SUSE

SUSE Best Practices

Systems Management

Kernel Module Package Manual for SUSE Linux Enterprise Server 16.0

SUSE Linux Enterprise 16.0
SUSE Linux Micro 6.2

Scott Bahling, Head of Partner Technology Management (SUSE)

SUSE

Kernel Module Package Manual for SUSE Linux Enterprise Server 16.0

Date: 2026-05-21

This document specifies the requirements for RPM packages that contain kernel modules, and describes the processes surrounding those packages including building, signing, installing and upgrading. A complete example is given and explained. This version of the Kernel Module Packages Manual applies to SUSE Linux Enterprise Server 16.0.

Contents

1 Overview

SUSE-based distributions use the RPM Package Manager for software management. As such, any external kernel modules (these are kernel modules not included in SUSE kernel packages) should be packaged in RPM packages. These RPMs should be built in accordance with specific guidelines to ensure that the resulting kernel module packages (KMPs) can be installed and updated appropriately, in synchronization with kernel updates.

This document specifies the requirements for RPM packages that contain kernel modules, and describes the processes surrounding those packages including building, signing, installing and upgrading. A complete example is given and explained.

This version of the Kernel Module Packages Manual applies to SUSE Linux Enterprise Server 16.0

2 Scope

This version of the Kernel Module Packages Manual applies to:

- SUSE Linux Enterprise Server 16.0
- SUSE Linux Micro 6.2

Developers who want to build for earlier versions of SUSE Linux Enterprise Server should use the previous versions of the Kernel Module Package Manual.

Appendix B provides a list of changes that have been made to this document to accommodate new features.

3 Background

The SUSE Linux Enterprise Server kernel supports adding functionality at runtime through kernel-loadable modules. It includes more than 2000 modules, about 60 percent of which are hardware drivers. These modules are shipped as part of the kernel packages. In some cases it is desirable to add additional modules or replace existing ones. For example, a driver for a particular storage controller that was not available at the time of product release might be added later to support new hardware.

Kernel modules interact with the kernel by the means of exported symbols, in a way similar to how user space binaries use shared libraries. The `/proc/kallsyms` file lists all symbols currently known to the kernel. To ensure that the kernel and modules refer to the same symbols, a version

checksum (modversion) is added to each symbol. The checksum is computed from the symbol's type: in the case of function symbols, the checksum is determined by the function's parameters and return type.

When any of a function's parameters or the return type changes, the checksum changes as well. This includes all the data types involved recursively:

If a function takes a **struct task_struct** as parameter and **struct task_struct** includes a field of type **struct dentry**, then a change to **struct dentry** will cause the symbol's version checksum to change as well. Symbol version checksums for different kernel flavors (for example **kernel-default** versus **kernel-rt**) will not match, and symbol versions of the same kernel package on different architectures will not match either. This mechanism ensures that the kernel and kernel modules agree on the types of data structures that they use to communicate.

Unless symbol version checking is disabled, modules will load only if the checksums of the symbols they use match the checksums of the symbols that the kernel exports. The exported symbols and their version checksums comprise the kernel Application Binary Interface (kABI). When an updated kernel includes kABI changes, kernel modules that use any modified symbols must be updated as well.

During their multi-year lifecycle, products like SUSE Linux Enterprise Server undergo continuous changes. Different kinds of updates like major and minor version releases, maintenance/security updates, and customer-specific updates (Program Temporary Fixes) are released. The Application Binary Interface (ABI) between the kernel and kernel modules is volatile. Some kernel updates will change the kernel ABI (kABI) by adding or removing exported symbols, or existing symbol checksums can change in a kernel update because of changes in data structures they reference. SUSE strives to keep the kernel ABI stable in maintenance and security and customer-specific updates, but sometimes changes cannot be avoided. In service packs or minor releases, SUSE reserves the right to introduce more intrusive changes, which increases the likelihood of ABI changes. SUSE believes that the added flexibility outweighs the disadvantages of breaking older modules. For full discussion of this topic, see the documentation SUSE SolidDriver Documentation: SUSE Kernel ABI Stability[1] and The Linux Kernel Driver Interface from Greg Kroah-Hartman (also provided as stable_api_nonsense.txt in the upstream kernel source tree).

SUSE Linux-based operating systems include technology to ensure that kernel modules can be reused or updated in synchronization with kernel updates. To use this technology, kernel modules must be packaged into Kernel Module Packages (KMPs) as defined in this document.

1 https://drivers.suse.com/doc/SolidDriver/SUSE_Kernel_ABI_Stability.html

4 Kernel packages

Each product based on SUSE Linux 16 contains a set of kernel packages that share the same version and release number; they are built from the same kernel sources. These packages are:

4.1 kernel-FLAVOR, kernel-FLAVOR-base

The binary kernel packages. Each architecture has its own set of kernel flavors (for example `kernel-default`, `kernel-debug`, etc.). These are the packages that the kernel modules will be used with.

The `kernel-FLAVOR-base` packages are subsets of the `kernel-FLAVOR` packages, intended for use with minimal installs. They are not installed by default.

4.2 kernel-source

The kernel source tree, generated by unpacking the vanilla kernel sources and applying all necessary patches. Although the `kernel-FLAVOR` packages technically are not built from the kernel-source package, they are built from the same source tree. This tree should be used for module building.

4.3 kernel-devel, kernel-macros

Kernel-level headers, makefiles, and RPM macros and templates required for development of external kernel modules.

4.4 kernel-syms, kernel-FLAVOR-devel

Kernel symbol version information for compiling external modules. The `kernel-FLAVOR-devel` package is required for building external modules. If this package is not used, the resulting modules will be missing symbol version information, which will cause them to break during kernel updates. The `kernel-syms` package is a placeholder package which depends on the `kernel-FLAVOR-devel` packages for all kernel flavors.

For more information, refer to the document “Working With The SUSE 2.6.x and 3.x Kernel Sources” from Andreas Gruenbacher and Michal Marek. This document is provided as `README.SUSE` in the SUSE `kernel-source` package.

5 Kernel modules

Documentation on general kernel module building can be found in abundance on the Internet. Two good lectures are:

- Peter Jay Salzman, Michael Burian, Ori Pomerantz: The Linux Kernel Module Programming Guide, http://www.tldp.org/LDP/lkmpg/2.6/html/index.html
- Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman: Linux Device Drivers, Third Edition, February 2005, http://www.oreilly.com/catalog/linuxdrive3/ (also available online at http://lwn.net/Kernel/LDD3/)

SUSE-specific information is found in the above-mentioned README.SUSE in the `kernel-source` package.

When built, kernel module binaries are installed below `/usr/lib/modules/VERSION-RELEASE-FLAVOR` on the file system (example: `/usr/lib/modules/6.12.0-160000.5-default` for the SUSE Linux Enterprise Server 16.0 `kernel-default-6.12.0-160000.5` package). Different kernels have different module directories, and will usually not see each other's modules.

Update modules are modules intended to replace or augment the modules that are provided in the kernel packages. Update modules must be stored below the `/usr/lib/modules/VERSION-RELEASE-FLAVOR/updates/` directory. Modules in the `updates/` directory have precedence over other modules with the same name. Never replace modules from the kernel package by overwriting files: this would lead to inconsistencies between the file system and the RPM database.

Note: Where to store modules

Modules intended to take precedence over in-kernel modules of the same name should be stored below `/usr/lib/modules/VERSION-RELEASE-FLAVOR/updates/`. Other add-on modules can be stored below `/usr/lib/modules/VERSION-RELEASE-FLAVOR/extra/`.

Modules usually remain compatible with a range of `kernel-FLAVOR` packages. To make such modules visible to other `kernel-FLAVOR` packages, symbolic links to compatible modules are put in `/usr/lib/modules/VERSION-RELEASE-FLAVOR/weak-updates/` directories. Modules in the `weak-updates/` directory have lower priority than modules in the `updates/` directory, but higher priority than all other modules in `/usr/lib/modules/VERSION-RELEASE-FLAVOR`. If more than one compatible module is available for a kernel, the module with the highest kernel release is chosen. Kernel Module Packages must never directly install modules into `weak-updates/` directories.

Module loading order is controlled by the `/usr/lib/depmod.d/00-system.conf` file which is part of the suse-module-tools package.

Kernel modules must never be installed as individual files on a production system, but always as part of a Kernel Module Package.

6 Kernel Module Packages

SUSE has worked closely with the Linux Foundation Driver Backport Workgroup to establish a standard structure for building Kernel Module Packages for all RPM-based distributions. The information in this document includes the standards as appropriate.

Kernel Module Package spec files define a main package, and a sub-package for each kernel flavor supported. The kernel-flavor-specific sub-packages are defined with the **%kernel_module_package** RPM macro. The macro automatically determines for which kernel flavors to generate sub-packages. Several options are available to modify the macro's behavior, which are described below:

%kernel_module_package [-f filelist] [-p preamble] [-n name] [-v version] [-r release] [-t template] [-x flavor] [-b]

The main package of a Kernel Module Package can either contain no **%files** section, in which case `rpm` will not create a binary package with the main package's name, or the files section can also be used for the user space part associated with the kernel modules that end up in the kernel specific sub-packages. The example Kernel Module Package in Appendix A has a main package without a **%files** section.

Kernel Module Packages must adhere to the following rules:

- The package name should consist of two components: a unique provider prefix, and a driver name. Hyphens are disallowed in the provider prefix, and allowed in the driver name. The provider prefix serves to create a non-overlapping name space for all providers. The sub-package names are composed of the main package name, followed by a dash, the string "kmp", followed by another dash and the flavor of the supported kernel. The first component (main package name) can be overridden with a different value by using the `-n` option of the **%kernel_module_package** macro.
- The Kernel Module Package version can have an arbitrary value. The sub-package versions are composed of the main package version, followed by an underscore, and the version of the kernel source used during the build. Since sub-packages already include the support-

ed kernel's flavor in their name, the flavor is not again included in the sub-package's version. Dashes in the kernel release are replaced by underscores. The first component (main package version) can be overridden with the `-v` option of the **%kernel_module_package** macro.

- The Kernel Module Package `Release` tag can be assigned freely as required. It must be incremented at least once for each package release.
- The sub-package release numbers equal the main package's release number. It can be overridden with the `-r` option of the **%kernel_module_package macro**.
- The appropriate `Requires` and `Provides` tags are computed automatically by the rpm build as described in the RPM `Provides` and `Requires` section below. `Provides` and `Requires` tags in the spec file will only be effective for the main package.
- Kernel modules must be installed below either `/usr/lib/modules/VERSION-RELEASE-FLAVOR/extra/`. for additional modules not part of the default kernel installation or `/usr/lib/modules/VERSION-RELEASE-FLAVOR/updates/` for updates to existing kernel modules.
- Packages must be signed with a public/private key pair, and the public key of the private/public key-pair used for signing must be made known to RPM. See *Section 9.1, "Signing packages"* for details.

Kernel Module Package sub-package tags can be specified under a `%package KMP` sub-package section. The `Summary KMP` and `%description KMP` tags can be used to give summary and descriptions that differ from the main package If the `%package KMP` section is not defined, the summary and description of the main packages will be applied to the sub-packages.

The **%kernel_module_package** macro uses a default sub-package template that should work for most KMPs. This template can be overridden using the macro's `-t` option. The default template takes care of the following:

- When a KMP is installed, `depmod` is called to update module dependency information and various maps. Symbolic links pointing at the new modules are created in other kernels' `weak-modules/` directories for all compatible modules. Initial RAM disks used during boot-

ing are re-created automatically if they contain some of the added modules. Using the macro's `-b` option will force the recreation of the initial RAM disk regardless of whether the existing RAM disk contains modules with the same names as the modules being installed.

Note: RAM disk rebuild

The `-b` option simply forces a RAM disk rebuild using the existing RAM disk configuration. If the existing RAM disk configuration does not include previous versions of the new modules being installed, using the `-b` option will not include the new modules in the new RAM disk. In such cases, the Kernel Module Package spec file also needs to make appropriate RAM disk configuration changes to include the new modules. This can for example be done by creating or updating files in `/etc/dracut.conf.d`.

- When a Kernel Module Package is removed, `depmod` is called to update module dependency information and various maps. The symbolic links pointing to the modules being removed are removed as well. Initial RAM disks are re-created in case they did contain some of the removed modules.

By default, each kernel-specific sub-package will have the following list of files, which can separately be overridden with the `-f` option:

```
%defattr (-,root,root)
/usr/lib/modules/%2-%1
```

Additional sub-package preamble lines such as `Requires`, `Provides`, and `Obsoletes` tags can be specified with the `-p` option. File name arguments specified in `-f`, `-p` and `-t` should be given as absolute path names (for example `%_sourcedir/file`) and should be listed as sources. The following substitutions are defined in those files:

- %1 Flavor of the sub-package (for example: default)
- %2 Kernel release string without flavor (for example: 6.12.0-160000.5)
- %{-v*} The sub-package version
- %{-r*} The sub-package release

Some Kernel Module Packages may make sense only for some of the kernel flavors a given architecture supports. A list of flavors to exclude from the build should be passed with the `-x` option to the **%kernel_module_package** macro.

Appendix A contains an example Kernel Module Package spec file and the source code referenced by it. When this spec file and its accompanying source is built into an RPM as described in Section 7, “Building Kernel Module Packages”, the **%kernel_module_package_buildreqs** macro in the BuildRequires tag of the spec file will pull the `module-init-tools`, `kernel-source`, `kernel-syms` and `kernel-devel` packages into the build root.

Now assume that the required packages are available in SUSE Linux Enterprise Server 16.0 (kernel 6.12.0-160000.5), and that the default kernel flavor is available on that platform. Assuming a release number of “0”, `rpm` would then create the following packages:

`suse-hello-kmp-default-1.0_k6.12.0_160000.5-160000.7.1.x86_64.rpm`

The generated packages would contain the following module, and require and provide the following symbols:

Package	`suse-hello-kmp-default`

Requires	`ksym(default:__fentry__) = bdfb6dbb`
	`ksym(default:__x86_return_thunk) = 5b8239ca`
	`ksym(default:_printk) = 92997ed8`
	`ksym(default:module_layout) = fa596f96`
	`ksym(default:param_ops_int) = df399adb`

Provides	`ksym(default:exported_function) = e52d5bcf`
Modules	`/usr/lib/modules/6.12.0-160000.5-default/updates/hello.ko`

7 RPM Provides and Requires

Kernels export symbols that kernel modules use. Symbols have version checksums attached, and the checksums of the exported kernel symbols must match the checksums of the kernel symbols that the Kernel Module Package uses.

Similarly, at an RPM level, each `kernel-FLAVOR` package (for example `kernel-default`) provides the symbols and checksums that are exported by that flavor of the kernel, and each Kernel Module Package requires the specific kernel symbols and checksums needed by the module(s) that it contains. Installation of a KMP will succeed as long as an installed kernel package provides the symbols and checksums that are required by the KMP.

When modules in Kernel Module Packages export additional symbols, such symbols are mapped to the `Provides` of those packages. Modules in other Kernel Module Packages may require those symbols. As an example, assume that a Kernel Module Package provides an exported function as `_ksym(default:exported_function) = e52d5bcf_`. Any Kernel Module Package that uses this function would require this same symbol and checksum.

7.1 %kernel_module_package_buildreqs Macro

For building external modules, you must have the `kernel-FLAVOR-devel` (and `kernel-devel`) packages installed in the build environment. Using the **%kernel_module_package_buildreqs** macro in a **BuildRequires** line in spec files takes care of this. The **%kernel_module_package_buildreqs** macro specifies, among other requirements, the `kernel-syms` package. The `kernel-syms` package pulls in the `kernel-FLAVOR-devel` package (and the `kernel-devel` package) via a dependency.

SUSE updates the **%kernel_module_package_buildreqs** macro whenever there is a change in build requirements from one SUSE Linux Enterprise Server release to another. By using the macro in your spec files, you ensure compatibility across different releases of SUSE Linux Enterprise Server.

Note: Building KMPs with proper symbol version dependencies

Without the `kernel-syms` package the module build can still succeed, depending on how you do the build. However, the resulting modules will have module symbol versions disabled. Kernel Module Packages without module symbol versions will appear to match any kernel although in fact they do not. This can easily lead to very hard-to-diagnose system malfunctions.

8 Building Kernel Module Packages

In addition to the C and kernel programming skills required for writing the kernel module source code, creating proper Kernel Module Packages requires some familiarity with the `rpm` command and with build environments. For more information on kernel module building refer to the above-mentioned Linux Kernel Module Programming Guide (http://www.oreilly.com/catalog/linuxdrive3/) and the book Linux Device Drivers (http://www.oreilly.com/catalog/linuxdrive3/). Additional SUSE-specific kernel and kernel module information can be found in the README.SUSE

in the `kernel-source` package. SUSE recommends using the example package found in Appendix A as a template to reduce the complexities related to RPM. A lot of additional information on RPM can be found at http://www.rpm.org/, including a reference to the excellent Maximum RPM.

SUSE strongly recommends using the kernel build infrastructure (`kbuild`) for building and installing the kernel modules, as done in the example package. `kbuild` is documented in the file `/usr/src/linux/Documentation/kbuild/` from the `kernel-source` package. Trying to emulate `kbuild` will lead to various problems including miscompilations and missing or wrong symbol versions, and increased support load because of subtle breakages.

To achieve consistent and reproducible builds in a defined environment independent of the software installed on the system used for building, use the build script from the `build.rpm` package found in the SLES Package Hub extension. This script sets up a build environment from the available packages provided to the script. The package build process enters that environment using `chroot` (see the chroot(1) manual page at https://linux.die.net/man/1/chroot). When building Kernel Module Packages with `build.rpm`, the following options of the build script are particularly relevant:

--root directory

Define the directory in which to set up the build environment. Defaults to the `BUILD_ROOT` environment variable, and to `/var/tmp/build-root` if unset.

--RPMs path1[:path2:...]

Define where build will look for packages for constructing the build environment. The directories are searched recursively. Packages found earlier in the path have precedence over packages found later, similar to how the PATH environment variable works. Defaults to the `BUILD_RPMS` environment variable, and to `/media/dvd/suse` if unset. The `--rpms` option must only be specified once.

--clean, --no-init

Reconstruct the build environment entirely from scratch (`--clean`), or start the build without initializing the build environment (`--no-init`), which skips checking whether all packages in the build environment are up-to-date.

Build stores the created packages below `home/abuild/rpmbuild/` in the build environment.

On dual-architecture machines, packages for the other supported architecture can be built by running the build script inside an architecture selector. On Intel 64/AMD64, the selector is called *linux32*, on IBM POWER this is *ppc32*, and on IBM Z the selector is called *s390x*. The same build environment cannot be reused for different architectures unless it is reinitialized with build's `--clean` option.

See the build(1) manual page for further information.

9 Signing

Signing (as applied to a piece of software) is the process of digitally tagging the software to verify the author and guarantee that the software has not been altered after it has been signed. SUSE Linux Enterprise Server includes utilities to sign and validate signatures on packages and repositories. In addition, SUSE Linux Enterprise Server includes technology to sign and validate signatures on kernel modules.

The following sections describe how to sign packages and kernel modules. The topic of repository signing is beyond the scope of this document.

9.1 Signing packages

All packages that are provided in SUSE Linux Enterprise Server are digitally signed with the SUSE Build key. SolidDriver packages that are built on the SolidDriver build server by the SUSE SolidDriver team are automatically signed with the SUSE SolidDriver key. For more information, visit https://drivers.suse.com/doc/Usage/Package_Signing_Key.html#package-signing-key. Partners who build and/or provide their own packages are encouraged to sign them with official company keys to allow customers to validate the integrity of the package.

For testing purposes, developers can sign packages using their own personal and/or test keys. RPM uses GnuPG (gpg) for signing. To sign packages, a private/public key pair must be installed on the GNU Privacy Guard (GPG) keyring of the signing user (see the `--gen-key` option in the gpg(1) manual page at https://linux.die.net/man/1/gpg). Then the following command can be used to sign a package:

```
$ rpm --eval “%define _signature gpg” \
      --eval “%define _gpg_name build@suse.com” \
      --addsign package.rpm
```

(Replace **build@suse.com** with the identity that identifies your signing key).

A package can only be signed once. Another `--addsign` operation will replace an existing old signature, and will add the new one.

The public key used for signing must then be exported into a file with the command:

```
$ gpg --armor --export build >build-pubkey.txt
```

Next, import the key into the RPM database with the command:

```
$ rpm --import build-pubkey.txt
```

You can verify that both package signing and key import have succeeded with RPM's `--checksig` option (note the "gpg" in the output):

```
$ rpm --checksig package.rpm
package.rpm: (sha1) dsa sha1 md5 gpg OK
```

The public key exported to `build-pubkey.txt` must be delivered to customers in a way that they will trust. It must be imported into the RPM database on systems on which the signed packages are to be installed.

9.2 Signing module object files

Although using signed packages and other operating system security features can secure an installed and running system, they cannot prevent system subversion before the operating system has booted. To address pre-OS security concerns, the UEFI Secure Boot specification (see http://www.uefi.org) details a protocol to prevent the loading of boot loaders or kernels (including modules) that are not signed with an approved digital key stored in the system firmware.

The UEFI Secure Boot specification allows for variation in implementation. A simple way to implement secure boot is to ensure that the base system (as provided by the system vendor) contains all the keys that will be used by the boot loader, the operating system, and any drivers. But having the system vendor simply place all needed keys into the firmware is not a full solution, as it does not give appropriate control to the system user or owner. SUSE's secure boot implementation addresses this control issue by extending the secure-boot-enabled EFI shim loader to accept keys that have been approved by the system owner. Thus, if there is a need to load a module with an unrecognized key, the key can be added to the "approved key" database (reboot and system-owner approval required).

9.2.1 Creating a key and certificate

Module signing requires having access to a digital key and certificate. Official keys and certificates are generally maintained by an organization's security team or by build services (such as the Open Build Service or the SolidDriver build service). Developers and packagers can also generate their own keys and certificates for example for testing purposes.

To create a key and certificate using the `openssl req` command, type the following:

```
export USER="your company name"
openssl req -new -x509 -newkey rsa:2048 -sha256 -keyout key.asc -out cert.der \
            -outform der -nodes -days 4745 -addext "extendedKeyUsage=codeSigning" \
            -subj "/CN=$USER/"
```

The above sequence of commands will create a `key.asc` key file and a `cert.der x509` certificate in the current working directory. The `4745` option generates a certificate which will be valid for 13 years.

Note: Extended Key Usage (EKU) setting

The `-addext "extendedKeyUsage=codeSigning"` openssl option is required to accept the certificate for signature verification of kernel modules.

The `-addext` option requires OpenSSL version 1.1.1 or later. For earlier versions of OpenSSL, consult the documentation for setting the EKU.

9.2.2 Signing modules during packaging

Signing modules as part of the packaging process requires making several changes to the KMP spec file. The spec file template in Appendix A.1 includes these changes, which are:

1. List the certificate file as a **%Source** file. The top-level directory of the build structure (where the spec file is located) should include both a private key file and a certificate file. The spec file should list the certificate as a **%Source** file. The spec file should not list the key file (since the private key should not be included in the source KMP).

Note: Naming

To be recognized by the kernel Makefile, the key file must be named `signing_key.priv` and the certificate file must be named `signing_key.x509`. The example above describes how to use the openssl req command to create a `key.asc` key file and a `cert.der` certificate file; to use these files at packaging-time, they should be renamed to `signing_key.priv` and `signing_key.x509`.

2. Invoke the %kernel_module_package macro with the -c %_sourcedir/signing_key.x509 option to generate a <name>-ueficert package which installs the certificate and calls the mokutil utility to enroll the public key. The actual module signing is handled in the %install section of the spec file.
3. Add %install section code to invoke the kernel-sign-file file to sign the modules.

Note: Own keys and certificates

The Appendix A.1 sample spec file is designed to be used by developers and packagers who provide their own keys and certificates and their own build environments. Developers or packagers who use the Open Build Service should modify their spec file as described in the pesign-obs-integration package README (https://github.com/openSUSE/pesign-obs-integration/blob/master/README).

9.3 Signing modules in an existing KMP

Often the developer or packager does not have access to corporate signing keys and signing infrastructure. Those exist behind highly secure environemts and processes. Therefore the keys are not available at package build time and signing takes place as a secondary operation.

The `pesign-obs-integration` package in SUSE Linux Enterprise Server provides a `modsign-repackage` utility that can be used to sign kernel modules in an existing KMP. `modsign-repackage` unpacks the original RPM, signs any included modules and re-creates the RPM. It also creates a second `<NAME>-ueficert RPM` that installs the certificate and calls the `mokutil` utility to enroll the public key. The re-packaged RPM will have a dependency on the `<NAME>-ueficert` RPM, ensuring that the certificates will be installed at the same time as the module(s).

To repackage the `suse-hello-kmp-default-1.0_k6.12.0_160000.5-160000.7.1.x86_64.rpm` package with the key and the certificate created above, do the following:

```
$ modsign-repackage -c ./cert.der -k ./key.asc ./suse-hello-kmp-
default-1.0_k6.12.0_160000.5-160000.7.11.0_k4.4.73_5-0.x86_64.rpm
```

The above command creates the following directories and files in the current working directory:

```
./RPMS/
   x86_64/
     suse-hello-kmp-default-1.0_k6.12.0_160000.5-160000.7.11.0_k4.4.73_5-0.x86_64.rpm
     suse-hello-ueficert-1.0-0.x86_64.rpm
```

To see the module signature, unpack the repackaged Kernel Module Package and use the `modinfo` command to view the module signature info:

```
rpm2cpio suse-hello-kmp-default-1.0_k6.12.0_160000.5-160000.7.11.0_k4.4.73_5-0.x86_64.rpm
 | cpio -idv
modinfo ./usr/lib/modules/6.12.0-160000.5-default/updates/hello.ko | grep signature
```

Note: rpmbuild must be installed

modsign-repackage requires the /usr/bin/rpmbuild utility which is provided by the rpmbuild package.

After the process is complete, you can use `modsign-verify` to confirm that the modules inside the new package are correctly signed with the new key.

Module signatures vs. package signatures

It is crucial to distinguish between two different types of signatures:

- Module signature: The signature on the kernel module files (.ko) inside the package. This is what `modsign-repackage` and `modsign-verify` manage.
- Package signature: The signature on the KMP file itself (for example, an `rpm` or `deb` package signature) that verifies the package's integrity.

The `modsign-repackage` tool only signs the modules. During this process, any existing package-level signature is removed. This means you will almost always need to re-sign the entire package file after running the tool.

Resigning workflow

Follow these steps to properly resign the kernel modules within a KMP:

1. Prepare the KMP files that contain the kernel modules you need to resign.
2. Resign Modules: Run the `modsign-repackage` command on the KMP using the desired signing keys. This creates a new, repackaged KMP file where all kernel modules inside are signed with the new key.
3. Verify Modules: Run `modsign-verify` on the newly created KMP file. This utility checks the signatures of the .ko modules inside the package to confirm they are valid.
4. Resign the Package: Since `modsign-repackage` removed the original package-level signature (from step 3), the new KMP file itself is now unsigned. You should re-sign the entire package file (for example, using `rpmsig` for an RPM) to ensure package integrity and allow package managers to verify it.

9.4 Installing secure-boot-enabled KMPs

As discussed above, secure-boot-enabled KMPs include an additional `<NAME>-ueficert` package to install the certificate and enroll the public key. The `<NAME>-kmp-<flavor>` packages require the `<NAME>-ueficert` package.

After the `<NAME>-ueficert` package is installed, the system must be rebooted and the newly-enrolled key approved by the system owner before the key (and thus the signed modules) can be used.

The `mokutil` utility can also be used on its own to view and manage keys in the key database.

10 Installing a Driver Update Disk (DUD) in Agama

A Driver Update (DUD) is a mechanism used to patch or extend the Agama installer itself. The DUD is applied to the installation environment during boot time.

DUDs should be used if modules are needed during installation. Otherwise providing RPMs for post installation installing/updating is fine.

Supported Formats: A DUD can be provided in two supported formats:

- Special DUD archives that can be created with the `mkdud` utility.
- RPM packages: Note that, when using this format, the kernel modules will only be installed and loaded in the installer. The RPM and kernel modules will not be installed and loaded on the target system.

DUD, RPM(s), installation process:

1. Host the DUD, RPM(s) on a URI the target system can reach.
2. Start the installation using the standard SUSE Linux media.
3. Highlight the "Install SLE 16" menu option and press 'e' to edit the boot loader options.
4. On the line starting with "linux", add `inst.dud=URI_OF_FILE` to the end of the line with `URI_OF_FILE` pointing to the DUD, or RPMs. Examples:
   - instdud = https://myserver.mydoman/path_to_dud/dud.iso
   - instdud = usb:///dud.img
   - instdud = http://download.packages.com/some-project/some.rpm
5. Boot the installer and continue the installation as usual.

Note: Agama URL Formats

Note: Besides the http/https URL format, Agama also supports other URL formats. See the URLs section in the Agama documentation: https://agama-project.github.io/docs/user/reference/urls

Also note that, unlike `linuxrc` of SUSE Linux Enterprise Server 15, Agama does not traverse file systems or devices to detect and load DUD's. With Agama, the URI must point to a single file (RPM, or DUD image) that will be loaded and processed by Agama.

To create a DUD by yourself, refer to the following document: https://github.com/openSUSE/mkdud

For more Agama DUD boot options, refer to: https://agama-project.github.io/docs/user/reference/boot_options

11 Installing the system and Kernel Module Packages

Initial system installation is performed by Agama using various installation media (CDs or DVDs, network locations, etc.). As previously noted, support for additional hardware that the installation media do not provide can be added with Driver Update Disks. This is essential for enabling hardware required for booting, such as storage controllers.

Driver Update Disks provide two types of modules: Those which the kernel that runs the installation uses, and those which are installed onto the target system. Both types of modules are provided by including Kernel Module Packages on the update media. Following the initial installation, additional driver packages can be installed or updated using Zypper.

Note: initrd

Any drivers required for getting to and accessing the root file system must be part of the initial RAM disk. The SUSE Linux Enterprise Server installer will automatically include necessary kernel modules in the `initrd` created during installation. But when Kernel Module Packages are installed manually or are updated, it can be necessary to explicitly rebuild the `initrd` to include the new modules. As noted in *Section 6, "Kernel Module Packages"*, a Kernel Module Package will automatically rebuild the `initrd` in its `%postinstall` script if the module being installed is already part of the existing disk or if the `-b` option to the **%kernel_module_package** macro is used.

The `-b` option forces a RAM disk rebuild using the existing RAM disk configuration. If that configuration does not already include the module being installed, using the -b option alone will not add it. In such cases, the Kernel Module Package must also modify the RAM disk configuration (for example, by creating or updating files in `/etc/dracut.conf.d`) to ensure the new module is included.

11.1 Kernel updates and Kernel Module Packages

After all software repositories that should be checked for updates have been added, the package manager will automatically detect when new kernel packages and new Kernel Module Packages become available. The dependencies between those packages will ensure that the installed kernel packages match the installed Kernel Module Packages.

Appendix A: Sample source for the "SUSE Hello" Kernel Module Package

The following sample is described in *Section 6, "Kernel Module Packages"*. For a sample spec file that signs modules during packaging, see Appendix A.1.

suse-hello.spec

```
# norootforbuild

Name:           suse-hello
Version:        1.0
Release:        0
Summary:        Sample Kernel Module Package
License:        GPL-2.0
Group:          System/Kernel
Source0:        %{name}-%{version}.tar.bz2
BuildRequires:  %kernel_module_package_buildreqs
BuildRoot:      %{_tmppath}/%{name}-%{version}-build

%kernel_module_package

%description
This package contains the hello.ko module.

%prep
%setup
set -- *
mkdir source
mv "$@" source/
mkdir obj

%build
for flavor in %flavors_to_build; do
       rm -rf obj/$flavor
       cp -r source obj/$flavor
       make -C %{kernel_source $flavor} modules M=$PWD/obj/$flavor
done

%install
export INSTALL_MOD_PATH=$RPM_BUILD_ROOT
export INSTALL_MOD_DIR=updates
for flavor in %flavors_to_build; do
       make -C %{kernel_source $flavor} modules_install M=$PWD/obj/$flavor
done
```

The following two files should be compressed to form the `suse-hello-1.0.tar.bz2` TAR archive referenced as **Source0** in the `suse-hello.spec` file above.

suse-hello-1.0/Kbuild

```
obj-m       := hello.o
hello-y     += main.o
```

suse-hello-1.0/main.c

```
/*
 * main.c - A demo kernel module.
 *
 * Copyright (C) 2003, 2004, 2005, 2006
 * Andreas Gruenbacher <agruen@suse.de>, SUSE Labs
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation.
 *
 * A copy of the GNU General Public License can be obtained from
 * http://www.gnu.org/.
 */

#include <linux/module.h>
#include <linux/init.h>

MODULE_AUTHOR("Andreas Gruenbacher <agruen@suse.de>");
MODULE_DESCRIPTION("Hello world module");
MODULE_LICENSE("GPL");

int param;

module_param(param, int, 0);
MODULE_PARM_DESC(param, "Example parameter");

void exported_function(void)
{
        printk(KERN_INFO "Exported function called.\n");
}
EXPORT_SYMBOL_GPL(exported_function);

int __init init_hello(void)
{
        printk(KERN_INFO "Hello world.\n");
        return 0;
}
```

```
void __exit exit_hello(void)
{
        printk(KERN_INFO "Goodbye world.\n");
}

module_init(init_hello);
module_exit(exit_hello);
```

Appendix A1: Sample spec file for signing modules during packaging

The following spec file can be used to sign modules during packaging as described in *Section 9.2.2, "Signing modules during packaging"*.

suse-hello.spec

```
# norootforbuild

Name:           suse-hello
Version:        1.0
Release:        0
Summary:        Sample Kernel Module Package
License:        GPL-2.0
Group:          System/Kernel
Source0:        %{name}-%{version}.tar.bz2
# Required to sign modules:  Include certificate named “signing_key.x509”
# Build structure should also include a private key named “signing_key.priv”
# Private key should not be listed as a source file
Source1:        signing_key.x509
BuildRequires:  %kernel_module_package_buildreqs
BuildRoot:      %{_tmppath}/%{name}-%{version}-build

# Required to sign modules:  The -c option tells the macro to generate a
# suse-hello-ueficert subpackage that enrolls the certificate
%kernel_module_package -c %_sourcedir/signing_key.x509

%description
This package contains the hello.ko module.

%prep
%setup
# Required to sign modules:  Copy the signing key to the build area
cp %_sourcedir/signing_key.* .
set -- *
mkdir source
mv "$@" source/
mkdir obj

%build
for flavor in %flavors_to_build; do
       rm -rf obj/$flavor
       cp -r source obj/$flavor
       make -C %{kernel_source $flavor} modules M=$PWD/obj/$flavor
done
```

```
%install
export INSTALL_MOD_PATH=$RPM_BUILD_ROOT
export INSTALL_MOD_DIR=updates
for flavor in %flavors_to_build; do
       make -C %{kernel_source $flavor} modules_install M=$PWD/obj/$flavor
       # Required to sign modules:  Invoke kernel-sign-file to sign each module
       for x in $(find $INSTALL_MOD_PATH/usr/lib/modules/*-$flavor/ -name '*.ko'); do
              /usr/lib/rpm/pesign/kernel-sign-file -i pkcs7 sha256 $PWD/obj/$flavor/
signing_key.priv $PWD/obj/$flavor/signing_key.x509 $x
       done
done
```

Appendix B: Changes and references

Documentation updates

References

- SUSE SolidDriver Documentation: Kernel Module Packages Manual for Code 11, Kernel Module Packages Manual
- SUSE SolidDriver Documentation: SUSE Kernel ABI Stability, https://drivers.suse.com/doc/SolidDriver/SUSE_Kernel_ABI_Stability.html
- Greg Kroah-Hartman: The Linux Kernel Driver Interface, http://www.kroah.com/log/linux/stable_api_nonsense.html (also provided as stable_api_nonsense.txt in the upstream kernel source tree)
- Andreas Gruenbacher, Michal Marek: Working With The SUSE 2.6.x and 3.x Kernel Sources (provided as README.SUSE in SUSE kernel-source packages)
- Peter Jay Salzman, Michael Burian, Ori Pomerantz: The Linux Kernel Module Programming Guide, http://www.tldp.org/LDP/lkmpg/2.6/html/index.html
- Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman: Linux Device Drivers, Third Edition, February 2005, http://lwn.net/Kernel/LDD3/
- UEFI Specification: http://www.uefi.org/specifications
- Creating Add-Ons: https://www.novell.com/developer/creating_add-ons.html

12 Legal notice

13 GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all

Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
   Permission is granted to copy, distribute and/or modify this document
   under the terms of the GNU Free Documentation License, Version 1.2
```

```
   or any later version published by the Free Software Foundation;
   with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
   A copy of the license is included in the section entitled “GNU
   Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “ with…Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
   Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.