Where To Start

Modules are usually one-object beasts, perhaps with a panel or two attached. Before writing a module, you need to be reasonably acquainted with Objective-C, Interface Builder, and the SoundKit. From there, it shouldn't be too tough.

Tools

Take a gander at the following masterworks of spaghetti code, included with the application:

ModuleController.h

The Module Controller was designed as the gateway between your module object(s) and the application code. It is the object to which your module object should send messages if it wants the program to do things. The Module Controller's methods encompass most everything the program can actually do.

Module.h handles much of these little nitpicky things in its skeleton code. See below.

Try not to send messages to other program objects other than the Module Controller sounds and soundviews--my code is so nasty for the time being, you might wind up doing something you wished you hadn't. In addition, I make no promises about future versions of this program other than an honest attempt to be backwards compatible with earlier versions of the Module Controller.

Module.h and Module.m

Module is merely skeleton code that creates an ID for the Module Controller, and imports the application, the Module Controller, the AppKit and SoundKit, and various C routines. To make a module, your main object really should descend from Module.h.

Imports.h

Imports.h imports the kitchen sink. To use most objects and libraries, including all application objects, you only need to import Imports.h.

How To

Much of the following stuff will be done by modifying the program's nib file with Interface Builder. The rest is typing in code that Interface Builder has parsed out.

First, create your main object as a subclass of Module.h. Note that Module.h has an ID for the Module Controller, and Module.m takes in Imports.h. The Module Controller ID is where you will send practically all of your messages, and Imports.h has most of the include and import files you will need. Write out the skeleton code from Interface Builder. Be sure to include the module's object in your project directory.

Next, create whatever nib files, interface stuff, etc., you need. Include these in the application's project. If you don't need any (like the Reverse module doesn't need any)

Then, write the code for the module. Libraries for interfacing with the application are found in ModuleController.h. To communicate with the program, send messages to TheModuleController.

Finally, instantiate your object(s) in the application's nib file. Include a menu item which starts your module; this should go in the Modules menu. And be sure to connect the instance of ModuleController to your objects, so they know where they're sending messages to. Compile the program, and you're in business!

One Note
Be sure to inform the program that a sound has been edited, selected, zoomed, or whatnot *after* editing the sound. These procedures change the way the sound is displayed, and may also compact it, change its reduction, or other nasty things, so although you want to tell the program that the sound has changed, change the sound *first*, and then your module won't die.

About Preferences
A late addition to the program is the Preferences Manager. This system is open for you to add preferences, and some examples have been put in already. It's not well commented, so it's a jungle out there, but feel free to use it.

Program Structure
There are quite a few separate objects in this program.

paste_0.eps ¬

File Controller: Maintains all windows, sounds, and soundviews.
Edit Controller: Controls all editing functions and zooming stuff.
Module Controller: Acts as gateway for modules to gain access to inside parts of program.

Sound Manager: Controls playing and recording of sounds.
Console Manager: Manages the console window.
Info Manager: Manages the information window.
Preferences Manager: Handles all preferences storage, and the preferences window
Sound Table and
String Tables: Data Types used by the File Controller alone to manage the sounds, soundviews, windows, and sound files.
Update View: A custom Sound View.
Module.h The object you descend from to make a module.

Imports.h Modules automatically import this. All objects import it. It's got most everything anyone needs to import.... :)
FourSounds.tiff The Program Icon

I *hope* that's it! Good luck.

Sean Luke
June 15, 1992