NAME

firmware_register, **firmware_unregister**, **firmware_get**, **firmware_get_flags**, **firmware_put** - firmware image loading and management

SYNOPSIS

```
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/linker.h>
#include <sys/firmware.h>

struct firmware {
        const char      *name;          /* system-wide name */
        const void      *data;          /* location of image */
        size_t          datasize;  /* size of image in bytes */
        unsigned int    version;   /* version of the image */
};
const struct firmware *
firmware_register(const char *imagename, const void *data, size_t datasize, unsigned int version,
    const struct firmware *parent);

int
firmware_unregister(const char *imagename);

const struct firmware *
firmware_get(const char *imagename);

const struct firmware *
firmware_get_flags(const char *imagename, uint32_t flags);

void
firmware_put(const struct firmware *fp, int flags);
```

DESCRIPTION

The **firmware** abstraction provides a convenient interface for loading **firmware images** into the kernel, and for accessing such images from kernel components.

A **firmware image** (or **image** for brevity) is an opaque block of data residing in kernel memory. It is associated to a unique **imagename** which constitutes a search key, and to an integer **version** number, which is also an opaque piece of information for the firmware subsystem.

An image is registered with the **firmware** subsystem by calling the function **firmware_register**(), and unregistered by calling **firmware_unregister**(). These functions are usually (but not exclusively) called by specially crafted kernel modules that contain the firmware image. The modules can be statically compiled in the kernel, or loaded by **/boot/loader**, manually at runtime, or on demand by the firmware subsystem.

Clients of the firmware subsystem can request access to a given image by calling the function **firmware_get**() with the **imagename** they want as an argument, or by calling **firmware_get_flags**() with the **imagename** and **flags** they want as an arguments. If a matching image is not already registered, the firmware subsystem will try to load it using the mechanisms specified below (typically, a kernel module with **firmware_register** the same name as the image).

API DESCRIPTION

The kernel **firmware_register** firmware API is made of the following functions:

firmware_register() registers with the kernel an image of size **datasize** located at address **data**, under the name **imagename**.

The function returns NULL on error (e.g. because an image with the same name already exists, or the image table is full), or a *const struct firmware* * pointer to the image requested.

firmware_unregister() tries to unregister the firmware image **imagename** from the system. The function is successful and returns 0 if there are no pending references to the image, otherwise it does not unregister the image and returns EBUSY.

firmware_get() and **firmware_get_flags**() return the requested firmware image. The *flags* argument may be set to FIRMWARE_GET_NOWARN to indicate that errors on firmware load or registration should only be logged in case of **booverbose**. If the image is not yet registered with the system, the functions try to load it. This involves the linker subsystem and disk access, so **firmware_get**() or **firmware_get_flags**() must not be called with any locks (except for *Giant*). Note also that if the firmware image is loaded from a filesystem it must already be mounted. In particular this means that it may be necessary to defer requests from a driver attach method unless it is known the root filesystem is already mounted.

On success, **firmware_get**() and **firmware_get_flags**() return a pointer to the image description and increase the reference count for this image. On failure, the functions return NULL.

firmware_put() drops a reference to a firmware image. The *flags* argument may be set to FIRMWARE_UNLOAD to indicate that firmware_put is free to reclaim resources associated with the firmware image if this is the last reference. By default a firmware image will be deferred to a

taskqueue(9) thread so the call may be done while holding a lock. In certain cases, such as on driver detach, this cannot be allowed.

FIRMWARE LOADING MECHANISMS

As mentioned before, any component of the system can register firmware images at any time by simply calling **firmware_register**().

This is typically done when a module containing a firmware image is given control, whether compiled in, or preloaded by **/boot/loader**, or manually loaded with kldload(8). However, a system can implement additional mechanisms to bring these images in memory before calling **firmware_register**().

When **firmware_get**() or **firmware_get_flags**() does not find the requested image, it tries to load it using one of the available loading mechanisms. At the moment, there is only one, namely **Loadable kernel modules**.

A firmware image named **foo** is looked up by trying to load the module named **foo.ko**, using the facilities described in kld(4). In particular, images are looked up in the directories specified by the sysctl variable **kern.module_path** which on most systems defaults to **/boot/kernel;/boot/modules**.

Note that in case a module contains multiple images, the caller should first request a **firmware_get**() or **firmware_get_flags**() for the first image contained in the module, followed by requests for the other images.

BUILDING FIRMWARE LOADABLE MODULES

A firmware module is built by embedding the **firmware image** into a suitable loadable kernel module that calls **firmware_register**() on loading, and **firmware_unregister**() on unloading.

Various system scripts and makefiles let you build a module by simply writing a Makefile with the following entries:

```
KMOD=   imagename
FIRMWS= image_file:imagename[:version]
.include <bsd.kmod.mk>
```

where KMOD is the basename of the module; FIRMWS is a list of colon-separated tuples indicating the image_file's to be embedded in the module, the imagename and version of each firmware image.

If you need to embed firmware images into a system, you should write appropriate entries in the <files.arch> file, e.g. this example is from **sys/arm/xscale/ixp425/files.ixp425**:

```
ixp425_npe_fw.c                 optional npe_fw             \
        compile-with    "${AWK} -f $S/tools/fw_stub.awk                 \
                                IxNpeMicrocode.dat:npe_fw -mnpe -c${.TARGET}"       \
        no-implicit-rule before-depend local                \
        clean           "ixp425_npe_fw.c"
#
# NB: ld encodes the path in the binary symbols generated for the
#     firmware image so link the file to the object directory to
#     get known values for reference in the _fw.c file.
#
IxNpeMicrocode.fwo  optional npe_fw                                 \
        dependency      "IxNpeMicrocode.dat"                        \
        compile-with    "${LD} -b binary -d -warn-common            \
                                -r -d -o ${.TARGET} IxNpeMicrocode.dat"     \
        no-implicit-rule                                \
        clean           "IxNpeMicrocode.fwo"
```

Firmware was previously committed to the source tree as uuencoded files, but this is no longer required; the binary firmware file should be committed to the tree as provided by the vendor.

Note that generating the firmware modules in this way requires the availability of the following tools: awk(1), make(1), the compiler and the linker.

SEE ALSO

kld(4), module(9)

/usr/share/examples/kld/firmware

HISTORY

The **firmware** system was introduced in FreeBSD 6.1.

AUTHORS

This manual page was written by Max Laier <*mlaier@FreeBSD.org*>.