SUSE

SUSE Best Practices

Systems Management

Kernel Module Packages Manual

SUSE-based Distributions

SUSE Linux Enterprise Server 11
SUSE Linux Enterprise Server 12 GA and SP1

Document Owner: Ann Davis, Senior Software Engineer (SUSE)

SUSE

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.

Contents

1 Scope

This version of the Kernel Module Packages Manual applies to:

- the Code 11 code base, which includes openSUSE 11.1 and newer, SUSE Linux Enterprise Server or Desktop 11 (along with service packs)
- SUSE Linux Enterprise Server or Desktop 12 (along with service packs)
- All products and extensions based on SUSE Linux Enterprise Server 11 and SUSE Linux Enterprise Server 12

Versions of this document for Code 9 and Code 10 are available as well and can be found at http://www.novell.com/developer/kernel_module_packages_manuals.html

This document's Appendix B highlights Code 10 to Code 11 changes and secure boot changes for SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop 11 SP3 and later versions.

2 Background

The Linux kernel supports adding functionality at runtime through kernel-loadable modules. It includes more than 1500 modules, about 75 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-xen**) will not match, and symbol versions of the same kernel package on

different architectures (for example **kernel-default** on i386 versus x86_64) 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 its multi-year lifecycle, products like SUSE Linux Enterprise Server undergo continuous changes. In addition, different kinds of updates like service packs (SPs), 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, 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” at https://drivers.suse.com/doc/SolidDriver/SUSE_Kernel_ABI_Stability.html and “The Linux Kernel Driver Interface” from Greg Kroah-Hartman at http://www.kroah.com/log/linux/stable_api_nonsense.html (also provided as **stable_api_nonsense.txt** in the upstream kernel source tree).

SUSE-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.

3 Kernel Packages

All SUSE products based on kernels 2.6.x, 3.0.x, and 4.4.x contain a set of kernel packages that share the same version and release number; they are built from the same kernel sources. These packages are:

`kernel-FLAVOR`, `kernel-FLAVOR-base`

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

`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.

`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-source and kernel-devel packages used for compiling external modules must match each other exactly.

Starting with SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop 11 SP1, the `kernel-syms` package is a place-holder 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.

4 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 README.SUSE in the kernel-source package.

When built, kernel module binaries are installed below `/lib/modules/VERSION-RELEASE-FLAVOR` on the file system (example: `/lib/modules/3.0.101-63-default` for the `kernel-default-3.0.101-63` package). Different kernels have different module directories, and will usually not see each others 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 `/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 `/lib/modules/VERSION-RELEASE-FLAVOR/updates/`. Other add-on modules can be stored below `/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 `/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 `/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 install modules into weak-updates or directories.

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

5 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] [-c module-signing-certificate]

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 supported 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** 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 **`rpm`** as described in the RPM Provides and Requires section below. Requires and Provides tags in the spec file will only be effective for the main package.

- Kernel modules must be installed below `/lib/modules/VERSION-RELEASE-FLAVOR/updates/`.

- 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 the section “Signing Kernel Module Packages” below for details.

- If a kernel module package is intended to support UEFI Secure Boot, the modules in the package must be signed with a private key and the corresponding public key must be provided at package installation time. The macro's `-c` option provides a way to create a separate package to provide the UEFI certificate with the required public key. Other spec file changes can be included to actually sign the modules. The `-c` option is available starting with SUSE Linux Enterprise Server 11 SP3.

The **%description** tag will be applied to both the main package and the sub-packages.

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

- When a Kernel Module Package package 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 booting 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. The `-b` option is available starting with SUSE Linux Enterprise Server 11 SP2).

- 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)
/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: 2.6.27.8-1)
- %{-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 x86_64 RPM as described in section *Section 7, "Building Kernel Module Packages"*, the **BuildRequires** tag in the spec file will pull the `module-init-tools`, `kernel-source`, `kernel-syms` and `kernel-devel` packages into the build root.

Note: Dependency

The **%kernel_module_package_buildreqs** macro does not need to explicitly list "kernel-source" since the `kernel-syms` package has a dependency on the `kernel-source` package.

Now assume that the required packages are available in SUSE Linux Enterprise Server 11 SP4 (kernel 3.0.101-63), and that the default, trace, and Xen kernel flavors are available on that platform. Assuming a release number of "0", `rpm` would then create the following packages:

- `suse-hello-kmp-default-1.0_3.0.101_63-0.x86_64.rpm`
- `suse-hello-kmp-trace-1.0_3.0.101_63-0.x86_64.rpm`
- `suse-hello-kmp-xen-1.0_3.0.101_63-0.x86_64.rpm`

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

TABLE 1: PACKAGE INFORMATION

Package	Requires	Provides	Modules
suse-hello-kmp-default	kernel(default:kernel_printk) = f06347de1657cfa8 ...	suse-hello-kmp = 1.0_3.0.101_63 ksym(default:exported_function) = e52d5bcf suse-hello-kmp-default = 1.0_3.0.101_63-0	/lib/modules/3.0.101-63-default/ updates/hello.ko
suse-hello-kmp-trace	kernel(trace:kernel_printk) = 7a87f1ab614120c0 ...	suse-hello-kmp = 1.0_3.0.101_63 ksym(trace:exported_function) = e52d5bcf suse-hello-kmp-trace = 1.0_3.0.101_63-0	/lib/modules/2.6.27.8_1.0-trace/ updates/hello.ko
suse-hello-kmp-xen	kernel(xen:kernel_printk) = 69e7f9b015806194 ...	suse-hello-kmp = 1.0_3.0.101_63 ksym(xen:exported_function) = e52d5bcf	/lib/modules/2.6.27.8_1.0-xen/ updates/hello.ko

Package	Requires	Provides	Modules
		suse-hello-kmp-xen = 1.0_3.0.101_63-0	

Note: “k” Character

KMPs built for SUSE Linux Enterprise Server 12 releases will include the “k” character before the kernel version in the KMP version string. As an example, the suse-hello-kmp-default package built for SUSE Linux Enterprise Server 12 SP2 (kernel 4.4.21-69) would have the full file name suse-hello-kmp-default-k4.4.21_69-0.x86_64.rpm.

6 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. In SUSE Linux Enterprise Server 11 releases, kernels do not provide each kernel symbol individually; instead they group symbols together into classes (called “symsets”) and provide checksums of these classes.

Note: Number of Classes

The number of classes is much smaller than the full number of symbols provided by the kernel.

Similarly, SUSE Linux Enterprise Server 11 Kernel Module Packages require checksums for classes rather than for individual symbols. In SUSE Linux Enterprise Server 12 releases, kernels provide checksums for each individual symbol. Therefore, SUSE Linux Enterprise Server 12 Kernel Module Packages require checksums for individual symbols.

When modules in Kernel Module Packages export additional symbols, such symbols are mapped to per-symbol **Provides** of those packages. Modules in other Kernel Module Packages can require those symbols; they would also do so on a per-symbol basis. 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 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 **rpm** and with build environments. For more information on kernel module building refer to the above-mentioned “Linux Kernel Module Programming Guide” and the “Linux Device Drivers” book. 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 /usr/src/linux/Documentation/kbuild/ from the kernel-source package. Trying to emulate kbuild will lead to various problems including mis-compilations 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. This script sets up a build environment from the RPM packages the script is pointed at. The packages are then built in this environment using **chroot** (see the chroot(1) manual page). All SUSE packages are built using the same mechanism. 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 `usr/src/packages/` 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 x86_64, the selector is called linux32, on ppc64 this is ppc32, and on s390x the selector is called s390. 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.

Note: Building External Modules

For building external modules, you need to have both the kernel-source and kernel-<flavor>-devel packages installed in the build environment. The BuildRequires line in spec files takes care of this: the **%kernel_module_package_buildreqs** macro specifies the kernel-syms package, which pulls in the kernel-source package and the kernel-<flavor>-devel packages because of its dependency on them. Note that without the kernel-syms the module build can still succeed depending on how you do the build, but 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 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 since it was signed. SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop include utilities to sign and validate signatures on packages and repositories. In addition, SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop 11 SP3 and later versions include 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.

8.1 Signing Packages

All packages that are provided in SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop are digitally signed with the SUSE Build key. SolidDriver/PLDP packages that are built on the SolidDriver Build Server by the SUSE SolidDriver team are automatically signed with the SUSE SolidDriver/PLDP key (see 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 their own keys.

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 (replace build@suse.com with the identity that identifies your signing key):

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

Note that 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
```

Then, 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.

8.2 Signing Module Object Files *(UEFI Secure Boot)*

Although using signed packages and other OS security features can secure an installed and running system, they cannot prevent system subversion before the OS has booted. To address pre-OS security concerns, the UEFI 2.2 Secure Boot specification (see http://www.uefi.org/specs (http://www.uefi.org/specs)) 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 OS, 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/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).

Note: Module Without Signature

If a module has no signature, it cannot be loaded on a secure-boot-enabled system. Users who wish to load unsigned modules on SUSE Linux Enterprise Server/SUSE Linux Enterprise Desktop 11 SP3 or later versions must disable secure boot.

8.2.1 Creating a Key and Certificate

There are several ways to sign modules, but all methods require providing a key and certificate. Official keys and certificates can be provided by an organization's security team or by build services (such as the Open Build Service or the SolidDriver/PLDP Build Service). Developers and packagers can also generate their own keys and certificates 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 -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.

8.2.2 Signing an Existing KMP

The `pesign-obs-integration` package in SUSE Linux Enterprise Server 11 SP3 and later versions 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 use **`modsign-repackage`** with the **key.asc** key and the **cert.der** certificate created above to repackage a `./suse-hello-kmp-default-1.0_3.0.101_63-0.x86_64.rpm` package, do the following:

```
modsign-repackage -c ./cert.der -k ./key.asc ./suse-hello-kmp-
default-1.0-3.0.101_63-0.x86_64.rpm
```

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

```
    ./RPMS/
      x86_64/
          suse-hello-kmp-default-1.0-3.0.101_63-0.x86_64.rpm
          suse-hello-ueficert-1.0-0.x86_64.rpm
```

8.2.3 Signing Modules During Packaging

Signing modules as part of the packaging process requires making a few changes to the KMP spec file template. The spec file template in Appendix A includes these changes along with conditionals to ensure that the changes will apply only when building KMPs for SUSE Linux Enterprise Server or SUSE Linux Enterprise Desktop 11 SP3 or later versions. The changes 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. Add code to determine the build target's service pack level. This is done by adding the `sles-release` (or `sled-release`) package to **%BuildRequires** and then defining a **%sle_version** macro based on the contents of the `/etc/SuSE-release` file.

3. For SUSE Linux Enterprise Server 11 SP3 and later versions, invoke the **%kernel_module_package** macro with the `-c %_sourcedir/signing_key.x509` option to specify the certificate (and thus the key) to use in signing. Using the kernel_module_package `-c` option does not cause any module signing; it simply ensures the creation of 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.

4. For SUSE Linux Enterprise Server 11 SP3 and later versions, add **%install** section code to ensure that the modules will get signed. This is done by setting the CONFIG_MODULE_SIG_ALL kernel configuration parameter. When CONFIG_MODULE_SIG_ALL is set, the `make modules_install` step automatically includes module signing.

Note: Own Keys and Certificates

The Appendix A sample spec file is designed to be used by developers and packagers who provide their own keys and certificates. Partners who use the Open Build Service will not need to provide keys and certificates and thus should use the spec file in Appendix A.1.

8.2.4 Installation of 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.

9 Deploying Kernel Module Packages

Kernel Module Packages can be distributed on Driver Update Disks, as add-on products, in full Bootable Driver Kits/Installation Kits, or simply as stand-alone RPMs.

If a Kernel Module Package's driver is required to boot an installation kernel, the Kernel Module Package should be provided on a Driver Update Disk (DUD) or in a Bootable Driver Kit/Installation Kit. Otherwise, it is recommended to provide Kernel Module Packages as add-on products complete with URL(s) for functioning update sites.

10 System Installation and Kernel Module Packages

Initial system installation is carried out by YaST from some installation media (CDs or DVDs, network locations, etc.). As noted above, support for additional hardware that the installation media do not provide can be added with Driver Update Disks or Bootable Driver Kits/Installation Kits. This is most important to enable hardware needed for booting, such as storage controllers.

Update media such as Driver Update Disks and Bootable Driver Kits/Installation Kits provide two kinds of modules: those which the kernel that runs the installation uses, and those which are installed onto the final target system. Both types of modules are provided by including Kernel Module Packages on the update media. In addition, update media can contain scripts which are run at specific times during the installation. The “Update Media HOWTO” at ftp://ftp.suse.com/pub/people/hvogel/Update-Media-HOWTO describes in more detail what a Driver Update Disk must contain to work.

After the initial YaST installation, additional driver packages can be installed using any of the mechanisms for installing RPM packages (YaST Add-on Products, YaST Software Management, YaST Online Update, the **`rpm`** command, etc.). The add-on product format supports the ability to register the system for an update site.

Note: initrd

Any drivers required for getting to and accessing the root file system must be part of the initial RAM disk (initrd). YaST will automatically include necessary kernel modules in the initrd created during installation. But when Kernel Module Packages are installed by hand or updated, it can be necessary to explicitly rebuild the initrd to include the new modules. A Kernel Module Package will automatically rebuild the initrd in its **%postinstall** script if the module being installed is already part of the existing initrd or if the `-b` option to the **%kernel_module_package** macro is used.

The `-b` option simply forces an initrd rebuild using the existing initrd configuration: If the existing initrd configuration (for example, the **/etc/sysconfig/kernel INITRD_MODULES** variable) does not include the module being installed, then simply using the `-b` option will not include the new module in the new initrd. In such cases, the Kernel Module Package also needs to make appropriate initrd configuration changes (such as updating INITRD_MODULES) to include the new module.

11 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.

12 Appendix A: Sample Source for `suse-hello` Kernel Module Package

The following sample is described in the section *Section 5, "Kernel Module Packages"*. The spec file includes conditional code that will build secure-boot-enabled packages for SUSE Linux Enterprise Server 11 SP3 and later versions and non-secure-boot-enabled packages for SUSE Linux Enterprise Server 11 – SUSE Linux Enterprise Server 11 SP2. The conditional code is shown in bold and can be removed if there is no need to build secure-boot-enabled packages.

When using the sample to build secure-boot-enabled packages, the build structure must also include `signing_key.priv` and `signing_key.x509` files as described in the *Section 8.2.3, "Signing Modules During Packaging"* section above. The `signing_key.*` files must be located in the same directory as the spec file.

suse-hello.spec

```
# norootforbuild

Name:                   suse-hello
BuildRequires:  %kernel_module_package_buildreqs
# Required to support secure-boot:  Include sles-release in order to determine
# service-pack version
BuildRequires:  sles-release
License:                GPL
Group:          System/Kernel
Summary:                Sample Kernel Module Package
Version:                1.0
Release:                0
Source0:                %{name}-%{version}.tar.bz2
# Required to support secure-boot:  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
BuildRoot:              %{_tmppath}/%{name}-%{version}-build

# Required to support secure-boot:  Determine service-pack level
%define sle_version %(tr '\\n' ' ' < /etc/SuSE-release | sed -rn 's/^SUSE Linux
Enterprise ([A-z]+) ([0-9]+).*PATCHLEVEL = ([0-9]+).*$/\\2\\3/p')

# Required to support secure-boot:  The -c option tells the macro to generate a
# suse-hello-ueficert subpackage that enrolls the certificate
%if 0%{?sle_version} > 112
%kernel_module_package -c %_sourcedir/signing_key.x509
%else
%kernel_module_package
%endif

%description
This package contains the hello.ko module.

%prep
%setup
# Required to support secure-boot:  Copy the signing key to the build area
%if 0%{?sle_version} > 112
cp %_sourcedir/signing_key.* .
```

```
%endif
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
     # Required to support secure-boot:  By default, kernel modules are not
      # signed by make. The CONFIG_MODULE_SIG_ALL=y setting overrides this for
      # flavors with module signing enabled.
      unset CONFIG_MODULE_SIG_ALL
      if grep '^CONFIG_MODULE_SIG=y' %{kernel_source $flavor}/.config; then
         export CONFIG_MODULE_SIG_ALL=y
      fi
      make -C %{kernel_source $flavor} modules_install M=$PWD/obj/$flavor
done

%changelog
* Fri Apr 27 2017 – andavis@suse.com
- Typo fixes; remove excluded flavors from kernel_module_package macro line
* Wed Apr 24 2013 - mmarek@suse.cz
- Sign the module by a supplied keypair.
* Tue Dec 22 2008 - andavis@suse.com
- Updated to reflect CODE 11 changes and LF standard spec file work.
* Sat Jan 28 2006 - agruen@suse.de
- Initial package.
```

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);
```

13 Appendix A.1: Sample Spec File for Use with SUSE Build Services

Packagers who use the Open Build Service (see https://build.opensuse.org/) to create KMPs can take advantage of the fact that this build service provides keys and certificates for package and module signing. The following spec file can be used in the Open Build Service to build secure-boot-enabled KMPs for SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop 11 SP3 and later versions. It can also be used to build non-secure-boot-enabled KMPs for all versions from SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop 11 up to SUSE Linux Enterprise Server and SUSE Linux Enterprise Desktop 11 SP2. The secure-boot conditional code is emphasized in bold.

A sample package similar to this one is available in the Open Build Service at https://build.opensuse.org/package/show/home:andavis:PLDP/suse-hello-1.0

suse-hello.spec

```
# The following line tells the buildservice to save the project certificate as
# %_sourcedir/_projectcert.crt

# needssslcertforbuild

Name:                   suse-hello
BuildRequires:  %kernel_module_package_buildreqs
# Required to support secure-boot:  Include sles-release in order to determine
# service-pack version
BuildRequires:  sles-release
License:                GPL
Group:              System/Kernel
Summary:                Sample Kernel Module Package
Version:                1.0
Release:                0
Source0:                %{name}-%{version}.tar.bz2
BuildRoot:              %{_tmppath}/%{name}-%{version}-build

# Required to support secure-boot:  Determine service-pack level
%define sle_version %(tr '\\n' ' ' < /etc/SuSE-release | sed -rn 's/^SUSE Linux
Enterprise ([A-z]+) ([0-9]+).*PATCHLEVEL = ([0-9]+).*$/\\2\\3/p')

# Required to support secure-boot:  The -c option tells the macro to generate a
# suse-hello-ueficert supackage that enrolls the certificate
# The _projectcert.crt certificate is provided by the build service
%if 0%{?sle_version} > 112
%kernel_module_package -c %_sourcedir/_projectcert.crt
```

```
%else
%kernel_module_package
%endif

%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 BRP_PESIGN_FILES variable must be set to a space separated list of
# directories or patterns matching files that need to be signed.  E.g., packages
# that include firmware files would set BRP_PESIGN_FILES='*.ko /lib/firmware'
export BRP_PESIGN_FILES='*.ko'

%changelog
* Fri Apr 27 2017 – andavis@suse.com
- Typo fixes; remove excluded flavors from kernel_module_package macro line.
* Wed Apr 24 2013 - mmarek@suse.cz
- Sign the module by a supplied keypair.
* Tue Dec 22 2008 - andavis@suse.com
- Updated to reflect CODE 11 changes and LF standard spec file work.
* Sat Jan 28 2006 - agruen@suse.de
- Initial package.
```

14 Appendix B: Changes, Updates and References

14.1 Code 10 to Code 11 Notes for Kernel Module Packagers

Following is a brief list of CODE 10 to CODE 11 changes which can affect kernel module packagers

- *Kernel packages*:
  - The Code 11 kernel is provided via two packages: `kernel-FLAVOR-base` and `kernel-FLAVOR`. `kernel-FLAVOR` is dependent on `kernel-FLAVOR-base`. Both packages are required to provide checksums for all the kernel symbol groups.
  - Code 11 automatically installs the **-pae** kernel flavor for 32-bit pae-enabled systems.
  - The Code 11 `kernel-source` package does not include the sample Kernel Module Package source provided in Appendix A.
  - Kernel packages in SUSE Linux Enterprise Server 12 or SUSE Linux Enterprise Desktop 12 (including all service packs) provide checksums for individual symbols rather than for groups of symbols (symsets).
- *Kernel module packages*:
  - Starting with SUSE Linux Enterprise Server 11 SP3 or SUSE Linux Enterprise Desktop 11 SP3, kernel module packages can contain modules that have been signed in order to support UEFI Secure Boot.
  - Kernel Module Packages built for SUSE Linux Enterprise Server 12 or SUSE Linux Enterprise Desktop 12 (including all service packs) require checksums for individual symbols rather than for groups of symbols (symsets).

- *Package and file locations*:
  - In Code 11, several packages have been moved from the base SUSE Linux Enterprise Server product to the SUSE Linux Enterprise SDK. Developers building Kernel Module Packages on Code 11 will need to include the SDK in their build environment to ensure that all build-time package dependencies are resolved.
  - In Code 11, the RPM macros used to build Kernel Module Packages are provided in the `kernel-source` package (instead of the RPM package) and installed under `/etc/rpm` and `/usr/lib/rpm`.
- *RPM macro changes*:
  - Code 11 includes RPM macros to facilitate creating Kernel Module Package build structures that support multiple RPM-based distributions. The spec file template in Appendix A includes these macros:
    - %kernel_module_package_buildreqs - used for "BuildRequires".
    - **%kernel_module_package** - used instead of **%suse_kernel_module_package** (keep in mind that the options are slightly different: with **%kernel_module_package**, `-x` is used to exclude flavors from the build, and `-t` replaces `-s` as the option to override the default sub-package template. See section *Section 5, "Kernel Module Packages"* for a complete description of all **%kernel_module_package** options.)
    - **%kernel_source** – used with the `$flavor` argument to specify the location of the top-level kernel Makefile.
  - Starting with SUSE Linux Enterprise Server 11 SP2 or SUSE Linux Enterprise Desktop 11 SP2, the **%kernel_module_package** macro includes a `-b` option that can be used to force initial RAM disk creation when a KMP is installed.
  - Starting with SUSE Linux Enterprise Server/SUSE Linux Enterprise Desktop 11 SP3, the **%kernel_module_package macro** includes a `-c <module-signing-certificate>` option that can be used when building secure-boot-enabled packages.

14.2 Documentation Updates

April 27, 2017 – andavis@suse.com

- Update to incorporate SUSE Linux Enterprise Server/SUSE Linux Enterprise Desktop 12 changes

June 27, 2013 – andavis@suse.com

- Change Novell to SUSE
- Add `-b` option to kernel_module_package macro
- Support building of secure-boot-enabled packages

August 13, 2009 – andavis@suse.com

- Correctly reflect behavior of `-f` option to kernel_module_package macro

December 22, 2008 – andavis@suse.com

- Initial Version: Update Code 10 Kernel Module Packages Manual, and adapt it to Code 11 process

14.3 References

- 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://www.oreilly.com/catalog/linuxdrive3/ (also available online at http://lwn.net/Kernel/LDD3/
- Andreas Gruenbacher: Kernel Module Packages Manual for CODE 10, http://www.novell.com/developer/kernel_module_packages_manuals.html

- Documentation of The Linux Foundation Driver Backport Workgroup, http://www.linux-foundation.org/en/Driver_Backport
- UEFI Specification, http://www.uefi.org/specs

15 Legal notice

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 non-commercially. 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 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.