

NAME

cpack-generators – CPack Generator Reference

GENERATORS**CPack Archive Generator**

CPack generator for packaging files into an archive, which can have any of the following formats:

- 7Z – 7zip – (.7z)

Added in version 3.1.

- TAR (.tar)

Added in version 4.0.

- TBZ2 (.tar.bz2)
- TGZ (.tar.gz)
- TXZ (.tar.xz)

Added in version 3.1.

- TZ (.tar.Z)
- TZST (.tar.zst)

Added in version 3.16.

- ZIP (.zip)

When this generator is called from **CPackSourceConfig.cmake** (or through the **package_source** target), then the generated archive will contain all files in the project directory, except those specified in *CPACK_SOURCE_IGNORE_FILES*. The following is one example of packaging all source files of a project:

```
set(CPACK_SOURCE_GENERATOR "TGZ")
set(CPACK_SOURCE_IGNORE_FILES
    \\ .git/
    build/
    ". *~$")
)
set(CPACK_VERBATIM_VARIABLES YES)
include(CPack)
```

When this generator is called from **CPackConfig.cmake** (or through the **package** target), then the generated archive will contain all files that have been installed via CMake's *install()* command (and the deprecated commands *install_files()*, *install_programs()*, and *install_targets()*).

Variables specific to CPack Archive generator**CPACK_ARCHIVE_FILE_NAME**

Added in version 3.9.

Archive name for component-based packages, without extension.

Default*CPACK_PACKAGE_FILE_NAME*

The extension is appended automatically.

If *CPACK_COMPONENTS_GROUPING* is set to **ALL_COMPONENTS_IN_ONE**, this will be the name of the one output archive.

Changed in version 4.0: This variable also works for non-component packages.

CPACK_ARCHIVE_<component>_FILE_NAME

Added in version 3.9.

Component archive name without extension.

Default

<*CPACK_ARCHIVE_FILE_NAME*>-<**component**>, with spaces replaced by '-'.
 The extension is appended automatically. Note that <**component**> is all uppercase in the variable name.

The extension is appended automatically. Note that <**component**> is all uppercase in the variable name.

CPACK_ARCHIVE_FILE_EXTENSION

Added in version 3.25.

Archive file extension.

Default

Default values are given in the list above.

CPACK_ARCHIVE_COMPONENT_INSTALL

Enable component packaging.

Default

OFF

If enabled (**ON**) multiple packages are generated. By default a single package containing files of all components is generated.

Variables used by CPack Archive generator

These variables are used by the Archive generator, but are also available to CPack generators which are essentially archives at their core. These include:

- *CPack Cygwin Generator*
- *CPack FreeBSD Generator*

CPACK_ARCHIVE_THREADS

Added in version 3.18.

The number of threads to use when performing the compression.

Default

value of *CPACK_THREADS*

If set to **0**, the number of available cores on the machine will be used instead. Note that not all compression modes support threading in all environments.

Added in version 3.21: Official CMake binaries available on **cmake.org** now ship with a **liblzma** that supports parallel compression. Older versions did not.

CPack Bundle Generator

CPack Bundle generator (macOS) specific options

Variables specific to CPack Bundle generator

Installers built on macOS using the Bundle generator use the aforementioned DragNDrop (**CPACK_DMG_XXX**) variables, plus the following Bundle-specific parameters (**CPACK_BUNDLE_XXX**).

CPACK_BUNDLE_NAME

The name of the generated bundle. This appears in the macOS Finder as the bundle name. Required.

CPACK_BUNDLE_PLIST

Path to an macOS Property List (**.plist**) file that will be used for the generated bundle. This assumes that the caller has generated or specified their own **Info.plist** file. Required.

CPACK_BUNDLE_ICON

Path to an macOS icon file that will be used as the icon for the generated bundle. This is the icon that appears in the macOS Finder for the bundle, and in the macOS dock when the bundle is opened. Required.

CPACK_BUNDLE_STARTUP_COMMAND

Path to a startup script. This is a path to an executable or script that will be run whenever an end-user double-clicks the generated bundle in the macOS Finder. Optional.

CPACK_BUNDLE_APPLE_CERT_APP

Added in version 3.2.

The name of your Apple supplied code signing certificate for the application. The name usually takes the form **Developer ID Application: [Name]** or **3rd Party Mac Developer Application: [Name]**. If this variable is not set the application will not be signed.

CPACK_BUNDLE_APPLE_ENTITLEMENTS

Added in version 3.2.

The name of the Property List (**.plist**) file that contains your Apple entitlements for sandboxing your application. This file is required for submission to the macOS App Store.

CPACK_BUNDLE_APPLE_CODESIGN_FILES

Added in version 3.2.

A list of additional files that you wish to be signed. You do not need to list the main application folder, or the main executable. You should list any frameworks and plugins that are included in your app bundle.

CPACK_BUNDLE_APPLE_CODESIGN_PARAMETER

Added in version 3.3.

Additional parameter that will be passed to **codesign**. Default value: **--deep -f**

CPACK_COMMAND_CODESIGN

Added in version 3.2.

Path to the **codesign(1)** command used to sign applications with an Apple cert. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it).

CPack Cygwin Generator

Cygwin CPack generator (Cygwin).

Variables affecting the CPack Cygwin generator

- Added in version 3.18: *CPACK_ARCHIVE_THREADS*

Variables specific to CPack Cygwin generator

The following variable is specific to installers build on and/or for Cygwin:

CPACK_CYGWIN_PATCH_NUMBER

The Cygwin patch number. *FIXME*: This documentation is incomplete.

CPACK_CYGWIN_PATCH_FILE

The Cygwin patch file. *FIXME*: This documentation is incomplete.

CPACK_CYGWIN_BUILD_SCRIPT

The Cygwin build script. *FIXME*: This documentation is incomplete.

CPack DEB Generator

The built in (binary) CPack DEB generator (Unix only)

Variables specific to CPack Debian (DEB) generator

The CPack DEB generator may be used to create DEB package using *CPack*. The CPack DEB generator is a *CPack* generator thus it uses the **CPACK_XXX** variables used by *CPack*.

The CPack DEB generator should work on any Linux host but it will produce better deb package when Debian specific tools **dpkg-xxx** are usable on the build system.

The CPack DEB generator has specific features which are controlled by the specifics **CPACK_DEBIAN_XXX** variables.

CPACK_DEBIAN_<COMPONENT>_XXXX variables may be used in order to have **component** specific values. Note however that **<COMPONENT>** refers to the **grouping name** written in upper case. It may be either a component name or a component GROUP name.

Here are some CPack DEB generator wiki resources that are here for historic reasons and are no longer maintained but may still prove useful:

- <https://gitlab.kitware.com/cmake/community/-/wikis/doc/cpack/Configuration>
- <https://gitlab.kitware.com/cmake/community/-/wikis/doc/cpack/PackageGenerators#deb-unix-only>

List of CPack DEB generator specific variables:

CPACK_DEB_COMPONENT_INSTALL

Enable component packaging for CPackDEB

Mandatory

No

Default

OFF

If enabled (**ON**) multiple packages are generated. By default a single package containing files of all components is generated.

CPACK_DEBIAN_PACKAGE_NAME**CPACK_DEBIAN_<COMPONENT>_PACKAGE_NAME**

Set Package control field (variable is automatically transformed to lower case).

Mandatory

Yes

Default

- *CPACK_PACKAGE_NAME* for non-component based installations
- *CPACK_DEBIAN_PACKAGE_NAME* suffixed with *-<COMPONENT>* for component-based installations.

Added in version 3.5: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_NAME** variables.

See <https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-source>

CPACK_DEBIAN_FILE_NAME**CPACK_DEBIAN_<COMPONENT>_FILE_NAME**

Added in version 3.6.

Package file name.

Mandatory

Yes

Default

<CPACK_PACKAGE_FILE_NAME>[-<component>].deb

This may be set to:

DEB-DEFAULT

Tell CPack to automatically generate the package file name in deb format:

<PackageName>_<VersionNumber>-<DebianRevisionNumber>_<DebianArchitecture>

This setting recommended as the preferred behavior, but for backward compatibility with the CPack DEB generator in CMake prior to version 3.6, this is not the default. Without this, duplicate names may occur. Duplicate files get overwritten and it is up to the packager to set the variables in a manner that will prevent such errors.

<file-name>[.deb]

Use the given file name.

Changed in version 3.29: The **.deb** suffix will be automatically added if the file name does not end in **.deb** or **.ipk**. Previously the suffix was required.

<file-name>.ipk

Added in version 3.10.

Use the given file name. The **.ipk** suffix is used by the OPKG packaging system.

CPACK_DEBIAN_PACKAGE_EPOCH

Added in version 3.10.

The Debian package epoch

Mandatory

No

Default

None

Optional number that should be incremented when changing versioning schemas or fixing mistakes in the version numbers of older packages.

CPACK_DEBIAN_PACKAGE_VERSION

The Debian package version

Mandatory

Yes

Default

CPACK_PACKAGE_VERSION

This variable may contain only alphanumerics (A–Za–z0–9) and the characters . + – ~ (full stop, plus, hyphen, tilde) and should start with a digit. If *CPACK_DEBIAN_PACKAGE_RELEASE* is not set then hyphens are not allowed.

NOTE:

For backward compatibility with CMake 3.9 and lower a failed test of this variable's content is not a hard error when both *CPACK_DEBIAN_PACKAGE_RELEASE* and *CPACK_DEBIAN_PACKAGE_EPOCH* variables are not set. An author warning is reported instead.

CPACK_DEBIAN_PACKAGE_RELEASE

Added in version 3.6.

The Debian package release – Debian revision number.

Mandatory

No

Default

None

This is the numbering of the DEB package itself, i.e. the version of the packaging and not the version of the content (see *CPACK_DEBIAN_PACKAGE_VERSION*). One may change the default value if the previous packaging was buggy and/or you want to put here a fancy Linux distro specific numbering.

CPACK_DEBIAN_PACKAGE_ARCHITECTURE**CPACK_DEBIAN_<COMPONENT>_PACKAGE_ARCHITECTURE**

The Debian package architecture

Mandatory

Yes

Default

Output of **dpkg --print-architecture** (or **i386** if **dpkg** is not found)

Added in version 3.6: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_ARCHITECTURE** variables.

CPACK_DEBIAN_PACKAGE_DEPENDS

CPACK_DEBIAN_<COMPONENT>_PACKAGE_DEPENDS

Sets the Debian dependencies of this package.

Mandatory

No

Default

- An empty string for non-component based installations
- **CPACK_DEBIAN_PACKAGE_DEPENDS** for component-based installations.

Added in version 3.3: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_DEPENDS** variables.

NOTE:

If **CPACK_DEBIAN_PACKAGE_SHLIBDEPS** or more specifically **CPACK_DEBIAN_<COMPONENT>_PACKAGE_SHLIBDEPS** is set for this component, the discovered dependencies will be appended to **CPACK_DEBIAN_<COMPONENT>_PACKAGE_DEPENDS** instead of **CPACK_DEBIAN_PACKAGE_DEPENDS**. If **CPACK_DEBIAN_<COMPONENT>_PACKAGE_DEPENDS** is an empty string, only the automatically discovered dependencies will be set for this component.

Changed in version 3.31: The variable is always expanded as a list. Before it was expanded only if used in cooperation with **CPACK_DEB_COMPONENT_INSTALL**, **CPACK_DEBIAN_PACKAGE_SHLIBDEPS** or **CPACK_DEBIAN_<COMPONENT>_PACKAGE_SHLIBDEPS**. This meant that if a component had no shared libraries discovered (e.g. a package composed only of scripts) you had to join the list by yourself to obtain a valid Depends field.

Example:

```
set(CPACK_DEBIAN_PACKAGE_DEPENDS "libc6 (>= 2.3.1-6), libc6 (< 2.4)")
list(APPEND CPACK_DEBIAN_PACKAGE_DEPENDS cmake)
```

CPACK_DEBIAN_ENABLE_COMPONENT_DEPENDS

Added in version 3.6.

Sets inter-component dependencies if listed with **CPACK_COMPONENT_<compName>_DEPENDS** variables.

Mandatory

No

Default

None

CPACK_DEBIAN_PACKAGE_MAINTAINER

The Debian package maintainer

Mandatory

Yes

Default**CPACK_PACKAGE_CONTACT****CPACK_DEBIAN_PACKAGE_DESCRIPTION****CPACK_DEBIAN_<COMPONENT>_DESCRIPTION**

The Debian package description

Mandatory

Yes

Default

- *CPACK_DEBIAN_<COMPONENT>_DESCRIPTION* (component based installers only) if set, or *CPACK_DEBIAN_PACKAGE_DESCRIPTION* if set, or
- *CPACK_COMPONENT_<compName>_DESCRIPTION* (component based installers only) if set, or *CPACK_PACKAGE_DESCRIPTION* if set, or
- content of the file specified in *CPACK_PACKAGE_DESCRIPTION_FILE* if set

If after that description is not set, *CPACK_PACKAGE_DESCRIPTION_SUMMARY* going to be used if set. Otherwise, *CPACK_PACKAGE_DESCRIPTION_SUMMARY* will be added as the first line of description as defined in *Debian Policy Manual*.

Added in version 3.3: Per-component **CPACK_COMPONENT_<compName>_DESCRIPTION** variables.

Added in version 3.16: Per-component **CPACK_DEBIAN_<COMPONENT>_DESCRIPTION** variables.

Added in version 3.16: The **CPACK_PACKAGE_DESCRIPTION_FILE** variable.

CPACK_DEBIAN_PACKAGE_SECTION**CPACK_DEBIAN_<COMPONENT>_PACKAGE_SECTION**

Set Section control field e.g. admin, devel, doc, ...

Mandatory

Yes

Default**devel**

Added in version 3.5: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_SECTION** variables.

See <https://www.debian.org/doc/debian-policy/ch-archive.html#s-subsections>

CPACK_DEBIAN_ARCHIVE_TYPE

Added in version 3.7.

Deprecated since version 3.14.

The archive format used for creating the Debian package.

Mandatory

Yes

Default

gnutar

Possible value is: **gnutar**

NOTE:

This variable previously defaulted to the **paxr** value, but **dpkg** has never supported that tar format. For backwards compatibility the **paxr** value will be mapped to **gnutar** and a deprecation message will be emitted.

CPACK_DEBIAN_COMPRESSION_TYPE

Added in version 3.1.

The compression used for creating the Debian package.

Mandatory

Yes

Default

gzip

Possible values are:

lzma	Lempel–Ziv–Markov chain algorithm
xz	XZ Utils compression
bzip2	bzip2 Burrows–Wheeler algorithm
gzip	GNU Gzip compression
zstd	Added in version 3.22.

Zstandard compression

CPACK_DEBIAN_PACKAGE_PRIORITY

CPACK_DEBIAN_<COMPONENT>_PACKAGE_PRIORITY

Set Priority control field e.g. required, important, standard, optional, extra

Mandatory

Yes

Default

optional

Added in version 3.5: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_PRIORITY** variables.

See <https://www.debian.org/doc/debian-policy/ch-archive.html#s-priorities>

CPACK_DEBIAN_PACKAGE_HOMEPAGE

The URL of the web site for this package, preferably (when applicable) the site from which the original source can be obtained and any additional upstream documentation or information may be found.

Mandatory

No

Default*CMAKE_PROJECT_HOMEPAGE_URL*

Added in version 3.12: The **CMAKE_PROJECT_HOMEPAGE_URL** variable.

NOTE:

The content of this field is a simple URL without any surrounding characters such as `<>`.

CPACK_DEBIAN_PACKAGE_SHLIBDEPS**CPACK_DEBIAN_<COMPONENT>_PACKAGE_SHLIBDEPS**

May be set to ON in order to use **dpkg-shlibdeps** to generate better package dependency list.

Mandatory

No

Default

- *CPACK_DEBIAN_PACKAGE_SHLIBDEPS* if set or
- **OFF**

NOTE:

You may need set *CMAKE_INSTALL_RPATH* to an appropriate value if you use this feature, because if you don't **dpkg-shlibdeps** may fail to find your own shared libs. See - <https://gitlab.kitware.com/cmake/community/-/wikis/doc/cmake/RPATH-handling>

NOTE:

You can also set *CPACK_DEBIAN_PACKAGE_SHLIBDEPS_PRIVATE_DIRS* to an appropriate value if you use this feature, in order to please **dpkg-shlibdeps**. However, you should only do this for private shared libraries that could not get resolved otherwise.

Added in version 3.3: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_SHLIBDEPS** variables.

Added in version 3.6: Correct handling of **\$ORIGIN** in *CMAKE_INSTALL_RPATH*.

CPACK_DEBIAN_PACKAGE_SHLIBDEPS_PRIVATE_DIRS

Added in version 3.20.

May be set to a list of directories that will be given to **dpkg-shlibdeps** via its **-l** option. These will be searched by **dpkg-shlibdeps** in order to find private shared library dependencies.

Mandatory

No

Default

None

NOTE:

You should prefer to set *CMAKE_INSTALL_RPATH* to an appropriate value if you use **dpkg-shlibdeps**. The current option is really only needed for private shared library dependencies.

CPACK_DEBIAN_PACKAGE_DEBUG

May be set when invoking `cpack` in order to trace debug information during the CPack DEB generator run.

Mandatory

No

Default

None

CPACK_DEBIAN_PACKAGE_PREDEPENDS**CPACK_DEBIAN_<COMPONENT>_PACKAGE_PREDEPENDS**

Sets the **Pre-Depends** field of the Debian package. Like *Depends*, except that it also forces **dpkg** to complete installation of the packages named before even starting the installation of the package which declares the pre-dependency.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_PREDEPENDS` for component-based installations.

Added in version 3.4: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_PREDEPENDS** variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps>

CPACK_DEBIAN_PACKAGE_ENHANCES**CPACK_DEBIAN_<COMPONENT>_PACKAGE_ENHANCES**

Sets the **Enhances** field of the Debian package. Similar to *Suggests* but works in the opposite direction: declares that a package can enhance the functionality of another package.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_ENHANCES` for component-based installations.

Added in version 3.4: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_ENHANCES** variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps>

CPACK_DEBIAN_PACKAGE_BREAKS**CPACK_DEBIAN_<COMPONENT>_PACKAGE_BREAKS**

Sets the **Breaks** field of the Debian package. When a binary package (P) declares that it breaks other packages (B), **dpkg** will not allow the package (P) which declares **Breaks** be **unpacked** unless the packages that will be broken (B) are deconfigured first. As long as the package (P) is configured, the previously deconfigured packages (B) cannot be reconfigured again.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_BREAKS` for component-based installations.

Added in version 3.4: Per-component `CPACK_DEBIAN_<COMPONENT>_PACKAGE_BREAKS` variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-breaks>

CPACK_DEBIAN_PACKAGE_CONFLICTS**CPACK_DEBIAN_<COMPONENT>_PACKAGE_CONFLICTS**

Sets the **Conflicts** field of the Debian package. When one binary package declares a conflict with another using a **Conflicts** field, **dpkg** will not allow them to be unpacked on the system at the same time.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_CONFLICTS` for component-based installations.

Added in version 3.4: Per-component `CPACK_DEBIAN_<COMPONENT>_PACKAGE_CONFLICTS` variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-conflicts>

NOTE:

This is a stronger restriction than *Breaks*, which prevents the broken package from being configured while the breaking package is in the "Unpacked" state but allows both packages to be unpacked at the same time.

CPACK_DEBIAN_PACKAGE_PROVIDES**CPACK_DEBIAN_<COMPONENT>_PACKAGE_PROVIDES**

Sets the **Provides** field of the Debian package. A virtual package is one which appears in the **Provides** control field of another package.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_PROVIDES` for component-based installations.

Added in version 3.4: Per-component `CPACK_DEBIAN_<COMPONENT>_PACKAGE_PROVIDES` variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-virtual>

CPACK_DEBIAN_PACKAGE_REPLACES

CPACK_DEBIAN_<COMPONENT>_PACKAGE_REPLACES

Sets the **Replaces** field of the Debian package. Packages can declare in their control file that they should overwrite files in certain other packages, or completely replace other packages.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_REPLACES` for component-based installations.

Added in version 3.4: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_REPLACES** variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps>

CPACK_DEBIAN_PACKAGE_RECOMMENDS**CPACK_DEBIAN_<COMPONENT>_PACKAGE_RECOMMENDS**

Sets the **Recommends** field of the Debian package. Allows packages to declare a strong, but not absolute, dependency on other packages.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_RECOMMENDS` for component-based installations.

Added in version 3.4: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_RECOMMENDS** variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps>

CPACK_DEBIAN_PACKAGE_SUGGESTS**CPACK_DEBIAN_<COMPONENT>_PACKAGE_SUGGESTS**

Sets the **Suggests** field of the Debian package. Allows packages to declare a suggested package install grouping.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_SUGGESTS` for component-based installations.

Added in version 3.4: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_SUGGESTS** variables.

See <https://www.debian.org/doc/debian-policy/ch-relationships.html#s-binarydeps>

CPACK_DEBIAN_PACKAGE_GENERATE_SHLIBS

Added in version 3.6.

Mandatory

No

Default**OFF**

Allows to generate shlibs control file automatically. Compatibility is defined by `CPACK_DEBIAN_PACKAGE_GENERATE_SHLIBS_POLICY` variable value.

NOTE:

Libraries are only considered if they have both library name and version set. This can be done by setting `SOVERSION` property with `set_target_properties()` command.

CPACK_DEBIAN_PACKAGE_GENERATE_SHLIBS_POLICY

Added in version 3.6.

Compatibility policy for auto-generated shlibs control file.

Mandatory

No

Default

=

Defines compatibility policy for auto-generated shlibs control file. Possible values: =, >=

See <https://www.debian.org/doc/debian-policy/ch-sharedlibs.html#s-sharedlibs-shlibdeps>

CPACK_DEBIAN_PACKAGE_CONTROL_EXTRA**CPACK_DEBIAN_<COMPONENT>_PACKAGE_CONTROL_EXTRA**

This variable allow advanced user to add custom script to the control.tar.gz. Typical usage is for conffiles, postinst, postrm, prerm.

Mandatory

No

Default

None

Usage:

```
set(CPACK_DEBIAN_PACKAGE_CONTROL_EXTRA
    "${CMAKE_CURRENT_SOURCE_DIR}/prerm;${CMAKE_CURRENT_SOURCE_DIR}/postrm")
```

Added in version 3.4: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_CONTROL_EXTRA** variables.

CPACK_DEBIAN_PACKAGE_CONTROL_STRICT_PERMISSION**CPACK_DEBIAN_<COMPONENT>_PACKAGE_CONTROL_STRICT_PERMISSION**

Added in version 3.4.

This variable indicates if the Debian policy on control files should be strictly followed.

Mandatory

No

Default**FALSE**

Usage:

```
set(CPACK_DEBIAN_PACKAGE_CONTROL_STRICT_PERMISSION TRUE)
```

This overrides the permissions on the original files, following the rules set by Debian policy - <https://www.debian.org/doc/debian-policy/ch-files.html#s-permissions-owners>

NOTE:

The original permissions of the files will be used in the final package unless this variable is set to **TRUE**. In particular, the scripts should have the proper executable flag prior to the generation of the package.

CPACK_DEBIAN_PACKAGE_SOURCE**CPACK_DEBIAN_<COMPONENT>_PACKAGE_SOURCE**

Added in version 3.5.

Sets the **Source** field of the binary Debian package. When the binary package name is not the same as the source package name (in particular when several components/binaries are generated from one source) the source from which the binary has been generated should be indicated with the field **Source**.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_SOURCE` for component-based installations.

See <https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-f-source>

NOTE:

This value is not interpreted. It is possible to pass an optional revision number of the referenced source package as well.

CPACK_DEBIAN_PACKAGE_MULTIARCH**CPACK_DEBIAN_<COMPONENT>_PACKAGE_MULTIARCH**

Sets the **Multi-Arch** field of the Debian package. Packages can declare in their control file how they should handle situations, where packages for different architectures are being installed on the same machine.

Mandatory

No

Default

- An empty string for non-component based installations
- `CPACK_DEBIAN_PACKAGE_MULTIARCH` for component-based installations.

Added in version 3.31: Per-component **CPACK_DEBIAN_<COMPONENT>_PACKAGE_MULTIARCH** variables.

See <https://wiki.debian.org/MultiArch/Hints>

NOTE:

This value is validated. It must be one of the following values: **same**, **foreign**, **allowed**.

Packaging of debug information

Added in version 3.13.

DbgSYM packages contain debug symbols for debugging packaged binaries.

DbgSYM packaging has its own set of variables:

CPACK_DEBIAN_DEBUGINFO_PACKAGE**CPACK_DEBIAN_<component>_DEBUGINFO_PACKAGE**

Enable generation of dbgSYM .ddeb package(s).

Mandatory

No

Default

OFF

NOTE:

Setting this also strips the ELF files in the generated non-dbgSYM package, which results in debuginfo only being available in the dbgSYM package.

NOTE:

Binaries must contain debug symbols before packaging so use either **Debug** or **RelWithDebInfo** for *CMAKE_BUILD_TYPE* variable value.

Additionally, if *CPACK_STRIP_FILES* is set, the files will be stripped before they get to the DEB generator, so will not contain debug symbols and a dbgSYM package will not get built. Do not use with *CPACK_STRIP_FILES*.

Building Debian packages on Windows

Added in version 3.10.

To communicate UNIX file permissions from the install stage to the CPack DEB generator the **cmake_mode_t** NTFS alternate data stream (ADT) is used.

When a filesystem without ADT support is used only owner read/write permissions can be preserved.

Reproducible packages

Added in version 3.13.

The environment variable **SOURCE_DATE_EPOCH** may be set to a UNIX timestamp, defined as the number of seconds, excluding leap seconds, since 01 Jan 1970 00:00:00 UTC. If set, the CPack DEB generator will use its value for timestamps in the package.

CPack DragNDrop Generator

The DragNDrop CPack generator (macOS) creates a DMG image.

Variables specific to CPack DragNDrop generator

The following variables are specific to the DragNDrop installers built on macOS:

CPACK_DMG_VOLUME_NAME

The volume name of the generated disk image.

Default*CPACK_PACKAGE_FILE_NAME***CPACK_DMG_FORMAT**

The disk image format.

Default**UDZO**

Common values are **UDRO** (UDIF read-only), **UDZO** (UDIF zlib-compressed) or **UDBZ** (UDIF bzip2-compressed). Refer to **hdiutil(1)** for more information on other available formats.

CPACK_DMG_DS_STORE

Path to a custom **.DS_Store** file. This **.DS_Store** file can be used to specify the Finder window position/geometry and layout (such as hidden toolbars, placement of the icons etc.). This file has to be generated by the Finder (either manually or through AppleScript) using a normal folder from which the **.DS_Store** file can then be extracted.

CPACK_DMG_DS_STORE_SETUP_SCRIPT

Added in version 3.5.

Path to a custom AppleScript file. This AppleScript is used to generate a **.DS_Store** file which specifies the Finder window position/geometry and layout (such as hidden toolbars, placement of the icons etc.). By specifying a custom AppleScript there is no need to use **CPACK_DMG_DS_STORE**, as the **.DS_Store** that is generated by the AppleScript will be packaged.

CPACK_DMG_BACKGROUND_IMAGE**Default**

Path to an image file to be used as the background. This file will be copied to **.background/background.<ext>**, where **<ext>** is the original image file extension. The background image is installed into the image before **CPACK_DMG_DS_STORE_SETUP_SCRIPT** is executed or **CPACK_DMG_DS_STORE** is installed. By default no background image is set.

CPACK_DMG_DISABLE_APPLICATIONS_SYMLINK

Added in version 3.6.

Default behavior is to include a symlink to **/Applications** in the DMG. Set this option to **ON** to avoid adding the symlink.

CPACK_DMG_SLA_USE_RESOURCE_FILE_LICENSE

Added in version 3.23.

Default**OFF**

Control whether *CPACK_RESOURCE_FILE_LICENSE*, if set to a non-default value, is used as the license agreement provided when mounting the DMG. If **CPACK_DMG_SLA_USE_RESOURCE_FILE_LICENSE** is not set, *cpack(1)* defaults to off.

In a CMake project that uses the *CPack* module to generate **CPackConfig.cmake**, **CPACK_DMG_SLA_USE_RESOURCE_FILE_LICENSE** must be explicitly enabled by the project to activate the SLA. See policy *CMP0133*.

NOTE:

This option was added in response to macOS 12.0's deprecation of the **hdiutil udifrez** command to make its use optional. CPack 3.22 and below always use *CPACK_RESOURCE_FILE_LICENSE*, if set to a non-default value, as the DMG license.

CPACK_DMG_SLA_DIR

Added in version 3.5.

Directory where license and menu files for different languages are stored. Setting this causes CPack to look for a **<language>.menu.txt** and **<language>.license.txt** or **<language>.license.rtf** file for every language defined in *CPACK_DMG_SLA_LANGUAGES*. If both this variable and *CPACK_RESOURCE_FILE_LICENSE* are set, CPack will only look for the menu files and use the same license file for all languages. If both **<language>.license.txt** and **<language>.license.rtf** exist, the **.txt** file will be used.

Added in version 3.17: RTF support.

CPACK_DMG_SLA_LANGUAGES

Added in version 3.5.

Languages for which a license agreement is provided when mounting the generated DMG. A menu file consists of 9 lines of text. The first line is the name of the language itself, uppercase, in English (e.g. German). The other lines are translations of the following strings:

- Agree
- Disagree
- Print
- Save...
- You agree to the terms of the License Agreement when you click the "Agree" button.
- Software License Agreement
- This text cannot be saved. The disk may be full or locked, or the file may be locked.
- Unable to print. Make sure you have selected a printer.

For every language in this list, CPack will try to find files **<language>.menu.txt** and **<language>.license.txt** in the directory specified by the *CPACK_DMG_SLA_DIR* variable.

CPACK_DMG_<component>_FILE_NAME

Added in version 3.17.

File name when packaging **<component>** as its own DMG (*CPACK_COMPONENTS_GROUPING* set to **IGNORE**).

Default

CPACK_PACKAGE_FILE_NAME–**<component>**

CPACK_DMG_FILESYSTEM

Added in version 3.21.

Default

HFS+

The filesystem format. Common values are **APFS** and **HFS+**. See **man hdiutil** for a full list of

supported formats.

CPACK_COMMAND_HDIUTIL

Path to the **hdiutil(1)** command used to operate on disk image files on macOS. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it).

CPACK_COMMAND_SETFILE

Path to the **SetFile(1)** command used to set extended attributes on files and directories on macOS. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it).

CPACK_COMMAND_REZ

Path to the **Rez(1)** command used to compile resources on macOS. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it).

CPack External Generator

Added in version 3.13.

CPack provides many generators to create packages for a variety of platforms and packaging systems. The intention is for CMake/CPack to be a complete end-to-end solution for building and packaging a software project. However, it may not always be possible to use CPack for the entire packaging process, due to either technical limitations or policies that require the use of certain tools. For this reason, CPack provides the "External" generator, which allows external packaging software to take advantage of some of the functionality provided by CPack, such as component installation and the dependency graph.

Integration with External Packaging Tools

The CPack External generator generates a **.json** file containing the CPack internal metadata, which gives external software information on how to package the software. External packaging software may itself invoke CPack, consume the generated metadata, install and package files as required.

Alternatively CPack can invoke an external packaging software through an optional custom CMake script in *CPACK_EXTERNAL_PACKAGE_SCRIPT* instead.

Staging of installation files may also optionally be taken care of by the generator when enabled through the *CPACK_EXTERNAL_ENABLE_STAGING* variable.

JSON Format

The JSON metadata file contains a list of CPack components and component groups, the various options passed to *cpack_add_component()* and *cpack_add_component_group()*, the dependencies between the components and component groups, and various other options passed to CPack.

The JSON's root object will always provide two fields: **formatVersionMajor** and **formatVersionMinor**, which are always integers that describe the output format of the generator. Backwards-compatible changes to the output format (for example, adding a new field that didn't exist before) cause the minor version to be incremented, and backwards-incompatible changes (for example, deleting a field or changing its meaning) cause the major version to be incremented and the minor version reset to 0. The format version is always of the format **major.minor**. In other words, it always has exactly two parts, separated by a period.

You can request one or more specific versions of the output format as described below with *CPACK_EXTERNAL_REQUESTED_VERSIONS*. The output format will have a major version that exactly matches the requested major version, and a minor version that is greater than or equal to the requested minor version. If no version is requested with *CPACK_EXTERNAL_REQUESTED_VERSIONS*, the latest known major version is used by default. Currently, the only supported format is 1.0, which is described below.

Version 1.0

In addition to the standard format fields, format version 1.0 provides the following fields in the root:

components

The **components** field is an object with component names as the keys and objects describing the components as the values. The component objects have the following fields:

name The name of the component. This is always the same as the key in the **components** object.

displayName

The value of the **DISPLAY_NAME** field passed to *cpack_add_component()*.

description

The value of the **DESCRIPTION** field passed to *cpack_add_component()*.

isHidden

True if **HIDDEN** was passed to *cpack_add_component()*, false if it was not.

isRequired

True if **REQUIRED** was passed to *cpack_add_component()*, false if it was not.

isDisabledByDefault

True if **DISABLED** was passed to *cpack_add_component()*, false if it was not.

group Only present if **GROUP** was passed to *cpack_add_component()*. If so, this field is a string value containing the component's group.

dependencies

An array of components the component depends on. This contains the values in the **DEPENDS** argument passed to *cpack_add_component()*. If no **DEPENDS** argument was passed, this is an empty list.

installationTypes

An array of installation types the component is part of. This contains the values in the **INSTALL_TYPES** argument passed to *cpack_add_component()*. If no **INSTALL_TYPES** argument was passed, this is an empty list.

isDownloaded

True if **DOWNLOADED** was passed to *cpack_add_component()*, false if it was not.

archiveFile

The name of the archive file passed with the **ARCHIVE_FILE** argument to *cpack_add_component()*. If no **ARCHIVE_FILE** argument was passed, this is an empty string.

componentGroups

The **componentGroups** field is an object with component group names as the keys and objects describing the component groups as the values. The component group objects have the following fields:

name The name of the component group. This is always the same as the key in the **componentGroups** object.

displayName

The value of the **DISPLAY_NAME** field passed to *cpack_add_component_group()*.

description

The value of the **DESCRIPTION** field passed to *cpack_add_component_group()*.

parentGroup

Only present if **PARENT_GROUP** was passed to *cpack_add_component_group()*. If so, this field is a string value containing the component group's parent group.

isExpandedByDefault

True if **EXPANDED** was passed to *cpack_add_component_group()*, false if it was not.

isBold True if **BOLD_TITLE** was passed to *cpack_add_component_group()*, false if it was not.

components

An array of names of components that are direct members of the group (components that have this group as their **GROUP**). Components of subgroups are not included.

subgroups

An array of names of component groups that are subgroups of the group (groups that have this group as their **PARENT_GROUP**).

installationTypes

The **installationTypes** field is an object with installation type names as the keys and objects describing the installation types as the values. The installation type objects have the following fields:

name The name of the installation type. This is always the same as the key in the **installationTypes** object.

displayName

The value of the **DISPLAY_NAME** field passed to *cpack_add_install_type()*.

index The integer index of the installation type in the list.

projects

The **projects** field is an array of objects describing CMake projects which comprise the CPack project. The values in this field are derived from *CPACK_INSTALL_CMAKE_PROJECTS*. In most cases, this will be only a single project. The project objects have the following fields:

projectName

The project name passed to *CPACK_INSTALL_CMAKE_PROJECTS*.

component

The name of the component or component set which comprises the project.

directory

The build directory of the CMake project. This is the directory which contains the **cmake_install.cmake** script.

subDirectory

The subdirectory to install the project into inside the CPack package.

packageName

The package name given in *CPACK_PACKAGE_NAME*. Only present if this option is set.

packageVersion

The package version given in *CPACK_PACKAGE_VERSION*. Only present if this option is set.

packageDescriptionFile

The package description file given in *CPACK_PACKAGE_DESCRIPTION_FILE*. Only present if this option is set.

packageDescriptionSummary

The package description summary given in *CPACK_PACKAGE_DESCRIPTION_SUMMARY*. Only present if this option is set.

buildConfig

The build configuration given to CPack with the *cpack -C* option. Only present if this option is set.

defaultDirectoryPermissions

The default directory permissions given in *CPACK_INSTALL_DEFAULT_DIRECTORY_PERMISSIONS*. Only present if this option is set.

setDestdir

True if `CPACK_SET_DESTDIR` is true, false if it is not.

packagingInstallPrefix

The install prefix given in `CPACK_PACKAGING_INSTALL_PREFIX`. Only present if `CPACK_SET_DESTDIR` is true.

stripFiles

True if `CPACK_STRIP_FILES` is true, false if it is not.

warnOnAbsoluteInstallDestination

True if `CPACK_WARN_ON_ABSOLUTE_INSTALL_DESTINATION` is true, false if it is not.

errorOnAbsoluteInstallDestination

True if `CPACK_ERROR_ON_ABSOLUTE_INSTALL_DESTINATION` is true, false if it is not.

Variables specific to CPack External generator**CPACK_EXTERNAL_REQUESTED_VERSIONS**

This variable is used to request a specific version of the CPack External generator. It is a list of **major.minor** values, separated by semicolons.

If this variable is set to a non-empty value, the CPack External generator will iterate through each item in the list to search for a version that it knows how to generate. Requested versions should be listed in order of descending preference by the client software, as the first matching version in the list will be generated.

The generator knows how to generate the version if it has a versioned generator whose major version exactly matches the requested major version, and whose minor version is greater than or equal to the requested minor version. For example, if **CPACK_EXTERNAL_REQUESTED_VERSIONS** contains 1.0, and the CPack External generator knows how to generate 1.1, it will generate 1.1. If the generator doesn't know how to generate a version in the list, it skips the version and looks at the next one. If it doesn't know how to generate any of the requested versions, an error is thrown.

If this variable is not set, or is empty, the CPack External generator will generate the highest major and minor version that it knows how to generate.

If an invalid version is encountered in **CPACK_EXTERNAL_REQUESTED_VERSIONS** (one that doesn't match **major.minor**, where **major** and **minor** are integers), it is ignored.

CPACK_EXTERNAL_ENABLE_STAGING

This variable can be set to true to enable optional installation into a temporary staging area which can then be picked up and packaged by an external packaging tool. The top level directory used by CPack for the current packaging task is contained in **CPACK_TOPLEVEL_DIRECTORY**. It is automatically cleaned up on each run before packaging is initiated and can be used for custom temporary files required by the external packaging tool. It also contains the staging area **CPACK_TEMPORARY_DIRECTORY** into which CPack performs the installation when staging is enabled.

CPACK_EXTERNAL_PACKAGE_SCRIPT

This variable can optionally specify the full path to a CMake script file to be run as part of the CPack invocation. It is invoked after (optional) staging took place and may run an external packaging tool. The script has access to the variables defined by the CPack config file.

CPACK_EXTERNAL_BUILT_PACKAGES

Added in version 3.19.

The **CPACK_EXTERNAL_PACKAGE_SCRIPT** script may set this list variable to the full

paths of generated package files. CPack will copy these files from the staging directory back to the top build directory and possibly produce checksum files if the `CPACK_PACKAGE_CHECKSUM` is set.

CPack FreeBSD Generator

Added in version 3.10.

The built in (binary) CPack FreeBSD (pkg) generator (Unix only)

Variables affecting the CPack FreeBSD (pkg) generator

- Added in version 3.18: `CPACK_ARCHIVE_THREADS`

Variables specific to CPack FreeBSD (pkg) generator

The CPack FreeBSD generator may be used to create pkg(8) packages -- these may be used on FreeBSD, DragonflyBSD, NetBSD, OpenBSD, but also on Linux or OSX, depending on the installed package-management tools -- using *CPack*.

The CPack FreeBSD generator is a *CPack* generator and uses the **CPACK_XXX** variables used by *CPack*. It tries to reuse packaging information that may already be specified for Debian packages for the *CPack DEB Generator*. It also tries to reuse RPM packaging information when Debian does not specify.

The CPack FreeBSD generator should work on any host with libpkg installed. The packages it produces are specific to the host architecture and ABI.

The CPack FreeBSD generator sets package-metadata through **CPACK_FREEBSD_XXX** variables. The CPack FreeBSD generator, unlike the CPack Deb generator, does not specially support componentized packages; a single package is created from all the software artifacts created through CMake.

All of the variables can be set specifically for FreeBSD packaging in the CPackConfig file or in CMakeLists.txt, but most of them have defaults that use general settings (e.g. `CMAKE_PROJECT_NAME`) or Debian-specific variables when those make sense (e.g. the homepage of an upstream project is usually unchanged by the flavor of packaging). When there is no Debian information to fall back on, but the RPM packaging has it, fall back to the RPM information (e.g. package license).

CPACK_FREEBSD_PACKAGE_NAME

Sets the package name (in the package manifest, but also affects the output filename).

Mandatory

Yes

Default

- `CPACK_PACKAGE_NAME` (this is always set by CPack itself, based on `CMAKE_PROJECT_NAME`).

CPACK_FREEBSD_PACKAGE_COMMENT

Sets the package comment. This is the short description displayed by pkg(8) in standard "pkg info" output.

Mandatory

Yes

Default

- `CPACK_PACKAGE_DESCRIPTION_SUMMARY` (this is always set by CPack itself, if nothing else sets it explicitly).

CPACK_FREEBSD_PACKAGE_DESCRIPTION

Sets the package description. This is the long description of the package, given by "pkg info" with a specific package as argument.

Mandatory

Yes

Default

- *CPACK_DEBIAN_PACKAGE_DESCRIPTION* (this may be set already for Debian packaging, so it is used as a fallback).
- *CPACK_PACKAGE_DESCRIPTION_SUMMARY* (this is always set by CPack itself, if nothing else sets it explicitly).
- *PROJECT_DESCRIPTION* (this can be set with the **DESCRIPTION** parameter for *project()*).

CPACK_FREEBSD_PACKAGE_WWW

The URL of the web site for this package, preferably (when applicable) the site from which the original source can be obtained and any additional upstream documentation or information may be found.

Mandatory

Yes

Default

- *CPACK_PACKAGE_HOMEPAGE_URL*, or if that is not set,
- *CPACK_DEBIAN_PACKAGE_HOMEPAGE* (this may be set already for Debian packaging, so it is used as a fallback).

Added in version 3.12: The **CPACK_PACKAGE_HOMEPAGE_URL** variable.

CPACK_FREEBSD_PACKAGE_LICENSE

The license, or licenses, which apply to this software package. This must be one or more license-identifiers that pkg recognizes as acceptable license identifiers (e.g. "GPLv2").

Mandatory

Yes

Default

- *CPACK_RPM_PACKAGE_LICENSE*

CPACK_FREEBSD_PACKAGE_LICENSE_LOGIC

This variable is only of importance if there is more than one license. The default is "single", which is only applicable to a single license. Other acceptable values are determined by pkg -- those are "dual" or "multi" -- meaning choice (OR) or simultaneous (AND) application of the licenses.

Mandatory

No

Default

single

CPACK_FREEBSD_PACKAGE_MAINTAINER

The FreeBSD maintainer (e.g. **kde@freebsd.org**) of this package.

Mandatory

Yes

Default

none

CPACK_FREEBSD_PACKAGE_ORIGIN

The origin (ports label) of this package; for packages built by CPack outside of the ports system this is of less importance. The default puts the package somewhere under **misc/**, as a stopgap.

Mandatory

Yes

Default

misc/<package name>

CPACK_FREEBSD_PACKAGE_CATEGORIES

The ports categories where this package lives (if it were to be built from ports). If none is set a single category is determined based on the package origin.

Mandatory

Yes

Default

derived from **ORIGIN**

CPACK_FREEBSD_PACKAGE_DEPS

A list of package origins that should be added as package dependencies. These are in the form <category>/<packagename>, e.g. **x11/libkonq**. No version information needs to be provided (this is not included in the manifest).

Mandatory

No

Default

empty

CPack Inno Setup Generator

Added in version 3.27.

Inno Setup is a free installer for Windows programs by Jordan Russell and Martijn Laan (-<https://jrsoftware.org/isinfo.php>).

This documentation explains Inno Setup generator specific options.

The generator provides a lot of options like components. Unfortunately, not all features (e.g. component dependencies) are currently supported by Inno Setup and they're ignored by the generator for now.

CPack requires Inno Setup 6 or greater.

Added in version 3.30: The generator is now available on non-Windows hosts, but requires Wine to run the Inno Setup tools.

Variables specific to CPack Inno Setup generator

You can use the following variables to change the behavior of the CPack **INNOSETUP** generator:

General

None of the following variables is required to be set for the Inno Setup generator to work. If a variable is marked as mandatory below but not set, its default value is taken.

The variables can also contain Inno Setup constants like **{app}**. Please refer to the documentation of Inno Setup for more information.

If you're asked to provide the path to any file, you can always give an absolute path or in most cases the relative path from the top-level directory where all files being installed by an *install()* instruction reside.

CPack tries to escape quotes and other special characters for you. However, using special characters could cause problems.

The following variable simplifies the usage of Inno Setup in CMake:

CPACK_INNOSETUP_USE_CMAKE_BOOL_FORMAT

Inno Setup only uses **yes** or **no** as boolean formats meanwhile CMake uses a lot of alternative formats like **ON** or **OFF**. Having this option turned on enables an automatic conversion.

Consider the following example:

```
set(CMAKE_INNOSETUP_SETUP_AllowNoIcons OFF)
```

If this option is turned on, the following line will be created in the output script: **AllowNoIcons=no**. Else, the following erroneous line will be created: **AllowNoIcons=OFF**

The conversion is enabled in every Inno Setup specific variable.

Mandatory

Yes

Default

ON

Setup Specific Variables

CPACK_INNOSETUP_ARCHITECTURE

One of **x86**, **x64**, **arm64** or **ia64**. This variable specifies the target architecture of the installer. This also affects the Program Files folder or registry keys being used.

CPack tries to determine the correct value with a try compile (see *CMAKE_SIZEOF_VOID_P*), but this option can be manually specified too (especially when using **ia64** or cross-platform compilation).

Mandatory

Yes

Default

Either **x86** or **x64** depending on the results of the try-compile

CPACK_INNOSETUP_INSTALL_ROOT

If you don't want the installer to create the installation directory under Program Files, you've to specify the installation root here.

The full directory of the installation will be: **\${CPACK_INNOSETUP_INSTALL_ROOT}/\${CPACK_PACKAGE_INSTALL_DIRECTORY}**.

Mandatory

Yes

Default

{autopf}

CPACK_INNOSETUP_ALLOW_CUSTOM_DIRECTORY

If turned on, the installer allows the user to change the installation directory providing an extra wizard page.

Mandatory

Yes

Default

ON

CPACK_INNOSETUP_PROGRAM_MENU_FOLDER

The initial name of the start menu folder being created.

If this variable is set to `.`, then no separate folder is created, application shortcuts will appear in the top-level start menu folder.

Mandatory

Yes

Default

The value of `CPACK_PACKAGE_NAME`

CPACK_INNOSETUP_LANGUAGES

A *semicolon-separated list* of languages you want Inno Setup to include.

Currently available: **armenian, brazilianPortuguese, bulgarian, catalan, corsican, czech, danish, dutch, english, finnish, french, german, hebrew, icelandic, italian, japanese, norwegian, polish, portuguese, russian, slovak, slovenian, spanish, turkish** and **ukrainian**. This list might differ depending on the version of Inno Setup.

Mandatory

Yes

Default

english

CPACK_INNOSETUP_IGNORE_LICENSE_PAGE

If you don't specify a license file using `CPACK_RESOURCE_FILE_LICENSE`, CPack uses a file for demonstration purposes. If you want the installer to ignore license files at all, you can enable this option.

Mandatory

Yes

Default

OFF

CPACK_INNOSETUP_IGNORE_README_PAGE

If you don't specify a readme file using `CPACK_RESOURCE_FILE_README`, CPack uses a file for demonstration purposes. If you want the installer to ignore readme files at all, you can enable this option. Make sure the option is disabled when using a custom readme file.

Mandatory

Yes

Default

ON

CPACK_INNOSETUP_PASSWORD

Enables password protection and file encryption with the given password.

Mandatory

No

CPACK_INNOSETUP_USE_MODERN_WIZARD

Enables the modern look and feel provided by Inno Setup. If this option is turned off, the classic style is used instead. Images and icon files are also affected.

Mandatory

Yes

Default

OFF because of compatibility reasons

CPACK_INNOSETUP_ICON_FILE

The path to a custom installer **.ico** file.

Use `CPACK_PACKAGE_ICON` to customize the bitmap file being shown in the wizard.

Mandatory

No

CPACK_INNOSETUP_SETUP_<directive>

This group allows adapting any of the **[Setup]** section directives provided by Inno Setup where **directive** is its name.

Here are some examples:

```
set(CPACK_INNOSETUP_SETUP_WizardSmallImageFile "my_bitmap.bmp")
set(CPACK_INNOSETUP_SETUP_AllowNoIcons OFF) # This requires CPACK_INNOSETUP.
```

All of these variables have higher priority than the others. Consider the following example:

```
set(CPACK_INNOSETUP_SETUP_Password "admin")
set(CPACK_INNOSETUP_PASSWORD "secret")
```

The password will be **admin** at the end because **CPACK_INNOSETUP_PASSWORD** has less priority than **CPACK_INNOSETUP_SETUP_Password**.

Mandatory

No

File Specific Variables

Although all files being installed by an *install()* instruction are automatically processed and added to the installer, there are some variables to customize the installation process.

Before using executables (only **.exe** or **.com**) in shortcuts (e.g. `CPACK_CREATE_DESKTOP_LINKS`) or **[Run]** entries, you've to add the raw file name (without path and extension) to `CPACK_PACKAGE_EXECUTABLES` and create a start menu shortcut for them.

If you have two files with the same raw name (e.g. **a/executable.exe** and **b/executable.com**), an entry in the section is created twice. This will result in undefined behavior and is not recommended.

CPACK_INNOSETUP_CUSTOM_INSTALL_INSTRUCTIONS

This variable should contain a *semicolon-separated list* of pairs **path**, **instruction** and can be used to customize the install command being automatically created for each file or directory.

CPack creates the following Inno Setup instruction for every file...

```
Source: "absolute\path\to\my_file.txt"; DestDir: "{app}"; Flags: ignorevers.
```

...and the following line for every directory:

```
Name: "{app}\my_folder"
```

You might want to change the destination directory or the flags of **my_file.txt**. Since we can also provide a relative path, the line you'd like to have, is the following:

```
Source: "my_file.txt"; DestDir: "{userdocs}"; Flags: ignoreversion uninsnev
```

You would do this by using **my_file.txt** as **path** and **Source: "my_file.txt"; DestDir:**

"{userdocs}"; Flags: ignoreversion uninsneveruninstall as instruction.

You've to take care of the *escaping problem*. So the CMake command would be:

```
set(CPACK_INNOSETUP_CUSTOM_INSTALL_INSTRUCTIONS "my_file.txt;Source: \\\"my_
```

To improve readability, you should go around the escaping problem by using `CPACK_VERBATIM_VARIABLES` or by placing the instruction into a separate CPack project config file.

If you customize the install instruction of a specific file, you lose the connection to its component. To go around, manually add **Components: <component>**. You also need to add its shortcuts and **[Run]** entries by yourself in a custom section, since the executable won't be found anymore by `CPACK_PACKAGE_EXECUTABLES`.

Here's another example (Note: You've to go around the escaping problem for the example to work):

```
set(CPACK_INNOSETUP_CUSTOM_INSTALL_INSTRUCTIONS
    "component1/my_folder" "Name: \"{userdocs}\\my_folder\\"; Components: c
    "component2/my_folder2/my_file.txt" "Source: \"component2\\my_folder2\\
```

Mandatory

No

CPACK_INNOSETUP_MENU_LINKS

This variable should contain a *semicolon-separated list* of pairs **link**, **link name** and can be used to add shortcuts into the start menu folder beside those of the executables (see `CPACK_PACKAGE_EXECUTABLES`). While **link name** is the label, **link** can be a URL or a path relative to the installation directory.

Here's an example:

```
set(CPACK_INNOSETUP_MENU_LINKS
    "doc/cmake-@CMake_VERSION_MAJOR@.@CMake_VERSION_MINOR@/cmake.html"
    "CMake Help" "https://cmake.org" "CMake Web Site")
```

Mandatory

No

CPACK_INNOSETUP_CREATE_UNINSTALL_LINK

If this option is turned on, a shortcut to the application's uninstaller is automatically added to the start menu folder.

Mandatory

Yes

Default

OFF

CPACK_INNOSETUP_RUN_EXECUTABLES

A *semicolon-separated list* of executables being specified in `CPACK_PACKAGE_EXECUTABLES` which the user can run when the installer finishes.

They're internally added to the **[Run]** section.

Mandatory

No

Components Specific Variables

The generator supports components and also downloaded components. However, there are some features of components that aren't supported yet (especially component dependencies). These variables are ignored for now.

CPack will change a component's name in Inno Setup if it has a parent group for technical reasons. Consider using **group\component** as component name in Inno Setup scripts if you have the component **component** and its parent group **group**.

Here are some additional variables for components:

CPACK_INNOSETUP_<compName>_INSTALL_DIRECTORY

If you don't want the component **compName** to be installed under **{app}**, you've to specify its installation directory here.

Mandatory

No

CPACK_INNOSETUP_VERIFY_DOWNLOADS

This option only affects downloaded components.

If this option is turned on, the hashes of the downloaded archives are calculated during compile and download time. The installer will only proceed if they match.

Mandatory

Yes

Default

ON

Compilation and Scripting Specific Variables

CPACK_INNOSETUP_EXECUTABLE

The filename of the Inno Setup Script Compiler command.

Mandatory

Yes

Default

ISCC

CPACK_INNOSETUP_EXECUTABLE_ARGUMENTS

A *semicolon-separated list* of extra command-line options for the Inno Setup Script Compiler command.

For example: **/Qp;/Smysigntool=\$p**

Take care of the *escaping problem*.

Mandatory

No

CPACK_INNOSETUP_DEFINE_<macro>

This group allows to add custom define directives as command-line options to the Inno Setup Preprocessor command. Each entry emulates a **#define public <macro>** directive. Its macro is accessible from anywhere (**public**), so it can also be used in extra script files.

Macro names must not contain any special characters. Refer to the Inno Setup Preprocessor documentation for the detailed rules.

Consider the following example:

```
# The following line emulates: #define public MyMacro "Hello, World!"
set(CPACK_INNOSETUP_DEFINE_MyMacro "Hello, World!")
```

At this point, you can use **MyMacro** anywhere. For example in the following extra script:

```
AppComments={#emit "'My Macro' has the value: " + MyMacro}
```

Take care of the *escaping problem*.

Mandatory

No

CPACK_INNOSETUP_EXTRA_SCRIPTS

A *semicolon-separated list* of paths to additional **.iss** script files to be processed.

They're internally included at the top of the output script file using a **#include** directive.

You can add any section in your file to extend the installer (e.g. adding additional tasks or registry keys). Prefer using `CPACK_INNOSETUP_SETUP_<directive>` when extending the **[Setup]** section.

Mandatory

No

CPACK_INNOSETUP_CODE_FILES

A *semicolon-separated list* of paths to additional Pascal files to be processed.

This variable is actually the same as `CPACK_INNOSETUP_EXTRA_SCRIPTS`, except you don't have to add **[Code]** at the top of your file. Never change the current section in a code file. This will result in undefined behavior! Treat them as normal Pascal scripts instead.

Code files are included at the very bottom of the output script.

Mandatory

No

CPack IFW Generator

Added in version 3.1.

Configure and run the Qt Installer Framework to generate a Qt installer.

Overview

This *cpack generator* generates configuration and meta information for the *Qt Installer Framework* (QtIFW), and runs QtIFW tools to generate a Qt installer.

QtIFW provides tools and utilities to create installers for the platforms supported by Qt: Linux, Microsoft Windows, and macOS.

To make use of this generator, QtIFW needs to be installed. The *CPackIFW* module looks for the location of the QtIFW command-line utilities, and defines several commands to control the behavior of this generator. See *Hints for Finding QtIFW*.

Variables

You can use the following variables to change the behavior of the CPack **IFW** generator.

Debug

CPACK_IFW_VERBOSE

Added in version 3.3.

Set to **ON** to enable addition debug output. By default is **OFF**.

Package

CPACK_IFW_PACKAGE_TITLE

Name of the installer as displayed on the title bar. If not specified, it defaults to *CPACK_PACKAGE_DESCRIPTION_SUMMARY*.

CPACK_IFW_PACKAGE_PUBLISHER

Publisher of the software (as shown in the Windows Control Panel). If not specified, it defaults to *CPACK_PACKAGE_VENDOR*.

CPACK_IFW_PRODUCT_URL

URL to a page that contains product information on your web site.

CPACK_IFW_PACKAGE_ICON

Filename for a custom installer icon. It must be an absolute path. This should be a **.icns** file on macOS and a **.ico** file on Windows. It is ignored on other platforms.

CPACK_IFW_PACKAGE_WINDOW_ICON

Filename for a custom window icon in PNG format for the Installer application. It must be an absolute path.

CPACK_IFW_PACKAGE_LOGO

Filename for a logo image in PNG format, used as **QWizard::LogoPixmap**. It must be an absolute path.

CPACK_IFW_PACKAGE_WATERMARK

Added in version 3.8.

Filename for a watermark image in PNG format, used as **QWizard::WatermarkPixmap**. It must be an absolute path.

CPACK_IFW_PACKAGE_BANNER

Added in version 3.8.

Filename for a banner image in PNG format, used as **QWizard::BannerPixmap**. It must be an absolute path.

CPACK_IFW_PACKAGE_BACKGROUND

Added in version 3.8.

Filename for a background image in PNG format, used as **QWizard::BackgroundPixmap** (only used by **MacStyle**). It must be an absolute path.

CPACK_IFW_PACKAGE_WIZARD_STYLE

Added in version 3.8.

Wizard style to be used (**Modern**, **Mac**, **Aero** or **Classic**).

CPACK_IFW_PACKAGE_WIZARD_DEFAULT_WIDTH

Added in version 3.8.

Default width of the wizard in pixels. Setting a banner image will override this.

CPACK_IFW_PACKAGE_WIZARD_DEFAULT_HEIGHT

Added in version 3.8.

Default height of the wizard in pixels. Setting a watermark image will override this.

CPACK_IFW_PACKAGE_WIZARD_SHOW_PAGE_LIST

Added in version 3.20.

Set to **OFF** if the widget listing installer pages on the left side of the wizard should not be shown.

It is **ON** by default, but will only have an effect if using QtIFW 4.0 or later.

CPACK_IFW_PACKAGE_TITLE_COLOR

Added in version 3.8.

Color of the titles and subtitles (takes an HTML color code, such as **#88FF33**).

CPACK_IFW_PACKAGE_STYLE_SHEET

Added in version 3.15.

Filename for a stylesheet. It must be an absolute path.

CPACK_IFW_TARGET_DIRECTORY

Default target directory for installation. If *CPACK_PACKAGE_INSTALL_DIRECTORY* is set, this defaults to **@ApplicationsDir@/{CPACK_PACKAGE_INSTALL_DIRECTORY}**. If that variable isn't set either, the default used is **@RootDir@/usr/local**. Predefined variables of the form **@...@** are expanded by the *QtIFW scripting engine*.

CPACK_IFW_ADMIN_TARGET_DIRECTORY

Default target directory for installation with administrator rights.

You can use predefined variables.

CPACK_IFW_PACKAGE_REMOVE_TARGET_DIR

Added in version 3.11.

Set to **OFF** if the target directory should not be deleted when uninstalling.

Is **ON** by default

CPACK_IFW_PACKAGE_GROUP

The group, which will be used to configure the root package.

CPACK_IFW_PACKAGE_NAME

The root package name, which will be used if the configuration group is not specified.

CPACK_IFW_PACKAGE_START_MENU_DIRECTORY

Added in version 3.3.

Name of the default program group for the product in the Windows Start menu. If not specified, it defaults to *CPACK_IFW_PACKAGE_NAME*.

CPACK_IFW_PACKAGE_MAINTENANCE_TOOL_NAME

Added in version 3.3.

Filename of the generated maintenance tool. The platform-specific executable file extension will be appended.

If not specified, QtIFW provides a default name (**maintenancetool**).

CPACK_IFW_PACKAGE_MAINTENANCE_TOOL_INI_FILE

Added in version 3.3.

Filename for the configuration of the generated maintenance tool.

If not specified, QtIFW uses a default file name (**maintenancetool.ini**).

CPACK_IFW_PACKAGE_ALLOW_NON_ASCII_CHARACTERS

Added in version 3.3.

Set to **ON** if the installation path can contain non-ASCII characters. Only supported for QtIFW 2.0 and later. Older QtIFW versions will always allow non-ASCII characters.

CPACK_IFW_PACKAGE_ALLOW_SPACE_IN_PATH

Added in version 3.3.

Set to **OFF** if the installation path cannot contain space characters.

Is **ON** for QtIFW less 2.0 tools.

CPACK_IFW_PACKAGE_DISABLE_COMMAND_LINE_INTERFACE

Added in version 3.23.

Set to **ON** if command line interface features should be disabled. It is **OFF** by default and will only have an effect if using QtIFW 4.0 or later.

CPACK_IFW_PACKAGE_CONTROL_SCRIPT

Added in version 3.3.

Filename for a custom installer control script.

CPACK_IFW_PACKAGE_RESOURCES

Added in version 3.7.

List of additional resources (**.qrc** files) to include in the installer binary. They should be specified as absolute paths and no two resource files can have the same file name.

You can use the `cpack_ifw_add_package_resources()` command to resolve relative paths.

CPACK_IFW_PACKAGE_FILE_EXTENSION

Added in version 3.10.

The target binary extension.

On Linux, the name of the target binary is automatically extended with **.run**, if you do not specify the extension.

On Windows, the target is created as an application with the extension **.exe**, which is automatically added, if not supplied.

On Mac, the target is created as an DMG disk image with the extension **.dmg**, which is automatically added, if not supplied.

CPACK_IFW_REPOSITORIES_ALL

The list of remote repositories.

The default value of this variable is computed by CPack and contains all repositories added with *cpack_ifw_add_repository()* or updated with *cpack_ifw_update_repository()*.

CPACK_IFW_DOWNLOAD_ALL

If this is **ON**, all components will be downloaded. If not set, the behavior is determined by whether *cpack_configure_downloads()* has been called with the **ALL** option or not.

CPACK_IFW_PACKAGE_PRODUCT_IMAGES

Added in version 3.23.

A list of images to be shown on the **PerformInstallationPage**. These must be absolute paths and the images must be in PNG format.

This feature is available for QtIFW 4.0.0 and later.

CPACK_IFW_PACKAGE_PRODUCT_IMAGE_URLS

Added in version 3.31.

A list of URLs associated with the ProductImages. Only used if **CPACK_IFW_PACKAGE_PRODUCT_IMAGES** is defined and it has the same size.

This feature is available for QtIFW 4.0.0 and later.

CPACK_IFW_PACKAGE_RUN_PROGRAM

Added in version 3.23.

Command executed after the installer is finished, if the user accepts the action. Provide the full path to the application, as found when installed. This typically means the path should begin with the QtIFW predefined variable **@TargetDir@**.

This feature is available for QtIFW 4.0.0 and later.

CPACK_IFW_PACKAGE_RUN_PROGRAM_ARGUMENTS

Added in version 3.23.

List of arguments passed to the program specified in *CPACK_IFW_PACKAGE_RUN_PROGRAM*.

This feature is available for QtIFW 4.0.0 and later.

CPACK_IFW_PACKAGE_RUN_PROGRAM_DESCRIPTION

Added in version 3.23.

Text shown next to the check box for running the program after the installation. If *CPACK_IFW_PACKAGE_RUN_PROGRAM* is set but no description is provided, QtIFW will use a default message like **Run <Name> now**.

This feature is available for QtIFW 4.0.0 and later.

CPACK_IFW_PACKAGE_SIGNING_IDENTITY

Added in version 3.23.

Allows specifying a code signing identity to be used for signing the generated app bundle. Only available on macOS, ignored on other platforms.

CPACK_IFW_ARCHIVE_FORMAT

Added in version 3.23.

Set the format used when packaging new component data archives. If you omit this option, the **7z** format will be used as a default. Supported formats:

- 7z
- zip
- tar.gz
- tar.bz2
- tar.xz

NOTE:

If the Qt Installer Framework tools were built without libarchive support, only **7z** format is supported.

This feature is available for QtIFW 4.2.0 and later.

CPACK_IFW_ARCHIVE_COMPRESSION

Added in version 3.23.

Archive compression level. The allowable values are:

- 0 (*No compression*)
- 1 (*Fastest compression*)
- 3 (*Fast compression*)
- 5 (*Normal compression*)
- 7 (*Maximum compression*)
- 9 (*Ultra compression*)

If this variable is not set, QtIFW will use a default compression level, which will typically be 5 (*Normal compression*).

NOTE:

Some formats do not support all the possible values. For example **zip** compression only supports values from 1 to 7.

This feature is available for QtIFW 4.2.0 and later.

Components**CPACK_IFW_RESOLVE_DUPLICATE_NAMES**

Resolve duplicate names when installing components with groups.

CPACK_IFW_PACKAGES_DIRECTORIES

Additional prepared packages directories that will be used to resolve dependent components.

CPACK_IFW_REPOSITORIES_DIRECTORIES

Added in version 3.10.

Additional prepared repository directories that will be used to resolve and repack dependent components.

This feature is available for QtIFW 3.1 and later.

QtIFW Tools**CPACK_IFW_FRAMEWORK_VERSION**

Added in version 3.3.

The version of the QtIFW tools that will be used. This variable is set by the *CPackIFW* module.

The following variables provide the locations of the QtIFW command-line tools as discovered by the *CPackIFW* module. These variables are cached, and may be configured if needed.

CPACK_IFW_ARCHIVEGEN_EXECUTABLE

Added in version 3.19.

The path to **archivegen**.

CPACK_IFW_BINARYCREATOR_EXECUTABLE

The path to **binarycreator**.

CPACK_IFW_REPOGEN_EXECUTABLE

The path to **repogen**.

CPACK_IFW_INSTALLERBASE_EXECUTABLE

The path to **installerbase**.

CPACK_IFW_DEVTOOL_EXECUTABLE

The path to **devtool**.

Hints for Finding QtIFW

Generally, the CPack **IFW** generator automatically finds QtIFW tools. The following (in order of precedence) can also be set to augment the locations normally searched by *find_program()*:

CPACK_IFW_ROOT

Added in version 3.9.

CMake variable

CPACK_IFW_ROOT

Added in version 3.9.

Environment variable

QTIFWDIR

CMake variable

QTIFWDIR

Environment variable

NOTE:

The specified path should not contain **bin** at the end (for example: **D:\\DevTools\\QtIFW2.0.5**).

Other Settings**Online installer**

By default, this generator generates an *offline installer*. This means that all packaged files are fully contained in the installer executable.

In contrast, an *online installer* will download some or all components from a remote server.

The **DOWNLOADED** option in the `cpack_add_component()` command specifies that a component is to be downloaded. Alternatively, the **ALL** option in the `cpack_configure_downloads()` command specifies that **all** components are to be downloaded.

The `cpack_ifw_add_repository()` command and the `CPACK_IFW_DOWNLOAD_ALL` variable allow for more specific configuration.

When there are online components, CPack will write them to archive files. The help page of the `CPackComponent` module, especially the section on the `cpack_configure_downloads()` function, explains how to make these files accessible from a download URL.

Internationalization

Added in version 3.9.

Some variables and command arguments support internationalization via CMake script. This is an optional feature.

Installers created by QtIFW tools have built-in support for internationalization and many phrases are localized to many languages, but this does not apply to the description of your components and groups.

Localization of the description of your components and groups is useful for users of your installers.

A localized variable or argument can contain a single default value, and after that a set of pairs with the name of the locale and the localized value.

For example:

```
set(LOCALIZABLE_VARIABLE "Default value"
    en "English value"
    en_US "American value"
    en_GB "Great Britain value"
)
```

See Also

Qt Installer Framework Manual:

- Index page: <https://doc.qt.io/qtinstallerframework/index.html>
- Component Scripting: <https://doc.qt.io/qtinstallerframework/scripting.html>
- Predefined Variables: <https://doc.qt.io/qtinstallerframework/scripting.html#predefined-variables>
- Promoting Updates: <https://doc.qt.io/qtinstallerframework/ifw-updates.html>

Download Qt Installer Framework for your platform from Qt site:

https://download.qt.io/official_releases/qt-installer-framework

CPack NSIS Generator

CPack Nullsoft Scriptable Install System (NSIS) generator specific options.

Changed in version 3.22: The NSIS generator requires NSIS 3.03 or newer.

Variables specific to CPack NSIS generator

The following variables are specific to the graphical installers built on Windows Nullsoft Scriptable Install System.

CPACK_NSIS_INSTALL_ROOT

The default installation directory presented to the end user by the NSIS installer is under this root dir. The full directory presented to the end user is: `${CPACK_NSIS_INSTALL_ROOT}/${CPACK_PACKAGE_INSTALL_DIRECTORY}`

CPACK_NSIS_MUI_ICON

An icon filename. The name of a `*.ico` file used as the main icon for the generated install program.

CPACK_NSIS_MUI_UNIICON

An icon filename. The name of a `*.ico` file used as the main icon for the generated uninstall program.

CPACK_NSIS_INSTALLER_MUI_ICON_CODE

undocumented.

CPACK_NSIS_MUI_WELCOMEFINISHPAGE_BITMAP

Added in version 3.5.

The filename of a bitmap to use as the NSIS **MUI_WELCOMEFINISHPAGE_BITMAP**.

CPACK_NSIS_MUI_UNWELCOMEFINISHPAGE_BITMAP

Added in version 3.5.

The filename of a bitmap to use as the NSIS **MUI_UNWELCOMEFINISHPAGE_BITMAP**.

CPACK_NSIS_EXTRA_PREINSTALL_COMMANDS

Extra NSIS commands that will be added to the beginning of the install Section, before your install tree is available on the target system.

CPACK_NSIS_EXTRA_INSTALL_COMMANDS

Extra NSIS commands that will be added to the end of the install Section, after your install tree is available on the target system.

CPACK_NSIS_EXTRA_UNINSTALL_COMMANDS

Extra NSIS commands that will be added to the uninstall Section, before your install tree is removed from the target system.

CPACK_NSIS_COMPRESSOR

The arguments that will be passed to the NSIS **SetCompressor** command.

CPACK_NSIS_ENABLE_UNINSTALL_BEFORE_INSTALL

Ask about uninstalling previous versions first. If this is set to **ON**, then an installer will look for previous installed versions and if one is found, ask the user whether to uninstall it before proceeding with the install.

CPACK_NSIS_MODIFY_PATH

Modify **PATH** toggle. If this is set to **ON**, then an extra page will appear in the installer that will allow the user to choose whether the program directory should be added to the system **PATH** variable.

CPACK_NSIS_DISPLAY_NAME

The display name string that appears in the Windows *Apps & features* in *Control Panel*

CPACK_NSIS_PACKAGE_NAME

The title displayed at the top of the installer.

CPACK_NSIS_INSTALLED_ICON_NAME

A path to the executable that contains the installer icon.

CPACK_NSIS_HELP_LINK

URL to a web site providing assistance in installing your application.

CPACK_NSIS_URL_INFO_ABOUT

URL to a web site providing more information about your application.

CPACK_NSIS_CONTACT

Contact information for questions and comments about the installation process.

CPACK_NSIS_<compName>_INSTALL_DIRECTORY

Added in version 3.7.

Custom install directory for the specified component <compName> instead of **\$INSTDIR**.

CPACK_NSIS_CREATE_ICONS_EXTRA

Additional NSIS commands for creating *Start Menu* shortcuts.

CPACK_NSIS_DELETE_ICONS_EXTRA

Additional NSIS commands to uninstall *Start Menu* shortcuts.

CPACK_NSIS_EXECUTABLES_DIRECTORY

Creating NSIS *Start Menu* links assumes that they are in **bin** unless this variable is set. For example, you would set this to **exec** if your executables are in an **exec** directory.

CPACK_NSIS_MUI_FINISHPAGE_RUN

Specify an executable to add an option to run on the finish page of the NSIS installer.

CPACK_NSIS_MENU_LINKS

Specify links in **[application]** menu. This should contain a list of pair **link link name**. The link may be a URL or a path relative to installation prefix. Like:

```
set(CPACK_NSIS_MENU_LINKS
    "doc/cmake-@CMake_VERSION_MAJOR@.@CMake_VERSION_MINOR@/cmake.html"
    "CMake Help" "https://cmake.org" "CMake Web Site")
```

CPACK_NSIS_UNINSTALL_NAME

Added in version 3.17.

Specify the name of the program to uninstall the version. Default is **Uninstall**.

CPACK_NSIS_WELCOME_TITLE

Added in version 3.17.

The title to display on the top of the page for the welcome page.

CPACK_NSIS_WELCOME_TITLE_3LINES

Added in version 3.17.

Display the title in the welcome page on 3 lines instead of 2.

CPACK_NSIS_FINISH_TITLE

Added in version 3.17.

The title to display on the top of the page for the finish page.

CPACK_NSIS_FINISH_TITLE_3LINES

Added in version 3.17.

Display the title in the finish page on 3 lines instead of 2.

CPACK_NSIS_MUI_HEADERIMAGE

Added in version 3.17.

The image to display on the header of installers pages.

CPACK_NSIS_MANIFEST_DPI_AWARE

Added in version 3.18.

If set, declares that the installer is DPI-aware.

CPACK_NSIS_BRANDING_TEXT

Added in version 3.20.

If set, updates the text at the bottom of the install window. To set the string to blank, use a space (" ").

CPACK_NSIS_BRANDING_TEXT_TRIM_POSITION

Added in version 3.20.

If set, trim down the size of the control to the size of the branding text string. Allowed values for this variable are **LEFT**, **CENTER** or **RIGHT**. If not specified, the default behavior is **LEFT**.

CPACK_NSIS_EXECUTABLE

Added in version 3.21.

If set, specify the name of the NSIS executable. Default is **makensis**.

CPACK_NSIS_IGNORE_LICENSE_PAGE

Added in version 3.22.

If set, do not display the page containing the license during installation.

CPACK_NSIS_EXECUTABLE_PRE_ARGUMENTS

Added in version 3.25.

This variable is a *semicolon-separated list* of arguments to prepend to the nsis script to run. If the arguments do not start with a / or a -, it will add one automatically to the corresponding arguments. The command that will be run is:

```
makensis.exe <preArgs>... "nsisFileName.nsi" <postArgs>...
```

where **<preArgs>...** is constructed from **CPACK_NSIS_EXECUTABLE_PRE_ARGUMENTS** and **<postArgs>...** is constructed from **CPACK_NSIS_EXECUTABLE_POST_ARGUMENTS**.

CPACK_NSIS_EXECUTABLE_POST_ARGUMENTS

Added in version 3.25.

This variable is a *semicolon-separated list* of arguments to append to the nsis script to run. If the arguments do not start with a / or a -, it will add one automatically to the corresponding arguments. The command that will be run is:

```
makensis.exe <preArgs>... "nsisFileName.nsi" <postArgs>...
```

where **<preArgs>...** is constructed from **CPACK_NSIS_EXECUTABLE_PRE_ARGUMENTS** and **<postArgs>...** is constructed from **CPACK_NSIS_EXECUTABLE_POST_ARGUMENTS**.

CPack NuGet Generator

Added in version 3.12.

When build a NuGet package there is no direct way to control an output filename due a lack of the corresponding CLI option of NuGet, so there is no **CPACK_NUGET_PACKAGE_FILE_NAME** variable. To form the output filename NuGet uses the package name and the version according to its built-in rules.

Also, be aware that including a top level directory (**CPACK_INCLUDE_TOPLEVEL_DIRECTORY**) is ignored by this generator.

Variables specific to CPack NuGet generator

The CPack NuGet generator may be used to create NuGet packages using *CPack*. The CPack NuGet generator is a *CPack* generator thus it uses the **CPACK_XXX** variables used by *CPack*.

The CPack NuGet generator has specific features which are controlled by the specifics **CPACK_NUGET_XXX** variables. In the "one per group" mode (see **CPACK_COMPONENTS_GROUPING**), **<compName>** placeholder in the variables below would contain a group name (uppercased and turned into a "C" identifier).

List of CPack NuGet generator specific variables:

CPACK_NUGET_COMPONENT_INSTALL

Enable component packaging for CPack NuGet generator

Mandatory

No

Default

OFF

CPACK_NUGET_PACKAGE_DEBUG

Enable debug messages while executing CPack NuGet generator.

Mandatory

No

Default

OFF

Required metadata variables

CPACK_NUGET_PACKAGE_NAME

CPACK_NUGET_<compName>_PACKAGE_NAME

The NUGET package name. **CPACK_NUGET_PACKAGE_NAME** is used as the package **id** on *nuget.org*

Mandatory

Yes

Default

CPACK_PACKAGE_NAME

CPACK_NUGET_PACKAGE_VERSION**CPACK_NUGET_<compName>_PACKAGE_VERSION**

The NuGet package version.

Mandatory

Yes

Default

CPACK_PACKAGE_VERSION

CPACK_NUGET_PACKAGE_DESCRIPTION**CPACK_NUGET_<compName>_PACKAGE_DESCRIPTION**

A long description of the package for UI display.

Mandatory

Yes

Default

- *CPACK_COMPONENT_<compName>_DESCRIPTION*,
- *CPACK_COMPONENT_GROUP_<groupName>_DESCRIPTION*,
- *CPACK_PACKAGE_DESCRIPTION*

CPACK_NUGET_PACKAGE_AUTHORS**CPACK_NUGET_<compName>_PACKAGE_AUTHORS**

A comma-separated list of packages authors, matching the profile names on *nuget.org*. These are displayed in the NuGet Gallery on *nuget.org* and are used to cross-reference packages by the same authors.

Mandatory

Yes

Default

CPACK_PACKAGE_VENDOR

Optional metadata variables**CPACK_NUGET_PACKAGE_OWNERS****CPACK_NUGET_<compName>_PACKAGE_OWNERS**

Deprecated since version 3.30: Use authors (*CPACK_NUGET_PACKAGE_AUTHORS*) instead.

A comma-separated list of the package creators using profile names on *nuget.org*. This is often the same list as in authors, and is ignored when uploading the package to *nuget.org*.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_HOMEPAGE_URL**CPACK_NUGET_<compName>_PACKAGE_HOMEPAGE_URL**

An URL for the package's home page, often shown in UI displays as well as *nuget.org*.

Mandatory

No

Default

CPACK_PACKAGE_HOMEPAGE_URL

CPACK_NUGET_PACKAGE_LICENSEURL**CPACK_NUGET_<compName>_PACKAGE_LICENSEURL**

Deprecated since version 3.20: Use a local license file (- *CPACK_NUGET_PACKAGE_LICENSE_FILE_NAME*) or a *SPDX* license identifier (- *CPACK_NUGET_PACKAGE_LICENSE_EXPRESSION*) instead.

An URL for the package's license, often shown in UI displays as well as on *nuget.org*.

Mandatory

No

Default

None

Supported

NuGet 4.9.0 and above

CPACK_NUGET_PACKAGE_LICENSE_FILE_NAME**CPACK_NUGET_<compName>_PACKAGE_LICENSE_FILE_NAME**

Added in version 3.20.

The package's license file in **.txt** or **.md** format.

If **CPACK_NUGET_PACKAGE_LICENSE_FILE_NAME** is specified, **CPACK_NUGET_PACKAGE_LICENSE_EXPRESSION** is ignored.

Mandatory

No

Default

None

Supported

NuGet 4.9.0 and above

CPACK_NUGET_PACKAGE_LICENSE_EXPRESSION**CPACK_NUGET_<compName>_PACKAGE_LICENSE_EXPRESSION**

Added in version 3.20.

A Software Package Data Exchange *SPDX* license identifier such as **MIT**, **BSD-3-Clause**, or **LGPL-3.0-or-later**. In the case of a choice of licenses or more complex restrictions, compound license expressions may be formed using boolean operators, for example **MIT OR BSD-3-Clause**. See the *SPDX specification* for guidance on forming complex license expressions.

If **CPACK_NUGET_PACKAGE_LICENSE_FILE_NAME** is specified,

CPACK_NUGET_PACKAGE_LICENSE_EXPRESSION is ignored.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_ICONURL

CPACK_NUGET_<compName>_PACKAGE_ICONURL

Deprecated since version 3.20: Use a local icon file (*CPACK_NUGET_PACKAGE_ICON*) instead.

An URL for a 64x64 image with transparency background to use as the icon for the package in UI display.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_ICON

CPACK_NUGET_<compName>_PACKAGE_ICON

Added in version 3.20.

The filename of a 64x64 image with transparency background to use as the icon for the package in UI display.

Mandatory

No

Default

None

Supported

NuGet 5.3.0 and above

CPACK_NUGET_PACKAGE_README

CPACK_NUGET_<compName>_PACKAGE_README

Added in version 3.30.

The package path relative to the root of the package to a readme file. Supported file formats include only Markdown (*.md).

Mandatory

No

Default

None

Supported

NuGet 5.10.0 preview 2 and above

CPACK_NUGET_PACKAGE_REQUIRE_LICENSE_ACCEPTANCE

When set to a true value, the user will be prompted to accept the license before installing the package.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_DESCRIPTION_SUMMARY**CPACK_NUGET_<compName>_PACKAGE_DESCRIPTION_SUMMARY**

Deprecated since version 3.30: Summary is being deprecated. Use description (-CPACK_NUGET_PACKAGE_DESCRIPTION) instead.

A short description of the package for UI display. If omitted, a truncated version of description is used.

Mandatory

No

Default*CPACK_PACKAGE_DESCRIPTION_SUMMARY***CPACK_NUGET_PACKAGE_RELEASE_NOTES****CPACK_NUGET_<compName>_PACKAGE_RELEASE_NOTES**

A description of the changes made in this release of the package, often used in UI like the Updates tab of the Visual Studio Package Manager in place of the package description.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_COPYRIGHT**CPACK_NUGET_<compName>_PACKAGE_COPYRIGHT**

Copyright details for the package.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_LANGUAGE**CPACK_NUGET_<compName>_PACKAGE_LANGUAGE**

Added in version 3.20.

Locale specifier for the package, for example **en_CA**.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_TAGS**CPACK_NUGET_<compName>_PACKAGE_TAGS**

A space-delimited list of tags and keywords that describe the package and aid discoverability of packages through search and filtering.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_REPOSITORY_URL**CPACK_NUGET_<compName>_REPOSITORY_URL**

Added in version 3.30.

Repository metadata allows you to map the ***.nupkg** to the repository that built it. This should be a publicly available URL that can be invoked directly by a version control software. It should not be an HTML page as this is meant for the computer.

Mandatory

No

Default

None

Supported

NuGet 4.0 and above

CPACK_NUGET_PACKAGE_REPOSITORY_TYPE**CPACK_NUGET_<compName>_REPOSITORY_TYPE**

Added in version 3.30.

A type of the VCS repository. When uploading a package *tonug et.org*, the type is limited to 100 characters.

Mandatory

Yes, if repository URL has been specified

Default

None

Supported

NuGet 4.0 and above

CPACK_NUGET_PACKAGE_REPOSITORY_BRANCH**CPACK_NUGET_<compName>_REPOSITORY_BRANCH**

Added in version 3.30.

A VSC branch name to build the package.

Mandatory

No

Default

None

Supported

NuGet 4.6 and above

CPACK_NUGET_PACKAGE_REPOSITORY_COMMIT**CPACK_NUGET_<compName>_REPOSITORY_COMMIT**

Added in version 3.30.

A SHA-1 hash of the commit to build the package.

Mandatory

No

Default

None

Supported

NuGet 4.6 and above

CPACK_NUGET_PACKAGE_TITLE

CPACK_NUGET_<compName>_PACKAGE_TITLE

A human-friendly title of the package, typically used in UI displays as on *nuget.org* and the Package Manager in Visual Studio. If not specified, the package ID is used.

Mandatory

No

Default

- *CPACK_COMPONENT_<compName>_DISPLAY_NAME*,
- *CPACK_COMPONENT_GROUP_<groupName>_DISPLAY_NAME*

Dependencies specification

CPACK_NUGET_PACKAGE_DEPENDENCIES

CPACK_NUGET_<compName>_PACKAGE_DEPENDENCIES

A list of default (not framework-specific) package dependencies.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_DEPENDENCIES_<dependency>_VERSION

CPACK_NUGET_<compName>_PACKAGE_DEPENDENCIES_<dependency>_VERSION

A *version specification* for the particular dependency, where <dependency> is an item of the dependency list (see above).

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_TFMS

CPACK_NUGET_<compName>_PACKAGE_TFMS

Added in version 3.30.

A list of Target Framework Monikers (TFMs) for the package, e.g., "net47;netcoreapp21". For each of these TFMs a *dependency group* will be generated in the dependencies block of the NuGet package. Framework-specific dependencies can be added to these groups with the TFM dependency lists (see below).

This variable is particularly useful for fixing warnings *NU5128*.

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_DEPENDENCIES_<tfm>**CPACK_NUGET_<compName>_PACKAGE_DEPENDENCIES_<tfm>**

Added in version 3.30.

A list of package dependencies that apply specifically to the <tfm> framework, where <tfm> is an item from the TFMs list (see above).

Mandatory

No

Default

None

CPACK_NUGET_PACKAGE_DEPENDENCIES_<tfm>_<dependency>_VERSION**CPACK_NUGET_<compName>_PACKAGE_DEPENDENCIES_<tfm>_<dependency>_VERSION**

Added in version 3.30.

A *version specification* for the particular framework-specific dependency, where <dependency> is an item of the <tfm>-specific dependency list (see above).

Mandatory

No

Default

None

Example usage

```

set(CPACK_GENERATOR NuGet)
# Set up package metadata
set(CPACK_PACKAGE_NAME SamplePackage)
set(CPACK_PACKAGE_VERSION "1.0.0")
set(CPACK_PACKAGE_VENDOR "Example Inc")
set(CPACK_NUGET_PACKAGE_AUTHORS "ExampleInc")
set(CPACK_PACKAGE_DESCRIPTION "A .NET wrapper around the foobar library for frobb")
set(CPACK_PACKAGE_HOMEPAGE_URL "https://www.example.com")
set(CPACK_NUGET_PACKAGE_REPOSITORY_URL "https://github.com/example/libfoobar.git")
set(CPACK_NUGET_PACKAGE_REPOSITORY_TYPE git)
set(CPACK_NUGET_PACKAGE_LICENSE_EXPRESSION "MIT")
# Set up dependencies
set(CPACK_NUGET_PACKAGE_TFMS "net4;net6.0")
set(CPACK_NUGET_PACKAGE_DEPENDENCIES_net4 "Foo;Bar")
# NB: If a version number is omitted, the dependency will not be created
set(CPACK_NUGET_PACKAGE_DEPENDENCIES_net4_Foo_VERSION "1.23")
set(CPACK_NUGET_PACKAGE_DEPENDENCIES_net4_Bar_VERSION "4.3.2")
# NB: General dependencies (not framework-specific) go in this variable
set(CPACK_NUGET_PACKAGE_DEPENDENCIES "Baz")
set(CPACK_NUGET_PACKAGE_DEPENDENCIES_Baz_VERSION "9.8.6")
# NB: Since "net6.0" was listed but no dependencies have been specified, an empty
# will be added to the nuspec file for this framework. This can be used to address
include(CPack)

```

CPack PackageMaker Generator

Removed. This once generated PackageMaker installers, but the generator has been removed since CMake 3.24. Xcode no longer distributes the PackageMaker tools. Use the *CPack productbuild Generator* instead.

CPack productbuild Generator

Added in version 3.7.

productbuild CPack generator (macOS).

Variables specific to CPack productbuild generator

The following variable is specific to installers built on Mac macOS using ProductBuild:

CPACK_COMMAND_PRODUCTBUILD

Path to the **productbuild(1)** command used to generate a product archive for the macOS Installer or Mac App Store. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it).

CPACK_PRODUCTBUILD_IDENTIFIER

Added in version 3.23.

Set the unique (non-localized) product identifier to be associated with the product (i.e., **com.kitware.cmake**). Any component product names will be appended to this value.

CPACK_PRODUCTBUILD_IDENTITY_NAME

Added in version 3.8.

Adds a digital signature to the resulting package.

CPACK_PRODUCTBUILD_KEYCHAIN_PATH

Added in version 3.8.

Specify a specific keychain to search for the signing identity.

CPACK_COMMAND_PKGBUILD

Path to the **pkgbuild(1)** command used to generate an macOS component package on macOS. This variable can be used to override the automatically detected command (or specify its location if the auto-detection fails to find it).

CPACK_PKGBUILD_IDENTITY_NAME

Added in version 3.8.

Adds a digital signature to the resulting package.

CPACK_PKGBUILD_KEYCHAIN_PATH

Added in version 3.8.

Specify a specific keychain to search for the signing identity.

CPACK_PREFLIGHT_<COMP>_SCRIPT

Full path to a file that will be used as the **preinstall** script for the named <COMP> component's package, where <COMP> is the uppercased component name. No **preinstall** script is added if this variable is not defined for a given component.

CPACK_POSTFLIGHT_<COMP>_SCRIPT

Full path to a file that will be used as the **postinstall** script for the named <COMP> component's package, where <COMP> is the uppercased component name. No **postinstall** script is added if this variable is not defined for a given component.

CPACK_PRODUCTBUILD_RESOURCES_DIR

Added in version 3.9.

If specified the productbuild generator copies files from this directory (including subdirectories) to the **Resources** directory. This is done before the *CPACK_RESOURCE_FILE_WELCOME*, *CPACK_RESOURCE_FILE_README*, and *CPACK_RESOURCE_FILE_LICENSE* files are copied.

CPACK_PRODUCTBUILD_DOMAINS

Added in version 3.23.

This option enables more granular control over where the product may be installed. When it is set to true (see policy *CMP0161*), a **domains** element of the following form will be added to the productbuild Distribution XML:

```
<domains enable_anywhere="true" enable_currentUserHome="false" enable_local="true">
```

The default values are as shown above, but can be overridden with *CPACK_PRODUCTBUILD_DOMAINS_ANYWHERE*, *CPACK_PRODUCTBUILD_DOMAINS_USER*, and *CPACK_PRODUCTBUILD_DOMAINS_ROOT*.

CPACK_PRODUCTBUILD_DOMAINS_ANYWHERE

Added in version 3.23.

May be used to override the **enable_anywhere** attribute in the **domains** element of the Distribution XML. When set to true, the product can be installed at the root of any volume, including non-system volumes.

CPACK_PRODUCTBUILD_DOMAINS must be set to true for this variable to have any effect.

CPACK_PRODUCTBUILD_DOMAINS_USER

Added in version 3.23.

May be used to override the **enable_currentUserHome** attribute in the **domains** element of the Distribution XML. When set to true, the product can be installed into the current user's home directory. Note that when installing into the user's home directory, the following additional requirements will apply:

- The installer may not write outside the user's home directory.
- The install will be performed as the current user rather than as **root**. This may have ramifications for *CPACK_PREFLIGHT_<COMP>_SCRIPT* and *CPACK_POSTFLIGHT_<COMP>_SCRIPT*.
- Administrative privileges will not be needed to perform the install.

CPACK_PRODUCTBUILD_DOMAINS must be set to true for this variable to have any effect.

CPACK_PRODUCTBUILD_DOMAINS_ROOT

Added in version 3.23.

May be used to override the **enable_localSystem** attribute in the **domains** element of the Distribution XML. When set to true, the product can be installed in the root directory. This should normally be set to true unless the product should only be installed to the user's home directory.

CPACK_PRODUCTBUILD_DOMAINS must be set to true for this variable to have any effect.

Background Image

Added in version 3.17.

This group of variables controls the background image of the generated installer.

CPACK_PRODUCTBUILD_BACKGROUND

Adds a background to Distribution XML if specified. The value contains the path to image in **Resources** directory.

CPACK_PRODUCTBUILD_BACKGROUND_ALIGNMENT

Adds an **alignment** attribute to the background in Distribution XML. Refer to Apple documentation for valid values.

CPACK_PRODUCTBUILD_BACKGROUND_SCALING

Adds a **scaling** attribute to the background in Distribution XML. Refer to Apple documentation for valid values.

CPACK_PRODUCTBUILD_BACKGROUND_MIME_TYPE

Adds a **mime-type** attribute to the background in Distribution XML. The option contains MIME type of an image.

CPACK_PRODUCTBUILD_BACKGROUND_UTI

Adds an **uti** attribute to the background in Distribution XML. The option contains UTI type of an image.

CPACK_PRODUCTBUILD_BACKGROUND_DARKAQUA

Adds a background for the Dark Aqua theme to Distribution XML if specified. The value contains the path to image in **Resources** directory.

CPACK_PRODUCTBUILD_BACKGROUND_DARKAQUA_ALIGNMENT

Does the same as *CPACK_PRODUCTBUILD_BACKGROUND_ALIGNMENT* option, but for the dark theme.

CPACK_PRODUCTBUILD_BACKGROUND_DARKAQUA_SCALING

Does the same as *CPACK_PRODUCTBUILD_BACKGROUND_SCALING* option, but for the dark theme.

CPACK_PRODUCTBUILD_BACKGROUND_DARKAQUA_MIME_TYPE

Does the same as *CPACK_PRODUCTBUILD_BACKGROUND_MIME_TYPE* option, but for the dark theme.

CPACK_PRODUCTBUILD_BACKGROUND_DARKAQUA_UTI

Does the same as *CPACK_PRODUCTBUILD_BACKGROUND_UTI* option, but for the dark theme.

Distribution XML Template

CPack uses a template file to generate the **distribution.dist** file used internally by this package generator. Ordinarily, CMake provides the template file, but projects may supply their own by placing a file called **CPack.distribution.dist.in** in one of the directories listed in the *CMAKE_MODULE_PATH* variable. CPack will then pick up the project's template file instead of using its own.

The **distribution.dist** file is generated by performing substitutions similar to the *configure_file()* command. Any variable set when CPack runs will be available for substitution using the usual *@...@* form. The following variables are also set internally and made available for substitution:

CPACK_RESOURCE_FILE_LICENSE_NOPATH

Same as *CPACK_RESOURCE_FILE_LICENSE* except without the path. The named file will be available in the same directory as the generated **distribution.dist** file.

CPACK_RESOURCE_FILE_README_NOPATH

Same as *CPACK_RESOURCE_FILE_README* except without the path. The named file will be available in the same directory as the generated **distribution.dist** file.

CPACK_RESOURCE_FILE_WELCOME_NOPATH

Same as *CPACK_RESOURCE_FILE_WELCOME* except without the path. The named file will be available in the same directory as the generated **distribution.dist** file.

CPACK_APPLE_PKG_INSTALLER_CONTENT

Added in version 3.23.

This contains all the XML elements that specify installer-wide options (including domain details), default backgrounds and the choices outline.

CPACK_PACKAGEMAKER_CHOICES

Deprecated since version 3.23.

This contains only the XML elements that specify the default backgrounds and the choices outline. It does not include the installer-wide options or any domain details. Use **CPACK_APPLE_PKG_INSTALLER_CONTENT** instead.

CPack RPM Generator

The built in (binary) CPack RPM generator (Unix only)

Variables specific to CPack RPM generator

The CPack RPM generator may be used to create RPM packages using *CPack*. The CPack RPM generator is a *CPack* generator thus it uses the **CPACK_XXX** variables used by *CPack*.

The CPack RPM generator has specific features which are controlled by the specifics **CPACK_RPM_XXX** variables.

CPACK_RPM_<COMPONENT>_XXXX variables may be used in order to have **component-specific** values. Note however that **<COMPONENT>** refers to the **grouping name** written in upper case. It may be either a component name or a component GROUP name. Usually, those variables correspond to RPM spec file entities. One may find information about spec files here <https://rpm.org/documentation>.

Changed in version 3.6: **<COMPONENT>** part of variables is preferred to be in upper case (e.g. if component is named **foo** then use **CPACK_RPM_FOO_XXXX** variable name format) as is with other **CPACK_<COMPONENT>_XXXX** variables. For the purposes of back compatibility (CMake/CPack version 3.5 and lower) support for same cased component (e.g. **fOo** would be used as **CPACK_RPM_fOo_XXXX**) is still supported for variables defined in older versions of CMake/CPack but is not guaranteed for variables that will be added in the future. For the sake of back compatibility same cased component variables also override upper cased versions where both are present.

Here are some CPack RPM generator wiki resources that are here for historic reasons and are no longer maintained but may still prove useful:

- <https://gitlab.kitware.com/cmake/community/-/wikis/doc/cpack/Configuration>
- <https://gitlab.kitware.com/cmake/community/-/wikis/doc/cpack/PackageGenerators#rpm-unix-only>

List of CPack RPM generator specific variables:

CPACK_RPM_COMPONENT_INSTALL

Enable component packaging for CPack RPM generator

Mandatory

No

Default

OFF

If enabled (ON) multiple packages are generated. By default a single package containing files of all components is generated.

CPACK_RPM_PACKAGE_SUMMARY

CPACK_RPM_<component>_PACKAGE_SUMMARY

The RPM package summary.

Mandatory

Yes

Default

CPACK_PACKAGE_DESCRIPTION_SUMMARY

Added in version 3.2: Per-component **CPACK_RPM_<component>_PACKAGE_SUMMARY** variables.

CPACK_RPM_PACKAGE_NAME

CPACK_RPM_<component>_PACKAGE_NAME

The RPM package name.

Mandatory

Yes

Default

CPACK_PACKAGE_NAME

Added in version 3.5: Per-component **CPACK_RPM_<component>_PACKAGE_NAME** variables.

CPACK_RPM_FILE_NAME

CPACK_RPM_<component>_FILE_NAME

Added in version 3.6.

Package file name.

Mandatory

Yes

Default

<CPACK_PACKAGE_FILE_NAME>[-<component>].rpm with spaces replaced by '_'

This may be set to:

RPM-DEFAULT

Tell **rpmbuild** to automatically generate the package file name.

<file-name>[.rpm]

Use the given file name.

Changed in version 3.29: The **.rpm** suffix will be automatically added if missing. Previously the suffix was required.

NOTE:

By using user provided spec file, rpm macro extensions such as for generating **debuginfo** packages or by simply using multiple components more than one rpm file may be generated, either from a single spec file or from multiple spec files (each component execution produces its own spec file). In such cases duplicate file names may occur as a result of this variable setting or spec file content structure. Duplicate files get overwritten and it is up to the packager to set the variables in a manner that will prevent such errors.

CPACK_RPM_MAIN_COMPONENT

Added in version 3.8.

Main component that is packaged without component suffix.

Mandatory

No

Default

This variable can be set to any component or group name so that component or group rpm package is generated without component suffix in filename and package name.

CPACK_RPM_PACKAGE_EPOCH

Added in version 3.10.

The RPM package epoch

Mandatory

No

Default

Optional number that should be incremented when changing versioning schemas or fixing mistakes in the version numbers of older packages.

CPACK_RPM_PACKAGE_VERSION

The RPM package version.

Mandatory

Yes

Default

CPACK_PACKAGE_VERSION

CPACK_RPM_PACKAGE_ARCHITECTURE**CPACK_RPM_<component>_PACKAGE_ARCHITECTURE**

The RPM package architecture.

Mandatory

Yes

DefaultNative architecture output by **uname -m**

This may be set to **noarch** if you know you are building a **noarch** package.

Added in version 3.3: Per-component **CPACK_RPM_<component>_PACKAGE_ARCHITECTURE** variables.

CPACK_RPM_PACKAGE_RELEASE

The RPM package release.

Mandatory

Yes

Default

1

This is the numbering of the RPM package itself, i.e. the version of the packaging and not the version of the content (see **CPACK_RPM_PACKAGE_VERSION**). One may change the default value if the previous packaging was buggy and/or you want to put here a fancy Linux distro specific numbering.

NOTE:

This is the string that goes into the RPM **Release:** field. Some distros (e.g. Fedora, CentOS) require **1%{?dist}** format and not just a number. **%{?dist}** part can be added by setting **CPACK_RPM_PACKAGE_RELEASE_DIST**.

CPACK_RPM_PACKAGE_RELEASE_DIST

Added in version 3.6.

The dist tag that is added RPM **Release:** field.

Mandatory

No

Default**OFF**

This is the reported **%{dist}** tag from the current distribution or empty **%{dist}** if RPM macro is not set. If this variable is set then RPM **Release:** field value is set to **\${CPACK_RPM_PACKAGE_RELEASE}%{?dist}**.

CPACK_RPM_PACKAGE_LICENSE

The RPM package license policy.

Mandatory

Yes

Default

"unknown"

CPACK_RPM_PACKAGE_GROUP**CPACK_RPM_<component>_PACKAGE_GROUP**

The RPM package group.

Mandatory

Yes

Default`"unknown"`

Added in version 3.5: Per-component **CPACK_RPM_<component>_PACKAGE_GROUP** variables.

CPACK_RPM_PACKAGE_VENDOR

The RPM package vendor.

Mandatory

Yes

Default`CPACK_PACKAGE_VENDOR` if set or `"unknown"`**CPACK_RPM_PACKAGE_URL****CPACK_RPM_<component>_PACKAGE_URL**

The projects URL.

Mandatory

No

Default`CMAKE_PROJECT_HOMEPAGE_URL`

Added in version 3.12: The **CMAKE_PROJECT_HOMEPAGE_URL** variable.

CPACK_RPM_PACKAGE_DESCRIPTION**CPACK_RPM_<component>_PACKAGE_DESCRIPTION**

RPM package description.

Mandatory

Yes

Default

- `CPACK_COMPONENT_<compName>_DESCRIPTION` (component based installers only) if set,
- `CPACK_PACKAGE_DESCRIPTION_FILE` if set, or
- **no package description available**

Added in version 3.2: Per-component **CPACK_RPM_<component>_PACKAGE_DESCRIPTION** variables.

CPACK_RPM_COMPRESSION_TYPE

RPM compression type.

Mandatory

No

Default

(system default)

May be used to override RPM compression type to be used to build the RPM. For example some Linux distributions default to **xz** or **zstd**. Using this, one can specify a specific compression type

to be used.

Possible values are:

lzma	Lempel–Ziv–Markov chain algorithm
xz	XZ Utils compression
bzip2	bzip2 Burrows–Wheeler algorithm
gzip	GNU Gzip compression
zstd	Added in version 3.31.

Zstandard compression

CPACK_RPM_PACKAGE_AUTOREQ

CPACK_RPM_<component>_PACKAGE_AUTOREQ

RPM spec autoreq field.

Mandatory

No

Default

May be used to enable (**1, yes**) or disable (**0, no**) automatic shared libraries dependency detection. Dependencies are added to requires list.

NOTE:

By default automatic dependency detection is enabled by rpm generator.

CPACK_RPM_PACKAGE_AUTOPROV

CPACK_RPM_<component>_PACKAGE_AUTOPROV

RPM spec autoprov field.

Mandatory

No

Default

May be used to enable (**1, yes**) or disable (**0, no**) automatic listing of shared libraries that are provided by the package. Shared libraries are added to provides list.

NOTE:

By default automatic provides detection is enabled by rpm generator.

CPACK_RPM_PACKAGE_AUTOREQPROV

CPACK_RPM_<component>_PACKAGE_AUTOREQPROV

RPM spec autoreqprov field.

Mandatory

No

Default

Variable enables/disables autoreq and autoprov at the same time. See *CPACK_RPM_PACKAGE_AUTOREQ* and *CPACK_RPM_PACKAGE_AUTOPROV* for more details.

NOTE:

By default automatic detection feature is enabled by rpm.

CPACK_RPM_PACKAGE_REQUIRES**CPACK_RPM_<component>_PACKAGE_REQUIRES**

RPM spec requires field.

Mandatory

No

Default

May be used to set RPM dependencies (requires). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_REQUIRES "python >= 2.5.0, cmake >= 2.8")
```

The required package list of an RPM file could be printed with:

```
rpm -qp --requires file.rpm
```

CPACK_RPM_PACKAGE_CONFLICTS**CPACK_RPM_<component>_PACKAGE_CONFLICTS**

RPM spec conflicts field.

Mandatory

No

Default

May be used to set negative RPM dependencies (conflicts). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_CONFLICTS "libxml2")
```

The conflicting package list of an RPM file could be printed with:

```
rpm -qp --conflicts file.rpm
```

CPACK_RPM_PACKAGE_REQUIRES_PRE**CPACK_RPM_<component>_PACKAGE_REQUIRES_PRE**

Added in version 3.2.

RPM spec requires(pre) field.

Mandatory

No

Default

May be used to set RPM preinstall dependencies (requires(pre)). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_REQUIRES_PRE "shadow-utils, initscripts")
```

CPACK_RPM_PACKAGE_REQUIRES_POST**CPACK_RPM_<component>_PACKAGE_REQUIRES_POST**

Added in version 3.2.

RPM spec requires(post) field.

Mandatory

No

Default

May be used to set RPM postinstall dependencies (requires(post)). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_REQUIRES_POST "shadow-utils, initscripts")
```

CPACK_RPM_PACKAGE_REQUIRES_POSTUN

CPACK_RPM_<component>_PACKAGE_REQUIRES_POSTUN

Added in version 3.2.

RPM spec requires(postun) field.

Mandatory

No

Default

May be used to set RPM postuninstall dependencies (requires(postun)). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_REQUIRES_POSTUN "shadow-utils, initscripts")
```

CPACK_RPM_PACKAGE_REQUIRES_PREUN

CPACK_RPM_<component>_PACKAGE_REQUIRES_PREUN

Added in version 3.2.

RPM spec requires(preun) field.

Mandatory

No

Default

May be used to set RPM preuninstall dependencies (requires(preun)). Note that you must enclose the complete requires string between quotes, for example:

```
set(CPACK_RPM_PACKAGE_REQUIRES_PREUN "shadow-utils, initscripts")
```

CPACK_RPM_PACKAGE_SUGGESTS

CPACK_RPM_<component>_PACKAGE_SUGGESTS

RPM spec suggest field.

Mandatory

No

Default

May be used to set weak RPM dependencies (suggests). If **rpmbuild** doesn't support the **Suggests** tag, CPack will emit a warning and ignore this variable. Note that you must enclose the complete requires string between quotes.

CPACK_RPM_PACKAGE_PROVIDES**CPACK_RPM_<component>_PACKAGE_PROVIDES**

RPM spec provides field.

Mandatory

No

Default

May be used to set RPM dependencies (provides). The provided package list of an RPM file could be printed with:

```
rpm -qp --provides file.rpm
```

CPACK_RPM_PACKAGE_OBSOLETES**CPACK_RPM_<component>_PACKAGE_OBSOLETES**

RPM spec obsoletes field.

Mandatory

No

Default

May be used to set RPM packages that are obsoleted by this one.

CPACK_RPM_PACKAGE_RELOCATABLE

build a relocatable RPM.

Mandatory

No

Default

CPACK_PACKAGE_RELOCATABLE

If this variable is set to TRUE or ON, the CPack RPM generator will try to build a relocatable RPM package. A relocatable RPM may be installed using:

```
rpm --prefix or --relocate
```

in order to install it at an alternate place see rpm(8). Note that currently this may fail if *CPACK_SET_DESTDIR* is set to **ON**. If *CPACK_SET_DESTDIR* is set then you will get a warning message but if there is file installed with absolute path you'll get unexpected behavior.

CPACK_RPM_SPEC_INSTALL_POST

Deprecated since version 2.8.12: Use *CPACK_RPM_SPEC_MORE_DEFINE* instead.

Mandatory

No

Default

May be used to override the **__spec_install_post** section within the generated spec file. This affects the install step during package creation, not during package installation. For adding operations to be performed during package installation, use *CPACK_RPM_POST_INSTALL_SCRIPT_FILE* instead.

CPACK_RPM_SPEC_MORE_DEFINE

RPM extended spec definitions lines.

Mandatory

No

Default

May be used to add any **%define** lines to the generated spec file. An example of its use is to prevent stripping of executables (but note that this may also disable other default post install processing):

```
set(CPACK_RPM_SPEC_MORE_DEFINE "%define __spec_install_post /bin/true")
```

CPACK_RPM_PACKAGE_DEBUG

Toggle CPack RPM generator debug output.

Mandatory

No

Default

May be set when invoking cpack in order to trace debug information during CPack RPM run. For example you may launch CPack like this:

```
cpack -D CPACK_RPM_PACKAGE_DEBUG=1 -G RPM
```

CPACK_RPM_USER_BINARY_SPECFILE**CPACK_RPM_<componentName>_USER_BINARY_SPECFILE**

A user provided spec file.

Mandatory

No

Default

May be set by the user in order to specify a USER binary spec file to be used by the CPack RPM generator instead of generating the file. The specified file will be processed by `configure_file(@ONLY)`.

CPACK_RPM_GENERATE_USER_BINARY_SPECFILE_TEMPLATE

Spec file template.

Mandatory

No

Default

If set CPack will generate a template for USER specified binary spec file and stop with an error. For example launch CPack like this:

```
cpack -D CPACK_RPM_GENERATE_USER_BINARY_SPECFILE_TEMPLATE=1 -G RPM
```

The user may then use this file in order to hand-craft its own binary spec file which may be used with `CPACK_RPM_USER_BINARY_SPECFILE`.

CPACK_RPM_PRE_INSTALL_SCRIPT_FILE**CPACK_RPM_PRE_UNINSTALL_SCRIPT_FILE****CPACK_RPM_PRE_TRANS_SCRIPT_FILE**

Path to file containing pre install/uninstall/transaction script.

Mandatory

No

Default

May be used to embed a pre installation/uninstallation/transaction script in the spec file. The referred script file (or both) will be read and directly put after the **%pre** or **%preun** section. If **CPACK_RPM_COMPONENT_INSTALL** is set to ON the install/uninstall/transaction script for each component can be overridden with **CPACK_RPM_<COMPONENT>_PRE_INSTALL_SCRIPT_FILE**, **CPACK_RPM_<COMPONENT>_PRE_UNINSTALL_SCRIPT_FILE**, and **CPACK_RPM_<COMPONENT>_PRE_TRANS_SCRIPT_FILE**. One may verify which scriptlet has been included with:

```
rpm -qp --scripts package.rpm
```

Added in version 3.18: The **CPACK_RPM_PRE_TRANS_SCRIPT_FILE** variable.

CPACK_RPM_POST_INSTALL_SCRIPT_FILE**CPACK_RPM_POST_UNINSTALL_SCRIPT_FILE****CPACK_RPM_POST_TRANS_SCRIPT_FILE**

Path to file containing post install/uninstall/transaction script.

Mandatory

No

Default

May be used to embed a post installation/uninstallation/transaction script in the spec file. The referred script file (or both) will be read and directly put after the **%post** or **%postun** section. If **CPACK_RPM_COMPONENT_INSTALL** is set to ON the install/uninstall/transaction script for each component can be overridden with **CPACK_RPM_<COMPONENT>_POST_INSTALL_SCRIPT_FILE**, **CPACK_RPM_<COMPONENT>_POST_UNINSTALL_SCRIPT_FILE**, and **CPACK_RPM_<COMPONENT>_POST_TRANS_SCRIPT_FILE**. One may verify which scriptlet has been included with:

```
rpm -qp --scripts package.rpm
```

Added in version 3.18: The **CPACK_RPM_POST_TRANS_SCRIPT_FILE** variable.

CPACK_RPM_USER_FILELIST**CPACK_RPM_<COMPONENT>_USER_FILELIST****Mandatory**

No

Default

May be used to explicitly specify **%(<directive>)** file line in the spec file. Like **%config(noreplace)** or any other directive that be found in the **%files** section. Since the CPack RPM generator is generating the list of files (and directories) the user specified files of the **CPACK_RPM_<COMPONENT>_USER_FILELIST** list will be removed from the generated list. If referring to directories do not add a trailing slash.

Added in version 3.8: You can have multiple directives per line, as in **%attr(600,root,root)**

%config(noreplace).

CPACK_RPM_CHANGELOG_FILE

RPM changelog file.

Mandatory

No

Default

May be used to embed a changelog in the spec file. The referred file will be read and directly put after the **%changelog** section.

CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST

list of path to be excluded.

Mandatory

No

Default

The following paths are excluded by default:

- **/etc**
- **/etc/init.d**
- **/usr**
- **/usr/bin**
- **/usr/include**
- **/usr/lib**
- **/usr/libx32**
- **/usr/lib64**
- **/usr/share**
- **/usr/share/aclocal**
- **/usr/share/doc**

May be used to exclude path (directories or files) from the auto-generated list of paths discovered by CPack RPM. The default value contains a reasonable set of values if the variable is not defined by the user. If the variable is defined by the user then the CPack RPM generator will NOT any of the default path. If you want to add some path to the default list then you can use *CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST_ADDITION* variable.

Added in version 3.10: Added **/usr/share/aclocal** to the default list of excludes.

CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST_ADDITION

additional list of path to be excluded.

Mandatory

No

Default

May be used to add more exclude path (directories or files) from the initial default list of excluded paths. See *CPACK_RPM_EXCLUDE_FROM_AUTO_FILELIST*.

CPACK_RPM_RELOCATION_PATHS

Added in version 3.2.

Packages relocation paths list.

Mandatory

No

Default

May be used to specify more than one relocation path per relocatable RPM. Variable contains a list of relocation paths that if relative are prefixed by the value of *CPACK_RPM_<COMPONENT>_PACKAGE_PREFIX* or by the value of *CPACK_PACKAGING_INSTALL_PREFIX* if the component version is not provided. Variable is not component based as its content can be used to set a different path prefix for e.g. binary dir and documentation dir at the same time. Only prefixes that are required by a certain component are added to that component – component must contain at least one file/directory/symbolic link with *CPACK_RPM_RELOCATION_PATHS* prefix for a certain relocation path to be added. Package will not contain any relocation paths if there are no files/directories/symbolic links on any of the provided prefix locations. Packages that either do not contain any relocation paths or contain files/directories/symbolic links that are outside relocation paths print out an *AUTHOR_WARNING* that RPM will be partially relocatable.

CPACK_RPM_<COMPONENT>_PACKAGE_PREFIX

Added in version 3.2.

Per component relocation path install prefix.

Mandatory

No

Default

CPACK_PACKAGING_INSTALL_PREFIX

May be used to set per component *CPACK_PACKAGING_INSTALL_PREFIX* for relocatable RPM packages.

CPACK_RPM_NO_INSTALL_PREFIX_RELOCATION**CPACK_RPM_NO_<COMPONENT>_INSTALL_PREFIX_RELOCATION**

Added in version 3.3.

Removal of default install prefix from relocation paths list.

Mandatory

No

Default

CPACK_PACKAGING_INSTALL_PREFIX or
CPACK_RPM_<COMPONENT>_PACKAGE_PREFIX are treated as one of relocation paths

May be used to remove *CPACK_PACKAGING_INSTALL_PREFIX* and *CPACK_RPM_<COMPONENT>_PACKAGE_PREFIX* from relocatable RPM prefix paths.

CPACK_RPM_ADDITIONAL_MAN_DIRS

Added in version 3.3.

Mandatory

No

Default

Regular expressions that are added by default were taken from brp-compress RPM macro:

- /usr/man/man.*
- /usr/man/.*/man.*
- /usr/info.*
- /usr/share/man/man.*
- /usr/share/man/.*/man.*
- /usr/share/info.*
- /usr/kerberos/man.*
- /usr/X11R6/man/man.*
- /usr/lib/perl5/man/man.*
- /usr/share/doc/.*/man/man.*
- /usr/lib/.*/man/man.*

May be used to set additional man dirs that could potentially be compressed by brp-compress RPM macro. Variable content must be a list of regular expressions that point to directories containing man files or to man files directly. Note that in order to compress man pages a path must also be present in brp-compress RPM script and that brp-compress script must be added to RPM configuration by the operating system.

CPACK_RPM_DEFAULT_USER**CPACK_RPM_<compName>_DEFAULT_USER**

Added in version 3.6.

default user ownership of RPM content

Mandatory

No

Default

root

Value should be user name and not UID. Note that <compName> must be in upper-case.

CPACK_RPM_DEFAULT_GROUP**CPACK_RPM_<compName>_DEFAULT_GROUP**

Added in version 3.6.

default group ownership of RPM content

Mandatory

No

Default

root

Value should be group name and not GID. Note that **<compName>** must be in upper-case.

CPACK_RPM_DEFAULT_FILE_PERMISSIONS**CPACK_RPM_<compName>_DEFAULT_FILE_PERMISSIONS**

Added in version 3.6.

default permissions used for packaged files

Mandatory

No

Default

(system default)

Accepted values are lists with PERMISSIONS. Valid permissions are:

- **OWNER_READ**
- **OWNER_WRITE**
- **OWNER_EXECUTE**
- **GROUP_READ**
- **GROUP_WRITE**
- **GROUP_EXECUTE**
- **WORLD_READ**
- **WORLD_WRITE**
- **WORLD_EXECUTE**

Note that **<compName>** must be in upper-case.

CPACK_RPM_DEFAULT_DIR_PERMISSIONS**CPACK_RPM_<compName>_DEFAULT_DIR_PERMISSIONS**

Added in version 3.6.

default permissions used for packaged directories

Mandatory

No

Default

(system default)

Accepted values are lists with PERMISSIONS. Valid permissions are the same as for *CPACK_RPM_DEFAULT_FILE_PERMISSIONS*. Note that **<compName>** must be in upper-case.

CPACK_RPM_INSTALL_WITH_EXEC

Added in version 3.11.

force execute permissions on programs and shared libraries

Mandatory

No

Default

(system default)

Force set owner, group and world execute permissions on programs and shared libraries. This can be used for creating valid rpm packages on systems such as Debian where shared libraries do not have execute permissions set.

NOTE:

Programs and shared libraries without execute permissions are ignored during separation of debug symbols from the binary for debuginfo packages.

Packaging of Symbolic Links

Added in version 3.3.

The CPack RPM generator supports packaging of symbolic links:

```
execute_process(COMMAND ${CMAKE_COMMAND}
  -E create_symlink <relative_path_location> <symlink_name>)
install(FILES ${CMAKE_CURRENT_BINARY_DIR}/${symlink_name}
  DESTINATION <symlink_location> COMPONENT libraries)
```

Symbolic links will be optimized (paths will be shortened if possible) before being added to the package or if multiple relocation paths are detected, a post install symlink relocation script will be generated.

Symbolic links may point to locations that are not packaged by the same package (either a different component or even not packaged at all) but those locations will be treated as if they were a part of the package while determining if symlink should be either created or present in a post install script – depending on relocation paths.

Changed in version 3.6: Symbolic links that point to locations outside packaging path produce a warning and are treated as non relocatable permanent symbolic links. Previous versions of CMake produced an error in this case.

Currently there are a few limitations though:

- For component based packaging component interdependency is not checked when processing symbolic links. Symbolic links pointing to content of a different component are treated the same way as if pointing to location that will not be packaged.
- Symbolic links pointing to a location through one or more intermediate symbolic links will not be handled differently – if the intermediate symbolic link(s) is also on a relocatable path, relocating it during package installation may cause initial symbolic link to point to an invalid location.

Packaging of debug information

Added in version 3.7.

Debuginfo packages contain debug symbols and sources for debugging packaged binaries.

Debuginfo RPM packaging has its own set of variables:

CPACK_RPM_DEBUGINFO_PACKAGE**CPACK_RPM_<component>_DEBUGINFO_PACKAGE**

Enable generation of debuginfo RPM package(s).

Mandatory

No

Default

OFF

NOTE:

Binaries must contain debug symbols before packaging so use either **Debug** or **RelWithDebInfo** for *CMAKE_BUILD_TYPE* variable value.

Additionally, if *CPACK_STRIP_FILES* is set, the files will be stripped before they get to the RPM generator, so will not contain debug symbols and a debuginfo package will not get built. Do not use with *CPACK_STRIP_FILES*.

NOTE:

Packages generated from packages without binary files, with binary files but without execute permissions or without debug symbols will cause packaging termination.

CPACK_BUILD_SOURCE_DIRS

Provides locations of root directories of source files from which binaries were built.

Mandatory

Yes if *CPACK_RPM_DEBUGINFO_PACKAGE* is set

Default**NOTE:**

For CMake project *CPACK_BUILD_SOURCE_DIRS* is set by default to point to *CMAKE_SOURCE_DIR* and *CMAKE_BINARY_DIR* paths.

NOTE:

Sources with path prefixes that do not fall under any location provided with *CPACK_BUILD_SOURCE_DIRS* will not be present in debuginfo package.

CPACK_RPM_BUILD_SOURCE_DIRS_PREFIX**CPACK_RPM_<component>_BUILD_SOURCE_DIRS_PREFIX**

Prefix of location where sources will be placed during package installation.

Mandatory

Yes if *CPACK_RPM_DEBUGINFO_PACKAGE* is set

Default

/usr/src/debug/\${CPACK_PACKAGE_FILE_NAME} and for component packaging
/usr/src/debug/\${CPACK_PACKAGE_FILE_NAME}-<component>

NOTE:

Each source path prefix is additionally suffixed by **src_<index>** where index is index of the path used from *CPACK_BUILD_SOURCE_DIRS* variable. This produces **\${CPACK_RPM_BUILD_SOURCE_DIRS_PREFIX}/src_<index>** replacement path. Limitation is that replaced path part must be shorter or of equal length than the length of its replacement. If that is not the case either *CPACK_RPM_BUILD_SOURCE_DIRS_PREFIX* variable has to be set to a shorter path or source directories must be placed on a longer path.

CPACK_RPM_DEBUGINFO_EXCLUDE_DIRS

Directories containing sources that should be excluded from debuginfo packages.

Mandatory

No

Default

The following paths are excluded by default:

- `/usr`
- `/usr/src`
- `/usr/src/debug`

Listed paths are owned by other RPM packages and should therefore not be deleted on debuginfo package uninstallation.

CPACK_RPM_DEBUGINFO_EXCLUDE_DIRS_ADDITION

Paths that should be appended to `CPACK_RPM_DEBUGINFO_EXCLUDE_DIRS` for exclusion.

Mandatory

No

Default**CPACK_RPM_DEBUGINFO_SINGLE_PACKAGE**

Added in version 3.8.

Create a single debuginfo package even if components packaging is set.

Mandatory

No

Default

OFF

When this variable is enabled it produces a single debuginfo package even if component packaging is enabled.

When using this feature in combination with components packaging and there is more than one component this variable requires `CPACK_RPM_MAIN_COMPONENT` to be set.

NOTE:

If none of the `CPACK_RPM_<component>_DEBUGINFO_PACKAGE` variables is set then `CPACK_RPM_DEBUGINFO_PACKAGE` is automatically set to **ON** when `CPACK_RPM_DEBUGINFO_SINGLE_PACKAGE` is set.

CPACK_RPM_DEBUGINFO_FILE_NAME**CPACK_RPM_<component>_DEBUGINFO_FILE_NAME**

Added in version 3.9.

Debuginfo package file name.

Mandatory

No

Default

rpmbuild tool generated package file name

Alternatively provided debuginfo package file name must end with **.rpm** suffix and should differ from file names of other generated packages.

Variable may contain **@cpack_component@** placeholder which will be replaced by component name if component packaging is enabled otherwise it deletes the placeholder.

Setting the variable to **RPM-DEFAULT** may be used to explicitly set filename generation to default.

NOTE:

CPACK_RPM_FILE_NAME also supports rpmbuild tool generated package file name – disabled by default but can be enabled by setting the variable to **RPM-DEFAULT**.

Packaging of sources (SRPM)

Added in version 3.7.

SRPM packaging is enabled by setting *CPACK_RPM_PACKAGE_SOURCES* variable while usually using *CPACK_INSTALLED_DIRECTORIES* variable to provide directory containing CMakeLists.txt and source files.

For CMake projects SRPM package would be produced by executing:

```
cpack -G RPM --config ./CPackSourceConfig.cmake
```

NOTE:

Produced SRPM package is expected to be built with *cmake(1)* executable and packaged with *cpack(1)* executable so CMakeLists.txt has to be located in root source directory and must be able to generate binary rpm packages by executing *cpack -G* command. The two executables as well as rpmbuild must also be present when generating binary rpm packages from the produced SRPM package.

Once the SRPM package is generated it can be used to generate binary packages by creating a directory structure for rpm generation and executing rpmbuild tool:

```
mkdir -p build_dir/{BUILD,BUILDROOT,RPMS,SOURCES,SPECS,SRPMS}
rpmbuild --define "_topdir <path_to_build_dir>" --rebuild <SRPM_file_name>
```

Generated packages will be located in build_dir/RPMS directory or its sub directories.

NOTE:

SRPM package internally uses CPack/RPM generator to generate binary packages so CMakeScripts.txt can decide during the SRPM to binary rpm generation step what content the package(s) should have as well as how they should be packaged (monolithic or components). CMake can decide this for e.g. by reading environment variables set by the package manager before starting the process of generating binary rpm packages. This way a single SRPM package can be used to produce different binary rpm packages on different platforms depending on the platform's packaging rules.

Source RPM packaging has its own set of variables:

CPACK_RPM_PACKAGE_SOURCES

Should the content be packaged as a source rpm (default is binary rpm).

Mandatory

No

Default
OFF

NOTE:

For cmake projects `CPACK_RPM_PACKAGE_SOURCES` variable is set to **OFF** in `CPackConfig.cmake` and **ON** in `CPackSourceConfig.cmake` generated files.

CPACK_RPM_SOURCE_PKG_BUILD_PARAMS

Additional command-line parameters provided to `cmake(1)` executable.

Mandatory
No

Default

CPACK_RPM_SOURCE_PKG_PACKAGING_INSTALL_PREFIX

Packaging install prefix that would be provided in `CPACK_PACKAGING_INSTALL_PREFIX` variable for producing binary RPM packages.

Mandatory
Yes

Default

/

CPACK_RPM_BUILDREQUIRES

List of source rpm build dependencies.

Mandatory
No

Default

May be used to set source RPM build dependencies (BuildRequires). Note that you must enclose the complete build requirements string between quotes, for example:

```
set(CPACK_RPM_BUILDREQUIRES "python >= 2.5.0, cmake >= 2.8")
```

CPACK_RPM_REQUIRES_EXCLUDE_FROM

Added in version 3.22.

Mandatory
No

Default

May be used to keep the dependency generator from scanning specific files or directories for dependencies. Note that you can use a regular expression that matches all of the directories or files, for example:

```
set(CPACK_RPM_REQUIRES_EXCLUDE_FROM "bin/libqsqloci.*\\.so.*")
```

CPack WIX Generator

Use the *WiX Toolset* to produce a Windows Installer **.msi** database.

Added in version 3.7: The `CPACK_COMPONENT_<compName>_DISABLED` variable is now supported.

WiX Toolsets

CPack selects one of the following variants of the WiX Toolset based on the `CPACK_WIX_VERSION` variable:

- *WiX .NET Tools*
- *WiX Toolset v3*

WiX .NET Tools

Packaging is performed using the following tools:

wix build

Build WiX source files directly into a Windows Installer **.msi** database.

Invocations may be customized using tool-specific variables:

- *CPACK_WIX_BUILD_EXTENSIONS*
- *CPACK_WIX_BUILD_EXTRA_FLAGS*

WiX extensions must be named with the form **WixToolset.<Name>.wixext**.

CPack expects the **wix** .NET tool to be available for command-line use with any required WiX extensions already installed. Be sure the **wix** version is compatible with *CPACK_WIX_VERSION*, and that WiX extension versions match the **wix** tool version. For example:

1. Install the **wix** command-line tool using **dotnet**.

To install **wix** globally for the current user:

```
dotnet tool install --global wix --version 4.0.4
```

This places **wix.exe** in **%USERPROFILE%\dotnet\tools** and adds the directory to the current user's **PATH** environment variable.

Or, to install **wix** in a specific path, e.g., in **c:\WiX**:

```
dotnet tool install --tool-path c:\WiX wix --version 4.0.4
```

This places **wix.exe** in **c:\WiX**, but does *not* add it to the current user's **PATH** environment variable. The **WIX** environment variable may be set to tell CPack where to find the tool, e.g., **set WIX=c:\WiX**.

2. Add the WiX **UI** extension, needed by CPack's default WiX template:

```
wix extension add --global WixToolset.UI.wixext/4.0.4
```

Extensions added globally are stored in **%USERPROFILE%\wix**, or if the **WIX_EXTENSIONS** environment variable is set, in **%WIX_EXTENSIONS%\wix**.

WiX Toolset v3

Packaging is performed using the following tools:

candle Compiles WiX source files into **.wixobj** files.

Invocations may be customized using tool-specific variables:

- *CPACK_WIX_CANDLE_EXTENSIONS*
- *CPACK_WIX_CANDLE_EXTRA_FLAGS*

light Links **.wixobj** files into a Windows Installer **.msi** database.

Invocations may be customized using tool-specific variables:

- *CPACK_WIX_LIGHT_EXTENSIONS*

- `CPACK_WIX_LIGHT_EXTRA_FLAGS`

CPack invokes both tools as needed. Intermediate `.wixobj` files are considered implementation details.

WiX extensions must be named with the form **Wix<Name>Extension**.

CPack expects the above tools to be available for command-line use via the **PATH**. Or, if the **WIX** environment variable is set, CPack looks for the tools in `%WIX%` and `%WIX%\bin`.

Variables specific to CPack WIX generator

The following variables are specific to the installers built on Windows using WiX.

CPACK_WIX_VERSION

Added in version 3.30.

Specify the version of WiX Toolset for which the configuration is written. The value must be one of

- 4** Package using *WiX .NET Tools*.
- 3** Package using *WiX Toolset v3*. This is the default.

CPACK_WIX_UPGRADE_GUID

Upgrade GUID (**Product/@UpgradeCode**)

Will be automatically generated unless explicitly provided.

It should be explicitly set to a constant generated globally unique identifier (GUID) to allow your installers to replace existing installations that use the same GUID.

You may for example explicitly set this variable in your CMakeLists.txt to the value that has been generated per default. You should not use GUIDs that you did not generate yourself or which may belong to other projects.

A GUID shall have the following fixed length syntax:

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

(each X represents an uppercase hexadecimal digit)

CPACK_WIX_PRODUCT_GUID

Product GUID (**Product/@Id**)

Will be automatically generated unless explicitly provided.

If explicitly provided this will set the Product Id of your installer.

The installer will abort if it detects a pre-existing installation that uses the same GUID.

The GUID shall use the syntax described for `CPACK_WIX_UPGRADE_GUID`.

CPACK_WIX_LICENSE_RTF

RTF License File

If `CPACK_RESOURCE_FILE_LICENSE` has an `.rtf` extension it is used as-is.

If `CPACK_RESOURCE_FILE_LICENSE` has an `.txt` extension it is implicitly converted to RTF by the WIX Generator. The expected encoding of the `.txt` file is UTF-8.

With `CPACK_WIX_LICENSE_RTF` you can override the license file used by the WIX Generator in case `CPACK_RESOURCE_FILE_LICENSE` is in an unsupported format or the `.txt -> .rtf` conversion does not work as expected.

CPACK_WIX_PRODUCT_ICON

The Icon shown next to the program name in Add/Remove programs.

If set, this icon is used in place of the default icon.

CPACK_WIX_UI_REF

Specify the WiX UI extension's dialog set:

- With WiX *.NET Tools*, this is the Id of the `<ui:WixUI>` element in the default WiX template.
- With WiX *Toolset v3*, this is the Id of the `<UIRef>` element in the default WiX template.

The default is `WixUI_InstallDir` in case no CPack components have been defined and `WixUI_FeatureTree` otherwise.

CPACK_WIX_UI_BANNER

The bitmap will appear at the top of all installer pages other than the welcome and completion dialogs.

If set, this image will replace the default banner image.

This image must be 493 by 58 pixels.

CPACK_WIX_UI_DIALOG

Background bitmap used on the welcome and completion dialogs.

If this variable is set, the installer will replace the default dialog image.

This image must be 493 by 312 pixels.

CPACK_WIX_PROGRAM_MENU_FOLDER

Start menu folder name for launcher.

If this variable is not set, it will be initialized with `CPACK_PACKAGE_NAME`

Added in version 3.16: If this variable is set to `.`, then application shortcuts will be created directly in the start menu and the uninstaller shortcut will be omitted.

CPACK_WIX_CULTURES

Language(s) of the installer

Languages are compiled into the Wix UI extension library. To use them, simply provide the name of the culture. If you specify more than one culture identifier in a comma or semicolon delimited list, the first one that is found will be used. You can find a list of supported languages at: - https://docs.firegiant.com/wix3/wixui/wixui_localization/

CPACK_WIX_TEMPLATE

Template file for WiX generation

If this variable is set, the specified template will be used to generate the WiX wxs file. This should be used if further customization of the output is required. The template contents will override the effect of most `CPACK_WIX_` variables.

If this variable is not set, the default MSI template included with CMake will be used.

CPACK_WIX_PATCH_FILE

Optional list of XML files with fragments to be inserted into generated WiX sources.

Added in version 3.5: Support listing multiple patch files.

This optional variable can be used to specify an XML file that the WIX generator will use to inject fragments into its generated source files.

Patch files understood by the CPack WIX generator roughly follow this RELAX NG compact schema:

```

start = CPackWiXPatch

CPackWiXPatch = element CPackWiXPatch { CPackWiXFragment* }

CPackWiXFragment = element CPackWiXFragment
{
    attribute Id { string },
    fragmentContent*
}

fragmentContent = element * - CPackWiXFragment
{
    (attribute * { text } | text | fragmentContent)*
}

```

Currently fragments can be injected into most Component, File, Directory and Feature elements.

Added in version 3.3: The following additional special Ids can be used:

- **#PRODUCT** for the **<Product>** element.
- **#PRODUCTFEATURE** for the root **<Feature>** element.

Added in version 3.7: Support patching arbitrary **<Feature>** elements.

Added in version 3.9: Allow setting additional attributes.

The following example illustrates how this works.

Given that the WIX generator creates the following XML element:

```
<Component Id="CM_CP_applications.bin.my_libapp.exe" Guid="*" />
```

The following XML patch file may be used to inject an Environment element into it:

```

<CPackWiXPatch>
  <CPackWiXFragment Id="CM_CP_applications.bin.my_libapp.exe">
    <Environment Id="MyEnvironment" Action="set"
      Name="MyVariableName" Value="MyVariableValue" />
  </CPackWiXFragment>
</CPackWiXPatch>

```

CPACK_WIX_EXTRA_SOURCES

Extra WiX source files

This variable provides an optional list of extra WiX source files (**.wxs**) that should be compiled and linked. The paths must be absolute.

CPACK_WIX_EXTRA_OBJECTS

Extra WiX object files or libraries to use with *WiX Toolset v3*.

This variable provides an optional list of extra WiX object (**.wixobj**) and/or WiX library (**.wixlib**) files. The paths must be absolute.

CPACK_WIX_EXTENSIONS

Specify a list of additional extensions for WiX tools. See *WiX Toolsets* for extension naming patterns.

CPACK_WIX_<TOOL>_EXTENSIONS

Specify a list of additional extensions for a specific WiX tool. See *WiX Toolsets* for possible **<TOOL>** names.

CPACK_WIX_<TOOL>_EXTRA_FLAGS

Specify a list of additional command-line flags for a specific WiX tool. See *WiX Toolsets* for possible **<TOOL>** names.

Use it at your own risk. Future versions of CPack may generate flags which may be in conflict with your own flags.

CPACK_WIX_CMAKE_PACKAGE_REGISTRY

If this variable is set the generated installer will create an entry in the windows registry key **HKEY_LOCAL_MACHINE\Software\Kitware\CMake\Packages\<PackageName>** The value for **<PackageName>** is provided by this variable.

Assuming you also install a CMake configuration file this will allow other CMake projects to find your package with the *find_package()* command.

CPACK_WIX_PROPERTY_<PROPERTY>

Added in version 3.1.

This variable can be used to provide a value for the Windows Installer property **<PROPERTY>**

The following list contains some example properties that can be used to customize information under "Programs and Features" (also known as "Add or Remove Programs")

- ARPCOMMENTS – Comments
- ARPHELPLINK – Help and support information URL
- ARPURLINFOABOUT – General information URL
- ARPURLUPDATEINFO – Update information URL
- ARPHELPTELEPHONE – Help and support telephone number
- ARPSIZE – Size (in kilobytes) of the application

CPACK_WIX_ROOT_FEATURE_TITLE

Added in version 3.7.

Sets the name of the root install feature in the WIX installer. Same as **CPACK_COMPONENT_<compName>_DISPLAY_NAME** for components.

CPACK_WIX_ROOT_FEATURE_DESCRIPTION

Added in version 3.7.

Sets the description of the root install feature in the WIX installer. Same as `CPACK_COMPONENT_<compName>_DESCRIPTION` for components.

CPACK_WIX_SKIP_PROGRAM_FOLDER

Added in version 3.7.

If this variable is set to true, the default install location of the generated package will be `CPACK_PACKAGE_INSTALL_DIRECTORY` directly. The install location will not be located relatively below `ProgramFiles` or `ProgramFiles64`.

NOTE:

Installers created with this feature do not take differences between the system on which the installer is created and the system on which the installer might be used into account.

It is therefore possible that the installer e.g. might try to install onto a drive that is unavailable or unintended or a path that does not follow the localization or convention of the system on which the installation is performed.

CPACK_WIX_ROOT_FOLDER_ID

Added in version 3.9.

This variable allows specification of a custom root folder ID. The generator specific `<64>` token can be used for folder IDs that come in 32-bit and 64-bit variants. In 32-bit builds the token will expand empty while in 64-bit builds it will expand to **64**.

When unset generated installers will default installing to **ProgramFiles<64>Folder**.

CPACK_WIX_ROOT

This variable can optionally be set to the root directory of a custom WiX Toolset installation.

When unspecified CPack will try to locate a WiX Toolset installation via the **WIX** environment variable instead.

CPACK_WIX_CUSTOM_XMLNS

Added in version 3.19.

This variable provides a list of custom namespace declarations that are necessary for using WiX extensions. Each declaration should be in the form `name=url`, where `name` is the plain namespace without the usual `xmlns:` prefix and `url` is an unquoted namespace url. A list of commonly known WiX schemata can be found here: <https://docs.firegiant.com/wix3/xsd/>

CPACK_WIX_SKIP_WIX_UI_EXTENSION

Added in version 3.23.

If this variable is set to true, the default inclusion of the WiX **UI** extension is skipped, i.e., the `-ext WixUIExtension` or `-ext WixToolset.UI.wixext` flag is not passed to WiX tools.

CPACK_WIX_ARCHITECTURE

Added in version 3.24.

This variable can be optionally set to specify the target architecture of the installer. May for example be set to **x64** or **arm64**.

When unspecified, CPack will default to **x64** or **x86**.

CPACK_WIX_INSTALL_SCOPE

Added in version 3.29.

This variable can be optionally set to specify the **InstallScope** of the installer:

perMachine

Create an installer that installs for all users and requires administrative privileges. Start menu entries created by the installer are visible to all users.

This is the default. See policy *CMP0172*.

perUser

Not yet supported. This is reserved for future use.

NONE Create an installer without any **InstallScope** attribute.

This is supported only if *CPACK_WIX_VERSION* is not set, or is set to **3**.

Deprecated since version 3.29: This value is only for compatibility with the inconsistent behavior used by CPack 3.28 and older. The resulting installer requires administrative privileges and installs into the system-wide **ProgramFiles** directory, but the start menu entry and uninstaller registration are created only for the current user.

WARNING:

An installation performed by an installer created without any **InstallScope** cannot be cleanly updated or replaced by an installer with an **InstallScope**. In order to transition a project's installers from **NONE** to **perMachine**, the latter installer should be distributed with instructions to first manually uninstall any older version.

See <https://docs.firegiant.com/wix3/xsd/wix/package/>

COPYRIGHT

2000-2025 Kitware, Inc. and Contributors