ej technologies

exe4j Manual

© 2026 ej-technologies GmbH. All rights reserved.

Index

Introduction 3

Configuration wizard 4

Service options 9

Runtime API 10

Command line compiler 11

Ant task 13

What Is Exe4j?

exe4j creates Windows executables that invoke your Java applications.

The use of exe4j is not time-limited, but restricted to evaluation purposes. Evaluation warnings are removed after purchasing a license [(1)]. You can enter a permanent license key in the "Welcome" step of the exe4j wizard.

How do I continue?

- To get an overview of exe4j's features, have a look at the sample projects in the `demo` directory of your exe4j installation.
- When starting exe4j, a wizard [p. 4] will guide you step by step through collecting the necessary information to create your executable.
- A command line compiler [p. 11] is available to facilitate the inclusion of exe4j into an automated build process like ant [p. 13].

[(1)] https://www.ej-technologies.com/redir?product=exe4j&target=order

Exe4j Wizard

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. You can click on step names in the wizard index to navigate quickly to a selected step.

By default, the wizard starts with an empty configuration, if you would like to load a config file at startup, you can pass the path of the desired config file on the command line.

To try out exe4j, the `demo` directory in the exe4j installation 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

Project types

An exe4j project can be compiled in one of two modes:

- **Regular mode**

  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.

  On the "Application info" step, you enter the **distribution source directory** . The distribution source directory is the topmost directory under which all other directories of your application reside. When you select directories and files in the wizard through a file chooser, the paths will be converted to paths relative to the distribution source directory. One of those relative directories is the **executable directory** in the "Application info" step is the directory below the distribution source directory where the executable is to be placed.

- **JAR in EXE mode**

  In "JAR in EXE" mode, exe4j includes the JAR files specified in the class path configuration of the "Java invocation" step 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. The JAR files are extracted to a temporary directory at runtime and deleted after use.

Executable types

Executables created by exe4j can be one of the following three types:

- **GUI application**

  There is no terminal window associated with a GUI application. If stdout and stderr are not redirected, 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 "Executable info->Redirection" step, 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, both streams are printed in 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, see the help on service start options [p. 9].

Executable options

In the "Executable info->32 bit or 64-bit" step of the wizard, you can configure whether your executable should be a 32-bit executable or a 64-bit executable.

Note that it is not possible to create launchers that work with both 64-bit and 32-bit JREs. Because 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, you have to generate different executables for them.

In other sub-steps of the "Executable info" step, you can optionally configure redirection of stdout and stderr, a version info resource for the executable and options for the executable manifest. On the "Executable info->Manifest options", a non-standard execution level can be configured as well as the DPI awareness of your process which is important for High-DPI screens.

If your application can deal with different DPI settings, you can tell exe4j to add the manifest entry to the executable that enables DPI-awareness. If that entry is not added, the GUI will be scaled up automatically and may look blurry.

VM parameters

In the "Java invocation" step of the wizard, you enter the information required to start your application, including a list of VM parameters. There are several runtime-variables you can use to specify runtime directories in the VM parameters:

- **%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, this variable will contain the location of the temporary directory for the JAR files. In "regular mode" this variable is not used.

exe4j can add specific VM parameters depending on the Java version. To set this up, click on the *Configure version specific VM parameters* button. In the dialog, add rows for each range of Java versions that should receive specific VM parameters. If the Java version of the JVM that is used at runtime matches a configured version expression, the associated VM parameters will be appended to the common VM parameters. The search is stopped at the first matching entry. The syntax for the Java version expressions is explained by the help icon on the table header.

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 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.

Java invocation

On the "Java invocation" step of the launcher wizard you can configure both the module path and the class path. These settings correspond to the `--module-path` and the `-cp` parameters of the standard Java launcher. The module path is only applicable for Java 9 and higher. Like for the standard Java launcher, you can add directories, single archives or directories with archives. In addition, you can add archives from environment variables and from compiler variables.

For Java 9 and higher, you can choose a main class from either the module or the class path. If you choose the module path option, the syntax for the main class is `<module name>/<class name>` and corresponds to the `--module` parameter of the standard Java launcher. The chooser dialog shows all the available main classes and inserts the correct value automatically.

Like VM parameters, the list of fixed arguments supports launcher variables. Arguments on the command line are appended to the fixed list of arguments.

The launcher needs a runtime JAR file which is automatically added to the executable and extracted to a temporary directory at runtime. If you use the regular mode and not the "JAR in EXE" mode, you can prevent the extraction of files at runtime by deselecting the "Bundle runtime" option. In that case, you have to distribute the runtime file `exe4jlib.jar` and add it to the classpath or module path.

JRE selection

In the "JRE" step of the wizard, 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, but the maximum Java version can be left empty, so that any JRE or JDK with a higher version than the minimum version is acceptable.

In the "JRE->Search sequence" step, you can configure the way the exe4j executable looks for an appropriate JRE or JDK to start your Java application.

The following types of search sequence entries are available:

- **Search registry**

  Search the Windows registry for installed JREs and JDKs by Oracle.

- **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 through 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`.

To distribute your own JRE, put the JRE in your distribution and define a directory search sequence entry with the appropriate relative path (for example `jre`) as the first item.

It is possible to generate a log file that contains information about the JRE search sequence and any potential problems. To switch on logging, start the executable 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. If it is not possible to pass arguments, 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.

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 of the wizard.

Splash screen

Splash screens for executables generated by exe4j cannot be configured with the `-splash` VM parameter, but must be configured on the "Splash screen" step.

In addition, you can overlay lines of text for status and version information on the splash screen, by configuring them on the "Splash screen->Text lines" step. 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. 9]. If you include the variable `%VERSION%` in the version line text, it will be replaced with the product version defined in the "Executable info->Version info" step of the wizard. With the `-r` flag, you can override this setting for the command line compiler [p. 11].

Messages

In the "Messages" step of the wizard you can configure all messages that may be displayed by the generated executable. Default message sets for the exe4j executable are available in several languages. You can double-click on any message to edit it. If a message is modified from its default, a customization indicator and a *Reset* button is displayed in the table row.

Services

On the "Executable info->Service options" step of the wizard you can configure further options for executables whose type has been set to "Service" on the "Executable info" step.

Windows services are installed by passing `/install` to the generated service executable. The default start mode of the service can be set as:

- **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`, `/stop` and `/restart` 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.

For debugging purposes, you may want to run the executable on the command line without starting it as a service. This can be done with the `/run` parameter. In that case, all output will be printed on the console. If you want to keep the redirection settings, use the `/run-redirect` parameter instead.

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.

Runtime API

Controlling the splash screen from your application

If you have enabled a splash screen [p. 4] for your exe4j executable, you usually want to hide it once the application startup is finished. 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.

Receiving startup events in single instance mode

If you have enabled the `Allow only a single running instance of the application` checkbox on the "Executable info->Single instance mode" step, 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.

The scope of the single instance check can be per-user or global across all users. For the per-user scope, the "Per session" setting controls whether multiple RDP sessions for the same user can support one instance per session or only one instance across all sessions.

In single instance mode, you may want to receive notifications about multiple startups together with the command line parameters. If you have associated your 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.

Startup notifications only work when the same user starts the executable again. With the global scope, a startup of a different user will not produce a startup notification.

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.

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]
```

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 X.Y, built on 20YY-MM-DD
Unregistered evaluation version

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

Command line compiler options

The exe4j command line compiler [p. 11] 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.

- **-w or --fail-on-warning**

  If a warning is printed and this option is specified, the build will fail at the end. It does not fail immediately, so you can see all warnings and fix them all at once. The exit code in this case is 2 instead of 1 for an actual error and 0 for a successful execution.

- **-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 fall back to evaluation mode if the license key is not valid. If you want the compilation to fail instead, you can specify this option.

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

  override the application version defined in the "Executable info->Version info" step. 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 info" step of the exe4j wizard.

Overriding settings at build time

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 can be overridden with command-line options. Because the file format of exe4j's config files is in XML format, you can achieve arbitrary customizations by replacing tokens [p. 13] or by applying XSLT stylesheets to the config file.

Relative resource paths

If you would like to use relative paths for the distribution directory, the bitmap and icon files (for example, 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.

Using Exe4j With Ant

For integrating exe4j with your Ant script [1], 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 that 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.

Task properties

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. Either `true` or `false`.	No, verbose and quiet cannot **both** be `true`
quiet	Corresponds to the `--quiet` command-line option. Either `true` or `false`.	
failOnWarning	Corresponds to the `--fail-on-warning` command-line option. Either `true` or `false`.	
test	Corresponds to the `--test` command-line option. Either `true` or `false`.	No
release	Corresponds to the `--release` command-line option. 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.	No

[1] https://ant.apache.org

Attribute	Description	Required
license	Corresponds to the `--license` command-line option. If the license has not been configured yet, you can set the license key with this attribute.	No
destination	Corresponds to the `--destination` command-line option. Enter a directory where the generated launcher should be placed.	No

Modifying project files at build time

To customize aspects of the exe4j build that cannot be overridden with the 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, 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.