NAME

kobj - a kernel object system for FreeBSD

SYNOPSIS

#include <sys/param.h>
#include <sys/kobj.h>

void
kobj_class_compile(*kobj_class_t cls*);

void
kobj_class_compile_static(*kobj_class_t cls*, *kobj_ops_t ops*);

void
kobj_class_free(*kobj_class_t cls*);

kobj_t
kobj_create(*kobj_class_t cls*, *struct malloc_type *mtype*, *int mflags*);

void
kobj_init(*kobj_t obj*, *kobj_class_t cls*);

void
kobj_init_static(*kobj_t obj*, *kobj_class_t cls*);

void
kobj_delete(*kobj_t obj*, *struct malloc_type *mtype*);

DEFINE_CLASS(*name*, *kobj_method_t *methods*, *size_t size*);

DESCRIPTION

The kernel object system implements an object-oriented programming system in the FreeBSD kernel. The system is based around the concepts of interfaces, which are descriptions of sets of methods; classes, which are lists of functions implementing certain methods from those interfaces; and objects, which combine a class with a structure in memory.

Methods are called using a dynamic method dispatching algorithm which is designed to allow new interfaces and classes to be introduced into the system at runtime. The method dispatch algorithm is designed to be both fast and robust and is only slightly more expensive than a direct function call, making kernel objects suitable for performance-critical algorithms.

Suitable uses for kernel objects are any algorithms which need some kind of polymorphism (i.e., many different objects which can be treated in a uniform way). The common behaviour of the objects is described by a suitable interface and each different type of object is implemented by a suitable class.

The simplest way to create a kernel object is to call **kobj_create**() with a suitable class, malloc type and flags (see malloc(9) for a description of the malloc type and flags). This will allocate memory for the object based on the object size specified by the class and initialise it by zeroing the memory and installing a pointer to the class' method dispatch table. Objects created in this way should be freed by calling **kobj_delete**().

Clients which would like to manage the allocation of memory themselves should call **kobj_init**() or **kobj_init_static**() with a pointer to the memory for the object and the class which implements it. It is also possible to use **kobj_init**() and **kobj_init_static**() to change the class for an object. This should be done with care as the classes must agree on the layout of the object. The device framework uses this feature to associate drivers with devices.

The functions **kobj_class_compile**(), **kobj_class_compile_static**() and **kobj_class_free**() are used to process a class description to make method dispatching efficient. A client should not normally need to call these since a class will automatically be compiled the first time it is used. If a class is to be used before malloc(9) and mutex(9) are initialised, then **kobj_class_compile_static**() should be called with the class and a pointer to a statically allocated *kobj_ops* structure before the class is used to initialise any objects. In that case, also **kobj_init_static**() should be used instead of **kobj_init**().

To define a class, first define a simple array of *kobj_method_t*. Each method which the class implements should be entered into the table using the macro **KOBJMETHOD**() which takes the name of the method (including its interface) and a pointer to a function which implements it. The table should be terminated with two zeros. The macro **DEFINE_CLASS**() can then be used to initialise a *kobj_class_t* structure. The size argument to **DEFINE_CLASS**() specifies how much memory should be allocated for each object.

HISTORY

Some of the concepts for this interface appeared in the device framework used for the alpha port of FreeBSD 3.0 and more widely in FreeBSD 4.0.

AUTHORS

This manual page was written by Doug Rabson.