit an ordered list of classpaths. To add a classpath, click on the  and select either a .JAR, a .ZIP or a directory using the file chooser. You can remove or edit the classpath entries with the  and  buttons.

The items can be ordered to reflect the class look-up priorities of your application. Select an item, and press either  or  . The rules are simple: the higher in the list, the higher the priority in the classpath search sequence.

3.4. Selecting the most suitable JVM

Click on the JVM Selection tab to specify the Java look-up parameters.



The JVM Selection panel

3.4.1. Java Version

If your application requires a minimum or a maximum version of Java to run, the Version of Target VM subpanel will probably be of high interest for you.

Just leave the field empty if you do not wish to any constraint on the JVM version required for your application.

To specify a version, either as a minimum or a maximum, you must specify the major, minor, and sub-minor version number you want to set the constraint on. You don't need to specify all of those three numbers, but remember that if you leave it empty, it is considered as 0.

The values are inclusive. For instance, if you specify 1.4 as a maximum means that the java wrapper accepts 1.4.0 JVM, but not 1.4.1.

If your application runs exclusively with JVM 1.2 and 1.3, but not 1.1 nor 1.4 or above, just specify 1.2 for the minimum version, and 1.3.99 as a maximum version.

3.4.2. JVM Bundle

Even if the wrappers of JSmooth provide a reasonnable user experience for the users who do not have Java installed, you may wish to bundle a JRE with your application. In such a case, tick the Use JVM Bundle checkbox, and specify the directory location of the JRE.

Warning

What the "JVM Bundle" option really specifies, is a path (relative to the generated EXE) where a JRE can be found. This is NOT an option to bundle a JRE in the EXE, as many people think.

For the option to work correctly, you have to put a JRE in a directory near the EXE (generally in a subdirectory called "jre" or whatever). Once the exe is generated, it will FIRST try to locate the JRE at the location mentionned. If it can't be found there, then it will fallback in the normal jre look-up mode (search for a jre or a jdk in the Windows registry or in commonly-used environment variables). There is no JVM-version check when using a bundled JRE, as the packager is supposed to bundle a suitable JVM for the application.

Consequently, if you use the "Bundle JVM" option, you'll need to install yourself the JRE at the same relative path to the EXE.

For instance, take the following example:

```
+myprog/
|- myjar.jar
|- lib/
+ mylib1.jar
+ mylib2.jar
|- jre/
+ [full jre stuff here]
|- myexe.exe
```

In this case, the generated myexe.exe tries to use the jre in the "jre" sub-dir as its first-choice JRE. To deploy it, either simply zip all the "myprog" directory, or tell your favorite installer to set-up the jre directory in the same configuration as in your original folder tree.

3.4.3. The JVM Search sequence

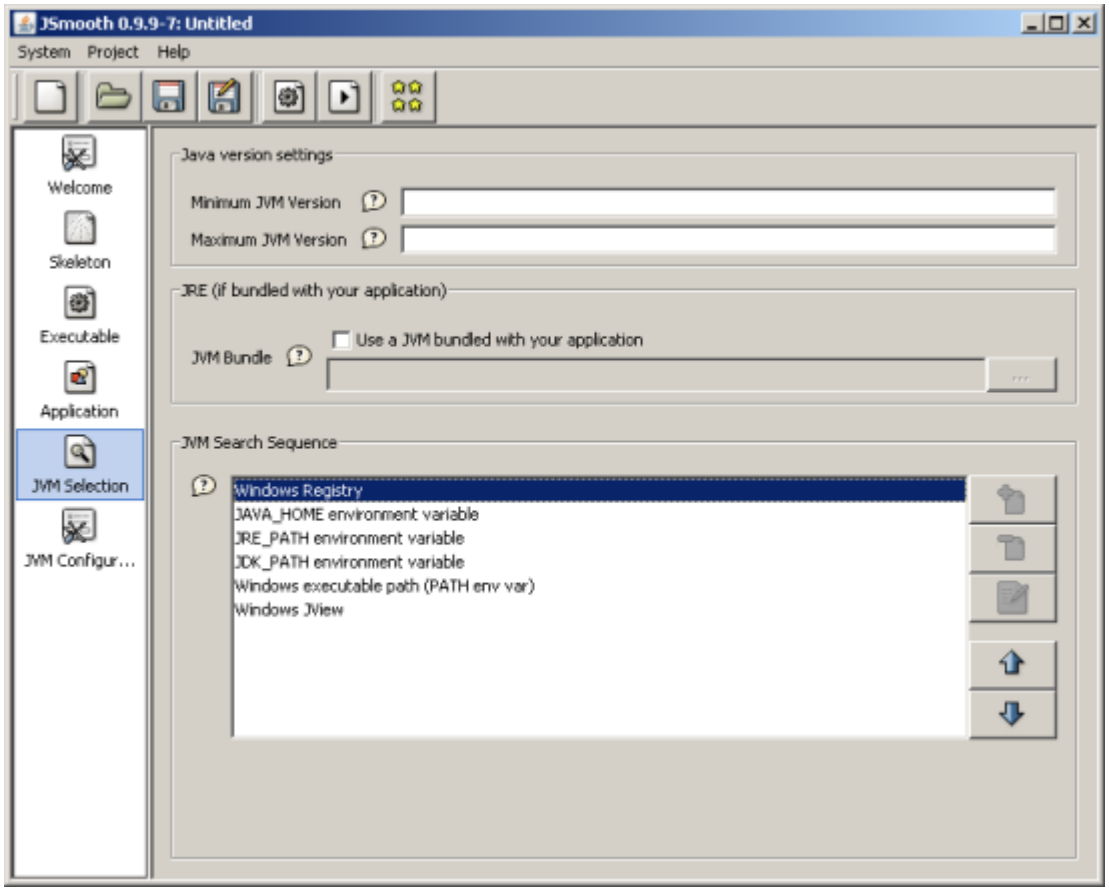
The wrappers are able to detect most, if not all, the JVM available on the end-user computer. Before launching the java application, they try to find a JVM using a preference order. The default is to use the Windows registry to look-up the JRE that have been installed, then use some environment variable, and finally try to use Microsoft's JView.

This search sequence is fine for most configuration, however you can still modify it to best suit your needs.

Use the  and  button to modify the priorities..

3.5. Configuring the JVM

The JVM Configuration offers the possibility to specify the parameters passed to the Java Virtual Machine when launching your application.



The JVM configuration panel

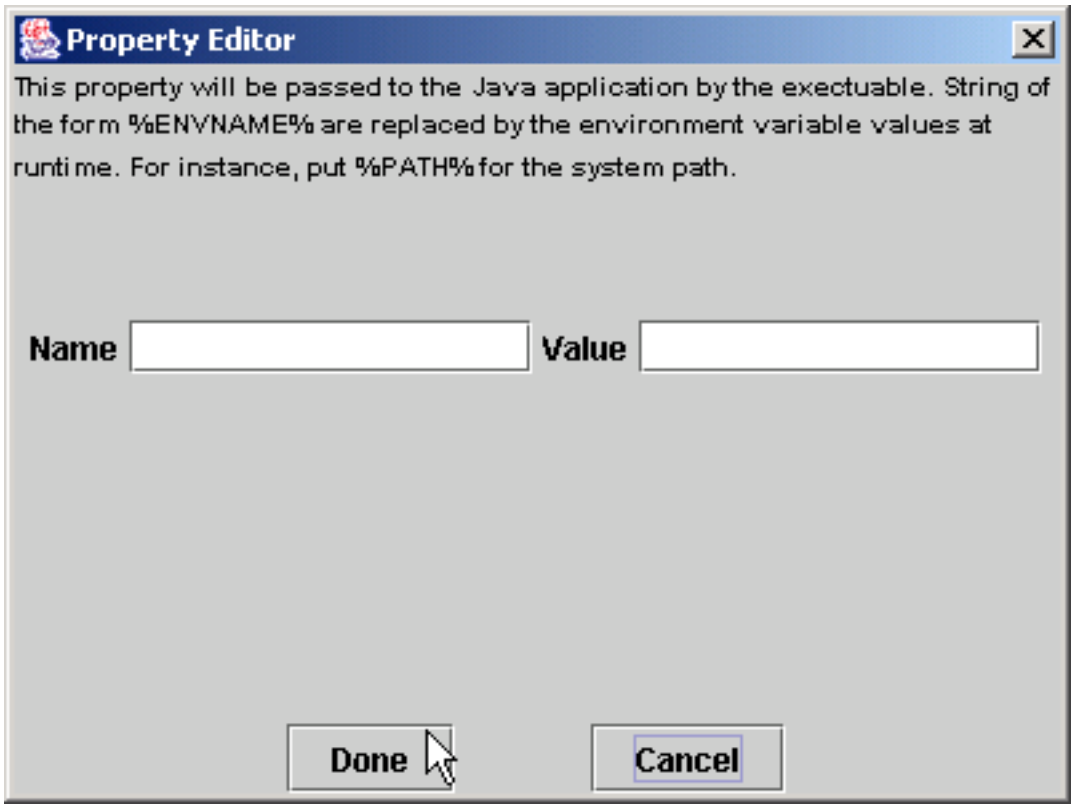
3.5.1. Options

The Options panel provides editable fields to configure the memory configuration of the JVM. Specify the numeric value in the text field, and the unit in the combo box.

3.5.2. Java Properties

Java properties are name/value pairs that are passed to the JVM, and are accessible by java applications using the System.getProperty() method.

To add a new Java Property, click on the  . A dialog box pops up.



Specify the name and the value for this property, and click on the Done button.

3.5.2.1. Special Values

You can use the Java Property mechanism to pass special values to your java app.

- Environment Variable: If you pass a string of the form %ENV% in the value field of the property, the wrappers will replace the string with the content of the ENV environment variable.

  For instance, to pass the content of the JAVA_HOME environment variable, adds %JAVA_HOME% in the Value field.

- JSmooth also makes some special variable accessible for your application.

Form	Meaning
`${EXECUTABLEPATH}`	Replaced by the path to the executable binary. For instance, if the executable binary launched is located at c:/program files/jsmooth/test.exe, this variable is replaced with c:/program files/jsmooth
`${EXECUTABLENAME}`	Replaced by the name of the executable binary. For instance, if the executable binary launched is located at c:/program files/jsmooth/test.exe, this variable is replaced with test.exe
`${COMPUTERNAME}`	Replaced by the name of the computer.
`${VMSELECTION}`	Replaced by the name of the selection method used to spawn the JVM. The value replaced is typically one of the following values:

	• `registry`, when the VM location is found in the registry • `jdkpath`, when the VM if found in the %JDK_HOME% variable • `jrepath`, when the VM if found in the %JRE_HOME% variable • `javahome`, when the VM if found in the %JAVA_HOME% variable • `jview`, if the VM is JView • `exepath`, if the VM was found in the windows path • `bundled`, if the VM bundled with the application is used
`${VMSPAWNTYPE}`	This variable is replaced by one of the following values: • `JVMDLL` if the JVM is spawned using a DLL loading and instanciation • `PROC` if the JVM is spawned in a new process using the Windows' CreateProcess function

4. Command Line

4.1. Running jsmoothcmd.exe

You can build a wrapped java application using the jsmoothcmd.exe command. However, note that it only takes as argument a project file created by the JSmoothGen graphical application. In other words, you can't create a project from scratch using only the jsmoothcmd program, which is provided only as a convenience for scripting purposes.
Usage

```
jsmoothcmd.exe [myproject.jsmooth]
```

The .jsmooth suffix may be ommited. The application either creates the executable binary successfully, or returns an error message.

5. Using JSmooth as an Ant Task

JSmooth can also be used as an ant task. This permits using it as a tool integrated in a standard ant deployment chain.

5.1. Defining the ant task

This is the first step to be able to use jsmooth in your ant scripts. You need to add a `taskdef` tag that provides to ant the information required to create the `jsmoothgen` task.

```
<taskdef name="jsmoothgen"
     classname="net.charabia.jsmoothgen.ant.JSmoothGen"
     classpath="path/to/jsmoothgen-ant.jar"/>
```

If your jsmooth project uses the Sun's JIMI library to recognize .ico files, you must add it to the classpath as well.

5.2. Using the `jsmoothgen` task

The task takes two mandatory parameters:

- `project`: a reference to the project file.
- `skeletonroot`: a reference to a directory where the skeletons can be found. Be careful, this is not the directory where the skeleton used by the project can be found, be rather the directory one level up.

Here is a sample, extracted from the jsmooth build.xml file:

```
<jsmoothgen project="jsprj/jsmoothgen.jsmooth"
      skeletonroot="${dist}/skeletons"/>
```

6. Using the JSmooth JNI functions

Some of the jsmooth wrappers provides Windows-specific JNI features that allow your application to optionnaly access native information.

In order to use those features, you must enable the JSmooth JNI option, and add the `jsmooth.Native` calls in your java code. You need to build your project using the `jnismooth.jar` library, but don't need to package it with your application, as it is embedded (as well as the native functions implementation) in the wrappers.

Before using any JNI feature from the `jsmooth.Native`, you must check that the native methods are correctly bound to their native implementation by using the `jsmooth.Native.isAvailable()` method. If this method returns false, you cannot use any of the native method.

6.1. Examples

6.1.1. Start the default application for a file

Using the `shellExecute()`, you can start the default application associated by windows to a document and perform any of the action among `SHELLEXECUTE_OPEN` (opens the file for visualization), `SHELLEXECUTE_EDIT` (opens the file for editing), `SHELLEXECUTE_PRINT` (print the document), `SHELLEXECUTE_FIND` (launch the windows Find utility on a specific folder), `SHELLEXECUTE_EXPLORE` (launch the windows file Explorer tool on a specific folder).

```
 if (jsmooth.Native.isAvailable())
   {
    jsmooth.Native.shellExecute(jsmooth.Native.SHELLEXECUTE_OPEN,
                                "c:\\myfile.pdf",  null, null,
```

```
                                    jsmooth.Native.SW_SHOWNORMAL);
  }
```

6.1.2. Launching the default web browser

There are some java libraries available around to launch the default browser on an URL, but using the `shellExecute()` is probably the best solution for the windows platform on this traditional problem.

```
URL target = "http://jsmooth.sourceforge.net/";
if (jsmooth.Native.isAvailable())
  {
    jsmooth.Native.shellExecute(jsmooth.Native.SHELLEXECUTE_OPEN,
                                target.toString(),
                                null, null,
                                jsmooth.Native.SW_NORMAL);
  }
 else
  {
    // Use a traditionnal java-based guessing method
    // to find and launch the default browser
  }
```

6.1.3. Reboot the computer

It may be useful to reboot or log off the computer/session programmatically at a given time. You can use the jsmooth `exitWindows()` method to shutdown or reboot the computer, but also disconnect the session of the user.

```
// Log off the user session
if (jsmooth.Native.isAvailable())
  {
    jsmooth.Native.exitWindows(jsmooth.Native.EXITWINDOWS_LOGOFF);
  }
```

6.1.4. Retrieve information on a disk drive

There is no way to distinguish between a removeable media, a cdrom drive, or a fixed disk using the standard java API. It is however a good usability point to use different icons to display fixed, removeable medias, remote drives, or cdrom. It may also be useful to know if there are enough free space on disk before saving a file, or to be aware that the filesystem used for a media is FAT16, in order to alert the user of a possible loss of precision in the filenames.

The `jsmooth.DriveInfo` object provides all this information for free, provided your application is launched with a jni-enabled jsmooth wrapper.

```
File f = new File("d:/my/file/somewhere");
jsmooth.DriveInfo di = jsmooth.Native.getDriveInfo(f);

if (di.getDriveType() == jsmooth.DriveInfo.DRIVE_REMOVABLE)
{
   // This file is on a removeable drive !
```

```
}

if (di.getFreeSpaceForUser() < (1024*1024*64))
{
  // there are less than 64MB free for the
  // user on this drive !
}
```

7. Frequently Asked Question

7.1. Wrappers Behaviour

7.1.1. How does a wrapper extract the jar file?

The jar file extraction is wrapper-specific. However, all the wrapper provided with JSmooth as of version 0.9.3 extract it in the default temporary directory. This is the standard behaviour expected by Windows application.

7.1.2. What happens to the extracted jar file when the java application exits?

Whenever possible, the wrappers delete the file on exit. Note however that this is not always possible. For example, an instance of a JVM created using the JVM DLL does not unload cleanly. This behaviour (not to call it a bug) is documented by Sun, and there is no known work-around for it.

Both Windowed and Console wrapper always prefer the JVM instanciation methods that allow them to delete the temp jar after the application exits. Note that embedding a jar in the exe is not required, it is always a safe choice to just leave it on the filesystem near the executable, either in the same directory, or in a sibling or parent directory.

Note also that deleting the files in the windows TEMP directory is not required for Windows applications, and most application just leave them, letting the operating system manage it. The standard behaviour of MS Windows is to propose the user to delete the temporary files when the disks are running low on available space.

7.1.3. Jsmooth allows only one embedded jar in the executable binary. Does it mean that all the java application must be entirely contained in this jar ?

No. While the jsmooth wrappers may use a jar to bootstrap the java application, you can reference other jar files in a jsmooth project. Those jar files will be accessible to your executable binary, provided the path specified is found at runtime.

It is often convenient to put the main jar in the executable, but you can just put a small jar which purpose is just to bootstrap the application (for instance by displaying a splash screen with specific information and calling the real main class).

7.1.4. How can I run the ant task or the command line in headless mode (so that it can run on unix servers without X-Window)?

The command line always starts in headless mode. For the ant task, you must configure the java environment running ant to run in headless mode. You have to setup the following property:

```
java.awt.headless=true
```

7.1.5. How do I pass JVM-specific arguments with the wrappers?

There is no much point in passing JVM-specific arguments for wrappers that are supposed to use any available JVM on the end-user's computer. However, JSmooth wrappers can use jsmooth-specific arguments passed on the command line (for command-line programs) or passed within a short-cut (for window-based programs).

The following arguments all start with a -J and are recognized the jsmooth wrappers (note the case-sensitiveness):

- `-Jminversion=xxx`: Alter the minimum version of the JVM required by the application. This is handy to test the behaviour of the wrapper when no suitable JVM is found. Values are of the form 1.4.2 or just 1.3 (for instance).
- `-Jmaxversion=xxx`: Alter the maximum version of the JVM required by the application. Values are of the form 1.4.2 or just 1.3 (for instance).
- `-Jmainclassname=fqdn.classname`: Specify a classname that should be used by the wrapper as the main class for the application.
- `-Jcurrentdir=DIR`: Specifies the current dir that the wrapper must set up for the application.
- `-Jmaxheap=SIZE`: The maximum heap size that the VM can allocate. The size can be specified in bytes, in kilobytes (suffix k), or in megabytes (suffix m). For instance `-Jmaxheap=256M` (256 MB of heap space can be allocated) or `-Jmaxheap=1200k` (maximum 1200 kilobytes can be allocated).
- `-Jinitialheap=SIZE`: The initial memory heap size that the VM allocates at startup. The size can be specified in bytes, in kilobytes (suffix k), or in megabytes (suffix m). For instance `-Jinitialheap=256M` (256 MB of heap space are allocated at startup) or `-Jinitialheap=1200k` (1200 kilobytes are allocated).

Wrapper-specific options can also be passed in order to override the defaults parameters set up at wrapper creation. Those options start with `-Jskel_` (names are case-sensitive as well).

- `-JskelDebug=0|1`: Specifies whether the wrapper shall start in debug mode or not. In debug mode, it outputs logs in a console (in the standard output for console applications, or in a shell console otherwise). This is usually useful to debug why a wrapper couldn't launch a VM. 0 to disable the debug mode, 1 to enable it.
- `-Jskel_Message=aString`: The message to display to the user when no JVM is found. This is a short sentence that proposes the user to go on downloading a JRE (or going to an URL, depending on the wrapper), or just cancel the action.
- `-Jskel_URL=http://myurl/wherever`: For the Windowed wrapper only, this is the URL where the user is redirected if no JRE is found and if they accept the action.
- `-Jskel_DownloadURL=http://myurl/wherever`: For the Autodownload and Custom Web Download wrappers only, this is the download URL that the wrapper has to download in order to install a JRE.

7.1.6. I have a filetype associated to my program. However, when I launch a document, my application can't find its resources, because the current directory is the directory of the document. How do I fix that ?

You can force the wrappers to always force the current directory of your application to be the executable's instead of the document's by enabling the "Sets the executable folder as current directory of the application" option (located in the executable settings panel).

This works correctly because Windows always provides the document as an absolute path argument of the application.

7.1.7. How do I convert my java application into a Windows service ?

There is nothing much to do to make a service from a java application. Just select the "Windows Service" wrapper, and build your executable as you would do for a standard application.

The behaviour of the service wrapper is very simple:

- First, you have to register the service in the Windows environement. Just run `your-app.exe install`. Once this is done, you can see the service added to the list of services available on the computer (look for a Services program in the Administration tools of Windows).
- Then you can start or stop your service, by calling the executable with the "start" or "stop" arguments (only the first argument of the command line is taken into account). If the "autostart" option is enabled, the service will be automatically when Windows is started.
- To uninstall the service, simply run the executable as `your-app.exe uninstall`, that's all.
- The JVM is started and your application launched when the the service starts. The JVM is stopped when the services stops (obviously). Note however that you can exit cleanly, as `System.exit(0)` is called by the wrapper right before exiting: just add a shutdown hook (`Runtime.addShutdownHook()`) and you're safe.

7.2. Trouble & issues with wrappers

7.2.1. The wrapped applications complain about a "MSVCRT.DLL not found" on Windows 95?

The MSVCRT.DLL file is required by the wrapped application but may be missing on some early Windows 95 computers that have never been upgraded. This file is often installed by third-party application, but it may happen that your users do not have it installed.

To solve this issue, either ask your user to upgrade to a decent OS, or just bundle this MSVCRT.DLL along with your application.

7.2.1. Something goes wrong at launch-time, and the wrapper won't start my java application. How can I investigate?

The first thing to do, is to run the wrapper with the additional argument `-JskelDebug=1` (the option is case-sensitive). This will start the wrapper in debug mode, displaying a console (or using the current one, for the console wrapper), and output many useful information.

8. License

The JSmooth generator is distributed under the terms of the GNU General Public Licence. Please read the License.txt file that comes with the package. Alternatively, you can find additional information on the GNU GPL license on the GNU Web Site [http://www.gnu.org]

The generated executables are distributed under the terms of the GNU Library General Public Licence and as such do not constrain the licence of the Jar it may embed.

9. Third-Party libraries used

This product includes software developed by L2FProd.com (http://www.L2FProd.com/). The l2fprod-common library is licensed under the terms of Apache Software License. Source code can be found at the following location: https://l2fprod-common.dev.java.net/

JSmooth includes the JOX library. JOX is a set of Java libraries that make it easy to transfer data between XML documents and Java beans. It is distributed under the terms of the Lesser GPL. Source code for JOX is available at the library's web site: http://www.wutka.com/jox.html.

JSmooth includes the DTDParser library. It is distributed under the terms of either an Apache-style license or the Lesser GPL license. Source code for is available at the library's web site: http://www.wutka.com/dtdparser.html.

The autodownload skeleton uses the libmspack library (http://www.kyz.uklinux.net/libmspack/). The licence is not exactly the LGPL but is OK with jsmooth usage.

The autodownload skeleton uses the fltk library (http://www.fltk.org/). FLTK is provided under the terms of the GNU Library Public License, Version 2 with exceptions that allow for static linking.