ej Technologies

exe4j Manual

© 2014 ej-technologies GmbH. All rights reserved.

Index

exe4j help 3

Licensing 4

1 exe4j wizard 5
1.1 Overview 5
1.2 Step 1: Welcome 6
1.3 Step 2: Project Type 6
1.4 Step 3: Application Info 6
1.5 Step 4: Executable Info 7
1.6 Step 5: Java Invocation 8
1.7 Step 6: JRE 10
1.8 Step 7: Splash Screen 11
1.9 Step 8: Messages 11
1.10 Step 9: Compile Executable 11
1.11 Step 10: Finish 12
1.12 Advanced options 12
1.12.1 Redirection 12
1.12.2 Service options 12
1.12.3 Version info 13
1.12.4 32-bit or 64-bit 14
1.12.5 Execution level 14
1.12.6 Native libraries 15
1.12.7 Search sequence 15
1.12.8 Preferred VM 16
1.12.9 Splash screen options 16
1.13 Dialogs 18
1.13.1 Main class selection dialog 18
1.13.2 Class path dialog 18
1.13.3 Search sequence dialog 19
1.13.4 Visual positioning 19

2 exe4j launcher API 21
2.1 Controlling the splash screen 21
2.2 Receiving startup notifications 21

3 exe4j command line compiler 22
3.1 Overview 22
3.2 Options 22
3.3 Using exe4J with ant 23
3.4 Relative resource paths 24

Welcome To Exe4j!

exe4j creates Windows executables that invoke your Java applications and integrates them into the Windows environment.

The use of exe4j s use is not time limited, but restricted to evaluation purposes. Evaluation warnings are removed after purchasing an exe4j license [p. 4] .

Quick start

- To get an overview of exe4j's features, please have a look at the sample projects in the *demo* directory of your exe4j installation.
- When starting exe4j, a wizard [p. 5] will guide you step by step through collecting the necessary information to create your executable.
- A command line compiler [p. 22] is available to facilitate the inclusion of exe4j into an automated build process like ant [p. 23] .

Exe4j Licensing

Without a valid license, exe4j may be used for evaluation purposes only. The evaluation period is not time limited, but excludes any use for creating distributions of commercial products.

exe4j licenses can be purchased easily and securely online. We accept a large variety of payment methods including credit cards, checks and purchase orders. Pricing information is available online.

You can enter your license key in the welcome step [p. 6] of the exe4j wizard. If a license has been entered, the licensing information is visible there. The exe4j command line compiler [p. 22] also prints licensing information except when invoked with the quiet option [p. 22] .

Please read the included file *`license.txt`* to learn more about the scope of the license. For licensing questions, please contact sales@ej-technologies.com.

License key dialog:

Together with your license key, you are asked for your name and - if applicable - for the name of your company.

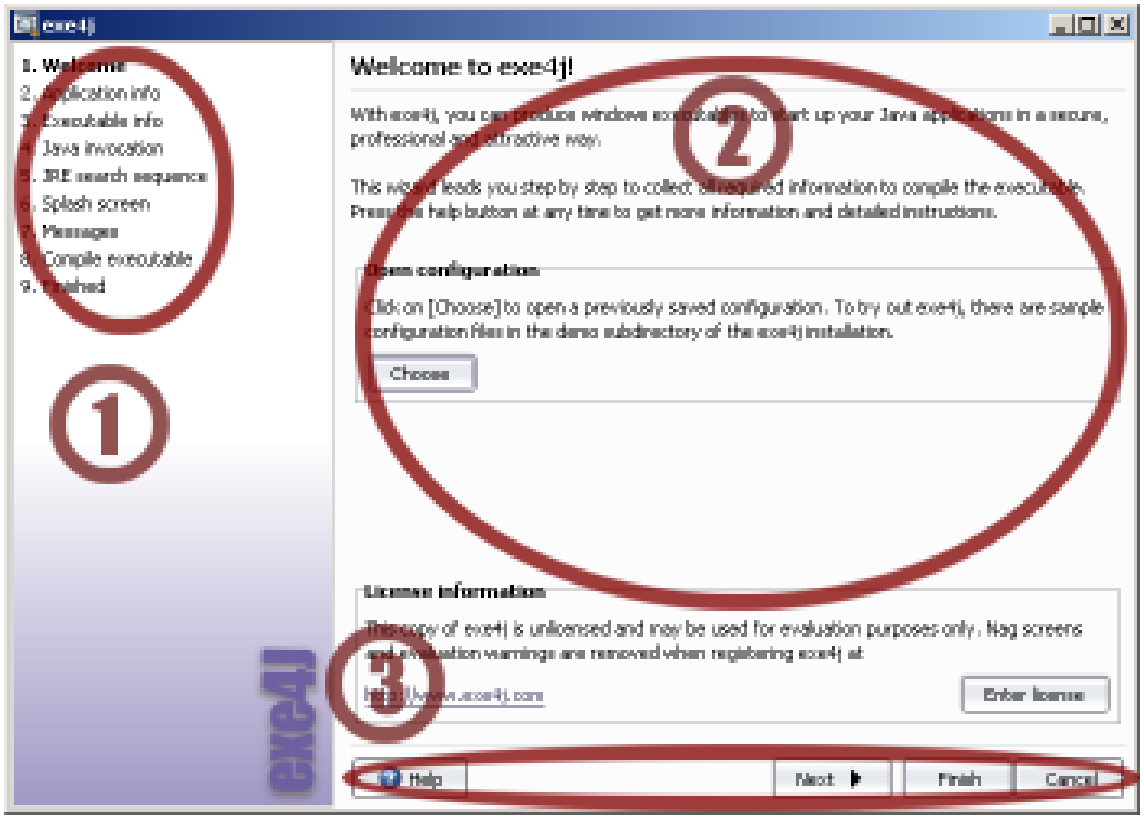
To make it easier for you to enter the license key, you can use the **[Paste from clipboard]** button, after copying any text fragment which contains the license key to your system clipboard. If a valid license key can be found in the clipboard content, it is extracted and displayed in the dialog.

1 Exe4j Wizard

1.1 Exe4j Wizard - Overview

When invoking exe4j from the start menu, the desktop icon or by executing *`bin\exe4j.exe`* in the exe4j installation directory, the exe4j wizard is started. It guides you step by step through completing the required information for building the executable.

The window of the wizard has three distinct areas:



- **(1) Index**

  The index of steps lists all steps of the wizard and highlights the current step in bold face. You can click on any step in the index to arbitrarily move between wizard steps. Alternatively, you can use the navigation controls (see below).

- **(2) Current step**

  The information for the current step is entered here. See the help pages for each step [p. 5] for specific information.

- **(3) Navigation**

  The navigation bar allows you to move back and forward through the steps of the wizard with the **[Next]** and **[Previous]** buttons. The **[Finish]** button allows you to complete the wizard immediately without moving through the remaining steps. Should any required information be missing, the wizard will alert you to it. The **[Help]** button shows context specific help that is also available by pressing **F1**. With the **[Cancel]** button you can leave the wizard at any time. Should you have made modifications to the configuration, you will be asked whether you want to save your changes before the wizard exits.

By default, the wizard starts with a template config file that contains default values where appropriate. The template config file is loaded from *config\template.exe4j* in the exe4j installation directory.

If you would like to load a config file at startup, you can pass the path of the desired config file to the wizard executable (*bin\exe4j.exe* in the exe4j installation directory).

1.2 Exe4j Wizard - Step 1: Welcome

In this step of the exe4j wizard [p. 5] , you can open previously saved configurations and enter your exe4j license information.

To **load an existing configuration**, click on the **[Open]** button in the `Open configuration` section. Locate the configuration file (extension **.exe4j*) in the file chooser and press **[OK]**. The path of the loaded configuration will be displayed next to the **[Open]** button.

exe4j remembers the last 10 previously loaded or saved configurations. To quickly reopen one of these configurations, click on the **[Reopen]** button and choose a configuration path from the popup menu that appears below the button. You can clear the contents of the popup menu by choosing the "Clear list" entry in the popup menu. If no configuration has been opened or saved with this copy of exe4j, or if the list has been cleared, the **[Reopen]** button is disabled.

To try out exe4j, the *$EXE4J_HOME\demo* directory contains three sample applications:

- a GUI application in the *gui* directory
- a command line application in the *cli* directory
- a service application in the *service* directory

License information is displayed at the bottom and can be entered or changed [p. 4] by clicking the **[Enter license]** or **[Change license]** buttons.

1.3 Exe4j Wizard - Step 2: Project Type

In this step of the exe4j wizard [p. 5] , you define the project type that determines whether exe4j compiles JAR files into the executable or not.

In the **regular mode**, exe4j is a pure launcher and relies on all JAR files and resources to be present in the distribution. In other words, the exe4j executable is an **addition to your distribution**, and not a replacement for it.

In the **JAR in EXE mode**, exe4j includes the JAR files specified in the class path configuration of the java invocation step [p. 8] into the executable. In this way, you can **distribute your application as a single executable** - provided it does not need additional support files and directories. In this mode, you can only select "archive" and "environment variable" in the classpath entry dialog [p. 18] . The JAR files are extracted to a temporary directory at runtime and deleted after use, the incurred overhead is minimal and usually not noticeable.

1.4 Exe4j Wizard - Step 3: Configure Application Info

In this step of the exe4j wizard [p. 5] , you enter the name and the most important directories of your application.

Enter the **name of your application** in the `General` section of this step. This name will be used in the error messages of the executable.

Depending on the project type, the following information is required:

- **Regular Mode**

Enter the **distribution source directory** in the `Directories` section of this step. The distribution source directory is the topmost directory under which all other directories of your application reside. Many directories and file paths that are referenced by the executable need to be known as relative rather than absolute paths. If you select such directories and files in later steps by means of a file chooser, the paths will be converted to paths **relative to the distribution source directory**.

Enter the **executable directory** in the `Directories` section of this step. The executable directory is the directory below the distribution source directory where the executable is to be placed. You can leave the output directory empty (or enter a `"."`) to make it equal to the distribution source directory. If you choose a directory with the file chooser (by clicking the **[...]** button), the directory will be converted into a path relative to the distribution source directory.

- **Jar-in-Exe Mode**

  Enter the **output directory** to determine where exe4j should place the generated application.

1.5 Exe4j Wizard - Step 4: Configure Executable

In this step of the exe4j wizard [p. 5], you enter the properties of the executable that is to be generated.

The following properties of the executable can be edited in the `Executable` section of this step:

- **Executable type**

  Executables created by exe4j can be either GUI or console applications:

  - **GUI application**

    There is no terminal window associated with a GUI application. If stdout and stderr are not redirected (see the redirection advanced step [p. 12]), both streams are inaccessible for the user. This corresponds to the behavior of *`javaw.exe`*.

    If you launch the executable from a console window, a GUI application can neither write to or read from that console window. Sometimes it might be useful to use the console, for example for seeing debug output or for simulating a console mode with the same executable. In this case you can select the **Allow -console parameter** check box. If the user supplies the -console parameter when starting the launcher from a console window, the launcher will try to acquire the console window and redirect stdout and stderr to it. If you redirect stderr and stdout in the redirection settings [p. 12], that output will not be written to the console.

  - **Console application**

    A console application has an associated terminal window. If a console application is opened from the Windows explorer, a new terminal window is opened. If stdout and stderr are not redirected (see the redirection advanced step [p. 12]), both streams are printed on the terminal window. This corresponds to the behavior of *`java.exe`*.

  - **Service**

    A Windows service runs independently of logged in users and can be run even if no user is logged on. The `main` method will be called when the service is started.

    To handle the shutdown of the service, you can use the `Runtime.addShutdownHook()` method to register a thread that will be executed before the JVM is terminated.

    For information on how services are installed or uninstalled, please see the help on service start options [p. 12].

- **Executable name**

  Enter the desired name of the executable without the trailing `.exe`.

- **Icon file**

  If you would like a custom icon to be compiled into your executable, select the `Icon file` checkbox and choose an icon file (extension *`*.ico`*).

- **Working directory**

  For some applications (especially GUI applications) you might want to change the working directory to a specific directory relative to the executable, for example to read config files that are in a fixed location. To do so, please select the `Change working directory to:` checkbox and enter a directory relative to the executable in the adjacent text field. To change the current directory to the same directory where the executable is located, please enter a single dot.

- **Allow only a single running instance of the application**

  If you select this checkbox, the generated exe4j executable can only be started once. Subsequent user invocations will bring the application to the front. In the `Controller` class of the exe4j launcher API you can register a startup handler to receive the command line parameters. In this way, you can handle file associations with a single application instance.

- **Fail if an exception in the main thread is thrown**

  Executables created by exe4j can monitor whether the main method throws an exception and show an error dialog in that case. This provides a generic startup error notification facility for the developer that handles a range of errors that would otherwise not be notified correctly. For example, if an uncaught exception is thrown during application startup, a GUI application might simply hang, leaving the user in the dark about the reasons for the malfunction. With the error message provided by the exe4j executable, reasons for startup errors are found much more easily.

1.6 Exe4j Wizard - Step 5: Configure Java Invocation

In this step of the exe4j wizard [p. 5], you enter the information required to start your application.

The following properties of the java invocation can be edited in the `General` section of this step:

- **Main class**

  Enter the fully qualified main class of your application. Next to the text field is a **[...]** chooser button that brings up a dialog with a list of all public main classes [p. 18] in the class path. To use this facility, you have to set up your classpath first (see below).

- **VM parameters**

  If there are any VM parameters you would like to specify for the invocation of your Java application, you can enter them here (e.g. `-Dmyapp.myproperty=true` or `-Xmx256m`).

  There are several runtime-variables you can use to specify runtime directories:

  - **%EXE4J_EXEDIR%**

    This is the directory where the executable is located.

  - **%EXE4J_JVM_HOME%**

    This is the directory of the JRE that your executable is running with.

  - **%EXE4J_TEMPDIR%**

    For the "JAR in EXE" mode [p. 6], this variable will contain the location of the temporary directory for the JAR files. In "regular mode" [p. 6] this variable is not used.

  These variables can be especially useful for adding JAR files to the bootclasspath.

  In addition to these VM parameters, a parameter file in the same directory as the executable is read and its contents are added to the existing VM parameters. The name of this parameter file is

the same as the exe file with the extension *.vmoptions. For example, if your exe file is named hello.exe, the name of the VM parameter file is hello.vmoptions. In this file, each line is interpreted as a single VM parameter. For example, the contents of the VM parameter file could be:

```
-Xmx128m
-Xms32m
```

It is possible to include other .vmoptions files from a .vmoptions file with the syntax

```
-include-options [path to other .vmoptions file]
```

You can use multiple includes in a single file, recursive includes are also supported. You can also add this option to the fixed VM parameters. In that way, you can prevent having to use the .vmoptions file right next to the executable.

This allows you to to centralize the user-editable VM options for multiple launchers and to have .vmoptions files in a location that can be edited by the user if the installation directory is not writable. You can use environment variables to find a suitable directory, for example

```
-include-options ${APPDATA}\My Application\my.vmoptions
```

or

```
-include-options ${USERPROFILE}\.myapp\my.vmoptions
```

In addition to the VM parameters you can also modify the classpath in the .vmoptions files with the following options:

- **-classpath [classpath]**

  Replace the classpath of the generated launcher.
- **-classpath/a [classpath]**

  Append to the classpath of the generated launcher.
- **-classpath/p [classpath]**

  Prepend to the classpath of the generated launcher.

You can use environment variables in the VM parameters with the following syntax: ${VARIABLE_NAME} where you replace VARIABLE_NAME with the desired environment variable.

- **arguments**

  If you need to specify arguments for your main class, you can enter them here. Arguments passed to the executable will be appended to these arguments.
- **Allow VM passthrough parameters**

  If you would like to allow the user to specify VM parameters with the syntax `-J[VM parameter]` (e.g. `-J-Xmx512m`), select the `Allow VM passthrough parameters` checkbox.

In the `Class path` section of this step you can configure the class path and the error handling for missing class path entries. The class path list shows all class path entries that have been added so far. The following types of class path entries [p. 18] are available:

- Scan directory
- Directory
- Archive
- Environment variable

The symbol prepended to an entry indicates that an error with that entry will lead to a startup failure with an error message displayed to the user. The control buttons on the right allow you to modify the contents of the class path list:

- Add class path entry (key `INS`)

  Invokes the class path entry dialog [p. 18] . Upon closing the class path entry dialog with the **[OK]** button, a new class path entry will be appended to the bottom of the class path list.

- Remove class path entry (key `DEL`)

  Removes the currently selected class path entry without further confirmation.

- Move class path entry up (key `ALT-UP`)

  Moves the selected class path entry up one position in the class path list.

- Move class path entry down (key `ALT-DOWN`)

  Moves the selected class path entry down one position in the class path list.

To change the error handling mode of a class path entry [p. 18] , select the class path entry and press **[Toggle 'fail on error']** right below the class path list or choose the corresponding menu item from the context menu.

1.7 Exe4j Wizard - Step 6: Configure JRE

In this step of the exe4j wizard [p. 5] , you enter the version requirements for the JRE or JDK that your application will be started with on the target system.

The **minimum Java version** must be specified in the `Java version` section of this step.

The **maximum Java version** can be specified in the `Java version` section of this step. If it is left empty, any JRE or JDK with a higher version than the minimum version is acceptable.

The maximum Java version can be entered with less numeric detail than the minimum Java version to prevent the use of a higher major or minor release. For example, a minimum version of 1.4.1 and a maximum version of 1.4 ensures that the highest available 1.4.x >= 1.4.1 JRE is used, but not a 1.5 JRE. Similarly, a minimum version of 1.4.1_03 and a maximum version of 1.4.1 ensures that the highest available 1.4.1 >= 1.4.1_03 JRE is used, but not a 1.4.2 JRE.

In addition to the version requirements, the following constraints are available:

- **allow JREs with a beta version number**

  By default, JREs with a beta version number or JREs from an early access release cycle will not be used by the launcher. If you would like to enable the use of these JREs, please check this option.

- **only allow JDKs and no JREs**

  Only JDKs will be used for launching your application, JREs will not be considered in the search sequence.

For more detailed requirements on which JRE should be used (e.g. for bundling your own JRE in the distribution), please see the search sequence advanced options step [p. 15] and the preferred VM advanced options step [p. 16] .

1.8 Exe4j Wizard - Step 7: Configure Splash Screen

In this step of the exe4j wizard [p. 5] , you can configure a splash screen for your application.

exe4j offers three different splash screen options

- **No splash screen**
- **exe4j splash screen**

  This is a native splash screen whose implementation is provided by exe4j. This splash screen works with all supported JVMs. Further configuration settings are available on the Advanced splash screen options tab [p. 16] .
- **Java 6+ splash screen**

  From Java 6 on, Java provides a splash screen implementation. If you would like to use this option, your project must at least specify a Java minimum version of 1.6 on the JRE step [p. 10] .

  Please note that you cannot specify the `-splash:` VM parameter in the "Java Invocation" step [p. 8] , since this VM parameter is parsed by the default Java launchers (*`java.exe`* and *`javaw.exe`*) which are not used by exe4j.

If you decide to display a splash screen, you have to enter an image file. The file format can be PNG, GIF or Windows BMP.

1.9 Exe4j Wizard - Step 8: Configure Messages Of The Executable

In this step of the exe4j wizard [p. 5] , you can configure the messages of the generated exe4j executable.

Default message sets for the exe4j executable are available in several languages. The language selector combo box at the top changes the contents of the message table below. Any customized messages are lost upon changing the language. A confirmation dialog is shown when this is the case.

The message table has 4 columns:

- **Customization indicator ("!")**

  If a message has been customized by you, the customization indicator column will display the symbol .
- **Message ID**

  The verbose ID of the message.
- **Message text**

  The text of the message that is displayed by the executable. Hitting the ENTER key or double clicking on this column allows you to edit this text in place.
- **Reset column**

  If a message has been customized, the last column displays a **[Reset]** button. Clicking on that button will change the message text back to the default text.

The context menu of the message table allows you to edit the selected message, reset it to the default text (if applicable) or reset all messages to the default text (if applicable).

1.10 Exe4j Wizard - Step 9: Compile Executable

In this step of the exe4j wizard [p. 5] , the executable is compiled.

The progress bar indicates the status of the compilation process. Hitting the **[Cancel]** button, stops the compilation process and moves back to the "configure messages" step [p. 11] . Should a compilation error occur, you will be notified with a message box and the "Finished" step [p. 12] will be displayed.

1.11 Exe4j Wizard - Step 10: Finish

This step of the exe4j wizard [p. 5] is displayed once the executable has been compiled and lets you save your configuration. You can start the compiled executable by pressing **[Click here to start the application]**. For console applications, a persistent terminal will be opened.

If the configuration was previously loaded and modified, the **[Save]** button will be enabled and saves back your changes to disk. The **[Save as]** button allows you to save your configuration to a different file.

The **[Exit]** button at the bottom of the dialog quits exe4j. The **[Restart]** button unloads the current configuration and restarts with the first step. In both cases, if your configuration is unsaved, you will be notified and given the opportunity to do so.

1.12 Advanced Options

1.12.1 Exe4j Wizard - Redirection

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 7] and choosing "Redirection" from the **[Advanced options]** popup menu or by clicking directly on the index.

In this step of the exe4j wizard [p. 5] , you can configure the redirection settings for stderr and stdout.

The following redirection settings can be edited:

- **Redirection of stderr**

  To redirect stderr to a file, select the `Redirect stderr` checkbox and enter a file name in the adjacent text field.
- **Redirection of stdout**

  To redirect stdout to a file, select the `Redirect stdout` checkbox and enter a file name in the adjacent text field.

File name are interpreted relative to the executable. Enter `/dev/null` if you want to suppress output completely. You can choose whether the redirection file is overwritten each time the launcher is started or if output should be appended to an existing redirection file.

Note that redirection files are created lazily. This means that if nothing is written to the redirected stream, the file file will not be created or overwritten.

You can use environment variables in the redirection files with the following syntax: `${VARIABLE_NAME}` where you replace VARIABLE_NAME with the desired environment variable.

1.12.2 Exe4j Wizard - Service Options

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 7] and choosing "Service options" from the **[Advanced options]** popup menu or by clicking directly on the index.

In this step of the exe4j wizard [p. 5] , you define further options for Windows service executables. All options on this screen will only be enabled if the selected executable type in the executable step [p. 7] is "Service".

Windows services are installed by passing `/install` to the generated service executable. The default start mode of the service can be determined in this step:

- **start on demand**

  In start on demand mode, your service must be manually started by the user in the Windows service manager. Use this option, if you're not sure if your users will actually want to run your application as a service, but you want to give them an easy way to do so. This installation mode can be forced if the user passes `/install-demand` to the generated executable instead of `/install`.

- **auto start**

  In auto start mode, your service is always started when Windows is booted. This installation mode can be forced if the user passes `/install-auto` to the generated executable instead of `/install`.

Windows services are always uninstalled by passing `/uninstall` to the generated service executable. All command line switches also work with a prefixed dash instead of a slash (like `-uninstall`) or two prefixed dashes (like `--uninstall`).

To start or stop the service, the `/start` and `/stop` options are available. In addition, a `/status` argument shows if the service is already running. The exit code of the status command is 0 when the service is running, 3 when it is not running and 1 when the state cannot be determined (for example when it is not installed).

As a second parameter after the `/install` parameter, you can optionally pass a **service name**. In that way you can

- install a service with a different service name than the default name.
- Use the same service executable to start multiple services with different names. To distinguish several running service instances at runtime, you can query the system property `exe4j.launchName` for the service name. Note that you also have to pass the same service name as the second parameter if you use the `/uninstall`, `/start` and `/stop` parameters.

In some situations, you might want to install the service as a non-interactive service meaning that the service will not have any possibility to access the GUI subsystem. In order to do that, add `non-interactive` after the `/install` parameter. A custom service name can still be specified after the `non-interactive` parameter.

If your service depends on another service, say a database, you can enter the service name of the other service in the `dependencies` text field. You do not have to enter core OS services such as filesystem or network, these services will always be initialized before your service is launched. If you have dependencies on multiple services, you can enter a list of these service names separated by commas.

1.12.3 Exe4j Wizard - Version Info

Note: this advanced option screen is reachable by selecting the "Executable" step [p. 7] and choosing "Version info" from the **[Advanced options]** popup menu or by clicking directly on the index.

In this step of the exe4j wizard [p. 5] , you can configure whether a version info resource should be generated in the executable and what values the version info fields should take.

A version info resource will enable the Windows operating system to determine meta information about your executable. This information is displayed in various locations. For example, when opening the property dialog for the executable in the Windows explorer, a "Version" tab will be present in the property dialog if you have chosen to generate the version info resource.

The version info resource consists of several pieces of information. If you check `Generate version info resource`, there are several fields whose values must be entered in the text fields on this step. Note that the "original file name" and the "product name" fields in the version info resource are filled in automatically by exe4j.

- **Product version**

  The product version must be composed of a maximum of 4 numbers, separated by spaces, commas or dots. By using the `-r` flag for the command line compiler [p. 22] , the product version can be overidden. The product version is also used in the splash screen version line config [p. 11] as a replacement value for the variable `%VERSION%`.

- **File version**

  If you want to specify a version for the file which is a different from the product version, you can do it here. If this field is left empty, the product version will be used for the file version.

- **Internal name**

  Choose a short internal name for identifying your application.

- **Company name**

  Enter the name of your company.

- **File description**

  Enter a description of the application.

- **Legal copyright**

  Enter a copyright statement for your application.

1.12.4 Exe4j Wizard - 32-bit Or 64-bit

In this step of the exe4j wizard [p. 5] , you can configure whether your executable should be a 32-bit executable or a 64-bit executable.

On Windows, a native executable can be either a 32-bit or a 64-bit executable. If you need a 64-bit JRE for your application you can choose to generate a 64-bit executable.

Note that it is not possible to create launchers that work with both 64-bit and 32-bit JREs. Since the launcher starts the JVM with the JNI interface by loading the JVM DLL, the architecture has to be the same. If you target both 32-bit and 64-bit JREs and operating systems, you have to generate different executables for them.

On a 64-bit Windows, there are separate system directories for 32-bit and 64-bit applications. If you enable the 64-bit executable mode in this step, those system directories will be appropriate for 64-bit applications, e.g. *`c:\Program Files`* instead of *`c:\Program Files (x86)`*.

1.12.5 Exe4j Wizard - Execution Level

In this step of the exe4j wizard [p. 5] , you can configure the execution level for your executable on Windows Vista and higher.

The execution level can be one of

- **As invoker**

  This is the default setting. The executable will be executed with the rights of the current token. If the user is an Administrator, this will be a filtered token so the executable will not have all administration rights.

- **Highest available**

  This level will raise the rights of the executable to the maximum extend available for the current user. This applies to Administrators that usually run with a filtered token. Windows Vista and higher will show a question to the user if he wants to elevate the rights of this application. For a standard user this is the same as "As invoker".

- **Require administrator**

This is the same as "Highest available" when the user is an Administrator running with a filtered token. If the user is a standard user, Windows Vista and higher will ask for the credentials of an Administrator account.

1.12.6 Exe4j Wizard - Native Library Directories

In this step of the exe4j wizard [p. 5] , you can configure directories that contain native libraries.

If your application uses native libraries that you would lke to load with a `System.loadLibrary()` call, the directory where the *`.dll`* is located must be included in the **PATH** environment variable. You can add such directories in the path list of this step.

- Add native library directory (key `INS`)

  Lets you add a new directory to the end of the list. Choose the native library directory in the file chooser that appears after clicking this button. The directory will be converted to a path relative to the distribution source directory.

- Remove native library directory (key `DEL`)

  Removes the currently selected native library directory entry without further confirmation.

- Move entry up (key `ALT-UP`)

  Moves the selected native library directory entry up one position in the path list.

- Move entry down (key `ALT-DOWN`)

  Moves the selected native library directory entry down one position in the path list.

1.12.7 Exe4j Wizard - Configure Search Sequence

Note: this advanced option screen is reachable by selecting the "JRE" step [p. 10] and choosing "Search Sequence" from the **[Advanced options]** popup menu or by clicking directly on the index.

In this step of the exe4j wizard [p. 5] , you can configure the way the exe4j executable looks for an appropriate JRE or JDK to start your Java application. The versions requirements for the JRE are specified in a different step [p. 10] .

The search sequence list shows all search sequence entries that have been added so far. For new configurations, a default search sequence is pre-defined. The following types of search sequence entries [p. 19] are available:

- Search registry
- Directory
- Environment variable

The control buttons on the right allow you to modify the contents of the search sequence list:

- Add search sequence entry (key `INS`)

  Invokes the search sequence entry dialog [p. 19] . Upon closing the search sequence entry dialog with the **[OK]** button, a new search sequence entry will be appended to the bottom of the search sequence list.

- Remove search sequence entry (key `DEL`)

Removes the currently selected search sequence entry without further confirmation.

- Move search sequence entry up (key `ALT-UP`)

  Moves the selected search sequence entry up one position in the class path list.

- Move search sequence entry down (key `ALT-DOWN`)

  Moves the selected search sequence entry down one position in the class path list.

It is possible to generate a log file that contains information about the JRE search sequence and any potential problems. In order to switch on logging, please define the environment variable `EXE4J_LOG=yes` and look for the newest text file whose name starts with `i4j_nlog_` in the Windows temp directory. This information can be used for debugging purposes. If you have a problem with JRE detection, please send this log file along with your support request.

An easier way for a user to create a log file is to start the launcher with the `/create-i4j-log` argument. The launcher will notify the user where the log is created and will offer to open an explorer window with the log file selected. After the message box, the launcher will continue to start up.

Tip: To distribute your own JRE, simply put the JRE in your distribution and define a directory search sequence entry with the appropriate relative path (e.g. *jre*) as the first item.

If the entire search sequence fails, exe4j will try the location defined by the environment variable `EXE4J_JAVA_HOME`. If that fails too, an error message will be displayed asking the user to define this variable. To supply a custom variable, define an appropriate environment variable search sequence entry and customize the corresponding error message in the messages step [p. 11] of the wizard.

1.12.8 Exe4j Wizard - Choose Preferred VM

Note: this advanced option screen is reachable by selecting the "JRE" step [p. 10] and choosing "Preferred VM" from the **[Advanced options]** popup menu or by clicking directly on the index.

In this step of the exe4j wizard [p. 5], you can configure the preferred VM that exe4j will choose to invoke your application. This setting only influences the choice of the VM type after a JRE has been selected according to the search sequence. The search sequence for the JRE is specified in a different step [p. 10].

After exe4j finds a suitable JRE or JDK, it tries to honor the setting you make in this step. You can select one of the following:

- **default VM**

  exe4j will use the default VM for the found JRE.

- **client hotspot VM**

  exe4j will try to use the client hotspot VM for the found JRE. This is equivalent to using the `-client` switch when invoking `java` from the command line.

- **server hotspot VM**

  exe4j will try to use the server hotspot VM for the found JRE. This is equivalent to using the `-server` switch when invoking `java` from the command line.

Please note that it is not an error if the selected JVM is not present for the found JRE. exe4j will simply use another JVM to launch your application in that case.

1.12.9 Exe4j Wizard - Splash Screen Options

Note: this advanced option screen is reachable by selecting the "Splash screen" step [p. 11] and choosing "Splash screen options" from the **[Advanced options]** popup menu or by clicking directly

on the index. The options in this step are only available if "exe4j splash screen" was chosen in the "Splash screen" step [p. 11] .

The behavior of the splash screen can be defined in the `General` section of this step:

- **Hide splash screen when first application window is shown**

  If this option is checked, the exe4j executable will monitor the state of your application and hide the native splash screen as soon as a window is opened. If you want to hide the splash screen programmatically, you can use exe4j's launcher API [p. 21] .
- **Splash screen is always on top**

  If this option is checked, the splash screen remains always on top of other windows opened by your application.

The `Status line` and `Version line` sections allow you to position the text lines on the splash screen and configure their font. The status line is dynamically updatable with exe4j's launcher API [p. 21] . If you include the variable `%VERSION%` in the version line text, it will be replaced with the product version defined in the version info step [p. 13] of the wizard. With the `-r` flag, you can override this setting for the command line compiler [p. 22] .

You can configure the following properties of a text line

- **Text**

  The (initial) text displayed in the text line.
- **Position**

  The x and y-coordinates of the text line on the splash screen. The origin of the coordinate system is the top left corner of the splash screen window.
- **Font**

  The font used for drawing the text line:

  - **Name**

    The name of the font. Please choose a common font name that is likely to be available on all target platforms. If unavailable at runtime, the `MS Dialog` font will be used as a fallback.
  - **Weight**

    The weight of the font. The 8 Windows standard font weights are offered as a choice.
  - **Size**

    The size of the font in points.
  - **Color**

    The color of the font. By clicking on **[...]**, a color chooser dialog is brought up.

To **visually position the text lines** with mouse and keyboard on the actual splash screen image, please click on the **[Position text lines visually]** button. The visual positioning dialog [p. 19] will then be displayed. On exiting the dialog with the **[OK]** button, the X/Y coordinate text fields (see above) will be updated for both text lines.

1.13 Dialogs

1.13.1 Main Class Selection Dialog

The main class selection dialog is shown when clicking on the **[...]** chooser button next to the main class text field in the Java invocation step [p. 8] .

Please choose a main class from the list and click on **[OK]** or double-click on the selected class. If you do not want to make a choice, please click on **[Cancel]**.

1.13.2 Classpath Entry Dialog

The class path entry dialog is shown when clicking on the add button in the "configure java invocation" step [p. 8] of the exe4j wizard. Upon closing this dialog with the **[OK]** button, a new class path entry will be appended to the bottom of the class path list of that step.

To define a class path entry, you first select the entry type, then check the `fail if an error occurs with this class path entry` checkbox in case you want the startup to be terminated if this class path entry is faulty and finally fill out the `Detail` section of the dialog which is dependent on the selected entry type. The following entry types are available:

- **Scan directory**

  Scan a directory for archives with the extensions *`*.jar`* and *`*.zip`* to be added to the class path. In the `Detail` section of the dialog you must choose a directory either by entering the path in the text field or by clicking **[...]** and choosing it with a file chooser.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this directory does not exist.

  Note: Not available if the project type [p. 6] is "JAR in EXE"

- **Directory**

  Add a directory to the class path. In the `Detail` section of the dialog you must choose a directory either by entering the path in the text field or by clicking **[...]** and choosing it with a file chooser.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this directory does not exist.

  Note: Not available if the project type [p. 6] is "JAR in EXE"

- **Archive**

  Add an archive with the extension *`*.jar`* or *`*.zip`* to the class path. In the `Detail` section of the dialog you must choose an archive either by entering the path in the text field or by clicking **[...]** and choosing it with a file chooser.

  The last path component can include a * as a placeholder for a frequently changing version number. This is not a wildcard for processing multiple matching paths, rather it is intended for systems like maven where the version number on dependencies is part of the file name and is frequently changed. An example is *`bin\commons-io-*.jar`* which will match a file like *`bin/commons-io-1.0.jar`* at compile time. This replacement is performed at compile-time and not a runtime.

  Error handling:

If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this archive does not exist.

- **% Environment variable**

  Add the contents of an environment variable to the class path. In the `Detail` section of the dialog you must enter the name of an environment variable.

  Error handling:

  If `fail if an error occurs with this class path entry` is checked, the application will terminate with an error message if this environment variable is not defined.

Except for the "Environment variable" classpath type, you can use environment variables in the text field with the following syntax: `${VARIABLE_NAME}` where you replace VARIABLE_NAME with the desired environment variable.

The directory of the JRE that your application is running with is contained in ${EXE4J_JVM_HOME}. If you've specified in the Configure JRE step [p. 10] , that only JDKs and no JREs should be used, you can append `\..` after the variable to change into the JDK home directory. For example, to reference `tools.jar`, you'd have to write `%EXE4J_JVM_HOME%\..\lib\tools.jar`.

Note that for path selections by means of a file chooser (**[...]** buttons), exe4j will try to convert the path to be relative to the distribution source directory.

1.13.3 Search Sequence Entry Dialog

The search sequence entry dialog is shown when clicking on the add button in the "configure search sequence" step [p. 15] of the exe4j wizard. Upon closing this dialog with the **[OK]** button, a new search sequence entry will be appended to the bottom of the search sequence list of that step.

To define a search sequence entry, you select the entry type and fill out the `Detail` section of the dialog which is dependent on the selected entry type. The following entry types are available:

- **Search registry**

  Search the Windows registry for installed JREs and JDKs by Sun Microsystems.

- **Directory**

  Look in the specified directory. This is especially useful if you distribute your own JRE along with your application. In that case, be sure to supply a relative path. Note that for path selections by means of the file chooser (**[...]** button), exe4j will try to convert the path to be relative to the distribution source directory.

- **% Environment variable**

  Look for a JRE of JDK in a location that is defined by an environment variable like `JAVA_HOME` or `MYAPP_JAVA_HOME`.

1.13.4 Visual Positioning Of Text Lines

The visual positioning dialog is shown when clicking on the **[Position text lines visually]** button in the "configure splash screen" step [p. 11] of the exe4j wizard. Upon closing this dialog with the **[OK]** button, the X/Y coordinate text fields will be updated for status and version text lines in that step.

The visual positioning dialog displays the selected image with overlaid status and text line placeholders that are surrounded on the left and bottom by lines. These lines flash for the selected text line. You

can position the selected text line on the image by dragging it with the mouse or using the cursor keys. Pressing CTRL with the cursor keys moves the text line in larger steps.

Please note that only the font color is reflected in the font of the text line placeholders. Font weight, font size and font name are only used in the native runtime version of the splash screen.

2 Exe4j Launcher API

2.1 Controlling The Splash Screen From Your Application

If you have enabled a splash screen [p. 11] for your exe4j executable, you usually want to hide it once the application startup is finished. With exe4j's auto-off mode [p. 11] enabled, the splash will be hidden automatically as soon as your application opens the first window.

However, you might want to hide the splash screen programmatically or update the contents of the status text line on the splash screen during the startup phase to provide more extensive feedback to your users.

With the exe4j launcher API you can

- **Hide the splash screen programatically**

  Invoke the static method `com.exe4j.Controller.hide()` as soon as you wish to hide the splash screen.
- **Update the status text line**

  Invoke the static method `com.exe4j.Controller.writeMessage(String message)` to change the text in the status line.

The launcher API of exe4j is contained in *`exe4jlib.jar`* which can be found in the top level directory of your exe4j installation.

Note: you do **not** have to add it to the classpath of your application and distribute it along with it, since that file is always contained in the executable.

2.2 Receiving Startup Events In Single Instance Mode

If you have enabled the single instance mode [p. 7] for your executable, the application can only be started once. For a GUI application, the existing application window is brought to front when a user executes the launcher another time.

However, you might want to receive notifications about multiple startups together with the command line parameters. If you have associated you executable with a file extension, you will likely want to handle multiple invocations in the same instance of your application. Alternatively, you might want to perform some action when another startup occurs.

With the exe4j launcher API you can write a class that implements the `com.exe4j.Controller.StartupListener` interface and register it with `com.exe4j.Controller.registerStartupListener(StartupListener startupListener)`. Your implementation of `startupPerformed(String parameters)` of the `StartupListener` interface will then be notified if another startup occurs.

The launcher API of exe4j is contained in *`exe4jlib.jar`* which can be found in the top level directory of your exe4j installation.

Note: you do **not** have to add it to the classpath of your application and distribute it along with it, since that file is always contained in the executable.

3 Exe4j Command Line Compiler

3.1 Exe4j Command Line Compiler

exe4j's command line compiler *exe4jc.exe* can be found in the *bin* directory of your exe4j installation. It operates on any config file with extension *.exe4j* that has been produced with the exe4j wizard. (*exe4j.exe*). The exe4j command line compiler is invoked as follows:

```
exe4jc [OPTIONS] [config file]
```

The available options are described here [p. 22] . A quick help for all options is printed to the terminal when invoking

```
exe4jc --help
```

A typical run of the exe4j command line compiler looks like this:

```
exe4j version 4.0, built on 2006-09-26
Unregistered evaluation version

Loading config file myapp.exe4j
Deleting temporary directory
Compiled executable for myapp in 0.8 seconds.
```

In order to facilitate the use of exe4j in automated build processes, the destination directory for the executable and the version text line of the splash screen [p. 11] can be overridden with command line options [p. 22] . Since the file format of exe4j's config files is xml-based, you can achieve arbitrary customizations by replacing tokens (see the ant integration help page [p. 23] for an example) or by applying XSLT stylesheets to the config file.

3.2 Options For The Exe4j Command Line Compiler

The exe4j command line compiler [p. 22] has the following options:

- **-h or --help**

  Displays a quick help for all available options.
- **-V or --version**

  Displays the version of exe4j in the following format:

  ```
  exe4j version 1.0, built on 2002-10-05
  ```
- **-v or --verbose**

  Enables verbose mode. In verbose mode, exe4j prints out information about internal processes. If you experience problems with exe4j, please make sure to include the verbose terminal output with your bug report.
- **-q or --quiet**

  Enables quiet mode. In quiet mode, no terminal output short of a fatal error will be printed.
- **-t or --test**

  Enables test mode. In test mode, no executable will be generated in the directory for the executable.
- **-L or --license=KEY**

  Update the license key on the command line. This is useful if you have installed exe4j on a headless system and cannot start the GUI. KEY must be replaced with your license key.
- **-x or --require-license**

  By default, exe4j will fallback to evaluation mode if the license key is not valid. If you want the compilation to fail instead, you can spcify this option.

- **-r STRING or --release=STRING**

  override the application version defined in the version info step [p. 13] . `STRING` must be replaced with the desired version number. The version number can only contain numbers and dots.

- **-d STRING or --destination=STRING**

  override the destination directory for the executable. `STRING` must be replaced with the desired directory. If the directory contains spaces, you must enclose STRING in quotation marks.

  Note that this option does not affect the interpretation of relative paths defined by the distribution source directory and the output directory as specified in the application step [p. 6] of the exe4j wizard.

3.3 Using Exe4j With Ant

Integrating exe4j with your ant script (read about ant at http://ant.apache.org) is easy. Just use the `exe4j` task that is provided in *`{exe4j installation directory}/bin/ant.jar`* and set the `projectfile` parameter to the exe4j config file that you want to build.

To make the `exe4j` task available to ant, you must first insert a `taskdef` element that tells ant where to find the task definition. Here is an example of using the task in an ant build file:

```
<taskdef name="exe4j"
         classname="com.exe4j.Exe4JTask"
         classpath="C:\Program Files\exe4j\bin\ant.jar"/>

<target name="launcher">
  <exe4j projectfile="myapp.exe4j"/>
</target>
```

The `taskdef` definition must occur only once per ant-build file and can appear anywhere on the top level below the `project` element.

Note: it is **not possible** to copy the *`ant.jar`* archive to the *`lib`* folder of your ant distribution. You have to reference a full installation of exe4j in the task definition.

The exe4j task supports the following parameters:

Attribute	Description	Required
projectfile	The exe4j config file for the launcher that should be generated.	Yes
verbose	Corresponds to the `--verbose` command line option [p. 22] . Either `true` or `false`.	No, verbose and quiet cannot **both** be `true`
quiet	Corresponds to the `--quiet` command line option [p. 22] . Either `true` or `false`.	
test	Corresponds to the `--test` command line option [p. 22] . Either `true` or `false`.	No
release	Corresponds to the `--release` command line option [p. 22] . Enter a version number like "`3.1.2`". The version number may only contain numbers and dots.	No
requirelicense	Corresponds to the `--require-license` command line option [p. 22] .	No

destination	Corresponds to the `--destination` command line option [p. 22] . Enter a directory where the generated launcher should be placed.	No

To customize aspects of the exe4j build that cannot be overridden with he above parameters, you can add appropriate tokens in the config file and use the `copy` task with a nested `filterset` element. For example, if the main class in

```
<java mainClass="com.mycorp.MyApp" ...
```

should by dynamically adjusted by ant, just edit the line to

```
<java mainClass="@MAIN_CLASS@" ...
```

and copy the template config file (here *`myapp_template.exe4j`*) with

```
<copy tofile="myapp.exe4j" file="myapp_template.exe4j">
  <filterset>
    <filter token="MAIN_CLASS" value="com.mycorp.MyOtherApp" />
  </filterset>
</copy>
```

before running the exe4j compiler as before.

3.4 Relative Resource Paths

If you would like to use relative paths for the distribution directory, the bitmap and icon files (e.g. for automated build processes in distributed environments) you can change these values manually in the config file.

If the mentioned paths are relative, they are interpreted relative to the location of the config file.