



Linux Ubuntu 26.04 Manual Pages on command 'jpackage.1'

\$ man jpackage.1

JPACKAGE(1)

JDK Commands

JPACKAGE(1)

NAME

jpackage - tool for packaging self-contained Java applications.

SYNOPSIS

jpackage [options]

options

Command-line options separated by spaces. See jpackage Options.

DESCRIPTION

The jpackage tool will take as input a Java application and a Java run-time image, and produce a Java application image that includes all the necessary dependencies. It will be able to produce a native package in a platform-specific format, such as an exe on Windows or a dmg on macOS. Each format must be built on the platform it runs on, there is no cross-platform support. The tool will have options that allow packaged applications to be customized in various ways.

JPACKAGE OPTIONS

Generic Options:

@filename

Read options from a file.

This option can be used multiple times.

--type or -t type

The type of package to create

Valid values are: {"app-image", "exe", "msi", "rpm", "deb", "pkg", "dmg"}

If this option is not specified a platform dependent default type will be created.

--app-version version

Version of the application and/or package

--copyright copyright

Copyright for the application

--description description

Description of the application

--help or -h

Print the usage text with a list and description of each valid option for the current platform to the output stream, and exit.

--icon path

Path of the icon of the application package

(absolute path or relative to the current directory)

--name or -n name

Name of the application and/or package

--dest or -d destination

Path where generated output file is placed

(absolute path or relative to the current directory).

Defaults to the current working directory.

--resource-dir path

Path to override jpackage resources

(absolute path or relative to the current directory)

Icons, template files, and other resources of jpackage can be over-ridden by adding replacement resources to this directory.

--temp directory

Path of a new or empty directory used to create temporary files

(absolute path or relative to the current directory)

If specified, the temp dir will not be removed upon the task completion and must be removed manually.

If not specified, a temporary directory will be created and removed upon the task completion.

--vendor vendor

Vendor of the application

--verbose

Enables verbose output.

`--version`

Print the product version to the output stream and exit.

Options for creating the runtime image:

`--add-modules module-name [,module-name...]`

A comma (",") separated list of modules to add

This module list, along with the main module (if specified) will be passed to jlink as the `--add-module` argument. If not specified, either just the main module (if `--module` is specified), or the default set of modules (if `--main-jar` is specified) are used.

This option can be used multiple times.

`--module-path` or `-p module-path [,module-path...]`

A `File.pathSeparator` separated list of paths

Each path is either a directory of modules or the path to a modular jar, and is absolute or relative to the current directory.

This option can be used multiple times.

`--jlink-options options`

A space separated list of options to pass to jlink

If not specified, defaults to "`--strip-native-commands --strip-debug --no-man-pages --no-header-files`"

This option can be used multiple times.

`--runtime-image directory`

Path of the predefined runtime image that will be copied into the application image (absolute path or relative to the current directory)

If `--runtime-image` is not specified, jpackage will run jlink to create the runtime image using options specified by `--jlink-options`.

Options for creating the application image:

`--input` or `-i directory`

Path of the input directory that contains the files to be packaged (absolute path or relative to the current directory)

All files in the input directory will be packaged into the application image.

`--app-content additional-content [,additional-content...]`

A comma separated list of paths to files and/or directories to add to the application

payload.

This option can be used more than once.

Options for creating the application launcher(s):

--add-launcher name=path

Name of launcher, and a path to a Properties file that contains a list of key, value pairs

(absolute path or relative to the current directory)

The keys "module", "main-jar", "main-class", "description", "arguments", "java-options", "icon", "launcher-as-service", "win-console", "win-shortcut", "win-menu", and "linux-shortcut" can be used.

These options are added to, or used to overwrite, the original command line options to build an additional alternative launcher. The main application launcher will be built from the command line options. Additional alternative launchers can be built using this option, and this option can be used multiple times to build multiple additional launchers.

--arguments arguments

Command line arguments to pass to the main class if no command line arguments are given to the launcher

This option can be used multiple times.

Value can contain substrings that will be expanded at runtime. Two types of such substrings are supported: environment variables and "APPDIR", "BINDIR", and "ROOTDIR" tokens.

An expandable substring should be enclosed between the dollar sign character (\$) and the first following non-alphanumeric character. Alternatively, it can be enclosed between "\${" and "}" substrings.

Expandable substrings are case-sensitive on Unix and case-insensitive on Windows. No string expansion occurs if the referenced environment variable is undefined.

Environment variables with names "APPDIR", "BINDIR", and "ROOTDIR" will be ignored, and these expandable substrings will be replaced by values calculated by the app launcher.

Prefix the dollar sign character with the backslash character (\) to prevent substring expansion.

--java-options options

Options to pass to the Java runtime

This option can be used multiple times.

Value can contain substrings that will be substituted at runtime, such as for the --arguments option.

--main-class class-name

Qualified name of the application main class to execute

This option can only be used if --main-jar is specified.

--main-jar main-jar

The main JAR of the application; containing the main class (specified as a path relative to the input path)

Either --module or --main-jar option can be specified but not both.

--module or -m module-name[/main-class]

The main module (and optionally main class) of the application

This module must be located on the module path.

When this option is specified, the main module will be linked in the Java runtime image. Either --module or --main-jar option can be specified but not both.

Platform dependent options for creating the application launcher:

Windows platform options (available only when running on Windows):

--win-console

Creates a console launcher for the application, should be specified for application which requires console interactions

macOS platform options (available only when running on macOS):

--mac-package-identifier identifier

An identifier that uniquely identifies the application for macOS

Defaults to the main class name.

May only use alphanumeric (A-Z,a-z,0-9), hyphen (-), and period (.) characters.

--mac-package-name name

Name of the application as it appears in the Menu Bar

This can be different from the application name.

This name must be less than 16 characters long and be suitable for displaying in the menu bar and the application Info window. Defaults to the application name.

--mac-package-signing-prefix prefix

When signing the application package, this value is prefixed to all components that

need to be signed that don't have an existing package identifier.

--mac-sign

Request that the package or the predefined application image be signed.

--mac-signing-keychain keychain-name

Name of the keychain to search for the signing identity

If not specified, the standard keychains are used.

--mac-signing-key-user-name name

Team or user name portion in Apple signing identities

--mac-app-store

Indicates that the jpackage output is intended for the Mac App Store.

--mac-entitlements path

Path to file containing entitlements to use when signing executables and libraries in the bundle

--mac-app-category category

String used to construct LSAApplicationCategoryType in application plist

The default value is "utilities".

Options for creating the application package:

--about-url url

URL of the application's home page

--app-image directory

Location of the predefined application image that is used to build an installable package (on all platforms) or to be signed (on macOS)
(absolute path or relative to the current directory)

--file-associations path

Path to a Properties file that contains list of key, value pairs
(absolute path or relative to the current directory)

The keys "extension", "mime-type", "icon", and "description" can be used to describe the association.

This option can be used multiple times.

--install-dir path

Absolute path of the installation directory of the application (on macOS or linux), or relative sub-path of the installation directory such as "Program Files" or "App? Data" (on Windows)

--license-file path

Path to the license file

(absolute path or relative to the current directory)

--runtime-image path

Path of the predefined runtime image to install

(absolute path or relative to the current directory)

Option is required when creating a runtime installer.

--launcher-as-service

Request to create an installer that will register the main application launcher as a background service-type application.

Platform dependent options for creating the application package:

Windows platform options (available only when running on Windows):

--win-dir-chooser

Adds a dialog to enable the user to choose a directory in which the product is installed.

--win-help-url url

URL where user can obtain further information or technical support

--win-menu

Request to add a Start Menu shortcut for this application

--win-menu-group menu-group-name

Start Menu group this application is placed in

--win-per-user-install

Request to perform an install on a per-user basis

--win-shortcut

Request to create a desktop shortcut for this application

--win-shortcut-prompt

Adds a dialog to enable the user to choose if shortcuts will be created by installer

--win-update-url url

URL of available application update information

--win-upgrade-uuid id

UUID associated with upgrades for this package

Linux platform options (available only when running on Linux):

--linux-package-name name

Name for Linux package

Defaults to the application name.

--linux-deb-maintainer email-address

Maintainer for .deb bundle

--linux-menu-group menu-group-name

Menu group this application is placed in

--linux-package-deps package-dep-string

Required packages or capabilities for the application

--linux-rpm-license-type type

Type of the license ("License: value" of the RPM .spec)

--linux-app-release release

Release value of the RPM <name>.spec file or Debian revision value of the DEB control file

--linux-app-category category-value

Group value of the RPM <name>.spec file or Section value of DEB control file

--linux-shortcut

Creates a shortcut for the application.

macOS platform options (available only when running on macOS):

--mac-dmg-content additional-content [,additional-content...]

Include all the referenced content in the dmg.

This option can be used more than once.

JPACKAGE EXAMPLES

Generate an application package suitable for the host system:

For a modular application:

```
jpackage -n name -p modulePath -m moduleName/className
```

For a non-modular application:

```
jpackage -i inputDir -n name \  
--main-class className --main-jar myJar.jar
```

From a pre-built application image:

```
jpackage -n name --app-image appImageDir
```

Generate an application image:

For a modular application:

```
jpackage --type app-image -n name -p modulePath \
```

-m moduleName/className

For a non-modular application:

```
jpackage --type app-image -i inputDir -n name \
--main-class className --main-jar myJar.jar
```

To provide your own options to jlink, run jlink separately:

```
jlink --output appRuntimeImage -p modulePath \
--add-modules moduleName \
--no-header-files [<additional jlink options>...]

jpackage --type app-image -n name \
-m moduleName/className --runtime-image appRuntimeImage
```

Generate a Java runtime package:

```
jpackage -n name --runtime-image <runtime-image>
```

Sign the predefined application image (on macOS):

```
jpackage --type app-image --app-image <app-image> \
--mac-sign [<additional signing options>...]
```

Note: the only additional options that are permitted in this mode are:

the set of additional mac signing options and --verbose

JPACKAGE AND JLINK

jpackage will use jlink to create Java Runtime unless the --runtime-image option is used.

The created Java Runtime image on Windows will include MS runtime libraries bundled with the JDK. If MS runtime libraries of a different version are needed for the application, the user will need to add/replace those themselves.

JPACKAGE RESOURCE DIRECTORY

Icons, template files, and other resources of jpackage can be over-ridden by adding replacement resources to this directory. jpackage will lookup files by specific names in the resource directory.

Resource directory files considered only when running on Linux:

<launcher-name>.png

Application launcher icon

Default resource is JavaApp.png

<launcher-name>.desktop

A desktop file to be used with xdg-desktop-menu command

Considered with application launchers registered for file associations and/or have an

icon

Default resource is template.desktop

Resource directory files considered only when building Linux DEB/RPM installer:

<package-name>-<launcher-name>.service

systemd unit file for application launcher registered as a background service-type

application

Default resource is unit-template.service

Resource directory files considered only when building Linux RPM installer:

<package-name>.spec

RPM spec file

Default resource is template.spec

Resource directory files considered only when building Linux DEB installer:

control

Control file

Default resource is template.control

copyright

Copyright file

Default resource is template.copyright

preinstall

Pre-install shell script

Default resource is template.preinstall

prerm Pre-remove shell script

Default resource is template.prerm

postinstall

Post-install shell script

Default resource is template.postinstall

postrm Post-remove shell script

Default resource is template.postrm

Resource directory files considered only when running on Windows:

<launcher-name>.ico

Application launcher icon

Default resource is JavaApp.ico

<launcher-name>.properties

Properties file for application launcher executable

Default resource is WinLauncher.template

Resource directory files considered only when building Windows MSI/EXE installer:

<application-name>-post-image.wsf

A Windows Script File (WSF) to run after building application image

main.wxs

Main WiX project file

Default resource is main.wxs

overrides.wxi

Overrides WiX project file

Default resource is overrides.wxi

service-installer.exe

Service installer executable

Considered if some application launchers are registered as background service-type applications

<launcher-name>-service-install.wxi

Service installer WiX project file

Considered if some application launchers are registered as background service-type applications

Default resource is service-install.wxi

<launcher-name>-service-config.wxi

Service installer WiX project file

Considered if some application launchers are registered as background service-type applications

Default resource is service-config.wxi

InstallDirNotEmptyDlg.wxs

WiX project file for installer UI dialog checking installation directory doesn't exist or is empty

Default resource is InstallDirNotEmptyDlg.wxs

ShortcutPromptDlg.wxs

WiX project file for installer UI dialog configuring shortcuts

Default resource is ShortcutPromptDlg.wxs

bundle.wxf

WiX project file with the hierarchy of components of application image

ui.wxf WiX project file for installer UI

os-condition.wxf

WiX project file with the condition to block installation on older versions of Windows

Default resource is os-condition.wxf

Default resource is os-condition.wxf

wix-conv.xsl

WiX source code converter. Used for converting WiX sources from WiX v3 to v4 schema

when WiX v4 or newer is used

Default resource is wix3-to-wix4-conv.xsl

Resource directory files considered only when building Windows EXE installer:

WinInstaller.properties

Properties file for the installer executable

Default resource is WinInstaller.template

<package-name>-post-msi.wsf

A Windows Script File (WSF) to run after building embedded MSI installer for EXE installer

Default resource is <package-name>-post-msi.wsf

installer.exe

Executable wrapper for MSI installer

Default resource is msiwrapper.exe

Resource directory files considered only when running on macOS:

<launcher-name>.icns

Application launcher icon

Default resource is JavaApp.icns

Info.plist

Application property list file

Default resource is Info-lite.plist.template

Runtime-Info.plist

Java Runtime property list file

Default resource is Runtime-Info.plist.template

<application-name>.entitlements

Signing entitlements property list file

Default resource is sandbox.plist

Resource directory files considered only when building macOS PKG/DMG installer:

<package-name>-post-image.sh

Shell script to run after building application image

Resource directory files considered only when building macOS PKG installer:

uninstaller

Uninstaller shell script

Considered if some application launchers are registered as background service-type applications

Default resource is uninstall.command.template

preinstall

Pre-install shell script

Default resource is preinstall.template

postinstall

Post-install shell script

Default resource is postinstall.template

services-preinstall

Pre-install shell script for services package

Considered if some application launchers are registered as background service-type applications

Default resource is services-preinstall.template

services-postinstall

Post-install shell script for services package

Considered if some application launchers are registered as background service-type applications

Default resource is services-postinstall.template

<package-name>-background.png

Background image

Default resource is background_pkg.png

<package-name>-background-darkAqua.png

Dark background image

Default resource is background_pkg.png

product-def.plist

Package property list file

Default resource is product-def.plist

<package-name>-<launcher-name>.plist

launchd property list file for application launcher registered as a background ser?

vice-type application

Default resource is launchd.plist.template

Resource directory files considered only when building macOS DMG installer:

<package-name>-dmg-setup.scpt

Setup AppleScript script

Default resource is DMGsetup.scpt

<package-name>-license.plist

License property list file

Default resource is lic_template.plist

<package-name>-background.tiff

Background image

Default resource is background_dmg.tiff

<package-name>-volume.icns

Volume icon

Default resource is JavaApp.icns